

PureBasic Reference Manual

6.21

<http://www.purebasic.com/>

June 11, 2025

Contents

I	General	38
1	Introduction	39
2	Terms And Conditions	43
3	System requirements	44
4	Installation	45
5	Order	46
6	Contact	48
7	Acknowledgements	49
II	The PureBasic Editor	51
8	Getting Started	52
9	Working with source files	53
10	Editing features	55
11	Managing projects	61
12	Compiling your programs	66
13	Using the debugger	74
14	Included debugging tools	80
15	Using the built-in Tools	87
16	Using external tools	94
17	Getting Help	99
18	Customizing the IDE	101
19	Command-line options for the IDE	116
III	Language Reference	118
20	Working with different number bases	119
21	Break : Continue	123
22	Using the command line compiler	125
23	Compiler Directives	128

24	Compiler Functions	133
25	Data	143
26	Debugger keywords in PureBasic	147
27	Define	149
28	Dim	151
29	Building a DLL	153
30	Enumerations	155
31	For : Next	158
32	ForEach : Next	160
33	General Syntax Rules	162
34	Global	166
35	Gosub : Return	169
36	Handles and Numbers	171
37	If : Else : EndIf	173
38	Import : EndImport	175
39	Includes Functions	177
40	Inline x86 ASM	179
41	Interfaces	183
42	Licenses for the PureBasic applications (without using 3D engine)	185
43	Licenses for the 3D engine integrated with PureBasic	203
44	Macros	230
45	Pointers and memory access	233
46	Migration guide	237
47	Migration from PureBasic 5.20 LTS to 5.40 LTS	238
48	Migration from PureBasic 5.30 to 5.40	241
49	Migration from PureBasic 5.40 to 5.50	244
50	Migration from PureBasic 5.50 to 5.60	245
51	Migration from PureBasic 5.60 to 5.72 LTS	246
52	Module	247
53	NewList	250
54	NewMap	252
55	Others Commands	255

56	Procedures	257
57	Protected	261
58	Prototypes	263
59	Pseudotypes	265
60	PureBasic objects	267
61	Building a PureLibrary	270
62	Repeat : Until	272
63	Residents	273
64	Runtime	274
65	Select : EndSelect	276
66	Using several PureBasic versions on Windows	278
67	Shared	279
68	Static	280
69	Structures	282
70	Subsystems	288
71	Threaded	290
72	UserGuide - Advanced functions	292
73	UserGuide - Constants	293
74	UserGuide - Storing data in memory	294
75	UserGuide - Decisions & Conditions	296
76	UserGuide - Compiler directives (for different behavior on different OS)	298
77	UserGuide - Reading and writing files	302
78	UserGuide - First steps with the Debug output (variables & operators)	305
79	UserGuide - Displaying graphics output & simple drawing	307
80	UserGuide - Building a graphical user interface (GUI)	312
81	UserGuide - Input & Output	316
82	UserGuide - Other Compiler keywords	317
83	UserGuide - Other Library functions	318
84	UserGuide - Loops	319
85	UserGuide - Memory access	321
86	UserGuide - Overview	327
87	UserGuide - Dynamic numbering of windows and gadgets using #PB_Any	328

88	UserGuide - Managing multiple windows with different content	334
89	UserGuide - Structuring code in Procedures	349
90	UserGuide - String Manipulation	354
91	UserGuide - Displaying text output (Console)	357
92	UserGuide - Some Tips & Tricks	360
93	UserGuide - Variables and Processing of variables	363
94	Unicode	366
95	Variables and Types	367
96	While : Wend	376
97	Windows Message Handling	377
98	With : EndWith	379
IV	Library Reference	381
99	2DDrawing	382
99.1	Red	382
99.2	Green	383
99.3	Blue	383
99.4	Alpha	384
99.5	RGB	384
99.6	RGBA	385
99.7	AlphaBlend	386
99.8	BackColor	387
99.9	Box	387
99.10	RoundBox	389
99.11	Circle	390
99.12	DrawImage	391
99.13	DrawAlphaImage	392
99.14	DrawingBuffer	392
99.15	DrawingBufferPitch	393
99.16	DrawingBufferPixelFormat	394
99.17	DrawingFont	395
99.18	DrawingMode	396
99.19	DrawRotatedText	399
99.20	FillArea	400
99.21	GrabDrawingImage	401
99.22	StartDrawing	402
99.23	DrawText	403
99.24	Ellipse	404
99.25	FrontColor	405
99.26	Line	406
99.27	LineXY	407
99.28	Plot	408
99.29	Point	409
99.30	StopDrawing	410
99.31	TextHeight	410
99.32	TextWidth	411
99.33	OutputDepth	411
99.34	OutputWidth	412
99.35	OutputHeight	412

99.36	CustomFilterCallback	413
99.37	GradientColor	415
99.38	ResetGradientColors	416
99.39	LinearGradient	417
99.40	CircularGradient	418
99.41	EllipticalGradient	419
99.42	BoxedGradient	421
99.43	ConicalGradient	422
99.44	CustomGradient	424
99.45	SetOrigin	425
99.46	GetOriginX	427
99.47	GetOriginY	427
99.48	ClipOutput	428
99.49	UnclipOutput	429
100	Array	430
100.1	ArraySize	430
100.2	CompareArray	432
100.3	CopyArray	433
100.4	FreeArray	434
101	AudioCD	435
101.1	AudioCDLength	435
101.2	AudioCDName	436
101.3	AudioCDTrackLength	436
101.4	AudioCDStatus	437
101.5	AudioCDTracks	437
101.6	AudioCDTrackSeconds	438
101.7	EjectAudioCD	438
101.8	InitAudioCD	439
101.9	PauseAudioCD	439
101.10	PlayAudioCD	440
101.11	ResumeAudioCD	440
101.12	StopAudioCD	441
101.13	UseAudioCD	441
102	Billboard	443
102.1	AddBillboard	443
102.2	BillboardGroupID	444
102.3	BillboardGroupMaterial	444
102.4	BillboardGroupX	444
102.5	BillboardGroupY	445
102.6	BillboardGroupZ	445
102.7	BillboardHeight	446
102.8	BillboardLocate	446
102.9	BillboardWidth	447
102.10	BillboardX	447
102.11	BillboardY	448
102.12	BillboardZ	448
102.13	ClearBillboards	449
102.14	CountBillboards	449
102.15	CreateBillboardGroup	449
102.16	BillboardGroupCommonDirection	451
102.17	BillboardGroupCommonUpVector	451
102.18	FreeBillboardGroup	452
102.19	HideBillboardGroup	452
102.20	IsBillboardGroup	453
102.21	MoveBillboard	453
102.22	MoveBillboardGroup	454

102.23	RemoveBillboard	454
102.24	ResizeBillboard	455
102.25	RotateBillboardGroup	455
103	CGI	457
103.1	CGICookieName	457
103.2	CGICookieValue	458
103.3	CountCGICookies	459
103.4	CountCGIParameters	460
103.5	CGIParameTerName	461
103.6	CGIParameTerValue	461
103.7	CGIParameTerType	462
103.8	CGIParameTerData	464
103.9	CGIParameTerDataSize	465
103.10	CGIBuffer	466
103.11	CGIVariable	467
103.12	FinishFastCGIRequest	470
103.13	InitCGI	471
103.14	InitFastCGI	472
103.15	ReadCGI	473
103.16	WriteCGIFile	474
103.17	WriteCGIData	475
103.18	WriteCGIHeader	476
103.19	WriteCGIString	477
103.20	WriteCGIStringN	478
103.21	WaitFastCGIRequest	479
104	Camera	481
104.1	CameraBackColor	481
104.2	CameraFollow	481
104.3	CameraFOV	483
104.4	CameraID	483
104.5	CameraCustomParameter	483
104.6	CheckObjectVisibility	484
104.7	CameraDirection	485
104.8	CameraDirectionX	486
104.9	CameraDirectionY	486
104.10	CameraDirectionZ	487
104.11	CameraFixedYawAxis	487
104.12	CameraLookAt	488
104.13	CameraProjectionMode	488
104.14	CameraProjectionX	489
104.15	CameraProjectionY	489
104.16	CameraRange	489
104.17	CameraRenderMode	490
104.18	CameraReflection	490
104.19	CameraRoll	491
104.20	CameraPitch	492
104.21	CameraYaw	492
104.22	CameraViewX	493
104.23	CameraViewY	493
104.24	CameraViewWidth	494
104.25	CameraViewHeight	494
104.26	CameraX	495
104.27	CameraY	495
104.28	CameraZ	496
104.29	CreateCamera	496
104.30	FreeCamera	498
104.31	IsCamera	499

104.32	MoveCamera	499
104.33	ResizeCamera	500
104.34	RotateCamera	501
105	Cipher	502
105.1	AddCipherBuffer	502
105.2	AESEncoder	503
105.3	AESDecoder	505
105.4	CreatePasswordHash	506
105.5	VerifyPasswordHash	507
105.6	DESFingerprint	507
105.7	DeriveCipherKey	508
105.8	StartFingerprint	509
105.9	FinishCipher	511
105.10	IsCipher	511
105.11	AddFingerprintBuffer	512
105.12	FinishFingerprint	513
105.13	IsFingerprint	513
105.14	FileFingerprint	514
105.15	Fingerprint	515
105.16	StringFingerprint	516
105.17	UseMD5Fingerprint	518
105.18	UseSHA1Fingerprint	519
105.19	UseSHA2Fingerprint	519
105.20	UseSHA3Fingerprint	520
105.21	UseCRC32Fingerprint	521
105.22	Base64DecoderBuffer	522
105.23	Base64EncoderBuffer	523
105.24	Base64Decoder	524
105.25	Base64Encoder	525
105.26	StartAESCipher	526
105.27	OpenCryptRandom	527
105.28	CloseCryptRandom	528
105.29	CryptRandom	528
105.30	CryptRandomData	529
106	Clipboard	531
106.1	ClearClipboard	531
106.2	GetClipboardImage	532
106.3	GetClipboardText	532
106.4	SetClipboardImage	533
106.5	SetClipboardText	534
107	Console	535
107.1	ClearConsole	535
107.2	CloseConsole	536
107.3	ConsoleError	537
107.4	ConsoleTitle	537
107.5	ConsoleColor	538
107.6	EnableGraphicalConsole	540
107.7	Inkey	541
107.8	Input	542
107.9	ConsoleLocate	543
107.10	ConsoleCursor	544
107.11	Print	545
107.12	PrintN	546
107.13	OpenConsole	547
107.14	ReadConsoleData	548
107.15	RawKey	549

107.16	WriteConsoleData	550
108	Database	552
108.1	AffectedDatabaseRows	552
108.2	CloseDatabase	553
108.3	DatabaseColumns	553
108.4	DatabaseColumnIndex	554
108.5	DatabaseColumnName	554
108.6	DatabaseColumnSize	555
108.7	DatabaseColumnType	555
108.8	DatabaseDriverDescription	556
108.9	DatabaseDriverName	557
108.10	DatabaseError	557
108.11	DatabaseID	558
108.12	DatabaseQuery	558
108.13	DatabaseUpdate	560
108.14	ExamineDatabaseDrivers	562
108.15	FinishDatabaseQuery	562
108.16	FirstDatabaseRow	563
108.17	GetDatabaseBlob	564
108.18	GetDatabaseDouble	565
108.19	GetDatabaseFloat	565
108.20	GetDatabaseLong	566
108.21	GetDatabaseQuad	567
108.22	GetDatabaseString	567
108.23	CheckDatabaseNull	568
108.24	IsDatabase	569
108.25	NextDatabaseDriver	569
108.26	NextDatabaseRow	570
108.27	OpenDatabase	570
108.28	OpenDatabaseRequester	571
108.29	PreviousDatabaseRow	572
108.30	SetDatabaseBlob	573
108.31	UseMySQLDatabase	574
108.32	UsePostgreSQLDatabase	575
108.33	UseSQLiteDatabase	576
108.34	UseODBCDatabase	577
108.35	SetDatabaseString	578
108.36	SetDatabaseLong	579
108.37	SetDatabaseQuad	580
108.38	SetDatabaseFloat	580
108.39	SetDatabaseDouble	581
108.40	SetDatabaseNull	582
109	Date	584
109.1	AddDate	584
109.2	ConvertDate	585
109.3	Date	586
109.4	DateUTC	586
109.5	Day	587
109.6	DayOfWeek	587
109.7	DayOfYear	588
109.8	Month	589
109.9	Year	589
109.10	Hour	590
109.11	Minute	591
109.12	Second	591
109.13	FormatDate	592
109.14	ParseDate	593

110	Debugger	594
110.1	CopyDebugOutput	594
110.2	ShowDebugOutput	595
110.3	CloseDebugOutput	595
110.4	ClearDebugOutput	596
110.5	DebuggerError	596
110.6	DebuggerWarning	597
110.7	SaveDebugOutput	597
110.8	ShowProfiler	598
110.9	ResetProfiler	599
110.10	StartProfiler	599
110.11	StopProfiler	600
110.12	ShowMemoryViewer	600
110.13	ShowLibraryViewer	601
110.14	ShowWatchlist	602
110.15	ShowVariableViewer	602
110.16	ShowCallstack	603
110.17	ShowAssemblyViewer	603
110.18	PurifierGranularity	604
111	Desktop	605
111.1	ExamineDesktops	605
111.2	DesktopDepth	606
111.3	DesktopResolutionX	607
111.4	DesktopResolutionY	607
111.5	DesktopScaledX	608
111.6	DesktopScaledY	609
111.7	DesktopUnscaledX	610
111.8	DesktopUnscaledY	610
111.9	DesktopFrequency	611
111.10	DesktopHeight	612
111.11	DesktopX	613
111.12	DesktopY	614
111.13	DesktopMouseX	614
111.14	DesktopMouseY	615
111.15	DesktopName	616
111.16	DesktopWidth	617
112	Dialog	619
112.1	CreateDialog	619
112.2	DialogError	620
112.3	DialogGadget	620
112.4	DialogWindow	621
112.5	DialogID	621
112.6	FreeDialog	622
112.7	IsDialog	622
112.8	OpenXMLDialog	623
112.9	RefreshDialog	631
112.10	UseDialogOpenGLGadget	632
112.11	UseDialogScintillaGadget	632
112.12	UseDialogWebGadget	633
112.13	UseDialogWebViewGadget	633
113	DragDrop	634
113.1	DragText	634
113.2	DragImage	636
113.3	DragFiles	637
113.4	DragPrivate	638
113.5	DragOSFormats	640

113.6	EnableGadgetDrop	641
113.7	EnableWindowDrop	643
113.8	EventDropAction	644
113.9	EventDropType	644
113.10	EventDropText	645
113.11	EventDropImage	646
113.12	EventDropFiles	646
113.13	EventDropPrivate	647
113.14	EventDropBuffer	647
113.15	EventDropSize	648
113.16	EventDropX	649
113.17	EventDropY	649
113.18	SetDragCallback	650
113.19	SetDropCallback	651
113.20	ExamineDraggedItems	652
113.21	NextDraggedItem	653
113.22	DraggedItemIndex	653
114	Engine3D	654
114.1	Add3DArchive	655
114.2	AmbientColor	656
114.3	AntialiasingMode	656
114.4	ConvertLocalToWorldPosition	656
114.5	ConvertWorldToLocalPosition	657
114.6	Engine3DStatus	658
114.7	EnableWorldCollisions	658
114.8	EnableWorldPhysics	659
114.9	ExamineWorldCollisions	659
114.10	NextWorldCollision	660
114.11	FirstWorldCollisionEntity	660
114.12	SecondWorldCollisionEntity	661
114.13	WorldCollisionContact	661
114.14	WorldCollisionNormal	661
114.15	WorldCollisionAppliedImpulse	662
114.16	FetchOrientation	662
114.17	SetOrientation	663
114.18	GetX	664
114.19	GetY	664
114.20	GetZ	664
114.21	GetW	665
114.22	Fog	665
114.23	InitEngine3D	666
114.24	LoadWorld	666
114.25	MousePick	667
114.26	PointPick	667
114.27	BodyPick	668
114.28	PickX	668
114.29	PickY	669
114.30	PickZ	669
114.31	RayCollide	669
114.32	RayCast	670
114.33	MouseRayCast	670
114.34	NormalX	671
114.35	NormalY	671
114.36	NormalZ	671
114.37	RayPick	672
114.38	ShowGUI	672
114.39	SetGUITheme3D	673
114.40	Parse3DScripts	673

114.41	RenderWorld	674
114.42	SetRenderQueue	674
114.43	SkyBox	675
114.44	SkyDome	676
114.45	CreateWater	677
114.46	WorldShadows	677
114.47	WorldGravity	678
114.48	WorldDebug	679
114.49	Pitch	679
114.50	Roll	680
114.51	Yaw	681
114.52	GetWorldAttribute	682
114.53	SetWorldAttribute	683
115	Entity	684
115.1	ApplyEntityForce	684
115.2	ApplyEntityImpulse	685
115.3	ApplyEntityTorque	685
115.4	ApplyEntityTorqueImpulse	686
115.5	CopyEntity	687
115.6	CreateEntity	687
115.7	EntityFixedYawAxis	688
115.8	EntityID	688
115.9	EntityLookAt	688
115.10	EntityVelocity	689
115.11	EntityAngularFactor	689
115.12	EntityLinearFactor	690
115.13	EntityCustomParameter	690
115.14	EntityBoundingBox	691
115.15	DisableEntityBody	692
115.16	EntityParentNode	692
115.17	FetchEntityMaterial	692
115.18	SetEntityMaterial	693
115.19	EntityCollide	693
115.20	CreateEntityBody	694
115.21	EntityRenderMode	695
115.22	AttachEntityObject	696
115.23	DetachEntityObject	697
115.24	EnableManualEntityBoneControl	697
115.25	MoveEntityBone	698
115.26	ScaleEntityBone	698
115.27	FreeEntityBody	699
115.28	FreeEntityJoints	699
115.29	EntityBoneX	700
115.30	EntityBoneY	700
115.31	EntityBoneZ	701
115.32	EntityBonePitch	701
115.33	EntityBoneYaw	702
115.34	EntityBoneRoll	702
115.35	EntityX	702
115.36	EntityY	703
115.37	EntityZ	703
115.38	FreeEntity	704
115.39	HideEntity	704
115.40	IsEntity	705
115.41	MoveEntity	705
115.42	RotateEntity	705
115.43	RotateEntityBone	706
115.44	ScaleEntity	707

115.45	EntityRoll	707
115.46	EntityPitch	708
115.47	EntityYaw	709
115.48	GetEntityAttribute	709
115.49	SetEntityAttribute	711
115.50	GetEntityCollisionMask	711
115.51	GetEntityCollisionGroup	712
115.52	SetEntityCollisionFilter	712
115.53	AddSubEntity	713
115.54	EntityDirection	714
115.55	EntityDirectionX	715
115.56	EntityDirectionY	716
115.57	EntityDirectionZ	716
115.58	GetEntityMesh	716
116	EntityAnimation	718
116.1	AddEntityAnimationTime	718
116.2	StartEntityAnimation	719
116.3	StopEntityAnimation	719
116.4	EntityAnimationStatus	720
116.5	EntityAnimationBlendMode	720
116.6	GetEntityAnimationTime	721
116.7	SetEntityAnimationTime	722
116.8	GetEntityAnimationLength	722
116.9	SetEntityAnimationLength	723
116.10	GetEntityAnimationWeight	723
116.11	SetEntityAnimationWeight	724
116.12	UpdateEntityAnimation	724
117	File	726
117.1	CloseFile	726
117.2	CreateFile	727
117.3	Eof	728
117.4	FileBuffersSize	728
117.5	FileID	729
117.6	FileSeek	729
117.7	FlushFileBuffers	730
117.8	IsFile	731
117.9	Loc	731
117.10	Lof	732
117.11	OpenFile	733
117.12	TruncateFile	734
117.13	ReadAsciiCharacter	734
117.14	ReadByte	735
117.15	ReadCharacter	735
117.16	ReadDouble	736
117.17	ReadFile	737
117.18	ReadFloat	738
117.19	ReadInteger	738
117.20	ReadLong	739
117.21	ReadQuad	739
117.22	ReadData	740
117.23	ReadString	741
117.24	ReadStringFormat	741
117.25	ReadUnicodeCharacter	742
117.26	ReadWord	743
117.27	WriteAsciiCharacter	743
117.28	WriteByte	744
117.29	WriteCharacter	744

117.30	WriteDouble	745
117.31	WriteFloat	746
117.32	WriteInteger	746
117.33	WriteLong	747
117.34	WriteData	747
117.35	WriteQuad	748
117.36	WriteString	749
117.37	WriteStringFormat	750
117.38	WriteStringN	751
117.39	WriteUnicodeCharacter	751
117.40	WriteWord	752
118	FileSystem	753
118.1	CopyDirectory	753
118.2	CopyFile	754
118.3	CreateDirectory	755
118.4	DeleteDirectory	755
118.5	DeleteFile	756
118.6	DirectoryEntryAttributes	756
118.7	DirectoryEntryDate	757
118.8	DirectoryEntryName	758
118.9	DirectoryEntryType	758
118.10	DirectoryEntrySize	759
118.11	ExamineDirectory	759
118.12	FinishDirectory	760
118.13	GetExtensionPart	761
118.14	GetFilePart	761
118.15	GetPathPart	762
118.16	IsDirectory	763
118.17	CheckFilename	763
118.18	FileSize	764
118.19	GetCurrentDirectory	764
118.20	GetHomeDirectory	765
118.21	GetUserDirectory	765
118.22	GetTemporaryDirectory	766
118.23	GetFileDate	767
118.24	GetFileAttributes	767
118.25	NextDirectoryEntry	769
118.26	RenameFile	769
118.27	SetFileDate	770
118.28	SetFileAttributes	771
118.29	SetCurrentDirectory	772
119	Font	773
119.1	FreeFont	773
119.2	FontID	773
119.3	IsFont	774
119.4	LoadFont	775
119.5	RegisterFontFile	776
120	Ftp	778
120.1	AbortFTPFile	778
120.2	CheckFTPConnection	778
120.3	CloseFTP	779
120.4	CreateFTPDDirectory	779
120.5	DeleteFTPDDirectory	780
120.6	DeleteFTPFile	780
120.7	ExamineFTPDDirectory	781
120.8	GetFTPDDirectory	782

120.9	FinishFTPDirectory	782
120.10	FTPDirectoryEntryAttributes	783
120.11	FTPDirectoryEntryDate	784
120.12	FTPDirectoryEntryName	784
120.13	FTPDirectoryEntryType	785
120.14	FTPDirectoryEntryRaw	785
120.15	FTPDirectoryEntrySize	786
120.16	FTPProgress	786
120.17	IsFtp	787
120.18	NextFTPDirectoryEntry	787
120.19	OpenFTP	788
120.20	ReceiveFTPFile	789
120.21	RenameFTPFile	790
120.22	SendFTPFile	790
120.23	SetFTPDirectory	791
121	Gadget	793
121.1	AddGadgetColumn	793
121.2	AddGadgetItem	794
121.3	ButtonImageGadget	796
121.4	ButtonGadget	797
121.5	CalendarGadget	798
121.6	CanvasGadget	800
121.7	CanvasOutput	805
121.8	CanvasVectorOutput	806
121.9	OpenGLGadget	807
121.10	CheckBoxGadget	811
121.11	ClearGadgetItems	812
121.12	CloseGadgetList	813
121.13	ComboBoxGadget	814
121.14	ContainerGadget	815
121.15	CountGadgetItems	817
121.16	DateGadget	817
121.17	DisableGadget	819
121.18	EditorGadget	820
121.19	ExplorerComboGadget	822
121.20	ExplorerListGadget	823
121.21	ExplorerTreeGadget	825
121.22	FrameGadget	827
121.23	FreeGadget	829
121.24	GadgetID	830
121.25	GadgetItemID	830
121.26	GadgetToolTip	831
121.27	GadgetX	832
121.28	GadgetY	832
121.29	GadgetHeight	833
121.30	GadgetType	834
121.31	GadgetWidth	835
121.32	GetActiveGadget	836
121.33	GetGadgetAttribute	836
121.34	GetGadgetColor	837
121.35	GetGadgetData	838
121.36	GetGadgetFont	839
121.37	GetGadgetItemAttribute	839
121.38	GetGadgetItemColor	840
121.39	GetGadgetItemData	841
121.40	GetGadgetState	841
121.41	GetGadgetItemText	842
121.42	GetGadgetItemState	843

121.43	GetGadgetText	845
121.44	HideGadget	845
121.45	HyperLinkGadget	846
121.46	ImageGadget	848
121.47	IPAddressGadget	849
121.48	IsGadget	850
121.49	ListIconGadget	851
121.50	ListViewGadget	854
121.51	MDIGadget	856
121.52	OpenGadgetList	858
121.53	OptionGadget	859
121.54	PanelGadget	860
121.55	ProgressBarGadget	862
121.56	RemoveGadgetColumn	863
121.57	RemoveGadgetItem	864
121.58	ResizeGadget	865
121.59	ScrollBarGadget	866
121.60	ScrollAreaGadget	868
121.61	SetActiveGadget	870
121.62	SetGadgetAttribute	871
121.63	SetGadgetColor	872
121.64	SetGadgetData	874
121.65	SetGadgetFont	875
121.66	SetGadgetItemAttribute	876
121.67	SetGadgetItemColor	877
121.68	SetGadgetItemData	878
121.69	SetGadgetItemImage	880
121.70	SetGadgetItemState	880
121.71	SetGadgetItemText	881
121.72	SetGadgetState	882
121.73	SetGadgetText	883
121.74	ShortcutGadget	884
121.75	SpinGadget	885
121.76	SplitterGadget	887
121.77	StringGadget	888
121.78	TextGadget	890
121.79	TrackBarGadget	891
121.80	TreeGadget	893
121.81	UseGadgetList	895
121.82	WebGadget	896
121.83	BindGadgetEvent	900
121.84	UnbindGadgetEvent	901
122	Gadget3D	902
122.1	AddGadgetItem3D	902
122.2	ButtonGadget3D	903
122.3	CheckBoxGadget3D	903
122.4	ClearGadgetItems3D	904
122.5	CloseGadgetList3D	904
122.6	ComboBoxGadget3D	905
122.7	ContainerGadget3D	906
122.8	CountGadgetItems3D	906
122.9	DisableGadget3D	907
122.10	EditorGadget3D	907
122.11	FrameGadget3D	908
122.12	FreeGadget3D	908
122.13	GadgetID3D	909
122.14	GadgetToolTip3D	909
122.15	GadgetX3D	910

122.16	GadgetY3D	910
122.17	GadgetHeight3D	911
122.18	GadgetType3D	911
122.19	GadgetWidth3D	912
122.20	GetActiveGadget3D	912
122.21	GetGadgetAttribute3D	913
122.22	GetGadgetData3D	913
122.23	GetGadgetItemData3D	914
122.24	GetGadgetState3D	915
122.25	GetGadgetItemText3D	915
122.26	GetGadgetItemState3D	916
122.27	GetGadgetText3D	916
122.28	HideGadget3D	917
122.29	ImageGadget3D	917
122.30	IsGadget3D	918
122.31	ListViewGadget3D	919
122.32	OpenGadgetList3D	919
122.33	OptionGadget3D	920
122.34	PanelGadget3D	920
122.35	ProgressBarGadget3D	921
122.36	RemoveGadgetItem3D	922
122.37	ResizeGadget3D	922
122.38	ScrollBarGadget3D	923
122.39	ScrollAreaGadget3D	924
122.40	SetActiveGadget3D	925
122.41	SetGadgetAttribute3D	925
122.42	SetGadgetData3D	926
122.43	SetGadgetItemData3D	927
122.44	SetGadgetItemState3D	927
122.45	SetGadgetItemText3D	928
122.46	SetGadgetState3D	928
122.47	SetGadgetText3D	929
122.48	SpinGadget3D	929
122.49	StringGadget3D	930
122.50	TextGadget3D	931
123	Help	932
123.1	CloseHelp	932
123.2	OpenHelp	932
124	Http	934
124.1	AbortHTTP	934
124.2	FinishHTTP	935
124.3	GetURLPart	935
124.4	HTTPProgress	936
124.5	HTTPInfo	937
124.6	HTTPMemory	938
124.7	HTTPProxy	939
124.8	HTTPTimeout	940
124.9	ReceiveHTTPFile	941
124.10	ReceiveHTTPMemory	942
124.11	HTTPRequest	943
124.12	HTTPRequestMemory	946
124.13	URLDecoder	947
124.14	URLEncoder	948
124.15	SetURLPart	949
125	Image	951
125.1	AddImageFrame	951

125.2	RemoveImageFrame	952
125.3	GetImageFrame	952
125.4	SetImageFrame	953
125.5	ImageFrameCount	953
125.6	GetImageFrameDelay	954
125.7	SetImageFrameDelay	954
125.8	CatchImage	955
125.9	CopyImage	956
125.10	CreateImage	956
125.11	EncodeImage	957
125.12	FreeImage	958
125.13	GrabImage	959
125.14	ImageDepth	959
125.15	ImageFormat	960
125.16	ImageHeight	961
125.17	ImageID	961
125.18	ImageOutput	962
125.19	ImageVectorOutput	962
125.20	ImageWidth	963
125.21	IsImage	964
125.22	LoadImage	964
125.23	ResizeImage	965
125.24	SaveImage	966
126	ImagePlugin	968
126.1	UseGIFImageDecoder	968
126.2	UseJPEGImageDecoder	969
126.3	UseJPEGImageEncoder	969
126.4	UseJPEG2000ImageDecoder	970
126.5	UseJPEG2000ImageEncoder	970
126.6	UsePNGImageDecoder	971
126.7	UsePNGImageEncoder	971
126.8	UseTGAImageDecoder	972
126.9	UseTIFFImageDecoder	972
127	Joint	974
127.1	EnableHingeJointAngularMotor	974
127.2	HingeJointMotorTarget	975
127.3	FreeJoint	975
127.4	IsJoint	976
127.5	GenericJoint	976
127.6	PointJoint	977
127.7	HingeJoint	978
127.8	ConeTwistJoint	979
127.9	SliderJoint	979
127.10	GetJointAttribute	980
127.11	SetJointAttribute	981
128	Joystick	982
128.1	InitJoystick	982
128.2	ExamineJoystick	982
128.3	JoystickAxisX	983
128.4	JoystickAxisY	984
128.5	JoystickAxisZ	984
128.6	JoystickName	985
128.7	JoystickButton	986
129	Json	987
129.1	AddJSONElement	987
129.2	AddJSONMember	988

129.3	CatchJSON	989
129.4	ClearJSONElements	989
129.5	ClearJSONMembers	990
129.6	ComposeJSON	991
129.7	CreateJSON	991
129.8	ExamineJSONMembers	992
129.9	ExportJSON	993
129.10	ExportJSONSize	994
129.11	ExtractJSONArray	994
129.12	ExtractJSONList	995
129.13	ExtractJSONMap	996
129.14	ExtractJSONStructure	997
129.15	FreeJSON	998
129.16	GetJSONBoolean	999
129.17	GetJSONDouble	999
129.18	GetJSONElement	1000
129.19	GetJSONFloat	1001
129.20	GetJSONInteger	1001
129.21	GetJSONMember	1002
129.22	GetJSONString	1003
129.23	GetJSONQuad	1003
129.24	InsertJSONArray	1004
129.25	InsertJSONList	1005
129.26	InsertJSONMap	1006
129.27	InsertJSONStructure	1006
129.28	IsJSON	1007
129.29	JSONArraySize	1008
129.30	JSONErrorLine	1008
129.31	JSONErrorMessage	1009
129.32	JSONErrorPosition	1009
129.33	JSONMemberKey	1010
129.34	JSONMemberValue	1010
129.35	JSONObjectSize	1011
129.36	JSONType	1012
129.37	JSONValue	1013
129.38	LoadJSON	1014
129.39	NextJSONMember	1015
129.40	ParseJSON	1015
129.41	RemoveJSONElement	1016
129.42	RemoveJSONMember	1017
129.43	ResizeJSONElements	1017
129.44	SaveJSON	1018
129.45	SetJSONArray	1019
129.46	SetJSONBoolean	1019
129.47	SetJSONDouble	1020
129.48	SetJSONFloat	1021
129.49	SetJSONInteger	1021
129.50	SetJSONNull	1022
129.51	SetJSONObject	1023
129.52	SetJSONQuad	1023
129.53	SetJSONString	1024

130 Keyboard

1025

130.1	InitKeyboard	1025
130.2	ExamineKeyboard	1025
130.3	KeyboardInkey	1026
130.4	KeyboardMode	1027
130.5	KeyboardPushed	1028
130.6	KeyboardReleased	1030

131 Library	1032
131.1 CloseLibrary	1032
131.2 CallCFunction	1033
131.3 CallCFunctionFast	1033
131.4 CallFunction	1034
131.5 CallFunctionFast	1035
131.6 CountLibraryFunctions	1036
131.7 ExamineLibraryFunctions	1036
131.8 GetFunction	1037
131.9 GetFunctionEntry	1038
131.10 IsLibrary	1038
131.11 LibraryFunctionAddress	1039
131.12 LibraryFunctionName	1039
131.13 LibraryID	1040
131.14 NextLibraryFunction	1040
131.15 OpenLibrary	1041
132 Light	1042
132.1 CopyLight	1042
132.2 CreateLight	1042
132.3 FreeLight	1043
132.4 HideLight	1044
132.5 IsLight	1044
132.6 GetLightColor	1045
132.7 SetLightColor	1045
132.8 SpotLightRange	1046
132.9 LightLookAt	1046
132.10 DisableLightShadows	1046
132.11 MoveLight	1047
132.12 LightDirection	1047
132.13 LightDirectionX	1048
132.14 LightDirectionY	1048
132.15 LightDirectionZ	1049
132.16 LightX	1049
132.17 LightY	1050
132.18 LightZ	1050
132.19 LightAttenuation	1051
132.20 RotateLight	1051
132.21 LightRoll	1052
132.22 LightPitch	1052
132.23 LightYaw	1053
132.24 LightID	1054
133 List	1055
133.1 AddElement	1055
133.2 ChangeCurrentElement	1056
133.3 ClearList	1058
133.4 CompareList	1059
133.5 CopyList	1060
133.6 FreeList	1060
133.7 ListSize	1061
133.8 DeleteElement	1062
133.9 FirstElement	1063
133.10 InsertElement	1064
133.11 LastElement	1065
133.12 ListIndex	1067
133.13 NextElement	1068
133.14 PreviousElement	1068
133.15 ResetList	1069

133.16	SelectElement	1070
133.17	SwapElements	1071
133.18	MoveElement	1072
133.19	PushListPosition	1073
133.20	PopListPosition	1074
133.21	MergeLists	1075
133.22	SplitList	1076
134	Mail	1078
134.1	AddMailAttachment	1078
134.2	AddMailAttachmentData	1080
134.3	AddMailRecipient	1081
134.4	CreateMail	1082
134.5	FreeMail	1083
134.6	GetMailAttribute	1084
134.7	GetMailBody	1084
134.8	IsMail	1085
134.9	MailProgress	1085
134.10	RemoveMailRecipient	1086
134.11	SendMail	1087
134.12	SetMailAttribute	1088
134.13	SetMailBody	1089
135	Map	1091
135.1	AddMapElement	1091
135.2	ClearMap	1092
135.3	CompareMap	1093
135.4	CopyMap	1094
135.5	FreeMap	1095
135.6	MapSize	1095
135.7	DeleteMapElement	1096
135.8	FindMapElement	1097
135.9	MapKey	1098
135.10	NextMapElement	1098
135.11	ResetMap	1099
135.12	PushMapPosition	1100
135.13	PopMapPosition	1101
136	Material	1102
136.1	AddMaterialLayer	1102
136.2	CopyMaterial	1103
136.3	CountMaterialLayers	1103
136.4	CreateMaterial	1104
136.5	CreateAnimatedMaterial	1104
136.6	CreateShader	1105
136.7	CreateShaderMaterial	1106
136.8	MaterialShaderAutoParameter	1110
136.9	MaterialShaderParameter	1113
136.10	MaterialShaderTexture	1114
136.11	DisableMaterialLighting	1115
136.12	FreeMaterial	1115
136.13	IsMaterial	1116
136.14	GetMaterialAttribute	1116
136.15	GetMaterialColor	1117
136.16	SetMaterialColor	1118
136.17	MaterialBlendingMode	1119
136.18	MaterialFilteringMode	1119
136.19	MaterialIID	1120
136.20	MaterialShadingMode	1120

136.21	MaterialCullingMode	1121
136.22	MaterialShininess	1122
136.23	MaterialTextureAliases	1122
136.24	GetScriptMaterial	1123
136.25	MaterialFog	1123
136.26	ReloadMaterial	1124
136.27	ResetMaterial	1124
136.28	SetMaterialAttribute	1125
136.29	ScrollMaterial	1126
136.30	RemoveMaterialLayer	1127
136.31	ScaleMaterial	1127
136.32	RotateMaterial	1128
136.33	MaterialAnimation	1129
137	Math	1130
137.1	Abs	1130
137.2	ACos	1131
137.3	ACosH	1131
137.4	ASin	1132
137.5	ASinH	1133
137.6	ATan	1133
137.7	ATan2	1134
137.8	ATanH	1134
137.9	Cos	1135
137.10	CosH	1136
137.11	Degree	1136
137.12	Exp	1137
137.13	Infinity	1137
137.14	Int	1138
137.15	IntQ	1139
137.16	IsInfinity	1139
137.17	IsNaN	1140
137.18	Pow	1141
137.19	Log	1141
137.20	Log10	1142
137.21	Mod	1142
137.22	NaN	1143
137.23	Radian	1143
137.24	Random	1144
137.25	RandomData	1145
137.26	RandomSeed	1146
137.27	Round	1147
137.28	Sign	1147
137.29	Sin	1148
137.30	SinH	1149
137.31	Sqr	1149
137.32	Tan	1150
137.33	TanH	1150
138	Memory	1152
138.1	AllocateMemory	1152
138.2	AllocateStructure	1153
138.3	CompareMemory	1154
138.4	CompareMemoryString	1154
138.5	CopyMemory	1155
138.6	CopyMemoryString	1156
138.7	FillMemory	1157
138.8	FreeMemory	1157
138.9	FreeStructure	1158

138.10	MemorySize	1159
138.11	MemoryStringLength	1159
138.12	MoveMemory	1160
138.13	ReAllocateMemory	1160
138.14	PeekA	1162
138.15	PeekB	1162
138.16	PeekC	1162
138.17	PeekD	1163
138.18	PeekI	1163
138.19	PeekL	1164
138.20	PeekW	1164
138.21	PeekF	1165
138.22	PeekQ	1165
138.23	PeekS	1165
138.24	PeekU	1166
138.25	PokeA	1167
138.26	PokeB	1167
138.27	PokeC	1167
138.28	PokeD	1168
138.29	PokeI	1168
138.30	PokeL	1169
138.31	PokeQ	1169
138.32	PokeW	1170
138.33	PokeF	1170
138.34	PokeS	1170
138.35	PokeU	1171
139	Menu	1172
139.1	CloseSubMenu	1172
139.2	CreateMenu	1173
139.3	CreateImageMenu	1174
139.4	CreatePopupMenu	1175
139.5	CreatePopupMenuImageMenu	1176
139.6	DisplayPopupMenu	1178
139.7	DisableMenuItem	1178
139.8	FreeMenu	1179
139.9	GetMenuItemState	1180
139.10	GetMenuItemText	1181
139.11	GetMenuTitleText	1181
139.12	HideMenu	1182
139.13	IsMenu	1182
139.14	MenuBar	1183
139.15	MenuHeight	1183
139.16	MenuItem	1184
139.17	MenuID	1185
139.18	MenuTitle	1186
139.19	OpenSubMenu	1187
139.20	SetMenuItemState	1188
139.21	SetMenuItemText	1189
139.22	SetMenuTitleText	1189
139.23	BindMenuEvent	1190
139.24	UnbindMenuEvent	1191
140	Mesh	1193
140.1	CreateMesh	1193
140.2	CreateDataMesh	1194
140.3	CopyMesh	1195
140.4	FreeMesh	1195
140.5	IsMesh	1196

140.6	LoadMesh	1196
140.7	MeshID	1197
140.8	GetMeshData	1197
140.9	SetMeshData	1198
140.10	BuildMeshShadowVolume	1199
140.11	CreateLine3D	1200
140.12	CreateCube	1201
140.13	CreateSphere	1202
140.14	CreateTube	1203
140.15	CreateTorus	1204
140.16	CreateCapsule	1205
140.17	CreateIcoSphere	1206
140.18	CreateCone	1207
140.19	CreateCylinder	1208
140.20	CreatePlane	1209
140.21	MeshDirectAdd	1210
140.22	AddSubMesh	1211
140.23	MeshIndexCount	1212
140.24	MeshVertexCount	1212
140.25	UpdateMeshBoundingBox	1213
140.26	UpdateMesh	1213
140.27	MeshIndex	1214
140.28	MeshRadius	1214
140.29	MeshVertex	1215
140.30	MeshVertexPosition	1215
140.31	MeshVertexNormal	1216
140.32	MeshVertexTangent	1216
140.33	MeshVertexColor	1217
140.34	MeshVertexTextureCoordinate	1217
140.35	MeshFace	1218
140.36	FinishMesh	1219
140.37	NormalizeMesh	1219
140.38	BuildMeshTangents	1220
140.39	AddMeshManualLOD	1220
140.40	BuildMeshLOD	1221
140.41	SaveMesh	1221
140.42	SetMeshMaterial	1222
140.43	SubMeshCount	1222
140.44	TransformMesh	1223
141	Mouse	1224
141.1	InitMouse	1224
141.2	ExamineMouse	1224
141.3	MouseButton	1225
141.4	MouseDeltaX	1226
141.5	MouseDeltaY	1226
141.6	MouseLocate	1227
141.7	MouseWheel	1227
141.8	MouseX	1228
141.9	MouseY	1228
141.10	ReleaseMouse	1229
142	Movie	1230
142.1	FreeMovie	1230
142.2	InitMovie	1231
142.3	IsMovie	1231
142.4	LoadMovie	1232
142.5	MovieAudio	1232
142.6	MovieHeight	1233

142.7	MovieInfo	1233
142.8	MovieLength	1234
142.9	MovieSeek	1234
142.10	MovieStatus	1235
142.11	MovieWidth	1235
142.12	PauseMovie	1236
142.13	PlayMovie	1236
142.14	ResizeMovie	1237
142.15	ResumeMovie	1237
142.16	StopMovie	1238
143	Music	1239
143.1	CatchMusic	1239
143.2	FreeMusic	1240
143.3	GetMusicPosition	1240
143.4	GetMusicRow	1241
143.5	IsMusic	1241
143.6	LoadMusic	1242
143.7	MusicVolume	1242
143.8	PlayMusic	1243
143.9	SetMusicPosition	1243
143.10	StopMusic	1244
144	Network	1245
144.1	CloseNetworkConnection	1245
144.2	ConnectionID	1246
144.3	ServerID	1246
144.4	CloseNetworkServer	1247
144.5	CreateNetworkServer	1247
144.6	ExamineIPAddresses	1248
144.7	FreeIP	1249
144.8	HostName	1250
144.9	IPString	1250
144.10	IPAddressField	1250
144.11	MakeIPAddress	1251
144.12	EventServer	1252
144.13	EventClient	1252
144.14	GetClientIP	1253
144.15	GetClientPort	1253
144.16	NetworkClientEvent	1253
144.17	NetworkServerEvent	1254
144.18	NextIPAddress	1255
144.19	OpenNetworkConnection	1256
144.20	ReceiveNetworkData	1257
144.21	SendNetworkData	1257
144.22	SendNetworkString	1258
144.23	UseNetworkTLS	1259
145	Node	1262
145.1	AttachNodeObject	1262
145.2	DetachNodeObject	1263
145.3	CreateNode	1264
145.4	NodeID	1264
145.5	NodeLookAt	1264
145.6	NodeX	1265
145.7	NodeY	1265
145.8	NodeZ	1266
145.9	FreeNode	1267
145.10	IsNode	1267

145.11	MoveNode	1268
145.12	RotateNode	1268
145.13	ScaleNode	1269
145.14	NodeFixedYawAxis	1269
145.15	NodeRoll	1270
145.16	NodePitch	1270
145.17	NodeYaw	1271
146	NodeAnimation	1272
146.1	CreateNodeAnimation	1272
146.2	FreeNodeAnimation	1273
146.3	CreateNodeAnimationKeyFrame	1273
146.4	GetNodeAnimationKeyFrameTime	1274
146.5	SetNodeAnimationKeyFramePosition	1275
146.6	GetNodeAnimationKeyFrameX	1275
146.7	GetNodeAnimationKeyFrameY	1276
146.8	GetNodeAnimationKeyFrameZ	1276
146.9	SetNodeAnimationKeyFrameRotation	1277
146.10	GetNodeAnimationKeyFramePitch	1277
146.11	GetNodeAnimationKeyFrameYaw	1278
146.12	GetNodeAnimationKeyFrameRoll	1278
146.13	SetNodeAnimationKeyFrameScale	1279
146.14	AddNodeAnimationTime	1279
146.15	StartNodeAnimation	1280
146.16	StopNodeAnimation	1280
146.17	NodeAnimationStatus	1281
146.18	GetNodeAnimationTime	1281
146.19	SetNodeAnimationTime	1282
146.20	GetNodeAnimationLength	1282
146.21	SetNodeAnimationLength	1283
146.22	GetNodeAnimationWeight	1283
146.23	SetNodeAnimationWeight	1283
147	OnError	1285
147.1	OnErrorExit	1285
147.2	OnErrorCall	1286
147.3	OnErrorGoto	1287
147.4	OnErrorDefault	1287
147.5	ErrorCode	1288
147.6	ErrorMessage	1289
147.7	ErrorLine	1289
147.8	ErrorFile	1290
147.9	ErrorAddress	1290
147.10	ErrorTargetAddress	1290
147.11	ErrorRegister	1291
147.12	RaiseError	1292
147.13	ExamineAssembly	1292
147.14	NextInstruction	1293
147.15	InstructionAddress	1294
147.16	InstructionString	1294
148	Packer	1296
148.1	AddPackFile	1296
148.2	AddPackDirectory	1297
148.3	AddPackMemory	1298
148.4	ClosePack	1298
148.5	CompressMemory	1299
148.6	ExaminePack	1300
148.7	NextPackEntry	1300

148.8	PackEntryDate	1301
148.9	PackEntryType	1301
148.10	PackEntrySize	1302
148.11	PackEntryName	1302
148.12	CreatePack	1303
148.13	OpenPack	1304
148.14	CatchPack	1305
148.15	UncompressMemory	1306
148.16	UncompressPackMemory	1307
148.17	UncompressPackFile	1307
148.18	UseZipPacker	1309
148.19	UseLzmaPacker	1309
148.20	UseTarPacker	1310
148.21	UseBriefLZPacker	1310
148.22	UseJcalg1Packer	1311
148.23	IsPack	1311
149	Particle	1312
149.1	CreateParticleEmitter	1312
149.2	IsParticleEmitter	1313
149.3	DisableParticleEmitter	1313
149.4	ParticleEmitterID	1314
149.5	ParticleEmitterX	1314
149.6	ParticleEmitterY	1314
149.7	ParticleEmitterZ	1315
149.8	ParticleEmitterAngle	1315
149.9	ParticleEmissionRate	1316
149.10	ParticleMaterial	1316
149.11	ParticleTimeToLive	1317
149.12	ParticleVelocity	1317
149.13	ParticleAcceleration	1317
149.14	ParticleSize	1318
149.15	ParticleColorRange	1318
149.16	ParticleColorFader	1319
149.17	FreeParticleEmitter	1319
149.18	HideParticleEmitter	1320
149.19	MoveParticleEmitter	1320
149.20	ParticleEmitterDirection	1321
149.21	ResizeParticleEmitter	1321
149.22	GetScriptParticleEmitter	1321
149.23	ParticleSpeedFactor	1322
149.24	ParticleScaleRate	1322
149.25	ParticleAngle	1323
150	Preference	1324
150.1	ClosePreferences	1324
150.2	CreatePreferences	1325
150.3	ExaminePreferenceGroups	1326
150.4	ExaminePreferenceKeys	1327
150.5	FlushPreferenceBuffers	1327
150.6	NextPreferenceGroup	1328
150.7	NextPreferenceKey	1329
150.8	PreferenceGroupName	1329
150.9	PreferenceKeyName	1330
150.10	PreferenceKeyValue	1331
150.11	OpenPreferences	1332
150.12	PreferenceGroup	1333
150.13	PreferenceComment	1334
150.14	ReadPreferenceDouble	1335

150.15	ReadPreferenceFloat	1336
150.16	ReadPreferenceInteger	1337
150.17	ReadPreferenceLong	1337
150.18	ReadPreferenceQuad	1338
150.19	ReadPreferenceString	1338
150.20	RemovePreferenceGroup	1339
150.21	RemovePreferenceKey	1340
150.22	WritePreferenceFloat	1341
150.23	WritePreferenceDouble	1342
150.24	WritePreferenceInteger	1342
150.25	WritePreferenceLong	1343
150.26	WritePreferenceQuad	1344
150.27	WritePreferenceString	1344
151	Printer	1346
151.1	DefaultPrinter	1346
151.2	NewPrinterPage	1346
151.3	PrinterOutput	1347
151.4	PrinterVectorOutput	1348
151.5	PrintRequester	1348
151.6	StartPrinting	1349
151.7	StopPrinting	1349
151.8	PrinterPageWidth	1350
151.9	PrinterPageHeight	1350
152	Process	1351
152.1	AvailableProgramOutput	1351
152.2	CloseProgram	1352
152.3	CountProgramParameters	1352
152.4	EnvironmentVariableName	1353
152.5	EnvironmentVariableValue	1353
152.6	ExamineEnvironmentVariables	1354
152.7	GetEnvironmentVariable	1354
152.8	IsProgram	1355
152.9	KillProgram	1355
152.10	NextEnvironmentVariable	1356
152.11	ProgramExitCode	1356
152.12	ProgramFilename	1357
152.13	ProgramID	1357
152.14	ProgramParameter	1358
152.15	ProgramRunning	1359
152.16	ReadProgramData	1359
152.17	ReadProgramError	1360
152.18	ReadProgramString	1361
152.19	RemoveEnvironmentVariable	1361
152.20	RunProgram	1362
152.21	SetEnvironmentVariable	1363
152.22	WaitProgram	1364
152.23	WriteProgramData	1364
152.24	WriteProgramString	1365
152.25	WriteProgramStringN	1366
153	RegularExpression	1367
153.1	CountRegularExpressionGroups	1367
153.2	CreateRegularExpression	1368
153.3	ExamineRegularExpression	1369
153.4	ExtractRegularExpression	1370
153.5	FreeRegularExpression	1371
153.6	IsRegularExpression	1371

153.7	MatchRegularExpression	1372
153.8	NextRegularExpressionMatch	1372
153.9	RegularExpressionMatchString	1373
153.10	RegularExpressionMatchPosition	1374
153.11	RegularExpressionMatchLength	1374
153.12	RegularExpressionGroup	1375
153.13	RegularExpressionGroupPosition	1376
153.14	RegularExpressionGroupLength	1376
153.15	RegularExpressionNamedGroup	1377
153.16	RegularExpressionNamedGroupPosition	1378
153.17	RegularExpressionNamedGroupLength	1379
153.18	ReplaceRegularExpression	1379
153.19	RegularExpressionError	1380
154	Requester	1382
154.1	ColorRequester	1382
154.2	FontRequester	1383
154.3	InputRequester	1384
154.4	MessageRequester	1385
154.5	NextSelectedFilename	1387
154.6	OpenFileRequester	1387
154.7	PathRequester	1389
154.8	SaveFileRequester	1390
154.9	SelectedFilePattern	1392
154.10	SelectedFontColor	1393
154.11	SelectedFontName	1393
154.12	SelectedFontSize	1394
154.13	SelectedFontStyle	1394
155	Runtime	1396
155.1	GetRuntimeInteger	1396
155.2	GetRuntimeDouble	1397
155.3	GetRuntimeString	1397
155.4	IsRuntime	1398
155.5	SetRuntimeDouble	1398
155.6	SetRuntimeInteger	1399
155.7	SetRuntimeString	1399
156	Scintilla	1400
156.1	InitScintilla	1400
156.2	ScintillaGadget	1401
156.3	ScintillaSendMessage	1402
157	Screen	1404
157.1	ChangeGamma	1404
157.2	ClearScreen	1405
157.3	CloseScreen	1405
157.4	FlipBuffers	1406
157.5	IsScreenActive	1406
157.6	ScreenID	1407
157.7	ScreenWidth	1407
157.8	ScreenHeight	1408
157.9	ScreenDepth	1408
157.10	SetFrameRate	1409
157.11	OpenScreen	1409
157.12	OpenWindowedScreen	1410
157.13	ScreenOutput	1413
157.14	ExamineScreenModes	1414
157.15	NextScreenMode	1414
157.16	ScreenModeDepth	1415

157.17	ScreenModeHeight	1416
157.18	ScreenModeRefreshRate	1416
157.19	ScreenModeWidth	1417
158	SerialPort	1418
158.1	AvailableSerialPortInput	1418
158.2	AvailableSerialPortOutput	1418
158.3	CloseSerialPort	1419
158.4	GetSerialPortStatus	1419
158.5	IsSerialPort	1420
158.6	SerialPortError	1420
158.7	SerialPortID	1421
158.8	OpenSerialPort	1422
158.9	ReadSerialPortData	1423
158.10	SerialPortTimeouts	1423
158.11	SetSerialPortStatus	1424
158.12	WriteSerialPortData	1425
158.13	WriteSerialPortString	1426
159	Skeleton	1427
159.1	CreateSkeleton	1427
159.2	CreateBone	1427
159.3	VertexBoneAssignment	1428
159.4	FinishBoneAssignment	1429
159.5	CreateSkeletonAnimation	1429
159.6	AddSkeletonAnimationKeyFrame	1430
160	Sort	1432
160.1	CustomSortArray	1432
160.2	CustomSortList	1434
160.3	SortArray	1436
160.4	SortList	1437
160.5	SortStructuredArray	1437
160.6	SortStructuredList	1439
160.7	RandomizeArray	1441
160.8	RandomizeList	1441
161	Sound	1443
161.1	CatchSound	1443
161.2	GetSoundPosition	1444
161.3	SetSoundPosition	1445
161.4	FreeSound	1446
161.5	InitSound	1447
161.6	IsSound	1448
161.7	LoadSound	1449
161.8	PauseSound	1450
161.9	ResumeSound	1451
161.10	PlaySound	1452
161.11	GetSoundFrequency	1453
161.12	SetSoundFrequency	1454
161.13	SoundStatus	1455
161.14	SoundPan	1456
161.15	SoundLength	1457
161.16	SoundVolume	1458
161.17	StopSound	1459
162	Sound3D	1461
162.1	FreeSound3D	1461
162.2	IsSound3D	1461
162.3	LoadSound3D	1462

162.4	PlaySound3D	1462
162.5	SoundVolume3D	1463
162.6	StopSound3D	1463
162.7	SoundID3D	1464
162.8	SoundRange3D	1464
162.9	SoundCone3D	1465
162.10	SoundListenerLocate	1465
163	SoundPlugin	1466
163.1	UseFLACSoundDecoder	1466
163.2	UseOGGSoundDecoder	1466
164	SpecialEffect	1468
164.1	CreateCompositorEffect	1468
164.2	CreateRibbonEffect	1469
164.3	RibbonEffectWidth	1469
164.4	AttachRibbonEffect	1470
164.5	DetachRibbonEffect	1470
164.6	CreateLensFlareEffect	1471
164.7	LensFlareEffectColor	1471
164.8	FreeEffect	1472
164.9	IsEffect	1472
164.10	HideEffect	1473
164.11	CompositorEffectParameter	1473
164.12	RibbonEffectColor	1474
165	Spline	1475
165.1	CreateSpline	1475
165.2	FreeSpline	1476
165.3	AddSplinePoint	1476
165.4	ClearSpline	1477
165.5	CountSplinePoints	1477
165.6	SplinePointX	1478
165.7	SplinePointY	1478
165.8	SplinePointZ	1479
165.9	UpdateSplinePoint	1479
165.10	ComputeSpline	1480
165.11	SplineX	1480
165.12	SplineY	1480
165.13	SplineZ	1481
166	Sprite	1482
166.1	CatchSprite	1482
166.2	ClipSprite	1483
166.3	CopySprite	1484
166.4	CreateSprite	1484
166.5	DisplaySprite	1486
166.6	DisplayTransparentSprite	1486
166.7	FreeSprite	1487
166.8	GrabSprite	1487
166.9	InitSprite	1488
166.10	IsSprite	1489
166.11	LoadSprite	1489
166.12	SaveSprite	1490
166.13	SpriteCollision	1491
166.14	SpriteDepth	1491
166.15	SpriteHeight	1492
166.16	SpriteID	1492
166.17	SpritePixelCollision	1493
166.18	SpriteWidth	1493

166.19	SpriteOutput	1494
166.20	TransparentSpriteColor	1494
166.21	RotateSprite	1495
166.22	SpriteBlendingMode	1495
166.23	SpriteQuality	1496
166.24	TransformSprite	1496
166.25	ZoomSprite	1497
167	StaticGeometry	1498
167.1	FreeStaticGeometry	1498
167.2	IsStaticGeometry	1498
167.3	CreateStaticGeometry	1499
167.4	AddStaticGeometryEntity	1500
167.5	BuildStaticGeometry	1501
168	StatusBar	1502
168.1	AddStatusBarField	1502
168.2	CreateStatusBar	1503
168.3	FreeStatusBar	1504
168.4	IsStatusBar	1504
168.5	StatusBarImage	1505
168.6	StatusBarID	1506
168.7	StatusBarText	1506
168.8	StatusBarProgress	1507
168.9	StatusBarHeight	1507
169	String	1509
169.1	Asc	1509
169.2	Bin	1510
169.3	Chr	1511
169.4	CountString	1512
169.5	EscapeString	1512
169.6	FindString	1513
169.7	Hex	1514
169.8	InsertString	1515
169.9	LCase	1516
169.10	Left	1516
169.11	Len	1517
169.12	LSet	1517
169.13	LTrim	1518
169.14	Mid	1519
169.15	RemoveString	1519
169.16	ReplaceString	1520
169.17	Right	1521
169.18	RSet	1522
169.19	RTrim	1522
169.20	StringByteLength	1523
169.21	StringField	1524
169.22	StrF	1524
169.23	StrD	1525
169.24	Str	1526
169.25	StrU	1526
169.26	ReverseString	1527
169.27	Space	1528
169.28	Trim	1528
169.29	UCase	1529
169.30	UnescapeString	1529
169.31	ValD	1530
169.32	ValF	1531

169.33	Val	1532
169.34	Ascii	1532
169.35	UTF8	1533
169.36	FormatNumber	1534
170	SysTray	1535
170.1	AddSysTrayIcon	1535
170.2	ChangeSysTrayIcon	1536
170.3	IsSysTrayIcon	1537
170.4	SysTrayIconMenu	1538
170.5	SysTrayIconToolTip	1539
170.6	RemoveSysTrayIcon	1539
171	System	1541
171.1	CocoaMessage	1541
171.2	CPUName	1542
171.3	Delay	1543
171.4	ElapsedMilliseconds	1543
171.5	DoubleClickTime	1544
171.6	OSVersion	1545
171.7	ComputerName	1546
171.8	UserName	1547
171.9	MemoryStatus	1547
171.10	CountCPUs	1548
172	Terrain	1549
172.1	FreeTerrain	1549
172.2	FreeTerrainBody	1550
172.3	SetupTerrains	1550
172.4	CreateTerrain	1551
172.5	CreateTerrainBody	1551
172.6	DefineTerrainTile	1552
172.7	AddTerrainTexture	1553
172.8	BuildTerrain	1553
172.9	TerrainLocate	1554
172.10	TerrainHeight	1554
172.11	TerrainTileHeightAtPosition	1555
172.12	TerrainTilePointX	1555
172.13	TerrainTilePointY	1556
172.14	TerrainTileSize	1556
172.15	GetTerrainTileHeightAtPoint	1557
172.16	SetTerrainTileHeightAtPoint	1557
172.17	UpdateTerrain	1558
172.18	TerrainTileLayerMapSize	1558
172.19	GetTerrainTileLayerBlend	1559
172.20	SetTerrainTileLayerBlend	1559
172.21	UpdateTerrainTileLayerBlend	1560
172.22	TerrainMousePick	1560
172.23	SaveTerrain	1561
172.24	TerrainRenderMode	1561
173	Text3D	1563
173.1	CreateText3D	1563
173.2	FreeText3D	1564
173.3	Text3DID	1564
173.4	IsText3D	1564
173.5	MoveText3D	1565
173.6	ScaleText3D	1566
173.7	Text3DCaption	1566
173.8	Text3DColor	1567

173.9	Text3DAlignment	1567
173.10	Text3DX	1568
173.11	Text3DY	1568
173.12	Text3DZ	1568
174	Texture	1570
174.1	CopyTexture	1570
174.2	CreateTexture	1570
174.3	CreateCubicTexture	1571
174.4	CreateRenderTexture	1572
174.5	UpdateRenderTexture	1573
174.6	SaveRenderTexture	1574
174.7	CreateCubeMapTexture	1574
174.8	EntityCubeMapTexture	1575
174.9	FreeTexture	1576
174.10	IsTexture	1576
174.11	GetScriptTexture	1577
174.12	LoadTexture	1577
174.13	TextureID	1578
174.14	TextureHeight	1578
174.15	TextureOutput	1578
174.16	TextureWidth	1579
175	Thread	1580
175.1	IsThread	1580
175.2	ThreadID	1581
175.3	CreateMutex	1581
175.4	CreateThread	1582
175.5	FreeMutex	1584
175.6	KillThread	1584
175.7	LockMutex	1585
175.8	PauseThread	1586
175.9	ResumeThread	1586
175.10	ThreadPriority	1587
175.11	TryLockMutex	1588
175.12	UnlockMutex	1589
175.13	WaitThread	1590
175.14	CreateSemaphore	1591
175.15	FreeSemaphore	1592
175.16	SignalSemaphore	1593
175.17	WaitSemaphore	1593
175.18	TrySemaphore	1594
176	ToolBar	1595
176.1	CreateToolBar	1595
176.2	FreeToolBar	1596
176.3	DisableToolBarButton	1597
176.4	GetToolBarButtonState	1598
176.5	IsToolBar	1599
176.6	SetToolBarButtonState	1599
176.7	ToolBarHeight	1600
176.8	ToolBarImageButton	1600
176.9	ToolBarSeparator	1601
176.10	ToolBarButtonText	1602
176.11	ToolBarToolTip	1603
176.12	ToolBarID	1604
177	VectorDrawing	1605
177.1	StartVectorDrawing	1607
177.2	StopVectorDrawing	1608

177.3	VectorOutputWidth	1608
177.4	VectorOutputHeight	1609
177.5	VectorResolutionX	1609
177.6	VectorResolutionY	1610
177.7	VectorUnit	1610
177.8	SaveVectorState	1611
177.9	RestoreVectorState	1612
177.10	BeginVectorLayer	1612
177.11	EndVectorLayer	1614
177.12	NewVectorPage	1614
177.13	FillVectorOutput	1615
177.14	ResetCoordinates	1615
177.15	TranslateCoordinates	1616
177.16	ScaleCoordinates	1618
177.17	RotateCoordinates	1619
177.18	SkewCoordinates	1621
177.19	FlipCoordinatesX	1623
177.20	FlipCoordinatesY	1624
177.21	ConvertCoordinateX	1626
177.22	ConvertCoordinateY	1627
177.23	ResetPath	1628
177.24	ClosePath	1628
177.25	MovePathCursor	1629
177.26	AddPathLine	1631
177.27	AddPathArc	1632
177.28	AddPathCurve	1633
177.29	AddPathBox	1634
177.30	AddPathCircle	1635
177.31	AddPathEllipse	1637
177.32	AddPathText	1638
177.33	AddPathSegments	1640
177.34	IsInsidePath	1641
177.35	IsInsideStroke	1642
177.36	IsPathEmpty	1644
177.37	StrokePath	1644
177.38	DotPath	1646
177.39	DashPath	1647
177.40	CustomDashPath	1649
177.41	FillPath	1650
177.42	ClipPath	1651
177.43	PathCursorX	1653
177.44	PathCursorY	1653
177.45	PathPointX	1654
177.46	PathPointY	1655
177.47	PathPointAngle	1655
177.48	PathLength	1656
177.49	PathBoundsX	1657
177.50	PathBoundsY	1658
177.51	PathBoundsWidth	1659
177.52	PathBoundsHeight	1659
177.53	PathSegments	1660
177.54	VectorSourceColor	1661
177.55	VectorSourceLinearGradient	1661
177.56	VectorSourceCircularGradient	1662
177.57	VectorSourceGradientColor	1664
177.58	VectorSourceImage	1664
177.59	DrawVectorImage	1666
177.60	DrawVectorText	1668
177.61	DrawVectorParagraph	1669

177.62	VectorFont	1671
177.63	VectorTextWidth	1671
177.64	VectorTextHeight	1673
177.65	VectorParagraphHeight	1675
177.66	PdfVectorOutput	1676
177.67	SvgVectorOutput	1677
178	Vehicle	1678
178.1	AddVehicleWheel	1678
178.2	ApplyVehicleForce	1679
178.3	ApplyVehicleBrake	1679
178.4	ApplyVehicleSteering	1680
178.5	CreateVehicle	1680
178.6	CreateVehicleBody	1681
178.7	GetVehicleAttribute	1682
178.8	SetVehicleAttribute	1683
179	VertexAnimation	1684
179.1	CreateVertexAnimation	1684
179.2	CreateVertexTrack	1684
179.3	CreateVertexPoseKeyFrame	1685
179.4	AddVertexPoseReference	1685
179.5	UpdateVertexPoseReference	1686
179.6	VertexPoseReferenceCount	1687
179.7	MeshPoseCount	1687
179.8	MeshPoseName	1688
180	WebView	1689
180.1	WebViewGadget	1689
180.2	BindWebViewCallback	1692
180.3	UnbindWebViewCallback	1694
180.4	WebViewExecuteScript	1694
180.5	WebViewProxy	1695
181	Window	1697
181.1	AddKeyboardShortcut	1697
181.2	AddWindowTimer	1700
181.3	RemoveWindowTimer	1702
181.4	EventTimer	1702
181.5	CloseWindow	1703
181.6	DisableWindow	1703
181.7	Event	1704
181.8	EventGadget	1704
181.9	EventMenu	1705
181.10	EventData	1706
181.11	EventType	1706
181.12	EventWindow	1708
181.13	GetActiveWindow	1708
181.14	GetWindowColor	1709
181.15	GetWindowData	1709
181.16	GetWindowState	1709
181.17	GetWindowTitle	1710
181.18	HideWindow	1711
181.19	IsWindow	1712
181.20	OpenWindow	1712
181.21	PostEvent	1714
181.22	RemoveKeyboardShortcut	1715
181.23	ResizeWindow	1716
181.24	SetActiveWindow	1716
181.25	SetWindowCallback	1717

181.26	SetWindowColor	1718
181.27	SetWindowData	1719
181.28	SetWindowState	1719
181.29	SetWindowTitle	1720
181.30	SmartWindowRefresh	1721
181.31	StickyWindow	1721
181.32	WindowEvent	1722
181.33	WaitWindowEvent	1725
181.34	BindEvent	1727
181.35	UnbindEvent	1729
181.36	WindowBounds	1730
181.37	WindowHeight	1731
181.38	WindowID	1731
181.39	WindowWidth	1732
181.40	WindowX	1732
181.41	WindowY	1733
181.42	WindowMouseX	1733
181.43	WindowMouseY	1734
181.44	WindowOutput	1735
181.45	WindowVectorOutput	1736
181.46	EventwParam	1737
181.47	EventlParam	1737
182	Window3D	1738
182.1	CloseWindow3D	1738
182.2	DisableWindow3D	1738
182.3	EventGadget3D	1739
182.4	EventType3D	1739
182.5	EventWindow3D	1740
182.6	InputEvent3D	1740
182.7	GetActiveWindow3D	1741
182.8	GetWindowTitle3D	1742
182.9	HideWindow3D	1742
182.10	IsWindow3D	1743
182.11	OpenWindow3D	1743
182.12	ResizeWindow3D	1744
182.13	SetActiveWindow3D	1744
182.14	SetWindowTitle3D	1745
182.15	WindowEvent3D	1745
182.16	WindowHeight3D	1746
182.17	WindowID3D	1746
182.18	WindowWidth3D	1747
182.19	WindowX3D	1747
182.20	WindowY3D	1748
183	XML	1749
183.1	IsXML	1750
183.2	FreeXML	1750
183.3	CreateXML	1751
183.4	LoadXML	1752
183.5	CatchXML	1752
183.6	ParseXML	1753
183.7	XMLStatus	1754
183.8	XMLError	1755
183.9	XMLErrorLine	1756
183.10	XMLErrorPosition	1756
183.11	SaveXML	1757
183.12	ExportXMLSize	1757
183.13	ExportXML	1758

183.14	ComposeXML	1759
183.15	FormatXML	1759
183.16	GetXMLEncoding	1760
183.17	SetXMLEncoding	1761
183.18	GetXMLStandalone	1761
183.19	SetXMLStandalone	1762
183.20	RootXMLNode	1762
183.21	MainXMLNode	1763
183.22	ChildXMLNode	1763
183.23	ParentXMLNode	1764
183.24	XMLChildCount	1764
183.25	NextXMLNode	1765
183.26	PreviousXMLNode	1765
183.27	XMLNodeFromPath	1766
183.28	XMLNodeFromID	1767
183.29	XMLNodeType	1767
183.30	GetXMLNodeText	1768
183.31	SetXMLNodeText	1769
183.32	GetXMLNodeOffset	1769
183.33	SetXMLNodeOffset	1770
183.34	GetXMLNodeName	1770
183.35	SetXMLNodeName	1771
183.36	XMLNodePath	1771
183.37	GetXMLAttribute	1772
183.38	SetXMLAttribute	1772
183.39	RemoveXMLAttribute	1773
183.40	ExamineXMLAttributes	1773
183.41	NextXMLAttribute	1774
183.42	XMLAttributeName	1774
183.43	XMLAttributeValue	1775
183.44	CreateXMLNode	1775
183.45	CopyXMLNode	1776
183.46	MoveXMLNode	1777
183.47	DeleteXMLNode	1777
183.48	ResolveXMLNodeName	1778
183.49	ResolveXMLAttributeName	1778
183.50	InsertXMLArray	1779
183.51	InsertXMLList	1780
183.52	InsertXMLMap	1782
183.53	InsertXMLStructure	1783
183.54	ExtractXMLArray	1784
183.55	ExtractXMLList	1785
183.56	ExtractXMLMap	1786
183.57	ExtractXMLStructure	1787

Part I

General

Chapter 1

Introduction

PureBasic is an "high-level" programming language based on established "BASIC" rules. It is mostly compatible with any other "BASIC" compiler. Learning PureBasic is very easy! PureBasic has been created for beginners and experts alike. Compilation time is extremely fast. This software has been developed for the Windows operating system. We have put a lot of effort into its realization to produce a fast, reliable and system-friendly language.

The syntax is easy and the possibilities are huge with the "advanced" functions that have been added to this language like pointers, structures, procedures, dynamic lists, maps, interfaces, modular programming (modules), an inline assembler and much more. For the experienced coder, there are no problems gaining access to any of the legal OS structures or API objects.

PureBasic is a portable programming language which currently works on Linux, Mac OS X and Windows computer systems and machines running Raspberry PI OS. This means that the same code can be compiled natively for the OS and use the full power of each. There are no bottlenecks like a virtual machine or a code translator, the generated code produces an optimized executable.

The main features of PureBasic

- Intel/AMD and others' x86 and x64, M1, M2, M3, arm32 and arm64 support
- 86 native libraries
- Hundreds of functions
- Built-in arrays, dynamic lists, complex structures, maps, pointers and variable definitions
- Supported types: Byte (8-bit), Word (16-bit), Long (32-bit), Quad (64-bit), Float (32-bit), Double (64-bit) and Characters
- User defined types (structures)
- Built-in string types (characters), including ascii and unicode
- Powerful macro support
- Constants, binary and hexadecimal numbers supported
- Expression reducer by grouping constants and numeric numbers together
- Standard arithmetic support in respect of sign priority and parenthesis: +, -, /, *, and, or, «, »
- Extremely fast compilation
- Procedure support for structured programming with local and global variables
- All Standard BASIC keywords: If-Else-EndIf, Repeat-Until, etc
- Specialized libraries to manipulate BMP pictures, windows, gadgets, DirectX, etc
- Specialized libraries are very optimized for maximum speed and compactness
- The Win32 API is fully supported as if they were BASIC keywords
- Inline Assembler
- Precompiled structures with constants files for extra-fast compilation
- Integrated debugger to follow the execution of a program and correct errors more easily
- Configurable CLI compiler
- Dedicated editor with automatically highlighted syntax
- Very high productivity, comprehensive keywords, online help

- System friendly, easy to install and easy to use
- IDE and help available in english, french and german
- SDK Visual C
- Compilation of windowed program, console and DLL
- Creation of integrated wysiwyg windows
- Creation of windows with automatic reorganization of gadgets (layout) with the 'Dialog' library
- 33 native gadgets
- 1 gadget specialized in OpenGL
- 1 scintilla gadget
- 1 webview gadget to easily create HTML/CSS based user interface
- Process, thread, mutex, semaphore
- Drag'n drop
- DPI for MS Windows and OSX
- QT and GTK3 for Linux
- DirectX and OpenGL systems
- Import of static (lib) or dynamic libraries (dll, so, etc.)
- Drawing with antialiasing with the 'Vector' library
- 3D functions with the 3D engine OGRE
- Pseudotypes: p-ascii, p-utf8, p-bstr, p-unicode, p-variant
- Array, linked-list (list), maps, Database (MySQL, SQLite, ODBC, PostgreSQL, Maria)
- json, xml
- Regular expression
- http, ftp, mail, application server / client, CGI and FastCGI
- Cypher
- OnError
- Compressor/decompressor: BriefLZ, JCALG1, LZMA, Tar, Zip
- ini file
- Printer, serial port
- Runtime

Here is the exhaustive list of all the libraries that PureBasic offers:

Windows & system :

```

arrays
cgi
cipher
clipboard
console
database
date
debugger
desktop
dialog
dragdrop
file
filesystem
font
ftp
gadget
help
http
json
library
lists
mail
maps
math
memory
menu
network
onerror

```

- packer
- preference
- printer
- process
- regularexpression
- requester
- runtimes
- scintilla
- serialport
- sort
- statusbar
- string
- system
- systray
- thread
- toolbar
- webview
- window
- xml

2D & Media :

- 2ddrawing
- audiocd
- font
- image
- imageplugin
- joystick
- keyboard
- mouse
- movie
- music
- screen
- sound
- soundplugin
- sprite
- vectordrawing

3D Engine :

- billboard
- camera
- engine3d
- entity
- entityanimation
- gadget3d
- joint
- light
- material
- mesh
- node
- nodeanimation
- particle
- skeleton
- sound3d
- specialeffect
- spline
- staticgeometry
- terrain
- text3d
- texture

vehicle
vertexanimation
window3d

Chapter 2

Terms And Conditions

This program is provided "**AS IS**". Fantaisie Software are NOT responsible for any damage (or damages) attributed to PureBasic. You are warned that you use PureBasic at your own risk. No warranties are implied or given by Fantaisie Software or any representative.

The demo version of this program may be freely distributed provided all contents, of the original archive, remain intact. You may not modify, or change, the contents of the original archive without express written consent from Fantaisie Software.

PureBasic has an user-based license. This means you can install it on every computer you need but you can't share it between two or more people.

All components, libraries, and binaries are copyrighted by Fantaisie Software. The PureBasic license explicitly forbids the creation of DLLs whose primary function is to serve as a 'wrapper' for PureBasic functions.

Fantaisie Software reserves all rights to this program and all original archives and contents.

PureBasic uses the Scintilla editing component and its related license can be consulted [here](#) .

PureBasic uses GCC compiler and its related license can be consulted [here](#) and [here](#)

The PureBasic uses FASM flat assembler and its related license can be consulted [here](#)

The PureBasic uses PellesC toolchain and its related license can be consulted [here](#)

PureBasic built-in libraries uses opensources components, the licenses can be consulted [here](#) and [here](#)

Chapter 3

System requirements

PureBasic will run on Windows Vista, Windows 7, Windows 8, Windows 10 and Windows 11 (in both 32-bit and 64-bit edition), Linux (kernel 2.2 or above), MacOS X (10.15 or above) and Raspberry PI OS (bookworm/debian 12 or above).

If there are any problems, please contact us: support@purebasic.com

Windows

For many graphic functions like 3D or Sprites there is needed the DirectX9.0c version. Windows Vista and later don't have installed this from start, so it need to be installed manually.

You can download and install it via the "DirectX End-User Runtime Web Installer" here:

<http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=35>.

Linux

Required packages to use PureBasic:

- sdl 1.2 devel (for 'Joystick' and 'AudioCD' libraries)
- gtk 3 (For GUI programs)
- libstdc++ devel
- gcc correctly installed
- iodb and iodb-devel to be able to use the Database commands (see www.iodb.org)
- libwebkit should be installed to have the WebGadget() working.
- libvlc-devel for the Movie commands Check the INSTALL and README file for more information.

MacOS X

You need to install the Apple developer tools to be able to use PureBasic. The tools can be found either on one of the install CDs or on the Apple web site: <http://developer.apple.com/>

Be aware to use the latest version of the tools (e.g. XCode), which is needed for your OS version! The commandline tools needs to be installed from XCode once installed.

Chapter 4

Installation

To install PureBasic, just click on the install wizard, follow the steps, and then click on the PureBasic icon (found on the desktop or in the start-menu) to launch PureBasic.

To use the command line compiler, open a standard command line window (CMD) and look in the "Compilers\" subdirectory for PBCompiler.exe. It's a good idea to consider adding the "PureBasic\Compilers\" directory to the PATH environment variable to make the compiler accessible from any directory.

Important note: to avoid conflicts with existing PureBasic installations (maybe even with user-libraries), please install a new PureBasic version always in its own new folder. See also the chapter Using several PureBasic versions .

Chapter 5

Order

PureBasic is a low-cost programming language. In buying PureBasic you will ensure that development will go further and faster. For personal use (i.e.: not used by a commercial organization) the updates are unlimited, unlike most other software out there. This means when you buy PureBasic you will get all future updates for free, on the web site. Better still, you get the four versions of PureBasic (MacOSX, Linux, Raspberry Pi, and Windows) for the same price! For ease of ordering, you can safely use our secure online method. Thanks a lot for your support!

The demo-version of PureBasic is limited as shown below:

- No DLLs can be created
- you can't use the whole external Win32 API support
- no development kit for external libraries
- maximum number of source lines: 800

Full version of PureBasic:

Price for full version: 79 Euros

There are special prices for company licences (499 Euros) and educational licenses (199 Euros for a school class).

Online Ordering

Payment online (www.purebasic.com) is available and very secure. (Note: Such orders are handled directly by Fred.)

If you live in Germany or Europe and prefer paying to bank account you can also send your registration to the German team member. In this case please send your order to following address:

André Beer
Siedlung 6
09548 Deutschneudorf
Germany
e-mail: andre@purebasic.com

Bank Account:

Deutsche Kreditbank AG
Account 15920010 - Bank code 12030000
(For transactions from EU countries: IBAN: DE03120300000015920010 -
BIC/Swift-Code: BYLADEM1001)

Paypal:
andrebeer@gmx.de
(This address can be used for Paypal transaction, if you want
personal contact to or an invoice from André.)

Delivering of the full version

The full version will be provided via your personal download account, which you will get on www.purebasic.com after successful registration. If you order from André, just write an e-mail with your full address or use this registration form and print it or send it via e-mail.

Chapter 6

Contact

Please send bug reports, suggestions, improvements, examples of source coding, or if you just want to talk to us, to any of the following addresses:

Frédéric 'AlphaSND' Laboureur

Fred 'AlphaSND' is the founder of Fantaisie Software and the main coder for PureBasic. All suggestions, bug reports, etc. should be sent to him at either address shown below:

s-mail :

Frédéric Laboureur
10, rue de Lausanne
67640 Fegersheim
France

e-mail : fred@purebasic.com

André Beer

André is responsible for the complete German translation of the PureBasic manual and website.

PureBasic can be ordered in Germany also directly at him.

Just write an email with your full address (for the registration) to him. If needed you can also get an invoice from him. For more details just take a look [here](#).

e-mail : andre@purebasic.com

Chapter 7

Acknowledgements

We would like to thank the many people who have helped in this ambitious project. It would not have been possible without them !

- All the registered users: To support this software... Many Thanks !

Coders

- **Timo 'Fr34k' Harter**: For the IDE, Debugger, many commands and the great ideas. PureBasic wouldn't be the same without him !
- **Benny 'Berikco' Sels**: To provide an amazing first class Visual Designer for PureBasic. That's great to have such kind of help !
- **Danilo Krahn**: To have done an huge work on the editor and on the core command set, without forget the tons of nice suggestions about code optimization and size reduction... Thanks a lot.
- **Sebastion 'PureFan' Lackner**: For the very nice OnError library, which adds professional error handling for final executables and several other commands !
- **Siegfried Rings**: Also for the very nice OnError library, as its a two men work :)
- **Marc Vitry**: To have kindly shared its SerialPort library sources which helped a lot for the native PureBasic library.
- **Stefan Moebius**: To have sent its DX9 subsystem sources, which have been the basis of the current DX9 subsystem for PureBasic.
- **Comtois**, "G-Rom" and "T-Myke": for the big help improving the 3D side of PureBasic ! That matters !
- **Gaetan Dupont-Panon**: For the wonderful new visual designer, which really rocks on Windows, Linux and OS X !

Miscellaneous

- **Roger Beausoleil**: The first to believe in this project, and his invaluable help with the main design and layout of PureBasic.
- **Andre Beer**: To spend time for improving the guides (including beginners guide) and do the complete translation into German. Big thanks!
- **Francis G. Loch**: To have corrected all the mistakes in the English guides ! Thanks again.
- **Steffen Haeuser**: For giving his valuable time, and assistance. For his invaluable explanations regarding aspects of PPC coding, and his very useful tips.
- **Martin Konrad**: A fantastic bug reporter for the Amiga version. To have done the forthcoming Visual Environment for PureBasic and some nice games.

- **James L. Boyd:** To have found tons of bugs in the x86 version and gave us tons of work :). Keep up the bug hunt !
- **Les:** For editing the English Fantaisie Software Web site & and this guide. This looks much better!
- **NAsm team:** To have done an incredible x86 assembler used to develop the PureBasic libraries
- **Tomasz Grysztar:** For FAsm, an excellent optimized x86 assembler currently used by PureBasic.
<http://flatassembler.net/>
- **Pelle Orinius:** For the very good linker and resource compiler (from PellesC) currently used by PureBasic, and the invaluable and instant help when integrating the new linker in the PureBasic package. Thank you !
- **Jacob Navia:** For the linker (from the Lcc32 package) which was used in previous PureBasic versions
- **Thomas Richter:** For creating "PoolMem." A patch that decreases the compilation time by a factor of 2-3! Many thanks for allowing it to be added to the main archive.
- **Frank Wille:** For allowing the use of your excellent assemblers: "pasm," and "PhxAss". For all your tips, and invaluable assistance, in their usage. To help us a lot about debugging PowerPC executables.
- **Rings, Paul, Franco and MrVain/Secretly:** For extensive bug reports and help us to have a compiler as reliable and bug free as possible.
- **David 'Tinman' McMinn:** To enhance greatly the help and give the opportunity to the world to really discover PureBasic !
- **David 'spikey' Roberts:** To help creating the PureBasic beginners guide. Thanks a lot!
- **Leigh:** To have created and hosted the first PureBasic forums during 2 years. It has been a great amount of work, especially when the community started to grow. Thanks again !
- **Jean R. VIALE:** To have greatly improved the french documentation, and still doing it !

Part II

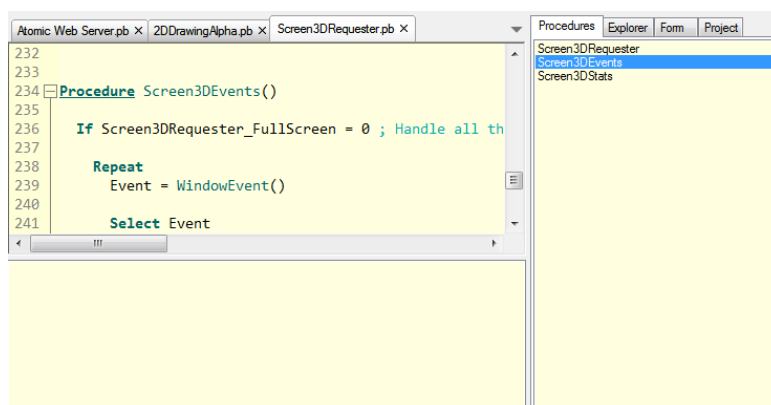
The PureBasic Editor

Chapter 8

Getting Started

The PureBasic IDE allows you to create and edit your PureBasic source codes, as well as run them, debug them and create the final executable. It has both an interface to the PureBasic Compiler , as well to the PureBasic Debugger .

The IDE main window contains of 3 major parts:



The code editing area (below the toolbar)

Here all the source codes are displayed. You can switch between them with the tabs located right above it.

The tools panel (on the right side by default)

Here you have several tools to make coding easier and increase productivity. The tools displayed here can be configured, and it can even be completely removed. See Customizing the IDE for more information.

The error log (located below the editing area)

In this area, the compiler errors and debugger messages are logged. It can be hidden/shown for each source code separately.

Other than that, there is the main menu and the toolbar. The toolbar simply provides shortcuts to menu features. It can be fully customized. To find out what each button does, move your mouse over it and wait until a small tool-tip appears. It shows the corresponding menu command. The menu commands are explained in the other sections.

Chapter 9

Working with source files

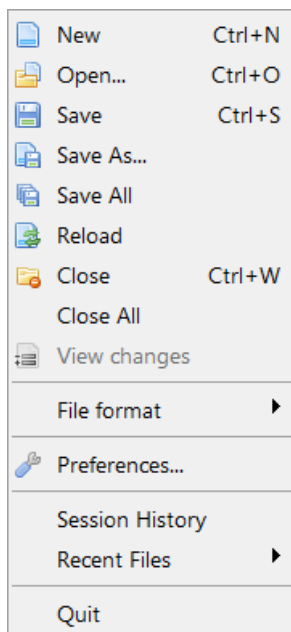
The file menu allows you to do basic file operations like opening and saving source codes.

You can edit multiple source code files at the same time. You can switch between them using the panel located under the Toolbar. Also the shortcut keys Ctrl+Tab and Ctrl+Shift+Tab can be used to jump to the next or previous open source file, respectively.

The IDE allows the editing of non-sourcecode text files. In this "plain text" mode, code-related features such as coloring, case correction, auto complete are disabled. When saving plain text files, the IDE will not append its settings to the end of the file, even if this is configured for code files in the Preferences .

Whether or not a file is considered a code-file or not depends on the file extension. The standard PureBasic file extensions (pb, pbi and pbf) are recognized as code files. More file extensions can be recognized as code files by configuring their extension in the "Editor" section of the Preferences .

Contents of the "File" menu:



New

Create a new empty source code file.

Open

Open an existing source code file for editing.

Any text file will be loaded into the source-editing field. You can also load binary files with the Open menu. These will be displayed in the internal File Viewer .

Save

Saves the currently active source to disk. If the file isn't saved yet, you will be prompted for a filename. Otherwise the code will be saved in the file it was saved in before.

Save As...

Save the currently active source to a different location than it was saved before. This prompts you for a new filename and leaves the old file (if any) untouched.

Save All

Saves all currently opened sources.

Reload

Reloads the currently active source code from disk. This discards any changes not yet saved.

Close

Closes the currently active source code. If it was the only open code, the IDE will display a new empty file.

Close All

Closes all currently opened sources.

View changes

Shows the changes made to the current source code compared to its version that exists on the hard drive.

File format

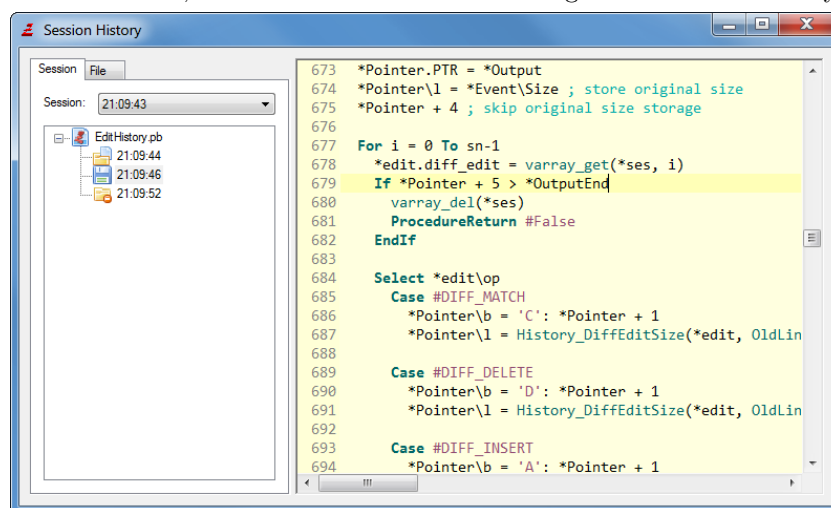
In this submenu you can select the text encoding as well as the newline format which should be used when the currently active source code is saved to disk. The IDE can handle files in Ascii or UTF-8. The newline formats it can handle are Windows (CRLF), Linux/Unix (LF) and MacOSX (CR). The defaults for newly created source codes can be set in the preferences .

Preferences

Here you can change all the settings that control the look & behavior of the IDE. For a detailed description of that see Customizing the IDE .

Session history

Session history is a powerful tool which regularly records changes made to any files in a database. A session is created when the IDE launch, and is closed when the IDE quits. This is useful to rollback to a previous version of a file, or to find back a deleted or corrupted file. It's like source backup tool, limited in time (by default one month of recording). It's not aimed to replace a real source code version control system like SVN or GIT. It's complementary to have finer change trace. The source code will be stored without encryption, so if you are working on sensitive source code, be sure to have this database file in a secure location, or disable this feature. To configure the session history tool, see preferences .



Recent Files

Here you can see a list of the last accessed files. Selecting a file in this submenu will open it again.

Quit

This of course closes the IDE. You will be asked to save any non-saved source codes.

Chapter 10

Editing features

The PureBasic IDE acts like any other Text Editor when it comes to the basic editing features. The cursor keys as well as Page Up/Page Down, Home and End keys can be used to navigate through the code. Ctrl+Home navigates to the beginning of the file and Ctrl+End to the End.

The default shortcuts Ctrl+C (copy), Ctrl+X (cut) and Ctrl+V (paste) can be used for editing. The "Insert" key controls whether text is inserted or overwritten. The Delete key does a forward delete.

Holding down the Shift key and using the arrow keys selects text.

The shortcuts Ctrl+Shift+Up and Ctrl+Shift+Down allow you to move a line up or down.

The shortcut Ctrl+D duplicates a selection.

In Windows, the shortcuts Ctrl+Plus and Ctrl+Minus allow you to zoom in and out the display of all tabs and Ctrl+0 restores the default zoom. Linux and OSX users can create them themselves. The shortcut Ctrl+mouse wheel only zooms the current tab.

The shortcuts Ctrl+Tab and Ctrl+Shift+Tab allow you to move around the open tabs.

There are many shortcuts available, like Ctrl+Shift+U/L/X to change the case, etc. Go to Preferences (menu "Files" / "Preferences / General / Shortcuts").

Furthermore, the IDE has many extra editing features specific to programming or PureBasic.

Indentation:

When you press enter, the indentation (number of space/tab at the beginning of the line) of the current and next line will be automatically corrected depending on the keywords that exist on these lines. A "block mode" is also available where the new line simply gets the same indentation as the previous one. The details of this feature can be customized in the preferences .

Tab characters:

By default, the IDE does not insert a real tab when pressing the Tab key, as many programmers see it as a bad thing to use real tabs in source code.

It instead inserts two spaces. This behavior can be changed in the Preferences. See Customizing the IDE for more information.

Special Tab behavior:

When the Tab key is pressed while nothing or only a few characters are selected, the Tab key acts as mentioned above (inserting a number of spaces, or a real tab if configured that way).

However when one or more full lines are selected, the reaction is different. In that case at the beginning of each selected line, it will insert spaces or a tab (depending on the configuration). This increases the indentation of the whole selected block.

Marking several lines of text and pressing Shift+Tab reverses this behavior. It removes spaces/tabs at the start of each line in order to reduce the indentation of the whole block.

Indentation/Alignment of comments:

Similar to the special tab behavior above, the keyboard shortcuts Ctrl+E and Ctrl+Shift+E (CMD+E and CMD+Shift+E on OSX) can be used to change the indentation of only the comments in a selected block of code. This helps in aligning comments at the end of code lines to make the code more readable. The used shortcut can be configured in the preferences .

Selecting blocks of code:

The shortcut Ctrl+M (CMD+M on OSX) can be used to select the block of code that contains caret position (i.e. the surrounding If block, loop or procedure). Repeated usage of the shortcut selects further surrounding code blocks.

The shortcut Ctrl+Shift+M (CMD+Shift+M on OSX) reverses the behavior and reverts the selection to the block that was selected before the last usage of the Ctrl+M shortcut.

The used shortcuts can be configured in the preferences .

Double-clicking on source text:

Double-clicking on a word selects the whole word as usual. However in some cases, double-clicking has a special meaning:

When double-clicking on the name of a procedure call, while holding down the Ctrl Key, the cursor automatically jumps to the declaration of this procedure (the source containing this must currently open or part of the same project).

When double-clicking on an IncludeFile or XincludeFile statement, the IDE will try to open that file. (This is only possible if the included file is written as a literal string, and not through for example a constant.)

In the same way, if you double-click on an IncludeBinary statement, the IDE will try to display that file in the internal file viewer .

Marking of matching Braces and Keywords:

```
46      Select Event
47      Case #PB_Event_Gadget
48          Select EventGadget()
49
50          Case 1
51              SetGadgetState(10,
52
53          EndSelect
54
55      Case #PB_Event_Menu ; We
56          SetGadgetText(10, GetGa
57
58      EndSelect
59
```

When the cursor is on an opening or closing brace the IDE will highlight the other brace that matches it. If a matching brace could not be found (which is a syntax error in PureBasic) the IDE will highlight the current brace in red. This same concept is applied to keywords. If the cursor is on a Keyword such as "If", the IDE will underline this keyword and all keywords that belong to it such as "Else" or "EndIf". If there is a mismatch in the keywords it will be underlined in red. The "Goto matching Keyword" menu entry described below can be used to quickly move between the matching keywords.

The brace and keyword matching can be configured in the Preferences .

Command help in the status bar:

ButtonGadget(#Gadget, x, y, Width, Height, Text\$ [, Flags]) - Create a button gadget in the current GadgetList.

While typing, the IDE will show the needed parameters for any PureBasic function whose parameters you are currently typing. This makes it easy to see any more parameters you still have to add to this function. This also works for procedures, prototypes, interfaces or imported functions in your code as long as they are declared in the same source code or project.

Folding options:

```
319 Procedure.s ZipFileReques
377
378 Procedure.s OpenZipFileRe
379   ProcedureReturn ZipFile
380 EndProcedure
381
```

When special folding keywords are encountered (`Procedure` / `EndProcedure` by default. More can be added), the IDE marks the region between these keywords on the left side next to the line numbers with a `[-]` at the starting point, followed by a vertical line to the end point.

By clicking on the `[-]`, you can hide ("fold") that section of source code to keep a better overview of larger source files. The `[-]` will turn into a `[+]`. By clicking again, the code will again be shown ("unfolded") again.

Note: Even though the state of these folded code lines is remembered when you save/reopen the file, the actual created code file always contains all lines. This only affects the display of the code in the IDE, not the code itself.

Another default fold keyword is `";{"` and `";}"`. Since `";"` marks a comment in PB, these will be totally ignored by the compiler. However, they provide the possibility to place custom fold points that do not correspond to a specific PB keyword.

Auto complete:

```
7
8 prod
9 Procedure
10 ProcedureC
11 ProcedureDLL
12 ProcedureReturn
```

So that you do not have to remember the exact name of every command, there is the Auto complete feature to make things easier.

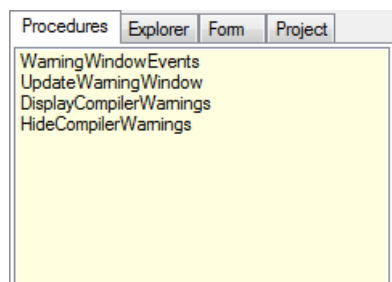
After you have typed the beginning of a command, a list of possible matches to the word start you have just typed will be displayed. A list of options is also displayed when you typed a structured variable or interface followed by a `"\"`.

You can then select one of these words with the up/down keys and insert it at the point you are by pressing the Tab key. You can also continue typing while the list is open. It will select the first match that is still possible after what you typed, and close automatically when either you have just typed an exact match or if there are no more possible matches in the list.

Escape closes the auto complete list at any time. It also closes if you click with the mouse anywhere within the IDE.

Note: You can configure what is displayed in the Auto complete list, as well as turning off the automatic popup (requiring a keyboard shortcut such as Ctrl+Space to open list) in the Preferences. See the Auto complete section of Customizing the IDE for more information.

















Tools Panel on the side:



Many tools to make navigating/editing the source code easier can be added to the Tools Panel on the side of the editor window. For an overview of them and how to configure them, see Built-in Tools .

The Edit Menu:

Following is an explanation of the Items in the Edit menu. Note that many of the Edit menu items are also accessible by right clicking on the source code, which opens a popup menu.

	Undo	Ctrl+Z
	Redo	Ctrl+Y
<hr/>		
	Cut	Ctrl+X
	Copy	Ctrl+C
	Paste	Ctrl+V
<hr/>		
	Insert comments	Ctrl+B
	Remove comments	Ctrl+Shift+B
	Format indentation	Ctrl+I
<hr/>		
	Select All	Ctrl+A
<hr/>		
	Goto...	Ctrl+G
	Goto matching Keyword	Ctrl+K
	Goto recent Line	Ctrl+L
<hr/>		
	Toggle current fold	F4
	Toggle all folds	Ctrl+F4
<hr/>		
	Add/Remove Marker	Ctrl+F2
	Jump to Marker	F2
	Clear Markers	
<hr/>		
	Find/Replace...	Ctrl+F
	Find Next	F3
	Find in Files...	Ctrl+Shift+F

Undo

Undoes the last done action in the code editing area. There is an undo buffer, so several actions can be undone.

Redo

Redo the last action undone by the undo function.

Cut

Copy the selected part of the source code to the clipboard and remove it from the code.

Copy

Copy the selected text to the Clipboard without deleting it from the code.

Paste

Insert the content of the Clipboard at the current position in the code. If any text is selected before this, it will be removed and replaced with the content of the Clipboard.

Insert comments

Inserts a comment (";") before every line of the selected code block. This makes commenting large blocks of code easier than putting the ; before each line manually.

Remove comments

Removes the comment characters at the beginning of each selected line. This reverts the "Insert comments" command, but also works on comments manually set.

Format indentation

Reformats the indentation of the selected lines to align with the code above them and to reflect the keywords that they contain. The rules for the indentation can be specified in the preferences .

Select all Selects the whole source code.

Goto

This lets you jump to a specific line in your source code.

Goto matching Keyword

If the cursor is currently on a keyword such as "If" this menu option jumps directly to the keyword that matches it (in this case "EndIf").

Goto recent line

The IDE keeps track of the lines you view. For example if you switch to a different line with the above Goto function, or with the Procedure Browser tool. With this menu option you can jump back to the previous position. 20 such past cursor positions are remembered.

Note that this only records greater jumps in the code. Not if you just move up/down a few lines with the cursor keys.

Toggle current fold

This opens/closes the fold point in which the cursor is currently located.

Toggle all Folds

This opens/closes all fold points in the current source. Very useful to for example hide all procedures in the code. Or to quickly see the whole code again when some of the code is folded.

Add/Remove Marker

Markers act like Bookmarks in the source code. Their presence is indicated by a little arrow next to the line numbers. You can later jump to these markers with the "Jump to marker" command.

The "Add/Remove Marker" sets or removes a marker from the current line you are editing.

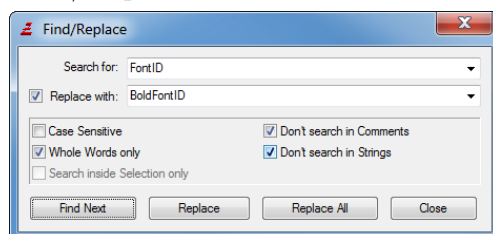
Note: You can also set/remove markers by holding down the Ctrl Key and clicking on the border that holds the markers (not the Line-number part of it).

Jump to Marker

This makes the cursor jump to the next marker position further down the code from the current cursor position. If there is no marker after the cursor position, it jumps to the first one in the source code. So by pressing the "Jump to Marker" shortcut (F2 by default) several times, you can jump to all the markers in the code.

Clear Markers This removes all markers from the current source code.

Find/Replace



The find/replace dialog enables you to search for specific words in your code, and also to replace them with something else.

The "Find Next" button starts the search. The search can be continued after a match is found with the Find Next menu command (F3 by default).

You can make the search more specific by enabling one of the checkboxes:

Case Sensitive : Only text that matches the exact case of the search word will be found.

Whole Words only : Search for the given word as a whole word. Do not display results where the search word is part of another word.

Don't search in Comments : Any match that is found inside a comment is ignored.

Don't search in Strings : Any match that is found inside a literal string (in " ") is ignored.

Search inside Selection only : Searches only the selected region of code. This is really useful only together with the "Replace All" button, in which case it will replace any found match, but only inside the selected region.

By enabling the "Replace with" checkbox, you go into replace mode. "Find Next" will still only search, but with each click on the "Replace" button, the next match of the search word will be replaced by whatever is inside the "Replace with" box.

By clicking on "Replace All", all matches from the current position downwards will be replaced (unless "Search inside Selection only" is set).

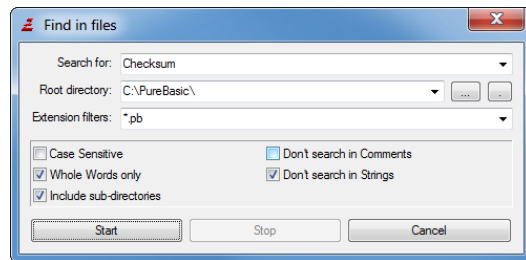
Find Next

This continues the search for the next match of the last search started by the Find/Replace dialog.

Find Previous

This continues the search for the previous match of the last search started by the Find/Replace dialog.

Find in Files



The Find in Files Dialog lets you carry out a search inside many files in a specific directory.

You have to specify a search keyword, as well as a base directory ("root directory") in which to search. You can customize the searched files by specifying extension filters. Any number of filters can be given separated by ",". (*. * or an empty extension field searches all files). As with "Find/Replace", there are checkboxes to make the search more specific.

The "Include sub-directories" checkbox makes it search (recursively) inside any subdirectory of the given root directory too.

When starting the search, a separate window will be opened displaying the search results, giving the file, line number as well as the matched line of each result.

Double-clicking on an entry in the result window opens that file in the IDE and jumps to the selected result line.

Chapter 11

Managing projects

The IDE comes with features to easily handle larger projects. These features are completely optional. Programs can be created and compiled without making use of the project management. However, once a program consists of a number of source code and maybe other related files, it can be simpler to handle them all in one project.

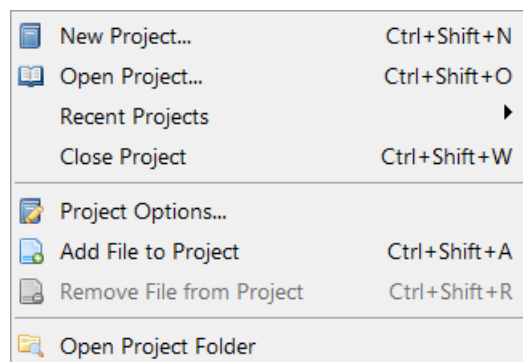
Project management overview

A project allows the management of multiple source codes and other related files in one place with quick access to the files through the project tool . Source files included in a project can be scanned for AutoComplete even if they are not currently open in the IDE. This way functions, constants, variables etc. from the entire project can be used with AutoComplete. The project can also remember the source files that are open when the project is closed and reopen them the next time to continue working exactly where you left off.

Furthermore, a project keeps all the compiler settings in one place (the project file) and even allows to manage multiple "compile targets" per project. A compile target is just a set of compiler options. This way multiple versions of the same program, or multiple smaller programs in one project can be easily compiled at once.

To compile a project from a script or makefile, the IDE provides command-line options to compile a project without opening a user interface. See the section on command-line options for more details. All filenames and paths in a project are stored relative to the project file which allows a project to be easily moved to another location as long as the relative directory structure remains intact.

The Project menu



New Project

Creates a new project. If there is a project open at the time it will be closed. The project options window will be opened where the project filename has to be specified and the project can be configured.

Open Project

Opens an existing project. If there is a project open at the time it will be closed. Previously open source codes of the project will be opened as well, depending on the project configuration.

Recent Projects

This submenu shows a list of recently opened project files. Selecting one of the entries opens this project.

Close Project

Closes the currently open project. The settings will be saved and the currently open source files of the project will be closed, depending on the project configuration.

Project Options

Opens the project options window. See below for more information.

Add File to Project

Adds the currently active source code to the current project. Files belonging to the project are marked with a ">" in the file panel.

Remove File from Project

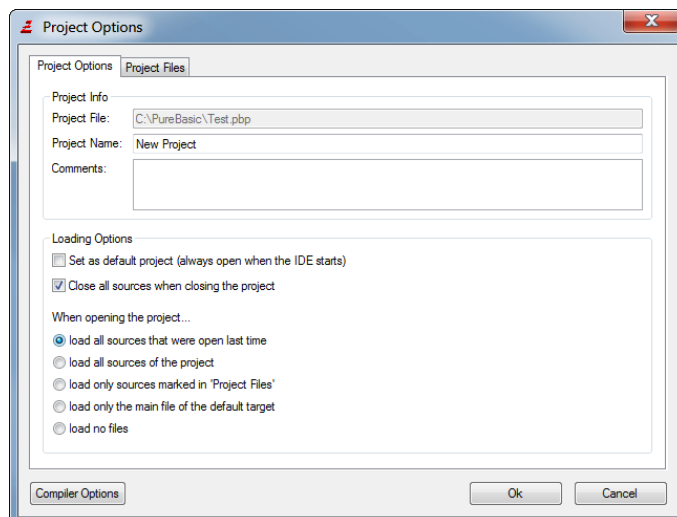
Removes the currently active source from the current project.

Open Project folder

Opens the folder that contains the project file in whatever file manager is available on the system.

The project options window

The project options window is the central configuration for the project. The general project settings as well as the settings for the individual files in the project can be made here.



The following settings can be made on the "Project Options" tab:

Project File

Shows the filename of the project file. This can only be changed during project creation.

Project Name

The name of the project. This name is displayed in the IDE title bar and in the "Recent Projects" menu.

Comments

This field allows to add some comments to the project. They will be displayed in the project info tab.

Set as default project

The default project will be loaded on every start of the IDE. Only one project can be the default project at a time. If there is no default project, the IDE will load the project that was open when the IDE was closed last time if there was one.

Close all sources when closing the project

If enabled, all sources that belong to the project will be closed automatically when the project is closed. When opening the project...

load all sources that where open last time

When the project is opened, all the sources that were open when the project was closed will be opened again.

load all sources of the project

When the project is opened, all (source-)files of the project will be opened.

load only sources marked in 'Project Files'

When the project is opened, only the files that are marked in the 'Project Files' tab will be opened. This way you can start a session always with this set of files open.

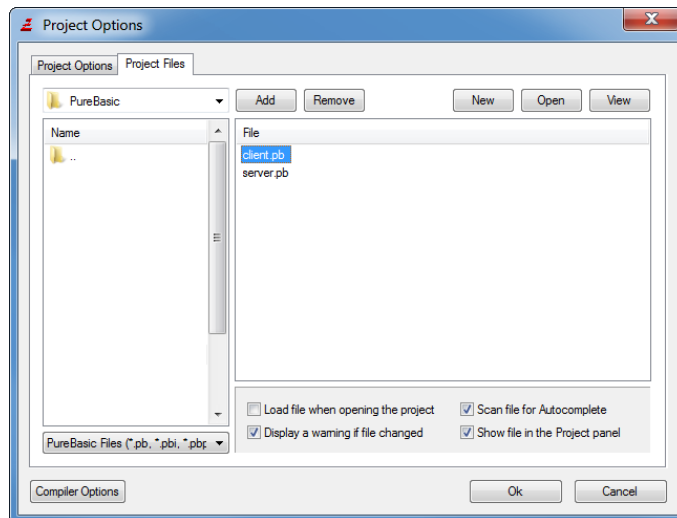
load only the main file of the default target

When the project is opened, the main file of the default target will be opened too.

load no files

No source files are opened when the project is opened.

The "Project Files" tabs shows the list of files in the project on the right and allows changing their settings. The explorer on the left is for the selection of new files to be added.



The buttons on the top have the following function:

Add

Add the selected file(s) in the explorer to the project.

Remove

Remove the selected files in the file list from the project.

New

Shows a file requester to select a filename for a new source file to create. The new file will be created, opened in the IDE and also added to the project.

Open

Shows a file requester to select an existing file to open. The file will be opened in the IDE and added to the project.

View

Opens the selected file(s) in the file list in the IDE or if they are binary files in the FileViewer.

The checkboxes on the bottom specify the options for the files in the project. They can be applied to a single file or to multiple files at once by selecting the files and changing the state of the checkboxes. The settings have the following meaning:

Load file when opening the project

Files with this option will be loaded when the project is open and the "load only sources marked in 'Project Files'" option is specified on the "Project Options" tab.

Display a warning if file changed

When the project is closed, the IDE will calculate a checksum of all files that have this option set and display a warning if the file has been modified when the project is opened the next time. This allows to be notified when a file that is shared between multiple projects has been edited while working on another project. This option should be disabled for large data files to speed up project loading and saving, or for files which are changed frequently to avoid getting a warning every time the project is opened.

Scan file for AutoComplete

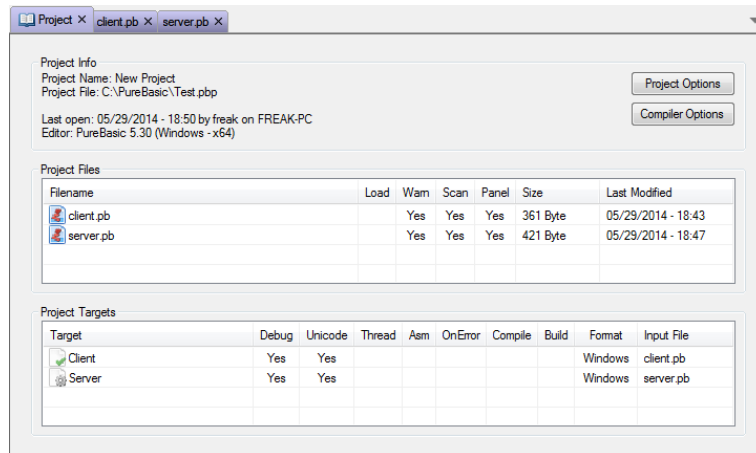
Files with this option will be scanned for AutoComplete data even when they are not currently loaded in the IDE. This option is on by default for all non-binary files. It should be turned off for all files that do not contain source code as well as for any files where you do not want the items to turn up in the AutoComplete list.

Show file in Project panel

Files with this option will be displayed in the project side-panel. If the project has many files it may make sense to hide some of them from the panel to have a better overview and faster access to the important files in the project.

The project overview

When a project is open, the first tab of the file panel shows an overview of the project and its files.

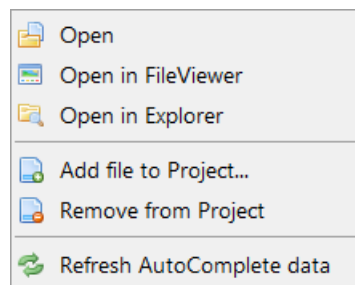


Project Info

This section shows some general info about the project, such as the project filename, its comments or when and where the project was last opened.

Project Files

This section shows all files in the project and their settings from the Project Options window. Double-clicking on one of the files opens the file in the IDE. Right-clicking displays a context menu with further options:



Open - Open the file in the IDE.

Open in FileViewer - Open the file in the FileViewer of the IDE.

Open in Explorer - Open the file in the operating systems file manager.

Add File to Project - Add a new file to the project.

Remove File from Project - Remove the selected file(s) from the project.

Refresh AutoComplete data - Rescan the file for AutoComplete items.

Project Targets

This section shows all compile targets in the project and some of their settings. Double-clicking on one of the targets opens this target in the compiler options. Right-clicking on one of the targets displays a context menu with further options:

Edit target - Open the target in the compiler options.

Set as default target - Set this target as the default target.

Enable in 'Build all Targets' - Include this target in the 'Build all Targets' compiler menu option.

The project panel

There is a sidepanel tool which allows quick access to the files belonging to the project. For more information see the built-in tools section.

Chapter 12

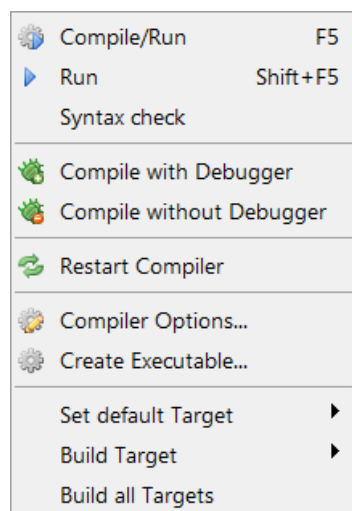
Compiling your programs

Compiling is simple. Just select "Compile/Run" (F5 by default) and your program will be compiled and executed for a testing run.

To customize the compiling process, you can open the "Compiler options" dialog. The settings made there are associated with the current source file or the current project, and also remembered when they are closed. The place where this information is saved can be configured. By default, it is saved at the end of the source code as a comment (invisible in the IDE).

In case of an error that prevents the compiler from completing the compilation, it aborts and displays an error-message. This message is also logged in the error log, and the line that caused the error is marked. A number of functions from older versions of PureBasic that have been removed from the package still exist for a while as a compatibility wrapper to allow older codes to be tested/ported more easily. If such a function is used in the code, the compiler will issue a warning. A window will be opened displaying all warnings issued during compilation. Double-clicking on a warning will display the file/line that caused the warning. Note that such compatibility wrappers will not remain indefinitely but will be removed in a future update, so it is recommended to fix issues that cause a compiler warning instead of relying on such deprecated functions.

The compiler menu



Compile/Run

This compiles the current source code with the compiler options set for it and executes it. The executable file is stored in a temporary location, but it will be executed with the current path set to the directory of the source code; so loading a file from the same directory as the source code will work.

The source code need not be saved for this (but any included files must be saved).

The "Compile/Run" option respects the debugger setting (on or off) from the compiler options or

debugger menu (they are the same).

Run

This executes the last compiled source code once again. Whether or not the debugger is enabled depends on the setting of the last compilation.

Compile with Debugger

This is the same as "Compile/Run" except that it ignores the debugger setting and enabled the debugger for this compilation. This is useful when you usually have the debugger off, but want to have it on for just this one compilation.

Compile without Debugger

Same as "Compile with Debugger" except that it forces the debugger to be off for this compilation.

Restart Compiler (not present on all OS)

This causes the compiler to restart. It also causes the compiler to reload all the libraries and resident files, and with that, the list of known PureBasic functions, Structures, Interfaces and Constants is updated too. This function is useful when you have added a new User Library to the PB directory, but do not want to restart the whole IDE. It is especially useful for library developers to test their library.

Compiler Options

This opens the compiler options dialog, that lets you set the options for the compilation of this source file.

Create executable

This opens a save dialog, asking for the executable name to create. If the executable format is set to DLL, it will create a DLL on Windows, shared object on Linux and dylib on OS X. When creating an executable on OS X, appending '.app' at the executable name will create a bundled executable with the necessary directory structure, including the icon. If no '.app' is set, then it will create a regular console-like executable.

Set default Target

When a project is open, this submenu shows all compile targets and allows to quickly switch the current default target. The default target is the one which is compiled/executed with the "Compile/Run" menu entry.

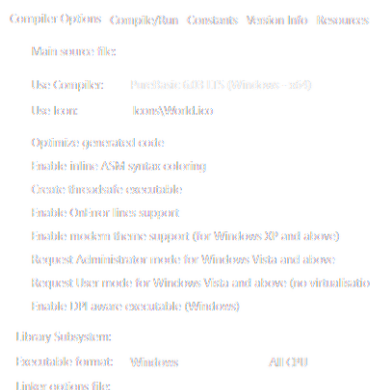
Build Target

When a project is open, this submenu shows all compile targets and allows to directly compile one of them.

Build all Targets

When a project is open, this menu entry compiles all targets that have this option enabled in the compiler options. A window is opened to show the build progress.

Compiler options for non-project files



Main source file

By enabling this option, you can define another file that will be the one sent to the compiler instead of this one. The use of this is that when you are editing a file that does not run by itself, but is included into another file, you can tell the compiler to use that other file to start the compilation.

Note: When using this option, you MUST save your source before compiling, as only files that are written to disk will be used in this case. Most of the compiler settings will be taken from the main source file, so when setting this, they are disabled. Only some settings like the debugger setting will be

used from the current source.

Use Compiler

This option allows the selection of a different compiler to use instead of the compiler of the current PureBasic version. This makes it easy to compile different versions of the same program (x86 and x64) without having to start up the IDE for the other compiler just for the compilation. Additional compilers for this option have to be configured in the preferences .

If the compiler version matches that of the default compiler but the target processor is different then the built-in debugger of the IDE can still be used to debug the compiled executable. This means that an executable compiled with the x86 compiler can be debugged using the x64 IDE and vice versa. If the version does not match then the standalone debugger that comes with the selected compiler will be used for debugging to avoid version conflicts.

Use Icon (Windows and MacOS X only)

Here you can set an icon that will be displayed when viewing the created executable in the explorer. It is also displayed in the title bar of your programs windows and the Taskbar.

Windows: The icon must be in ICO format (Windows Icon).

MacOS X: The icon must be in ICNS format (Macintosh Icon). To create such an icon, you should create PNG files in the dimensions 128x128, 48x48, 32x32 and 16x16 of your image and then use the tool "Icon Composer" that comes with the OSX developer tools to create the ICNS file. It should be located in /Developer/Applications/Utilities/. To be displayed once the application has just been created, the Finder may need to be restarted.

Optimize generated code

This enables the C code optimizer. It simply sets gcc's optimization flags on -O2 when enabled, on -O0 otherwise. [See here.](#)

Enable inline ASM syntax coloring

This enables the inline ASM syntax coloring. See the Inline x86 ASM section of the help-file for more information on this option.

Create thread-safe executable

This tells the compiler to use a special version of certain commands to make them safe to be used in threads. See the Thread library for more information.

This also enables the Debugger to display correct information if threads are used. Without this option, the debugger might output wrong line numbers when threads are involved for example.

Enable modern theme support (Windows only)

Adds support for skinned windows on Windows Vista, Windows 7 or Windows 8.

Request Administrator mode for Windows Vista and above (Windows only)

The created executable will always be started with administrator rights on Windows Vista and above (it will not launch if the administrator password is not entered). This option should be set for programs that need to access restricted folders or restricted areas of the registry to get full access.

If this option is turned on, the standalone debugger will automatically selected when debugging, so the program can be tested in administrator mode.

Note: This option has no effect when the program is run on other versions of Windows.

Request User mode for Windows Vista and above (Windows only)

This option disables the "Virtualization" feature for this executable on Windows Vista and above.

Virtualization caused file and registry access to be redirected to a special user folder if the user does not have the needed rights to do the operation (this is done for compatibility with older programs).

Note that this redirection is done without notifying the user; this can lead to some confusion if he tries to find saved files on the file-system. Because of this, it is recommended to disable this feature if the program complies with the Windows Vista file/registry access rules.

Note: This option has no effect when the program is run on other versions of Windows. It cannot be combined with the "Administrator mode" option above.

Enable DPI Aware Executable (Windows only)

This option enable DPI awareness when creating an executable. That means than GUI created in PureBasic will scale automatically if the DPI of the screen is above 100%. Most of the process is seamless, but some case needs to be worked out, like pixel based gadgets (ImageGadget, CanvasGadget etc.).

Enable DLL Protection (Windows only)

Enable DLL preloading protection to the executable. It prevents that system DLLs are first searched for in the program directory instead of in the System32 directory of the Windows operating system.

Enable OnError lines support (Windows only)

Includes line numbers information with the executable for the OnError-Library .

Library Subsystem

Here you can select different subsystems for compilation. More than one subsystem can be specified, separated with a comma. For more information, see subsystems .

Executable format

This allows you to specify the created executable format:

Windows : a normal windows executable.

Console : an executable with a default console. This one still can create windows and such, but it always has a console open. When executed from a command prompt, this executable type uses the command terminal as its console and writes there, whereas the "Windows" executable would create a separate Console window when using OpenConsole() . This setting must be used to create a Console application that can have its input/output redirected with pipes.

Shared DLL : create a windows DLL. See Building a DLL for more info.

Note: When you do "Compile/Run" with a DLL source code, it is executed as a normal executable. A dll is only created when you use "create executable".

Cpu Optimisation (next to Executable format)

This setting allows to include Cpu optimised PB functions in your executable:

All CPU : The generic functions are included that run on all CPUs.

Dynamic CPU : The generic functions as well as any available CPU specific function are included. The function to execute is decided at runtime. This creates a bigger executable, but it will run as fast as possible on all CPUs.

All other options : Include only the functions for a specific CPU. The executable will not run on any Cpu that does not support this feature.

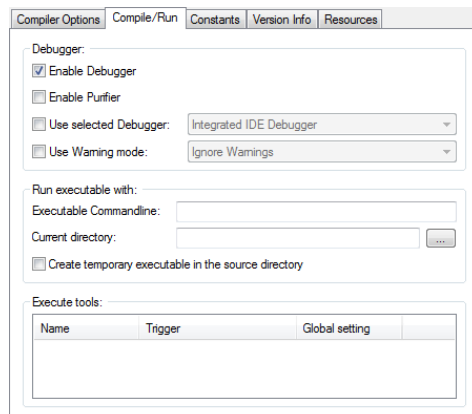
Note: No PB functions actually support this feature for now (it is ignored for them). However, some User Libraries include such optimisations.

Linker options file

A textfile can be specified here with further command-line options that should be passed to the linker when creating the executable. The file should contain one option per line.

Compile/Run

This section contains options that affect how the executable is run from the IDE for testing. Except for the tools option, they have no effect when the "Create executable" menu is used.



Enable Debugger

This sets the debugger state (on/off) for this source code, or if the main file option is used, for that file too. This can also be set from the debugger menu.

Enable Purifier

This enables purifier support for the debugger. The purifier can detect a certain type of programming errors such as writing past the end of an allocated memory buffer. See Included debugging tools for more details.

Use selected Debugger

This allows to choose a different debugger type for this file only. If this option is disabled, the default debugger is used; this can be specified in the preferences .

Use Warning mode

This allows to choose a different warning mode for this file only. If this option is disabled, the default setting is used which can be specified in the preferences . The available options are:

Ignore Warnings: Warnings will be ignored without displaying anything.

Display Warnings: Warnings will be displayed in the error log and the source code line will be marked, but the program continues to run.

Treat Warnings as Errors: A warning will be treated like an error.

Executable command-line

The string given here will be passed as the command-line to the program when running it from the IDE. The content of this string can be got with `ProgramParameter()` .

Current directory

The directory specified here will be set as the current directory for the program when running it from the IDE.

Create temporary executable in the source directory

With this option turned on, the temporary executable file for running the program from the IDE will be placed inside the source directory. This can be useful if the program depends on files inside the source directory for testing. With this option turned off, the executable is created in the systems temporary directory.

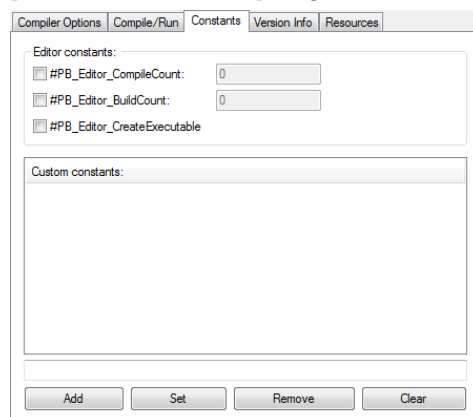
Execute tools

Here external tools can be enabled on a per-source basis. The "Global settings" column shows if the tool is enabled or disabled in the tools configuration . A tool will only be executed for the source if it is both enabled globally and for this source.

Note: For a tool to be listed here, it must have the "Enable Tool on a per-source basis" option checked in the tools configuration and be executed by a trigger that is associated with a source file (i.e. not executed by menu or by editor startup for example).

Constants

In this section, a set of special editor constants as well as custom constants can be defined which will be predefined when compiling this source.



#PB_Editor_CompileCount

If enabled, this constant holds the number of times that the code was compiled (both with "Compile/Run" and "Create Executable") from the IDE. The counter can be manually edited in the input field.

#PB_Editor_BuildCount

If enabled, this constant holds the number of times that the code was compiled with "Create Executable" only. The counter can be manually edited in the input field.

#PB_Editor_CreateExecutable

If enabled, this constant holds a value of 1 if the code is compiled with the "Create Executable" menu or 0 if "Compile/Run" was used.

Custom constants

Here, custom constants can be defined and then easily switched on/off through checkboxes. Constant definitions should be added as they would be written within the source code. This provides a way to enable/disable certain features in a program by defining a constant here and then checking in the source

for it to enable/disable the features with `CompilerIf/CompilerEndIf` .

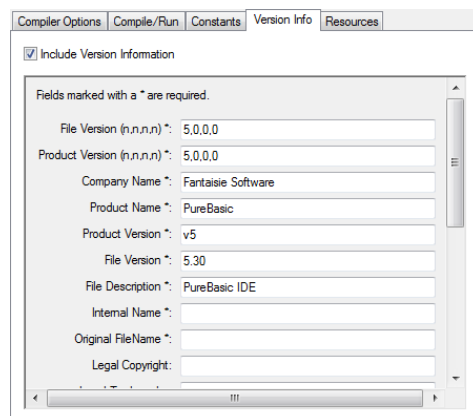
Inside the definition of these constants, environment variables can be used by specifying them in a "bash" like style with a "\$" in front. The environment variable will be replaced in the constant definition before compiling the source. This allows to pass certain options of the system that the code is compiled on to the program in the form of constants.

Example: `#Creator=$USERNAME`

Here, the `$USERNAME` will be replaced by the username of the logged in user on Windows systems. If an environment variable does not exist, it will be replaced by an empty string.

Note: To test within the source code if a constant is defined or not, the `Defined()` compiler function can be used.

Version Information



This is a Windows only feature. By enabling this, a resource is included in the executable with information about your program. It can be viewed by right-clicking on the executable in the windows explorer and selecting "Properties". Also it can be read by other programs such as setup tools. Fields marked with a * are required if you want to include the version info (if not all required fields are set, the information may not display correctly on some versions of Windows).

The first two fields MUST be composed of 4 numbers separated by commas. All other fields may be any string. In the 3 empty boxes, you can define your own fields to include in the Version info block.

In all the string fields, you may include special tokens that are replaced when compiling:

`%OS` : replaced with the version of Windows used to compile the program

`%SOURCE` : replaced with the filename (no path) of the source file.

`%EXECUTABLE` : replaced with the name of the created executable (this only works when "create executable" is used, not with "Compile/Run").

`%COMPILECOUNT` : replaced with the value of the `#PB_Editor_CompileCount` constant.

`%BUILDCOUNT` : replaced with the value of the `#PB_Editor_BuildCount` constant.

Furthermore, you can use any token listed with the `FormatDate()` command. These tokens will be replaced with their respective meaning in `FormatDate()` used with the date of the compilation (i.e. `%yy` gives the year of the compilation)

Meaning of the lower 3 fields:

File OS

Specifies the OS that this Program is compiled for (Using `VOS_DOS` or `VOS_WINDOWS16` makes little sense. They are only included for the sake of completeness).

File Type

Type of the executable (Here `VFT_UNKNOWN`, `VFT_APP` or `VFT_DLL` are the only ones that really make sense for PureBasic programs).

Language

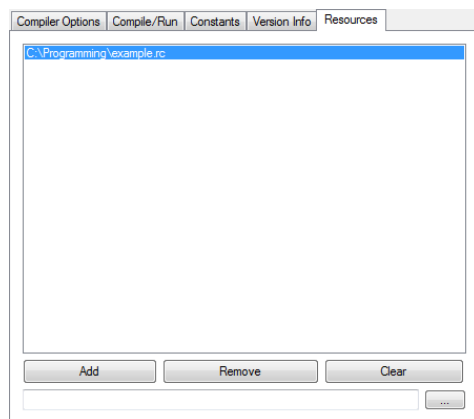
Specifies the language in which this version info is written.

The fields values can be accessed when compiling the program from the IDE using the following constants (same order):

```
#PB_Editor_FileVersionNumeric
#PB_Editor_ProductVersionNumeric
#PB_Editor_CompanyName
```

```
#PB_Editor_ProductName
#PB_Editor_ProductVersion
#PB_Editor_FileVersion
#PB_Editor_FileDescription
#PB_Editor_InternalName
#PB_Editor_OriginalFilename
#PB_Editor_LegalCopyright
#PB_Editor_LegalTrademarks
#PB_Editor_PrivateBuild
#PB_Editor_SpecialBuild
#PB_Editor_Email
#PB_Editor_Website
```

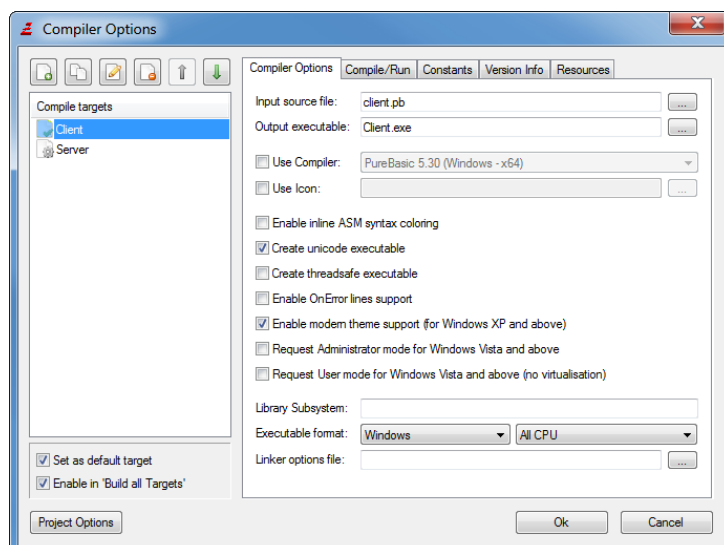
Resources



This is a Windows only feature. Here you can include as many Resource scripts (*.rc files) as you want. They will be compiled and included with the executable. You can use any resource editor (for example the [PellesC IDE](#)) to create such scripts.

Note: Since Resources are a specific to the Windows platform only, PB does not include a Library to manage them and they are not further documented here. See documentation on the [Windows API](#) and resources for more information.

Compiler options for projects



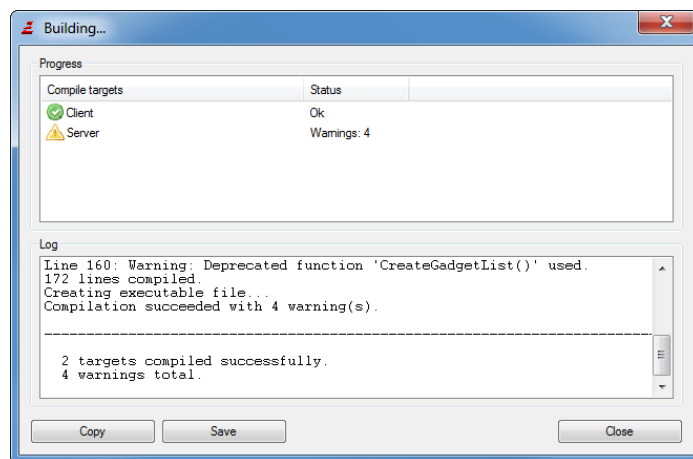
The compiler options for projects allow the definition of multiple compile targets. Each target is

basically a set of compiler options with a designated source file and output executable. The left side of the compiler options window is extended with the list of the defined compile targets. The toolbar on top of it allows to create, delete, copy, edit or move targets in the list.

The default target is the one which will be compiled when the "Compile/Run" menu entry is selected. It can be quickly switched with the "Set as default target" checkbox or from the compiler menu. The "Enable in 'Build all Targets'" option specifies whether or not the selected target will be built when the 'Build all Targets' menu entry is used.

The right side of the compiler options is almost the same as in the non-project mode and reflects the settings for the compile target that is currently selected on the left. The only difference are the "Input source file" and "Output executable" fields on the first tab. These fields have to be specified for all compile targets. Other than that, the compiler options are identical to the options described above. In project mode, the information about the compile target is stored in the project file and not in the individual source files. Information that belongs to the file (such as the folding state) are still saved for the individual source files in the location specified by the Preferences .

The Build progress window



When the 'Build all Targets' menu entry is selected on an open project, all targets that have the corresponding option set in the compiler options will be compiled in the order they are defined in the compiler options. The progress window shows the current compile progress as well as the status of each target. When the process is finished, the build log can be copied to the clipboard or saved to disk.

Chapter 13

Using the debugger

PureBasic provides a powerful debugger that helps you find mistakes and bugs in your source code. It lets you control the program execution, watch your variables , arrays or lists or display debug output of your programs. It also provides advanced features for assembly programmer to examine and modify the CPU registers or view the program stack, or the Memory of your program. It also provides the possibility to debug a program remotely over the network.

To enable the debugger for your program, you can select "Use Debugger" from the debugger menu, or set it in your programs Compiler options. By using the "Compile with Debugger" command from the Compiler menu, you can enable the debugger for just one compilation.

You can directly use debugger commands in your source, such as [CallDebugger](#), [Debug](#), [DebugLevel](#), [DisableDebugger](#) and [EnableDebugger](#).

The PureBasic debugger comes in 3 forms:

A Debugger integrated directly with the IDE, for an easy to use, quick way to debug your programs directly from the programming environment. This debugger also provides the most features.

A separate, standalone debugger, that is useful for some special purposes (for example, when the same program must be executed and debugged several times at once) or to be used with third party code Editors. It provides most of the features of the integrated IDE debugger, but because it is separate from the IDE, some of the efficiency of the direct access from the IDE is lost. The standalone debugger can be used to debug programs remotely through a network connection.

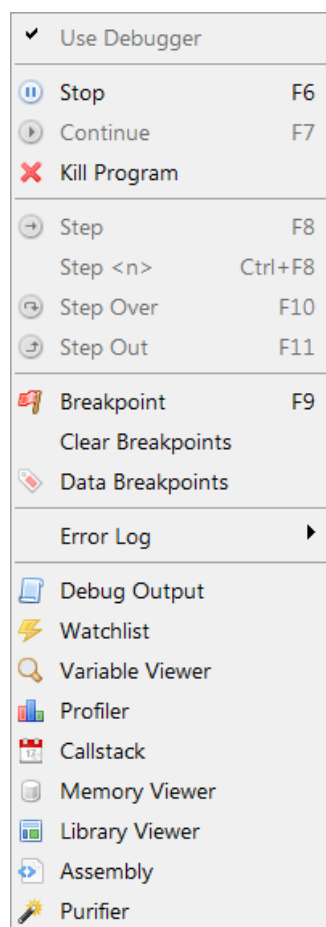
A console only debugger. This debuggers primary use is for testing non-graphical environment like on Linux systems without an X server, or to remotely develop through ssh.

The type of debugger that is used can be selected in the preferences .

All this debugging functionality however comes at a price. Running a program in debug mode is significantly slower in its execution that running it without the debugger. This should be no problem however, since this is for testing only anyway.

If you need to use the debugger, but have some parts in you program that require the full execution speed, you can disable the debugger in just that section with the [DisableDebugger](#) / [EnableDebugger](#) keywords.

The Debugger integrated into the IDE



You can access all the debugger features while the program is running from the debugger menu, or the corresponding toolbar buttons or shortcuts.

While you are debugging your program, all the source files that belong to that program (also included files) will be locked to read-only until the program has finished. This helps to ensure that the code that is marked as the currently executed one is has not actually been modified already without a recompilation. Note that a program can be run only one time at once in IDE debugger mode. If you try to executed it again, you are given the option to execute it with the standalone Debugger.

Tip:

The debugger menu is also added to the system-menu of the Main IDE window (the menu you get when clicking the PB icon in the left/top of the window). This allows you to access the debugger menu also from the Taskbar, by right-clicking on the Taskbar-Icon of the IDE.

Program Control

There are functions for basic control of the running program. You can halt the execution to examine variables and the code position or let the code execute line by line to follow the program flow. While the program is halted, the line that is currently being executed is marked in your source code (with very light-blue background color in the default colors).

The state of the program can be viewed in the IDE status bar, and in the Error log area.

Menu commands for program control:

Stop

Halts the Program and displays the current line.

Continue

Continues the program execution until another stop condition is met.

Kill Program

This forces the program to end, and closes all associated debugger windows.

Step

This executes one line of source code and then stops the execution again.

Step <n>

This will execute a number of steps that you can specify and then stop the execution again.

Step Over

This will execute the current line in the source and then stop again, just like the normal 'Step'. The difference is that if the current line contains calls to procedures, the execution will not stop inside these procedures like it does with the normal 'Step', but it will execute the whole procedure and stop after it returned. This allows to quickly skip procedures in step mode.

Step Out

This will execute the remaining code inside the current procedure and stop again after the procedure has returned. If the current line is not in any procedure, a normal 'Step' will be done.

Line Breakpoints

Breakpoints are another way to control the execution of your program. With the Breakpoint menu command, you mark the currently selected line as a breakpoint (or remove any breakpoint that exists in that line).

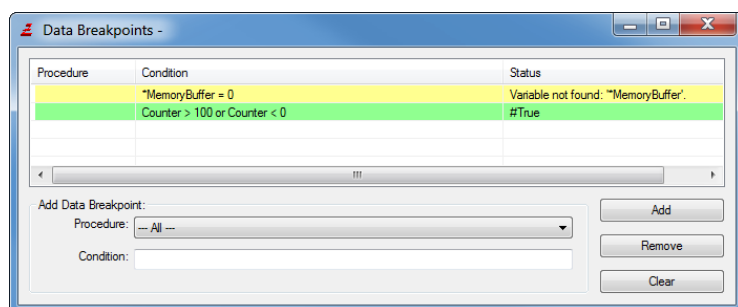
When the execution of the code reaches that line, it will stop at this point. Note that if you select a non-executable line (such as an empty line or a Structure definition), it will halt the execution on the next executable line after that.

After the execution of your program has stopped at a breakpoint, you can use any of the Program control commands to continue/end the execution.

Breakpoints can be set and removed dynamically, while your program is running, or while you are editing your source code. With the "Clear Breakpoints" command, you can clear all breakpoints in a source file.

Note: You can also set/remove Breakpoints by holding down the Alt Key and clicking on the border that contains the Breakpoint marks.

Data Breakpoints



In addition to the line specific breakpoints, the debugger also provides data breakpoints. Data breakpoints halt the program if a given condition is met. This way it is easy to find out when a variable or other value in the program changes and halt the program if that happens. The condition can be any PureBasic expression that can be evaluated to true or false. This can be anything that could be put after an **If** keyword, including logical operators such as **And**, **Or** or **Not**. Most functions of the Math, Memory and String libraries as well as all object validation functions in the form **IsXXX()** and the **XxxID** functions for getting the OS identifiers for an object are also available.

Example conditions:

```
1 MyVariable$ <> "Hello" Or Counter < 0 ; halt if MyVariable$ changes
   from "Hello" or if the Counter falls below zero
2 PeekL(*SomeAddress+500) <> 0 ; halt if the long value at
   the given memory location is not equal to zero
```

Data breakpoints can be added using the Data Breakpoint option from the Debugger menu. They can be limited to a specific procedure or they can be added for all code. The special "Main" entry of the

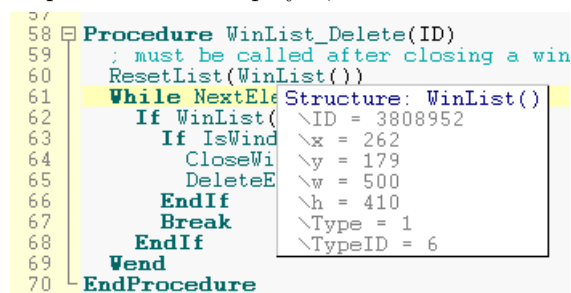
procedure selection specifies that the data breakpoint should only be checked when the execution is not in any procedure.

The status column shows the status of all breakpoint conditions on their last evaluation. This can be true, false or an error if the condition is not a valid expression. Once a condition is evaluated to true, the program execution will be halted. This condition is automatically removed from the list as soon as the program continues, so that it does not halt the program again immediately.

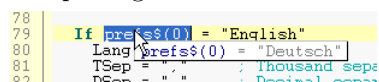
Note: Checking for data breakpoints slows down the program execution because the breakpoint conditions have to be re-evaluated for every executed line of code to check if the condition is met. So data breakpoints should only be added when needed to keep the program execution fast otherwise. Limiting a data breakpoint to a certain procedure also increases the speed because the check then only affects the given procedure and not the entire program.

Examining variables during runtime

The value of a variable can be very quickly viewed while the program is running by placing the mouse cursor over a variable in the source code and waiting for a brief moment. If the variable is currently in scope and can be displayed, its value will be shown as a tool-tip on the mouse location.



More complex expressions (for example array fields) can be viewed by selecting them with the mouse and placing the mouse cursor over the selection.



The debugger tools also offer a number of ways to examine the content of variables , arrays or lists .

Errors in the Program

If the debugger encounters an error in your program, it will halt the execution, mark the line that contains the error (red background in the default colors) and display the error-message in the error log and the status bar.

At this point, you can still examine the variables of your program, the callstack or the memory, however other features like the Register display or stack trace are not available after an error.

If the error is determined to be fatal (like an invalid memory access, or division by 0), you are not allowed to continue the execution from this point. If the error was reported by a PureBasic library, you are allowed to try to continue, but in many cases, this may lead to further errors, as simply continuing just ignores the displayed error.

After an error (even fatal ones), you have to use the "Kill Program" command to end the program and continue editing the source code. The reason why the program is not automatically ended is that this would not allow to use the other debugger features (like variable display) to find the cause of the error. Note: you can configure the debugger to automatically kill the program on any error. See Customizing the IDE for that.

Debugger warnings

In some cases the debugger cannot be sure whether a given parameter is actually an error in the program or was specified like that on purpose. In such a case, the debugger issues a warning. By default, a warning will be displayed with file and line number in the error log and the line will be marked (orange in the default colors). This way the warnings do not go unnoticed, but they do not interrupt the program flow. There is also the option of either ignoring all warnings or treating all warnings like errors (stopping the program). The handling of debugger warnings can be customized globally in the Preferences or for the current compiled program in the Compiler options .

The Error log

The error log is used to keep track of the compiler errors, as well as the messages from the debugging. Messages are always logged for the file they concern, so when an error happens in an included file , this file will be displayed, and a message logged for it.

The "Error log" submenu of the Debugger menu provides functions for that:

Show error log

Shows / hides the log for the current source.

Clear log

Clears the log for this file.

Copy log

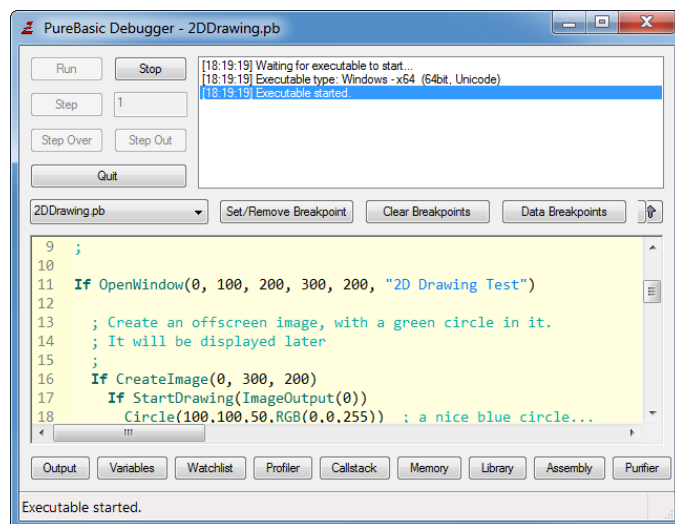
Copies the contents of the error log to the clipboard.

Clear Error marks

After you have killed the program, any error mark in the source file will remain. This is to help you identify the line that caused the problem and solve it. The "Clear error Marks" command can be used to remove these marks.

You can also configure the IDE to automatically clean the error marks when the program ends. See Configuring the IDE for that.

The Standalone Debugger



The standalone debugger is very similar to the one in the IDE, and will be explained here only briefly: On the Debugger window, you have control buttons to carry out the basic program control, as described above. The "Step" button carries out as many steps as are set in the edit field next to it. Closing the Debugger with "Quit" or the close button will also kill the debugged program.

The Error log area can be hidden by the up arrow button on the right side in order to make the debugger window smaller.

The code view is used to display the currently executed code line as well as any errors or breakpoints. Use the combo box above it to select the included file to view. The "Set Breakpoint", "Remove

Breakpoint” and ”Clear Breakpoints” can be used to manage breakpoints in the currently displayed source file. The code view also provides the mouse-over feature from the integrated debugger to quickly view the content of a variable.

The debugger tools can be accessed from the buttons below the code area. Their usage is the same as with the integrated IDE debugger.

Note: The Standalone Debugger has no configuration of its own. It will use the debugger settings and coloring options from the IDE. So if you use a third-party Editor and the standalone debugger, you should run the IDE at least once to customize the Debugger settings.

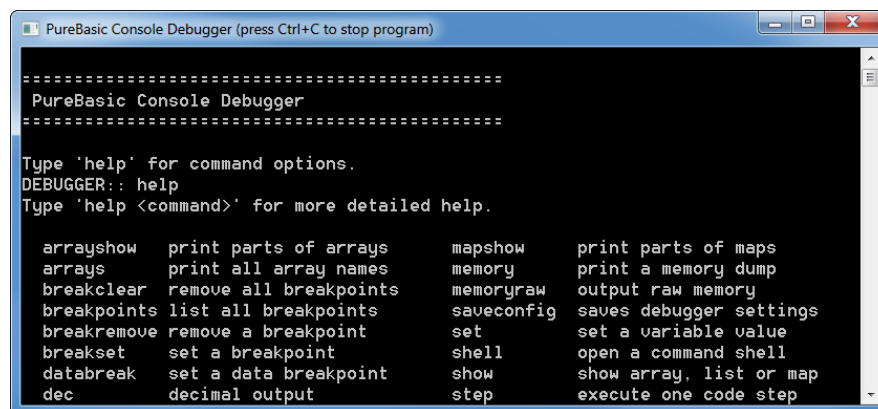
Executing the standalone debugger from the command-line:

To execute a program compiled on the command-line with enabled debugger (-d or /DEBUGGER switch), call the debugger like this:

```
pbdebugger <executable file> <executable command-line>
```

If you execute a debugger-enabled executable from the command-line directly, it will only use the command-line debugger.

The command-line debugger:



The command-line debugger is not a part of the IDE and therefore not explained in detail here.

While the program is running, hit Ctrl+C in the console to open a debugger console prompt. In this prompt type ”help” to get an overview of all available commands. Type ”help <commandname>” for a more detailed description of the command.

Debugging threaded programs:

To use the debugger with a program that creates threads , the ’Create thread-safe executable’ Option must be set in the Compiler options , as otherwise the information displayed by the debugger concerning line numbers, errors, local variables and such could be wrong due to the multiple threads.

The following features and limitations should be considered when debugging a threaded program:

While the program is running, the variable viewer, callstack display or assembly debugger will display information on the main thread only. When the program is stopped, they display information on the thread they were stopped in. So to examine local variables or the callstack of a thread, the execution must be halted within that thread (by putting a breakpoint or a [CallDebugger](#) statement there). The various ’Step’ options always apply to the thread where the execution was last stopped in.

If an error occurs, the execution is halted within that thread, so any information displayed by the variable viewer or callstack display is of the thread that caused the error.

The watchlist only watches local variables of the main thread, not those of any additional running threads.

While the execution is stopped within one thread, the execution of all other threads is suspended as well.

Chapter 14

Included debugging tools

These tools provide many features to inspect your program while it is running. They can not be used while you are editing the source code. These tools are available in both the integrated Debugger and the standalone debugger. The console debugger provides many of these features too, but through a debugger console.

Some of the tools include the viewing of variables. Here is an explanation of the common fields of all these variable displays:

Scope

The scope of a variable is the area in which it is valid. It can be global , local , shared , static or threaded , depending on how it is used in your source code. 'byref' ("by reference", i.e. using the address) is used to indicate an Array or List that was passed as parameter to a procedure.

Variable type

The variable type is indicated through a colored icon:

B : Byte

A : Ascii

C : Character

W : Word

U : Unicode

L : Long

I : Integer

Q : Quad

F : Float

D : Double

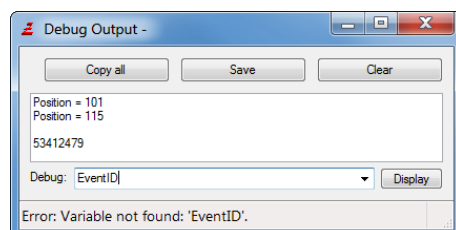
S : String

Sn : Fixed length string

A Structure is either marked as a dot, or with an arrow. If marked with an arrow, it can be expanded by double-clicking on it to view the members of this structure. A down arrow marks an expanded structure. A Structure marked with a dot cannot be expanded (usually because it is just a structure pointer).

Dynamic arrays inside structures are shown with the dimensions they are currently allocated with. Lists and maps inside structures are shown with their size and their current element (if any).

The Debug output window



In this window, the output of the [Debug](#) statement will be displayed. The Debug statement is a quick

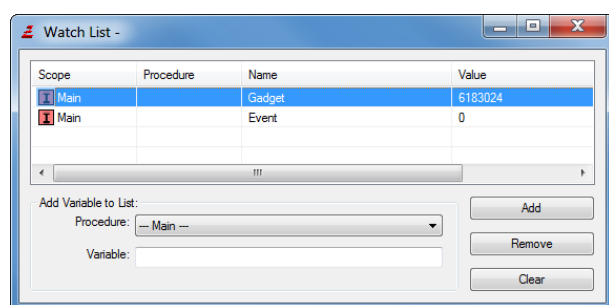
and simply way to print messages for debugging purposes.

The debug window will automatically open at the first output is produced by your program. If you then close it, it will not automatically open on subsequent messages, however they will still be logged. You can copy this output to the clipboard or save it to a file. There is also a button to clear the messages from the window.

The entry field at the bottom of the window allows an expression to be entered, which will be evaluated and the result printed in the output box. This allows to quickly check the state of variables or array fields without the need to look them up in one of the debugger tools. The evaluation is started by pressing Enter or clicking on the "Display" button. If the expression cannot be evaluated for some reason, an error-message is displayed in the statusbar.

The expression can be any valid PB expression (not including logical ones or containing PB keywords). It can contain variables , arrays , lists , constants and also some commands from the Math , Memory and String libraries.

The Watchlist



The watchlist can be used to track changes in variables of your program in real time, while the program is running. It can only display single variables (no full structures), however, these variables can be a part of structures. Elements of dynamic array, list or map inside structures can not be displayed in the watchlist.

To add a variable, select its procedure (if it is a local variable) or select "— Main —" if it is a global variable or part of an array or list . Then type the variable name, as you would access it in your source code into the Variable field and press Add.

Examples :

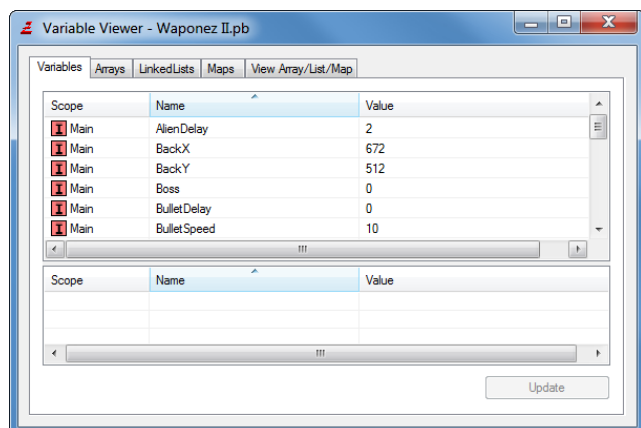
<code>MyVariable\$</code>	- add a normal string variable
<code>Array(1, 5)</code>	- add an array field
<code>Structure\subfield[5]\value</code>	- add a variable inside a structure
<code>MyList()\structuresubfield</code>	- add a variable inside a structured list

You can also add new watched variables from the VariableViewer, by right-clicking on them and selecting "add to watchlist"

In the list you will see the values of the watched variables. If the value is displayed as "—", it means that this variable is not valid at the current point in the source code; for example, this will occur if you watch a local variable, or a list element and the list has no current element.

The watched variables are remembered between the debugging sessions, and even saved with the compiler options, so you do not need to repopulate this list all the time.

The Variable Viewer



The Variable viewer allows to examine the program's variables, arrays, lists and maps. The individual tabs show global and threaded items in the top part and local, shared and static items in the bottom part.

The "Update" Button can be used to get the most recent data from the program. If the program is halted or in step mode, the content is updated on each step automatically. By right-clicking on any variable or Array/List field, you can copy that variable, or add it to the watchlist for real-time tracking of its value. On Windows, the content of the Variable viewer can be sorted by name, scope or variable value by clicking on the header of the appropriate column.

The 'Variables' tab

This tab, shows the variables of the program. By right-clicking on a variable, it is possible to add it to the watchlist.

The 'Arrays' tab

This tab shows a list of all arrays in the program and the dimensions in which they are currently defined (-1 means that Dim was not called yet). By right-clicking on an array, the content of the array can be viewed in the "View Array/List/Map" tab.

The 'Lists' tab

This tab shows a list of all Lists, the number of elements they currently hold ("-" indicates that NewList was not called yet), as well as the index of the current element of the list ("-" indicates that there is no current element). By right-clicking on a list, the content of the list can be viewed in the "View Array/List/Map" tab.

The 'Maps' tab

This tab shows a list of all maps, the number of elements they currently hold ("-" indicates that NewMap was not called yet), as well as the key of the current element of the map ("-" indicates that there is no current element). By right-clicking on a map, the content of the map can be viewed in the "View Array/List/Map" tab.

The 'View Array/List/Map' tab

This tab can be used to view individual entries of an array, a list or a map. This includes arrays, lists or maps inside structures as well. To do so enter the name of the array, map or list including a ")" at the end, select what kind of items to display and press "Display". Note that the content of the display is not automatically updated when the program is in step mode.

"Display all items" simply displays everything. "Display Non-zero items only" will only display those items that do not hold the value 0 or an empty string. This makes viewing large arrays/lists with only few valid items in them simpler. A structure is considered "zero" if all of its items either hold the value 0 or an empty string.

"Display Range" allows displaying only a specific range of an array, a list or a map. The range can be given for each Array dimension individually, separated by commas. If one dimension is not specified at all, all of its items will be displayed. Here are a few examples for valid range input:

```
"1-2, 2-5, 2" : first index between 1 and 2, a second index between 2
               and 5 and a third index of 2.
"1, 2-5"      : first index of 1 and a second index between 2 and 5.
"1, , 5"      : first index of 1, any second index and a third index
               of 5.
```

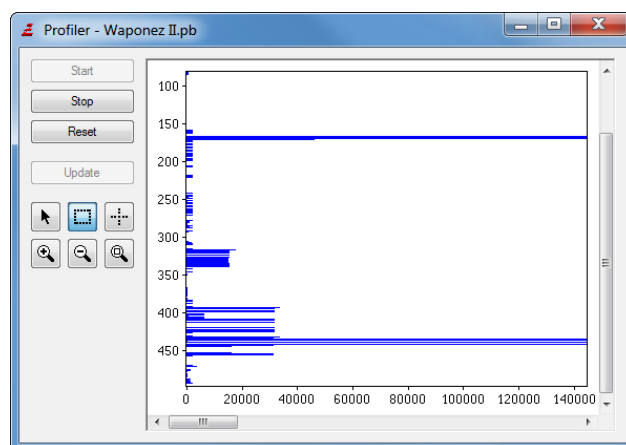
For list display, the "Display Range" option can be used to display a range of list elements by their index (zero based).

```
"0"      : first element
"1-3"    : second to the fourth element
```

For map display, the "Display Range" option can be used to filter the keys to be displayed. It has to contain a mask for the key string of the map elements (no quotation marks). A "?" matches one character, a "*" matches any number of characters. Here are a few examples for valid mask input:

```
"hat" : matches only the item with "hat" as key.
"?at" : matches items with "hat", "bat" etc as key.
"h*t" : matches items with keys that start with "h" and end with "t"
       and anything in between.
```

The Profiler



The profiler tool can count how often each line in the source code is executed. This collected data can be used to identify which portions of the code are used most often and where improvements make most sense. It also helps to identify problems where a piece of the code is executed too often as a result of an error.

Recording the data

The recording of the data can be controlled by the Start, Stop and Reset (to set all counts to 0) buttons in the profiler window. The Update button can be used to update the graph while the program is running. Each time the program is halted or in step mode, the data is updated automatically. By default, the profiler is recording data from the start of the program. This can be changed in the Preferences .

Examining the data

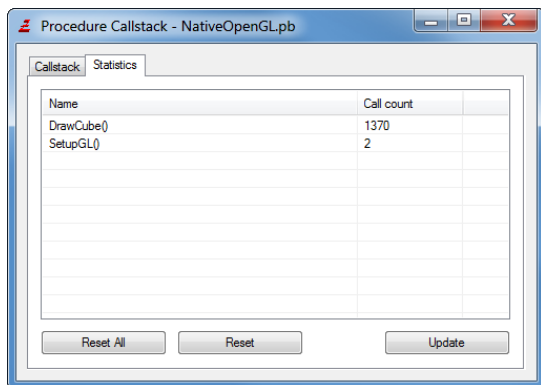
The recorded data is displayed as a graph, with the vertical axis showing the source line number and the horizontal axis showing how often the lines were executed. If the running program consists of more than one source file, a list of source files is presented below the graph. To display the graph for a file either select it, or check its checkbox. Multiple files can be shown in the graph at once to better compare them. Right-clicking on one of the filenames allows changing the color used to display that file in the graph.

Mouse modes in the graph

Right-clicking inside the graph brings up a popupmenu which allows zooming in or out or to show the source line that was clicked on in the IDE or code display of the debugger. The action taken by a left-click can be controlled by the buttons on the left side:

- Arrow button: Left-clicking and dragging allows to scroll the graph display.
- Box button: Left-clicking and dragging allows to select an area which will be zoomed in.
- Cross button: While this button is activated, moving the mouse on the graph displays a crosshair to better identify the line and call count under the mouse.
- Zoom buttons: These buttons allow to zoom in/out and zoom out so all lines can be viewed.

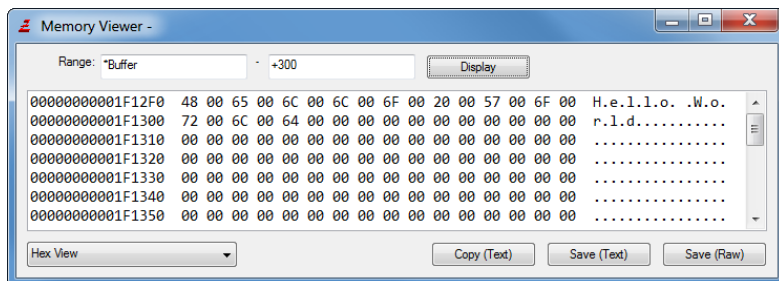
The Callstack viewer



The callstack viewer shows which nested procedure calls led to the current position in the code. Each entry in the list means one procedure that is currently open. It shows the line and file from which it was called, and the arguments used to call the procedure. By clicking on the Variables button for each procedure, you can look at the variables of that instance of the procedure.

This allows to easily trace, from which part of the code, a procedure was called. The callstack view does only automatically update when you stop the program, or use Step to execute single lines. While the program is running, you have to use the Update button to update the view for the current code position. The "Statistics" tab shows the number of times each procedure in the code was called. You can reset the count for all procedures with "Reset all", or for the currently marked entry with the "Reset" button. Like with the callstack, the updates are not automatic while the program is not stopped. Use the Update button for that.

The Memory Viewer



The memory viewer lets you view a memory area in your program. The range to view can be entered into the range fields as any valid PureBasic expression; this can be a normal decimal value, a hex number preceded by the \$ character or any other valid expression, including variables or pointers from the code. If the content of the second range field starts with a "+" sign, the content is interpreted as relative to the first field.

Example: ""*Buffer + 10" to "+30" will display the 30 bytes of memory starting at the location 10 bytes after what *Buffer points to.

If the memory area is valid for viewing, it will be displayed in the area below. If parts of the area are not valid for reading, you will get an error-message. The type of display can be changed with the combo box in the lower left corner. The following view modes are available:

Hex View

The memory will be displayed like in any hex viewer, giving the memory location in hex display on the left, followed by the hexadecimal byte values, and then the string representation in the right column.

Byte/Character/Word/Long/Quad/Float/Double table

The memory area will be shown as a table of the specified variable type. Whether or not this table is single-column or multi-column can be set in the Preferences (see Configuring the IDE). Data will be shown as decimal, octal or hex via the next button that cycles the mode.

String view ascii, unicode or utf-8

This displays the memory area as a string, with any non-string characters displayed in [] (for example "[NULL]" for the 0 byte.) A linebreak is added after newline characters and [NULL] to improve the

readability of the output. The memory area can be interpreted as an Ascii, Unicode or Utf8 string.

Data

With the Data check box, you will export the table view formatted as a Data Section on copy or save.

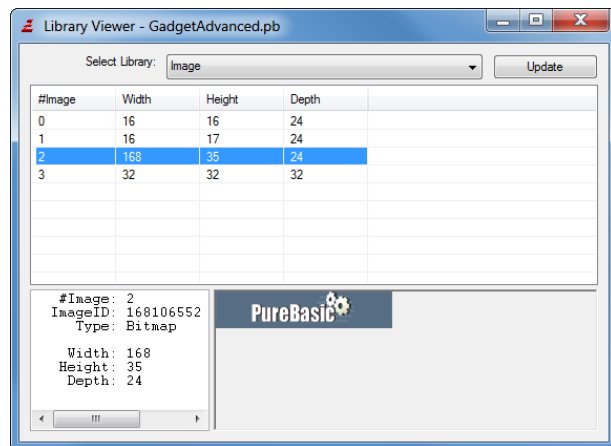
You can also export the viewed memory area from the memory viewer:

Copy (Text): Copies the displayed area as text to the clipboard.

Save (Text): Saves the displayed area as text to a file.

Save (Raw): Saves the memory area as raw binary to a file.

The Library Viewer



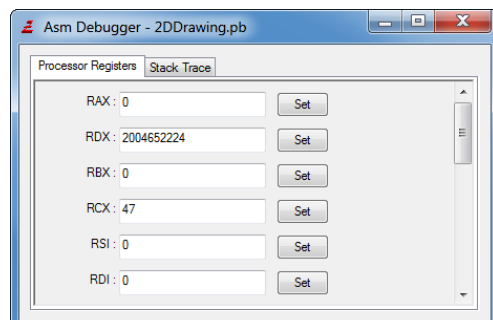
The Library Viewer provides information about the objects created by some libraries. For example, it allows to quickly check which images are currently loaded in the program, or which gadgets are created. Once the program is started, the combobox on top of the window can be used to select the library to view. The list below will then show all objects of the library that currently exist in the executable along with some additional information on each object. The "Update" button will update this list of objects. Selecting an object in the list will display more detailed information about it in the text area on the left, and if supported by the library also a visual display of the object on the right (for Images, Sprites, and so on).

If the combobox displays "No Information", this means that your executable has used no library that supports this feature.

Currently, the Library Viewer is supported by the following libraries:

Thread
Gadget
Window
File
Image
Sprite
XML

The Assembly Debugger



The ASM debugger is provided for advanced programmers to examine and change the CPU register content and to examine the programs stack for debugging of inline ASM code.

The Processor Register view is only available while the program execution is halted. By changing any of the register values and clicking "Set", you can modify the value in that register.

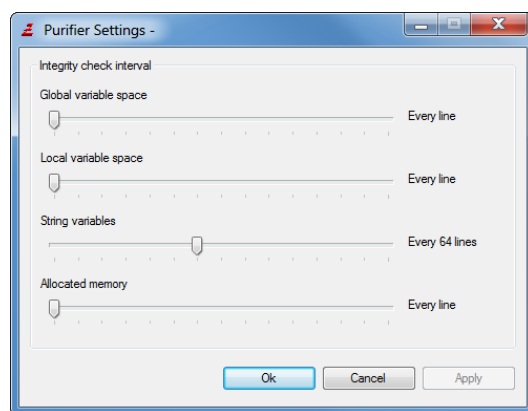
The Stack trace shows the content of the programs stack given in relation to the ESP register. If the current stack position is not aligned at a 4 byte boundary, there can be no information given about the content of the stack. In this case the stack is shown as a hex display.

If the stack pointer is properly aligned, the stack contents are displayed with comments about the meaning of the contained values (detailing the pushed registers and other values on a PureBasic procedure call).

The stack trace is updated automatically when you stop the execution or step through the program, unless you specify otherwise in the Preferences . If you disable the automatic update, there will be an "Update" button displayed to do so manually.

Note: The Assembly debugger is currently not available on MacOS X.

The Purifier



The purifier can detect memory errors such as writing past the end of an allocated memory buffer or string. Without the purifier some of these mistakes would lead to crashes and others would go unnoticed because the write operation overwrites some other valid memory.

The purifier requires special output from the compiler to work, which is why it is only available if the "Enable Purifier" option is set in the compiler options when the program is compiled.

The purifier detects these errors by placing a special 'salt'-value around global and local variables, strings and allocated memory buffers. These salt values are then checked at regular intervals and an error is displayed if they were changed. These checks slow down the program execution considerably especially for large programs, which is why the rate at which the checks are performed can be specified in the purifier window:

Global variable space

Defines the interval in source lines after which the global variables are checked.

Local variable space

Defines the interval in source lines after which the local variables are checked.

String variables

Defines the interval in source lines after which the memory used by string variables is checked.

Allocated memory

Defines the interval in source lines after which the memory allocated by `AllocateMemory()` is checked.

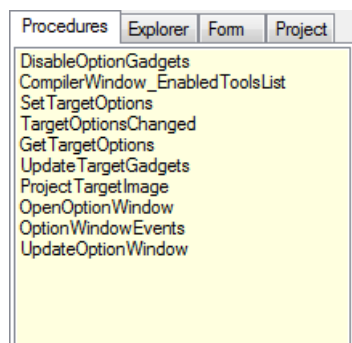
Chapter 15

Using the built-in Tools

The PureBasic IDE comes with many building tools, to make programming tasks easier and increase your productivity. Many of them can be configured to be either accessible from the Menu as separate windows, or to be permanently displayed in the Panel on the side of the editing area. For information on how to configure these tools and where they are displayed, see [Configuring the IDE](#) .

Tools for the Side Panel Area

Procedure Browser



This tool displays a list of all procedures and macros declared in the current source code. By double-clicking on an entry in that list, the cursor automatically jumps to that procedure.

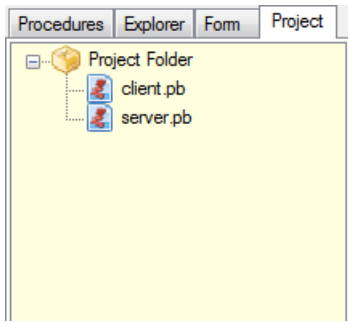
Macros will be marked in the list by a "+" sign before the name.

You can also place special comment marks in your code, that will be displayed in the list too. They look like this: ";- <description>". The ; starts a comment, the - that follows it immediately defines such a mark.

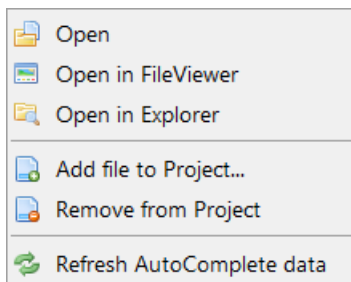
The description will be shown in the Procedure list, and clicking on it will jump to the line of this mark. Such a comment mark can be distinguished from a Procedure by the ">" that is displayed before it in the procedure list.

The list of procedures can be sorted, and it can display the procedure/macro arguments in the list. For these options, see [Configuring the IDE](#) .

Project Panel



This tool displays a tree of all files in the current project . A double-click on a file opens it in the IDE. This allows fast access to all files in the project. A right-click on a file opens a context menu which provides more options:



Open - Open the file in the IDE.

Open in FileViewer - Open the file in the FileViewer of the IDE.

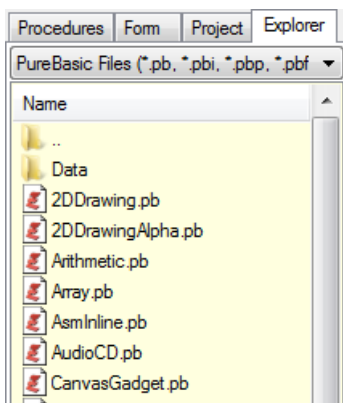
Open in Explorer - Open the file in the operating systems file manager.

Add File to Project - Add a new file to the project.

Remove File from Project - Remove the selected file(s) from the project.

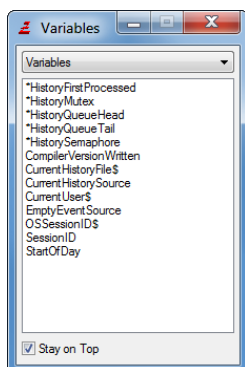
Refresh AutoComplete data - Rescan the file for AutoComplete items.

Explorer



The Explorer tool displays an explorer, from which you can select files and open them quickly with a double-click. PureBasic files (*.pb, *.pbi, *.pbp, *.pbf) will be loaded into the edit area and all other recognized files (text & binary) files will be displayed into the internal File Viewer.

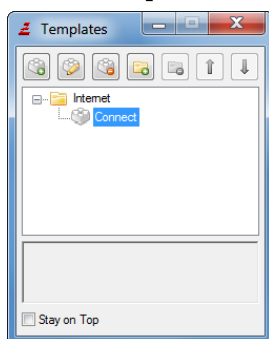
Variable Viewer



The variable viewer can display variables , Arrays , lists , Constants , Structures and Interfaces defined in your source code, or any currently opened file. You can configure what exactly it should display in the preferences .

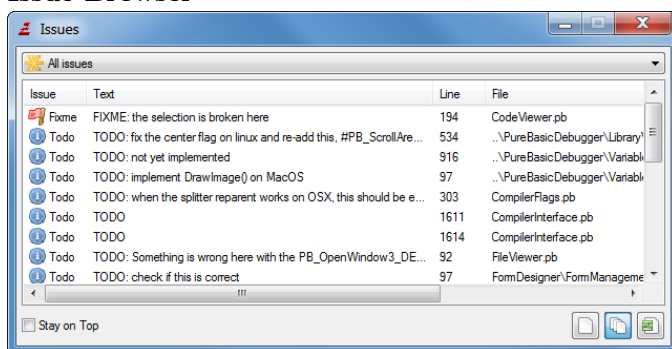
Note: The displaying of variables is somewhat limited for now. It can only detect variables explicitly declared with Define , Global , Shared , Protected or Static .

Code Templates



The templates tool allows you to manage a list of small code parts, that you can quickly insert into your source code with a double-click. It allows you to manage the codes in different directories, and put a comment to each code. This tool is perfect to manage small, often used code parts.

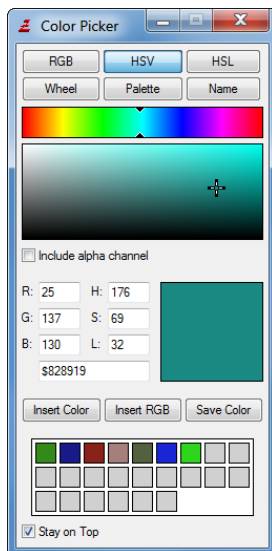
Issue Browser



The issue browser tool collects comments in the source code that fit a defined format and lists them ordered by priority. It can be used to track which areas of the source code still need to be worked on. Each displayed issue corresponds to one comment in the code. A double-click on the issue shows that code line. Issues can be displayed for the current file, or for multiple files (all open files, or all files that belong to the current project). The issue list can also be exported in CSV format.

To configure the collected issues, see the "Issues" section in the Preferences .

Color Picker



The color picker helps you to find the perfect color value for whatever task you need. The following methods of picking a color are available:

RGB: Select a color by choosing red, green and blue intensities.

HSV: Select a color by choosing hue, saturation and value.

HSL: Select a color by choosing hue, saturation and lightness.

Wheel: Select a color using the HSV model in a color wheel.

Palette: Select a color from a predefined palette.

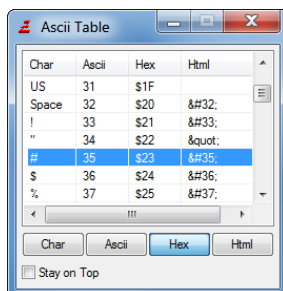
Name: Select a color from a palette by name.

The color selection includes an alpha component, if the "Include alpha channel" checkbox is activated.

The individual components (red/green/blue intensities or hue/saturation/lightness) as well as the hexadecimal representation of the current color can be seen and modified in the text fields.

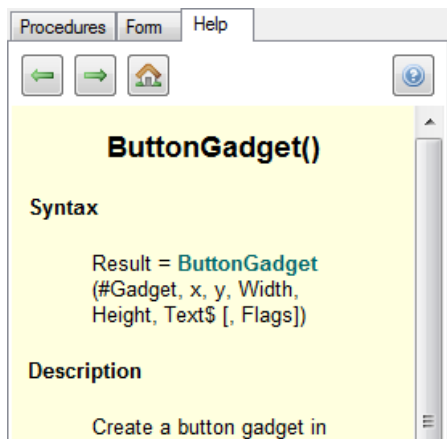
The "Insert Color" button inserts the hexadecimal value of the current color in the source code. The "Insert RGB" button inserts the color as a call to the RGB() or RGBA() function into the code. The "Save Color" button saves the current color to the history area at the bottom. Clicking on a color in the history makes it the current color again.

Character Table



The character table tool displays a table showing the first 256 unicode characters, together with their index in decimal and hex, as well as the corresponding html notation. By double-clicking on any line, this character will be inserted into the source code. With the buttons on the bottom, you can select which column of the table to insert on a double-click.

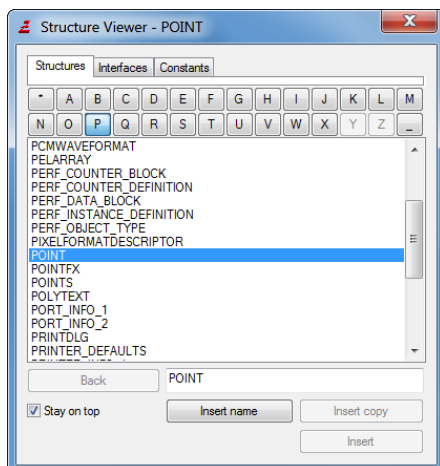
Help Tool



The Help Tool is an alternative viewer for the reference guide . It can be used to view the PureBasic manual side by side with the code. Whether or not the F1 shortcut opens the manual in the tool or as a separate window can be specified in the preferences .

Other built-in tools

Structure Viewer



The structure viewer allows you to view all the Structures, Interfaces and Constants predefined in PureBasic. Double-clicking on a Structure or Interface shows the declaration. On top of the list you can select a filter to display only entries that start with a given character.

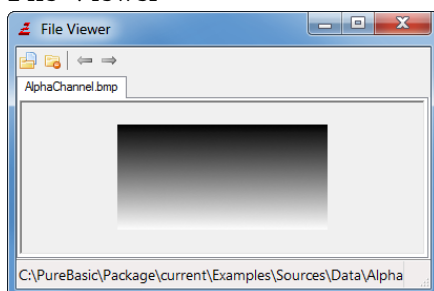
The "Back" button navigates back through the viewed entries.

"Insert name" inserts just the name of the selected entry.

"Insert copy" inserts a copy of the declaration of that entry.

"Insert" lets you enter a variable name and then inserts a definition of that variable and the selected entry and all elements of it.

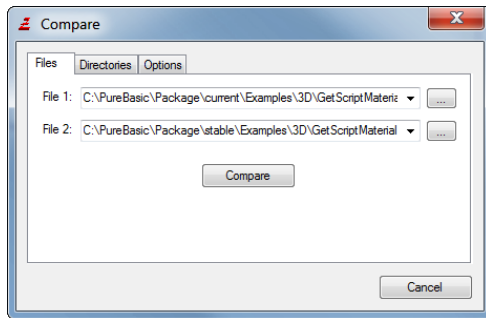
File Viewer



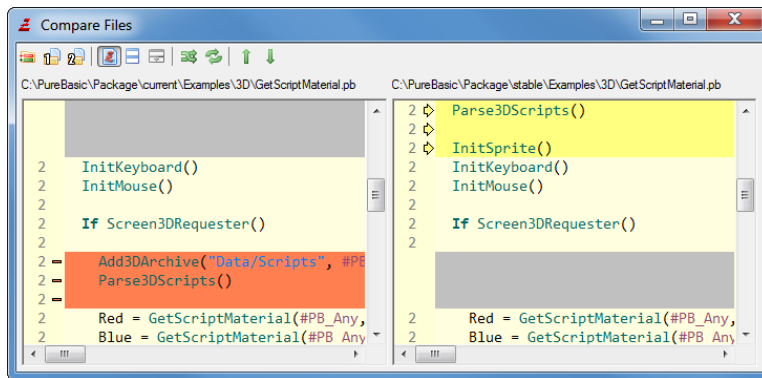
The internal file viewer allows you do display certain types of files. Text files, images and web pages

(windows only). Any unknown file type will be displayed in a hex-viewer. The "Open" button opens a new file, the "X button" closes it and the arrows can be used to navigate through the open files. Also any binary file that you attempt to open from the Explorer tool, or by double-clicking on an IncludeBinary keyword will be displayed in this file viewer.

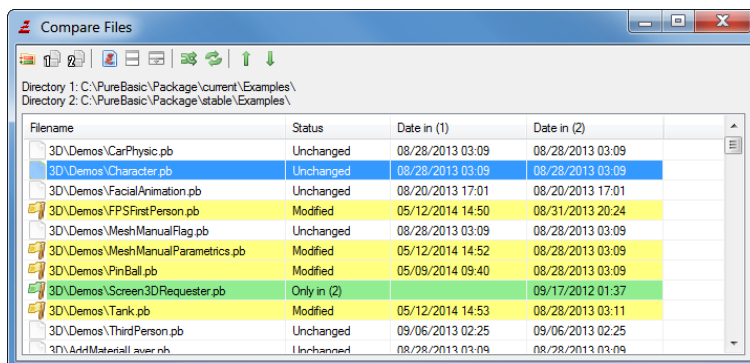
Compare Files/Folders



This tool can compare two (text-) files or two directories and highlight their differences. The "Options" tab can be used to ignore some differences such as spaces or upper/lowercase changes.



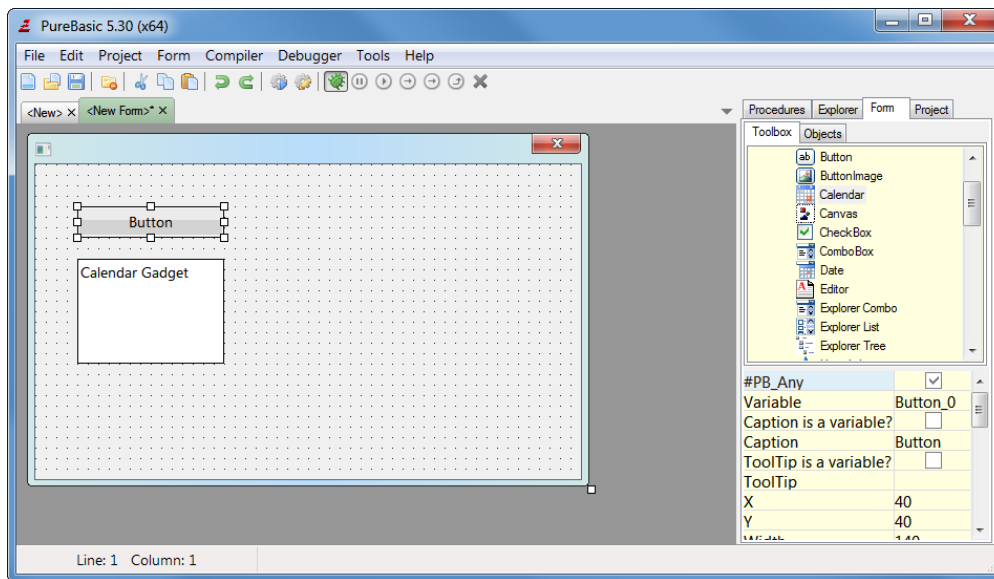
The files are shown side by side with the differences marked in the following way: Lines shown in red were removed in the file on the right, lines shown in green were added in the file on the right and lines shown in yellow were changed between the two files.



When comparing directories, the content of both directories is examined (with the option to filter the search by file extension and include subdirectories) and the files are marked in a similar way: Files in red do not exist in the second directory, files in green are new in the second directory and files in yellow were modified. A double-click on a modified file shows the modifications made to that file.

Other entries in the Tools menu

Form Designer



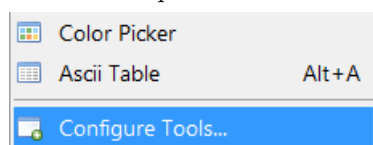
The Form Designer can be used to design the user interface for your application. For more information, see the form designer chapter.

Chapter 16

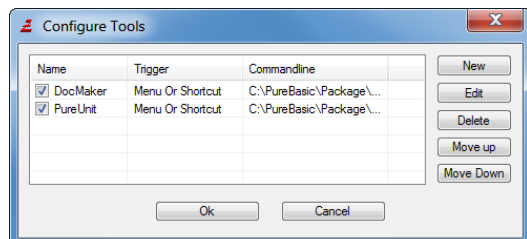
Using external tools

The PureBasic IDE allows you to configure external programs to be called directly from the IDE, through the Menu, Shortcuts, the Toolbar, or on special "triggers". The use of this is to make any other program you use while programming easily accessible.

You can also write your own little tools in PureBasic that will perform special actions on the source code you are currently viewing to automate common tasks. Furthermore, you can configure external file viewers to replace the internal File Viewer of the IDE for either specific file types or all files.



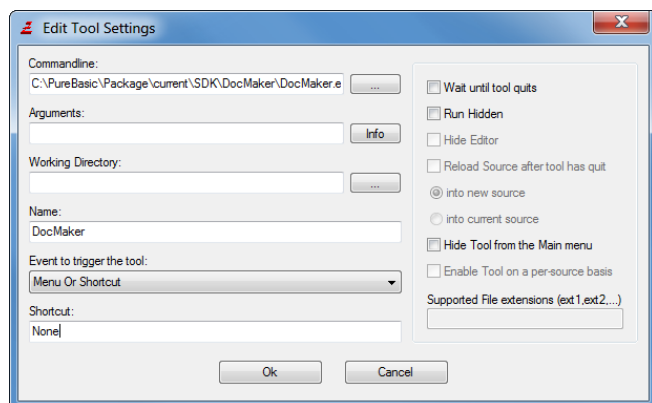
With the "Config tools" command in the Tools menu, you can configure such external tools. The list you will see displays all the configured tools in the order they appear in the Tools menu (if not hidden). You can add and remove tools here, or change the order by clicking "Move Up"/"Move Down " after selecting an item.



Any tool can be quickly enabled or disabled from the "Config tools" window with the checkbox before each tool entry. A checked checkbox means the tool is enabled, an unchecked one means it is currently disabled.

Configuring a tool

The basic things you need to set is the command-line of the program to run, and a name for it in the Tools list/Menu. Everything else is optional.



Command-line

Select the program name to execute here.

Arguments

Place command-line arguments that will be passed to the program here. You can place fixed options, as well as special tokens that will be replaced when running the program:

%PATH : will be replaced with the path of the current source code. Remains empty if the source was not saved.

%FILE : filename of the current source code. Remains empty if it has not yet been saved. If you configure the tool to replace the file viewer, this token represents the file that is to be opened. You should never modify the contents of the %FILE file. This file contains the state of the source code as the user last saved it. This could be much different from what is currently visible in the editor. If you overwrite this file, the user might lose data.

%TEMPFILE : When this option is given, the current source code is saved in a temporary file, and the filename is inserted here. You may modify or delete the file at will.

%COMPILEFILE : This token is only valid for the compilation triggers (see below). This is replaced with the temporary file that is sent to the compiler for compilation. By modifying this file, you can actually change what will be compiled.

%EXECUTABLE : This will be replaced by the name of the executable that was created in with the last "Create Executable". For the "After Compile/Run" trigger, this will be replaced with the name of the temporary executable file created by the compiler.

%CURSOR : this will be replaced by the current cursor position in the form of LINExCOLUMN.

%SELECTION : this will be replaced by the current selection in the form of LINESTARTxCOLUMNSTARTxLINEENDxCOLUMNEND. This can be used together with %TEMPFILE, if you want your tool to do some action based on the selected area of text.

%WORD : contains the word currently under the cursor.

%PROJECT : the full path to the directory containing the project file if a project is open.

%HOME : the full path to the purebasic directory

Note: for any filename or path tokens, it is generally a good idea to place them in "" (i.e.

"%TEMPFILE") to ensure also paths with spaces in them are passed correctly to the tool. These tokens and a description can also be viewed by clicking the "Info" button next to the Arguments field.

Working Directory

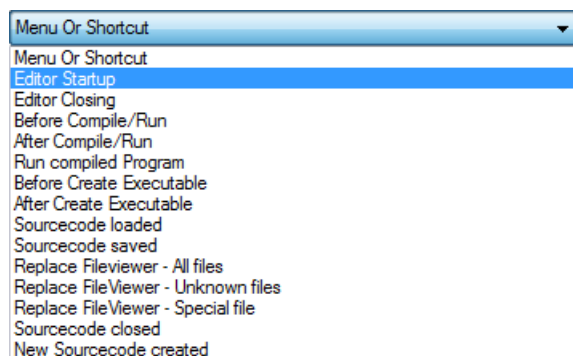
Select a directory in which to execute this tool. By specifying no directory here, the tool will be executed in the directory of the currently open source code.

Name

Select a name for the tool. This name will be displayed in the tools list, and if the tool is not hidden from the menu, also in the Tools menu.

Event to trigger the tool

Here you can select when the tool should be executed. Any number of tools can have the same trigger, they will all be executed when the trigger event happens. The order of their execution depends on the order they appear in the tools list.



Menu Or Shortcut

The tool will not be executed automatically. It will be run by a shortcut or from the Menu. Note: to execute a tool from the Toolbar, you have to add a button for it in the Toolbar configuration in the Preferences (see Configuring the IDE for more).

With this trigger set, the "Shortcut" option below becomes valid and lets you specify a shortcut that will execute this tool.

Editor Startup

The tool will be executed right after the IDE has been fully started.

Editor End

The tool will be executed right before the IDE ends. Note that all open sources have already been closed at this time.

Before Compile/Run

The tool will be executed right before the compiler is called to compile a source code. Using the %COMPILEFILE token, you can get the code to be compiled and modify it. This makes it possible to write a small pre-processor for the source code. Note that you should enable the "Wait until tool quits" option if you want your modifications to be given to the compiler.

After Compile/Run

The tool will be executed right after the compilation is finished, but before the executable is executed for testing. Using the %EXECUTABLE token, you can get access to the file that has just been created. Note that you can modify the file, but not delete it, as that results in an error-message when the IDE tries to execute the file.

Run compiled Program

The tool will be executed when the user selects the "Run" command from the compiler menu. The tool is executed before the executable is started. The %EXECUTABLE token is valid here too.

Before create Executable

The same as for the "Before Compile/Run" trigger applies here too, only that the triggering event is when the user creates the final executable.

After create Executable

The tool is executed after the compilation to create the final executable is complete. You can use the %EXECUTABLE token to get the name of the created file and perform any further action on it.

Source code loaded

The tool is executed after a source code has been loaded into the IDE. The %FILE and %PATH tokens are always valid here, as the file was just loaded from the disk.

Source code saved

The tool will be executed after a source code in the IDE has been saved successfully. The %FILE and %PATH tokens are always valid here, as the file has just been saved to disk.

Source code closed

The tool will be executed whenever a source file is about to be closed. At this point the file is still there, so you can still get its content with the %TEMPFILE token. %FILE will be empty if the file was never saved.

File Viewer All Files

The tool will completely replace the internal file viewer. If an attempt is made in the IDE to open a file that cannot be loaded into the edit area, the IDE will first try the tools that have a trigger set for the specific file type, and if none is found, the file will be directed to this tool. Use the %FILE token to get the filename of the file to be opened.

Note: Only one tool can have this trigger. Any other tools with this trigger will be ignored.

File Viewer Unknown file

This tool basically replaces the hex viewer, which is usually used to display unknown file types. It will be executed, when the file extension is unknown to the IDE, and if no other external tool is configured to handle the file (if a tool is set with the "File Viewer All Files" trigger, then this tool will never be called). Note: Only one tool can have this trigger set.

File Viewer Special file

This configures the tool to handle specific file extensions. It has a higher priority than the "File Viewer All files" or "File Viewer Unknown file" triggers and also higher than the internal file viewer itself.

Specify the extensions that the tool should handle in the edit box on the right. Multiple extensions can be given.

A common use for this trigger is for example to configure a program like Acrobat Reader to handle the "pdf" extension, which enables you to easily open pdf files from the Explorer, the File Viewer, or by double-clicking on an Includebinary statement in the source.

Open File with specific extension

This will be triggered for specific file extensions. It has a higher priority than the "Open File non-PB binary file" or "Open File non-PB text file" triggers. Specify the extensions that the tool should handle in the edit box on the right. Multiple extensions can be given.

Note: The Open File triggers will be active when you open a file via the File/Open menu and also, when you drag and drop a file to the IDE. They have higher priority for File/Open tasks. Only if there is no active tool, the File Viewer triggers will be handled.

Open File non-PB binary file

This one will be triggered for binary files, which are not part of PureBasic (more or less any)

Note: Only one tool can have this trigger. Any other tools with this trigger will be ignored.

Open File non-PB text file

This one will be triggered for text files, which are not part of PureBasic

Note: Only one tool can have this trigger set.

Other options on the right side

Wait until tool quits

The IDE will be locked for no input and cease all its actions until you tool has finished running. This option is required if you want to modify a source code and reload it afterwards, or have it passed on to the compiler for the compilation triggers.

Run hidden

Runs the program in invisible mode. Do not use this option for any program that might expect user input, as there will be no way to close it in that case.

Hide editor

This is only possible with the "wait until tool quits" option set. Hides the editor while the tool is running.

Reload Source after the tool has quit

This is only possible with the "wait until tool quits" option set, and when either the %FILE or %TEMPFILE tokens are used in the Arguments list.

After your program has quit, the IDE will reload the source code back into the editor. You can select whether it should replace the old code or be opened in a new code view.

Hide Tool from the Main menu

Hides the tool from the Tools menu. This is useful for tools that should only be executed by a special trigger, but not from the menu.

Enable Tool on a per-source basis

Tools with this option set will be listed in the "Execute tools" list in the compiler options , and only executed for sources where it is enabled there. Note that when disabling the tool with the checkbox here in the "Config tools" window, it will be globally disabled and not run for any source code, even if enabled there.

This option is only available for the following triggers:

- Before Compile/Run
- After Compile/Run
- Run compiled Program
- Before create Executable
- After create Executable
- Source code loaded
- Source code saved

- Source code closed

Supported File extensions

Only for the "File Viewer Special file" trigger. Enter the list of handled extensions here.

Tips for writing your own code processing tools

The IDE provides additional information for the tools in the form of environment variables. They can be easily read inside the tool with the commands of the Process library .

This is a list of provided variables. Note that those that provide information about the active source are not present for tools executed on IDE startup or end.

PB_TOOL_IDE	- Full path and filename of the IDE
PB_TOOL_Compiler	- Full path and filename of the Compiler
PB_TOOL_Preferences	- Full path and filename of the IDE's Preference file
PB_TOOL_Project	- Full path and filename of the currently open project (if any)
PB_TOOL_Language	- Language currently used in the IDE
PB_TOOL_FileList	- A list of all open files in the IDE, separated by Chr(10)
PB_TOOL_Debugger	- These variables provide the settings from the Compiler Options
PB_TOOL_InlineASM	window for the current source. They are set to "1" if the option
PB_TOOL_Unicode	is enabled, and "0" if not.
PB_TOOL_Thread	
PB_TOOL_XPSkin	
PB_TOOL_OnError	
PB_TOOL_SubSystem	- content of the "Subsystem" field in the compiler options
PB_TOOL_Executable	- same as the %COMPILEFILE token for the command-line
PB_TOOL_Cursor	- same as the %CURSOR token for the command-line
PB_TOOL_Selection	- same as the %SELECTION token for the command-line
PB_TOOL_Word	- same as the %WORD token for the command-line
PB_TOOL_MainWindow	- OS handle to the main IDE window
PB_TOOL_Scintilla	- OS handle to the Scintilla editing component of the current source

When the %TEMPFILE or %COMPILEFILE tokens are used, the IDE appends the compiler options as a comment to the end of the created temporary file, even if the user did choose to not save the options there when saving a source code.

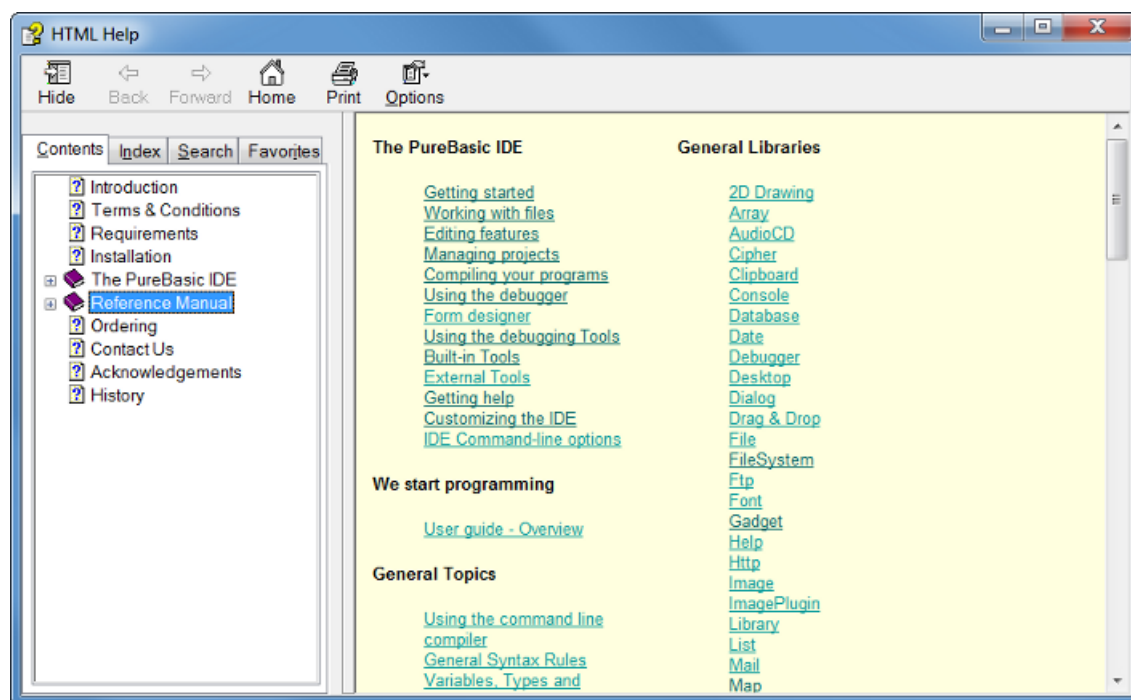
This enables your tool to read the compiler settings for this file, and take them into account for the actions your carries out.

Chapter 17

Getting Help

The PureBasic IDE provides ways to access the PureBasic help-file, as well as other files and documentation you want to view while programming.

Quick access to the reference guide



By pressing the help shortcut (F1 by default) or selecting the "Help..." command from the Help menu while the mouse cursor is over a PureBasic keyword or function, the help will be opened directly at the description of that keyword or function.

If the word at the cursor position has no help entry, the main reference page will be displayed.

The reference manual can also be viewed side by side with the source code using the Help Tool .

Quick access to Windows API help

There are two ways to get the same quick access as for the PureBasic functions (by pressing F1 with the cursor on the function) also for functions of the Windows API. To enable this feature you have to download additional help files for these functions:

Microsoft Platform SDK The Microsoft Platform SDK provides the most complete and up to date programming reference available for the windows platform. It provides information on all API functions, as well as overviews and introductions to the different technologies used when programming for the windows platform. It is however quite big (up to 400MB depending on the selected components). For the IDE help, you can either install the "February 2003" or the "Windows Server 2003 SP1" edition of the SDK.

It can be downloaded from here:

<https://www.microsoft.com/en-us/download/details.aspx?id=15656>

Note that the SDK can also be ordered on CD from this page. Also if you are the owner of any development products from Microsoft (such as Visual Studio), there might be a copy of the SDK included on one of the CDs that came with it.

The win32.hlp help-file There is a much smaller alternative to the complete SDK by Microsoft (7.5 MB download). This help is quite old (written for Windows95 in fact), so it does not provide any information on new APIs and technologies introduced since then.

However, it provides good information about commonly used API that is still valid today, as these mostly did not change. This download is recommended if you only need occasional help for API functions, but do not want to download the full SDK.

It can be downloaded from here:

<http://www.purebasic.com/download/WindowsHelp.zip>

To use it from the PureBasic IDE, just create a "Help" subdirectory in your PureBasic folder and copy the "win32.hlp" file into it.

Accessing external helpfiles from the IDE

If you have other helpfiles you wish to be able to access from the IDE, then create a "Help" subdirectory in your PureBasic folder and copy them to it. These files will appear in the "External Help" submenu of the Help menu, and in the popupmenu you get when right-clicking in the editing area. Chm and Hlp files will be displayed in the MS help viewer. The IDE will open the helpfiles in the internal fileviewer. So files like text files can be viewed directly like this. For other types, you can use the Config Tools menu to configure an external tool to handle the type of help-file you use. The help will then be displayed in that tool.

For example, if you have pdf helpfiles, configure an external tool to handle pdf files and put the files in the Help subdirectory of PureBasic. Now if you click the file in the "external help" menu, it will be opened in that external tool.

Chapter 18

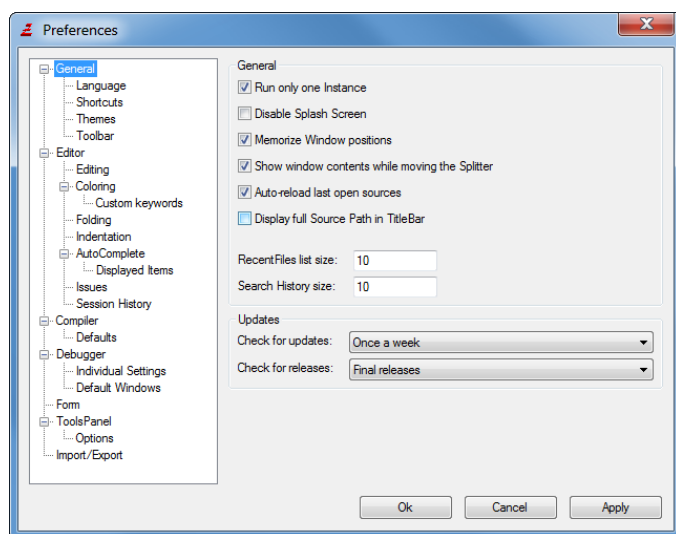
Customizing the IDE

The PureBasic IDE provides many options to customize or disable some of its features in order to become the perfect tool for you.

These options are accessible from the Preferences command in the File menu, and the meaning of each setting is described here.

Any changes made will only take effect once you click the "OK" button or "Apply".

General



Options that affect the general behavior of the IDE.

Run only one Instance

If set, prevents the IDE from being opened more than once. Clicking on a PB file in the explorer will open it in the already existing IDE instance instead of opening a new one.

Disable Splash screen

Disables the splash screen that is displayed on start-up.

Memorize Window positions

Remembers the position of all IDE windows when you close them. If you prefer to have all windows open at a specific location and size, enable this option, move all windows to the perfect position, then restart the IDE (to save all options) and then disable this option to always open all windows in the last saved position.

Show window contents while moving the Splitter

Enable this only if you have a fast computer. Otherwise moving the Splitter bar to the Error Log or Tools Panel may flicker a lot.

Auto-Reload last open sources

On IDE start-up, opens all the sources that were open when the IDE was closed the last time.

Display full Source Path in Title bar

If set, the IDE title bar will show the full path to the currently edited file. If not, only the filename is shown.

Enable Appearance Dark Mode

If set, allows the appearance 'Dark Mode'.

Recent Files list

This setting specifies how many entries are shown in the "Recent Files" submenu of the File menu.

Search History size

This setting specifies how many recent search words are remembered for "Find/Replace" and "Find in Files"

Check for updates

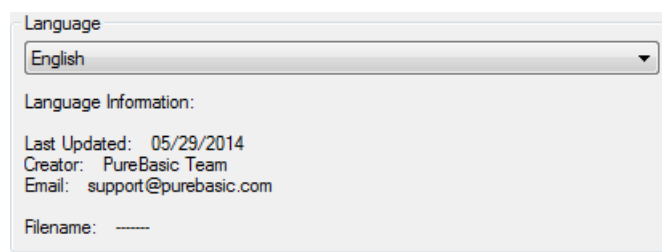
Specifies how often the IDE should check on the purebasic.com server for the availability of new updates.

An update check can also be performed manually at any time from the "Help" menu.

Check for releases

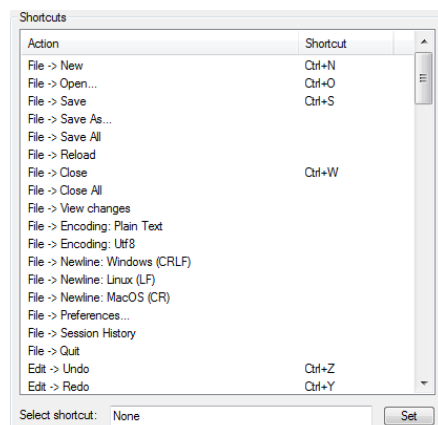
Specifies which kind of releases should cause a notification if they are available.

General - Language



This allows you to change the language of the IDE. The combo box shows the available languages, and you can view some information about the language file (for example who translated it and when it was last updated).

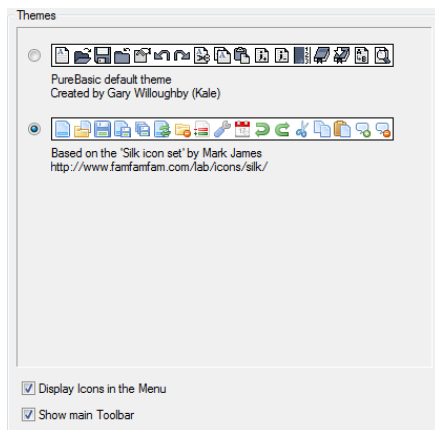
General - Shortcuts



Here you can fully customize all the shortcut commands of the IDE. Select an entry from the list, select the shortcut field, enter the new key combination and click "Set" to change the entry.

Note that Tab & Shift+Tab are reserved for block-indentation and un-indentation and cannot be changed. Furthermore some key combination might have a special meaning for the OS and should therefore not be used.

General - Themes



This section shows the available icon themes for the IDE and allows to select the theme to use. The IDE comes with two themes by default.

More themes can be easily added by creating a zip-file containing the images (in png format) and a "Theme.prefs" file to describe the theme. The zip-file has to be copied to the "Themes" folder in the PureBasic installation directory to be recognized by the IDE. The "SilkTheme.zip" file can be used as an example to create a new theme.

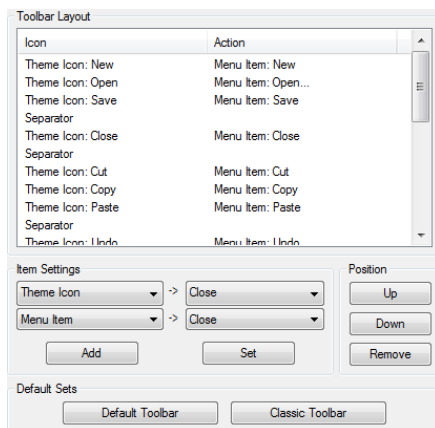
Display Icons in the Menu

Allows to hide/show the images in the IDE menus.

Show main Toolbar

Allows to hide/show the main toolbar in order to gain space for the editing area.

General - Toolbar



This allows to fully customize the main Toolbar. By selecting an entry and using the Buttons in the "Position" section, you can change the order. The "Item Settings" section can be used to modify the entry or add a new one. New ones are always added at the end of the list.

Types of items:

Separator : a vertical separator line.

Space : an empty space, the size of one toolbar icon.

Standard Icon : allows you to select a OS standard icon from the combo box on the right.

IDE Icon : allows you to select one of the IDE's own icons in the combo box on the right.

Icon File : allows you to specify your own icon file to use in the edit box on the right (PNG files are supported on all platforms, Windows additionally supports icon files).

If you do not select a separator or space, you can specify an action to carry out when the button is pressed:

Menu Item : carries out the menu command specified in the combo box on the right.

Run tool : executes the external tool specified in the combo box on the right.

The "Default Sets" section contains two standard toolbar sets which you can select, and later modify.

Editor

The screenshot shows a settings dialog box with two main sections: 'File selection' and 'Editor'. In the 'File selection' section, there are three checkboxes: 'Display multiple rows' (unchecked), 'Display close buttons in each tab' (checked), and 'Add a tab to create a new source' (unchecked). The 'Editor' section contains several checkboxes: 'Monitor open files for changes on disk' (checked), 'Auto-save before compiling' (checked), 'Save all sources with Auto-save' (checked), 'Memorize Cursor position' (checked), 'Memorize Marker positions' (checked), and 'Always hide the error log (ignore the per-source setting)' (unchecked). Below these are three fields: 'Save Settings to:' with a dropdown menu showing 'A common file project.cfg for every directory', 'Tab Length:' with a text box containing '2' and an unchecked checkbox 'Use real Tab (ASCII 9)', and 'Source Directory:' with a text box containing 'C:\PureBasic\Libraries\current\json\' and a browse button. At the bottom is a text box for 'Code file extensions:'.

Settings that affect the management of the source codes.

Monitor open files for changes on disk

Monitors all open files for changes that are made to the files on disk while they are edited in the IDE. If modifications are made by other programs, a warning is displayed with the choice to reload the file from disk.

Auto-save before compiling

Saves the current source code before each compile/run or executable creation. Note that any open include files are not automatically saved.

Save all sources with Auto-save

Saves all sources instead of just the current one with one of the Auto-save options.

Memorize cursor position

Saves the current cursor position, as well as the state of all folding marks with the compiler options for the source file.

Memorize Marker positions

Saves all the Markers with the options for the source file.

Always hide the error log

The error log can be shown/hidden on a per-source basis. This option provides a global setting to ignore the per-source setting and never display the error log. It also removes the corresponding menu entries from the IDE menu.

Save settings to

This option allows to specify where the compiler options of a source file are saved:

The end of the Source file

Saves the settings as a special comment block at the end of each source file.

The file <filename>.pb.cfg

Creates a .pb.cfg file for each saved source code that contains this information.

A common file project.cfg for every directory

Creates a file called project.cfg in each directory where PB files are saved. This one file will contain the options for all files in that directory.

Don't save anything

No options are saved. When reopening a source file, the defaults will always be used.

Tab Length

Allows to specify how many spaces are inserted each time you press the Tab key.

Use real Tab (Ascii 9)

If set, the tab key inserts a real tab character instead of spaces. If not set, there are spaces inserted when Tab is pressed.

Note that if real tab is used, the "Tab Length" option specifies the size of one displayed tab character.

Source Directory

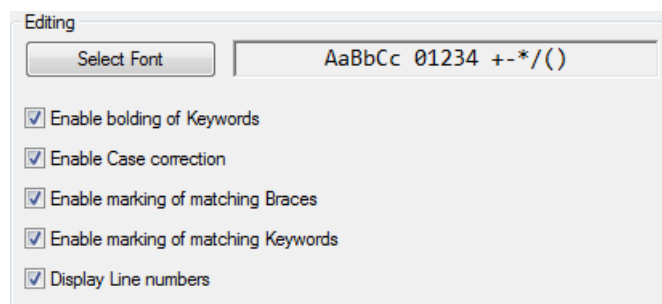
Specifies the default directory used in the Open and Save dialogs if no other files are currently open (if another file is open, its path will be used as default).

Set this to the path where you usually save the source codes.

Code file extensions

The IDE detects code files by their extension (pb, pbi or pbf by default). Non-code files are edited in a "plain text" mode in which code-related features are disabled. This setting causes the IDE to recognize further file extensions as code files. The field can contain a comma-separated list (i.e. "pbx, xyz") of extensions to recognize.

Editor - Editing



Use "Select Font" to change the font used to display the source code. To ensure a good view of the source code, it should be a fixed-size font, and possibly even one where bold characters have the same size as non-bold ones.

Enable bolding of keywords

If your font does not display bold characters in the same size as non-bold ones, you should disable this option. If disabled, the keywords will not be shown as bold.

Enable case correction

If enabled, the case of PureBasic keywords, PureBasic Functions as well as predefined constants will automatically be corrected while you type.

Enable marking of matching Braces

If enabled, the brace matching the one under the cursor will be highlighted.

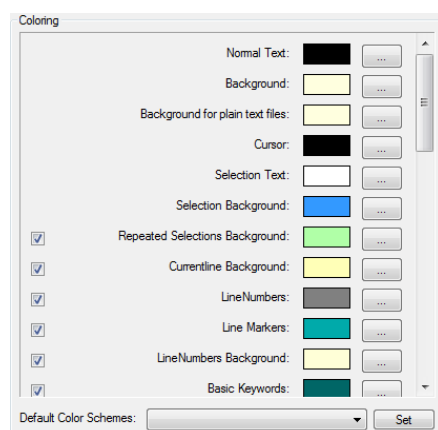
Enable marking of matching Keywords

If enabled, the keyword(s) matching the one under the cursor will be underlined.

Display line numbers

Shows or hides the line number column on the left.

Editor - Coloring



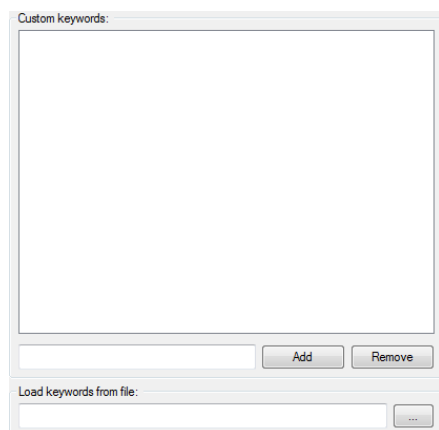
Here you can change the color settings for the syntax coloring, as well as the debugger marks. Default

color schemes can be selected from the box on the bottom, and also modified after they have been set. Individual color settings can be disabled by use of the checkboxes.

Note: The 'Accessibility' color scheme has (apart from high-contrast colors) a special setting to always use the system color for the selection in the code editor. This helps screen-reader applications to better detect the selected text.

More color schemes can be added by creating scheme definition files with the name format "SchemeName.prefs". The files have to be copied to the "ColorSchemes" folder in the PureBasic installation directory to be recognized by the IDE. The default scheme files can be used as examples to create a new scheme, or you can export your colors from the Import/Export section of the Preferences window (select "Include Color settings").

Editor - Coloring - Custom Keywords

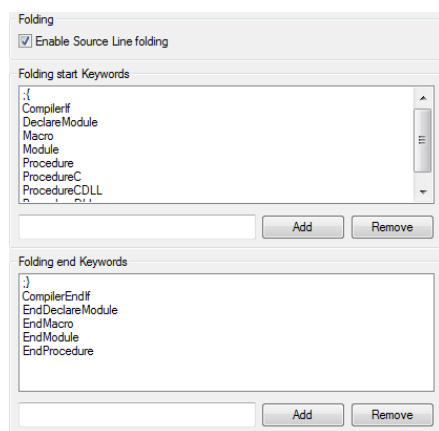


In this section, a list of custom keywords can be defined. These keywords can have a special color assigned to them in the coloring options and the IDE will apply case-correction to them if this feature is enabled. This allows for applying a special color to special keywords by preprocessor tools or macro sets, or to simply have some PB keywords colored differently.

Note that these keywords take precedence above all other coloring in the IDE, so this allows to change the color or case correction even for PureBasic keywords.

The keywords can be either entered directly in the preferences or specified in a text file with one keyword per line (or both).

Editor - Folding

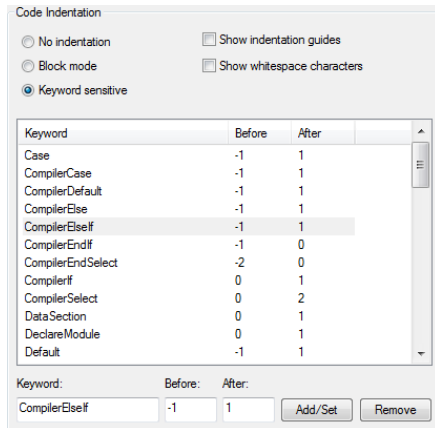


Here you can set the keywords in the source code that start/end a foldable section of code. You can add any number of words that will mark such a sections. You can also choose to completely disable the folding feature.

Words that are found inside comments are ignored, unless the defined keyword includes the comment

symbol at the start (like the default ";" keyword).
A keyword may not include spaces.

Editor - Indentation



Here you can specify how the editor handles code indentation when the return key is pressed.

No indentation

Pressing return always places the cursor at the beginning of the next line.

Block mode

The newly created line gets the same indentation as the one before it.

Keyword sensitive

Pressing the return key corrects the indentation of both the old line and the new line depending on the keywords on these lines. The rules for this are specified in the keyword list below. These rules also apply when the "Reformat indentation" item in the edit menu is used.

Show indentation guides

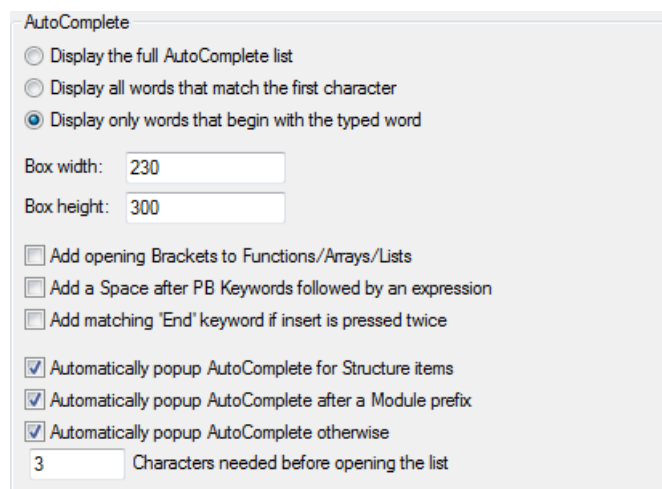
Causes vertical lines to be shown to visualize the indentation on each line. This makes it easier to see which source lines are on the same level of indentation.

Show whitespace characters

Causes whitespace characters to be visible as little dots (spaces) or arrows (tab characters).

The keyword list contains the keywords that have an effect on the indentation. The "Before" setting specifies the change in indentation on the line that contains the keyword itself while the "After" setting specifies the change that applies to the line after it.

Editor - Auto complete



Display the full Auto complete list

Always displays all keywords in the list, but selects the closest match.

Display all words that start with the first character

Displays only those words that start with the same character as you typed. The closest match is selected.

Display only words that start with the typed word

Does not display any words that do not start with what you typed. If no words match, the list is not displayed at all.

Box width / Box height

Here you can define the size of the auto complete list (in pixel). Note that these are maximum values. The displayed box may become smaller if there are only a few items to display.

Add opening Brackets to Functions/Arrays/Lists

Will automatically add a "(" after any function/Array/List inserted by auto complete. Functions with no parameters or lists get a "()" added.

Add a Space after PB Keywords followed by an expression

When inserting PB keywords that cannot appear alone, a space is automatically added after them.

Add matching End' keyword if insert shortcut is pressed twice

When the selected keyword has a complementary end keyword, for example "EndSelect" to "Select" or "EndIf" to "If", the insert shortcut (default Tab) can be repeated a second time to automatically add this text too. The end keyword will be inserted after the cursor, so you can continue typing after the first keyword that was inserted.

Automatically popup AutoComplete for Structure items

Displays the list automatically whenever a structured variable or interface is entered and the "\" character is typed after it to show the list of possible structure fields. If disabled, the list can still be displayed by pressing the keyboard shortcut to open the AutoComplete window (usually Ctrl+Space, this can be modified in the Shortcuts section of the Preferences).

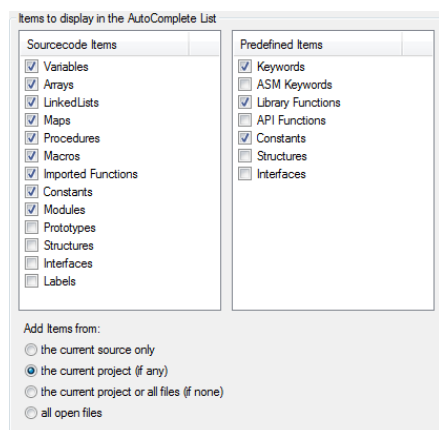
Automatically popup AutoComplete outside of Structures

Displays the list automatically when the current word is not a structure after a certain amount of characters has been typed, and a possible match in the list is found. If disabled, the list can still be displayed by pressing the assigned keyboard shortcut.

Characters needed before opening the list

Here you can specify how many characters the word must have minimum before the list is automatically displayed.

Editor - Auto complete - Displayed items



This shows a list of possible items that can be included with the possible matches in the AutoComplete list.

Source code Items

Items defined in the active source code, or other open sources (see below).

Predefined Items

Items that are predefined by PureBasic, such as the PureBasic keywords, functions or predefined constants.

Add Items from: the current source only

Source code items are only added from the active source code.

Add Items from: the current project (if any)

Source code items are added from the current project if there is one. The other source codes in the

project do not have to be currently open in the IDE for this.

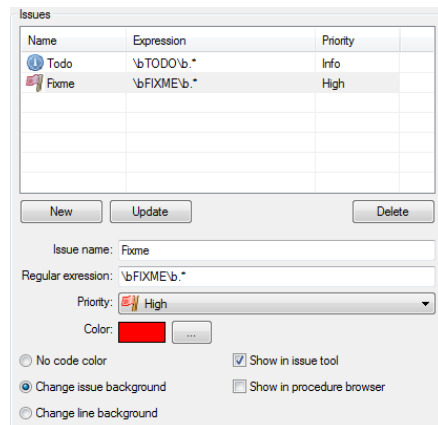
Add Items from: the current project or all files (if none)

Source code items are added from the current project. If the current source code does not belong to the open project then the items from all open source codes will be added.

Add Items from: all open files

Source code items are added from all currently open source codes.

Editor - Issues



Allows to configure the collection of 'issue' markers from comments in the source code. Issue markers can be displayed in the Issues or ProcedureBrowser tool, and they can be marked within the source code with a separate background color.

A definition for an issue consists of the following:

Issue name

A name for the type of issue.

Regular expression

A regular expression defining the pattern for the issue. This regular expression is applied to all comments in the source code. Each match of the expression is considered to match the issue type.

Two optional named groups (named "display" and "mark") can be included in the regex to control which part of the match is highlighted and which part is displayed in the tool.

Example: The default TODO pattern, but only display the part after the TODO in the tool

```
\bTODO\b(?:<display>.*)
```

Example: Match everything in {}, but only highlight the inner part (display the full match in the tool)

```
{(?:<mark>.*)}
```

Both named groups can also be used in a single regex for full control. If no groups with these names exist in the regex, the full match will be used.

Priority

Each issue type is assigned a priority. The priority can be used to order and filter the displayed issues in the issue tool.

Color

The color used to mark the issue in the source code (if enabled). The color will be used to mark the background of either only the issue text itself, or the entire code line depending on the coloring option.

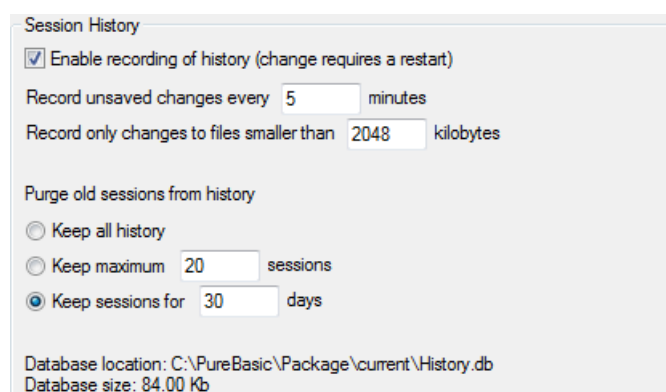
Show in issue tool

If enabled, any found issues of this type are listed in the issues tool. This option can be disabled to cause an issue to only be marked in the source code with a special background color if desired.

Show in procedure browser

If enabled, any found issues are shown as an entry in the procedure browser tool.

Editor - Session history



Allows to configure how the session history is recording changes.

Enable recording of history

Enable or disable the history session recording. When enabled, all the changes made to a file will be recorded in the background in a database. A session is created when the IDE launch, and is closed when the IDE quits. This is useful to rollback to a previous version of a file, or to find back a deleted or corrupted file. It's like a very powerful source backup tool, limited in time (by default one month of recording). It's not aimed to replace a real source versioning system like SVN or GIT. It's complementary to have finer change trace. The source code will be stored without encryption, so if you are working on sensitive source code, be sure to have this database file in a secure location, or disable this feature. It's possible to define the session history database location using an IDE command-line switch.

Record change every X minutes

Change the interval between each silent recording (when editing). A file will be automatically recorded when saving or closing it.

Record only changes to files smaller than X kilobytes

Change the maximum size (in kilobytes) of the files being recorded. This allow to exclude very big files which could make the database grow a lot.

Keep all history

Keep all the history, the database is never purged. It will always grows, so it should be watched.

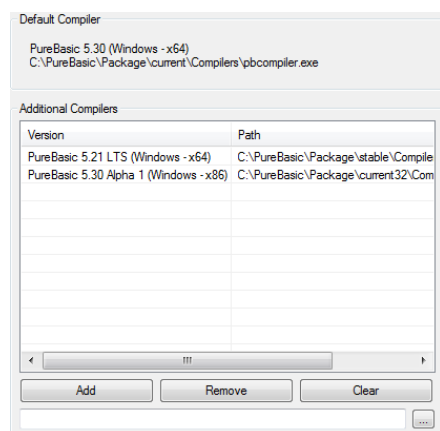
Keep maximum X sessions

After reaching the maximum number of sessions, the oldest session will be removed from the database.

Keep sessions for X days

After reaching the maximum number of days, the session will be removed from the database.

Compiler



This page allows to select additional compilers which should be available for compilation in the Compiler Options . This allows switching between different compilers of the same version (like the x86 and x64 compilers) or even switching between different versions easily.

Any PureBasic compiler starting from Version 4.10 can be added here. The target processor of the

selected compilers does not have to match that of the default compiler, as long as the target operating system is the same. The list displays the compiler version and path of the selected compilers. The information used by the IDE (for code highlighting, auto complete, structure viewer) always comes from the default compiler. The additional compilers are only used for compilation.

Compiler - Defaults

Default Settings for new Files

☒ Show Error Log ☐ #PB_Editor_CompileCount

☒ Enable Debugger ☐ #PB_Editor_BuildCount

☐ Enable Purifier ☐ #PB_Editor_CreateExecutable

☐ Enable inline ASM syntax coloring ☐ Enable OnError lines support

☒ Create unicode executable ☐ Create threadsafe executable

☒ Enable modern theme support (for Windows XP and above)

☐ Request Administrator mode for Windows Vista and above

☐ Request User mode for Windows Vista and above (no virtualisation)

☐ Create temporary executable in the source directory

Current directory:

Library Subsystem:

Executable format:

CPU Optimisation:

Sourcefile Text encoding:

Sourcefile Newline format:

This page allows setting the default compiler options that will be used when you create a new source code with the IDE.

For an explanation of the meaning of each field, see the Compiler Options .

Debugger

Choose Debugger Type

Choose Warning level

General Options

☒ Memorize debugger window positions

☒ Keep all debugger windows on top

☐ Bring debugger windows to front when one is focused

☒ Display Timestamp in Error log

☐ Stop execution at program start

☐ Stop execution before program end

☐ Kill Program after an Error

☒ Keep Error marks after program end

☐ Clear Errorlog on each run

☒ Display compilation errors in a window

Timeout for Debugger startup (ms):

Settings for the internal Debugger, or the Standalone Debugger. The command-line debugger is configured from the command-line only.

Debugger Type

Select the type of debugger you want to use when compiling from the IDE here.

Choose Warning level

Select the action that should be taken if the debugger issues a warning. The available options are:

Ignore Warnings: Warnings will be ignored without displaying anything.

Display Warnings: Warnings will be displayed in the error log and the source code line will be marked, but the program continues to run.

Treat Warnings as Errors: A warning will be treated like an error.

See Using the debugger for more information on debugger warnings and errors.

Memorize debugger window positions

The same as the "Memorize Window positions" for in the General section, but for all Debugger windows.

Keep all debugger windows on top

All debugger windows will be kept on top of all other windows, even from other applications.

Bring Debugger windows to front when one is focused

With this option set, focusing one window that belongs to the debugger of a file, all windows that belong to the same debugging session will be brought to the top.

Display Timestamp in error log

Includes the time of the event in the error log.

Stop execution at program start

Each program will be started in an already halted mode, giving you the opportunity to start moving step-by-step, right from the start of the program.

Stop execution before program end

Stops the program execution right before the executable would unload. This gives you a last chance to use the debugging tools to examine Variables or Memory before the program ends.

Kill Program after an Error

If a program encounters an error, it will be directly ended and all debugger windows closed. This gives the opportunity to directly modify the code again without an explicit "Kill Program", but there is no chance to examine the program after an error.

Keep Error marks after program end

Does not remove the lines marked with errors when the program ends. This gives the opportunity to still see where an error occurred while editing the code again.

The marks can be manually removed with the "Clear error marks" command in the "Error log" submenu of the debugger menu.

Clear error log on each run

Clears the log window when you execute a program. This ensures that the log does not grow too big (this option is also available with the command-line debugger selected).

Timeout for Debugger startup

Specifies the time in milliseconds how long the debugger will wait for a program to start up before giving up. This timeout prevents the debugger from locking up indefinitely if the executable cannot start for some reason.

Debugger - Individual Settings

This allows setting options for the individual debugger tools. The "Display Hex values" options switch between displaying Byte, Long and Word as decimal or hexadecimal values.

The screenshot shows the 'Debugger - Individual Settings' dialog box. It is organized into several sections with expandable/collapsible headers. The 'Debug Output' section is expanded, showing four checkboxes: 'Display Hex values' (unchecked), 'Add Timestamp' (unchecked), 'Display debug output in the error log' (unchecked), and 'Use a custom font:' (unchecked) with a 'Select Font' button. The 'Profiler' section is expanded, showing 'Start Profiler on program startup' (checked). The 'Asm Debugger' section is expanded, showing 'Display Registers as hex' (unchecked), 'Display Stack as hex' (unchecked), and 'Update Stack trace automatically' (checked). The 'Memory Viewer' section is expanded, showing 'Display Hex values' (unchecked) and 'Array view in one column only' (unchecked). The 'Variable Viewer' section is expanded, showing 'Display Hex values' (unchecked).

Debug Output Add Timestamp

Adds a timestamp to the output displayed from the Debug command.

Debug Output - Display debug output in the error log

With this option enabled, a Debug command in the code will not open the Debug Output window , but instead show the output in the error log .

Debug Output Use custom font

A custom font can be selected here for the debug output window . This allows to specify a smaller font for much output or a proportional one if this is desired.

Profiler - Start Profiler on program startup

Determines whether the profiler tool should start recording data when the program starts.

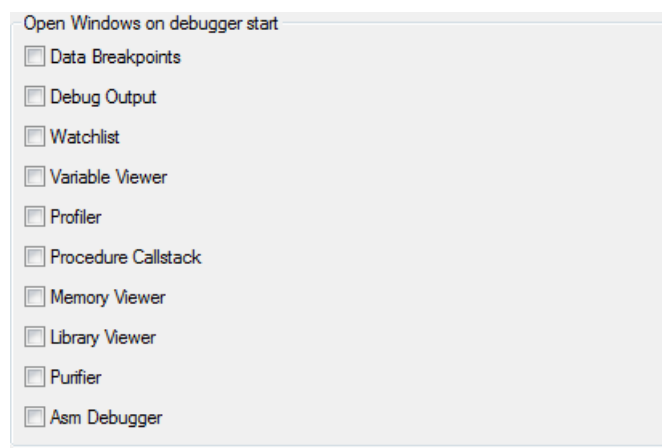
ASM Debugger Update Stack trace automatically

Updates the stack trace automatically on each step/stop you do. If disabled, a manual update button will be displayed in the ASM window.

Memory Viewer Array view in one column only

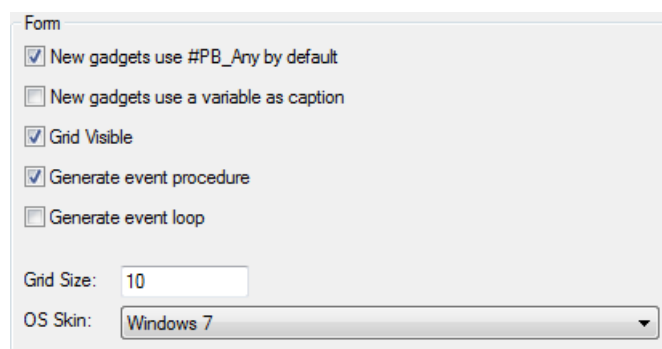
If the memory area is displayed in array view, this option selects whether it should be multi-column (with 16 bytes displayed in each column) or with only one column.

Debugger - Default Windows



The debugger tools you select on this page will automatically be opened each time you run a program with enabled debugger.

Form



Allows to customize the integrated form designer behavior.

New gadgets use #PB_Any by default

If enabled, new gadget creation will use #PB_Any instead of static enumeration numbering.

New gadgets use a variable as caption

If enabled, new gadget will use a variable instead of static text as caption. It can be useful to easily localize the form.

Grid visible

If enabled, the grid will be visible on the form designer, to ease gadget alignment.

Generate event procedure

If enabled, an event procedure will be automatically generated (and updated).

Generate event loop

If enabled, a basic event loop will be generated for the form.

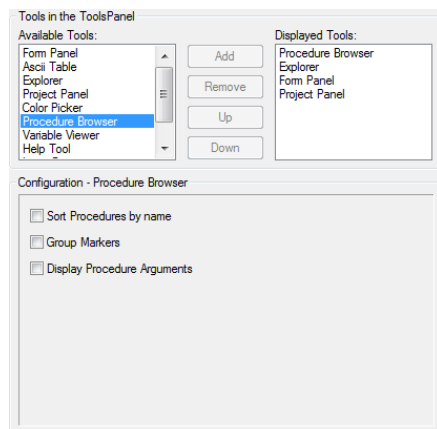
Grid size

Space between two grid points, in pixels.

OS skin

Skin to use for the form designer.

Tools Panel



This allows configuring the internal tools that can be displayed in the side panel. Each tool that is in the "Displayed Tools" list is displayed in the Panel on the side of the edit area. Each tool that is not listed there is accessible from the Tools menu as a separate window.

Put only those tools in the side panel that you use very frequently, and put the most frequently used first, as it will be the active one once you open the IDE.

By selecting a tool in either of the lists, you get more configuration options for that tool (if there are any) in the "Configuration" section below.

Here is an explanation of those tools that have special options:

Explorer

You can select between a Tree or List display of the file-system. You can also set whether the last displayed directory should be remembered, or if the Source Path should be the default directory when the IDE is started.

Procedure Browser

"Sort Procedures by Name" : sorts the list alphabetically (by default they are listed as they appear in the code).

"Group Markers" : groups the ";" markers together.

"Display Procedure Arguments" : Displays the full declaration of each procedure in the list.

Variable Viewer

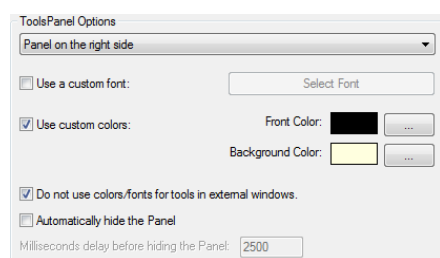
The "Display Elements from all open sources" option determines whether the list should only include items from this code, or from all open codes.

Furthermore you can select the type of items displayed in the Variable viewer.

Help Tool

"Open sidebar help on F1": specifies whether to open the help tool instead of the separate help viewer when F1 is pressed.

Tools panel - Options



Here you can customize the appearance of the Tools Panel a bit more. You can select the side on which it will be displayed, a Font for its text, as well as a foreground and background color for the displayed tools. The font and color options can be disabled to use the OS defaults instead.

Do not use colors/fonts for tools in external windows

If set, the color/font options only apply to the Tools displayed in the Panel, those that you open from the Tools menu will have the default colors.

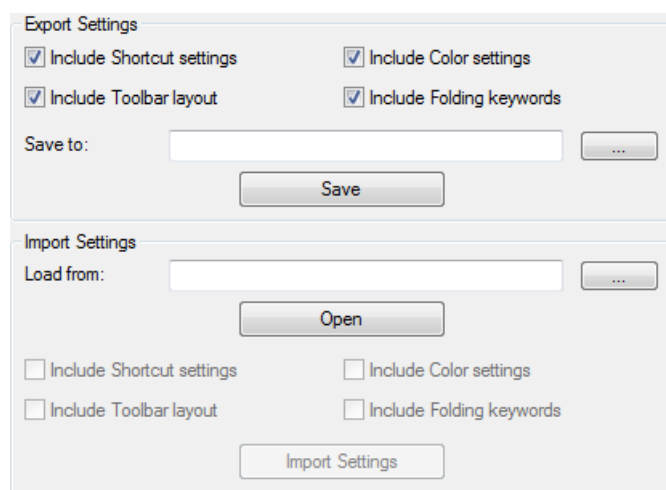
Automatically hide the Panel

To save space, the Panel will be hidden if the mouse is not over it. Moving the mouse to the side of the IDE will show it again.

Milliseconds delay before hiding the Panel

Sets a timeout in ms, after which the Panel is hidden if you leave it with the mouse.

Import/Export



This section allows you to export the layout settings of the IDE in a platform independent format, which allows you to import them again into the PureBasic IDE on another Operating System, or to share your options with other PB users.

To export your settings, select what types of settings you want to include, select a filename and press the "Save" button.

To import settings, select the filename and press "Open". You will then see the options that are included in this file as enabled checkboxes. After selecting what you want to import, click the "Import Settings" button.

For the new settings to take effect, you have to first click the apply button.

Note: You can import the style files from the jaPBe Editor, but only the color settings will be imported.

Chapter 19

Command-line options for the IDE

The PureBasic IDE allows you to modify the paths and files being used from the command-line. This allows you to create several shortcuts that start the IDE with different configurations for different users, or for different projects.

There are also options for compiling PureBasic projects directly from the command-line. Building a project from the command-line involves the same actions like at choosing the 'Build Target' or 'Build all Targets' from the compiler menu .

General options:

<code>/VERSION</code>	displays the IDE version and exits
<code>/HELP or /?</code> arguments	displays a description of the command-line arguments

Options for launching the IDE:

<code>/P <Preferences file></code>	loads/saves all the configuration to/from the given file
<code>/T <Templates file></code>	loads/saves the code templates from/to the given file
<code>/A <tools file></code>	loads/saves the configuration of the external tool from/to this file
<code>/S <Source path></code>	overwrites the "Source path" setting from the preferences
<code>/E <Explorer path></code>	starts the Explorer tool with the given path
<code>/L <Line number></code>	moves the cursor to the given line number in the last opened file
<code>/H <HistoryDatabase></code>	specify the file to use for the session history database
<code>/NOEXT</code>	disables the registering of the .pb extension in the registry
<code>/LOCAL</code>	puts all preferences in the PureBasic directory instead of the user profile location
<code>/PORTABLE</code>	the same as /LOCAL and /NOEXT combined

Options for building projects:

<code>/BUILD <file></code>	specifies the project file to build
<code>/TARGET <target></code>	specifies the target to build (the default is to build all targets)
<code>/QUIET</code>	hides all build messages except errors
<code>/READONLY</code>	does not update the project file after compiling (with new access time and build counters)

The default files for `/P` `/T` and `/A` are saved in the `%APPDATA%\PureBasic\` directory on the system.

The `/NOEXT` command is useful when you have several different PB versions at once (for testing of beta versions for example), but want the `.pb` extension to be associated with only one of them. The `/PORTABLE` command can be used to keep all configuration inside the local directory to easily copy PureBasic to different computers (or run it from USB sticks for example).
Example:

```
1 PureBasic.exe Example.pb /PORTABLE
```

You can also put the filenames of source files to load on the command-line. You can even specify wildcards for them (so with `"*.pb"` you can load a whole directory).

Part III

Language Reference

Chapter 20

Working with different number bases

(Note: These examples use the \wedge symbol to mean 'raised to the power of' - this is only for convenience in this document, PureBasic does not currently have a power-of operator! Use the PureBasic command `Pow()` from the "Math" Library instead.)

Introduction

A number base is a way of representing numbers, using a certain amount of possible symbols per digit. The most common you will know is base 10 (a.k.a. decimal), because there are 10 digits used (0 to 9), and is the one most humans can deal with most easily.

The purpose of this page is to explain different number bases and how you can work with them. The reason for this is that computers work in binary (base 2) and there are some cases where it is advantageous to understand this (for example, when using logical operators, bit masks, etc).

Overview of number bases

Decimal System

Think of a decimal number, and then think of it split into columns. We'll take the example number 1234. Splitting this into columns gives us:

1	2	3	4
---	---	---	---

Now think of what the headings for each column are. We know they are units, tens, hundreds and thousands, laid out like this:

1000	100	10	1
1	2	3	4

We can see that the number 1234 is made up out of

1*1000=	1000
+ 2* 100=	200
+ 3* 10=	30
+ 4* 1=	4
Total	=1234

If we also look at the column headings, you'll see that each time you move one column to the left we multiply by 10, which just so happens to be the number base. Each time you move one column to the right, you divide by 10. The headings of the columns can be called the weights, since to get the total

number we need to multiply the digits in each column by the weight. We can express the weights using indices. For example 10^2 means '10 raised to the power of two' or $1*10*10$ (=100). Similarly, 10^4 means $1*10*10*10*10$ (=10000). Notice the pattern that whatever the index value is, is how many times we multiply the number being raised. 10^0 means 1 (since we multiply by 10 no times). Using negative numbers shows that we need to divide, so for example 10^{-2} means $1/10/10$ (=0.01). The index values make more sense when we give each column a number - you'll often see things like 'bit 0' which really means 'binary digit in column 0'.

In this example, ^ means raised to the power of, so 10^2 means 10 raised to the power of 2.

Column number	3	2	1	0
Weight (index type)	10^3	10^2	10^1	10^0
Weight (actual value)	1000	100	10	1
Example number (1234)	1	2	3	4

A few sections ago we showed how to convert the number 1234 into its decimal equivalent. A bit pointless, since it was already in decimal but the general method can be shown from it - so this is how to convert from any number to its decimal value:

B = Value of number base

1) Separate the number in whatever base you have into it's columns.

For

example, if we had the value 'abcde' in our fictional number base 'B',

the columns would be: a b c d e

2) Multiply each symbol by the weight for that column (the weight being

calculated by 'B' raised to the power of the column number):

$a * B^4 = a * B * B * B * B$

$b * B^3 = b * B * B * B$

$c * B^2 = c * B * B$

$d * B^1 = d * B$

$e * B^0 = e$

3) Calculate the total of all those values. By writing all those values in their

decimal equivalent during the calculations, it becomes far easier to see the

result and do the calculation (if we are converting into decimal).

Converting in the opposite direction (from decimal to the number base 'B') is done using division instead of multiplication:

1) Start with the decimal number you want to convert (e.g. 1234).

2) Divide by the target number base ('B') and keep note of the result and the remainder.

3) Divide the result of (2) by the target number base ('B') and keep note of the result and the remainder.

4) Keep dividing like this until you end up with a result of 0.

5) Your number in the target number base is the remainders written in the order

most recently calculated to least recent. For example, your number would be

the remainders of the steps in this order 432.

More specific examples will be given in the sections about the specific number bases.

Binary System

Everything in a computer is stored in binary (base 2, so giving symbols of '0' or '1') but working with binary numbers follows the same rules as decimal. Each symbol in a binary number is called a bit, short for binary digit. Generally, you will be working with bytes (8-bit), words (16-bit) or longwords (32-bit) as these are the default sizes of PureBasic's built in types. The weights for a byte are shown:

(^ means 'raised to the power of', number base is 2 for binary)								
Bit/column number	7	6	5	4	3	2	1	0
Weight (index)	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Weight (actual value)	128	64	32	16	8	4	2	1

So, for example, if we had the number 00110011 (Base 2), we could work out the value of it like this:

```
0 * 128
+ 0 * 64
+ 1 * 32
+ 1 * 16
+ 0 * 8
+ 0 * 4
+ 1 * 2
+ 1 * 1
= 51
```

An example of converting back would be if we wanted to write the value 69 in binary. We would do it like this:

```
69 / 2 = 34 r 1      ^
34 / 2 = 17 r 0      /|\
17 / 2 = 8 r 1       |
8 / 2 = 4 r 0        |      Read remainders in this direction
4 / 2 = 2 r 0        |
2 / 2 = 1 r 0        |
1 / 2 = 0 r 1        |
(Stop here since the result of the last divide was 0)
```

Read the remainders backwards to get the value in binary = 1000101

Another thing to consider when working with binary numbers is the representation of negative numbers. In everyday use, we would simply put a negative symbol in front of the decimal number. We cannot do this in binary, but there is a way (after all, PureBasic works mainly with signed numbers, and so must be able to handle negative numbers). This method is called 'twos complement' and apart from all the good features this method has (and won't be explained here, to save some confusion) the simplest way to think of it is the weight of the most significant bit (the MSb is the bit number with the highest weight, in the case of a byte, it would be bit 7) is actually a negative value. So for a two's complement system, the bit weights are changed to:

(^ means 'raised to the power of', number base is 2 for binary)								
Bit/column number	7	6	5	4	3	2	1	0
Weight (index)	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Weight (actual value)	-128	64	32	16	8	4	2	1

and you would do the conversion from binary to decimal in exactly the same way as above, but using the new set of weights. So, for example, the number 10000000 (Base 2) is -128, and 10101010 (Base 2) is -86.

To convert from a positive binary number to a negative binary number and vice versa, you invert all the bits and then add 1. For example, 00100010 would be made negative by inverting -> 11011101 and adding 1 -> 11011110.

This makes converting from decimal to binary easier, as you can convert negative decimal numbers as their positive equivalents (using the method shown above) and then make the binary number negative at the end.

Binary numbers are written in PureBasic with a percent symbol in front of them, and obviously all the bits in the number must be a '0' or a '1'. For example, you could use the value %110011 in PureBasic to mean 51. Note that you do not need to put in the leading '0's (that number would really be %00110011) but it might help the readability of your source if you put in the full amount of bits.

Hexadecimal System

Hexadecimal (for base 16, symbols '0'-'9' then 'A'-'F') is a number base which is most commonly used when dealing with computers, as it is probably the easiest of the non-base 10 number bases for humans to understand, and you do not end up with long strings of symbols for your numbers (as you get when working in binary).

Hexadecimal mathematics follows the same rules as with decimal, although you now have 16 symbols per column until you require a carry/borrow. Conversion between hexadecimal and decimal follows the same patterns as between binary and decimal, except the weights are in multiples of 16 and divides are done by 16 instead of 2:

Column number	3	2	1	0
Weight (index)	16^3	16^2	16^1	16^0
Weight (actual value)	4096	256	16	1

Converting the hexadecimal value BEEF (Base 16) to decimal would be done like this:

```

B * 4096 = 11 * 4096
+ E * 256 = 14 * 256
+ E * 16 = 14 * 16
+ F * 1 = 15 * 1
=
48879

```

And converting the value 666 to hexadecimal would be done like this:

```

666 / 16 = 41 r 10 ^
41 / 16 = 2 r 9  /\
remembering to convert
2 / 16 = 0 r 2  |  to hex digits where required
(Stop here, as result is 0)
Hexadecimal value of 666 is 29A

```

The really good thing about hexadecimal is that it also allows you to convert to binary very easily. Each hexadecimal digit represents 4 bits, so to convert between hexadecimal and binary, you just need to convert each hex digit to 4 bits or every 4 bits to one hex digit (and remembering that 4 is an equal divisor of all the common lengths of binary numbers in a CPU). For example:

Hex number	5	9	D	F	4E
Binary value	0101	1001	1101	1111	01001110

When using hexadecimal values in PureBasic, you put a dollar sign in front of the number, for example \$BEEF.

Chapter 21

Break : Continue

Syntax

`Break [Level]`

Description

`Break` provides the ability to exit during any iteration, for the following loops: Repeat , For , ForEach and While . The optional 'level' parameter specifies the number of loops to exit from, given several nested loops. If no parameter is given, `Break` exits from the current loop.

Example: Simple loop

```
1  For k=0 To 10
2      If k=5
3          Break ; Will exit directly from the For/Next loop
4      EndIf
5      Debug k
6  Next
```

Example: Nested loops

```
1  For k=0 To 10
2      Counter = 0
3      Repeat
4          If k=5
5              Break 2 ; Will exit directly from the Repeat/Until and For/Next
loops
6          EndIf
7          Counter+1
8      Until Counter > 1
9      Debug k
10 Next
```

Syntax

`Continue`

Description

`Continue` provides the ability to skip straight to the end of the current iteration (bypassing any code in between) in the following loops: Repeat , For , ForEach , and While .

Example

```
1  For k=0 To 10
2      If k=5
3          Continue ; Will skip the 'Debug 5' and go directly to the next
           iteration
4      EndIf
5      Debug k
6  Next
```

Chapter 22

Using the command line compiler

Introduction

The command line compiler is located in the subdirectory 'compilers' from the PureBasic folder. The easier way to access it is to add this directory in the PATH variable, which will give access to all the commands of this directory at all times.

There is two compilers for PureBasic:

- 'pbcompiler' which is generating assembly code (x86 and x64)
- 'pbcompilerc' (or 'pbcompiler' on platform which doesn't support the assembly backend) which is generating C code

The C backend compiler is available on all supported platforms, while the ASM backend compiler is only supported on Windows (x86, x64), Linux (x86, x64) and OS X (x64).

The backend C compiler does not allow entering assembly code.

On Windows a 'PureBasic-CLI.cmd' file is available in the PureBasic root directly to easily open a commandline prompt to use the PureBasic commandline tools.

Cross-platform command switches

Note: All switches name which begin with '/' are only valid on Windows (ie: '/DEBUGGER'). If you need to write cross-plaform commandline, you can use the '-' or '-' format (ie: '-d' or '-debugger').

-h, -help, /?: displays a quick help about the compiler.

-d, -debugger, /DEBUGGER: enables the debugger support.

-cl, -console, /CONSOLE: output file in Console format. Default format on Windows is Win32.

-pf, -purifier, /PURIFIER: enables the purifier. The debugger needs to be activated to have any effect.

-o, -output, /OUTPUT "filename": creates a standalone executable or DLL specified by the filename at the desired path location. On MacOS X it's possible to create an application bundle by adding '.app' to the name of the executable. This way the whole directory structure will be automatically created.

-r, -resident, /RESIDENT "filename": creates a resident file specified by the filename.

-i, -import, /IMPORT "filename": creates an import file to the given filename. Only one

Import/EndImport block is allowed in the file. The imported functions will be loaded automatically for every PureBasic projects.

-l or -linenumbering, /LINENUMBERING: adds lines and files information to the executable, which can slow it considerably. This allows the OnError library to report the file and line-number of an error.

-q, -quiet, /QUIET: disables all unnecessary text output, very useful when using another editor.

-sb, -standby, /STANDBY: loads the compiler in memory and wait for external commands (editor, scripts...). More information about using this flag is available in the file 'CompilerInterface.txt' from the PureBasic 'SDK' directory.

-ir, -ignoreresident, /IGNORERESIDENT "filename": doesn't load the specified resident file when the compiler starts. It's mostly useful when updating a resident which already exists, so it won't load it.

-co, -constant, /CONSTANT name=value: creates the specified constant with the given expression.

Example: 'pbcompiler test.pb /CONSTANT MyConstant=10'. The constant `#MyConstant` will be

created on the fly with a value of 10 before the program gets compiled.

-t, -thread, /THREAD: uses thread safe runtime for strings and general routines.

-s, -subsystem, /SUBSYSTEM "name": uses the specific subsystem to replace a set of internal functions. The subsystem name is not case-sensitive. For more information, see subsystems .

-k, -check, /CHECK: checks the syntax only, doesn't create or launch the executable.

-z, -optimizer, /OPTIMIZER: enable the code optimizer.

-c, -commented, /COMMENTED: creates a commented 'purebasic.asm' or 'purebasic.c' output file when creating an executable. This file can be re-assembled with the '-reasm' option later when the needed modifications have been made. This option is for advanced programmers only.

-ra, -reasm, /REASM: re-assembles the 'purebasic.asm' or 'purebasic.c' file into an executable. This allow to use the '-commented' option, modify the generated output and creates the executable again. This option is for advanced programmers only.

-pp, -preprocess, /PREPROCESS "filename": preprocess the source code and write the output in the specified "Filename". The processed file is a single file with all macro expanded, compiler directive handled and multiline resolved. This allows an easier parsing of a PureBasic source file, as all is expanded and is available in a flat file format. No comments are included by default, but the flag '-commented' can be used to have all untouched original source as comments and allow comments processing. The preprocessed file can be recompiled as any other PureBasic source file to create the final executable.

-ls, -liststructures, /LISTSTRUCTURES: creates a file with all the built-in structures known by the compiler. The filename can be specified with '-output'.

-lf, -listfunction, /LISTFUNCTIONS: creates a file with all the built-in purebasic commands known by the compiler. The filename can be specified with '-output'.

-li, -listinterfaces, /LISTINTERFACES: creates a file with all the built-in interfaces known by the compiler. The filename can be specified with '-output'.

-qs, -querystructure, /QUERYSTRUCTURE "name": creates a file with the structure definition. The filename can be specified with '-output'.

-g, -language, /LANGUAGE "language": uses the specified language for the compiler error messages.

-v, -version, /VERSION: displays the compiler version.

Examples:

```
CLI> pbcompiler sourcecode.pb
```

The compiler will compile the source code and execute it.

```
CLI> pbcompiler sourcecode.pb --debugger
```

The compiler will compile the source code and execute it with debugger.

Windows specific command switches

/ICON "IconName.ico" : add the specified icon to the executable.

/DLL: output file is a DLL .

/XP: Add the Windows theme support to the executable.

/MMX, /3DNOW, /SSE or /SSE2: Creates a processor specific executable. This means if a processor specific routine is available it will be used for this executable. Therefore, the executable will run correctly only on computer with this kind of CPU.

/DYNAMICCPU: Creates a executable containing all the available processor specific routines. When the program is started, it looks for the processor type and then select the more appropriates routines to use. This makes a bigger executable, but result in the fastest possible program.

/RESOURCE "Filename": Add a Windows resource file (.rc) to the created executable or DLL. This should be not a compiled resource file, but an ascii file with the directives in it. It's possible to specify only one resource, but this file can include other resource script if needed.

/LINKER "ResponseFile": Specify a file which contains commands to be passed directly to the linker. On Windows, PureBasic use the PellesC linker (polink), more information about the possible options can be found in the related documentation.

The following two compiler options are needed for creating programs running on Microsoft Vista OS or above (Windows 7/8/10). They are both options for the included manifest, so they are ignored on older windows versions. With none of these switches, the exe will run as a normal user as well, but with virtualization turned on (i.e. registry and file redirection). It is recommended to use the /USER switch

to turn of virtualization for all programs that comply to the standard user privileges, as it is only intended for compatibility for older programs. These options are also available in the IDE compiler options .

/ADMINISTRATOR: Will cause the program to request admin privileges at start. The program will not run without it. This option is needed. Note: You can also debug programs with this flag, but only with the standalone gui debugger (as it must run in elevated mode as well).

/USER: The program will run as the user who started it. Virtualization for the exe is turned off.

/DPIAWARE: Enable DPI support to the executable.

/DLLPROTECTION: Enable DLL preloading protection to the executable. It prevents that system DLLs are first searched for in the program directory instead of in the System32 directory of the Windows operating system.

/UCRT: Use the dynamic universal CRT when creating the executable or DLL. It is recommended when creating an executable or a DLL which doesn't need to support older Windows version below Windows 10. The resulting executable or DLL will be smaller and will get automatic security fixes from the CRT when Microsoft updates it. This option is always enabled on PureBasic for Windows arm64.

Examples:

```
CLI> pbcompiler "C:\Project\Source\DLLSource.pb" /EXE  
      "C:\Project\project.dll" /DLL
```

The compiler will compile the source code (here with full path) and create the DLL "project.dll" in the given directory.

Linux specific command switches

-so or -sharedobject "filename": Create a dynamic library (shared object).

-mmx, -3dnow, -sse or -sse2: Creates a processor specific executable. This means if a processor specific routine is available it will be used for this executable. Therefore, the executable will run correctly only on computer with this kind of CPU.

-dc or -dynamiccpu: Creates a executable containing all the available processor specific routines. When the program is started, it looks for the processor type and then select the more appropriate routines to use. This makes a bigger executable, but result in the fastest possible program.

OS X specific command switches

-n or -icon "filename.icns" : add the specified icon to the executable.

-dl or -dylib "filename": Create a dynamic library (dylib object).

-f or -front: put the launched process to front.

-ibp or -ignorebundlepath: don't use the bundle path as current directory.

Chapter 23

Compiler Directives

Syntax

```
CompilerIf <constant expression>
...
[CompilerElseIf]
...
[CompilerElse]
...
CompilerEndIf
```

Description

If the <constant expression> result is true, the code inside the `CompilerIf` will be compiled, else it will be totally ignored. It's useful when building multi-OSes programs to customize some programs part by using OS specific functions. The `And` and `Or` Keywords can be used in <constant expression> to combine multiple conditions.

Example

```
1  CompilerIf #PB_Compiler_OS = #PB_OS_Linux And #PB_Compiler_Processor
   = #PB_Processor_x86
2      ; some Linux and x86 specific code.
3  CompilerEndIf
```

Syntax

```
CompilerSelect <numeric constant>
  CompilerCase <numeric constant>
  ...
  [CompilerDefault]
  ...
CompilerEndSelect
```

Description

Works like a regular `Select : EndSelect` except that only one numeric value is allowed per case. It will tell the compiler which code should be compiled. It's useful when building multi-OSes programs to customize some programs part by using OS specific functions.

Example

```
1  CompilerSelect  #PB_Compiler_OS
2      CompilerCase  #PB_OS_MacOS
3          ; some Mac OS X specific code
4      CompilerCase  #PB_OS_Linux
5          ; some Linux specific code
6  CompilerEndSelect
```

Syntax

```
CompilerError <string constant>
CompilerWarning <string constant>
```

Description

Generates an error or a warning, as if it was a syntax error and display the associated message. It can be useful when doing specialized routines, or to inform a source code is not available on an particular OS. Note: `CompilerWarning` displays warnings, but the compilation process continues, while the command `CompilerError` stops the compilation process.

Example: Generates an error

```
1  CompilerIf  #PB_Compiler_OS = #PB_OS_Linux
2      CompilerError  "Linux isn't supported, sorry."
3  CompilerElse
4      CompilerError  "OS supported, you can now comment me."
5  CompilerEndIf
```

Example: Generates a warning

```
1  CompilerIf  #PB_Compiler_OS = #PB_OS_Linux
2      CompilerWarning  "Linux isn't supported, sorry."
3  CompilerElse
4      CompilerWarning  "OS supported, you can now comment me."
5  CompilerEndIf
```

Syntax

```
EnableExplicit
DisableExplicit
```

Description

Enables or disables the explicit mode. When enabled, all the variables which are not explicitly declared with `Define` , `Global` , `Protected` or `Static` are not accepted and the compiler will raise an error. It can help to catch typo bugs.

Example

```
1 EnableExplicit
2
3 Define a
4
5 a = 20 ; Ok, as declared with 'Define'
6 b = 10 ; Will raise an error here
```

Syntax

```
EnableASM
DisableASM
```

Description

Enables or disables the inline assembler. When enabled, all the assembler keywords are available directly in the source code, see the inline assembler section for more information.

Example

```
1 ; x86 assembly example
2 ;
3 Test = 10
4
5 EnableASM
6 MOV dword [v_Test],20
7 DisableASM
8
9 Debug Test ; Will be 20
```

Syntax

```
DisablePureLibrary <LibraryName>
```

Description

Disables the specified PureLibrary. All its associated commands won't be available when compiling this program. This is especially useful when creating a PureLibrary .

Example

```
1 DisablePureLibrary Engine3D
2
3 InitEngine3D() ; Error, command not found
```

Reserved Constants

The PureBasic compiler has several reserved constants which can be useful to the programmer:

```

#PB_Compiler_OS : Determines on which OS the compiler is currently
running. It can be one of the following values:
    #PB_OS_Windows : The compiler is running on Windows
    #PB_OS_Linux    : The compiler is running on Linux
    #PB_OS_MacOS    : The compiler is running on Mac OS X

#PB_Compiler_Processor : Determines the processor type for which the
program is created. It can be one of the following:
    #PB_Processor_x86      : x86 processor architecture (also called
IA-32 or x86-32)
    #PB_Processor_x64      : x86-64 processor architecture (also called
x64, AMD64 or Intel64)
    #PB_Processor_arm32    : arm32 processor architecture
    #PB_Processor_arm64    : arm64 processor architecture (also called
M1 on Apple computers)

#PB_Compiler_Backend : Determines which kind of compiler is currently
used. It can be one of the following:
    #PB_Backend_Asm        : The compiler generating assembly code is used.
    #PB_Backend_C          : The compiler generating C code is used.

#PB_Compiler_ExecutableFormat : Determines executable format. It can
be one of the following:
    #PB_Compiler_Executable : Regular executable
    #PB_Compiler_Console    : Console executable (have an effect only
on Windows, other act like a regular executable)
    #PB_Compiler_DLL        : Shared DLL (dynlib on MacOS X and shared
object on Linux)

#PB_Compiler_Date        : Current date, at the compile time, in the
PureBasic date
format (local time).
#PB_Compiler_File        : Full path and name of the file being
compiled, useful for debug
purpose.
#PB_Compiler_FilePath    : Full path of the file being compiled, useful
for debug
purpose.
#PB_Compiler_Filename    : Filename (without path) of the file being
compiled, useful for debug
purpose.
#PB_Compiler_Line        : Line number of the file being compiled,
useful for debug
purpose.
#PB_Compiler_Procedure    : Current procedure name, if the line is inside
a procedure
.
#PB_Compiler_Module      : Current module name, if the line is inside a
module
.
#PB_Compiler_Version      : Compiler version, in integer format in the
form '420' for 4.20.
#PB_Compiler_Home        : Full path of the PureBasic directory, can be
useful to locate include files

#PB_Compiler_Debugger    : Set to 1 if the runtime debugger
is enabled, set to 0 else. When an executable
is created, the debugger is always disabled
(this constant will be 0).

```



```

#PB_Compiler_Thread      : Set to 1 if the executable is compiled in
    thread safe mode, set to 0 else.
#PB_Compiler_Unicode     : Set to 1 if the executable is compiled in
    unicode
mode, set to 0 else.
#PB_Compiler_LineNumbering : Set to 1 if the executable is compiled
    with OnError line numbering
support, set to 0 else.
#PB_Compiler_InlineAssembly: Set to 1 if the executable is compiled
    with inline assembly
support, set to 0 else.
#PB_Compiler_EnableExplicit: Set to 1 if the executable is compiled
    with enable explicit support, set to 0 else.
#PB_Compiler_IsMainFile   : Set to 1 if the file being compiled is
    the main file, set to 0 else.
#PB_Compiler_IsIncludeFile : Set to 1 if the file being compiled has
    been included by another file, set to 0 else.
#PB_Compiler_32Bit        : Set to 1 if the compiler generates 32-bit
    code, set to 0 else.
#PB_Compiler_64Bit        : Set to 1 if the compiler generates 64-bit
    code, set to 0 else.
#PB_Compiler_Optimizer     : Set to 1 if the compiler generates optimized
    code, set to 0 else.
#PB_Compiler_DPIAware     : Set to 1 if the compiler generates a DPI
    aware executable, set to 0 else.

```

Chapter 24

Compiler Functions

Syntax

```
Size = SizeOf(Type)
```

Description

[SizeOf](#) can be used to find the size of any complex Structure , built-in type (word, float, etc.), Interface or even ReferenceLink "variables" "variables" (Structures with same name as variable take precedence). This can be useful in many areas such as calculating memory requirements for operations, using API commands, etc.

As a compiler function, [SizeOf\(x\)](#) is assigned to a constant and does not require assignment to another variable if inside a loop or often called procedure.

As a compile time function [SizeOf](#) doesn't work with runtime Array, List or Map. [ArraySize\(\)](#) , [ListSize\(\)](#) or [MapSize\(\)](#) can be used instead.

Note: A Character (.c) variable is unicode and uses 2 bytes. An Ascii variable (.a) is Ascii and uses 1 byte.

Parameters

Type The type of the object.

Return value

The size of the object in memory, in bytes.

Example: 1

```
1 char.c='!'
2 Debug SizeOf(char); display 2
3
4 ascii.a='!'
5 Debug SizeOf(ascii); display 1
```

Example: 2

```

1  Structure Person
2      Name.s
3      ForName.s
4      Age.w
5  EndStructure
6
7  Debug "The size of my friend is "+Str(Sizeof(Person))+" bytes" ; will
      be 10 on 32-bit compiler as a string pointer is 4 bytes in memory
8
      ; will
      be 18 on 64-bit compiler as a string pointer is 8 bytes in memory
9
10 John.Person\Name = "John"
11
12 Debug SizeOf(John) ; will be the same

```

Syntax

```

Index = OffsetOf(Structure\Field)
Index = OffsetOf(Interface\Function())

```

Description

OffsetOf can be used to find out the address offset of a Structure field or the address offset of an Interface function. When used with an Interface , the function index is the memory offset, so it will be `IndexOfTheFunction*SizeOf(Integer)`.

Parameters

Structure\Field or **Interface\Function()** The field of the structure or function of the interface.

Return value

Returns the index of the field or function, zero otherwise.

Example

```

1  Structure Person
2      Name.s
3      ForName.s
4      Age.w
5  EndStructure
6
7  Debug OffsetOf(Person\Age) ; will be 8 on 32-bit compiler as a
      string pointer is 4 bytes in memory
8
      ; will be 16 on 64-bit compiler as a
      string pointer is 8 bytes in memory
9
10 Interface ITest
11     Create()
12     Destroy(Flags)
13 EndInterface
14

```

```

15 | Debug OffsetOf(ITest\Destroy()) ; will be 4 on 32-bit compiler as a
    | pointer is 4 bytes in memory
16 |                                ; will be 8 on 64-bit compiler as a
    | pointer is 8 bytes in memory

```

Syntax

`Type = TypeOf(Object)`

Description

`TypeOf` can be used to find out the type of a variable , or a structure field .

Parameters

Object The object to use.

Return value

The type of the object.

The type can be one of the following values:

```

#PB_Byte
#PB_Word
#PB_Long
#PB_String
#PB_Structure
#PB_Float
#PB_Character
#PB_Double
#PB_Quad
#PB_List
#PB_Array
#PB_Integer
#PB_Map
#PB_Ascii
#PB_Unicode
#PB_Interface

```

Example

```

1 | Structure Person
2 |     Name.s
3 |     ForName.s
4 |     Age.w
5 | EndStructure
6 |
7 | If TypeOf(Person\Age) = #PB_Word
8 |     Debug "Age is a 'Word'"
9 | EndIf
10 |
11 | Surface.f
12 | If TypeOf(Surface) = #PB_Float

```

```
13     Debug "Surface is a 'Float'"
14 EndIf
```

Syntax

`Result = Subsystem(<constant string expression>)`

Description

`Subsystem` can be used to find out if a subsystem is in use for the program being compiled. The specified subsystem name is not case-sensitive.

Parameters

constant string expression The subsystem name.

Windows: DirectX9, DirectX11

Linux : Gtk2, Qt

MacOS X: None

Return value

Returns nonzero if the subsystem is used, zero otherwise.

Example

```
1 CompilerIf Subsystem("OpenGL")
2     Debug "Compiling with the OpenGL subsystem"
3 CompilerEndIf
```

Syntax

`Result = Defined(Name, Type)`

Description

`Defined` checks if a particular object of a code source like structure , interface , variables etc. is already defined or not.

Parameters

Name The name of the object. The 'Name' parameter has to be specified without any extra decoration (ie: without the '#' for a constant , without '()' for an array , a list or a map).

Type The 'Type' parameter can be one of the following values:

```
#PB_Constant
#PB_Variable
#PB_Array
#PB_List
#PB_Map
```

```

#PB_Structure
#PB_Interface
#PB_Procedure
#PB_Function
#PB_OSFunction
#PB_Label
#PB_Prototype
#PB_Module
#PB_Enumeration

```

Return value

Returns nonzero if the object is defined, zero otherwise.

Example

```

1  #PureConstant = 10
2
3  CompilerIf Defined(PureConstant, #PB_Constant)
4      Debug "Constant 'PureConstant' is declared"
5  CompilerEndIf
6
7  Test = 25
8
9  CompilerIf Defined(Test, #PB_Variable)
10     Debug "Variable 'Test' is declared"
11 CompilerEndIf

```

Syntax

```
InitializeStructure(*Pointer, Structure)
```

Description

[InitializeStructure](#) initialize the specified structured memory area. It initializes structure members of type Array, List and Map, other members are not affected (.s, .l, .i etc).

Parameters

***Pointer** The memory address to use.

Structure 'Structure' is the name of the structure which should be used to perform the initialization. There is no internal check to ensures the structure match the memory area. Warning: multiple calls to [InitializeStructure](#) create a memory leak because the old members are not freed ([ClearStructure](#) has to be called before calling [InitializeStructure](#) once more). This function is for advanced users and should be used with care. To allocate dynamic structure, use [AllocateStructure](#)() ().

Return value

None.

Example

```
1  Structure People
2      Name$
3      Age.l
4      List Friends.s()
5  EndStructure
6
7  *Student.People = AllocateMemory(SizeOf(People))
8  InitializeStructure(*Student, People)
9
10 ; Now the list is ready to use
11 ;
12 AddElement(*Student\Friends())
13 *Student\Friends() = "John"
14
15 AddElement(*Student\Friends())
16 *Student\Friends() = "Yann"
17
18 ; Print out the list content
19 ;
20 ForEach *Student\Friends()
21     Debug *Student\Friends()
22 Next
```

Syntax

Result = CompareStructure(*Pointer1, *Pointer2, Structure [, Flags])

Description

`CompareStructure` compares the memory of two structures for equality. The comparison is recursively applied also to child elements such as arrays , lists , and maps .

Parameters

***Pointer1** The memory address of the first structured variable to be tested.

***Pointer2** The memory address of the second structured variable to be tested.

Structure The structure used.

Flags (optional) It can have one of the following values:

```
#PB_String_CaseSensitive : String comparison is case sensitive
                          (a=a). (default)
#PB_String_NoCase       : String comparison is case
                          insensitive(a=A).
#PB_Memory_FollowPointers: If a structure element is a pointer
                          that is not 0, recursively compare the pointer target.
                          The default is to compare only the
                          pointer value itself.
```

Remarks

Caution: The `#PB_Memory_FollowPointers` option is for advanced users and requires special care to avoid crashes. If this option is used then all pointer values must point to valid and initialized memory or have the value 0. It is also not allowed to have loops in the pointed elements (a chain of pointers that refers back to itself).

Return value

Returns nonzero if the structures are the same or zero if they differ.

Example

```
1  Structure People
2      Name$
3      LastName$
4      Map Friends$()
5      Age.l
6  EndStructure
7
8  StudentA.People\Name$ = "Paul "
9  StudentA\LastName$ = "Morito"
10 StudentA\Friends$("Tom") = "Jones "
11 StudentA\Friends$("Jim") = "Doe "
12
13 StudentB.People\Name$ = "Paul "
14 StudentB\LastName$ = "Morito"
15 StudentB\Friends$("Tom") = "Jones "
16 StudentB\Friends$("Jim") = "Doe "
17
18 StudentC.People\Name$ = "Paul "
19 StudentC\LastName$ = "Morito"
20 StudentC\Friends$("Tom") = "Hanks " ; Not the same as in StudentA
21 StudentC\Friends$("Jim") = "Doe "
22
23 Debug CompareStructure(@StudentA, @StudentB, People) ; Equal
24 Debug CompareStructure(@StudentA, @StudentC, People) ; Not equal
```

Syntax

`CopyStructure(*Source, *Destination, Structure)`

Description

`CopyStructure` copy the memory of a structured memory area to another. This is useful when dealing with dynamic allocations, through pointers . Every fields will be duplicated, even array , list , and map . The destination structure will be automatically cleared before doing the copy, it's not needed to call `ClearStructure` before `CopyStructure`. Warning: the destination should be a valid structure memory area, or a cleared memory area. If the memory area is not cleared, it could crash, as random values will be used by the clear routine. There is no internal check to ensures that the structure match the two memory area. This function is for advanced users and should be used with care.

Parameters

***Source** The memory address containing the structure to copy.

***Destination** The memory address of the copy.

Structure The name of the structure that should be used to perform the copy.

Return value

None.

Example

```
1  Structure People
2      Name$
3      LastName$
4      Map Friends$()
5      Age.1
6  EndStructure
7
8  Student.People\Name$ = "Paul"
9  Student\LastName$ = "Morito"
10 Student\Friends$("Tom") = "Jones"
11 Student\Friends$("Jim") = "Doe"
12
13 CopyStructure(@Student, @StudentCopy.People, People)
14
15 Debug StudentCopy\Name$
16 Debug StudentCopy\LastName$
17 Debug StudentCopy\Friends$("Tom")
18 Debug StudentCopy\Friends$("Jim")
```

Syntax

`ClearStructure(*Pointer, Structure)`

Description

`ClearStructure` clears a structured memory area. This is useful when the structure contains strings, array, list or map which have been allocated internally by PureBasic. This function is for advanced users and should be used with care.

Parameters

***Pointer** The memory address containing the structure to be erased.

Structure The name of the structure which should be used to perform the clearing. All the fields will be set to zero, even native type like long, integer etc. There is no internal check to ensures the structure match the memory area.

Return value

None.

Example

```
1  Structure People
2      Name$
3      LastName$
4      Age.l
5  EndStructure
6
7  Student.People\Name$ = "Paul"
8  Student\LastName$ = "Morito"
9  Student\Age = 10
10
11 ClearStructure(@Student, People)
12
13 ; Will print empty strings as the whole structure has been cleared.
14 ; All other fields have been reset to zero.
15 ;
16 Debug Student\Name$
17 Debug Student\LastName$
18 Debug Student\Age
```

Syntax

`ResetStructure(*Pointer, Structure)`

Description

`ResetStructure` clears a structured memory area and initialize it to be ready to use. This is useful when the structure contains strings, array, list or map which have been allocated internally by PureBasic. This function is for advanced users and should be used with care.

Parameters

***Pointer** The memory address containing the structure to be reinitialized.

Structure The name of the structure which should be used to perform the clearing.

Return value

None.

Example

```
1  Structure Person
2      Map Friends.s()
3  EndStructure
4
```

```

5 | Henry.Person\Friends("1") = "Paul"
6 |
7 | ResetStructure(@Henry, Person)
8 |
9 | ; Will print an empty string as the whole structure has been reset.
   | The map is still usable but empty.
10 | ;
11 | Debug Henry\Friends("1")

```

Syntax

`Result = Bool(<boolean expression>)`

Description

`Bool` can be used to evaluate a boolean expression outside of regular conditional operator like `If`, `While`, `Until` etc.

Parameters

boolean expression The Boolean expression to test.

Return value

Returns `#True` if the boolean expression is true, `#False` otherwise.

Example

```

1 | Hello$ = "Hello"
2 | World$ = "World"
3 |
4 | Debug Bool(Hello$ = "Hello") ; will print 1
5 | Debug Bool(Hello$ <> "Hello" Or World$ = "World") ; will print 1

```

Chapter 25

Data

Introduction

PureBasic allows the use of `Data`, to store predefined blocks of information inside of your program. This is very useful for default values of a program (language string for example) or, in a game, to define the sprite way to follow (precalculated).

`DataSection` must be called first to indicate a data section follow. This means all labels and data component will be stored in the `data` section of the program, which has a much faster access than the code section. `Data` will be used to enter the data. `EndDataSection` must be specified if some code follows the data declaration. One of good stuff is you can define different `Data` sections in the code without any problem. `Restore` and `Read` command will be used to retrieve the data. These functions are not thread safe so don't use them inside a thread.

Commands

Syntax

`DataSection`

Description

Start a data section.

Example

```
1  DataSection
2      NumericData:
3      Data.w 120, 250, 645 ; 3 'word' sized numbers
4  EndDataSection
```

Syntax

`EndDataSection`

Description

End a data section.

Example

```
1  DataSection
2      NumericData:
3      Data.w 0, 1, 2
4  EndDataSection
```

Syntax

`Data.TypeName`

Description

Defines data. The type can only be a native basic type (integer, long, word, byte, ascii, unicode, float, double, quad, character, string). Any number of data can be put on the same line, each one delimited with a comma ','. A literal string cannot exceed 8192 characters.

Example

```
1  DataSection
2      MixedData:
3      Data.l 100, 200, -250, -452, 145
4      Data.s "Hello", "This", "is ", "What ?"
5  EndDataSection
```

For advanced programmers: it's also possible to put a procedure address or a label address inside `Data` when its type is set to integer (.i). (Using the 'integer' type will store the (different) addresses accurate on both 32-bit and 64-bit environments.) This can be used to build easy virtual function tables for example.

Example

```
1  Procedure Max(Number, Number2)
2  EndProcedure
3
4  Label:
5
6  DataSection
7      MixedData:
8      Data.i ?Label, @Max()
9  EndDataSection
```

Example

```
1  Interface MyObject
2      DoThis()
3      DoThat()
4  EndInterface
5
6  Procedure This(*Self)
7      MessageRequester("MyObject", "This")
```

```

8      EndProcedure
9
10     Procedure That(*Self)
11         MessageRequester("MyObject", "That")
12     EndProcedure
13
14     m.MyObject = ?VTable
15
16     m\DoThis()
17     m\DoThat()
18
19
20     DataSection
21         VTable:
22             Data.i ?Procedures
23         Procedures:
24             Data.i @This(), @That()
25     EndDataSection

```

Syntax

```
Restore label
```

Description

This keyword is useful to set the start indicator for the [Read](#) to a specified label. All labels used for this purpose should be placed within the [DataSection](#) because the data is treated as a separate block from the program code when it is compiled and may become disassociated from a label if the label were placed outside of the DataSection.

Example

```

1      Restore StringData
2      Read.s MyFirstData$
3      Read.s MySecondData$
4
5      Restore NumericalData
6      Read.l a
7      Read.l b
8
9      Debug MyFirstData$
10     Debug a
11
12     End
13
14     DataSection
15         NumericalData:
16             Data.l 100, 200, -250, -452, 145
17
18         StringData:
19             Data.s "Hello", "This", "is ", "What ?"
20     EndDataSection

```

Syntax

`Read[.<type>] <variable>`

Description

Read the next available data. The next available data can be changed by using the [Restore](#) command. By default, the next available data is the first data declared. The type of data to read is determined by the type suffix. The default type will be used if it is not specified. It is however advisable to use the ad-hoc type in order to avoid the error message that will appear when you read data string and to avoid an integer type confusion which is a 'Long' type for 32-bit compilers and which is a 'Quad' type for 64-bit compilers.

Example

```
1  Restore StringData
2  Read.s MyFirstData$
3
4  Restore NumericalData
5  Read a ; Beware, it will read an Integer (Long in 32-bits or Quad
6      in 64-bits compilers)
7  Read.q b
8  Read c ; Beware, read a 'Quad' in a 64-bit compiler even if the
9      Data is a 'Long'!
10 Read.l d
11
12 Debug MyFirstData$ ; Display Hello
13 Debug a ; Display 100
14 Debug b ; Display 1111111111111111
15 Debug c ; Beware, the display depends of your compiler! : 200 in
16     32-bits or 1288490189000 in 64-bits
17 Debug d ; Idem : 300 in 32-bits or 400 in 64-bits
18
19 End
20
21 DataSection
22     NumericalData:
23         Data.i 100
24         Data.q 1111111111111111
25         Data.l 200, 300, 400
26
27     StringData:
28         Data.s "Hello", "This", "is ", "What ?"
29 EndDataSection
```

Chapter 26

Debugger keywords in PureBasic

Overview

A complete description of all functions of the powerful debugger you will find in the extra chapters Using the debugger or Using the debugging Tools .

Following is a list of special keywords to control the debugger from your source code. There is also a Debugger library which provides further functions to modify the behavior of the debugger should it be present. Several compiler constants useful also for debugging purposes you find in the Compiler directives section.

Syntax

```
CallDebugger
```

Description

This invokes the "debugger" and freezes the program immediately.

Syntax

```
Debug <expression> [, DebugLevel]
```

Description

Display the DebugOutput window and the result inside it. The expression can be any valid PureBasic expression, from numeric to string. An important point is the Debug command and its associated expression is totally ignored (not compiled) when the debugger is deactivated.

Note: This is also true, if you're using complete command lines after Debug (e.g. Debug LoadImage(1,"test.bmp")). They will not be compiled with disabled debugger!

This means this command can be used to trace easily in the program without having to comment all the debug commands when creating the final executable.

The 'DebugLevel' is the level of priority of the debug message. All normal debug message (without specifying a debug level) are automatically displayed. When a level is specified then the message will be only displayed if the current DebugLevel (set with the following [DebugLevel](#) command) is equals or above this number. This allows hierarchical debug mode, by displaying more and more precise information in function of the used DebugLevel.

Syntax

```
DebugLevel <constant expression>
```

Description

Set the current debug level for the 'Debug' message.

Note: The debug level is set at compile time, which means you have to put the [DebugLevel](#) command before any other Debug commands to be sure it affects them all.

Syntax

```
DisableDebugger
```

Description

This will disable the debugger checks on the source code which follow this command. It doesn't not fully turn off the debugger, so performance checks should not be done using [DisableDebugger](#) command, but by disabling the debugger before compiling the program.

Syntax

```
EnableDebugger
```

Description

This will enable the debugger checks on the source code which follow this command (if the debugger was previously disabled with [DisableDebugger](#)).

Note: [EnableDebugger](#) doesn't have an effect, if the debugger is completely disabled in the IDE (look at Compiler settings).

Chapter 27

Define

Syntax

```
Define.<type> [<variable> [= <expression>], <variable> [=
    <expression>], ...]
```

Description

Assign the same data type to a series of variables .

Without this keyword, variables are created with the default type of PureBasic which is the type INTEGER. As a reminder, the type INTEGER is:

4 bytes (with a 32-bit compiler) ranging from -2147483648 to + 2147483647

8 bytes (with a 64-bit compiler) ranging from -9223372036854775808 to +9223372036854775807

Define is very flexible because it also allows you to assign a different type to a particular variable within a series.

Define can also be used with arrays , the lists and the maps .

Example: Serial assignment

```
1 Define.b a, b = 10, c = b*2, d ; these 4 variables will be byte
   typed (.b)
```

Example: Mix of individual and serial assignments

```
1 Define.q a, b.w, c, d, st.s = "ok" ; a, c, and d are Quad (.q), b
   is a Word (.w) and st a String (.s)
2
3 Debug SizeOf(a) ; will print 8
4 Debug SizeOf(b) ; will print 2, because it doesn't have the default
   type
5 Debug SizeOf(c) ; will print 8
6 Debug SizeOf(d) ; will print 8
7 Debug st ; will print ok
```

Syntax

```
Define <variable>.<type> [= <expression>] [, <variable>.<type> [=
    <expression>], ...]
```

Description

Alternative possibility for the variable declaration using [Define](#).

Example

```
1  Define MyChar.c
2  Define MyLong.l
3  Define MyWord.w
4
5  Debug SizeOf(MyChar)      ; will print 2
6  Debug SizeOf(MyLong)     ; will print 4
7  Debug SizeOf(MyWord)     ; will print 2
```

Chapter 28

Dim

Syntax

```
Dim name.<type>(<expression>, [<expression>], ...)
```

Description

Dim is used to create new arrays (the initial value of each element will be zero). An array in PureBasic can be of any types, including structured , and user defined types. Once an array is defined it can be resized with **ReDim**. Arrays are dynamically allocated which means a variable or an expression can be used to size them. To view all commands used to manage arrays, see the Array library.

When you define a new array, please note that it will have one more element than you used as parameter, because the numbering of the elements in PureBasic (like in other BASIC's) starts at element 0. For example when you define **Dim**(10) the array will have 11 elements, elements 0 to 10. This behavior is different for static arrays in structures . Static arrays use brackets "[]", for example `ArrayStatic[2]` has only 2 elements from 0 to 1 and library functions `Array` don't work with them. The new arrays are always locals, which means `Global` or `Shared` commands have to be used if an array declared in the main source need to be used in procedures. It is also possible to pass an array as parameter to a procedure - by using the keyword **Array**. It will be passed "by reference" (which means, that the array will not be copied, instead the functions in the procedure will manipulate the original array).

To delete the content of an array and release its used memory during program flow, call `FreeArray()` .

If **Dim** is used on an existing array, it will reset its contents to zero.

For fast swapping of array contents the `Swap` keyword is available.

Note: Array bound checking is only done when the runtime Debugger is enabled.

Example

```
1 Dim MyArray(41)
2 MyArray(0) = 1
3 MyArray(1) = 2
```

Example: Multidimensional array

```
1 Dim MultiArray.b(NbColumns, NbLines)
2 MultiArray(10, 20) = 10
3 MultiArray(20, 30) = 20
```

Example: Array as procedure parameter

```
1  Procedure fill(Array Numbers(1), Length) ; the 1 stands for the
   number of dimensions in the array
2      For i = 0 To Length
3          Numbers(i) = i
4      Next
5  EndProcedure
6
7  Dim Numbers(10)
8  fill(Numbers(), 10) ; the array Numbers() will be passed as
   parameter here
9
10 Debug Numbers(5)
11 Debug Numbers(10)
```

Syntax

`ReDim name.<type>(<expression>, [<expression>], ...)`

Description

`ReDim` is used to 'resize' an already declared array while preserving its content. The new size can be larger or smaller, but the number of dimension of the array cannot be changed.

If `ReDim` is used with a multi-dimension array, only its last dimension can be changed.

Example

```
1  Dim MyArray.1(1) ; We have 2 elements
2  MyArray(0) = 1
3  MyArray(1) = 2
4
5  ReDim MyArray(4) ; Now we want 5 elements
6  MyArray(2) = 3
7
8  For k = 0 To 2
9      Debug MyArray(k)
10 Next
```

Example: Multi-dimension array

```
1  Dim MyTab.1(1,1,1)
2
3  ; ReDim MyTab(4,1,1) ; NO !
4  ; ReDim MyTab(1,4,1) ; NO !
5  ReDim MyTab(1,1,4) ; YES, only its last dimension can be changed!
6  MyTab(1,1,4) = 3
```

Chapter 29

Building a DLL

PureBasic allows to create standard Microsoft Windows DLL (Dynamic Linked Library), shared objects (.so) on Linux, and dynamic libraries (.dylib) on MacOS X. The DLL code is like a PureBasic code excepts than no real code should be written outside of procedure.

When writing a DLL, all the code is done inside procedures. When a procedure should be public (ie: accessible by third programs which will use the DLL), the keyword `ProcedureDLL` (or `ProcedureCDLL` if the procedure needs to be in 'CDecl' format, which is not the case of regular Windows DLL) is used instead of `Procedure` (and `DeclareDLL` or `DeclareCDLL` if are used instead of `Declare`). This is the only change to do to a program.

When this is done, select 'Shared DLL' as output format ('Compiler Option' window in the PureBasic editor or /DLL switch in command line) and a DLL with the name you set (in the save-requester when using the IDE) will be created in the selected directory.

Example

```
1  ProcedureDLL MyFunction()
2      MessageRequester("Hello", "This is a PureBasic DLL !", 0)
3  EndProcedure
4
5      ; Now the client program, which use the DLL
6      ;
7      If OpenLibrary(0, "PureBasic.dll")
8          CallFunction(0, "MyFunction")
9          CloseLibrary(0)
10     EndIf
```

For advanced programmers: there is 4 special procedures which are called automatically by the OS when one of the following events happen:

- DLL is attached to a new process
- DLL is detached from a process
- DLL is attached to a new thread
- DLL is detached from a thread

To handle that, it's possible to declare 4 special procedures called: `AttachProcess`(Instance), `DetachProcess`(Instance), `AttachThread`(Instance) and `DetachThread`(Instance). This means these 4 procedures names are reserved and can't be used by the programmer for other purposes.

Notes about creating DLL's:

- The declaration of arrays, lists or map with `Dim`, `NewList` or `NewMap` must always be done inside the procedure `AttachProcess`.
- Don't write program code outside procedures. The only exception is the declaration of variables or structures.
- Default values in procedure parameters have no effect.

- DirectX initialization routines must not be written in the AttachProcess procedure.

Note about returning strings from DLL's:

If you want to return a string out of a DLL, the string has to be declared as Global before using it.

Example

```
1 Global ReturnString$
2
3 ProcedureDLL.s MyFunction(var.s)
4     ReturnString$ = var + " test"
5     ProcedureReturn ReturnString$
6 EndProcedure
```

Without declaring it as Global first, the string is local to the ProcedureDLL and can't be accessed from outside.

When using CallFunction() (or one of its similar CallXXX functions) on a DLL function you will get a pointer on the return string, which you could read with PeekS() .

Example

```
1 String.s = PeekS(CallFunction(0, "FunctionName", Parameter1,
    Parameter2))
```

Here a complete code example:

Chapter 30

Enumerations

Syntax

```
Enumeration [name] [<constant> [Step <constant>]]  
    #Constant1  
    #Constant2 [= <constant>]  
    #Constant3  
    ...  
EndEnumeration
```

```
EnumerationBinary [name] [<constant>]  
    #Constant1  
    #Constant2 [= <constant>]  
    #Constant3  
    ...  
EndEnumeration
```

Description

[Enumerations](#) are very handy to declare a sequence of constants quickly without using fixed numbers. The first constant found in the enumeration will get the number 0 and the next one will be 1 etc. It's possible to change the first constant number and adjust the step for each new constant found in the enumeration. If needed, the current constant number can be altered by affecting with '=' the new number to the specified constant. As [Enumerations](#) only accept integer numbers, floats will be rounded to the nearest integer.

A name can be set to identify an enumeration and allow to continue it later. It is useful to group objects altogether while declaring them in different code place.

For advanced user only: the reserved constant `#PB_Compiler_EnumerationValue` store the next value which will be used in the enumeration. It can be useful to get the last enumeration value or to chain two enumerations.

[EnumerationBinary](#) can be used to create enumerations suitable for flags. The first item value is 1.

Example: Simple enumeration

```
1  Enumeration  
2      #GadgetInfo ; will be 0  
3      #GadgetText ; will be 1  
4      #GadgetOK   ; will be 2  
5  EndEnumeration
```


Example: Enumeration with step

```
1 Enumeration 20 Step 3
2   #GadgetInfo ; will be 20
3   #GadgetText ; will be 23
4   #GadgetOK   ; will be 26
5 EndEnumeration
```

Example: Enumeration with dynamic change

```
1 Enumeration
2   #GadgetInfo ; will be 0
3   #GadgetText = 15 ; will be 15
4   #GadgetOK   ; will be 16
5 EndEnumeration
```

Example: Named enumeration

```
1 Enumeration Gadget
2   #GadgetInfo ; will be 0
3   #GadgetText ; will be 1
4   #GadgetOK   ; will be 2
5 EndEnumeration
6
7 Enumeration Window
8   #FirstWindow ; will be 0
9   #SecondWindow ; will be 1
10 EndEnumeration
11
12 Enumeration Gadget
13   #GadgetCancel ; will be 3
14   #GadgetImage  ; will be 4
15   #GadgetSound  ; will be 5
16 EndEnumeration
```

Example: Getting next enumeration value

```
1 Enumeration
2   #GadgetInfo ; will be 0
3   #GadgetText ; will be 1
4   #GadgetOK   ; will be 2
5 EndEnumeration
6
7 Debug "Next enumeration value: " + #PB_Compiler_EnumerationValue ;
   will print 3
```

Example: Binary enumeration

```
1 EnumerationBinary
2   #Flags1 ; will be 1
3   #Flags2 ; will be 2
```

```
4      #Flags3 ; will be 4
5      #Flags4 ; will be 8
6      #Flags5 ; will be 16
7  EndEnumeration
```

Chapter 31

For : Next

Syntax

```
For <variable> = <expression1> To <expression2> [Step <constant>]  
    ...  
Next [<variable>]
```

Description

For : Next is used to create a loop within a program with the given parameters. At each loop the <variable> value is increased by a 1, (or of the "Step value" if a **Step** value is specified) and when the <variable> value is above the <expression2> value, the loop stop.

With the Break command its possible to exit the **For : Next** loop at any moment, with the Continue command the end of the current iteration can be skipped.

The **For : Next** loop works only with integer values, at the expressions as well at the **Step** constant. The **Step** constant can also be negative.

Example

```
1 For k = 0 To 10  
2   Debug k  
3 Next
```

In this example, the program will loop 11, time (0 to 10), then quit.

Example

```
1 For k = 10 To 1 Step -1  
2   Debug k  
3 Next
```

In this example, the program will loop 10 times (10 to 1 backwards), then quit.

Example

```
1 a = 2  
2 b = 3
```

```
3  For k = a+2 To b+7 Step 2
4      Debug k
5  Next k
```

Here, the program will loop 4 times before quitting, (k is increased by a value of 2 at each loop, so the k value is: 4-6-8-10). The "k" after the "Next" indicates that "Next" is ending the "For k" loop. If another variable, is used the compiler will generate an error. It can be useful to nest several "For/Next" loops.

Example

```
1  For x=0 To 10
2      For y=0 To 5
3          Debug "x: " + x + " y: " + y
4      Next y
5  Next x
```

Note: Be aware, that in PureBasic the value of <expression2> ('To' value) can also be changed inside the For : Next loop. This can lead to endless loops when wrongly used.

Chapter 32

ForEach : Next

Syntax

```
ForEach List() Or Map()  
...  
Next [List() Or Map()]
```

Description

ForEach loops through all elements in the specified list or map starting from the first element up to the last. If the list or the map is empty, **ForEach : Next** exits immediately. To view all commands used to manage lists, please click [here](#) . To view all commands used to manage maps, please click [here](#) . When used with list, it's possible to delete or add elements during the loop. As well it's allowed to change the current element with `ChangeCurrentElement()` . After one of the mentioned changes the next loop continues with the element following the current element. When used with map, please do not put a key (see example below). With the `Break` command its possible to exit the **ForEach : Next** loop at any moment, with the `Continue` command the end of the current iteration can be skipped.

Example: list

```
1  NewList Number()  
2  
3  AddElement(Number())  
4  Number() = 10  
5  
6  AddElement(Number())  
7  Number() = 20  
8  
9  AddElement(Number())  
10 Number() = 30  
11  
12 ForEach Number()  
13     Debug Number() ; Will output 10, 20 and 30  
14 Next
```

Example: Map

```
1  NewMap Country.s()
2
3  Country("US") = "United States"
4  Country("FR") = "France"
5  Country("GE") = "Germany"
6
7  ForEach Country()
8      Debug Country()
9      ;Debug Country("FR") ; No, do not put a key because sometimes it can
      cause an infinite loop
10 Next
```

Chapter 33

General Syntax Rules

PureBasic has established rules which never change. These are:

Comments

Example

```
1  If a = 10 ; This is a comment to indicate something.
```

Keywords

All **keywords** are used for general things inside PureBasic, like creating arrays (Dim) or lists (NewList), or controlling the program flow (If : Else : EndIf). They are not followed by the brackets '()', which are typically used for PureBasic **functions**.

Example

```
1  If a = 1      ; If, Else and EndIf are keywords; while 'a = 1'  
2      ...      ; is a variable used inside an expression.  
3  Else  
4      ...  
5  EndIf
```

Keywords are regularly described in the chapters on the left side of the index page in the reference manual.

Functions

Every **function** must be followed by an opening round bracket '(' character, otherwise it will not be considered a function (this applies also to functions without parameters).

Example

```
1   EventWindow() ; it is a function.
2   EventWindow   ; it is a variable.
```

Functions are regularly included in the PureBasic "Command libraries", described on the right side of the index page in the reference manual.

Constants

All constants are preceded by the hash `#` character. They can only be declared once in the source and always keep their predefined values. (The compiler replaces all constant names with their corresponding values when compiling the executable.)

Example

```
1   #Hello = 10 ; it is a constant.
2   Hello  = 10 ; it is a variable.
```

Literal strings

Literal strings are declared using the `"` character. Escape sequences are supported by prepending the `**` character to the literal string. The supported escape sequences are:

<code>\a:</code>	alarm	<code>Chr(7)</code>
<code>\b:</code>	backspace	<code>Chr(8)</code>
<code>\f:</code>	formfeed	<code>Chr(12)</code>
<code>\n:</code>	newline	<code>Chr(10)</code>
<code>\r:</code>	carriage return	<code>Chr(13)</code>
<code>\t:</code>	horizontal tab	<code>Chr(9)</code>
<code>\v:</code>	vertical tab	<code>Chr(11)</code>
<code>\":</code>	double quote	<code>Chr(34)</code>
<code>\\:</code>	backslash	<code>Chr(92)</code>

There are two special constants for strings:

```
#Empty$: represents an empty string (exactly the same as "")
#Null$ : represents a null string. This can be used for API
         functions requiring a null pointer to a string, or to really
         free a string.
```

Warning: On Windows, `\t` does not work with the graphical functions of the 2DDrawing and VectorDrawing libraries.

Example

```
1   a$ = "Hello world" ; standard string
2   b$ = ~"Escape\nMe !" ; string with escape sequences
```


Labels

All labels must be followed by a colon : character. Label names may not contain any operators (+,-,...) or special characters (ß,ä,ö,ü,...). Labels defined inside a procedure will be available only in that procedure.

Example

```
1 I_am_a_label :
```

Modules

Modules use a :: (double colon) when referencing a module name. Be careful not to confuse these with the single colon used by labels.

Example

```
1 Debug CarModule :: NbCars
```

Expressions

An expression is something which can be evaluated. An expression can mix any variables, constants, or functions, of the same type. When using numbers in an expression, you can prefix them with the \$ symbol to indicate a hexadecimal number, or with the % symbol to indicate a binary number. Without either of those, the number will be treated as decimal. Strings must be enclosed within straight double quotes.

Example

```
1 a = a + 1 + (12 * 3)
2
3 a = a + WindowHeight(#Window) + b/2 + #MyConstant
4
5 If a <> 12 + 2
6     b + 2 >= c + 3
7 EndIf
8
9 a$ = b$ + "this is a string value" + c$
10
11 Foo = Foo + $69 / %1001 ; Hexadecimal and binary number usage
```

Concatenation of commands

Any number of commands can be added to the same line by using the : option.

Example

```
1  If IsCrazy = 0 : MessageRequester("Info", "Not Crazy") : Else :  
    MessageRequester("Info", "Crazy") : EndIf
```

Line continuation

Long expressions can be split across several lines. A split line has to end with one of the following operators: plus (+), comma (,), or (||), And, Or, Xor.

Example

```
1  Text$ = "Very very very very long text" + #LF$ +  
2      "another long text" + #LF$ +  
3      " and the end of the long text"  
4  
5  MessageRequester("Hello this is a very long title",  
6      "And a very long message, so we can use the  
    multiline" + #LF$ + Text$,  
7      #PB_MessageRequester_Ok)
```

Typical words in this manual

Words used in this manual:

- <variable> : a basic variable.
- <expression> : an expression as explained above.
- <constant> : a numeric constant.
- <label> : a program label.
- <type> : any type, (standard or structured).

Others

- In this guide, all topics are listed in alphabetical order to decrease any search time.
- **Return values** of commands are always Integer if no other type is specified in the Syntax line of the command description.
- In the PureBasic documentation, the terms "commands" and "functions" are used interchangeably, regardless of whether the function returns a value or not. To learn whether value is returned by a specific command or function, consult the description provided in the command's documentation.

Chapter 34

Global

Syntax

```
Global [.<type>] <variable[.<type>]> [= <expression>] [, ...]
```

Description

Global provides the ability for variables to be defined as global, i.e., variables defined as such may then be accessed within a Procedure . In this case the command **Global** must be called for the according variables, **before** the declaration of the procedure. This rule is true everywhere except in a single case: The modules do not have access to the global declared variables outside this module. Each variable may have a default value directly assigned to it. If a type is specified for a variable after **Global**, the default type is changed through the use of this declaration. **Global** may also be used with arrays , lists and maps .

In order to use local variables within a procedure, which have the same name as global variables, take a look at the Protected and Static keywords.

Example: With variables

```
1 Global a.l, b.b, c, d = 20
2
3 Procedure Change()
4     Debug a ; Will be 10 as the variable is global
5 EndProcedure
6
7 a = 10
8 Change()
```

Example: With array

```
1 Global Dim Array(2)
2
3 Procedure Change()
4     Debug Array(0) ; Will be 10 as the array is global
5 EndProcedure
6
7 Array(0) = 10
8 Change()
```

Example: With default type

```
1 ; 'Angle' and 'Position' will be a float, as they didn't have a
   specified type
2 ;
3 Global.f Angle, Length.b, Position
```

Example: Complex with a module

```
1 Global Var_GlobalOutsideModule = 12
2 Debug Var_GlobalOutsideModule ; Display 12
3
4 DeclareModule Ferrari
5 Global Var_GlobalModule = 308
6 #FerrariName$ = "458 Italia"
7 Debug #FerrariName$ ; Display "458 Italia"
8
9 ; Debug Var_GlobalOutsideModule ; Error, the variable already exists
10 Debug Var_GlobalModule ; Display 308
11
12 Declare CreateFerrari()
13 EndDeclareModule
14
15
16 ; Private
17 ;
-----
18 Module Ferrari
19 Debug Var_GlobalOutsideModule ; Display 0 <== Look !
20 Debug Var_GlobalModule ; Display 308
21
22 Global Initialized = 205
23 Debug Initialized ; Display 205
24
25 Procedure Init()
26 Debug Var_GlobalOutsideModule ; Display 0
27 Debug Var_GlobalModule ; Display 308
28 Debug "InitFerrari()"
29 EndProcedure
30
31 Procedure CreateFerrari() ; Public
32 Init()
33 Debug "CreateFerrari()"
34 Debug Var_GlobalOutsideModule ; Display 0
35 Debug Var_GlobalModule ; Display 308
36 EndProcedure
37
38 EndModule
39
40
41 ; Main Code
42 ;
-----
43 Procedure Init()
44
45 Debug " init() from main code."
46 Debug Var_GlobalOutsideModule ; Display 12
```

```

47     Debug Var_GlobalModule      ; Display 0
48 EndProcedure
49
50 Init()
51
52 Ferrari::CreateFerrari()
53 Debug Ferrari::#FerrariName$ ; Display 458 Italia
54 Debug Var_GlobalOutsideModule ; Display 12
55 ; Debug Var_GlobalModule      ; Error, the variable already exists
56
57
58 UseModule Ferrari
59
60 CreateFerrari()
61 Debug #FerrariName$           ; Display 458 Italia
62 Debug Var_GlobalOutsideModule ; Display 12
63 Debug Var_GlobalModule        ; Display 308 <== Look !
64
65 UnuseModule Ferrari
66 ; Debug #FerrariName$         ; Error, does not exist
67 Debug Var_GlobalOutsideModule ; Display 12
68 Debug Var_GlobalModule        ; Display 0 <== Look !

```

Chapter 35

Gosub : Return

Syntax

```
Gosub MyLabel
```

```
MyLabel:
```

```
...
```

```
Return
```

Description

Gosub stands for 'Go to sub routine'. A label must be specified after **Gosub**, at that point the program execution continues immediately after the position defined by that label, and will do so until encountering a **Return**. When a return is reached, the program execution is then transferred immediately below the **Gosub**.

Gosub is useful when building fast structured code.

Another technique which may be used in order to insert a sub routine into a standalone program component is to use procedures . **Gosub** may only be used within the main body of the source code, and may not be used within procedures .

Example

```
1  a = 1
2  b = 2
3  Gosub ComplexOperation
4  Debug a
5  End
6
7  ComplexOperation:
8      a = b*2+a*3+(a+b)
9      a = a+a*a
10 Return
```

Syntax

```
FakeReturn
```

Description

If the command Goto is used within the body of a sub routine, [FakeReturn](#) must be used. [FakeReturn](#) simulates a return without actually executing a return, and if it is not used, the program will crash.

Note: To exit a loop safely, Break should be used instead of Goto.

Example

```
1  Gosub SubRoutine1
2
3  SubRoutine1:
4      ...
5      If a = 10
6          FakeReturn
7          Goto Main_Loop
8      EndIf
9  Return
```

Chapter 36

Handles and Numbers

Numbers

All created objects are identified by an arbitrary number (which is not the object's handle, as seen below). In this manual, these numbers are marked as `#Number` (for example, each gadget created have a `#Gadget` number).

The numbers you assign to them do not need to be constants, but they need to be unique for each object in your program (an image can get the same number as a gadget, because these are different types of objects). These numbers are used to later access these objects in your program.

For example, the event handling functions return these numbers:

```
1  EventGadget()
2  EventMenu()
3  EventWindow()
```

Handles

All objects also get a unique number assigned to them by the system. These identifiers are called handles. Sometimes, a PureBasic function doesn't need the number as argument, but the handle. In this manual, such things are mentioned, as an ID.

Example

```
1  ImageGadget(#Gadget, x, y, Width, Height, ImageID [, Flags])
2  ; #Gadget needs to be the number you want to assign to the Gadget
3  ; ImageID needs to a handle to the image.
```

To get the handle to an object, there are special functions like:

```
1  FontID()
2  GadgetID()
3  ImageID()
4  ThreadID()
5  WindowID()
```

Also, most of the functions that create these objects also return this handle as a result, if they were successful. This is only the case if `#PB_Any` was not used to create the object. If `#PB_Any` is used, these commands return the object number that was assigned by PB for them, not the handle.

Example

```
1 GadgetHandle = ImageGadget(...)
```

Chapter 37

If : Else : EndIf

Syntax

```
If <expression>
...
[ElseIf <expression>]
...
[Else]
...
EndIf
```

Description

The **If** structure is used to achieve tests, and/or change the programmes direction, depending on whether the test is true or false. **ElseIf** optional command is used for any number of additional tests if the previous test was not true. The **Else** optional command is used to execute a part of code, if all previous tests were false. Any number of **If** structures may be nested together.

Short-circuit evaluations for expressions are supported, meaning if a test is true, all following tests will be ignored and not even run.

Example: Basic test

```
1  a = 5
2  If a = 10
3      Debug "a = 10 "
4  Else
5      Debug "a <> 10 "
6  EndIf
```

Example: Multiple test

```
1  a = 10
2  ; b = 15
3  c = 20
4
5  If (a = 10) And (b >= 10) Or (c = 20)
6      If b = 15
7          Debug "b = 15 "
8      Else
```

```
9      Debug "Other possibility"
10    EndIf
11  Else
12    Debug "Test failure"
13  EndIf
```

Example: Short-circuit test

```
1  Procedure DisplayHello()
2    Debug "Hello"
3    ProcedureReturn 1
4  EndProcedure
5
6  a = 10
7  If a = 10 Or DisplayHello() = 1 ; a is equal to 10, so the second
   test is fully ignored
8    Debug "Test success"
9  Else
10    Debug "Test failure"
11  EndIf
```

Chapter 38

Import : EndImport

Syntax

```
Import "Filename "  
    FunctionName.<type>(<parameter>, [, <parameter> [= DefaultValue]...])  
    [As "SymbolName"]  
    ...  
    VariableName.<type> [As "SymbolName"]  
EndImport
```

Description

For advanced programmers. `Import : EndImport` allows to easy declare external functions and variables from a library (.lib) or an object (.obj) file.

Once declared, the imported functions are directly available for use in the program, like any other commands. The compiler doesn't check if the functions really exists in the imported file, so if an error occurs, it will be reported by the linker.

This feature can replace the `OpenLibrary()` / `CallFunction()` sequence as it has some advantages: type checking is done, number of parameters is validated. Unlike `CallFunction()`, it can deal with double, float and quad without any problem.

The last parameters can have a default value (need to be a constant expression), so if these parameters are omitted when the function is called, the default value will be used.

By default the imported function symbol is 'decorated' in the following way: `_FunctionName@callsize`. That should work for most of the functions which use the standard call convention (stdcall). If the library is a C one, and the function are not stdcall, the `ImportC` variant should be used instead. In this case, the default function symbol is decorated like: `_FunctionName`.

The pseudotypes can be used for the parameters, but not for the returned value.

Remarks

On x64, there is only one calling convention, so `ImportC` will behave the same as `Import`.

Example

```
1  CompilerIf #PB_Compiler_OS <> #PB_OS_Windows  
2      CompilerError "This sample only works on Windows"  
3  CompilerEndIf  
4  
5  Import "User32.lib"
```

```

6
7     ; No need to use 'As' as PureBasic decorates the function correctly
8     ; We also define the 'Flags' as optional, with a default value of 0
    (when omitted)
9     ;
10    MessageBoxW(Window.i, Body$, Title$, Flags.i = 0)
11
12 EndImport
13
14 MessageBoxW(0, "Hello", "World") ; We don't specify the flags

```

Example: With pseudotypes

```

1  CompilerIf #PB_Compiler_OS <> #PB_OS_Windows
2      CompilerError "This sample only works on Windows"
3  CompilerEndIf
4
5  Import "User32.lib"
6
7      ; We use the 'p-unicode' pseudotype for the string parameters, as
8      ; MessageBoxW() is an unicode only function. The compiler will
9      ; automatically converts the strings to unicode when needed.
10     ;
11     MessageBoxA(Window.i, Body.p-ascii, Title.p-ascii, Flags.i = 0)
12
13 EndImport
14
15 MessageBoxA(0, "Hello", "World")

```

Chapter 39

Includes Functions

Syntax

```
IncludeFile "Filename"
```

Description

`IncludeFile` will always include the specified source file, at the current place in the code (even if `XIncludeFile` has been called for this file before).

Example

```
1 IncludeFile "Sources\myfile.pb" ; This file will be inserted in the
   current code.
```

This command is useful, if you want to split your source code into several files, to be able to reuse parts e.g. in different projects.

Syntax

```
XIncludeFile "Filename"
```

Description

`XIncludeFile` is similar to `IncludeFile` excepts it avoids to include the same file twice.

Example

```
1 XIncludeFile "Sources\myfile.pb" ; This file will be inserted.
2 XIncludeFile "Sources\myfile.pb" ; This file will be ignored along
   with all subsequent calls.
```

This command is useful, if you want to split your source code into several files, to be able to reuse parts e.g. in different projects.

Syntax

```
IncludeBinary "filename"
```

Description

`IncludeBinary` will include the named file at the current place in the code. Including should be done inside a Data block.

Example

```
1 DataSection
2   MapLabel :
3   IncludeBinary "Data\map.data"
4 EndDataSection
```

This command is especially useful in combination with the Catch-commands (currently there are `CatchImage()` , `CatchSound()` , `CatchSprite()`) to include images, sounds, sprites etc. into the executable.

Syntax

```
IncludePath "path"
```

Description

`IncludePath` will specify a default path for all files included after the call of this command. This can be very handy when you include many files which are in the same directory.

Example

```
1 IncludePath "Sources\Data"
2 IncludeFile "Sprite.pb"
3 XIncludeFile "Music.pb"
```

Chapter 40

Inline x86 ASM

Introduction

Using inline assembly code is not done the same way with the ASM backend and the C backend. With the compiler using the ASM backend, PureBasic allows you to include any x86 assembler commands (including MMX and FPU one) directly in the source code, as if it was a real assembler (X86 and X64 processors only). And it gives you even more: you can use directly any variables or pointers in the assembler keywords, you can put any ASM commands on the same line, ... On Windows and Linux, PureBasic uses **fasm** (<http://flatassembler.net>), so if you want more information about the syntax, just read the **fasm** guide.

On OS X, PureBasic uses **yasm** (<http://yasm.tortall.net/>), so if you want more information about the syntax, just read the **yasm** guide.

To activate the inline assembler use the compiler directives `EnableASM` and `DisableASM` .

It's possible to enable the ASM syntax coloring in the IDE with the "enable inline ASM syntax coloring" compiler option .

Rules

You have several rules to closely follow if you want to include ASM in a BASIC code :

- The used variables or pointers must be declared **before** to use them in an assembler keyword. Their names in assembler are 'v_variablename' and 'p_pointername', and in a procedure their names are 'p.v_variablename' and 'p.p_pointername'.
- Labels: The labels need to be referenced in lowercase when using the in-line ASM. When you reference a label , you must put the prefix 'll_' before the name.

If the label is defined in a procedure , then its prefix is 'll_procedurename_', in lowercase.

When you reference a module item, you must put the prefix 'module_name.ll_' all in lowercase before the item.

If the label is defined in a procedure inside a module, then its prefix is 'module_name.ll_procedurename_', in lowercase.

Example

```
1  DeclareModule MyModule
2      LabelDeclareModule: ;Its name is mymodule.ll_labeldeclaremodule:
3      Declare Init()
4  EndDeclareModule
5
6  Module MyModule
7      Procedure Init()
```



```

8      LabelModuleProcedure: ; Its name is
mymodule.ll_init_labelmoduleprocedure:
9      Debug "InitFerrari()"
10     EndProcedure
11
12     LabelModule1: ;Its name is mymodule.l_labelmodule1:
13 EndModule
14
15 Procedure Test (*Pointer, Variable)
16     TokiSTART: ;Its name is ll_test_tokistart:
17
18     ! MOV dword [p.p_Pointer], 20
19     ! MOV dword [p.v_Variable], 30
20     Debug *Pointer ;Its name is p.p_Pointer
21     Debug Variable ;Its name is p.v_Variable
22 EndProcedure
23
24 VAR=1 ;Its name is v_VAR
25 *Pointt=AllocateMemory(10) ;Its name is p_Pointt
26
27 MyModule::Init()
28 Test(0,0)
29
30 Label1: ;Its name is l_label1:
31
32 !jmp l_labelend ; An instruction in assembler has to use the rules
above. Here it's l_namelabel
33 ;...
34 LabelEnd: ;Its name is l_labelend:

```

- The errors in an ASM part are not reported by PureBasic but by FASM. Just check your code if a such error happen.
- With enabled InlineASM you can't use ASM keywords as label names in your source.
- On x86 processors, the available volatile registers are: eax, ecx and edx, xmm0, xmm1, xmm2 and xmm3. All others must be always preserved.
- On x64 processors, the available volatile registers are: rax, rcx, rdx, r8, r9, xmm0, xmm1, xmm2 and xmm3. All others must be always preserved.
- Windows only: an ASM help-file could be downloaded [here](#). If you place the 'ASM.HLP' in the 'Help/' folder of PureBasic, you can also get help on ASM keywords with F1. (Note: this feature is only enabled, when InlineASM is enabled).

When using assembler in a procedure , you have to be aware of several important things:

- To return directly the 'eax' (or 'rax' on x64) register content, just use [ProcedureReturn](#), without any expression. It will let the 'eax' (or 'rax' on x64) register content untouched and use it as return-value.

Example

```

1  Procedure.l MyTest()
2      MOV eax, 45
3      ProcedureReturn ; The returned value will be 45
4  EndProcedure

```

- Local variables in PureBasic are directly indexed by the stack pointer, which means if the stack pointer change via an ASM instruction (like PUSH, POP etc..) the variable index will be wrong and direct variable reference won't work anymore.
- It's possible to pass directly an assembly line to the assembler without being processed by the compiler by using the '!' character at the line start. This allow to have a full access to the assembler directives.

When using this, it's possible to reference the local variables using the notation 'p.v_variablename' for a regular variable or 'p.p_variablename' for a pointer.

Example

```
1  Procedure Test(*Pointer, Variable)
2      ! MOV dword [p.p_Pointer], 20
3      ! MOV dword [p.v_Variable], 30
4      Debug *Pointer
5      Debug Variable
6  EndProcedure
7
8  Test(0, 0)
```

"Features of inline assembler with compiler using C backend"

You can use the gcc syntax: <https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html>.

Example: Pi

```
1  pi  Procedure.d get_pi()
2
3      Define.d pi
4
5      !asm (
6      !      "fldpi\n"
7      !      "fstpl    %[v_pi]\n"
8      !      :[v_pi]"=m"(v_pi)::
9      !);
10
11     ProcedureReturn pi
12 EndProcedure
13
14 Define.d n
15 n=get_pi()
16 Debug n
```

Example: bswap

```
1  Procedure bswap(v.l)
2      Protected ret.l
3      CompilerIf #PB_Compiler_Backend = #PB_Backend_C
4          !".intel_syntax noprefix";
5          !"mov eax, v_v";
6          !"bswap eax";
7          !"mov v_ret, eax";
8      CompilerElse
9          !mov eax, [p.v_v]
10         !bswap eax
11         !mov [p.v_ret],eax
12     CompilerEndIf
13     ProcedureReturn ret
```

```

14 EndProcedure
15
16 x.i = $FF000000
17 Debug RSet(Hex(x),8,"0")
18 x = bswap(x)
19 Debug RSet(Hex(x),8,"0")

```

Example: Fibonacci

```

Procedure.q fib(ub.l, *f) If (ub<3) ProcedureReturn 1 EndIf ; !set_dpfpu(); !asm ( ! "mov %[p_f],
%%rax;" ! "fldz;" ! "fistl (%%rax);" ! "add $8, %%rax;" ! "fld1;" ! "fistl (%%rax);" ! "add $8, %%rax;" !
"mov %[v_ub], %%ecx;" ! "sub $2, %%ecx;" ! "fib2:" ! "fch %st(1);" ! "fadd %st(1),%st(0);" ! "fld
%st(0);" ! "fstpq (%%rax);" ! "add $8, %%rax;" ! "dec %%ecx;" ! "jg fib2;" ! "fstp %st(0);" ! "fstp
%st(0);" ! :[p_f]"=m"(p_f) ! :[v_ub]"m"(v_ub) ! : "rax", "ecx" !); ProcedureReturn EndProcedure Dim
fibonacci.q(20) n=20 fib(n+1, @fibonacci(0)) For i.l=0 To n Debug fibonacci(i) Next

```

Chapter 41

Interfaces

Syntax

```
Interface <name> [Extends <name>]  
    <Method[.<type>](<>)>  
    ...  
EndInterface
```

Description

Interfaces are used to access Object Oriented modules, such as COM (Component Object Model) or DirectX dynamic libraries (DLL). These types of libraries are becoming more and more common in Windows, and through the use of interfaces, the ability to access these modules easily (and without any performance hit) is realized. It also introduces the necessary basis for Object Oriented programming within PureBasic, but the use of interfaces requires some advanced knowledge. Most of the standard Windows interfaces have already been implemented within a resident file and this allows direct use of these objects.

The optional **Extends** parameter may be used to extend another interface with new functions (these functions are commonly called 'methods' in Object Oriented (OO) languages such as C++ or Java). All functions contained within the extended interface are then made available within the new interface and will be placed before the new functions. This is useful in order to do basic inheritance of objects.

Arrays can be passed as parameters using the **Array** keyword, lists using the **List** keyword and maps using the **Map** keyword.

A return type may be defined in the interface declaration by adding the type after the method.

SizeOf may be used with Interfaces in order to get the size of the interface and OffsetOf may be used to retrieve the index of the specified function.

The pseudotypes may be used for the parameters of the functions, but not for the return value.

Note: The concept of objects, and the capability provided within PureBasic for their use, has been developed for, and mainly targeted towards, experienced programmers. However, an understanding of these concepts and capabilities are in no way a prerequisite for creating professional software or games.

Example: Basic example of object call

```
1 ; In order to access an external object (within a DLL for example),  
2 ; the objects' interface must first be declared:  
3  
4 Interface MyObject  
5     Move(x,y)  
6     MoveF(x.f,y.f)  
7     Destroy()  
8 EndInterface
```

```

9
10 ; CreateObject is the function which creates the object, from the DLL,
11 ; whose interface has just been defined.
12 ; Create the first object...
13 ;
14 Object1.MyObject = MyCreateObject()
15
16 ; And the second one.
17 ;
18 Object2.MyObject = MyCreateObject()
19
20 ; Then the functions which have just been defined, may
21 ; be used, in order to act upon the desired object.
22 ;
23 Object1.Move(10, 20)
24 Object1.Destroy()
25
26 Object2.MoveF(10.5, 20.1)
27 Object2.Destroy()

```

Example: Example with 'Extends'

```

1 ; Define a basic Cube interface object.
2 ;
3 Interface Cube
4     GetPosition()
5     SetPosition(x)
6     GetWidth()
7     SetWidth(Width)
8 EndInterface
9
10 Interface ColoredCube Extends Cube
11     GetColor()
12     SetColor(Color)
13 EndInterface
14
15 Interface TexturedCube Extends Cube
16     GetTexture()
17     SetTexture(TextureID)
18 EndInterface
19
20 ; The interfaces for 3 different objects have now been defined, these
21 ; objects include:
22 ; - 'Cube' which has the: Get/SetPosition() and Get/SetWidth()
23 ;   functions.
24 ; - 'ColoredCube' which has the: Get/SetPosition(), Get/SetWidth()
25 ;   and Get/SetColor() functions.
26 ; - 'TexturedCube' which has the: Get/SetPosition(), Get/SetWidth()
27 ;   and Get/SetTexture() functions.
28 ;

```

Chapter 42

Licenses for the PureBasic applications (without using 3D engine)

This program makes use of the following components:

Component: MD5

Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

License to copy and use this software is granted provided that it is identified as the "RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing this software or this function.

License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this documentation and/or software.

Component: AES

Optimized ANSI C code for the Rijndael cipher (now AES)

@authorVincent Rijmen <vincent.rijmen@esat.kuleuven.ac.be>
@authorAntoon Bosselaers <antoon.bosselaers@esat.kuleuven.ac.be>
@authorPaulo Barreto <paulo.barreto@terra.com.br>

This code is hereby placed in the public domain.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Component: SHA1

SHA-1 in C
By Steve Reid <steve@edmweb.com>
100% Public Domain

Component: zlib

Copyright (C) 1995-2012 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly	Mark Adler
jloup@gzip.org	madler@alumni.caltech.edu

Component: libpq

Portions Copyright (c) 1996-2011, PostgreSQL Global Development Group
Portions Copyright (c) 1994, The Regents of the University of California

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement

is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY
FOR
DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES,
INCLUDING
LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS
DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF
THE
POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER
IS
ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO
OBLIGATIONS TO
PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Component: sqlite3

The author disclaims copyright to this source code. In place of
a legal notice, here is a blessing:

May you do good and not evil.
May you find forgiveness for yourself and forgive others.
May you share freely, never taking more than you give.

Component: libjpeg

The authors make NO WARRANTY or representation, either express or
implied,
with respect to this software, its quality, accuracy, merchantability,
or
fitness for a particular purpose. This software is provided "AS IS",
and you,
its user, assume the entire risk as to its quality and accuracy.

This software is copyright (C) 1991-2012, Thomas G. Lane, Guido
Vollbeding.

All Rights Reserved except as specified below.

Permission is hereby granted to use, copy, modify, and distribute this
software (or portions thereof) for any purpose, without fee, subject to
these
conditions:

(1) If any part of the source code for this software is distributed,
then this
README file must be included, with this copyright and no-warranty notice
unaltered; and any additions, deletions, or changes to the original
files
must be clearly indicated in accompanying documentation.
(2) If only executable code is distributed, then the accompanying
documentation must state that "this software is based in part on the
work of
the Independent JPEG Group".

(3) Permission *for* use of this software is granted only *if* the user accepts full responsibility *for* any undesirable consequences; *the authors accept NO LIABILITY for* damages of any kind.

These conditions apply *to* any software derived from *or* based on the IJG code, *not just to* the unmodified library. *If* you use our work, you ought *to* acknowledge us.

Permission is *NOT* granted *for* the use of any IJG author's name or company name in advertising or publicity relating to this software or products derived from it. This software may be referred to only as "the Independent JPEG Group's software".

We specifically permit and encourage the use of this software as the basis of commercial products, provided that all warranty or liability claims are assumed by the product vendor.

Component: libpng

libpng versions 1.2.6, August 15, 2004, through 1.5.12, July 11, 2012, are Copyright (c) 2004, 2006-2012 Glenn Randers-Pehrson, and are distributed according to the same disclaimer and license as libpng-1.2.5 with the following individual added to the list of Contributing Authors:

Cosmin Truta

libpng versions 1.0.7, July 1, 2000, through 1.2.5, October 3, 2002, are Copyright (c) 2000-2002 Glenn Randers-Pehrson, and are distributed according to the same disclaimer and license as libpng-1.0.6 with the following individuals added to the list of Contributing Authors:

Simon-Pierre Cadieux
Eric S. Raymond
Gilles Vollant

and with the following additions to the disclaimer:

There is no warranty against interference with your enjoyment of the library or against infringement. There is no warranty that our efforts or the library will fulfill any of your particular purposes or needs. This library is provided with all faults, and the entire risk of satisfactory quality, performance, accuracy, and effort is with the user.

libpng versions 0.97, January 1998, through 1.0.6, March 20, 2000, are Copyright (c) 1998, 1999, 2000 Glenn Randers-Pehrson, and are distributed according to the same disclaimer and license as libpng-0.96, with the following individuals added to the list of Contributing

Authors:

Tom Lane
Glenn Randers-Pehrson
Willem van Schaik

libpng versions 0.89, June 1996, through 0.96, May 1997, are
Copyright (c) 1996, 1997 Andreas Dilger
Distributed according to the same disclaimer and license as libpng-0.88,
with the following individuals added to the list of Contributing

Authors:

John Bowler
Kevin Bracey
Sam Bushell
Magnus Holmgren
Greg Roelofs
Tom Tanner

libpng versions 0.5, May 1995, through 0.88, January 1996, are
Copyright (c) 1995, 1996 Guy Eric Schalnat, Group 42, Inc.

For the purposes of this copyright and license, "Contributing Authors"
is defined as the following set of individuals:

Andreas Dilger
Dave Martindale
Guy Eric Schalnat
Paul Schmidt
Tim Wegner

The PNG Reference Library is supplied "AS IS". The Contributing Authors
and Group 42, Inc. disclaim all warranties, expressed or implied,
including, without limitation, the warranties of merchantability and of
fitness for any purpose. The Contributing Authors and Group 42, Inc.
assume no liability for direct, indirect, incidental, special,
exemplary,
or consequential damages, which may result from the use of the PNG
Reference Library, even if advised of the possibility of such damage.

Permission is hereby granted to use, copy, modify, and distribute this
source code, or portions hereof, for any purpose, without fee, subject
to the following restrictions:

1. The origin of this source code must not be misrepresented.
2. Altered versions must be plainly marked as such and must not
be misrepresented as being the original source.
3. This Copyright notice may not be removed or altered from
any source or altered source distribution.

The Contributing Authors and Group 42, Inc. specifically permit, without
fee, and encourage the use of this source code as a component to
supporting the PNG file format in commercial products. If you use this
source code in a product, acknowledgment is not required but would be
appreciated.

Component: OpenJPEG

Copyright (c) 2002-2007, Communications and Remote Sensing Laboratory,
Universite catholique de Louvain (UCL), Belgium
Copyright (c) 2002-2007, Professor Benoit Macq
Copyright (c) 2001-2003, David Janssens
Copyright (c) 2002-2003, Yannick Verschuere
Copyright (c) 2003-2007, Francois-Olivier Devaux and Antonin Descampe
Copyright (c) 2005, Herve Drolon, FreeImage Team
Copyright (c) 2006-2007, Parvatha Elangovan
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS 'AS
IS'
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
THE
POSSIBILITY OF SUCH DAMAGE.

Component: libtiff

Copyright (c) 1988-1997 Sam Leffler
Copyright (c) 1991-1997 Silicon Graphics, Inc.

Permission to use, copy, modify, distribute, and sell this software and
its documentation for any purpose is hereby granted without fee,
provided
that (i) the above copyright notices and this permission notice appear
in
all copies of the software and related documentation, and (ii) the
names of
Sam Leffler and Silicon Graphics may not be used in any advertising or
publicity relating to the software without the specific, prior written
permission of Sam Leffler and Silicon Graphics.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY

WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Component: libmodplug (Module)

This source code is public domain.

Component: OnError

Copyright (c) 2003 Siegfried Rings and Sebastian Lackner
All rights reserved.

Component: udis86 (OnError)

Copyright (c) 2002-2009 Vivek Thampi
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS

SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Component: brieflz

Copyright (c) 2002-2004 by Joergen Ibsen / Jibz
All Rights Reserved

<http://www.ibsensoftware.com/>

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Component: jcalg1

This software is provided as-is, without warranty of ANY KIND, either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. The author shall NOT be held liable for ANY damage to you, your computer, or to anyone or anything else, that may result from its use, or misuse. Basically, you use it at YOUR OWN RISK.

Component: lzma

LZMA SDK is written and placed in the public domain by Igor Pavlov.

Some code in LZMA SDK is based on public domain code from another developers:

- 1) PPMd var.H (2001): Dmitry Shkarin
- 2) SHA-256: Wei Dai (Crypto++ library)

Component: libzip

Copyright (C) 1999-2008 Dieter Baron and Thomas Klausner
The authors can be contacted at <libzip@nih.at>

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The names of the authors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS ‘‘AS IS’’ AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Component: pcre

PCRE LICENCE

PCRE is a library of functions to support regular expressions whose syntax and semantics are as close as possible to those of the Perl 5 language.

Release 8 of PCRE is distributed under the terms of the "BSD" licence, as specified below. The documentation for PCRE, supplied in the "doc" directory, is distributed under the same terms as the software itself. The data in the testdata directory is not copyrighted and is in the public domain.

The basic library functions are written in C and are freestanding. Also included in the distribution is a set of C++ wrapper functions, and a just-in-time compiler that can be used to optimize pattern matching.

These are both optional features that can be omitted when the library is built.

THE BASIC LIBRARY FUNCTIONS

Written by: Philip Hazel

Email local part: ph10
Email domain: cam.ac.uk

University of Cambridge Computing Service,
Cambridge, England.

Copyright (c) 1997-2020 University of Cambridge
All rights reserved.

PCRE JUST-IN-TIME COMPILATION SUPPORT

Written by: Zoltan Herczeg
Email local part: hzmester
Email domain: freemail.hu

Copyright(c) 2010-2020 Zoltan Herczeg
All rights reserved.

STACK-LESS JUST-IN-TIME COMPILER

Written by: Zoltan Herczeg
Email local part: hzmester
Email domain: freemail.hu

Copyright(c) 2009-2020 Zoltan Herczeg
All rights reserved.

THE C++ WRAPPER FUNCTIONS

Contributed by: Google Inc.

Copyright (c) 2007-2012, Google Inc.
All rights reserved.

THE "BSD" LICENCE

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

* Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

* Neither the name of the University of Cambridge nor the name of Google
Inc. nor the names of their contributors may be used to endorse or
promote products derived from this software without specific prior
written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Component: scintilla

License for Scintilla and SciTE

Copyright 1998-2003 by Neil Hodgson <neilh@scintilla.org>

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

NEIL HODGSON DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL NEIL HODGSON BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Component: expat

Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd
and Clark Cooper

Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006 Expat maintainers.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included

in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Component: miniaudio

Copyright 2023 David Reid

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Component: libogg

Copyright (c) 2002, Xiph.org Foundation

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Xiph.org Foundation nor the names of its contributors may be used to endorse or promote products derived from

this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FOUNDATION OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Component: libvorbis

Copyright (c) 2002-2004 Xiph.org Foundation

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Xiph.org Foundation nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FOUNDATION OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Component: neuquant

NeuQuant Neural-Net Quantization Algorithm Interface

Copyright (c) 1994 Anthony Dekker

NEUQUANT Neural-Net quantization algorithm by Anthony Dekker, 1994.
See "Kohonen neural networks for optimal colour quantization"
in "Network: Computation in Neural Systems" Vol. 5 (1994) pp 351-367.

for a discussion of the algorithm.

See also <http://members.ozemail.com.au/~dekker/NEUQUANT.HTML>

Any party obtaining a copy of these files from the author, directly or indirectly, is granted, free of charge, a full and unrestricted irrevocable, world-wide, paid up, royalty-free, nonexclusive right and license to deal in this software and documentation files (the "Software"), including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons who receive copies from any such party to do so, with the only requirement being that this copyright notice remain intact.

Modified to quantize 32bit RGBA images for the pngnq program.

Also modified to accept a number of colors argument.

Copyright (c) Stuart Coyle 2004-2006

Rewritten by Kornel Lesinski (2009)

Euclidean distance, color matching dependent on alpha channel

and with gamma correction. code refreshed for modern

compilers/architectures:

ANSI C, floats, removed pointer tricks and used arrays and structs.

Component: libmariadb

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <<https://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work. 1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL. 2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of

your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
- c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
- d) Do one of the following:
 - 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
 - 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.
- e) Provide Installation Information, but only if

you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
- b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License â or any later versionâ applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public

License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

Chapter 43

Licenses for the 3D engine integrated with PureBasic

This program makes use of the following components:

Component: OGRE

OGRE (www.ogre3d.org) is made available under the MIT License.

Copyright (c) 2000-2012 Torus Knot Software Ltd

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Component: CEGUI

Copyright (C) 2004 - 2006 Paul D Turner & The CEGUI Development Team

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Component: bullet

Copyright (c) 2003-2010 Erwin Coumans
<http://continuousphysics.com/Bullet/>

This software is provided 'as-is', without any express or implied warranty.

In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim

that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

Component: FreeImage

FreeImage Public License - Version 1.0

1. Definitions.

1.1. "Contributor" means each entity that creates or contributes to the creation of Modifications.

1.2. "Contributor Version" means the combination of the Original Code, prior Modifications used by a Contributor, and the Modifications made by that particular Contributor.

1.3. "Covered Code" means the Original Code or Modifications or the combination of the Original Code and Modifications, in each case including portions thereof.

1.4. "Electronic Distribution Mechanism" means a mechanism generally accepted in the software development community for the electronic transfer of data.

1.5. "Executable" means Covered Code in any form other than Source Code.

1.6. "Initial Developer" means the individual or entity identified as the Initial Developer in the Source Code notice required by Exhibit A.

1.7. "Larger Work" means a work which combines Covered Code or portions thereof with code not governed by the terms of this License.

1.8. "License" means this document.

1.9. "Modifications" means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:

A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.

B. Any new file that contains any part of the Original Code or previous Modifications.

1.10. "Original Code" means Source Code of computer software code which is described in the Source Code notice required by Exhibit A as Original Code, and which, at the time of its release under this License is not already Covered Code governed by this License.

1.11. "Source Code" means the preferred form of the Covered Code for making modifications to it, including all modules it contains, plus any associated

interface definition files, scripts used to control compilation and installation of an Executable, or a list of source code differential comparisons against either the Original Code or another well known, available Covered Code of the Contributor's choice. The Source Code can be in a compressed or archival form, provided the appropriate decompression or de-archiving software is widely available for no charge.

1.12. "You" means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License or a future version of this License issued under Section 6.1. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of fifty percent (50%) or more of the outstanding shares or beneficial ownership of such entity.

2. Source Code License.

2.1. The Initial Developer Grant. The Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

(a) to use, reproduce, modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, or as part of a Larger Work; and

(b) under patents now or hereafter owned or controlled by Initial Developer, to make, have made, use and sell ("Utilize") the Original Code (or portions thereof), but solely to the extent that any such patent is reasonably necessary to enable You to Utilize the Original Code (or portions thereof) and not to any greater extent that may be necessary to Utilize further Modifications or combinations.

2.2. Contributor Grant. Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

(a) to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof) either on an unmodified basis, with other Modifications, as Covered Code or as part of a Larger Work; and

(b) under patents now or hereafter owned or controlled by Contributor, to Utilize the Contributor Version (or portions thereof), but solely to the extent that any such patent is reasonably necessary to enable You to Utilize the Contributor Version (or portions thereof), and not to any greater extent that may be necessary to Utilize further Modifications or combinations.

3. Distribution Obligations.

3.1. Application of License. The Modifications which You create or to which You contribute are governed by the terms of this License, including without limitation Section 2.2. The Source Code version of Covered Code may be distributed only under the terms of this License or a future version of this License released under Section 6.1, and You must include a copy of this License with every copy of the Source Code You distribute. You may not offer or impose any terms on any Source Code version that alters or restricts the applicable version of this License or the recipients' rights hereunder. However, You may include an additional document offering the additional rights described in Section 3.5.

3.2. Availability of Source Code. Any Modification which You create *or to which* You contribute must be made available in Source Code form under the terms of this License either on the same media *as* an Executable version *or via* an accepted Electronic Distribution Mechanism *to* anyone *to* whom you made an Executable version available; *and if made available via Electronic Distribution Mechanism*, must remain available *for* at least twelve (12) months after the *date* it initially became available, *or* at least six (6) months after a subsequent version of that particular Modification has been made available *to* such recipients. You are responsible *for* ensuring that the Source Code version

remains available even if the Electronic Distribution Mechanism is maintained by a third party.

3.3. Description of Modifications. You must cause all Covered Code to which you contribute to contain a file documenting the changes You made to create that Covered Code and the date of any change. You must include a prominent statement that the Modification is derived, directly or indirectly, from Original Code provided by the Initial Developer and including the name of the Initial Developer in (a) the Source Code, and (b) in any notice in an Executable version or related documentation in which You describe the origin or ownership of the Covered Code.

3.4. Intellectual Property Matters

(a) Third Party Claims. If You have knowledge that a party claims an intellectual property right in particular functionality or code (or its utilization under this License), you must include a text file with the source code distribution titled "LEGAL" which describes the claim and the party making the claim in sufficient detail that a recipient will know whom to contact. If you obtain such knowledge after You make Your Modification available as described in Section 3.2, You shall promptly modify the LEGAL file in all copies You make available thereafter and shall take other steps (such as notifying appropriate mailing lists or newsgroups) reasonably calculated to inform those who received the Covered Code that new knowledge has been obtained.

(b) Contributor APIs. If Your Modification is an application programming interface and You own or control patents which are reasonably necessary to implement that API, you must also include this information in the LEGAL file.

3.5. Required Notices. You must duplicate the notice in Exhibit A in each file of the Source Code, and this License in any documentation for the Source Code, where You describe recipients' rights relating to Covered Code. If You created one or more Modification(s), You may add your name as a Contributor to the notice described in Exhibit A. If it is not possible to put such notice in a

particular Source Code file due to its structure, then you must include such notice in a location (such as a relevant directory file) where a user would be likely to look for such a notice. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Code. However, You may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear than any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.6. Distribution of Executable Versions. You may distribute Covered Code in Executable form only if the requirements of Section 3.1-3.5 have been met for that Covered Code, and if You include a notice stating that the Source Code version of the Covered Code is available under the terms of this License, including a description of how and where You have fulfilled the obligations of Section 3.2. The notice must be conspicuously included in any notice in an Executable version, related documentation or collateral in which You describe recipients' rights relating to the Covered Code. You may distribute the Executable version of Covered Code under a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable version does not attempt to limit or alter the recipient's rights in the Source Code version from the rights set forth in this License. If You distribute the Executable version under a different license You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or any Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.7. Larger Works. You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Code.

4. Inability to Comply Due to Statute or Regulation.

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Code due to statute or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be included in the LEGAL file described in Section 3.4 and must be included with all distributions of the Source Code. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Application of this License.

This License applies to code to which the Initial Developer has attached the notice in Exhibit A, and to related Covered Code.

6. Versions of the License.

6.1. New Versions. Floris van den Berg may publish revised and/or new versions of the License from time to time. Each version will be given a distinguishing version number.

6.2. Effect of New Versions. Once Covered Code has been published under a particular version of the License, You may always continue to use it under the terms of that version. You may also choose to use such Covered Code under the terms of any subsequent version of the License published by Floris van den Berg. No one other than Floris van den Berg has the right to modify the terms applicable to Covered Code created under this License.

6.3. Derivative Works. If you create or use a modified version of this License (which you may only do in order to apply it to code which is not already Covered Code governed by this License), you must (a) rename Your license so that the phrases "FreeImage", "FreeImage Public License", "FIPL", or any

confusingly
similar phrase do not appear anywhere in your license and (b) otherwise
make it
clear that your version of the license contains terms which differ
from the
FreeImage Public License. (Filling in the name of the Initial
Developer,
Original Code or Contributor in the notice described in Exhibit A
shall not of
themselves be deemed to be modifications of this License.)

7. DISCLAIMER OF WARRANTY.

COVERED CODE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS"
BASIS, WITHOUT
WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING,
WITHOUT
LIMITATION, WARRANTIES THAT THE COVERED CODE IS FREE OF DEFECTS,
MERCHANTABILITY,
FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. THE ENTIRE RISK AS
TO THE
QUALITY AND PERFORMANCE OF THE COVERED CODE IS WITH YOU. SHOULD ANY
COVERED CODE
PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR
ANY OTHER
CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR
CORRECTION.
THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS
LICENSE. NO
USE OF ANY COVERED CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS
DISCLAIMER.

8. TERMINATION.

This License and the rights granted hereunder will terminate
automatically if
You fail to comply with terms herein and fail to cure such breach
within 30 days
of becoming aware of the breach. All sublicenses to the Covered Code
which are
properly granted shall survive any termination of this License.
Provisions
which, by their nature, must remain in effect beyond the termination
of this
License shall survive.

9. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT
(INCLUDING
NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL THE INITIAL DEVELOPER,
ANY OTHER
CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED CODE, OR ANY SUPPLIER OF ANY
OF SUCH
PARTIES, BE LIABLE TO YOU OR ANY OTHER PERSON FOR ANY INDIRECT,
SPECIAL,
INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING,
WITHOUT
LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER

FAILURE OR
MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN
IF SUCH
PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH
DAMAGES. THIS
LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR
PERSONAL
INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT
APPLICABLE LAW
PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE
EXCLUSION OR
LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THAT
EXCLUSION AND
LIMITATION MAY NOT APPLY TO YOU.

10. U.S. GOVERNMENT END USERS.

The Covered Code is a "commercial item," as that term is defined in
48 C.F.R.
2.101 (Oct. 1995), consisting of "commercial computer software" and
"commercial
computer software documentation," as such terms are used in 48
C.F.R. 12.212
(Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1
through
227.7202-4 (June 1995), all U.S. Government End Users acquire Covered
Code with
only those rights set forth herein.

11. MISCELLANEOUS.

This License represents the complete agreement concerning subject
matter hereof.
If any provision of this License is held to be unenforceable, such
provision
shall be reformed only to the extent necessary to make it
enforceable. This
License shall be governed by Dutch law provisions (except to
the extent
applicable law, if any, provides otherwise), excluding its
conflict-of-law
provisions. With respect to disputes in which at least one party is
a citizen
of, or an entity chartered or registered to do business in, the The
Netherlands:
(a) unless otherwise agreed in writing, all disputes relating to
this License
(excepting any dispute relating to intellectual property rights)
shall be
subject to final and binding arbitration, with the losing party paying
all costs
of arbitration; (b) any arbitration relating to this Agreement shall be
held in
Almelo, The Netherlands; and (c) any litigation relating to this
Agreement shall
be subject to the jurisdiction of the court of Almelo, The Netherlands
with the
losing party responsible for costs, including without limitation,
court costs

and reasonable attorneys fees and expenses. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License.

12. RESPONSIBILITY FOR CLAIMS.

Except in cases where another Contributor has failed to comply with Section 3.4, You are responsible for damages arising, directly or indirectly, out of Your utilization of rights under this License, based on the number of copies of Covered Code you made available, the revenues you received from utilizing such rights, and other relevant factors. You agree to work with affected parties to distribute responsibility on an equitable basis.

EXHIBIT A.

"The contents of this file are subject to the FreeImage Public License Version 1.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://home.wxs.nl/~flvdborg/freeimage-license.txt>

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

Component: FreeType

The FreeType Project LICENSE

2006-Jan-27

Copyright 1996-2002, 2006 by
David Turner, Robert Wilhelm, and Werner Lemberg

Introduction =====

The FreeType Project is distributed in several archive packages; some of them may contain, in addition to the FreeType font engine, various tools and contributions which rely on, or relate to, the FreeType Project.

This license applies to all files found in such packages, and

which do not fall under their own explicit license. The license affects thus the FreeType font engine, the test programs, documentation and makefiles, at the very least.

This license was inspired by the BSD, Artistic, and IJG (Independent JPEG Group) licenses, which all encourage inclusion and use of free software in commercial and freeware products alike. As a consequence, its main points are that:

- o We don't promise that this software works. However, we will be interested in any kind of bug reports. ('as is' distribution)
- o You can use this software for whatever you want, in parts or full form, without having to pay us. ('royalty-free' usage)
- o You may not pretend that you wrote this software. If you use it, or only parts of it, in a program, you must acknowledge somewhere in your documentation that you have used the FreeType code. ('credits')

We specifically permit and encourage the inclusion of this software, with or without modifications, in commercial products. We disclaim all warranties covering The FreeType Project and assume no liability related to The FreeType Project.

Finally, many people asked us for a preferred form for a credit/disclaimer to use in compliance with this license. We thus encourage you to use the following text:

```
""
Portions of this software are copyright © <year> The FreeType
Project (www.freetype.org). All rights reserved.
""
```

Please replace <year> with the value from the FreeType version you actually use.

Legal Terms

=====

0. Definitions

Throughout this license, the terms 'package', 'FreeType Project', and 'FreeType archive' refer to the set of files originally distributed by the authors (David Turner, Robert Wilhelm, and Werner Lemberg) as the 'FreeType Project', be they named as alpha, beta or final release.

'You' refers to the licensee, or person using the project, where 'using' is a generic term including compiling the project's source code as well as linking it to form a 'program' or 'executable'. This program is referred to as 'a program using the FreeType engine'.

This license applies to all files distributed in the original FreeType Project, including all source code, binaries and

documentation, unless otherwise stated in the file in its original, unmodified form as distributed in the original archive. If you are unsure whether or not a particular file is covered by this license, you must contact us to verify this.

The FreeType Project is copyright (C) 1996-2000 by David Turner, Robert Wilhelm, and Werner Lemberg. All rights reserved except as specified below.

1. No Warranty

THE FREETYPE PROJECT IS PROVIDED 'AS IS' WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL ANY OF THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES CAUSED BY THE USE OR THE INABILITY TO USE, OF THE FREETYPE PROJECT.

2. Redistribution

This license grants a worldwide, royalty-free, perpetual and irrevocable right and license to use, execute, perform, compile, display, copy, create derivative works of, distribute and sublicense the FreeType Project (in both source and object code forms) and derivative works thereof for any purpose; and to authorize others to exercise some or all of the rights granted herein, subject to the following conditions:

- o Redistribution of source code must retain this license file ('FTL.TXT') unaltered; any additions, deletions or changes to the original files must be clearly indicated in accompanying documentation. The copyright notices of the unaltered, original files must be preserved in all copies of source files.
- o Redistribution in binary form must provide a disclaimer that states that the software is based in part of the work of the FreeType Team, in the distribution documentation. We also encourage you to put an URL to the FreeType web page in your documentation, though this isn't mandatory.

These conditions apply to any software derived from or based on the FreeType Project, not just the unmodified files. If you use our work, you must acknowledge us. However, no fee need be paid to us.

3. Advertising

Neither the FreeType authors and contributors nor you shall use the name of the other for commercial, advertising, or promotional purposes without specific prior written permission.

We suggest, but do not require, that you use one or more of the following phrases to refer to this software in your documentation or advertising materials: 'FreeType Project', 'FreeType Engine', 'FreeType library', or 'FreeType Distribution'.

As you have not signed this license, you are not required to accept it. However, as the FreeType Project is copyrighted material, only this license, or another one contracted with the authors, grants you the right to use, distribute, and modify it. Therefore, by using, distributing, or modifying the FreeType Project, you indicate that you understand and accept all the terms of this license.

4. Contacts

There are two mailing lists related to FreeType:

- o freetype@nongnu.org

Discusses general use and applications of FreeType, as well as future and wanted additions to the library and distribution. If you are looking for support, start in this list if you haven't found anything to help you in the documentation.

- o freetype-devel@nongnu.org

Discusses bugs, as well as engine internals, design issues, specific licenses, porting, etc.

Our home page can be found at

<http://www.freetype.org>

Component: libogg

Copyright (c) 2002, Xiph.org Foundation

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Xiph.org Foundation nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FOUNDATION OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Component: libvorbis

Copyright (c) 2002-2004 Xiph.org Foundation

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Xiph.org Foundation nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FOUNDATION OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Component: zlib

Copyright (C) 1995-2012 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source

distribution.

Jean-loup Gailly
jloup@gzip.org

Mark Adler
madler@alumni.caltech.edu

Component: pcre

PCRE LICENCE

PCRE is a library of functions to support regular expressions whose
syntax
and semantics are as close as possible to those of the Perl 5 language.

Release 8 of PCRE is distributed under the terms of the "BSD" licence,
as
specified below. The documentation for PCRE, supplied in the "doc"
directory, is distributed under the same terms as the software itself.
The data
in the testdata directory is not copyrighted and is in the public
domain.

The basic library functions are written in C and are freestanding. Also
included in the distribution is a set of C++ wrapper functions, and a
just-in-time compiler that can be used to optimize pattern matching.

These
are both optional features that can be omitted when the library is
built.

THE BASIC LIBRARY FUNCTIONS

Written by: Philip Hazel
Email local part: ph10
Email domain: cam.ac.uk

University of Cambridge Computing Service,
Cambridge, England.

Copyright (c) 1997-2020 University of Cambridge
All rights reserved.

PCRE JUST-IN-TIME COMPILATION SUPPORT

Written by: Zoltan Herczeg
Email local part: hzmester
Email domain: freemail.hu

Copyright(c) 2010-2020 Zoltan Herczeg
All rights reserved.

STACK-LESS JUST-IN-TIME COMPILER

Written by: Zoltan Herczeg
Email local part: hzmester
Email domain: freemail.hu

Copyright(c) 2009-2020 Zoltan Herczeg
All rights reserved.

THE C++ WRAPPER FUNCTIONS

Contributed by: Google Inc.

Copyright (c) 2007-2012, Google Inc.
All rights reserved.

THE "BSD" LICENCE

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

* Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

* Neither the name of the University of Cambridge nor the name of Google
Inc. nor the names of their contributors may be used to endorse or
promote products derived from this software without specific prior
written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
THE
POSSIBILITY OF SUCH DAMAGE.

Component: MeshMagick

Copyright (c) 2010 Daniel Wickert, Henrik Hinrichs, Sascha Kolewa,
Steve Streeting

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Component: OgreBullet

Copyright 2007 Paul "Tuan Kuran" Cheyrou-Lagrèze.

This file is part of OgreBullet an integration layer between the OGRE 3D graphics engine and the Bullet physic library.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING

FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN
THE SOFTWARE.

Component: OgreProcedural

This source file is part of ogre-procedural

For the latest info, see <http://code.google.com/p/ogre-procedural/>

Copyright (c) 2010 Michael Broutin

Permission is hereby granted, free of charge, to any person obtaining a
copy
of this software and associated documentation files (the "Software"),
to deal
in the Software without restriction, including without limitation the
rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or
sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included
in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN
THE SOFTWARE.

Components:

- OpenAL
 - OgreAL
 - zziplib
 - Hydrax
-

GNU LIBRARY GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1991 Free Software Foundation, Inc.
675 Mass Ave, Cambridge, MA 02139, USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the library GPL. It is

numbered 2 because it goes with version 2 of the ordinary GPL.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs.

This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is

the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program.

However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

GNU LIBRARY GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a

portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) The modified work must itself be a software library.
- b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
- d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If

identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License.

Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
- b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues),

conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our

decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Chapter 44

Macros

Syntax

```
Macro <name> [(Parameter [, ...])]  
...  
EndMacro
```

Description

Macros are a very powerful feature, mainly useful for advanced programmers. A macro is a placeholder for some code (one keyword, one line or even many lines), which will be directly inserted in the source code at the place where a macro is used. In this, it differs from procedures, as the procedures doesn't duplicate the code when they are called.

The `Macro` : `EndMacro` declaration must be done before the macro will be called for the first time.

Because macros will be completely replaced by their related code at compile time, they are not local to a procedure.

A macro can not have a return type nor typed parameters. When a macro has some parameters, they are replaced in the macro code by the literal expression which is passed to the called macro. No evaluation is done at this stage, which is very important to understand: the evaluation of a line is started once all the macros found on this line are expanded.

Macros are divided into two categories: simple (without parameters) and complex (with parameters, needs the parentheses when calling it). When using no parameters, it's possible to replace any word with another word (or any expression). Macros can't be called recursively. Macro declaration can't be nested (it's not possible to define a macro inside another one).

Example: Simple macro

```
1  Macro MyNot  
2      Not  
3  EndMacro  
4  
5  a = 0  
6  If MyNot a ; Here the line will be expanded to : 'If Not a'  
7      Debug "Ok"  
8  EndIf
```

When using parameters, it's possible to do very flexible macros. The special concatenation character `'#'` can be used to create new labels or keyword by mixing the macro code and the parameter expression (spaces are not accepted between each words by the concatenation character). It's also possible to define default values for parameters, so they can be omitted when calling the macro.

Example: Macro with parameter

```
1 Macro UMsgBox(Title, Body)
2   MessageRequester(Title, UCase(Body), 0)
3 EndMacro
4
5 Text$ = "World"
6 UMsgBox("Hello", "-" + Text$ + "-") ; Here the line will be expanded like
   that:
7                                     ; 'MessageRequester("Hello",
   UCase("-" + Text$ + "-"), 0)'
```

Example: Macro with default parameter

```
1 Macro UMsgBox(Title, Body = "Ha, no body specified")
2   MessageRequester(Title, UCase(Body), 0)
3 EndMacro
4
5 UMsgBox("Hello") ; Here the line will be expanded like that:
6                 ; 'MessageRequester("Hello", UCase("Ha, no body
   specified"), 0)'
```

Example: Macro parameter concatenation

```
1 Macro XCase(Type, Text)
2   Type#Case(Text)
3 EndMacro
4
5 Debug XCase(U, "Hello")
6 Debug XCase(L, "Hello")
```

Example: Advanced multi-line macro

```
1 Macro DoubleQuote
2   "
3 EndMacro
4
5 Macro Assert(Expression)
6   CompilerIf #PB_Compiler_Debugger ; Only enable assert in debug mode
7   If Expression
8     Debug "Assert (Line " + #PB_Compiler_Line + "): " +
       DoubleQuote#Expression#DoubleQuote
9   EndIf
10  CompilerEndIf
11 EndMacro
12
13 Assert(10 <> 10) ; Will display nothing
14 Assert(10 <> 15) ; Should display the assert
```

Syntax

UndefineMacro <name>

Description

[UndefineMacro](#) allows to undefine a previously defined macro, and redefine it in a different manner. Once the macro has been undefined, it is no more available for use.

Example: Undefine macro

```
1  Macro Test
2      Debug "1"
3  EndMacro
4
5  Test ; Call the macro
6
7  UndefineMacro Test ; Undefine the macro, it no more exists
8
9  Macro Test ; Now we can redefine the macro
10     Debug "2"
11 EndMacro
12
13 Test ; Call the macro
```

Syntax

MacroExpandedCount

Description

[MacroExpandedCount](#) allows to get the expanded count (number of time the macro has been expanded/called). It can be useful to generate unique identifiers in the same macro for every expansion (like label, procedure name etc.).

Example: Expanded count

```
1  Macro Test
2      Debug MacroExpandedCount
3  EndMacro
4
5  Test ; Call the macro
6  Test ; Call the macro
7  Test ; Call the macro
```

Chapter 45

Pointers and memory access

Pointers

Example

```
1 *MyScreen.Screen = OpenScreen(0, 320, 200, 8, 0)
2 mouseX = *MyScreen\MouseX ; Assuming the Screen structure contains a
  MouseX field
```

There are only three valid methods to set the value of a pointer:

- Get the result from a function (as shown in the above example)
- Copy the value from another pointer
- Find the address of a variable, procedure or label (as shown below)

Note: unlike C/C++, in PureBasic the '*' is **always** part of the item name. Therefore '*ptr' and 'ptr' are two different variables. 'ptr' is a variable (regular one) storing a value, '*ptr' is another variable of pointer type storing an address.

Pointers and memory size

Because pointers receive only addresses as values, the memory size of a pointer is the space allowing to store an absolute address of the processor:

- On 32-bit processors the address space is limited to 32-bit, so a pointer takes 32-bit (4 bytes, like a 'long') in memory
- On 64-bit processors it takes 64-bit (8 bytes, like a 'quad') in memory, because the absolute address is on a 64-bit range.

As a consequence the type of a pointer depends of the CPU address mode, ('long' on 32-bit CPU and 'quad' on 64-bit one for example), so a pointer is a variable of type pointer.

It results from this that assigning a native type to a pointer (*Pointer.l, *Pointer.b ...) makes no sense. Note:

- Every time a memory address needs to be stored in a variable, it should be done through a pointer. This guarantees address integrity at the compilation time whatever the CPU address mode is.
- PureBasic x86 does not generate 64-bit executables. For PureBasic programs compiled with this, the system grants them only an addressing with 32-bit pointers.

Pointers and structures

By assigning a structure to a pointer (for example *MyPointer.Point) it allows to access any memory address in a structured way (with the operator '\').

Example: Pointers and variables

```
1 Define Point1.Point, Point2.Point
2 *CurrentPoint.Point = @Point1 ; Pointer declaration, associated to a
  structure and initialized with Point1's address
```

```

3  *CurrentPoint \x = 10          ; Assign value 10 to Point1\x
4  *CurrentPoint.Point = @Point2 ; move to Point2's address
5  *CurrentPoint \x = 20          ; Assign value 20 to Point2\x
6  Debug Point1\x
7  Debug Point2\x

```

Example: Pointers and array

```

1  Define Point1.Point, Point2.Point
2  Dim *Points.Point(1) ; 2 slots array
3  *Points(0) = @Point1 ; Assign the first point variable to the first
   array slot
4  *Points(1) = @Point2 ; Same for second
5
6  *Points(0)\x = 10 ; Modify the variables through the pointers
7  *Points(1)\x = 20 ;
8
9  Debug Point1\x
10 Debug Point2\x

```

Pointers allow to move, to read and to write easily in memory. Furthermore they allow programmers to reach big quantities of data without supplementary cost further to data duplication. Copying a pointer is much faster.

Pointers are also available in structures, for more information see the structures chapter .

Pointers and character strings

All variables have a permanent size in memory (2 bytes for Word, 4 bytes for a Long, etc.) except for strings variables with lengths that can change. So string variables are managed by a different way of other variables.

Thus a structure field, that makes reference to a string, store only the memory address of the string instead of the string itself: a such structure field is a pointer towards a string.

Example

```

1  Text$ = "Hello"
2  *Text = @Text$          ; *Text store the address of the string in
   memory
3  *Pointer.String = @*Text ; *Pointer points on *Text
4  Debug *Pointer\s        ; Display the string living at the address
   stored in *Pointer (i.e. @Text$)

```

Pointers Arithmetic

Arithmetic operations on the pointers are possible and practical by using SizeOf() .

Example

```

1  Dim Array.Point(1)          ; Array of points
2
3  *Pointer.Point = @Array()    ; Store the array address
4  *Pointer\x = 10              ; Change the first array element values
5  *Pointer\y = 15
6
7  *Pointer + SizeOf(Point)     ; Move to the next array element
8

```

```

9      *Pointer\x = 7                ; Change the second array element values
10     *Pointer\y = 9
11
12     ; Display results
13     For i = 0 To 1
14         Debug Array(i)\x
15         Debug Array(i)\y
16     Next i

```

Addresses of variables

To get the address of a variable in your code, you use the at symbol (@). A common reason for using this is when you want to pass a structured type variable to a procedure . You must pass a pointer to this variable as you cannot pass structured variables directly.

Example

```

1      Structure astruct
2          a.w
3          b.l
4          c.w
5      EndStructure
6
7      Procedure SetB(*myptr.astruct)
8          *myptr\b = 69
9      EndProcedure
10
11     Define.astruct myvar
12
13     SetB(@myvar)
14
15     Debug myvar\b

```

Addresses of literal strings

To get the address of literal string, you can use the at symbol (@) in front of it. String constants are also supported.

Example

```

1      *String = @"Test"
2      Debug PeekC(*String) ; Will display 84, which is the value of 'T'

```

Addresses of procedures

For advanced programmers. The most common reason to get the address of a procedure is when dealing with the OS at a low-level. Some OSes allow you to specify callback or hook functions (for some operations) which get called by the OS and allows the programmer to extend the ability of the OS routine. The address of a procedure is found in a similar way to variables .

Example

```
1  Procedure WindowCB(WindowID.i, Message.i, wParam.i, lParam.i)
2      ; This is where the processing of your callback procedure would be
      performed
3  EndProcedure
4
5      ; A special callback for the Windows OS allowing you to process
      window events
6  SetWindowCallback( @WindowCB() )
```

Addresses of labels

It can also be useful to find the address of labels in your code. This can be because you want to access the code or data stored at that label, or any other good reason you can think of. To find the address of a label, you put a question mark (?) in front of the label name.

Example

```
1  Debug "Size of data file = " + Str(?endofmydata - ?mydata)
2
3  DataSection
4      mydata:
5          IncludeBinary "somefile.bin"
6      endofmydata:
7  EndDataSection
```

Chapter 46

Migration guide

Introduction

PureBasic is a modern programming language which evolves quickly to follow technologic changes and brings up-to-date commandset to the programmer. It sometimes involves to change or redesign some part of the language. While we try to keep these breaking changes to a minimum, it can happen and this migration guide will help to update your source from one version to another.

If you need stability instead of cutting edge features, we highly recommend to stick to 'LTS' (Long Term Support) version of PureBasic, which are released every 2 years, and are actively maintained for bug fixes.

Regular releases

migration from 5.50 to 5.60

migration from 5.40 to 5.50

migration from 5.30 to 5.40

LTS releases

migration from 5.60 to 5.72 LTS

migration from 5.20 LTS to 5.40 LTS

Chapter 47

Migration from PureBasic 5.20 LTS to 5.40 LTS

Billboard library

AddBillboard(): code change

```
1 ; Old
2 AddBillboard(Billboard, BillboardGroup, x, y, z)
3
4 ; New
5 Billboard = AddBillboard(BillboardGroup, x, y, z)
```

Cipher library

ExamineMD5Fingerprint(): code change

```
1 ; Old
2 ExamineMD5Fingerprint(#FingerPrint)
3
4 ; New
5 UseMD5Fingerprint()
6 StartFingerprint(#FingerPrint, #PB_Cipher_MD5)
```

ExamineSHA1Fingerprint(): code change

```
1 ; Old
2 ExamineSHA1Fingerprint(#FingerPrint)
3
4 ; New
5 UseSHA1Fingerprint()
6 StartFingerprint(#FingerPrint, #PB_Cipher_SHA1)
```

MD5FileFingerprint(): code change

```
1 ; Old
2 Result$ = MD5FileFingerprint(Filename$)
3
4 ; New
5 UseMD5Fingerprint()
6 Result$ = FileFingerprint(Filename$, #PB_Cipher_MD5)
```

MD5Fingerprint(): code change

```
1 ; Old
2 Result$ = MD5Fingerprint(*Buffer, Size)
3
4 ; New
5 UseMD5Fingerprint()
6 Result$ = Fingerprint(*Buffer, Size, #PB_Cipher_MD5)
```

SHA1FileFingerprint(): code change

```
1 ; Old
2 Result$ = SHA1FileFingerprint(Filename$)
3
4 ; New
5 UseSHA1Fingerprint()
6 Result$ = FileFingerprint(Filename$, #PB_Cipher_SHA1)
```

SHA1Fingerprint(): code change

```
1 ; Old
2 Result$ = SHA1Fingerprint(*Buffer, Size)
3
4 ; New
5 UseSHA1Fingerprint()
6 Result$ = Fingerprint(*Buffer, Size, #PB_Cipher_SHA1)
```

CRC32FileFingerprint(): code change

```
1 ; Old
2 Result = CRC32FileFingerprint(Filename$)
3
4 ; New
5 UseCRC32Fingerprint()
6 Result.1 = Val("$"+FileFingerprint(Filename$, #PB_Cipher_CRC32))
```

CRC32Fingerprint(): code change

```
1 ; Old
2 Result = CRC32Fingerprint(*Buffer, Size)
3
4 ; New
5 UseCRC32Fingerprint()
6 Result.1 = Val("$"+Fingerprint(*Buffer, Size, #PB_Cipher_CRC32))
```

NextFingerprint(): rename only

```
1 ; Old
2 NextFingerprint(#FingerPrint, *Buffer, Size)
3
4 ; New
5 AddFingerprintBuffer(#FingerPrint, *Buffer, Size)
```

Mail library

SendMail(): code change if the 'Asynchronous' parameter was used

```

1 ; Old
2 SendMail(#Mail, Smtip$, Port, 1)
3
4 ; New
5 SendMail(#Mail, Smtip$, Port, #PB_Mail_Aynchronous)

```

Packer library

RemovePackFile(): removed

PackerEntrySize(): `#PB_Packer_CompressedSize` support removed for ZIP and 7z archives

XML library

CreateXMLNode(): code change

```

1 ; Old
2 Node = CreateXMLNode(ParentNode)
3 SetXMLNodeName(Node, "Name")
4
5 ; New
6 Node = CreateXMLNode(ParentNode, "Name")

```

Screen library

AvailableScreenMemory() removed as new API doesn't support this info anymore. It was mostly returning '0' anyway.

Window library

`#PB_Event_SizeWindow` and `#PB_Event_MoveWindow` are no more realtime on Windows, use `BindEvent()` to get real time update.

Engine3D library

WorldCollisionAppliedImpulse() now returns a float about the applied impulse. GetX/Y/Z() are no more supported.

Various

DataSection label within Procedure are now local labels.

ASM local label prefix has been changed from "l_" to "ll_", to avoid possible clash with main labels.

`#PB_LinkedList` constant has been renamed to `#PB_List` for better consistency

Chapter 48

Migration from PureBasic 5.30 to 5.40

Cipher library

ExamineMD5Fingerprint(): code change

```
1 ; Old
2 ExamineMD5Fingerprint(#FingerPrint)
3
4 ; New
5 UseMD5Fingerprint()
6 StartFingerprint(#FingerPrint, #PB_Cipher_MD5)
```

ExamineSHA1Fingerprint(): code change

```
1 ; Old
2 ExamineSHA1Fingerprint(#FingerPrint)
3
4 ; New
5 UseSHA1Fingerprint()
6 StartFingerprint(#FingerPrint, #PB_Cipher_SHA1)
```

MD5FileFingerprint(): code change

```
1 ; Old
2 Result$ = MD5FileFingerprint(Filename$)
3
4 ; New
5 UseMD5Fingerprint()
6 Result$ = FileFingerprint(Filename$, #PB_Cipher_MD5)
```

MD5Fingerprint(): code change

```
1 ; Old
2 Result$ = MD5Fingerprint(*Buffer, Size)
3
4 ; New
5 UseMD5Fingerprint()
6 Result$ = Fingerprint(*Buffer, Size, #PB_Cipher_MD5)
```

SHA1FileFingerprint(): code change

```
1 ; Old
```

```

2   Result$ = SHA1FileFingerprint(Filename$)
3
4   ; New
5   UseSHA1FingerPrint()
6   Result$ = FileFingerprint(Filename$, #PB_Cipher_SHA1)

```

SHA1Fingerprint(): code change

```

1   ; Old
2   Result$ = SHA1Fingerprint(*Buffer, Size)
3
4   ; New
5   UseSHA1FingerPrint()
6   Result$ = Fingerprint(*Buffer, Size, #PB_Cipher_SHA1)

```

CRC32FileFingerprint(): code change

```

1   ; Old
2   Result = CRC32FileFingerprint(Filename$)
3
4   ; New
5   UseCRC32FingerPrint()
6   Result.1 = Val("$"+FileFingerprint(Filename$, #PB_Cipher_CRC32))

```

CRC32Fingerprint(): code change

```

1   ; Old
2   Result = CRC32Fingerprint(*Buffer, Size)
3
4   ; New
5   UseCRC32FingerPrint()
6   Result.1 = Val("$"+Fingerprint(*Buffer, Size, #PB_Cipher_CRC32))

```

NextFingerprint(): rename only

```

1   ; Old
2   NextFingerprint(#FingerPrint, *Buffer, Size)
3
4   ; New
5   AddFingerprintBuffer(#FingerPrint, *Buffer, Size)

```

Mail library

SendMail(): code change if the 'Asynchronous' parameter was used

```

1   ; Old
2   SendMail(#Mail, Smtip$, Port, 1)
3
4   ; New
5   SendMail(#Mail, Smtip$, Port, #PB_Mail_Asynchronous)

```

Packer library

RemovePackFile(): removed

PackerEntrySize(): #PB_Packer_CompressedSize support removed for ZIP and 7z archives

Screen library

AvailableScreenMemory() removed as new API doesn't support this info anymore. It was mostly returning '0' anyway.

Engine3D library

WorldCollisionAppliedImpulse() now returns a float about the applied impulse. GetX/Y/Z() are no more supported.

Chapter 49

Migration from PureBasic 5.40 to 5.50

Particule library

ParticleVelocity(): code changed to support current velocity.

```
1 ; Old
2 ParticleVelocity(#ParticleEmitter, Minimum, Maximum)
3
4 ; New
5 ParticleVelocity(#ParticleEmitter, Mode, Value)
```

Others

PureBasic internal manipulation of strings is now unicode only.

ASCII mode is no longer supported internally. The function `*Result = Ascii(String$)` allows the manipulation of ASCII strings by the user.

Chapter 50

Migration from PureBasic 5.50 to 5.60

Cipher library

Base64Encoder: function renamed

```
1 ; Old
2 Base64Encoder ()
3
4 ; New
5 Base64EncoderBuffer ()
```

Base64Decoder: function renamed

```
1 ; Old
2 Base64Decoder ()
3
4 ; New
5 Base64DecoderBuffer ()
```

Others

'Define.b' standalone syntax, to change default type for untyped variables is now forbidden and will result in a 'Syntax error'. Removing the statement will solve this (ensure to type your untyped variables accordingly). Note: 'Define.b a, b, c' is still supported.

Chapter 51

Migration from PureBasic 5.60 to 5.72 LTS

Math library

Sign(): now returns an integer instead of a float.

```
1 ; Old
2 Var.d(f) = Sign(X)
3
4 ; New
5 Var.i = Sign(X)
```

Window library

PostEvent(): now returns a result as it could fail in some extreme case

Others

In MS WIndows, the default font changed from MS Shell Dlg to Segoe UI size 9.

Chapter 52

Module

Syntax

```
DeclareModule <name>
    ...
EndDeclareModule

Module <name>
    ...
EndModule

UseModule <name>
UnuseModule <name>
```

Description

Modules are an easy way to isolate code parts from the main code, allowing code reuse and sharing without risk of name conflict. In some other programming languages, modules are known as 'namespaces'. A module must have a [DeclareModule](#) section (which is the public interface) and an associated [Module](#) section (which is the implementation). Only items declared in the [DeclareModule](#) section will be accessible from outside the module. All the code put in the [Module](#) section will be kept private to this module. Items from main code like procedures, variables etc. won't be accessible inside the module, even if they are global. A module can be seen as a blackbox, an empty code sheet where item names can not conflict with the main code. It makes it easier to write specific code, as simple names can be reused within each module without risk of conflict.

Items allowed in the [DeclareModule](#) section can be the following: procedure (only procedure declaration allowed), structure, macro, variable, constant, enumeration, array, list, map and label.

To access a module item from outside the module, the module name has to be specified followed by the '::' separator. When explicitly specifying the module name, the module item is available everywhere in the code source, even in another module. All items in the [DeclareModule](#) section can be automatically imported into another module or in the main code using [UseModule](#). In the case of a module name conflict, the module items won't be imported and a compiler error will be raised. [UnuseModule](#) removes the module items. [UseModule](#) is not mandatory to access a module item, but the module name needs to be specified.

To share information between modules, a common module can be created and then used in every module which needs it. It's the common way have global data available for all modules.

Default items available in modules are all PureBasic commands, structure and constants. Therefore module items can not be named like internal PureBasic commands, structures or constants.

All code put inside [DeclareModule](#) and [Module](#) sections is executed like any other code when the program flow reaches the module.

When the statements [Define](#), [EnableExplicit](#), [EnableASM](#) are used inside a module, they have no effect outside the respective module, and vice versa.

Note: modules are not mandatory in PureBasic but are recommended when building big projects. They help to build more maintainable code, even if it is slightly more verbose than module-free code. Having a [DeclareModule](#) section make the module pretty much self-documented for use when reusing and sharing it.

Example

```
1
2 ; Every items in this sections will be available from outside
3 ;
4 DeclareModule Ferrari
5     #FerrariName$ = "458 Italia"
6
7     Declare CreateFerrari()
8 EndDeclareModule
9
10 ; Every items in this sections will be private. Every names can be
    used without conflict
11 ;
12 Module Ferrari
13
14     Global Initialized = #False
15
16     Procedure Init() ; Private init procedure
17         If Initialized = #False
18             Initialized = #True
19             Debug "InitFerrari()"
20         EndIf
21     EndProcedure
22
23     Procedure CreateFerrari()
24         Init()
25         Debug "CreateFerrari()"
26     EndProcedure
27
28 EndModule
29
30
31 Procedure Init() ; Main init procedure, doesn't conflict with the
    Ferrari Init() procedure
32     Debug "Main init()"
33 EndProcedure
34
35 Init()
36
37 Ferrari::CreateFerrari()
38 Debug Ferrari::#FerrariName$
39
40 Debug "-----"
41
42 UseModule Ferrari ; Now import all public item directly in the main
    program scope
43
44 CreateFerrari()
45 Debug #FerrariName$
```

Example: With a common module

```
1
2 ; The common module, which will be used by others to share data
3 ;
4 DeclareModule Cars
5   Global NbCars = 0
6 EndDeclareModule
7
8 Module Cars
9 EndModule
10
11 ; First car module
12 ;
13 DeclareModule Ferrari
14 EndDeclareModule
15
16 Module Ferrari
17   UseModule Cars
18
19   NbCars+1
20 EndModule
21
22 ; Second car module
23 ;
24 DeclareModule Porsche
25 EndDeclareModule
26
27 Module Porsche
28   UseModule Cars
29
30   NbCars+1
31 EndModule
32
33 Debug Cars::NbCars
```

Chapter 53

NewList

Syntax

```
NewList name.<type>()
```

Description

`NewList` allows to declare a new dynamic list. Each element of the list is allocated dynamically. There are no element limits, so there can be as many as needed. A list can have any Variables standard or structured type. To view all commands used to manage lists, see the List library.

The new list are always locals, which means Global or Shared commands have to be used if a list declared in the main source need to be used in procedures. It is also possible to pass a list as parameter to a procedure by using the keyword `List`.

For fast swapping of list contents the `Swap` keyword is available.

Example: Simple list

```
1  NewList MyList()
2
3  AddElement(MyList())
4  MyList() = 10
5
6  AddElement(MyList())
7  MyList() = 20
8
9  AddElement(MyList())
10 MyList() = 30
11
12 ForEach MyList()
13     Debug MyList()
14 Next
```

Example: List as procedure parameter

```
1  NewList Test()
2
3  ; Adding the first 2 items to the list:
4  AddElement(Test())
5  Test() = 1
```

```

6  AddElement(Test())
7  Test() = 2
8
9  Procedure DebugList(count, List ParameterList())
10 ; Adding further items to the list:
11 For i=1 To count
12     AddElement(ParameterList())
13     ParameterList() = i
14 Next
15
16 ; Display all items in the list:
17 i = 1
18 ForEach ParameterList()
19     MessageRequester("List", "Entry" + Str(i) + " = " +
20     Str(ParameterList()))
21     i + 1
22 Next
23 EndProcedure
24
25 ; Add another 3 items to the list and display the list contents:
26 DebugList(3, Test())

```


Chapter 54

NewMap

Syntax

```
NewMap name.<type>([Slots])
```

Description

[NewMap](#) allows to declare a new map, also known as hashtable or dictionary. It allows to quickly reference an element based on a key. Each key in the map are unique, which means it can't have two distinct elements with the same key. There are no element limits, so there can be as many as needed. A map can have any standard or structured type. To view all commands used to manage maps, see the [Map](#) library.

When using a new key, a new element is automatically added to the map even without an assignment. If another element with the same key is already in the map, it will be replaced by the new one. Once an element has been accessed or created, it becomes the current element of the map, and further access to this element can be done without specifying the key. This is useful when using structured map, as no more element lookup is needed to access different structure field.

New maps are always locals by default, so Global or Shared commands have to be used if a map declared in the main source needs to be used in procedures. It is also possible to pass a map as parameter to a procedure by using the keyword [Map](#).

For fast swapping of map elements the Swap keyword is available.

The optional 'Slots' parameter defines how much slots the map will have to store its elements. The more slots it has, the faster it will be to access an element, but the more memory it will use. It's a tradeoff depending on how many elements the map will ultimately contain and how fast the random access should be. The default value is 512. This parameter has no impact about the number of elements a map can contain.

Example: Simple map

```
1  NewMap Country.s()
2
3  Country("GE") = "Germany"
4  Country("FR") = "France"
5  Country("UK") = "United Kingdom"
6
7  Debug Country("FR")
8
9  ForEach Country()
10     Debug Country()
11 Next
```

Example: Map as procedure parameter

```
1  NewMap Country.s()
2
3  Country("GE") = "Germany"
4  Country("FR") = "France"
5  Country("UK") = "United Kingdom"
6
7  Procedure DebugMap(Map ParameterMap.s())
8
9      ParameterMap("US") = "United States"
10
11     ForEach ParameterMap()
12         Debug ParameterMap()
13     Next
14
15 EndProcedure
16
17 DebugMap(Country())
```

Example: Structured map

```
1  Structure Car
2      Weight.l
3      Speed.l
4      Price.l
5  EndStructure
6
7  NewMap Cars.Car()
8
9  ; Here we use the current element after the new insertion
10 ;
11 Cars("Ferrari F40")\Weight = 1000
12 Cars()\Speed = 320
13 Cars()\Price = 500000
14
15 Cars("Lamborghini Gallardo")\Weight = 1200
16 Cars()\Speed = 340
17 Cars()\Price = 700000
18
19 ForEach Cars()
20     Debug "Car name: "+MapKey(Cars())
21     Debug "Weight: "+Str(Cars()\Weight)
22 Next
```

Example: Item created without assignment

```
1  NewMap IconMap()
2
3  IconMap("1") = 1
4  IconMap("2") = 2
5
6  Debug FindMapElement(IconMap(), "3") ; Displays 0 because the item
   does not exist
7
```

```
8 | If IconMap("3") : EndIf ; PureBasic creates the item without
   | assignment because a new key is detected
9 |
10| Debug FindMapElement(IconMap(), "3") ; Displays the memory address of
   | the element
```

Chapter 55

Others Commands

Syntax

```
Goto <label>
```

Description

This command is used to transfer the program directly to the labels position. Be cautious when using this function, as incorrect use could cause a program to crash...

Note: To exit a loop safely, you must always use Break instead of Goto, and never use it inside a Select/EndSelect block (unless you have the ability to correctly manage the stack yourself).

Syntax

```
End [ExitCode]
```

Description

Ends the program execution correctly. The 'ExitCode' optional parameter can be specified if the program need to returns an error code (widely used in console programs).

The 'ExitCode' can be further used e.g. with the ProgramExitCode() command.

Syntax

```
Swap <expression>, <expression>
```

Description

Swaps the value of the both expression, in an optimized way. The both <expression> have to be a variable , an array element, a list element or a map element (structured or not) and have to be one of the PureBasic native type like long (.l), quad (.q), string etc.

Example: Swapping of strings

```
1 Hello$ = "Hello"
2 World$ = "World"
3
4 Swap Hello$, World$
5
6 Debug Hello$+" "+World$
```

Example: Swapping of multi-dimensional arrays elements

```
1 Dim Array1(5,5)
2 Dim Array2(5,5)
3 Array1(2,2) = 10 ; set initial contents
4 Array2(3,3) = 20
5
6 Debug Array1(2,2) ; will print 10
7 Debug Array2(3,3) ; will print 20
8
9 Swap Array1(2,2) , Array2(3,3) ; swap 2 arrays elements
10
11 Debug "Array contents after swapping:"
12 Debug Array1(2,2) ; will print 20
13 Debug Array2(3,3) ; will print 10
```

Chapter 56

Procedures

Syntax

```
Procedure[.<type>] name(<parameter1[.<type>]> [, <parameter2[.<type>]  
    [= DefaultValue]>, ...])  
    ...  
    [ProcedureReturn value]  
EndProcedure
```

Description

A **Procedure** is a part of code independent from the main code which can have any parameters and its own variables . In PureBasic, a recurrence is fully supported for the procedures and any procedure can call it itself. At each call of the procedure the variables inside will start with a value of 0 (null). To access main code variables, they have to be shared by using Shared or Global keywords (see also the Protected and Static keywords).

The last parameters can have a default value (needs to be a constant expression), so if these parameters are omitted when the procedure is called, the default value will be used.

Arrays can be passed as parameters using the **Array** keyword, lists using the **List** keyword and maps using the **Map** keyword.

A procedure can return a value or a string if necessary. You have to set the type after **Procedure** and use the **ProcedureReturn** keyword at any moment inside the procedure. A call of **ProcedureReturn** exits immediately the procedure, even when its called inside a loop.

ProcedureReturn can't be used to return an array , list or a map . For this purpose pass the array, the list or the map as parameter to the procedure.

If no value is specified for **ProcedureReturn**, the return value will be zero if the compiler uses the C backend and for all processors managed by PureBasic, or the content of the 'eax' register in 32 bits (X86 processors) or 'rax' in 64 bits (X64 processors) if the compiler uses the ASM back-end, (see for more information).

Note: To return strings from DLLs, see DLLs . For advanced programmers **ProcedureC** is available and will declare the procedure using 'CDecl' instead of 'StandardCall' calling convention. For more information on library calling standards (cdecl, stdcall, fastcall), see [here](#).

Selected procedures can be executed asynchronously to the main program by using of threads .

Example: Procedure with a numeric variable as return-value

```
1  Procedure Maximum(nb1, nb2)  
2      If nb1 > nb2  
3          Result = nb1  
4      Else  
5          Result = nb2
```

```

6      EndIf
7
8      ProcedureReturn Result
9  EndProcedure
10
11  Result = Maximum(15, 30)
12  Debug Result

```

Example: Procedure with a string as return-value

```

1  Procedure.s Attach(String1$, String2$)
2      ProcedureReturn String1$+" "+String2$
3  EndProcedure
4
5  Result$ = Attach("PureBasic", "Coder")
6  Debug Result$

```

Example: Parameter with default value

```

1  Procedure a(a, b, c=2)
2      Debug c
3  EndProcedure
4
5  a(10, 12)          ; 2 will be used as default value for 3rd parameter
6  a(10, 12, 15)

```

Example: List as parameter

```

1  NewList Test.Point()
2
3  AddElement(Test())
4  Test()\x = 1
5  AddElement(Test())
6  Test()\x = 2
7
8  Procedure DebugList(c.l, List ParameterList.Point())
9
10     AddElement(ParameterList())
11     ParameterList()\x = 3
12
13     ForEach ParameterList()
14         MessageRequester("List", Str(ParameterList()\x))
15     Next
16
17 EndProcedure
18
19 DebugList(10, Test())

```

Example: Array as parameter

```

1 Dim Table.Point(10, 15)
2
3 Table(0,0)\x = 1
4 Table(1,0)\x = 2
5
6 Procedure TestIt(c.l, Array ParameterTable.Point(2)) ; The table
   support 2 dimensions
7
8     ParameterTable(1, 2)\x = 3
9     ParameterTable(2, 2)\x = 4
10
11 EndProcedure
12
13 TestIt(10, Table())
14
15 MessageRequester("Table", Str(Table(1, 2)\x))

```

Example: Static, dynamic array and passing a Structure to a Procedure

```

1 Structure Whatever
2     a.l
3     b.l[2] ; Static array (Standard C) with 2 values b[0] and
   b[1], not resizable
4     Array c.l(3,3) ; Dynamic array with 16 values c(0,0) to c(3,3),
   resizable with ReDim()
5 EndStructure
6
7 MyVar.Whatever
8
9 Procedure MyProcedure(*blahblah.Whatever)
10     *blahblah\a = 5
11     *blahblah\b[0] = 1
12     *blahblah\b[1] = 2
13     *blahblah\c(3,3) = 33
14 EndProcedure
15
16 MyProcedure(@MyVar)
17 Debug MyVar\a
18 Debug MyVar\b[0]
19 Debug MyVar\b[1]
20 Debug MyVar\c(3,3)

```

Example: Call a function by its name

```

1 Prototype Function()
2
3 Runtime Procedure Function1()
4     Debug "I call Function1 by its name"
5 EndProcedure
6
7 Procedure LaunchProcedure(Name.s)
8     Protected ProcedureName.Function = GetRuntimeInteger(Name + "()")
9     ProcedureName()
10 EndProcedure

```



```
11 |
12 | LaunchProcedure("Function1") ; Display "I call Function1 by its name"
```

Syntax

```
Declare[.<type>] name(<parameter1[.<type>]> [, <parameter2[.<type>] [=
    DefaultValue]>, ...])
```

Description

Sometimes a procedure need to call another procedure which isn't declared before its definition. This is annoying because the compiler will complain 'Procedure <name> not found'. [Declare](#) can help in this particular case by declaring only the header of the procedure. Nevertheless, the [Declare](#) and real [Procedure](#) declaration must be identical (including the correct type and optional parameters, if any). For advanced programmers [DeclareC](#) is available and will declare the procedure using 'CDecl' instead of 'StandardCall' calling convention.

Example

```
1 | Declare Maximum(Value1, Value2)
2 |
3 | Procedure Operate(Value)
4 |     Maximum(10, 2) ; At this time, Maximum() is unknown.
5 | EndProcedure
6 |
7 | Procedure Maximum(Value1, Value2)
8 |     ProcedureReturn 0
9 | EndProcedure
```

Chapter 57

Protected

Syntax

```
Protected[.<type>] <variable[.<type>]> [= <expression>] [, ...]
```

Description

Protected allows a variable to be accessed only in a Procedure even if the same variable has been declared as Global in the main program. **Protected** in its function is often known as 'Local' from other BASIC dialects. Each variable can have a default value directly assigned to it. If a type is specified after **Protected**, the default type is changed for this declaration. **Protected** can also be used with arrays , lists and maps .

The value of the local variable will be reinitialized at each procedure call. To avoid this, you can use the keyword **Static** , to separate global from local variables while keeping their values.

Example: With variable

```
1  Global a
2  a = 10
3
4  Procedure Change()
5      Protected a
6      a = 20
7  EndProcedure
8
9  Change()
10 Debug a ; Will print 10, as the variable has been protected.
```

Example: With array

```
1  Global Dim Array(2)
2  Array(0) = 10
3
4  Procedure Change()
5      Protected Dim Array(2) ; This array is protected, it will be local.
6      Array(0) = 20
7  EndProcedure
8
9  Change()
```

10 | `Debug Array(0) ; Will print 10, as the array has been protected.`

Chapter 58

Prototypes

Syntax

```
Prototype.<type> name(<parameter>, [, <parameter> [= DefaultValue]...])
```

Description

For advanced programmers. **Prototype** allows to declare a type which will map a function. It's useful when used with a variable, to do a function pointer (the variable value will be the address the function to call, and can be changed at will).

This feature can replace the `OpenLibrary()` and `CallFunction()` sequence as it has some advantages: type checking is done, number of parameters is validated.

Unlike `CallFunction()`, it can deal with double, float and quad variables without any problem. To get easily the pointer of a library function, use `GetFunction()`.

The last parameters can have a default value (need to be a constant expression), so if these parameters are omitted when the function is called, the default value will be used.

By default the function will use the standard call convention (`stdcall` on x86, or `fastcall` on x64). If the function pointer will be a C one, the **PrototypeC** variant should be used instead. For more information on library calling standards (`cdecl`, `stdcall`, `fastcall`), see [here](#).

The pseudotypes can be used for the parameters, but not for the returned value.

Example

```
1  CompilerIf #PB_Compiler_OS <> #PB_OS_Windows
2      CompilerError "This sample only works on Windows"
3  CompilerEndIf
4
5  Prototype.i ProtoMessageBox(Window.i, Body$, Title$, Flags.i = 0)
6
7  If OpenLibrary(0, "User32.dll")
8
9      ; 'MsgBox' is a variable with a 'ProtoMessageBox' type
10     ;
11     MsgBox.ProtoMessageBox = GetFunction(0, "MessageBoxW")
12
13     MsgBox(0, "Hello", "World") ; We don't specify the flags
14 EndIf
```

Example: With pseudotypes

```
1  CompilerIf #PB_Compiler_OS <> #PB_OS_Windows
2      CompilerError "This sample only works on Windows"
3  CompilerEndIf
4
5  ; We use the 'p-ascii' pseudotype for the string parameters, as
6  ; MessageBoxA() is an ascii only function. The compiler will
7  ; automatically converts the strings to unicode when needed.
8  ;
9  Prototype.i ProtoMessageBoxA(Window.i, Body.p-ascii, Title.p-ascii,
    Flags.i = 0)
10
11  If OpenLibrary(0, "User32.dll")
12
13      ; 'MsgBox' is a variable with a 'ProtoMessageBoxA' type
14      ;
15      MsgBox.ProtoMessageBoxA = GetFunction(0, "MessageBoxA")
16
17      MsgBox(0, "Hello", "World") ; We don't specify the flags
18  EndIf
```

Chapter 59

Pseudotypes

Description

For advanced programmers. The pseudotypes are a way to ease the programming when dealing with external libraries which need datatypes that are not internally supported by PureBasic. In this case, it is possible to use the predefined pseudotype which will do the appropriate conversion on the fly, without extra work. As they are not 'real' types, the chosen naming scheme is explicitly different: a 'p-' prefix (for 'pseudo') is part of the type name. The available pseudotypes are:

`p-ascii`: acts as a string type, but will always convert the string to `ascii` before calling the function, even when the program is compiled in `unicode` mode.

It is very useful when accessing a `shared` library which doesn't have unicode support in an unicode program for example.

`p-utf8`: acts as a string type, but will always convert the string to `utf8` before calling the function. It is very useful when accessing a shared library which needs its unicode

string be passed as UTF8 instead of `ascii` or `unicode` strings.

`p-bstr`: acts as a string type, but will always convert the string to `bstr` before calling the function. It is very useful when accessing a shared library which needs `bstr` parameters, like COM components.

`p-unicode`: acts as a string type, but will always convert the string to `unicode` before calling the function, even when the program is compiled in `ascii` mode.

It is very useful when accessing a shared library which only supports unicode in an `ascii` program for example.

`p-variant`: acts as a numeric type, will adjust the function call to use the `variant` parameter

correctly. It is very useful when accessing a shared library which needs 'variant' parameters, like COM components.

The pseudotypes can only be used with prototypes, interfaces and imported functions. The pseudotypes does the appropriate conversion only if it is necessary.

Example

```
1  Import "User32.lib"
2
3      ; We use the 'p-ascii' pseudotype for the string parameters, as
4      ; MessageBoxA() is an ASCII only function. The compiler will
5      ; automatically converts the strings to ASCII when needed.
6      ;
7      MessageBoxA(Window.i, Body.p-ascii, Title.p-ascii, Flags.i = 0)
8
9  EndImport
10
11      ; It will work correctly even if the internal PureBasic strings are
12      ; in unicode
13      ; because the compiler will take care of the ASCII conversion
14      ; automatically.
15      ;
16      MessageBoxA(0, "Hello", "World")
```

Chapter 60

PureBasic objects

Introduction

The purpose of this section is to describe the behavior, creation, and handling of objects in PureBasic. For the demonstration, we will use the Image object, but the same logic applies to all other PureBasic objects. When creating an Image object, we can do it in two ways: indexed and dynamic.

I. Indexed numbering

The static, indexed way, allows you to reference an object by a predefined numeric value. The first available index number is 0 and subsequent indexes are allocated sequentially. This means that if you use the number 0 and then the number 1000, 1001 indexes will be allocated and 999 (from 1 to 999) will be unused, which is not an efficient way to use indexed objects. If you need a more flexible method, use the dynamic way of allocating objects, as described in section II. The indexed way offers several advantages:

- Easier handling, since no variables or arrays are required.
- 'Group' processing, without the need to use an intermediate array.
- Use the object in procedures without declaring anything in global (if using a constant or a number).
- An object that is associated with an index is automatically freed when reusing that index.

The maximum index number is limited to an upper bound, depending of the object type (usually from 5000 to 60000). Enumerations are strongly recommended if you plan to use sequential constants to identify your objects (which is also recommended).

Example

```
1 CreateImage(0, 640, 480) ; Create an image, the n°0
2 ResizeImage(0, 320, 240) ; Resize the n°0 image and change its
  @ReferenceLink "handles" "handle"
```

Example

```
1 CreateImage(2, 640, 480) ; Create an image, the n°2
```



```

2 | ResizeImage(2, 320, 240) ; Resize the n°2 image and change its
   | @ReferenceLink "handles" "handle"
3 | CreateImage(2, 800, 800) ; Create a new image in the n°2 index, the
   | old one is automatically free'ed

```

Example

```

1 | For k = 0 To 9
2 |   CreateImage(k, 640, 480) ; Create 10 different images, numbered
   | from 0 to 9
3 |   ResizeImage(k, 320, 240) ; Resize images into half width/height
   | and change its @ReferenceLink "handles" "handle"
4 | Next

```

Example

```

1 | #ImageBackground = 0
2 | #ImageButton     = 1
3 |
4 | CreateImage(#ImageBackground, 640, 480) ; Create an image (n°0)
5 | ResizeImage(#ImageBackground, 320, 240) ; Resize the background image
6 | CreateImage(#ImageButton, 800, 800) ; Create an image (n°1)

```

II. Dynamic numbering

Sometimes, indexed numbering isn't very handy to handle dynamic situations where we need to deal with an unknown number of objects. PureBasic provides an easy and complementary way to create objects in a dynamic manner. Both methods (indexed and dynamic) can be used together at the same time without any conflict. To create a dynamic object, you just have to specify the `#PB_Any` constant instead of the indexed number, and the dynamic number will be returned as result of the function. Then just use this number with the other object functions in the place where you would use an indexed number (except to create a new object). This way of object handling can be very useful when used in combination with a list, which is also a dynamic way of storage.

Example

```

1 | DynamicImage1 = CreateImage(#PB_Any, 640, 480) ; Create a
   | dynamic image
2 | ResizeImage(DynamicImage1, 320, 240) ; Resize the
   | DynamicImage1

```

A complete example of dynamic objects and lists can be found [here](#):
 Further description and an example of dynamic numbering multiple windows and gadgets can be found in the related 'Beginners chapter'.

Overview of the different PureBasic objects

Different PureBasic objects (windows, gadgets, sprites, etc.) can use the same range of object numbers again. So the following objects can be enumerated each one starting at 0 (or other value) and PureBasic differs them by their type:

- Database
- Dialog
- Entity
- File
- FTP
- Gadget (including the ScintillaGadget())
- Gadget3D
- Image
- Library
- Light
- Mail
- Material
- Menu (not MenuItem() , as this is no object)
- Mesh
- Movie
- Music
- Network
- Node
- Particle
- RegularExpression
- SerialPort
- Sound
- Sound3D
- Sprite
- StatusBar
- Texture
- ToolBar
- Window
- Window3D
- XML

Chapter 61

Building a PureLibrary

Introduction

PureBasic allows to easily create custom libraries to extend the core PureBasic commandset with more commands. The generated library will be located in the PureBasic/PureLibraries/UserLibraries/ folder, so be sure to have the proper access rights when trying to generate one.

For now, the purelibrary creation is only available from commandline, using the C backend compiler using the '-purelibrary' option. It's also possible to use the great library creation tool from the IDE by 'Pf Shadoko' found [here](#) which also come with some more features.

As a PureLibrary adds new command to the standard commandset, the new command name must be different than the internal commands. A PureLibrary should always been compiled with the PureBasic compiler it will be used on. If a PureLibrary doesn't come with it's associated code source, it will be very likely to break in a future version of PureBasic and should be avoided.

A PureLibrary can also be created using C/C++ or ASM, please look in the PureLibrary/SDK/ folder.

Exporting functions

No code should be written outside procedures, except for object declaration.

To export a function when creating a PureLibrary, it needs to be declared as **ProcedureDLL**. If optional parameters are needed another function with the same name but with an incremented number can be used. A 'QuickHelp' comment can also be added to have the quick help displayed in the IDE when using the command.

Example:

```
1 ; QuickHelp MyMax(Min [, Max [, Flags, Mode]]) - A standard min/max
  function
2 ProcedureDLL MyMax3(a, b, c, d)
3 ; Your code here
4 ProcedureReturn a
5 EndProcedure
6
7 ProcedureDLL MyMax2(a, b)
8 ProcedureReturn MyMax3(a, b, 0, 0)
9 EndProcedure
10
11 ProcedureDLL MyMax(a)
12 ProcedureReturn MyMax2(a, 0)
13 EndProcedure
```

Automatic functions

There are two special functions for automatic initialization and to free the library: `InitPureLibrary()` and `FreePureLibrary()`. Unlike other library functions, these are not declared with [ProcedureDLL](#) but with [Procedure](#). These two functions are automatically called when the program is launched and when the program is terminated.

Example:

```
1  Procedure InitPureLibrary()  
2      ; Your init routine here  
3  EndProcedure  
4  
5  Procedure FreePureLibrary()  
6      ; Your free routine here  
7  EndProcedure
```

Disabling a PureLibrary

When coding the PureLibrary, it might be useful to ignore the current library with the [DisablePureLibrary](#) so the function name can be used again.

Example:

```
1  DisablePureLibrary MyCoolLib  
2  
3      ; All functions found in 'MyCoolLib' will be ignored, so their name  
      can be used again with ProcedureDLL
```

Removing a PureLibrary

To remove a custom PureLibrary, just delete the according file in the `PureBasic/PureLibraries/UserLibraries/` folder.

Chapter 62

Repeat : Until

Syntax

```
Repeat
  ...
Until <expression> [or Forever]
```

Description

This function will loop until the <expression> becomes true. The number of loops is unlimited. If an endless loop is needed then use the **Forever** keyword instead of **Until**. With the Break command, it is possible to exit the **Repeat : Until** loop during any iteration. With Continue command, the end of the current iteration may be skipped.

Example

```
1  a=0
2  Repeat
3    a=a+1
4  Until a>100
```

This will loop until "a" takes a value > to 100, (it will loop 101 times).

Chapter 63

Residents

Description

Residents are precompiled files which are loaded when the compiler starts. They can be found in the 'residents' folder of the PureBasic installation path. A resident file must have the extension '.res' and can contain the following items: structures , interfaces , prototypes , macros and constants . It can not contain dynamic code or procedures .

When a resident is loaded, all its content is available for the program being compiled. That's why all built-in constants like `#PB_Event_CloseWindow` are available, they are in the 'PureBasic.res' file. All the API structures and constants are also in a resident file. Using residents is a good way to store the common macros, structure and constants so they will be available for every programs. When distributing an user library, it's also a nice solution to provide the needed constants and structures, as PureBasic does.

To create a new resident, the command-line compiler needs to be used, as there is no option to do it from the IDE. It is often needed to use `/IGNORERESIDENT` and `/CREATERESIDENT` at the same time to avoid duplicate errors, as the previous version of the resident is loaded before creating the new one.

Residents greatly help to have a faster compilation and compiler start, as all the information is stored in binary format. It is much faster to load than parsing an include file at every compilation.

Chapter 64

Runtime

Syntax

```
Runtime Variable
Runtime #Constant
Runtime Procedure() declaration
Runtime Enumeration declaration
```

Description

For advanced programmers. [Runtime](#) is used to create runtime accessible list of programming objects like variables, constants and procedures. Once compiled a program doesn't have variable, constant or procedure label anymore as everything is converted into binary code. [Runtime](#) enforces the compiler to add an extra reference for a specific object to have it available through the Runtime library. The objects can be manipulated using their string reference, even when the program is compiled.

To illustrate the use of [Runtime](#): the Dialog library use it to access the event procedure associated to a gadget . The name of the procedure to use for the event handler is specified in the XML file (which is text format), and then the dialog library use `GetRuntimeInteger()` to resolve the procedure address at runtime. It's not needed to recompile the program to change it.

Another use would be adding a small realtime scripting language to the program, allowing easy modification of exposed variables, using runtime constants values. While it could be done manually by building a map of objects, the [Runtime](#) keyword allows to do it in a standard and unified way.

Example: Procedure

```
1 Runtime Procedure OnEvent()
2     Debug "OnEvent "
3 EndProcedure
4
5 Debug GetRuntimeInteger("OnEvent()") ; Will display the procedure
    address
```

Example: Enumeration

```
1 Runtime Enumeration
2     #Constant1 = 10
3     #Constant2
4     #Constant3
5 EndEnumeration
```

```

6
7   Debug GetRuntimeInteger("#Constant1")
8   Debug GetRuntimeInteger("#Constant2")
9   Debug GetRuntimeInteger("#Constant3")

```

Example: Variable

```

1   Define a = 20
2   Runtime a
3
4   Debug GetRuntimeInteger("a")
5   SetRuntimeInteger("a", 30)
6
7   Debug a ; the variable has been modified

```

Example: Call a function by its name

```

1   Prototype Function()
2
3   Runtime Procedure Function1()
4       Debug "I call Function1 by its name"
5   EndProcedure
6
7   Runtime Procedure Function2()
8       Debug "I call Function2 by its name"
9   EndProcedure
10
11  Procedure LaunchProcedure(Name.s)
12      Protected ProcedureName.Function = GetRuntimeInteger(Name + "()")
13      ProcedureName()
14  EndProcedure
15
16  LaunchProcedure("Function1") ; Display "I call Function1 by its name"
17  LaunchProcedure("Function2") ; Display "I call Function2 by its name"

```


Chapter 65

Select : EndSelect

Syntax

```
Select <expression1>
  Case <expression> [, <expression> [<numeric expression> To <numeric
    expression>]]
    ...
  [Case <expression>]
    ...
  [Default]
    ...
EndSelect
```

Description

Select provides the ability to determine a quick choice. The program will execute the <expression1> and retain its value in memory. It will then compare this value to all of the **Case** <expression> values and if a given **Case** <expression> value is true, it will then execute the corresponding code and quit the **Select** structure. **Case** supports multi-values and value ranges through the use of the optional **To** keyword (numeric values only). When using the **To** keyword, the range must be in ascending order (lower to higher). If none of the **Case** values are true, then the **Default** code will be executed (if specified). Note: **Select** will accept floats as <expression1> but will round them down to the nearest integer (comparisons will be done only with integer values).

Example: Simple example

```
1  Value = 2
2
3  Select Value
4    Case 1
5      Debug "Value = 1"
6
7    Case 2
8      Debug "Value = 2"
9
10   Case 20
11     Debug "Value = 20"
12
13   Default
14     Debug "I don't know"
15 EndSelect
```

Example: Multicase and range example

```
1  Value = 2
2
3  Select Value
4      Case 1, 2, 3
5          Debug "Value is 1, 2 or 3"
6
7      Case 10 To 20, 30, 40 To 50
8          Debug "Value is between 10 and 20, equal to 30 or between 40 and
9              50"
10
11     Default
12         Debug "I don't know"
13 EndSelect
```

Chapter 66

Using several PureBasic versions on Windows

Overview

It is possible to install several PureBasic versions on your hard disk at the same time. This is useful to finish one project with an older PureBasic version, and start developing a new project with a new PureBasic version.

How to do it

Create different folders like "PureBasic_v3.94" and "PureBasic_v4" and install the related PureBasic version in each folders.

When one "PureBasic.exe" is started, it will assign all ".pb" files with this version of PureBasic. So when a source code is loaded by double-clicking on the related file, the currently assigned PureBasic version will be started. Beside PureBasic will change nothing, which can affect other PureBasic versions in different folders.

To avoid the automatic assignment of ".pb" files when starting the IDE, a shortcut can be created from PureBasic.exe with "/NOEXT" as parameter. The command line options for the IDE are described here . **Note:** Since PureBasic 4.10, the settings for the IDE are no longer saved in the PureBasic directory but rather in the %APPDATA%\PureBasic directory. To keep the multiple versions from using the same configuration files, the /P /T and /A switches can be used. Furthermore there is the /PORTABLE switch which puts all files back into the PureBasic directory and disabled the creation of the .pb extension.

Chapter 67

Shared

Syntax

```
Shared <variable> [, ...]
```

Description

`Shared` allows a variable , an array , a list or a map to be accessed within a procedure . When `Shared` is used with an array, a list or a map, only the name followed by '()' must be specified.

Example: With variable

```
1  a = 10
2
3  Procedure Change()
4      Shared a
5      a = 20
6  EndProcedure
7
8  Change()
9  Debug a      ; Will print 20, as the variable has been shared.
```

Example: With array and list

```
1  Dim Array(2)
2  NewList List()
3  AddElement(List())
4
5  Procedure Change()
6      Shared Array(), List()
7      Array(0) = 1
8      List() = 2
9  EndProcedure
10
11 Change()
12 Debug Array(0) ; Will print 1, as the array has been shared.
13 Debug List()   ; Will print 2, as the list has been shared.
```

Chapter 68

Static

Syntax

```
Static[.<type>] <variable[.<type>]> [= <constant expression>] [, ...]
```

Description

Static allows to create a local persistent variable in a Procedure even if the same variable has been declared as Global in the main program. If a type is specified after **Static**, the default type is changed for this declaration. **Static** can also be used with arrays , lists and maps . When declaring a static array, the dimension parameter has to be a constant value.

The value of the variable isn't reinitialized at each procedure call, means you can use local variables parallel to global variables (with the same name), and both will keep their values. Each variable can have a default value directly assigned to it, but it has to be a constant value.

Beside **Static** you can use the keyword **Protected** , to separate global from local variables, but with **Protected** the local variables will not keep their values.

Example: With variable

```
1  Global a
2  a = 10
3
4  Procedure Change()
5      Static a
6      a+1
7      Debug "In Procedure: "+Str(a) ; Will print 1, 2, 3 as the variable
      increments at each procedure call.
8  EndProcedure
9
10 Change()
11 Change()
12 Change()
13 Debug a ; Will print 10, as the static variable doesn't affect global
    one.
```

Example: With array

```
1  Global Dim Array(2)
2  Array(0) = 10
```

```

3
4 Procedure Change()
5     Static Dim Array(2)
6     Array(0)+1
7     Debug "In Procedure: "+Str(Array(0)) ; Will print 1, 2, 3 as the
        value of the array field increments at each procedure call.
8 EndProcedure
9
10 Change()
11 Change()
12 Change()
13 Debug Array(0) ; Will print 10, as the static array doesn't affect
    global one.

```

Example: With multiple procedure

```

1 Procedure Foo()
2     Static x = 100 ; the declaration and the assignment is done only
        once at program start.
3
4     Debug x
5     x + 1
6 EndProcedure
7
8 Foo() ; Display 100
9 Foo() ; Display 101
10 Foo() ; Display 102
11
12 Debug "---"
13
14 Procedure Bar()
15     Static x ; the declaration is done only once at program start.
16     x = 100 ; the assignment is done on every Procedure call.
17
18     Debug x
19     x + 1
20 EndProcedure
21
22 Bar() ; Display 100
23 Bar() ; Display 100
24 Bar() ; Display 100

```

Chapter 69

Structures

Syntax

```
Structure <name> [Extends <name>] [Align <numeric constant expression>]  
...  
EndStructure
```

Description

Structure is useful to define user type, and access some OS memory areas. Structures can be used to enable faster and easier handling of data files. It is very useful as you can group into the same object the information which are common. Structures fields are accessed with the `\` option. Structures can be nested. Static arrays are supported inside structures.

Structure fields must have an explicit type among all Basic Types supported by PureBasic, i.e. Byte (.b), Ascii (.a), Character (.c), Word (.w), Unicode (.u), Long (.l), Integer (.i), Float (.f), Quad (.q), Double (.d), String (.s) and Fixed String (.s{ Length}).

Dynamic objects like arrays, lists and maps are also supported inside structure and are automatically initialized when the object using the structure is created. To declare such field, use the following keywords: **Array**, **List** and **Map**.

Generally, structures are used in association with a variable, an array, a list, or a map. However, advanced users will be able to allocate a memory structure with `AllocateStructure()` and free it with `FreeStructure()`. It is also possible to initialize a structure in memory with `InitializeStructure()`, to copy it with `CopyStructure()`, reset it to zero with `ClearStructure()` and reinitialize it with `ResetStructure()`. It's possible to perform a full structure copy by using the equal affectation between two structure element of the same type.

The optional **Extends** parameter allows to extends another structure with new fields. All fields found in the extended structure will be available in the new structure and will be placed before the new fields. This is useful to do basic inheritance of structures.

For advanced users only. The optional **Align** parameter allows to adjust alignment between every structure field. The default alignment is 1, meaning no alignment. For example, if the alignment is set to 4, every field offset will be on a 4 byte boundary. It can help to get more performance while accessing structure fields, but it can use more memory, as some space between each fields will be wasted. The special value `#PB_Structure_AlignC` can be used to align the structure as it would be done in language C, useful when importing C structures to use with API functions.

`SizeOf` can be used with structures to get the size of the structure and `OffsetOf` can be used to retrieve the address offset of the specified field.

Please note, that in structures a **static array[]** doesn't behave like the normal BASIC array (defined using `Dim`) to be conform to the C/C++ structure format (to allow direct API structure porting). This means that `a[2]` will allocate an array from 0 to 1 where `Dim a(2)` will allocate an array from 0 to 2. And library functions `Array` don't work with them.

When using pointers in structures, the `*` has to be omitted when using the field, once more to ease API code porting. It can be seen as an oddity (and to be honest, it is) but it's like that since the very start of

PureBasic and many, many sources rely on that so it won't be changed.

When using a lot of structure fields you can use the With : EndWith keywords to reduce the amount of code to type and ease its readability.

Example

```
1  Structure Person
2      Name.s
3      ForName.s
4      Age.w
5  EndStructure
6
7  Dim MyFriends.Person(100)
8
9  ; Here the position '0' of the array MyFriend()
10 ; will contain one person and it's own information
11
12 MyFriends(0)\Name = "Andersson"
13 MyFriends(0)\Forname = "Richard"
14 MyFriends(0)\Age = 32
```

Example: A more complex structure (Nested and static array)

```
1  Structure Window
2      *NextWindow.Window ; Points to another window object
3      x.w
4      y.w
5      Name.s[10] ; 10 Names available (from 0 to 9)
6  EndStructure
```

Example: Extended structure

```
1  Structure MyPoint
2      x.l
3      y.l
4  EndStructure
5
6  Structure MyColoredPoint Extends MyPoint
7      color.l
8  EndStructure
9
10 ColoredPoint.MyColoredPoint\x = 10
11 ColoredPoint.MyColoredPoint\y = 20
12 ColoredPoint.MyColoredPoint\color = RGB(255, 0, 0)
```

Example: Structure copy

```
1  Structure MyPoint
2      x.l
3      y.l
4  EndStructure
5
```



```

6 LeftPoint.MyPoint\x = 10
7 LeftPoint\y = 20
8
9 RightPoint.MyPoint = LeftPoint
10
11 Debug RightPoint\x
12 Debug RightPoint\y

```

Example: Dynamic object

```

1 Structure Person
2     Name$
3     Age.l
4     List Friends$()
5 EndStructure
6
7 John.Person
8 John\Name$ = "John"
9 John\Age = 23
10
11 ; Now, add some friends to John
12 ;
13 AddElement(John\Friends$())
14 John\Friends$() = "Jim"
15
16 AddElement(John\Friends$())
17 John\Friends$() = "Monica"
18
19 ForEach John\Friends$()
20     Debug John\Friends$()
21 Next

```

Example: Static, dynamic array and passing a structure to a procedure

```

1 Structure Whatever
2     a.l
3     b.l[2] ; Static array (Standard C) with 2 values b[0] and
4             b[1], not resizable
5     Array c.l(3,3) ; Dynamic array with 16 values c(0,0) to c(3,3),
6                     resizable with ReDim()
7 EndStructure
8
9 MyVar.Whatever
10
11 Procedure MyProcedure(*blahblah.Whatever)
12     *blahblah\a = 5
13     *blahblah\b[0] = 1
14     *blahblah\b[1] = 2
15     *blahblah\c(3,3) = 33
16 EndProcedure
17
18 MyProcedure(@MyVar)
19 Debug MyVar\a
20 Debug MyVar\b[0]
21 Debug MyVar\b[1]

```

```

20  Debug MyVar\c(3,3)
21
22  ;Debug MyVar\c(0,10) ; Out-of-bounds index error
23  ReDim MyVar\c(3,10) ; Beware, only the last dimension can be resized!
24  Debug MyVar\c(0,10) ; Cool, the array is bigger now!

```

Example: Nested structure(s)

```

1  Structure pointF
2      x.f
3      y.f
4  EndStructure
5
6  Structure Field
7      Field1.q
8      Field2.s{6}
9      Field3.s
10     Array Tab.pointF(3)
11 EndStructure
12
13 Define MyVar.Field
14
15 MyVar\Tab(3)\x = 34.67

```

Example: Alignment

```

1  Structure Type Align 4
2      Byte.b
3      Word.w
4      Long.l
5      Float.f
6  EndStructure
7
8  Debug OffsetOf(Type\Byte) ; will print 0
9  Debug OffsetOf(Type\Word) ; will print 4
10 Debug OffsetOf(Type\Long) ; will print 8
11 Debug OffsetOf(Type\Float) ; will print 12

```

Example: Pointers

```

1  Structure Person
2      *Next.Person ; Here the '*' is mandatory to declare a pointer
3      Name$
4      Age.b
5  EndStructure
6
7  Timo.Person\Name$ = "Timo"
8  Timo\Age = 25
9
10 Fred.Person\Name$ = "Fred"
11 Fred\Age = 25
12
13 Timo\Next = @Fred ; When using the pointer, the '*' is omitted

```

```

14
15  Debug Timo\Next\Name$ ; Will print 'Fred'

```

Syntax

```

StructureUnion
    Field1.Type
    Field2.Type
    ...
EndStructureUnion

```

Description

Structure union are only useful for advanced programmers who want to save some memory by sharing some fields inside the same structure. It's like the 'union' keyword in C/C++.

Note: Each field in the [StructureUnion](#) declaration can be of a different type .

Example

```

1  Structure Type
2      Name$
3      StructureUnion
4          Long.l      ; Each field (Long, Float and Byte) resides at the
                        same address in memory.
5          Float.f      ;
6          Byte.b      ;
7      EndStructureUnion
8  EndStructure

```

Example: Example RGB

```

1  Structure StrColor
2      Red.a
3      Green.a
4      Blue.a
5      Alpha.a
6  EndStructure
7
8  Structure StrColorU
9      StructureUnion
10         Component.StrColor
11         Color.l
12         Byte.a[4]
13     EndStructureUnion
14 EndStructure
15
16 Define Color1.StrColorU
17
18 Color1\Color = RGBA($10,$20,$30,$FF)
19 Debug "hex = " + Hex(Color1\Color, #PB_Long) ;hex = FF302010
    (little-endian cpu)
20 Debug "r = " + Hex(Color1\Component\Red) ;r = 10
21 Debug "g = " + Hex(Color1\Component\Green) ;g = 20

```

```

22  Debug "b = " + Hex(Color1\Component\Blue)      ;b = 30
23  Debug "a = " + Hex(Color1\Component\Alpha)     ;a = FF
24
25  Debug "by0 = " + Hex(Color1\Byte[0])           ;by0 = 10
26  Debug "by1 = " + Hex(Color1\Byte[1])           ;by1 = 20
27  Debug "by2 = " + Hex(Color1\Byte[2])           ;by2 = 30
28  Debug "by3 = " + Hex(Color1\Byte[3])           ;by3 = FF

```

Example: Example date handling

```

1  Structure date ; jj.mm.yyyy
2      day.s{2}
3      dot.s{1}
4      month.s{2}
5      dot.s{1}
6      year.s{4}
7  EndStructure
8
9  Structure dateU
10     StructureUnion
11         s.s{10}
12         d.date
13     EndStructureUnion
14 EndStructure
15
16 Dim d1.dateU(365)
17
18 ;An array
19 d1(0)\s = "05.04.2028"
20
21 Debug d1(0)\d\day
22 Debug d1(0)\d\month
23 Debug d1(0)\d\year
24
25 ;A variable
26 d2.date2\s = "15.11.2030"
27
28 Debug d2\d\day
29 Debug d2\d\month
30 Debug d2\d\year

```

Chapter 70

Subsystems

Introduction

PureBasic integrated commandset relies on available OS libraries. Sometimes, there is different way to achieve the same goal and when it makes sense, PureBasic offers the possibility to change the used underlying libraries for specific commands, without changing one line of source code. For example, on Windows there is the 'DirectX11' subsystem available, which will use DirectX functions to render sprites, instead of OpenGL (which is the default subsystem). It can be useful to use DirectX instead of OpenGL when wanting a bit quicker rendering on Windows.

To enable a subsystem, its name has to be set in the IDE compiler options , or through the /SUBSYSTEM command-line switch. The subsystem name is not case-sensitive. This is a compile time option, which means an executable can not embed more than one subsystem at once for a specific library. If multiple support is needed (for example shipping an OpenGL and DirectX version of a game), two executables needs to be created.

The available subsystems are located in the PureBasic 'subsystems' folder. Any residents or libraries found in this drawer will have precedency over the default libraries and residents, when a subsystem is specified. Any number of different subsystems can be specified (as long it doesn't affect the same libraries).

The [Subsystem](#) compiler function can be used to detect if a specific subsystem is used for the compilation.

Available subsystems

Here is a list of available subsystems, and the affected libraries:

Windows

```
DirectX9: use DirectX 9 instead of OpenGL. Affected libraries:
- Sprite
- Screen
- Note: 3D engine no more available as it uses OpenGL
DirectX11: use DirectX 11 instead of OpenGL. Affected libraries:
- Sprite
- Screen
- Note: 3D engine no more available as it uses OpenGL
```

Linux

```
Gtk2: Use GTK2 instead of GTK3 for GUI toolkit. Affected libraries:
- 2D Drawing
- AudioCD
- Clipboard
- Desktop
```

- Drag & Drop
- Font
- Gadget
- Image
- Menu
- Movie
- Printer
- Requester
- Scintilla
- StatusBar
- SysTray
- Toolbar
- Window

Qt: Use QT instead of GTK3 for GUI toolkit. Affected libraries:

- 2D Drawing
- AudioCD
- Clipboard
- Desktop
- Drag & Drop
- Font
- Gadget
- Image
- Menu
- Movie
- Printer
- Requester
- Scintilla
- StatusBar
- SysTray
- Toolbar
- Window

MacOS X

None

Chapter 71

Threaded

Syntax

```
Threaded[.<type>] <variable[.<type>]> [= <constant expression>] [, ...]
```

Description

Threaded allows to create a thread based persistent variable , arrays (except multi-dimensional arrays), lists or maps . This means every thread will have its own version of the object. This is only useful when writing multithreaded programs. If a type is specified after **Threaded**, the default type is changed for this declaration.

Each variable can have a default value directly assigned to it, but it has to be a constant value.

Threaded initialization is done at thread first start. That implies when declaring and assigning a threaded variable in the same time, the variable is assigned for all threads. See example 2. When declaring a threaded array, the dimension parameter has to be a constant value.

A Threaded object can't be declared in a procedure, its scope is always global.

Example: 1 With variables

```
1  Threaded Counter
2
3  Counter = 128
4
5  Procedure Thread(Parameter)
6
7      Debug Counter ; Will display zero as this thread doesn't have used
      this variable for now
8      Counter = 256
9      Debug Counter ; Will display 256
10
11 EndProcedure
12
13 Thread = CreateThread(@Thread(), 0)
14 WaitThread(Thread) ; Wait for thread ending
15
16 Debug Counter ; Will display 128, even if Counter has been changed in
    the thread
```

Example: 2 All threads

```
1 Threaded Counter = 128 ; Defined for all threads
2
3 Procedure Thread(Parameter)
4
5     Debug Counter ; Will display 128 because a programme started,
        starts a thread too
6     Counter = 256
7     Debug Counter ; Will display 256
8
9 EndProcedure
10
11 Thread = CreateThread(@Thread(), 0)
12 WaitThread(Thread) ; Wait for thread ending
13
14 Debug Counter ; Will display 128, even if Counter has been changed in
        the thread
```


Chapter 72

UserGuide - Advanced functions

<empty for now>

UserGuide Navigation

< Previous: Other Library functions || Overview || Next: Some Tips & Tricks >

Chapter 73

UserGuide - Constants

In addition to variables PureBasic provides a method to define constants too. In fact it provides several. Well have a quick look at them now. Predefined constants - provided either by PureBasic itself (all begin with `#PB_`), or from the API for the operating system. The IDEs Structure Viewer tool has a panel which shows all the predefined constants. User defined constants - by defining a constant name with the prefix `#` you can provide your own constants to make code more readable.

```
1  #MyConstant1 = 10
2  #MyConstant2 = "Hello, World!"
```

Enumerations PureBasic will automatically number a series of constants sequentially in an Enumeration, by default enumerations will begin from zero but this can be altered, if desired.

```
1  Enumeration
2  ?? #MyConstantA
3  ?? #MyConstantB
4  ?? #MyConstantC
5  EndEnumeration
6
7  Enumeration 10 Step 5
8  ?? #MyConstantD
9  ?? #MyConstantE
10 ?? #MyConstantF
11 EndEnumeration
12
13 Debug #MyConstantA ; will be 0
14 Debug #MyConstantB ; will be 1
15 Debug #MyConstantC ; will be 2
16
17 Debug #MyConstantD ; will be 10
18 Debug #MyConstantE ; will be 15
19 Debug #MyConstantF ; will be 20
```

UserGuide Navigation

< Previous: Variables || Overview || Next: Decisions & Conditions >

Chapter 74

UserGuide - Storing data in memory

This example gathers information about the files in the logged on user's home directory into a structured list. For now the output isn't very exciting but we will come back to this example later on and make it a bit more friendly in several different ways.

```
1  ; This section describes the fields of a structure or record, mostly
   integers in this case,
2  ; but notice the string for the file name and the quad for the file
   size.
3  Structure FILEITEM
4      Name.s
5      Attributes.i
6      Size.q
7      DateCreated.i
8      DateAccessed.i
9      DateModified.i
10 EndStructure
11
12 ; Now we define a new list of files using the structure previously
   specified
13 ; and some other working variables we'll use later on.
14 NewList Files.FILEITEM()
15 Define.s Folder
16 Define.l Result
17
18 ; This function gets the home directory for the logged on user.
19 Folder = GetHomeDirectory()
20
21 ; Open the directory to enumerate all its contents.
22 Result = ExamineDirectory(0, Folder, " *.*")
23
24 ; If this is ok, begin enumeration of entries.
25 If Result
26     ; Loop through until NextDirectoryEntry(0) becomes zero -
       indicating that there are no more entries.
27     While NextDirectoryEntry(0)
28         ; If the directory entry is a file, not a folder.
29         If DirectoryEntryType(0) = #PB_DirectoryEntry_File
30             ; Add a new element to the list.
31             AddElement(Files())
32             ; And populate it with the properties of the file.
33             Files()\Name = DirectoryEntryName(0)
34             Files()\Size = DirectoryEntrySize(0)
```

```

35     Files()\Attributes = DirectoryEntryAttributes(0)
36     Files()\DateCreated = DirectoryEntryDate(0, #PB_Date_Created)
37     Files()\DateAccessed = DirectoryEntryDate(0, #PB_Date_Accessed)
38     Files()\DateModified = DirectoryEntryDate(0, #PB_Date_Modified)
39     EndIf
40     Wend
41     ; Close the directory.
42     FinishDirectory(0)
43 EndIf
44
45 ; Shows the results in the debug window (if there is no entry,
    nothing will be displayed)
46 ForEach Files()
47     Debug "Filename = " + Files()\Name
48     Debug "Size = " + Str(Files()\Size)
49     Debug "Attributes = " + StrU(Files()\Attributes)
50     Debug "Created = " + StrU(Files()\DateCreated)
51     Debug "Accessed = " + StrU(Files()\DateAccessed)
52     Debug "Modified = " + StrU(Files()\DateModified)
53 Next Files()

```

Ok, firstly, the dates in the output are just numbers - this isn't very helpful, so let's make them look a bit more familiar. Replace the last three Debug statements with these:

```

1     ...
2     Debug "Created = " + FormatDate("%dd/%mm/%yyyy",
    Files()\DateCreated)
3     Debug "Accessed = " + FormatDate("%dd/%mm/%yyyy",
    Files()\DateAccessed)
4     Debug "Modified = " + FormatDate("%dd/%mm/%yyyy",
    Files()\DateModified)

```

The FormatDate() function takes a date in PureBasic's own numeric date format and displays it in a format that we can specify. So now things should begin to look a bit more sensible. Finally, for now, the list isn't in any particular order, so let's sort the list before we display it. Add this line before the comment about showing the list and the ForEach loop.

```

1     ...
2
3     ; Sort the list into ascending alphabetical order of file name.
4     SortStructuredList(Files(), #PB_Sort_Ascending,
    OffsetOf(FILEITEM\Name), #PB_String)
5
6     ; If there are some entries in the list, show the results in the
    debug window.
7     ...

```

This command takes the structured list, and resorts it into ascending order (#PB_Sort_Ascending), of the Name field of the structure (OffsetOf(FILEITEM\Name)), which is a string value (#PB_String).

UserGuide Navigation

< Previous: String Manipulation || Overview || Next: Input & Output >

Chapter 75

UserGuide - Decisions & Conditions

There are different ways of processing data obtained from user input or other way (loading from a file, ...). The common arithmetic functions (+, -, *, /, ...) can be combined with conditions. You can use the If : Else/ElseIf : EndIf set of keywords or the Select : Case/Default : EndSelect keywords, just use what is the best for your situation!

This example shows the use of If : ElseIf : Else : EndIf for creating a message, possibly for showing in the status bar of a form (GUI) or something similar, based upon the number of items and filtered items in an, imaginary, list. Note that unlike some other BASIC languages, PureBasic doesn't use the "Then" keyword and that there is no space in the ElseIf and EndIf keywords.

```
1 Define.l Items = 10, Filter = 6
2 Define.s Message
3
4 If Items = 0
5     Message = "List is empty."
6
7 ElseIf Items = 1 And Filter = 0
8     Message = "One item. Not shown by filter."
9
10 ElseIf Items > 1 And Filter = 0
11     Message = StrU(Items) + " items. All filtered."
12
13 ElseIf Items > 1 And Filter = 1
14     Message = StrU(Items) + " items. One shown by filter."
15
16 ElseIf Items = Filter
17     Message = StrU(Items) + " items. None filtered."
18
19 Else
20     ; None of the other conditions were met.
21     Message = StrU(Items) + " items. " + StrU(Filter) + " shown by
    filter."
22
23 EndIf
24
25 Debug Message
```

This example shows the use of Select : Case : Default : EndSelect to categorize the first 127 ASCII characters into groups. The "For : Next" loop counts to 130 to demonstrate the Default keyword.

```
1 Define.c Char
2 Define.s Message
3
```

```

4  For Char = 0 To 130
5
6      Select Char
7
8          Case 0 To 8, 10 To 31, 127
9              Message = StrU(Char) + " is a non-printing control code."
10
11          Case 9
12              Message = StrU(Char) + " is a tab."
13
14          Case 32
15              Message = StrU(Char) + " is a space."
16
17          Case 36, 128
18              Message = StrU(Char) + " is a currency symbol. (" + Chr(Char) +
19                  " )"
20
21          Case 33 To 35, 37 To 47, 58 To 64, 91 To 96
22              Message = StrU(Char) + " is a punctuation mark or math symbol.
23                  (" + Chr(Char) + " )"
24
25          Case 48 To 57
26              Message = StrU(Char) + " is a numeral. (" + Chr(Char) + " )"
27
28          Case 65 To 90
29              Message = StrU(Char) + " is an upper case letter. (" +
30                  Chr(Char) + " )"
31
32          Case 97 To 122
33              Message = StrU(Char) + " is a lower case letter. (" + Chr(Char)
34                  + " )"
35
36          Default
37              ; If none of the preceding Case conditions are met.
38              Message = "Sorry, I don't know what " + StrU(Char) + " is!"
39
40      EndSelect
41
42      Debug Message
43
44  Next Char

```

UserGuide Navigation

< Previous: Constants || Overview || Next: Loops >

Chapter 76

UserGuide - Compiler directives (for different behavior on different OS)

This will be our last visit to the File Properties program. There is one limitation discussed previously to overcome and we've left it until now because it is a special case.

So far the Attributes column on the display has simply been an integer . This is because the return values of the GetFileAttributes() and DirectoryEntryAttributes() instructions have a different meaning on Windows systems compared with Mac and Linux systems.

We can't allow for this difference at run-time, however we can use Compiler Directives to have the program behave differently on the three different operating systems.

Add this new procedure declaration to that section.

```
1  Declare.s AttributeString(Attributes.l)
```

Add this new procedure to the implementation section.

```
1  Procedure.s AttributeString(Attributes.l)
2      ; Converts an integer attributes value into a string description.
3      ; Supports Linux, Mac and Windows system's attributes.
4
5      Protected.s Result
6
7      Result = ""
8
9      CompilerIf #PB_Compiler_OS = #PB_OS_Windows
10         ?? ??
11         ?? ?? ; These are the attributes for Windows systems.
12         ?? ?? ; A logical-and of the attribute with each constant filters
13         out that bit and can then be used for comparison.
14
15         ?? ?? If Attributes & #PB_FileSystem_Archive
16             ?? ?? ?? Result + "A"
17         ?? ?? Else
18             ?? ?? ?? Result + " "
19         ?? ?? EndIf
20
21         ?? ?? If Attributes & #PB_FileSystem_Compressed
22             ?? ?? ?? Result + "C"
23         ?? ?? Else
24             ?? ?? ?? Result + " "
25         ?? ?? EndIf
26
27         ?? ?? If Attributes & #PB_FileSystem_Hidden
```

```

27     ?? ?? ?? Result + "H"
28 ?? ?? Else
29     ?? ?? ?? Result + " "
30 ?? ?? EndIf
31 ?? ??
32 ?? ?? If Attributes & #PB_FileSystem_ReadOnly
33     ?? ?? ?? Result + "R"
34 ?? ?? Else
35     ?? ?? ?? Result + " "
36 ?? ?? EndIf
37 ?? ??
38 ?? ?? If Attributes & #PB_FileSystem_System
39     ?? ?? ?? Result + "S"
40 ?? ?? Else
41     ?? ?? ?? Result + " "
42 ?? ?? EndIf
43 ?? ??
44 CompilerElse
45
46     ; These are the attributes for Mac and Linux systems.
47
48     If Attributes & #PB_FileSystem_Link
49         Result + "L "
50     Else
51         Result + " "
52     EndIf
53
54     ; User attributes.
55     If Attributes & #PB_FileSystem_ReadUser
56         Result + "R"
57     Else
58         Result + " "
59     EndIf
60
61     If Attributes & #PB_FileSystem_WriteUser
62         Result + "W"
63     Else
64         Result + " "
65     EndIf
66
67     If Attributes & #PB_FileSystem_ExecUser
68         Result + "X "
69     Else
70         Result + " "
71     EndIf
72
73     ; Group attributes.
74     If Attributes & #PB_FileSystem_ReadGroup
75         Result + "R"
76     Else
77         Result + " "
78     EndIf
79
80     If Attributes & #PB_FileSystem_WriteGroup
81         Result + "W"
82     Else
83         Result + " "
84     EndIf
85

```



```

86     If Attributes & #PB_FileSystem_ExecGroup
87         Result + "X "
88     Else
89         Result + "  "
90     EndIf
91
92     ; All attributes.
93     If Attributes & #PB_FileSystem_ReadAll
94         Result + "R"
95     Else
96         Result + "  "
97     EndIf
98
99     If Attributes & #PB_FileSystem_WriteAll
100         Result + "W"
101     Else
102         Result + "  "
103     EndIf
104
105     If Attributes & #PB_FileSystem_ExecAll
106         Result + "X"
107     Else
108         Result + "  "
109     EndIf
110
111     CompilerEndIf
112
113     ; Return the result.
114     ProcedureReturn Result
115
116 EndProcedure

```

Finally, replace these two lines in the ListLoad procedure

```

1     ; Convert the attributes to a string, for now.
2     Attrb = StrU(Files()\Attributes)

```

with these,

```

1     ; Call AttributeString to convert the attributes to a string
    representation.
2     Attrb = AttributeString(Files()\Attributes)

```

Now when the program is executed a string display will be shown instead of the integer values. On a Windows system it would look something like this (assuming all attributes are set):

```

1     ACHRS

```

and on the other two systems:

```

1     L RWX RWX RWX

```

The CompilerIf instruction works much like an If instruction - however it is the compiler that makes the decision at compile-time, rather than the executable at run-time. This means that we can include different code to handle the file attributes on the different operating systems.

The lines between:

```

1     CompilerIf #PB_Compiler_OS = #PB_OS_Windows

```

and

```
1 CompilerElse
```

will be compiled on Windows systems. The constant `#PB_Compiler_OS` is automatically defined by PureBasic to allow this kind of program logic.

The other section will be used on Mac and Linux systems - which work the same way, conveniently. If these operating systems had different attribute values too, then we could use `CompilerSelect` and `CompilerCase` to make a three-way determination.

```
1 CompilerSelect #PB_Compiler_OS
2
3 CompilerCase #PB_OS_Linux
4     ; Code for Linux systems.
5
6
7 CompilerCase #PB_OS_MacOS
8     ; Code for Mac systems.
9
10 CompilerCase #PB_OS_Windows
11     ; Code for Windows systems.
12
13 CompilerEndSelect
```

The last compiler directive that we're going to discuss here is: `EnableExplicit` .

There is a good reason for using this directive. It requires that all variables must be defined explicitly before usage, in some way, (using `Define` , `Dim` , `Global` , `Protected` , `Static` etc.) Doing so eliminates the possibility of logic errors due to mistyped variable names being defined "on-the-fly". This type of subtle error will not affect a program's compilation but could well present as an inconvenient bug at run-time. Using this directive eliminates this kind of problem as a compiler error would occur.

For example:

```
1 EnableExplicit
2
3 Define.l Field, FieldMax
4
5 ; ...
6
7 If Field < FieldMax
8     ; If EnableExplicit is omitted this section of code may not execute
9     when intended because FieldMax will be zero.
10 EndIf
```

UserGuide Navigation

< Previous: Structuring code in Procedures || Overview || Next: Reading and writing files >

Chapter 77

UserGuide - Reading and writing files

This example will write 100 random records each containing a byte , a floating-point number , a long integer and a string . It then reads all the records back and displays them in the debug window . It demonstrates the `GetTemporaryDirectory()` , `CreateFile()` , `OpenFile()` , `Eof()` and a number of Read and Write data instructions too.

It works fine as far as it goes, but has a drawback. As the string value has a variable length - you can't randomly access the records because you can't predict where each new record will start in the file. They must be all be read back in the same sequence as they were written. This isn't a problem with the small number of records created here but this could be an inconvenience with a larger file. PureBasic offers a way to handle this situation too - but an example would be too complex for this topic. See the Database sections of the help-file or reference manual to see how it could be done.

```
1  EnableExplicit
2  ; Define the constants:
3  #MAXBYTE = 255
4  #MAXLONG = 2147483647
5
6  ; Define some variables.
7  Define.f Float
8  Define.i Count, File
9  Define.s Folder, FileName, String
10
11 ; Create a temporary file name.
12 Folder = GetTemporaryDirectory()
13 FileName = Folder + "test.data"
14
15 ; Create the temporary file.
16 ; If #PB_Any is used, CreateFile returns the file's number.
17 ; Useful if you may have more than one file open simultaneously.
18 File = CreateFile(#PB_Any, FileName)
19
20 If File
21     ; If this was successful - write 100 random records.
22     For Count = 1 To 100
23
24         ; Write a random byte (0 - 255).
25         WriteByte(File, Random(#MAXBYTE))
26
27         ; Create and write a random float.
28         ; This calculation is there to make the number have a
        floating-point component (probably).
29         Float = Random(#MAXLONG) / ((Random(7) + 2) * 5000)
30         WriteFloat(File, Float)
```

```

31
32     ; Write a random long.
33     WriteLong(File, Random(#MAXLONG))
34
35     ; Create and write a random string in Unicode format.
36     ; Note the use of WriteStringN to delimit the string with an end
of line marker.
37     String = "String " + StrU(Random(#MAXLONG))
38     WriteStringN(File, String, #PB_Unicode)
39
40     Next Count
41
42     ; Close the file.
43     CloseFile(File)
44
45 Else
46     ; If this was unsuccessful.
47     Debug "Could not create the file: " + FileName
48
49 EndIf
50
51 ; Open the file for reading this time.
52 File = ReadFile(#PB_Any, FileName)
53
54 If File
55     ; If this was successful - read and display records from the file.
56
57     ; Reset the counter.
58     Count = 1
59
60     ; Loop until the 'end of file' is reached.
61     ; This will read all of the records regardless of how many there
are.
62     While Eof(File) = 0
63
64         ; Print a header line.
65         Debug
"-----
66
67         Debug "[" + StrU(Count) + "]"
68         Count + 1
69         ; Read a byte value and print it.
70         Debug StrU(ReadByte(File), #PB_Byte)
71
72         ; Read a float value.
73         Debug StrF(ReadFloat(File))
74
75         ; Read a long value.
76         Debug StrU(ReadLong(File), #PB_Long)
77
78         ; Read a string value.
79         Debug ReadString(File, #PB_Unicode)
80
81     Wend
82
83     ; Print the trailing line.
84     Debug
"-----
85

```

```
86         ; Close the file.
87         CloseFile(File)
88
89         ; Tidy up.
90         DeleteFile(FileName)
91
92     Else
93         ; If this was unsuccessful.
94         Debug "Could not open the file: " + FileName
95
96     EndIf
```

UserGuide Navigation

< Previous: Compiler directives || Overview || Next: Memory access >

Chapter 78

UserGuide - First steps with the Debug output (variables & operators)

Normally we would present here the typical "Hello World". You want to see it? Ok here we go with two examples:

Debug output

"Hello World" in the Debug output window:

```
1  Debug "Hello World!"
```

MessageRequester

"Hello World" in a MessageRequester() :

```
1  MessageRequester("", "Hello World!")
```

Advanced Debug example

We now continue with a short example using the available variable types, arithmetic operators and displaying the result:

```
1  a = 5
2  b = 7
3  c = 3
4
5  d = a + b ; we use the values stored in variables 'a' and 'b' and
   save the sum of them in 'd'
6  d / c ; we directly use the value of 'd' (= 12) divided by the
   value of 'c' (= 3) and save the result in 'd'
7
8  Debug d ; will output 4
```

This way you have used variables "on the fly". To force the compiler always declaring variables before their first use, just use the keyword EnableExplicit .

UserGuide Navigation

Overview || Next: Variables >

Chapter 79

UserGuide - Displaying graphics output & simple drawing

This example show how to create a simple drawing. It uses the 2D drawing commands to draw two sine waves at different frequencies and shows the harmonic produced by combining the two waves. It uses procedures , which we will discuss in more detail later on, to break the drawing tasks into three self-contained tasks:

Drawing the axes - demonstrates the Line() command.

Drawing the legend - demonstrates the Box() and DrawText() commands.

Drawing the wave forms - demonstrates the LineXY() command and shows how to use color.

```
1  ; Window
2  Enumeration
3      #WinHarmonic
4  EndEnumeration
5
6  ; Gadgets
7  Enumeration
8      #txtPlot1
9      #cboPlot1
10     #txtPlot2
11     #cboPlot2
12     #imgPlot
13 EndEnumeration
14
15 ; Image
16 Enumeration
17     #drgPlot
18 EndEnumeration
19
20 ; Image dimensions are used in several places so define constants.
21 #imgPlotX = 8
22 #imgPlotY = 40
23 #imgPlotW = 745
24 #imgPlotH = 645
25
26 ; Event variables
27 Define.l Event, EventWindow, EventGadget, EventType, EventMenu
28
29 ; Implementation
30 Procedure CreateWindow()
31     ; Creates the window and gadgets.
32
```



```

33     If OpenWindow(#WinHarmonic, 30, 30, #imgPlotW + 20, #imgPlotH + 55,
    "Harmonics", #PB_Window_SystemMenu | #PB_Window_MinimizeGadget |
    #PB_Window_TitleBar)
34
35         ; This is a non-visual gadget used to draw the image, later its
    contents will be displayed in #imgPlot.
36         CreateImage(#drgPlot, #imgPlotW - 5, #imgPlotH - 5, 24)
37
38         ; Label for the Plot 1 combo.
39         TextGadget(#txtPlot1, 2, 5, 50, 25, "Plot 1:")
40
41         ; The Plot 1 combo.
42         ComboBoxGadget(#cboPlot1, 55, 5, 150, 25)
43         AddGadgetItem(#cboPlot1, 0, "Sin(X)")
44         AddGadgetItem(#cboPlot1, 1, "Sin(X * 2)")
45         AddGadgetItem(#cboPlot1, 2, "Sin(X * 3)")
46         AddGadgetItem(#cboPlot1, 3, "Sin(X * 4)")
47         AddGadgetItem(#cboPlot1, 4, "Sin(X * 5)")
48         AddGadgetItem(#cboPlot1, 5, "Sin(X * 6)")
49
50         ; Select Sin(X)
51         SetGadgetState(#cboPlot1, 0)
52
53         ; Label for the Plot 2 combo.
54         TextGadget(#txtPlot2, 230, 5, 50, 25, "Plot 2:")
55
56         ; The Plot 2 combo.
57         ComboBoxGadget(#cboPlot2, 280, 5, 150, 25)
58         AddGadgetItem(#cboPlot2, 0, "Sin(X)")
59         AddGadgetItem(#cboPlot2, 1, "Sin(X * 2)")
60         AddGadgetItem(#cboPlot2, 2, "Sin(X * 3)")
61         AddGadgetItem(#cboPlot2, 3, "Sin(X * 4)")
62         AddGadgetItem(#cboPlot2, 4, "Sin(X * 5)")
63         AddGadgetItem(#cboPlot2, 5, "Sin(X * 6)")
64
65         ; Select Sin(X * 2), otherwise the initial display is a bit
    uninteresting.
66         SetGadgetState(#cboPlot2, 1)
67
68         ; The visual image gadget on the window.
69         ImageGadget(#imgPlot, #imgPlotX, #imgPlotY, #imgPlotW, #imgPlotH,
    0, #PB_Image_Border)
70
71     EndIf
72
73 EndProcedure
74
75 Procedure PlotAxes()
76     ; Draws the axes on the image #drgPlot.
77
78     ; Send drawing commands to #drgPlot.
79     StartDrawing(ImageOutput(#drgPlot))
80
81     ; Draw a white background.
82     Box(0, 0, ImageWidth(#drgPlot), ImageHeight(#drgPlot), RGB(255,
    255, 255))
83
84     ; Draw the axes in black.
85     Line(1, 1, 1, ImageHeight(#drgPlot) - 2, RGB(0, 0, 0))

```

```

86     Line(1, (ImageHeight(#drgPlot) - 2) / 2, ImageWidth(#drgPlot) - 2,
1, RGB(0, 0, 0))
87
88     ; Finished drawing.
89     StopDrawing()
90 EndProcedure
91
92 Procedure PlotLegend(alngPlot1, alngPlot2)
93     ; Draws the legend on the image #drgPlot.
94
95     Protected.s strFunc1, strFunc2, strLabel1, strLabel2, strLabel3
96
97     ; Set label text 1.
98     If alngPlot1 = 0
99         strFunc1 = "Sin(X)"
100     Else
101         strFunc1 = "Sin(X * " + StrU(alngPlot1 + 1) + ")"
102     EndIf
103
104     ; Set label text 2.
105     If alngPlot2 = 0
106         strFunc2 = "Sin(X)"
107     Else
108         strFunc2 = "Sin(X * " + StrU(alngPlot2 + 1) + ")"
109     EndIf
110
111     ; Set label text.
112     strLabel1 = "Y = " + strFunc1
113     strLabel2 = "Y = " + strFunc2
114     strLabel3 = "Y = " + strFunc1 + " + " + " + strFunc2
115
116     ; Draw legend.
117     StartDrawing(ImageOutput(#drgPlot))
118
119     ; Box.
120     DrawingMode(#PB_2DDrawing_Outlined)
121     Box(20, 10, TextWidth(strLabel3) + 85, 80, RGB(0, 0, 0))
122
123     ; Label 1.
124     Line(30, 30, 50, 1, RGB(0, 0, 255))
125     DrawText(95, 22, strLabel1, RGB(0, 0, 0), RGB(255, 255, 255))
126
127     ; Label 2.
128     Line(30, 50, 50, 1, RGB(0, 255, 200))
129     DrawText(95, 42, strLabel2, RGB(0, 0, 0), RGB(255, 255, 255))
130
131     ; Label 3.
132     Line(30, 70, 50, 1, RGB(255, 0, 0))
133     DrawText(95, 62, strLabel3, RGB(0, 0, 0), RGB(255, 255, 255))
134
135     StopDrawing()
136
137 EndProcedure
138
139 Procedure PlotFunction(alngPlot1, alngPlot2)
140     ; Draws the waveforms on the image #drgPlot.
141
142     Protected.l lngSX, lngEX
143     Protected.f fltRad1, fltRad2, fltSY1, fltEY1, fltSY2, fltEY2,

```

```

144 fltSY3, fltEY3
145 StartDrawing(ImageOutput(#drgPlot))
146
147 ; Set initial start points for each wave.
148 lngSX = 1
149 fltSY1 = ImageHeight(#drgPlot) / 2
150 fltSY2 = fltSY1
151 fltSY3 = fltSY1
152
153 ; Plot wave forms.
154 For lngEX = 1 To 720
155     ; Sine function works in radians, so convert from degrees and
    calculate sine.
156
157     ; Function 1
158     If alngPlot1 = 0
159         fltRad1 = Sin(Radian(lngEX))
160     Else
161         ; If the function should have a multiplier, account for this.
162         fltRad1 = Sin(Radian(lngEX) * (alngPlot1 + 1))
163     EndIf
164
165     ; Function 2
166     If alngPlot2 = 0
167         fltRad2 = Sin(Radian(lngEX))
168     Else
169         fltRad2 = Sin(Radian(lngEX) * (alngPlot2 + 1))
170     EndIf
171
172     ; Plot function 1 in blue.
173     ; Calculate end Y point.
174     fltEY1 = (ImageHeight(#drgPlot) / 2) + (fltRad1 * 100)
175     ; Draw a line from the start point to the end point.
176     LineXY(lngSX, fltSY1, lngEX, fltEY1, RGB(0, 0, 255))
177     ; Update the next start Y point to be the current end Y point.
178     fltSY1 = fltEY1
179
180     ; Plot function 2 in green.
181     fltEY2 = (ImageHeight(#drgPlot) / 2) + (fltRad2 * 100)
182     LineXY(lngSX, fltSY2, lngEX, fltEY2, RGB(0, 255, 200))
183     fltSY2 = fltEY2
184
185     ; Plot harmonic in red.
186     fltEY3 = (ImageHeight(#drgPlot) / 2) + ((fltRad1 + fltRad2) *
100)
187     LineXY(lngSX, fltSY3, lngEX, fltEY3, RGB(255, 0, 0))
188     fltSY3 = fltEY3
189
190     ; Update the start X point to be the current end X point.
191     lngSX = lngEX
192 Next lngEX
193
194 StopDrawing()
195
196 EndProcedure
197
198 ; - Main
199 CreateWindow()

```

```

200 PlotAxes()
201 PlotLegend(GetGadgetState(#cboPlot1), GetGadgetState(#cboPlot2))
202 PlotFunction(GetGadgetState(#cboPlot1), GetGadgetState(#cboPlot2))
203
204 ; Reload the image gadget now drawing is complete.
205 ImageGadget(#imgPlot, #imgPlotX, #imgPlotY, #imgPlotW, #imgPlotH,
    ImageID(#drgPlot), #PB_Image_Border)
206
207 ;- Event loop
208 Repeat
209     Event = WaitWindowEvent()
210     EventWindow = EventWindow()
211     EventGadget = EventGadget()
212     EventType = EventType()
213
214     Select Event
215     Case #PB_Event_Gadget
216         If EventGadget = #txtPlot1 Or EventGadget = #txtPlot2
217             ; Do nothing.
218         ElseIf EventGadget = #imgPlot
219             ; Do nothing.
220         ElseIf EventGadget = #cboPlot1 Or EventGadget = #cboPlot2
221             ; If one of the combo boxes changed, redraw the image.
222             PlotAxes()
223             PlotLegend(GetGadgetState(#cboPlot1),
                GetGadgetState(#cboPlot2))
224             PlotFunction(GetGadgetState(#cboPlot1),
                GetGadgetState(#cboPlot2))
225             ImageGadget(#imgPlot, #imgPlotX, #imgPlotY, #imgPlotW,
                #imgPlotH, ImageID(#drgPlot), #PB_Image_Border)
226         EndIf
227     Case #PB_Event_CloseWindow
228         If EventWindow = #WinHarmonic
229             CloseWindow(#WinHarmonic)
230             Break
231         EndIf
232     EndSelect
233 ForEver

```

UserGuide Navigation

< Previous: Building a graphical user interface (GUI) || Overview || Next: Structuring code in Procedures >

Chapter 80

UserGuide - Building a graphical user interface (GUI)

In addition to the console window , PureBasic supports the creation of graphical user interfaces (GUI) too. So let's revisit the file properties example from previous items again and turn it into a GUI application.

Note that PureBasic provides a far easier way of getting this particular job done already - the `ExplorerListGadget()` ; but, as the example is intended to introduce managing GUI elements, using that gadget would defeat this object a bit.

```
1 ; The structure for file information as before.
2 Structure FILEITEM
3     Name.s
4     Attributes.i
5     Size.q
6     DateCreated.i
7     DateAccessed.i
8     DateModified.i
9 EndStructure
10
11 ; This is a constant to identify the window.
12 Enumeration
13     #WindowFiles
14 EndEnumeration
15
16 ; This is an enumeration to identify controls which will appear on
    the window.
17 Enumeration
18     #Folder
19     #Files
20 EndEnumeration
21
22 ; Now we define a list of files using the structure previously
    specified.
23 NewList Files.FILEITEM()
24
25 ; And some working variables to make things happen.
26 Define.s Access, Attrib, Create, Folder, Modify, Msg, Num, Size
27 Define.l Result, Flags
28
29 ; These variables will receive details of GUI events as they occur in
    the program.
30 Define.l Event, EventWindow, EventGadget, EventType, EventMenu
```

```

31
32 ; This function gets the home directory for the logged on user.
33 Folder = GetHomeDirectory()
34
35 ; Open the directory to enumerate its contents.
36 Result = ExamineDirectory(0, Folder, " *.*")
37
38 ; If this is ok, begin enumeration of entries.
39 If Result
40     ; Loop through until NextDirectoryEntry(0) becomes zero -
    indicating that there are no more entries.
41     While NextDirectoryEntry(0)
42         ; If the directory entry is a file, not a folder.
43         If DirectoryEntryType(0) = #PB_DirectoryEntry_File
44
45             ; Add a new element to the list.
46             AddElement(Files())
47             ; And populate it with the properties of the file.
48             Files()\Name = DirectoryEntryName(0)
49             Files()\Size = DirectoryEntrySize(0)
50             Files()\Attributes = DirectoryEntryAttributes(0)
51             Files()\DateCreated = DirectoryEntryDate(0, #PB_Date_Created)
52             Files()\DateAccessed = DirectoryEntryDate(0, #PB_Date_Accessed)
53             Files()\DateModified = DirectoryEntryDate(0, #PB_Date_Modified)
54         EndIf
55     Wend
56     ; Close the directory.
57     FinishDirectory(0)
58 EndIf
59
60 ; Sort the list into ascending alphabetical order of file name.
61 SortStructuredList(Files(), #PB_Sort_Ascending,
    OffsetOf(FILEITEM\Name), #PB_String)
62
63 ; The interesting stuff starts to happen here...
64
65 ; This line defines a flag for the window attributes by OR-ing
    together the desired attribute constants.
66 Flags = #PB_Window_SystemMenu | #PB_Window_SizeGadget |
    #PB_Window_MinimizeGadget | #PB_Window_MaximizeGadget |
    #PB_Window_TitleBar
67
68 ; Open a GUI window.
69 OpenWindow(#WindowFiles, 50, 50, 450, 400, "File Properties", Flags)
70 ; A text gadget to show the name of the folder.
71 TextGadget(#Folder, 5, 40, 440, 25, Folder)
72 ; A list icon gadget to hold the file list and properties.
73 ListIconGadget(#Files, 5, 70, 440, 326, "#", 35)
74 ; Add columns to the ListIconGadget to hold each property.
75 AddGadgetColumn(#Files, 1, "Name", 200)
76 AddGadgetColumn(#Files, 2, "Created", 100)
77 AddGadgetColumn(#Files, 3, "Accessed", 100)
78 AddGadgetColumn(#Files, 4, "Modified", 100)
79 AddGadgetColumn(#Files, 5, "Attributes", 150)
80 AddGadgetColumn(#Files, 6, "Size", 100)
81
82 ; Load the files into the list view.
83 ForEach Files()
84     ; Display the item number and file name.

```

```

85     Num = StrU(ListIndex(Files()) + 1)
86
87     ; These lines convert the three date values to something more
familiar.
88     Create = FormatDate("%dd/%mm/%yyyy", Files()\DateCreated)
89     Access = FormatDate("%dd/%mm/%yyyy", Files()\DateAccessed)
90     Modify = FormatDate("%dd/%mm/%yyyy", Files()\DateModified)
91
92     ; Convert the file size to a padded string the same as with the
index value above,
93     ; but allow space for the maximum size of a quad.
94     Size = StrU(Files()\Size)
95
96     ; Convert the attributes to a string, for now.
97     Attrib = StrU(Files()\Attributes)
98
99     ; Build a row string.
100    ; The Line Feed character 'Chr(10)' tells the gadget to move to the
next column.
101    Msg = Num + Chr(10) + Files()\Name + Chr(10) + Create + Chr(10) +
Access + Chr(10) + Modify + Chr(10) + Attrib + Chr(10) + Size
102
103    ; Add the row to the list view gadget.
104    AddGadgetItem(#Files, -1, Msg)
105  Next Files()
106
107  ; This is the event loop for the window.
108  ; It will deal with all the user interaction events that we wish to
use.
109
110  Repeat
111    ; Wait until a new window or gadget event occurs.
112    Event = WaitWindowEvent()
113    ; In programs with more than one form, which window did the event
occur on.
114    EventWindow = EventWindow()
115    ; Which gadget did the event occur on.
116    EventGadget = EventGadget()
117    ; What sort of event occurred.
118    EventType = EventType()
119
120    ; Take some action.
121    Select Event
122
123        Case #PB_Event_Gadget
124            ; A gadget event occurred.
125            If EventGadget = #Folder
126                ElseIf EventGadget = #Files
127            EndIf
128
129        Case #PB_Event_CloseWindow
130            ; The window was closed.
131            If EventWindow = #WindowFiles
132                CloseWindow(#WindowFiles)
133                Break
134            EndIf
135
136    EndSelect
137

```

```
138 |         ; Go round and do it again.  
139 |         ; In practice the loop isn't infinite because it can be stopped by  
      | clicking the window's Close button.  
140 | Forever
```

At this point the application already has some useful features. However, it has some problems too:

- 1) You can't choose a folder to show.
- 2) You can't update the list contents without closing and restarting the program.
- 3) If you resize the window, the gadgets don't resize with it.
- 4) The attributes column is still not very useful.

We will revisit this program again later on to fix all these issues.

UserGuide Navigation

< Previous: Displaying text output (Console) || Overview || Next: Displaying graphics output & simple drawing >

Chapter 81

UserGuide - Input & Output

Every PureBasic application can communicate and interact with the user on different ways.

Thereby we distinguish between

- a) the pure output of information
- b) the interaction of the PureBasic application with the user, when user-input will be taken and the results will be outputted again.

It's not possible anymore, to use a simple "PRINT" command to output some things on the screen, like it was possible on DOS operating systems (OS) without a graphical user interface (GUI) years ago.

Today such a GUI is always present, when you use an actual OS like Windows, Mac OSX or Linux.

For the output of information we have different possibilities:

- Debug window (only possible during programming with PureBasic)
- MessageRequester() (output of shorter text messages in a requester window)
- Files (for saving the results in a text-file, etc.)
- Console (for simple and almost non-graphic text output, most similar to earlier DOS times)
- Windows and gadgets (standard windows with GUI elements on the desktop of the OS, e.g. for applications)
- Screen (Output of text and graphics directly on the screen, e.g. for games)

To be able to **record and process input by the user** of the PureBasic application, the three last-mentioned output options have also the possibility to get user-input:

- in the console using Input()
- in the window using WaitWindowEvent() / WindowEvent() , which can get the events occurred in the window , like clicking on a button or the entered text in a StringGadget()
- in the graphics screen using the keyboard
- there is as well the possibility to get user-input using the InputRequester()

UserGuide Navigation

< Previous: Storing data in memory || Overview || Next: Displaying text output (Console) >

Chapter 82

UserGuide - Other Compiler keywords

<empty for now>

UserGuide Navigation

< Previous: Managing multiple windows || Overview || Next: Other Library functions >

Chapter 83

UserGuide - Other Library functions

<empty for now>

UserGuide Navigation

< Previous: Other Compiler keywords || Overview || Next: Advanced functions >

Chapter 84

UserGuide - Loops

Data, Events or many other things can also be processed using loops, which are always checked for a specific condition. Loops can be: Repeat : Until , Repeat : Forever , While : Wend , For : Next , ForEach : Next .

In this loop the counter A is increased by two each time, this loop will always perform the same number of iterations.

```
1 Define.i A
2 For A = 0 To 10 Step 2
3 ?? Debug A
4 Next A
```

This loop will increment the variable B by a random amount between 0 and 20 each time, until B exceeds 100. The number of iterations actually performed in the loop will vary depending on the random numbers. The check is performed at the start of the loop - so if the condition is already true, zero iterations may be performed. Take the ; away from the second line to see this happen.

```
1 Define.i B
2 ; B = 100
3 While B < 100
4 ?? B + Random(20)
5 ?? Debug B
6 Wend
```

This loop is very similar to the last except that the check is performed at the end of the loop. So one iteration, at least, will be performed. Again remove the ; from the second line to demonstrate.

```
1 Define.i C
2 ; C = 100
3 Repeat
4 ?? C + Random(20)
5 ?? Debug C
6 Until C > 99
```

This loop is infinite. It won't stop until you stop it (use the red X button on the IDE toolbar).

```
1 Define.i D
2 Repeat
3 ?? Debug D
4 ForEver
```

There is a special loop for working with lists and maps , it will iterate every member of the list (or map) in turn.

```
1  NewList Fruit.s()
2
3  AddElement(Fruit())
4  Fruit() = "Banana"
5
6  AddElement(Fruit())
7  Fruit() = "Apple"
8
9  AddElement(Fruit())
10 Fruit() = "Pear"
11
12 AddElement(Fruit())
13 Fruit() = "Orange"
14
15 ForEach Fruit()
16 ??   Debug Fruit()
17     Next Fruit()
```

UserGuide Navigation

< Previous: Decisions & Conditions || Overview || Next: String Manipulation >

Chapter 85

UserGuide - Memory access

Some PureBasic instructions, for example those from the Cipher library and many operating system API calls, require a pointer to a memory buffer as an argument rather than the data directly itself.

PureBasic provides a number of instructions to manipulate memory buffers to facilitate this.

This example uses a buffer to read a file from disk into memory. It then converts the buffer content into a hexadecimal and text display in a ListIconGadget() as a simple hex viewer application.

An ExplorerListGadget() is used to display the contents of the user's home directory, initially, and to allow selection of a file. Two buttons are provided, one to display a file and another to clear the display. The ListIcon Gadget is divided into nine columns, the first shows the base offset of each line in the list, the next show eight byte values offset from the base value and the ninth shows the string equivalent of these eight values.

Two pointers are used - the first (*Buffer) contains the memory address of the complete file. The second (*Byte), in the procedure "FileDisplay", demonstrates the use of pointer arithmetic and the Peek instruction to obtain individual values from within the buffer.

Finally, the "FileClose" procedure demonstrates the use of the FillMemory() instruction to overwrite the buffer's contents and FreeMemory() to de-allocate the memory buffer.

```
1  ; - Compiler Directives
2  EnableExplicit
3
4  ; - Constants
5  ; Window
6  Enumeration
7      #WindowHex
8  EndEnumeration
9
10 ; Gadgets
11 Enumeration
12     #GadgetFiles
13     #GadgetOpen
14     #GadgetClose
15     #GadgetHex
16 EndEnumeration
17
18 ; - Variables
19 Define.l Event, EventWindow, EventGadget, EventType, EventMenu
20 Define.l Length
21 Define.s File
22 Define *Buffer
23
24 ; - Declarations
25 Declare WindowCreate()
26 Declare WindowResize()
```

```

27 Declare FileClose()
28 Declare FileDisplay()
29 Declare FileRead()
30
31 ; - Implementation
32 Procedure WindowCreate()
33     ; Create the window.
34
35     Protected.l Col
36     Protected.s Label
37
38     If OpenWindow(#WindowHex, 50, 50, 500, 400, "Hex View",
39         #PB_Window_SystemMenu | #PB_Window_SizeGadget |
40         #PB_Window_MinimizeGadget | #PB_Window_TitleBar)
41
42         ; Set minimum window size.
43         WindowBounds(#WindowHex, 175, 175, #PB_Ignore, #PB_Ignore)
44
45         ; Create Explorer List and set to user's home directory.
46         ExplorerListGadget(#GadgetFiles, 5, 5, 490, 175,
47             GetHomeDirectory())
48
49         ; Buttons.
50         ButtonGadget(#GadgetOpen, 5, 185, 80, 25, "Open")
51         ButtonGadget(#GadgetClose, 100, 185, 80, 25, "Close")
52
53         ; List Icon Gadget.
54         ListIconGadget(#GadgetHex, 5, 215, 490, 180, "Offset", 80,
55             #PB_ListIcon_AlwaysShowSelection | #PB_ListIcon_GridLines |
56             #PB_ListIcon_FullRowSelect)
57
58         ; Column Headings.
59         For Col = 0 To 7
60             Label = RSet(Hex(Col, #PB_Byte), 2, "0")
61             AddGadgetColumn(#GadgetHex, Col + 1, Label, 38)
62         Next Col
63         AddGadgetColumn(#GadgetHex, 9, "Text", 80)
64
65     EndIf
66 EndProcedure
67
68 Procedure WindowResize()
69     ; Resize gadgets to new window size.
70
71     Protected.l X, Y, W, H
72
73     ; Explorer List
74     W = WindowWidth(#WindowHex) - 10
75     H = (WindowHeight(#WindowHex) - 35) / 2
76     ResizeGadget(#GadgetFiles, #PB_Ignore, #PB_Ignore, W, H)
77
78     ; Buttons
79     Y = GadgetHeight(#GadgetFiles) + 10
80     ResizeGadget(#GadgetOpen, #PB_Ignore, Y, #PB_Ignore, #PB_Ignore)
81     ResizeGadget(#GadgetClose, #PB_Ignore, Y, #PB_Ignore, #PB_Ignore)
82
83     ; List Icon View
84     Y = (WindowHeight(#WindowHex) / 2) + 23

```

```

81     W = WindowWidth(#WindowHex) - 10
82     H = WindowHeight(#WindowHex) - (Y + 5)
83     ResizeGadget(#GadgetHex, #PB_Ignore, Y, W, H)
84
85 EndProcedure
86
87 Procedure FileClose()
88     ; Clear the list view and release the memory buffer.
89
90     Shared Length, *Buffer
91
92     ClearGadgetItems(#GadgetHex)
93     FillMemory(*Buffer, Length)
94     FreeMemory(*Buffer)
95
96 EndProcedure
97
98 Procedure FileDisplay()
99     ; Display the file buffer in the list view.
100
101     Shared Length, *Buffer
102
103     Protected *Byte
104     Protected Peek
105     Protected.l Rows, Cols, Offset
106     Protected.s OffsetString, Row, String
107
108     ; Clear current contents.
109     ClearGadgetItems(#GadgetHex)
110
111     ; Loop through rows.
112     For Rows = 0 To Length - 1 Step 8
113
114         ; Clear the text value for each row.
115         String = ""
116
117         ; Convert the offset value to a fixed length string.
118         Row = RSet(Hex(Rows, #PB_Long), 6, "0") + Chr(10)
119
120         ; Loop through columns.
121         For Cols = 0 To 7
122
123             ; Calculate the offset for the current column.
124             Offset = Rows + Cols
125
126             ; Compare the offset with the file length.
127             If Offset < Length
128                 ; The offset is less than the length of the file.
129
130                 ; Obtain the byte from the buffer.
131                 *Byte = *Buffer + Offset
132                 Peek = PeekB(*Byte)
133
134                 ; Convert the byte to text.
135                 Row + RSet(Hex(Peek, #PB_Byte), 2, "0") + Chr(10)
136
137                 ; Add the character to the text version.
138                 Select Peek
139

```



```

140         Case 0 To 31, 127
141             ; Unprintable characters.
142             String + Chr(129)
143
144         Default
145             ; Printable characters.
146             String + Chr(Peek)
147
148     EndSelect
149
150 Else
151     ; The offset is greater than the length of the file.
152
153     ; Add an empty column.
154     Row + Chr(10)
155
156 EndIf
157
158 Next Cols
159
160 ; Add the text version at the end of the hex columns.
161 Row + String
162
163 ; Add the completed row to the list view.
164 AddGadgetItem(#GadgetHex, -1, Row)
165
166 Next Rows
167
168 EndProcedure
169
170 Procedure FileRead()
171     ; Read the file into the memory buffer.
172
173     Shared Length, File, *Buffer
174
175     Protected.b ReadByte
176     Protected.l FileNumber, ReadLong, Size
177
178     ; Stop if file is empty.
179     If File = ""
180         ProcedureReturn
181     EndIf
182
183     ; Stop if file size is invalid.
184     Size = FileSize(File)
185     If Size < 1
186         ProcedureReturn
187     EndIf
188
189     ; Open the file.
190     FileNumber = OpenFile(#PB_Any, File)
191     Length = Lof(FileNumber)
192
193     If File And Length
194
195         ; Allocate a memory buffer to hold the file.
196         *Buffer = AllocateMemory(Length)
197
198         ; Read the file into the buffer.

```

```

199         Length = ReadData(FileNumber, *Buffer, Length)
200
201     EndIf
202
203     ; Close the file.
204     CloseFile(FileNumber)
205
206 EndProcedure
207
208 ; - Main
209 WindowCreate()
210
211 ; - Event Loop
212 Repeat
213
214     ; Obtain event parameters.
215     Event = WaitWindowEvent()
216     EventGadget = EventGadget()
217     EventType = EventType()
218     EventWindow = EventWindow()
219
220     ; Handle events.
221     Select Event
222
223         Case #PB_Event_Gadget
224             If EventGadget = #GadgetFiles
225                 ; Do nothing.
226
227             ElseIf EventGadget = #GadgetOpen
228                 File = GetGadgetText(#GadgetFiles) +
GetGadgetItemText(#GadgetFiles, GetGadgetState(#GadgetFiles))
229                 If FileSize(File) > 0
230                     FileRead()
231                     FileDisplay()
232                 EndIf
233
234             ElseIf EventGadget = #GadgetClose
235                 FileClose()
236
237             ElseIf EventGadget = #GadgetHex
238                 ; Do nothing.
239
240             EndIf
241
242         Case #PB_Event_CloseWindow
243             If EventWindow = #WindowHex
244                 CloseWindow(#WindowHex)
245                 Break
246             EndIf
247
248         Case #PB_Event_SizeWindow
249             WindowResize()
250
251     EndSelect
252
253 ForEver

```

UserGuide Navigation

< Previous: Reading and writing files || Overview || Next: Dynamic numbering using #PB_Any >

Chapter 86

UserGuide - Overview

We start programming... this part of the PureBasic manual should guide you through some basic stuff, which you should learn while starting to program with PureBasic.

The following topics in this chapter should give you some ideas, where to start with PureBasic. It shouldn't replace any larger tutorial or the massive information, which can be found on the popular PureBasic forums. So the following information texts are short, but they include some "keywords" and links to further information, which are included in this reference manual. This chapter covers only a small part of the 1500+ commands available in PureBasic!

Topics in this chapter:

- First steps
- Variables and Processing of variables
- Constants
- Decisions & Conditions
- Loops
- String Manipulation
- Storing data in memory
- Input & Output
- Displaying text output (Console)
- Building a graphical user interface (GUI)
- Displaying graphics output & simple drawing
- Structuring code in Procedures
- Compiler directives (for different behavior on different OS)
- Reading and writing files
- Memory access
- Dynamic numbering using #PB_Any
- Managing multiple windows with different content
- Other Compiler keywords
- Other Library functions
- Advanced functions
- Some Tips & Tricks

UserGuide Navigation

Reference manual || Next: First steps >

Chapter 87

UserGuide - Dynamic numbering of windows and gadgets using #PB_Any

If you've looked at the help articles for the `OpenWindow` command or for any of the gadget creation commands (for example `ButtonGadget()`) or if you have experimented with the Form Designer tool, you may have noticed references to a special constant called `#PB_Any`. In this article we're going to look into this a little further to find out why it's so important.

So far all of our examples have used a group of constants, an enumeration, to identify a single window and each gadget on that window. This is fine in the simple programs we've demonstrated so far but presents a problem in more complex programs - only one of each of these windows can exist at the same time.

So what happens if a program needs to provide more than one copy of a window? Maybe to have several files open at once or possibly to provide several different views of the same file.

This is where the special `#PB_Any` constant comes in. When this constant is used as the argument to the functions that support it, a unique reference number is automatically generated and returned as a result of the function.

Providing we keep track of all these references we can use this to our advantage. Organising this is a little more complex than the examples we've seen so far but the increased flexibility that it can provide makes this well worth the effort.

This example program provides a window upon which a regular polygon is drawn in blue on a circumscribing grey circle. A combo box is provided to allow a selection of polygons to be drawn. A menu is provided to allow the creation of new windows or the closing of the current window.

There are several things to notice about this program:

In the Enumerations section, note that there are only enumerations for the menu items.

These will be shared on all the windows although each window will have its own menu bar.

In the Structures section, note that the `POLYGONWINDOW` structure contains four integer values and so provides a place to store references for a menu bar, a label, a combo box and an image plot.

In the Variables section note that in addition to the usual variables to receive event details, a map called `ActiveWindows` is created using the `POLYGONWINDOW` structure previously defined.

Look at the `CreatePolygonWindow` procedure.

When we create the window we capture the result of the `OpenWindow()` function in a variable called `ThisWindow`. We then convert this to a string value and use this as the key for a new map entry.

When we then create the menu, label, combo and image gadgets on the new window the references returned by these functions are stored in the map too.

Look at the `ResizePolygonWindow` procedure.

Notice how the value of `EventWindow` is passed in from the event loop into this procedure.

This value is then used to retrieve child control references from the map and these references are then used to resize the gadgets.

Look at the DestroyPolygonWindow procedure.

Here the child control references are removed from the map when a window is closed. If the map size reaches zero there are no more open windows and DestroyPolygonWindow sets an event flag to tell the program to end.

Start the program up.

- Use the New Window menu item to open up two or three new polygon windows.
- Use the combo box in each one to select a different shape notice that each window works independently of all the others.
- Resize some of the windows notice that they can all be resized independently of each other and that the polygon resizes with the window too.

Finally, note that a map isnt the only way to achieve this effect, a List or an Array could be used to do the job too, if you prefer, although the code to implement these alternatives would need to be slightly different to that presented here because of the differences in the way those collections work.

```
1 ; Compiler Directives
2 EnableExplicit
3
4 ; Constants
5 CompilerIf Defined(Blue, #PB_Constant) = #False
6     #Blue = 16711680
7     #Gray = 8421504
8     #White = 16777215
9 CompilerEndIf
10
11 ; - Enumerations
12 ; The menu commands will be the same on all the windows.
13 Enumeration
14     #MenuNew
15     #MenuClose
16 EndEnumeration
17
18 ; - Structures
19 ; This structure will hold references to the unique elements of a
    window.
20 Structure POLYGONWINDOW
21     Menu.i
22     LabelSides.i
23     ComboSides.i
24     ImagePlot.i
25 EndStructure
26
27 ; - Variables
28
29 ; This map uses the previously defined structure to hold references
    for all the open windows.
30 NewMap ActiveWindows.POLYGONWINDOW()
31
32 ; Event variables.
33 Define.i Event, EventWindow, EventGadget, EventType, EventMenu,
    EventQuit
34 Define.s EventWindowKey
35
36 ; Implementation.
37 Procedure.i CreatePolygonWindow()
38     ; Creates a new window and gadgets, adding it and its child gadgets
        to the tracking map.
39     Shared ActiveWindows()
40     Protected.i ThisWindow
41     Protected.s ThisKey
```

```

42
43   ThisWindow = OpenWindow(#PB_Any, 50, 50, 300, 300, "Polygon",
#PB_Window_SystemMenu | #PB_Window_SizeGadget |
#PB_Window_MinimizeGadget | #PB_Window_TitleBar)
44
45   WindowBounds(ThisWindow, 250, 250, #PB_Ignore, #PB_Ignore)
46
47   If ThisWindow
48       ; Maps take a string value as key so convert the integer
ThisWindow to a string.
49       ThisKey = StrU(ThisWindow)
50
51       ; Add a map element to hold the new gadget references.
52       AddMapElement(ActiveWindows(), ThisKey)
53
54       ; Create the menu bar.
55       ActiveWindows(ThisKey)\Menu = CreateMenu(#PB_Any,
WindowID(ThisWindow))
56       MenuTitle("Window")
57       MenuItem(#MenuNew, "New Window")
58       MenuItem(#MenuClose, "Close Window")
59
60       ; Create the child gadgets and store their references in the map.
61       With ActiveWindows()
62           ; A label for the combo.
63           \LabelSides = TextGadget(#PB_Any, 5, 5, 150, 20, "Number of
Sides:")
64
65           ; The Sides combo.
66           \ComboSides = ComboBoxGadget(#PB_Any, 160, 5, 100, 25)
67           AddGadgetItem(\ComboSides, 0, "Triangle")
68           AddGadgetItem(\ComboSides, 1, "Diamond")
69           AddGadgetItem(\ComboSides, 2, "Pentagon")
70           AddGadgetItem(\ComboSides, 3, "Hexagon")
71           AddGadgetItem(\ComboSides, 4, "Heptagon")
72           AddGadgetItem(\ComboSides, 5, "Octagon")
73           AddGadgetItem(\ComboSides, 6, "Nonagon")
74           AddGadgetItem(\ComboSides, 7, "Decagon")
75
76           ; Select Triangle.
77           SetGadgetState(\ComboSides, 0)
78
79           ; The visual image gadget on the window.
80           \ImagePlot = ImageGadget(#PB_Any, 5, 35, 290, 240, 0,
#PB_Image_Border)
81           EndWith
82       EndIf
83
84       ; Return the reference to the new window.
85       ProcedureReturn ThisWindow
86   EndProcedure
87
88   Procedure DestroyPolygonWindow(Window.i)
89       ; Remove Window from the ActiveWindows map, close the window and
set the quit flag, if appropriate.
90       Shared EventQuit, ActiveWindows()
91       Protected.s ThisKey
92
93       ; Convert the integer Window to a string.

```

```

94     ThisKey = StrU(Window)
95
96     ; Delete the map entry.
97     DeleteMapElement(ActiveWindows(), ThisKey)
98
99     ; Close the window.
100    CloseWindow(Window)
101
102    ; Check if there are still open windows.
103    If MapSize(ActiveWindows()) = 0
104        EventQuit = #True
105    EndIf
106 EndProcedure
107
108 Procedure.i ResizePolygonWindow(Window.i)
109     ; Resize the child gadgets on Window.
110     ; In practice only the ImageGadget needs to be resized in this
111     example.
112     Shared ActiveWindows()
113     Protected.i ThisImage
114     Protected.i X, Y, W, H
115     Protected.s ThisKey
116
117     ; Obtain references to the affected gadgets from the map.
118     ThisKey = StrU(Window)
119     ThisImage = ActiveWindows(ThisKey)\ImagePlot
120
121     ; Resize gadgets.
122     W = WindowWidth(Window) - 15
123     H = WindowHeight(Window) - 70
124     ResizeGadget(ThisImage, #PB_Ignore, #PB_Ignore, W, H)
125 EndProcedure
126
127 Procedure PlotPolygon(Window.i)
128     ; Draw the polygon image and transfer it to the image gadget.
129     Shared ActiveWindows()
130     Protected.f Radius, OriginX, OriginY, StartX, StartY, EndX, EndY
131     Protected.i Sides, Vertex, Width, Height, ThisCombo, ThisImage,
132     ThisPlot
133     Protected.s ThisKey
134
135     ; Check if the event is for the right window.
136     If Not IsWindow(Window) : ProcedureReturn : EndIf
137
138     ; Obtain references to the affected gadgets from the map.
139     ThisKey = StrU(Window)
140     ThisCombo = ActiveWindows(ThisKey)\ComboSides
141     ThisImage = ActiveWindows(ThisKey)\ImagePlot
142
143     ; Calculate dimensions and origin.
144     Sides = GetGadgetState(ThisCombo) + 3
145     Width = GadgetWidth(ThisImage) - 4
146     Height = GadgetHeight(ThisImage) - 4
147     OriginX = Width/2
148     OriginY = Height/2
149     If Width < Height
150         Radius = OriginX - 50
151     Else
152         Radius = OriginY - 50

```



```

151     EndIf
152
153     ; Create a new image.
154     ThisPlot = CreateImage(#PB_Any, Width, Height)
155     StartDrawing(ImageOutput(ThisPlot))
156
157     ; Draw a white background.
158     Box(0, 0, Width, Height, #White)
159
160     ; Draw a gray circumscribing circle.
161     Circle(OriginX, OriginY, Radius, #Gray)
162
163     ; Draw the polygon.
164     For Vertex = 0 To Sides
165
166         ; Calculate side start point.
167         StartX = OriginX + (Radius * Cos(2 * #PI * Vertex/Sides))
168         StartY = OriginY + (Radius * Sin(2 * #PI * Vertex/Sides))
169
170         ; and end point.
171         EndX = OriginX + (Radius * Cos(2 * #PI * (Vertex + 1)/Sides))
172         EndY = OriginY + (Radius * Sin(2 * #PI * (Vertex + 1)/Sides))
173
174         ; Draw the side in blue.
175         LineXY(StartX, StartY, EndX, EndY, #Blue)
176
177     Next Vertex
178
179     ; Fill the polygon in blue
180     FillArea(OriginX, OriginY, #Blue, #Blue)
181
182     StopDrawing()
183
184     ; Transfer the image contents to the visible gadget.
185     SetGadgetState(ThisImage, ImageID(ThisPlot))
186
187     ; Destroy the temporary image.
188     FreeImage(ThisPlot)
189 EndProcedure
190
191 ; - Main
192
193 ; Create the first window.
194 EventWindow = CreatePolygonWindow()
195 ResizePolygonWindow(EventWindow)
196 PlotPolygon(EventWindow)
197
198 ; - Event loop
199 Repeat
200     Event = WaitWindowEvent()
201     EventWindow = EventWindow()
202     EventWindowKey = StrU(EventWindow)
203     EventGadget = EventGadget()
204     EventType = EventType()
205     EventMenu = EventMenu()
206
207     Select Event
208     Case #PB_Event_Gadget
209         ; A gadget event has occurred.

```

```

210     If EventGadget = ActiveWindows(EventWindowKey)\LabelSides
211         ; Do nothing.
212
213     ElseIf EventGadget = ActiveWindows(EventWindowKey)\ComboSides
214         ; If the combo box changes, redraw the image.
215         PlotPolygon(EventWindow)
216
217         ; Update the windows title to reflect the new shape.
218         SetWindowTitle(EventWindow,
GetGadgetText(ActiveWindows(EventWindowKey)\ComboSides))
219
220     ElseIf EventGadget = ActiveWindows(EventWindowKey)\ImagePlot
221         ; Do nothing.
222
223     EndIf
224
225     Case #PB_Event_Menu
226         ; A menu event has occurred.
227         If EventMenu = #MenuNew
228             EventWindow = CreatePolygonWindow()
229             ResizePolygonWindow(EventWindow)
230             PlotPolygon(EventWindow)
231
232             ElseIf EventMenu = #MenuClose
233                 DestroyPolygonWindow(EventWindow)
234
235             EndIf
236
237         Case #PB_Event_Repaint
238             ; A window's content has been invalidated.
239             PlotPolygon(EventWindow)
240
241         Case #PB_Event_SizeWindow
242             ; A window has been resized.
243             ResizePolygonWindow(EventWindow)
244             PlotPolygon(EventWindow)
245
246         Case #PB_Event_CloseWindow
247             ; A window has been closed.
248             DestroyPolygonWindow(EventWindow)
249
250     EndSelect
251
252     Until EventQuit = #True

```

UserGuide Navigation

< Previous: Memory access || Overview || Next: Managing multiple windows >

Chapter 88

UserGuide - Managing multiple windows with different content

In the previous article we examined one way in which a program can support multiple instances of a single type of window . In this one we are going to extend this concept further developing a program that can support multiple instances of several different types of window, in this case three:

- The Button window contains a list view and two buttons labelled 'Add' and 'Remove'. When the 'Add' button is clicked a random integer is added to the list view, when the 'Remove' button is clicked the currently highlighted entry in the list view is removed.
- The Date window contains a list view and two buttons in the same way as the Button window but also contains a calendar gadget too, the window layout is altered to accommodate this additional control. When the 'Add' button is clicked, it is the currently selected date that is added to the list view.
- The Track window contains two track bars , with a value between 0 and 100, and a string gadget . When the track bars are moved the string gadget is updated with the value of the second track bar subtracted from the first.

Each window contains a menu bar with items to create a new instance of any of the three supported window types or to close the current window.

Things to notice about this program are:

In the Structures section 4 structures are defined. The first, BASEWINDOW, defines a WindowClass value and a Menu value these values are common to each window type.

The remaining structures extend the BASEWINDOW structure with values for each of the unique controls that they require and which are not provided for by the BASEWINDOW structure.

In the Variables section note that again a map called ActiveWindows is created, however this time it is of integer type, it doesnt use any of the defined structures. There is a good reason for this, we need to store three different structure types to make this program work and we cant do this in a single map.

Also notice that *EventGadgets is defined using the BASEWINDOW structure.

Now look at the CreateButtonWindow procedure . As before we use #PB_Any to create the window and all the gadgets .

However this time the results are not stored directly in the ActiveWindows map. Instead we use the AllocateMemory function to allocate memory for a BUTTONWINDOW structure, we then store a pointer to this memory allocation in the ActiveWindows map. This is how we get around the problem of not being able to store all three of the different structures in the same map.

We also set the WindowClass value in the structure to #WindowClassButton to indicate which type of window, and therefore which type of structure, has been created we will need to know this later on.

There are two more CreateWindow procedures this time one for each class of the other window types.

They work in a similar way to that described, differing only where the windows gadgets are different and setting a different value in WindowClass.

Similarly we provide DestroyWindow and ResizeWindow procedures to take care of these functions.

We also provide a new procedure EventsButtonWindow. This procedure knows what to do when any of the gadgets on the window are activated by the user. Similar procedures are provided for the other

window types too.

In all these procedures we use the ActiveWindows map to retrieve the pointer to the memory allocation. We can then use this pointer to retrieve the references to the actual controls that we need to work with in each of these procedures:

```
1  *ThisData = ActiveWindows(ThisKey)
2  *ThisData\ListView ...
```

Each procedure only knows how to handle one type of window so before we start work we check the WindowClass value to make sure that a window of the correct type has been supplied as the argument something like this:

```
1  If *ThisData\WindowClass <> #WindowClassButton
```

The event loop is a bit different too. For each event type there is a determination like this:

```
1  ; Use *EventGadgets\WindowClass to call the correct resize window
    procedure.
2  Select *EventGadgets\WindowClass ...
```

Although the memory allocations actually made by the CreateWindow procedures will be of the BUTTONWINDOW, DATEWINDOW or TRACKWINDOW type we can use *EventGadgets this way because it is defined as the BASEWINDOW type and BASEWINDOW is the ancestor structure to the other structures.

Providing we dont attempt to change any of the stored values using *EventGadgets and weve no reason to do so all should be well.

Finally, we dispatch the recorded event values in EventWindow, EventGadget, EventType straight through to the event procedures and let them worry about getting the jobs done.

```
1  ; - Constants
2  #DateFormat = "%dd/%mm/%yyyy"
3
4  ; - Enumerations
5  Enumeration
6      #WindowClassButton = 1
7      #WindowClassDate
8      #WindowClassTrack
9  EndEnumeration
10
11 ; The menu commands will be the same on all the windows.
12 Enumeration
13     #MenuNewButton
14     #MenuNewDate
15     #MenuNewTrack
16     #MenuClose
17 EndEnumeration
18
19 ; - Structures
20 Structure BASEWINDOW
21     WindowClass.i
22     Menu.i
23 EndStructure
24
25 Structure BUTTONWINDOW Extends BASEWINDOW
26     ListView.i
27     AddButton.i
28     RemoveButton.i
29 EndStructure
30
```

```

31 Structure DATEWINDOW Extends BASEWINDOW
32     Calendar.i
33     AddButton.i
34     RemoveButton.i
35     ListView.i
36 EndStructure
37
38 Structure TRACKWINDOW Extends BASEWINDOW
39     TrackBar1.i
40     TrackBar2.i
41     Label.i
42     Difference.i
43 EndStructure
44
45 ; - Variables
46 ; This map will track all the active windows as before, however as
47 ; the structure for each window class
48 ; differs we will be storing pointers to structures in the map not
49 ; the gadget references directly.
50 NewMap ActiveWindows.i()
51
52 ; These values will be used to give new windows a unique label.
53 Define.i Buttons, Dates, Tracks
54
55 ; Event variables.
56 ; Notice the type of *EventGadgets.
57 Define.i Event, EventWindow, EventGadget, EventType, EventMenu,
58     EventQuit
59 Define.s EventWindowKey
60 Define *EventGadgets.BASEWINDOW
61
62 ; - Button Window
63 Procedure.i CreateButtonWindow()
64     ; Creates a new Button window and adds it to the tracking map,
65     ; allocates memory for gadget references, creates the gadgets
66     ; and stores these references in the memory structure.
67     Shared Buttons, ActiveWindows()
68     Protected *ThisData.BUTTONWINDOW
69     Protected.i ThisWindow
70     Protected.s ThisKey, ThisTitle
71
72     ; Set the window caption.
73     Buttons + 1
74     ThisTitle = "Button Window " + StrU(Buttons)
75
76     ; Open the window.
77     ThisWindow = OpenWindow(#PB_Any, 30, 30, 225, 300, ThisTitle,
78     #PB_Window_SystemMenu | #PB_Window_SizeGadget |
79     #PB_Window_MinimizeGadget | #PB_Window_TitleBar)
80
81     ; Check that the OpenWindow command worked.
82     If ThisWindow
83         ; Set minimum window dimensions.
84         WindowBounds(ThisWindow, 220, 100, #PB_Ignore, #PB_Ignore)
85
86         ; Convert the window reference to a string to use as the map key
87         ; value.
88         ThisKey = StrU(ThisWindow)
89

```

```

84         ; Allocate memory to store the gadget references in.
85         *ThisData = AllocateMemory(SizeOf(BUTTONWINDOW))
86     EndIf
87
88     ; Check that the memory allocation worked.
89     If *ThisData
90         ; Store the window reference and memory pointer values in the map.
91         ActiveWindows(ThisKey) = *ThisData
92
93         ; Set the window class.
94         *ThisData\WindowClass = #WindowClassButton
95
96         ; Create the menu bar.
97         *ThisData\Menu = CreateMenu(#PB_Any, WindowID(ThisWindow))
98
99         ; If the menu creation worked, create the menu items.
100        If *ThisData\Menu
101            MenuItem("Window")
102            MenuItem(#MenuNewButton, "New Button Window")
103            MenuItem(#MenuNewDate, "New Date Window")
104            MenuItem(#MenuNewTrack, "New Track Window")
105            MenuItem(#MenuClose, "Close Window")
106        EndIf
107
108        ; Create the window gadgets.
109        *ThisData\ListView = ListViewGadget(#PB_Any, 10, 10, 200, 225)
110        *ThisData\AddButton = ButtonGadget(#PB_Any, 15, 245, 90, 30,
111        "Add")
112        *ThisData\RemoveButton = ButtonGadget(#PB_Any, 115, 245, 90, 30,
113        "Remove")
114    Else
115        ; Memory allocation failed.
116        CloseWindow(ThisWindow)
117    EndIf
118
119    ; Set the return value.
120    If ThisWindow > 0 And *ThisData > 0
121        ; Return the reference to the new window.
122        ProcedureReturn ThisWindow
123    Else
124        ; Return 0
125        ProcedureReturn 0
126    EndIf
127 EndProcedure
128
129 Procedure.i DestroyButtonWindow(Window.i)
130     ; Remove Window from the ActiveWindows map, release the allocated
131     memory,
132     ; close the window and set the quit flag, if appropriate.
133     Shared EventQuit, ActiveWindows()
134     Protected *ThisData.BUTTONWINDOW
135     Protected.s ThisKey
136
137     ; Convert the integer Window to a string.
138     ThisKey = StrU(Window)
139
140     ; Obtain the reference structure pointer.
141     *ThisData = ActiveWindows(ThisKey)

```

```

140 ; Check that a valid pointer was obtained, if not stop.
141 If *ThisData = 0
142     ProcedureReturn #False
143 EndIf
144
145 ; Check that it is the correct window type, if not stop.
146 If *ThisData\WindowClass <> #WindowClassButton
147     ProcedureReturn #False
148 EndIf
149
150 ; Release the memory allocation.
151 FreeMemory(*ThisData)
152
153 ; Delete the map entry.
154 DeleteMapElement(ActiveWindows(), ThisKey)
155
156 ; Close the window.
157 CloseWindow(Window)
158
159 ; Check if there are still open windows.
160 If MapSize(ActiveWindows()) = 0
161     EventQuit = #True
162 EndIf
163
164 ; Set the successful return value.
165 ProcedureReturn #True
166 EndProcedure
167
168 Procedure.i ResizeButtonWindow(Window.i)
169 ; Resize the child gadgets on Window.
170 Shared ActiveWindows()
171 Protected *ThisData.BUTTONWINDOW
172 Protected.i X, Y, W, H
173 Protected.s ThisKey
174
175 ; Obtain the reference structure pointer.
176 ThisKey = StrU(Window)
177 *ThisData = ActiveWindows(ThisKey)
178
179 ; Check that a valid pointer was obtained, if not stop.
180 If *ThisData = 0
181     ProcedureReturn #False
182 EndIf
183
184 ; Check that it is the correct window type, if not stop.
185 If *ThisData\WindowClass <> #WindowClassButton
186     ProcedureReturn #False
187 EndIf
188
189 ; Resize list view.
190 W = WindowWidth(Window) - 25
191 H = WindowHeight(Window) - 85
192 ResizeGadget(*ThisData\ListView, #PB_Ignore, #PB_Ignore, W, H)
193
194 ; Recenter buttons.
195 X = WindowWidth(Window)/2 - 95
196 Y = WindowHeight(Window) - 65
197 ResizeGadget(*ThisData\AddButton, X, Y, #PB_Ignore, #PB_Ignore)
198

```

```

199     X = WindowWidth(Window)/2 + 5
200     ResizeGadget(*ThisData\RemoveButton, X, Y, #PB_Ignore, #PB_Ignore)
201
202     ProcedureReturn #True
203 EndProcedure
204
205 Procedure.i EventsButtonWindow(Window, Gadget, Type)
206     ; Handle events for a button window.
207     Shared Buttons, ActiveWindows()
208     Protected *ThisData.BUTTONWINDOW
209     Protected.i NewValue, Index
210     Protected.s ThisKey
211
212     ; Convert the integer Window to a string.
213     ThisKey = StrU(Window)
214
215     ; Obtain the reference structure pointer.
216     *ThisData = ActiveWindows(ThisKey)
217
218     ; Check that a valid pointer was obtained, if not stop.
219     If *ThisData = 0
220         ProcedureReturn #False
221     EndIf
222
223     ; Check that it is the correct window type, if not stop.
224     If *ThisData\WindowClass <> #WindowClassButton
225         ProcedureReturn #False
226     EndIf
227
228     Select Gadget
229     Case *ThisData\AddButton
230         NewValue = Random(2147483647)
231         AddGadgetItem(*ThisData\ListView, -1, StrU(NewValue))
232
233     Case *ThisData\RemoveButton
234         Index = GetGadgetState(*ThisData\ListView)
235         If Index >= 0 And Index <= CountGadgetItems(*ThisData\ListView)
236             RemoveGadgetItem(*ThisData\ListView, Index)
237         EndIf
238
239     Case *ThisData\ListView
240         ; Do nothing.
241     EndSelect
242 EndProcedure
243
244 ; - Date Window
245 Procedure.i CreateDateWindow()
246     ; Creates a new Date window and adds it to the tracking map,
247     ; allocates memory for gadget references, creates the gadgets
248     ; and stores these references in the memory Structure.
249     Shared Dates, ActiveWindows()
250     Protected *ThisData.DATEWINDOW
251     Protected.i ThisWindow
252     Protected.s ThisKey, ThisTitle
253
254     Dates + 1
255     ThisTitle = "Date Window " + StrU(Dates)
256     ThisWindow = OpenWindow(#PB_Any, 30, 30, 310, 420, ThisTitle ,
        #PB_Window_SystemMenu | #PB_Window_SizeGadget |

```



```

#PB_Window_MinimizeGadget | #PB_Window_TitleBar)
257
258 ; Check that the OpenWindow command worked.
259 If ThisWindow
260 ; Set minimum window dimensions.
261 WindowBounds(ThisWindow, 310, 245, #PB_Ignore, #PB_Ignore)
262
263 ; Convert the window reference to a string to use as the map key
value.
264 ThisKey = StrU(ThisWindow)
265
266 ; Allocate memory to store the gadget references in.
267 *ThisData = AllocateMemory(SizeOf(DATEWINDOW))
268 EndIf
269
270 ; Check that the memory allocation worked.
271 If *ThisData
272 ; Store the window reference and memory pointer values in the map.
273 ActiveWindows(ThisKey) = *ThisData
274
275 ; Set the window class.
276 *ThisData\WindowClass = #WindowClassDate
277
278 ; Create the menu bar.
279 *ThisData\Menu = CreateMenu(#PB_Any, WindowID(ThisWindow))
280
281 ; If the menu creation worked, create the menu items.
282 If *ThisData\Menu
283 MenuItem("Window")
284 MenuItem(#MenuNewButton, "New Button Window")
285 MenuItem(#MenuNewDate, "New Date Window")
286 MenuItem(#MenuNewTrack, "New Track Window")
287 MenuItem(#MenuClose, "Close Window")
288 EndIf
289
290 ; Create the window gadgets.
291 *ThisData\Calendar = CalendarGadget(#PB_Any, 10, 10, 182, 162)
292 *ThisData\AddButton = ButtonGadget(#PB_Any, 210, 10, 90, 30,
"Add")
293 *ThisData\RemoveButton = ButtonGadget(#PB_Any, 210, 45, 90, 30,
"Remove")
294 *ThisData\ListView = ListViewGadget(#PB_Any, 10, 187, 290, 200)
295 Else
296 ; Memory allocation failed.
297 CloseWindow(ThisWindow)
298 EndIf
299
300 ; Set the return value.
301 If ThisWindow > 0 And *ThisData > 0
302 ; Return the reference to the new window.
303 ProcedureReturn ThisWindow
304 Else
305 ; Return 0
306 ProcedureReturn 0
307 EndIf
308
309 EndProcedure
310
311 Procedure.i DestroyDateWindow(Window.i)

```

```

312 ; Remove Window from the ActiveWindows map, release the allocated
memory,
313 ; close the window and set the quit flag, if appropriate.
314 Shared EventQuit, ActiveWindows()
315 Protected *ThisData.DATEWINDOW
316 Protected.s ThisKey
317
318 ; Convert the integer Window to a string.
319 ThisKey = StrU(Window)
320
321 ; Obtain the reference structure pointer.
322 *ThisData = ActiveWindows(ThisKey)
323
324 ; Check that a valid pointer was obtained.
325 If *ThisData = 0
326     ProcedureReturn #False
327 EndIf
328
329 ; Check that it is the correct window type, if not stop as this
procedure can't destroy this window.
330 If *ThisData\WindowClass <> #WindowClassDate
331     ProcedureReturn #False
332 EndIf
333
334 ; Release the memory allocation.
335 FreeMemory(*ThisData)
336
337 ; Delete the map entry.
338 DeleteMapElement(ActiveWindows(), ThisKey)
339
340 ; Close the window.
341 CloseWindow(Window)
342
343 ; Check if there are still open windows.
344 If MapSize(ActiveWindows()) = 0
345     EventQuit = #True
346 EndIf
347
348 ; Set the successful return value.
349 ProcedureReturn #True
350 EndProcedure
351
352 Procedure.i ResizeDateWindow(Window.i)
353 ; Resize the child gadgets on Window.
354 Shared ActiveWindows()
355 Protected *ThisData.DATEWINDOW
356 Protected.i X, Y, W, H
357 Protected.s ThisKey
358
359 ; Obtain the reference structure pointer.
360 ThisKey = StrU(Window)
361 *ThisData = ActiveWindows(ThisKey)
362
363 ; Check that a valid pointer was obtained, if not stop.
364 If *ThisData = 0
365     ProcedureReturn #False
366 EndIf
367
368 ; Check that it is the correct window type, if not stop.

```

```

369     If *ThisData\WindowClass <> #WindowClassDate
370         ProcedureReturn #False
371     EndIf
372
373     ; Resize list view.
374     W = WindowWidth(Window) - 20
375     H = WindowHeight(Window) - 220
376     ResizeGadget(*ThisData\ListView, #PB_Ignore, #PB_Ignore, W, H)
377
378     ProcedureReturn #True
379 EndProcedure
380
381 Procedure.i EventsDateWindow(Window, Gadget, Type)
382     ; Handle events for a Date window.
383     Shared Buttons, ActiveWindows()
384     Protected *ThisData.DATEWINDOW
385     Protected.i NewValue, Index
386     Protected.s ThisKey
387
388     ; Convert the integer Window to a string.
389     ThisKey = StrU(Window)
390
391     ; Obtain the reference structure pointer.
392     *ThisData = ActiveWindows(ThisKey)
393
394     ; Check that a valid pointer was obtained, if not stop.
395     If *ThisData = 0
396         ProcedureReturn #False
397     EndIf
398
399     ; Check that it is the correct window type, if not stop.
400     If *ThisData\WindowClass <> #WindowClassDate
401         ProcedureReturn #False
402     EndIf
403
404     Select Gadget
405     Case *ThisData\AddButton
406         NewValue = GetGadgetState(*ThisData\Calendar)
407         AddGadgetItem(*ThisData\ListView, -1, FormatDate(#DateFormat,
408 NewValue))
409
410     Case *ThisData\RemoveButton
411         Index = GetGadgetState(*ThisData\ListView)
412         If Index >= 0 And Index <= CountGadgetItems(*ThisData\ListView)
413             RemoveGadgetItem(*ThisData\ListView, Index)
414         EndIf
415
416     Case *ThisData\Calendar, *ThisData\ListView
417         ; Do nothing.
418     EndSelect
419 EndProcedure
420
421 ; - Track Window
422 Procedure.i CreateTrackWindow()
423     ; Creates a new Track window and adds it to the tracking map,
424     ; allocates memory for gadget references, creates the gadgets
425     ; and stores these references in the memory Structure.
426     Shared Tracks, ActiveWindows()
427     Protected *ThisData.TRACKWINDOW

```

```

427 Protected.i ThisWindow, ThisSum
428 Protected.s ThisKey, ThisTitle
429
430 Tracks + 1
431 ThisTitle = "Track Bar Window " + StrU(Tracks)
432 ThisWindow = OpenWindow(#PB_Any, 30, 30, 398, 130, ThisTitle,
#PB_Window_SystemMenu | #PB_Window_SizeGadget |
#PB_Window_MinimizeGadget | #PB_Window_TitleBar)
433
434 ; Check that the OpenWindow command worked.
435 If ThisWindow
436     ; Set minimum window dimensions.
437     WindowBounds(ThisWindow, 135, 130, #PB_Ignore, 130)
438
439     ; Convert the window reference to a string to use as the map key
value.
440     ThisKey = StrU(ThisWindow)
441
442     ; Allocate memory to store the gadget references in.
443     *ThisData = AllocateMemory(SizeOf(TRACKWINDOW))
444 EndIf
445
446 ; Check that the memory allocation worked.
447 If *ThisData
448     ; Store the window reference and memory pointer values in the map.
449     ActiveWindows(ThisKey) = *ThisData
450
451     ; Set the window class.
452     *ThisData\WindowClass = #WindowClassTrack
453
454     ; Create the menu bar.
455     *ThisData\Menu = CreateMenu(#PB_Any, WindowID(ThisWindow))
456
457     ; If the menu creation worked, create the menu items.
458     If *ThisData\Menu
459         MenuItem("Window")
460         MenuItem(#MenuNewButton, "New Button Window")
461         MenuItem(#MenuNewDate, "New Date Window")
462         MenuItem(#MenuNewTrack, "New Track Window")
463         MenuItem(#MenuClose, "Close Window")
464     EndIf
465
466     ; Create the window gadgets.
467     *ThisData\TrackBar1 = TrackBarGadget(#PB_Any, 10, 10, 375, 25, 0,
100, #PB_TrackBar_Ticks)
468     *ThisData\TrackBar2 = TrackBarGadget(#PB_Any, 10, 40, 375, 25, 0,
100, #PB_TrackBar_Ticks)
469     *ThisData\Label = TextGadget(#PB_Any, 10, 75, 80, 25,
"Difference:")
470     *ThisData\Difference = StringGadget(#PB_Any, 90, 75, 290, 25,
"0", #PB_String_ReadOnly)
471 Else
472     ; Memory allocation failed.
473     CloseWindow(ThisWindow)
474 EndIf
475
476 ; Set the return value.
477 If ThisWindow > 0 And *ThisData > 0
478     ; Return the reference to the new window.

```

```

479     ProcedureReturn ThisWindow
480 Else
481     ; Return 0
482     ProcedureReturn 0
483 EndIf
484 EndProcedure
485
486 Procedure.i DestroyTrackWindow(Window.i)
487     ; Remove Window from the ActiveWindows map, release the allocated
memory,
488     ; close the window and set the quit flag, if appropriate.
489     Shared EventQuit, ActiveWindows()
490     Protected *ThisData.DATEWINDOW
491     Protected.s ThisKey
492
493     ; Convert the integer Window to a string.
494     ThisKey = StrU(Window)
495
496     ; Obtain the reference structure pointer.
497     *ThisData = ActiveWindows(ThisKey)
498
499     ; Check that a valid pointer was obtained.
500     If *ThisData = 0
501         ProcedureReturn #False
502     EndIf
503
504     ; Check that it is the correct window type, if not stop as this
procedure can't destroy this window.
505     If *ThisData\WindowClass <> #WindowClassTrack
506         ProcedureReturn #False
507     EndIf
508
509     ; Release the memory allocation.
510     FreeMemory(*ThisData)
511
512     ; Delete the map entry.
513     DeleteMapElement(ActiveWindows(), ThisKey)
514
515     ; Close the window.
516     CloseWindow(Window)
517
518     ; Check if there are still open windows.
519     If MapSize(ActiveWindows()) = 0
520         EventQuit = #True
521     EndIf
522
523     ; Set the successful return value.
524     ProcedureReturn #True
525 EndProcedure
526
527 Procedure.i ResizeTrackWindow(Window.i)
528     ; Resize the child gadgets on Window.
529     Shared ActiveWindows()
530     Protected *ThisData.TRACKWINDOW
531     Protected.i X, Y, W, H
532     Protected.s ThisKey
533
534     ; Obtain the reference structure pointer.
535     ThisKey = StrU(Window)

```

```

536 *ThisData = ActiveWindows(ThisKey)
537
538 ; Check that a valid pointer was obtained, if not stop.
539 If *ThisData = 0
540     ProcedureReturn #False
541 EndIf
542
543 ; Check that it is the correct window type, if not stop.
544 If *ThisData\WindowClass <> #WindowClassTrack
545     ProcedureReturn #False
546 EndIf
547
548 ; Resize track bars.
549 W = WindowWidth(Window) - 20
550 ResizeGadget(*ThisData\TrackBar1, #PB_Ignore, #PB_Ignore, W,
#PB_Ignore)
551 ResizeGadget(*ThisData\TrackBar2, #PB_Ignore, #PB_Ignore, W,
#PB_Ignore)
552
553 ; Resize string.
554 W = WindowWidth(Window) - 110
555 ResizeGadget(*ThisData\Difference, #PB_Ignore, #PB_Ignore, W,
#PB_Ignore)
556
557 ProcedureReturn #True
558 EndProcedure
559
560 Procedure.i EventsTrackWindow(Window, Gadget, Type)
561 ; Handle events for a Track window.
562 Shared Buttons, ActiveWindows()
563 Protected *ThisData.TRACKWINDOW
564 Protected.i NewValue
565 Protected.s ThisKey
566
567 ; Convert the integer Window to a string.
568 ThisKey = StrU(Window)
569
570 ; Obtain the reference structure pointer.
571 *ThisData = ActiveWindows(ThisKey)
572
573 ; Check that a valid pointer was obtained, if not stop.
574 If *ThisData = 0
575     ProcedureReturn #False
576 EndIf
577
578 ; Check that it is the correct window type, if not stop.
579 If *ThisData\WindowClass <> #WindowClassTrack
580     ProcedureReturn #False
581 EndIf
582
583 Select Gadget
584     Case *ThisData\TrackBar1, *ThisData\TrackBar2
585         NewValue = GetGadgetState(*ThisData\TrackBar1) -
GetGadgetState(*ThisData\TrackBar2)
586         SetGadgetText(*ThisData\Difference, Str(NewValue))
587
588     Case *ThisData\Label, *ThisData\Difference
589         ; Do nothing.
590 EndSelect

```

```

591 EndProcedure
592
593 ; - Main
594
595 ; Create the first window.
596 EventWindow = CreateButtonWindow()
597 ResizeButtonWindow(EventWindow)
598
599 ; - Event loop
600 Repeat
601     Event = WaitWindowEvent()
602     EventWindow = EventWindow()
603     EventWindowKey = StrU(EventWindow)
604     EventGadget = EventGadget()
605     EventType = EventType()
606     EventMenu = EventMenu()
607     *EventGadgets = ActiveWindows(EventWindowKey)
608
609     Select Event
610         Case #PB_Event_Gadget
611             ; Check that a valid pointer was obtained.
612             If *EventGadgets > 0
613                 ; Use *EventGadgets\WindowClass to dispatch events to the
correct event procedure.
614                 Select *EventGadgets\WindowClass
615                     Case #WindowClassButton
616                         EventsButtonWindow(EventWindow, EventGadget, EventType)
617
618                     Case #WindowClassDate
619                         EventsDateWindow(EventWindow, EventGadget, EventType)
620
621                     Case #WindowClassTrack
622                         EventsTrackWindow(EventWindow, EventGadget, EventType)
623
624                     Default
625                         ; Do nothing
626                 EndSelect
627             EndIf
628
629         Case #PB_Event_Menu
630             Select EventMenu
631                 Case #MenuNewButton
632                     EventWindow = CreateButtonWindow()
633                     If EventWindow > 0
634                         ResizeButtonWindow(EventWindow)
635                     EndIf
636
637                 Case #MenuNewDate
638                     EventWindow = CreateDateWindow()
639                     If EventWindow > 0
640                         ResizeDateWindow(EventWindow)
641                     EndIf
642
643                 Case #MenuNewTrack
644                     EventWindow = CreateTrackWindow()
645                     If EventWindow > 0
646                         ResizeTrackWindow(EventWindow)
647                     EndIf
648

```

```

649         Case #MenuClose
650             ; Check that a valid pointer was obtained.
651             If *EventGadgets > 0
652                 ; Use *EventGadgets\WindowClass to call the correct
destroy window procedure.
653                 Select *EventGadgets\WindowClass
654                     Case #WindowClassButton
655                         DestroyButtonWindow(EventWindow)
656
657                     Case #WindowClassDate
658                         DestroyDateWindow(EventWindow)
659
660                     Case #WindowClassTrack
661                         DestroyTrackWindow(EventWindow)
662
663                     Default
664                         ; Do nothing
665                     EndSelect
666             EndIf
667         EndSelect
668
669     Case #PB_Event_CloseWindow
670         ; Check that a valid pointer was obtained.
671         If *EventGadgets > 0
672             ; Use *EventGadgets\WindowClass to call the correct destroy
window procedure.
673             Select *EventGadgets\WindowClass
674                 Case #WindowClassButton
675                     DestroyButtonWindow(EventWindow)
676
677                 Case #WindowClassDate
678                     DestroyDateWindow(EventWindow)
679
680                 Case #WindowClassTrack
681                     DestroyTrackWindow(EventWindow)
682
683                 Default
684                     ; Do nothing
685             EndSelect
686         EndIf
687
688     Case #PB_Event_SizeWindow
689         If *EventGadgets > 0
690             ; Use *EventGadgets\WindowClass to call the correct resize
window procedure.
691             Select *EventGadgets\WindowClass
692                 Case #WindowClassButton
693                     ResizeButtonWindow(EventWindow)
694
695                 Case #WindowClassDate
696                     ResizeDateWindow(EventWindow)
697
698                 Case #WindowClassTrack
699                     ResizeTrackWindow(EventWindow)
700
701                 Default
702                     ; Do nothing
703             EndSelect
704         EndIf

```



```
705  
706     EndSelect  
707  
708     Until EventQuit = #True
```

UserGuide Navigation

< Previous: Dynamic numbering of windows and gadgets || Overview || Next: Other Compiler keywords
>

Chapter 89

UserGuide - Structuring code in Procedures

We're going to revisit the file properties example again. This time to introduce procedures and to address some of the limitations identified in the program in previous items.

```
1 ; The structure for file information as before.
2 Structure FILEITEM
3     Name.s
4     Attributes.i
5     Size.q
6     DateCreated.i
7     DateAccessed.i
8     DateModified.i
9 EndStructure
10
11 ; This is a constant to identify the window.
12 Enumeration
13     #WindowFiles
14 EndEnumeration
15
16 ; This is an enumeration to identify controls that will appear on the
    window.
17 Enumeration
18     #FolderButton
19     #UpdateButton
20     #FolderText
21     #FilesList
22 EndEnumeration
23
24 Procedure FilesExamine(Folder.s, List Files.FILEITEM())
25     ; Obtains file properties from Folder into Files.
26
27     Protected.l Result
28
29     ; Clear current contents.
30     ClearList(Files())
31
32     ; Open the directory to enumerate its contents.
33     Result = ExamineDirectory(0, Folder, " *.*")
34
35     ; If this is ok, begin enumeration of entries.
36     If Result
```

```

37      ; Loop through until NextDirectoryEntry(0) becomes zero -
      indicating that there are no more entries.
38      While NextDirectoryEntry(0)
39          ; If the directory entry is a file, not a folder.
40          If DirectoryEntryType(0) = #PB_DirectoryEntry_File
41              ; Add a new element to the list.
42              AddElement(Files())
43              ; And populate it with the properties of the file.
44              Files()\Name = DirectoryEntryName(0)
45              Files()\Size = DirectoryEntrySize(0)
46              Files()\Attributes = DirectoryEntryAttributes(0)
47              Files()\DateCreated = DirectoryEntryDate(0, #PB_Date_Created)
48              Files()\DateAccessed = DirectoryEntryDate(0,
#PB_Date_Accessed)
49              Files()\DateModified = DirectoryEntryDate(0,
#PB_Date_Modified)
50              EndIf
51          Wend
52
53      ; Close the directory.
54      FinishDirectory(0)
55      EndIf
56
57      ; Sort the list into ascending alphabetical order of file name.
58      SortStructuredList(Files(), #PB_Sort_Ascending,
OffsetOf(FILEITEM\Name), #PB_String)
59      EndProcedure
60
61      Procedure.s FolderSelect(Folder.s)
62          ; Displays a path requester and returns the new path, or the old
        one if the requester is cancelled.
63          Protected.s SelectedPath
64
65          SelectedPath = PathRequester("Choose a folder.", Folder)
66
67          If SelectedPath = ""
68              SelectedPath = Folder
69          EndIf
70
71          ProcedureReturn SelectedPath
72      EndProcedure
73
74      Procedure LabelUpdate(Folder.s)
75          ; Updates the folder label.
76          SetGadgetText(#FolderText, Folder)
77      EndProcedure
78
79      Procedure ListLoad(ListView.l, List Files.FILEITEM())
80          ; Load the files properties from list Files() into the list view
        #FilesList.
81          Protected.s Access, Attrib, Create, Folder, Modify, Msg, Num, Size
82
83          ; Remove previous contents.
84          ClearGadgetItems(ListView)
85
86          ForEach Files()
87              ; Display the item number and file name.
88              Num = StrU(ListIndex(Files()) + 1)
89

```

```

90      ; These lines convert the three date values to something more
familiar.
91      Create = FormatDate("%dd/%mm/%yyyy", Files()\DateCreated)
92      Access = FormatDate("%dd/%mm/%yyyy", Files()\DateAccessed)
93      Modify = FormatDate("%dd/%mm/%yyyy", Files()\DateModified)
94
95      ; Convert the file size to a padded string the same as with the
index value above,
96      ; but allow space for the maximum size of a quad.
97      Size = StrU(Files()\Size)
98
99      ; Convert the attributes to a string, for now.
100     Attrb = StrU(Files()\Attributes)
101
102     ; Build a row string.
103     ; The Line Feed character 'Chr(10)' tells the gadget to move to
the next column.
104     Msg = Num + Chr(10) + Files()\Name + Chr(10) + Create + Chr(10) +
Access + Chr(10) + Modify + Chr(10) + Attrb + Chr(10) + Size
105
106     ; Add the row to the list view gadget.
107     AddGadgetItem(#FilesList, -1, Msg)
108     Next Files()
109 EndProcedure
110
111 Procedure WindowCreate()
112     ; Creates the wdwFiles window.
113     Protected Flags
114
115     ; This line defines a flag for the window attributes by OR-ing
together the desired attribute constants.
116     Flags = #PB_Window_SystemMenu | #PB_Window_SizeGadget |
#PB_Window_MinimizeGadget | #PB_Window_MaximizeGadget |
#PB_Window_TitleBar
117
118     ; Open a window.
119     OpenWindow(#WindowFiles, 50, 50, 450, 400, "File Properties", Flags)
120     ; A button to choose a folder.
121     ButtonGadget(#FolderButton, 5, 5, 100, 30, "Select Folder")
122     ; A button to update the list.
123     ButtonGadget(#UpdateButton, 112, 5, 100, 30, "Update List")
124     ; A text gadget to show the name of the folder.
125     TextGadget(#FolderText, 5, 40, 400, 25, "")
126     ; A list icon gadget to hold the file list and properties.
127     ListIconGadget(#FilesList, 5, 70, 400, 326, "#", 35)
128     ; Add columns to the ListIconGadget to hold each property.
129     AddGadgetColumn(#FilesList, 1, "Name", 200)
130     AddGadgetColumn(#FilesList, 2, "Created", 100)
131     AddGadgetColumn(#FilesList, 3, "Accessed", 100)
132     AddGadgetColumn(#FilesList, 4, "Modified", 100)
133     AddGadgetColumn(#FilesList, 5, "Attributes", 150)
134     AddGadgetColumn(#FilesList, 6, "Size", 100)
135 EndProcedure
136
137 Procedure WindowDestroy()
138     ; Closes the window.
139     ; If necessary, you could do other tidying up jobs here too.
140     CloseWindow(#WindowFiles)
141 EndProcedure

```

```

142
143 Procedure WindowResize()
144     ; Resizes window gadgets to match the window size.
145     ResizeGadget(#FolderText, #PB_Ignore, #PB_Ignore,
146     WindowWidth(#WindowFiles) - 10, #PB_Ignore)
147     ResizeGadget(#FilesList, #PB_Ignore, #PB_Ignore,
148     WindowWidth(#WindowFiles) - 10, WindowHeight(#WindowFiles) - 74)
149 EndProcedure
150
151 ;- Main
152 ; Now we define a list of files using the structure previously
153 ; specified.
154 NewList Files.FILEITEM()
155
156 ; And some working variables to make things happen.
157 Define.s Folder
158 Define.l Event, EventWindow, EventGadget, EventType, EventMenu
159
160 ; This function gets the home directory for the logged on user.
161 Folder = GetHomeDirectory()
162
163 ; Create the window and set the initial contents.
164 WindowCreate()
165 WindowResize()
166 LabelUpdate(Folder)
167 FilesExamine(Folder, Files())
168 ListLoad(#FilesList, Files())
169
170 ;- Event Loop
171 Repeat
172     ; Wait until a new window or gadget event occurs.
173     Event = WaitWindowEvent()
174     EventWindow = EventWindow()
175     EventGadget = EventGadget()
176     EventType = EventType()
177
178     ; Take some action.
179     Select Event
180     Case #PB_Event_Gadget
181         ; A gadget event occurred.
182         If EventGadget = #FolderButton
183             ; The folder button was clicked.
184             Folder = FolderSelect(Folder)
185             LabelUpdate(Folder)
186             FilesExamine(Folder, Files())
187             ListLoad(#FilesList, Files())
188
189             ElseIf EventGadget = #UpdateButton
190                 ; The update button was clicked.
191                 FilesExamine(Folder, Files())
192                 ListLoad(#FilesList, Files())
193
194             ElseIf EventGadget = #FolderText
195                 ; Do nothing here.
196
197             ElseIf EventGadget = #FilesList
198                 ; Do nothing here.
199
200         EndIf
201     EndSelect
202 EndRepeat

```

```

198
199     Case #PB_Event_SizeWindow
200         ; The window was moved or resized.
201         If EventWindow = #WindowFiles
202             WindowResize()
203         EndIf
204
205     Case #PB_Event_CloseWindow
206         ; The window was closed.
207         If EventWindow = #WindowFiles
208             WindowDestroy()
209             Break
210
211         EndIf
212     EndSelect
213 ForEver

```

Previously, we mentioned four limitations to this program. This new version uses procedures to address three of them.

1) You couldn't choose a folder to show.

The "FolderSelect" procedure shows a path requester to allow the user to select a folder. The variable "Folder" is updated with the result of this procedure. The button also calls "LabelUpdate", "FilesExamine" and "ListLoad" to display the contents of the new folder in the window .

2) You can't update the list contents without closing and restarting the program.

Now, when the "Update List" button is clicked, "FilesExamine" and "ListLoad" are called again to update the display.

3) If you resize the window, the gadgets don't resize with it.

The "WindowResize" procedure is called in the event loop to resize the gadgets when the form is resized. Also, although this program didn't really need to, this procedure is called after calling "WindowCreate" to make sure the gadgets are the right size initially.

Notice how several of the procedures are called more than once to perform similar but not identical functions. This improves the efficiency of the program.

We have one final limitation to overcome in a later item.

UserGuide Navigation

< Previous: Displaying graphics output & simple drawing || Overview || Next: Compiler directives >

Chapter 90

UserGuide - String Manipulation

The following example shows step by step the different commands of the string library - their purpose and their correct use.

```
1  Define.s String1, String2, String3
2
3  String1 = "The quick brown fox jumps over the lazy dog."
4
5  ; Left returns a number of characters from the left hand end of a
   string.
6  ; Mid returns a number of characters from the given start location in
   the middle of a string.
7  ; Right returns a number of characters from the right hand end of a
   string.
8  ; Space returns the specified number of space characters as a string.
9  ; Shows "The brown dog."
10 Debug "* Left, Mid and Right"
11 String2 = Left(String1, 3) + Space(1) + Mid(String1, 11, 5) +
   Space(1) + Right(String1, 4)
12 Debug String2
13
14 ; CountString returns the number of instances of the second string in
   the first string, it is case sensitive.
15 ; Shows 1.
16 Debug "* CountString"
17 Debug CountString(String1, "the")
18
19 ; However the LCase (and UCase) functions can be used to switch a
   string to all lower (or upper) case
20 ; Shows 2
21 Debug "* CountString and LCase"
22 String2 = LCase(String1)
23 Debug CountString(String2, "the")
24
25 ; FindString can be used to find the location of one string within
   another.
26 ; Shows 17.
27 Debug "* FindString"
28 Debug FindString(String1, "fox")
29
30 ; RemoveString can be used to remove one string from within another.
31 ; Shows The quick fox jumps over the lazy dog.
32 Debug "* RemoveString"
```

```

33 String2 = RemoveString(String1, " brown")
34 Debug String2
35
36 ; ReplaceString can be used to change the occurrence of a substring
   within another string.
37 ; Shows The quick brown fox jumps over the sleeping dog.
38 Debug "* ReplaceString"
39 String2 = ReplaceString(String1, "lazy", "sleeping")
40 Debug String2
41
42 ; StringByteLength returns the length of a string in bytes in the
   specified format, or the current default
43 ; if one is not specified (excluding the terminating null).
44 Debug "* StringByteLength"
45 ; Shows 44.
46 Debug StringByteLength(String1, #PB_Ascii)
47 ; Shows 88.
48 Debug StringByteLength(String1, #PB_Unicode)
49
50 ; StringField can be used to obtain an indexed substring from a
   target string.
51 ; Useful for converting strings to lists for example.
52 ; StringField will work with space as a delimiter too
53 ; but hopefully this example makes the functions behaviour more
   apparent.
54 ; Shows jumps.
55 Debug "* StringField"
56 String2 = ReplaceString(String1, " ", "\")
57 Debug String2
58 String3 = StringField(String2, 5, "\")
59 Debug String3
60
61 ; Trim removes white space characters from the start and end of a
   given string.
62 ; Similarly, LTrim acts on the left hand end (start) of a string and
   RTrim the right hand end.
63 Debug "* Trim, LTrim and RTrim"
64 String2 = Space(10) + String1 + Space(8)
65 Debug #DQUOTE$ + String2 + #DQUOTE$
66 String3 = Trim(String2)
67 Debug #DQUOTE$ + String3 + #DQUOTE$
68 String3 = LTrim(String2)
69 Debug #DQUOTE$ + String3 + #DQUOTE$
70 String3 = RTrim(String2)
71 Debug #DQUOTE$ + String3 + #DQUOTE$
72
73 ; LSet sets a string to be a specific length from the left hand end,
   padding with spaces,
74 ; or other specified character, as necessary.
75 ; If the string is already longer than the specified length it will
   be truncated.
76 Debug "*LSet"
77 Debug LSet("Abc", 10, "*")
78 Debug LSet("Abcd", 10, "*")
79 Debug LSet("Abcde", 10, "*")
80
81 ; Similarly RSet pads a string from its right hand end.
82 Debug "* RSet"
83 Debug RSet("1.23", 10, "0")

```



```

84 | Debug RSet("10.23", 10, "0")
85 | Debug RSet("100.23", 10, "0")
86 |
87 | ; Str converts a signed quad value to a string, similarly StrF
   |   converts floats,
88 | ; StrD converts doubles and StrU converts unsigned values, these two
   |   function have an optional
89 | ; parameter to specify the number of decimal places to show.
90 | Debug "* Str, StrF and StrD"
91 | Debug Str(100)
92 | Debug StrF(1.234, 3)
93 | Debug StrD(123456.789, 3)
94 |
95 | ; Val will convert a string value into its numeric (quad) equivalent.
96 | ; ValD and ValF perform the same function for floats and doubles.
97 | Debug "* Val"
98 | Debug Val("123")
99 |
100 | ; Bin will convert a numeric value into its binary equivalent.
101 | ; Hex will convert one into its hexadecimal equivalent.
102 | Debug "* Bin and Hex"
103 | Debug Bin(19)
104 | Debug Hex(19)

```

UserGuide Navigation

< Previous: Loops || Overview || Next: Storing data in memory >

Chapter 91

UserGuide - Displaying text output (Console)

In the previous topic Input & Output you already saw an overview about the different possibilities to output text to the user, and in the topic Storing Data in Memory , we started to build a small application to display the properties of files in a particular folder to the debug window .

Now we're going to revisit this example to improve the data output section to resolve some issues with using the debug window. Firstly, this window is only available in the PureBasic IDE which means its only useful to programmers, secondly it doesn't really give us much control over how our output looks. PureBasic provides a text mode window, or console window , which can be used in compiled programs. So let's update our example to use it instead.

First, we will need some extra working variables to make this work properly. Amend the variable definitions like this:

```
1  ...
2
3  ; Now we define a new list of files using the structure previously
   specified
4  NewList Files.FILEITEM()
5  Define.s Access, Attrib, Create, Folder, Modify, Msg, Num, Size
6  Define.l Result
7
8  ...
```

Next, remove the output section of code completely, from the comment line:

```
1  ; If there are some entries in the list, show the results in the
   debug window.
2  ...
```

Now replace this with:

```
1  ; Open a text mode screen to show the results.
2  OpenConsole()
3
4  ; Display a title.
5  ; PrintN displays the string given in the console window and moves
   the print position to the start of the next line afterwards.
6  ; Space(n) returns n spaces in a string.
7  PrintN("File list of " + Folder + ".")
8  PrintN("-----")
9  Msg = "Num Name"
10 PrintN(Msg)
```

```

11  Msg = Space(4) + "Create" + Space(5) + "Access" + Space(5) + "Modify"
    + Space(5) + "Attrib Size"
12  PrintN(Msg)
13
14  ; Loop through the list to display the results.
15  ForEach Files()
16
17      ; Tabulate the number of the list index.
18      ; ListIndex() returns the current position in the list, counting
    from zero.
19      ; StrU converts an unsigned number into a string.
20      ; RSet pads a string to a set length with the necessary number of a
    specified character at the front.
21      ; Here we use it to make sure all the index numbers are padded to 3
    characters in length.
22      Num = RSet(StrU(ListIndex(Files()) + 1), 3, " ")
23
24      ; Display the item number and file name.
25      Msg = Num + " " + Files()\Name
26      PrintN(Msg)
27
28      ; These lines convert the three date values to something more
    familiar.
29      Create = FormatDate("%dd/%mm/%yyyy", Files()\DateCreated)
30      Access = FormatDate("%dd/%mm/%yyyy", Files()\DateAccessed)
31      Modify = FormatDate("%dd/%mm/%yyyy", Files()\DateModified)
32
33      ; Convert the file size to a padded string the same as with the
    index value above,
34      ; but allow space for the maximum size of a quad.
35      Size = RSet(StrU(Files()\Size), 19)
36
37      ; Convert the attributes to a string, for now.
38      Attrib = RSet(StrU(Files()\Attributes), 6, " ")
39
40      ; Display the file's properties.
41      Msg = Space(4) + Create + " " + Access + " " + Modify + " " +
    Attrib + " " + Size
42      PrintN(Msg)
43
44      ; Display a blank line.
45      PrintN("")
46
47  Next Files()
48
49  ; Wait for the return key to be displayed, so the results can be
    viewed before the screen closes.
50  PrintN("")
51  PrintN("Press return to exit")
52  Input()

```

All being well the output should appear in a console window looking something like this:

```

1  File List of C:\Documents And Settings\user\
2  -----
3  Num Name
4      Create      Access      Modify      Attrib Size
5      1 NTUSER.DAT
6      03/07/2008  04/04/2011  02/04/2011      34      18874368

```

7						
8	2	kunzip.dll				
9		14/07/2008	04/04/2011	14/07/2008	32	18432
10						
11	3	ntuser.dat.LOG				
12		03/07/2008	04/04/2011	04/04/2011	34	1024
13						
14	4	ntuser.ini				
15		03/07/2008	02/04/2011	02/04/2011	6	278
16						
17		Press Return To exit				

This output is from a Windows XP system, later versions of Windows and Linux or Mac OSX will show different files of course.

Note for Linux/MacOS: Please note, to select "Console" as executable format in the compiler options .

UserGuide Navigation

< Previous: Input & Output || Overview || Next: Building a graphical user interface (GUI) >

Chapter 92

UserGuide - Some Tips & Tricks

”Using a map to index a list”

lists provide a powerful way to build a structured storage system - however they have a disadvantage. If you aren't sure exactly where in the list a particular item is, you must examine each entry in the list to find the right one.

Maps also provide a similar function but are indexed by a key value instead - however they too have a disadvantage, they don't maintain the order elements are inserted into the list.

However by using a combination of the two, you can neatly avoid both of these issues...

This example loads a structured list of book data and builds an index of ISBN numbers using a map . It then demonstrates how to access the list using the index in the map.

```
1  EnableExplicit
2
3  ; A book catalog structure.
4  Structure BOOK
5      Title.s
6      Author.s
7      ISBN13.s
8      Price.d
9  EndStructure
10
11 ; Create a list to hold the catalog entries.
12 NewList Catalog.BOOK()
13 ; Create a map to hold the ISBN index.
14 NewMap ISBN13.1()
15 ; Working variables.
16 Define.l Count, Index
17 Define.s ISBN
18
19 ; Add an empty element at the top of the list.
20 ; The first element in a list is numbered zero, however the map will
   return zero if a requested entry isn't present.
21 ; This empty element avoids a potential problem with an incorrect
   reference to the first catalog item being returned.
22 AddElement(Catalog())
23
24 For Count = 1 To 5
25
26     ; Read the data from the table into the list.
27     AddElement(Catalog())
28     Read.s Catalog()\Title
29     Read.s Catalog()\Author
```

```

30     Read.s Catalog()\ISBN13
31     Read.d Catalog()\Price
32
33     ; Index the ISBN to the map.
34     ISBN13(Catalog()\ISBN13) = ListIndex(Catalog())
35
36 Next Count
37
38 ; Find an entry.
39 ISBN = "978-0340896983"
40 Index = ISBN13(ISBN)
41
42 If Index > 0
43     Debug "Book with ISBN13 " + ISBN + " is at list index " +
44     StrU(Index) + "."
45     Debug ""
46
47     ; We can now directly select the right list element without having
48     to perform a search.
49     SelectElement(Catalog(), Index)
50     Debug "' ' + Catalog()\Title + " ' by " + Catalog()\Author
51     Debug "ISBN: " + Catalog()\ISBN13
52     Debug "Price: " + StrD(Catalog()\Price, 2)
53
54 Else
55     Debug "No book with that ISBN in the catalog."
56
57 EndIf
58
59 End
60
61 ; Some test data.
62 DataSection
63     BookData:
64
65     Data.s "Carte Blanche", "Jeffery Deaver", "978-1444716474"
66     Data.d 19.99
67
68     Data.s "One Day", "David Nicholls", "978-0340896983"
69     Data.d 7.99
70
71     Data.s "Madeleine", "Kate McCann", "978-0593067918"
72     Data.d 20.00
73
74     Data.s "The Dukan Diet", "Dr Pierre Dukan", "978-1444710335"
75     Data.d 8.99
76
77     Data.s "A Game of Thrones", "George R. R. Martin", "978-0006479888"
78     Data.d 9.99
79
80     Data.s "The Help", "Kathryn Stockett", "978-0141039282"
81     Data.d 8.99
82 EndDataSection

```

UserGuide Navigation

< Previous: Advanced functions || Overview

Chapter 93

UserGuide - Variables and Processing of variables

A is an integer - but note that you can't declare variables this way if you use the `EnableExplicit` directive. Outside of a procedure **A** will exist at Main scope, within a procedure it will become local.

```
1 A = 0
```

B is a long integer, **C** is a floating-point number, they will be initialized to zero.

```
1 Define B.l, C.f
```

D and **E** are long integers too. However, they will be initialized to the stated constants. Notice too the alternative syntax of `Define` used.

```
1 Define.l D = 10, E = 20
```

F is a string.

```
1 Define.s F
```

So is **G\$**, however, if you declare strings this way, you must always use the `$` notation.

```
1 G$ = "Hello, "
```

This won't work. (**G** becomes a new integer variable and a compiler error occurs).

```
1 G = "Goodbye, World!"
```

H is an array of 20 strings, array indexing begins at zero.

```
1 Dim H.s(19)
```

Now **H** is an array of 25 strings. If the array is resized larger, its original contents will be preserved.

```
1 ReDim H.s(24)
```

J will appear at Global , rather than Main, scope. It will appear in all procedures, but maybe a better way would be to use the `Shared` keyword inside a procedure on a main scope variable, so that the chances of accidental changes are minimized.

```
1 Global.i J
```


K will be a new, empty, list of strings at main scope.

```
1 NewList K.s()
```

M will be a new, empty, map of strings at main scope.

```
1 NewMap M.s()
```

Note that you can't use the alternative syntax of the Define keyword with NewList or NewMap though. A compiler error will result.

Within a procedure:

```
1 Procedure TestVariables()  
2 ??  
3 ?? ; N and P will be local.  
4 ?? Define.l N  
5 ?? Protected.l P  
6 ??  
7 ?? ; Q will be a static local.  
8 ?? Static.l Q  
9 ??  
10 ?? ; The main scope variable F and the string list K will be  
    available within this procedure.  
11 ?? Shared F, K()  
12 ??  
13 ?? ; The global scope variable J will be available here implicitly.  
14 ??  
15 EndProcedure
```

Using operators on variables:

```
1 ; Add two to A.  
2 A + 2  
3  
4 ; Bitwise Or with 21 (A will become 23)  
5 A | 21  
6  
7 ; Bitwise And with 1 (A will become 1)  
8 A & 1  
9  
10 ; Arithmetic shift left (A will become 2, that is 10 in binary).  
11 A << 1
```

String concatenation:

```
1 G$ + "World!"
```

Add an element to the K list:

```
1 AddElement(K())  
2 K() = "List element one"
```

Add an element to the M map:

```
1 M("one") = "Map element one"
```

UserGuide Navigation

< Previous: First steps || Overview || Next: Constants >

Chapter 94

Unicode

Introduction

Unicode is the term used for extended character sets which allows displaying text in many languages (including those with a lot of different characters like Japanese, Korean etc.). It solves the ASCII problem of having a lot of table dedicated for each language by having a unified table where each character has its own place.

To simplify, unicode can be seen as a big ascii table, which doesn't have 256 characters but up to 65536. Thus, each character in unicode will need 2 bytes in memory (this is important when dealing with pointers for example).

Here are some links to have a better understanding about unicode (must have reading):

[General unicode information](#)

[Unicode and BOM](#)

Unicode and Windows

PureBasic internally uses the UCS2 encoding which is also the format used by the Windows unicode API, so no conversions are needed at runtime when calling an OS function. When dealing with an API function, PureBasic will automatically use the unicode one if available (for example `MessageBox_()` will map to `MessageBoxW()`). All the API structures and constants supported by PureBasic are using unicode version.

UTF-8

UTF-8 is another unicode encoding, which is byte based. Unlike UCS2 which always takes 2 bytes per characters, UTF-8 uses a variable length encoding for each character (up to 4 bytes can represent one character). The biggest advantage of UTF-8 is the fact it doesn't includes null characters in its coding, so it can be edited like a regular text file. Also the ASCII characters from 1 to 127 are always preserved, so the text is kind of readable as only special characters will be encoded. One drawback is its variable length, so it all string operations will be slower due to needed pre-processing to locate a character is in the text.

PureBasic uses UTF-8 by default when writing string to files (File and Preference libraries), so all texts are fully cross-platform.

The PureBasic compiler also handles both Ascii and UTF-8 files (the UTF-8 files need to have the correct BOM header to be handled correctly). Both can be mixed in a single program without problem: an ascii file can include an UTF-8 file and vice-versa, as long as all string-literals (i.e. "x") and character-literals (i.e. 'x') in the ASCII files contain only characters with a code ≤ 127 . When developing, it's recommended to set the IDE in UTF-8 mode, so all the source files will be unicode ready.

Chapter 95

Variables and Types

Variables declaration

To declare a variable in PureBasic, simply type its name. You can also specify the type you want this variable to be. By default, when a data type is not indicated, the data is an integer. Variables do not need to be explicitly declared, as they can be used as "variables on-the-fly". The Define keyword can be used to declare multiple variables in one statement. If you don't assign an initial value to the variable, their value will be 0.

Example

```
1  a.b          ; Declare a variable called 'a' from byte (.b) type.
2  c.l = a*d.w ; 'd' is declared here within the expression and it's a
    word !
```

Notes:

Variable names must not start with a number (0,1,...), contain operators (+,-,...) or special characters (ß,ä,ö,ü,...).

The variables in PureBasic are not case sensitive, so "pure" and "PURE" are the same variable.

If you don't need to change the content of a variable during the program flow (e.g. you're using fixed values for ID's etc.), you can also take a look at constants as an alternative.

To avoid typing errors etc. it's possible to force the PureBasic compiler to always want a declaration of variables, before they are first used. Just use EnableExplicit keyword in your source code to enable this feature.

Basic types

PureBasic allows many type variables which can be standard integers, float, double, quad and char numbers or even string characters. Here is the list of the native supported types and a brief description :

Name	Extension	Memory consumption	Range
Byte +127	.b	1 byte	-128 to
Ascii +255	.a	1 byte	0 to
Character +65535	.c	2 bytes	0 to

Word		.w		2 bytes		-32768
to +32767						
Unicode		.u		2 bytes		0 to
+65535						
Long		.l		4 bytes		
-2147483648 to +2147483647						
Integer		.i		4 bytes (using 32-bit compiler)		
-2147483648 to +2147483647						
Integer		.i		8 bytes (using 64-bit compiler)		
-9223372036854775808 to +9223372036854775807						
Float		.f		4 bytes		
unlimited (see below)						
Quad		.q		8 bytes		
-9223372036854775808 to +9223372036854775807						
Double		.d		8 bytes		
unlimited (see below)						
String		.s		string length + 1		
unlimited						
Fixed String		.s{Length}		string length		
unlimited						

Unsigned variables: Purebasic offers native support for unsigned variables with byte and word types via the ascii (.a) and unicode (.u) types. The character (.c) type is an unsigned word in unicode that may be used as an unsigned type.

Notation of string variables: it is possible to use the '\$' as last char of a variable name to mark it as string. This way you can use 'a\$' and 'a.s' as different string variables. Please note, that the '\$' belongs to the variable name and must be always attached, unlike the '.s' which is only needed when the string variable is declared the first time.

```

1  a.s = "One string"
2  a$ = "Another string"
3  Debug a ; will give "One string"
4  Debug a$ ; will give "Another string"

```

Note: The floating numbers (floats + doubles) can also be written like this: 123.5e-20

```

1  value.d = 123.5e-20
2  Debug value ; will give 0.000000000000000001235

```

Operators

Operators are the functions you can use in expressions to combine the variables, constants, and whatever else. The table below shows the operators you can use in PureBasic, in no particular order (LHS = Left Hand Side, RHS = Right Hand Side).

Operator = (Equals)

This can be used in two ways. The first is to assign the value of the expression on the RHS to the variable on the LHS. The second way is when the result of the operator is used in an expression and is to test whether the values of the expressions on the LHS and RHS are the same (if they are the same this operator will return a true result, otherwise it will be false).

Example

```

1  a = b+c      ; Assign the value of the expression "b+c" to the
   variable "a"
2  If abc = def ; Test if the values of abc and def are the same, and
   use this result in the If command

```

Operator + (Plus)

Gives a result of the value of the expression on the RHS added to the value of the expression on the LHS. If the result of this operator is not used and there is a variable on the LHS, then the value of the expression on the RHS will be added directly to the variable on the LHS.

Example

```

1  number=something+2 ; Adds the value 2 to "something" and uses the
   result with the equals operator
2  variable+expression ; The value of "expression" will be added
   directly to the variable "variable"

```

Operator - (Minus)

Subtracts the value of the expression on the RHS from the value of the expression on the LHS. When there is no expression on the LHS this operator gives the negative value of the value of the expression on the RHS. If the result of this operator is not used and there is a variable on the LHS, then the value of the expression on the RHS will be subtracted directly from the variable on the LHS. This operator cannot be used with string type variables.

Example

```

1  var=#MyConstant-foo ; Subtracts the value of "foo" from "#MyConstant"
   and uses the result with the equals operator
2  another=another+ -var ; Calculates the negative value of "var" and
   uses the result in the plus operator
3  variable-expression ; The value of "expression" will be subtracted
   directly from the variable "variable"

```

Operator * (Multiplication)

Multiplies the value of the expression on the LHS by the value of the expression on the RHS. If the result of this operator is not used and there is a variable on the LHS, then the value of the variable is directly multiplied by the value of the expression on the RHS. This operator cannot be used with string type variables.

Example

```

1  total=price*count ; Multiplies the value of "price" by the value of
   "count" and uses the result with the equals operator
2  variable*expression ; "variable" will be multiplied directly by the
   value of "expression"

```

Operator / (Division)

Divides the value of the expression on the LHS by the value of the expression on the RHS. If the result of this operator is not used and there is a variable on the LHS, then the value of the variable is directly divided by the value of the expression on the RHS. This operator cannot be used with string type variables.

Example

```
1  count=total/price ; Divides the value of "total" by the value of
   "price" and uses the result with the equals operator
2  variable/expression ; "variable" will be divided directly by the
   value of "expression"
```

Operator & (Bitwise AND)

You should be familiar with binary numbers when using this operator. The result of this operator will be the value of the expression on the LHS anded with the value of the expression on the RHS, bit for bit. The value of each bit is set according to the table below. Additionally, if the result of the operator is not used and there is a variable on the LHS, then the result will be stored directly in that variable. This operator cannot be used with strings.

LHS		RHS		Result
0		0		0
0		1		0
1		0		0
1		1		1

Example

```
1  ; Shown using binary numbers as it will be easier to see the result
2  a.w = %1000 & %0101 ; Result will be 0
3  b.w = %1100 & %1010 ; Result will be %1000
4  bits = a & b ; AND each bit of a and b and use result in equals
   operator
5  a & b ; AND each bit of a and b and store result directly in variable
   "a"
```

Operator || (Bitwise OR)

You should be familiar with binary numbers when using this operator. The result of this operator will be the value of the expression on the LHS or'ed with the value of the expression on the RHS, bit for bit. The value of each bit is set according to the table below. Additionally, if the result of the operator is not used and there is a variable on the LHS, then the result will be stored directly in that variable. This operator cannot be used with strings.

LHS		RHS		Result
0		0		0
0		1		1
1		0		1
1		1		1

Example

```
1 ; Shown using binary numbers as it will be easier to see the result
2 a.w = %1000 | %0101 ; Result will be %1101
3 b.w = %1100 | %1010 ; Result will be %1110
4 bits = a | b ; OR each bit of a and b and use result in equals
  operator
5 a | b ; OR each bit of a and b and store result directly in variable
  "a"
```

Operator ! (Bitwise XOR)

You should be familiar with binary numbers when using this operator. The result of this operator will be the value of the expression on the LHS xor'ed with the value of the expression on the RHS, bit for bit. The value of each bit is set according to the table below. Additionally, if the result of the operator is not used and there is a variable on the LHS, then the result will be stored directly in that variable. This operator cannot be used with strings.

LHS		RHS		Result
0		0		0
0		1		1
1		0		1
1		1		0

Example

```
1 ; Shown using binary numbers as it will be easier to see the result
2 a.w = %1000 ! %0101 ; Result will be %1101
3 b.w = %1100 ! %1010 ; Result will be %0110
4 bits = a ! b ; XOR each bit of a and b and use result in equals
  operator
5 a ! b ; XOR each bit of a and b and store result directly in variable
  "a"
```

Operator * * (Bitwise NOT)

You should be familiar with binary numbers when using this operator. The result of this operator will be the not'ed value of the expression on the RHS, bit for bit. The value of each bit is set according to the table below. This operator cannot be used with strings.

RHS		Result
0		1
1		0

Example

```
1 ; Shown using binary numbers as it will be easier to see the result
2 a.w = ~%1000 ; Result will be %0111
3 b.w = ~%1010 ; Result will be %0101
```


Operator () (Parentheses)

You can use sets of parentheses to force part of an expression to be evaluated first, or in a certain order.

Example

```
1  a = (5 + 6) * 3 ; Result will be 33 since the 5+6 is evaluated first
2  b = 4 * (2 - (3 - 4)) ; Result will be 12 since the 3-4 is evaluated
   first, then the 2-result, then the multiplication
```

Operator < (Less than)

This is used to compare the values of the expressions on the LHS and RHS. If the value of the expression on the LHS is less than the value of the expression on the RHS this operator will give a result of true, otherwise the result is false.

Operator > (More than)

This is used to compare the values of the expressions on the LHS and RHS. If the value of the expression on the LHS is more than the value of the expression on the RHS this operator will give a result of true, otherwise the result is false.

Operator <=, =< (Less than or equal to)

This is used to compare the values of the expressions on the LHS and RHS. If the value of the expression on the LHS is less than or equal to the value of the expression on the RHS this operator will give a result of true, otherwise the result is false.

Operator >=, => (More than or equal to)

This is used to compare the values of the expressions on the LHS and RHS. If the value of the expression on the LHS is more than or equal to the value of the expression on the RHS this operator will give a result of true, otherwise the result is false.

Operator <> (Not equal to)

This is used to compare the values of the expressions on the LHS and RHS. If the value of the expression on the LHS is equal to the value of the expression on the RHS this operator will give a result of false, otherwise the result is true.

Operator And (Logical AND)

Can be used to combine the logical true and false results of the comparison operators to give a result shown in the following table.

LHS		RHS		Result

false		false		false

false		true		false
true		false		false
true		true		true

Operator Or (Logical OR)

Can be used to combine the logical true and false results of the comparison operators to give a result shown in the following table.

LHS		RHS		Result

false		false		false
false		true		true
true		false		true
true		true		true

Operator XOr (Logical XOR)

Can be used to combine the logical true and false results of the comparison operators to give a result shown in the following table. This operator cannot be used with strings.

LHS		RHS		Result

false		false		false
false		true		true
true		false		true
true		true		false

Operator Not (Logical NOT)

The result of this operator will be the not'ed value of the logical on the RHS. The value is set according to the table below. This operator cannot be used with strings.

RHS		Result

false		true
true		false

Operator « (Arithmetic shift left)

Shifts each bit in the value of the expression on the LHS left by the number of places given by the value of the expression on the RHS. Additionally, when the result of this operator is not used and the LHS contains a variable, that variable will have its value shifted. It probably helps if you understand binary numbers when you use this operator, although you can use it as if each position you shift by is multiplying by an extra factor of 2.

Example

```

1  a=%1011 << 1 ; The value of a will be %10110. %1011=11, %10110=22
2  b=%111 << 4 ; The value of b will be %1110000. %111=7, %1110000=112
3  c.l=$80000000 << 1 ; The value of c will be 0. Bits that are shifted
    off the left edge of the result are lost.
```

Operator » (Arithmetic shift right)

Shifts each bit in the value of the expression on the LHS right by the number of places given by the value of the expression on the RHS. Additionally, when the result of this operator is not used and the LHS contains a variable, that variable will have its value shifted. It probably helps if you understand binary numbers when you use this operator, although you can use it as if each position you shift by is dividing by an extra factor of 2.

Example

```
1 d=16 >> 1 ; The value of d will be 8. 16=%10000, 8=%1000
2 e.w=%10101010 >> 4 ; The value of e will be %1010. %10101010=170,
  %1010=10. Bits shifted out of the right edge of the result are lost
  (which is why you do not see an equal division by 16)
3 f.b=-128 >> 1 ; The value of f will be -64. -128=%10000000,
  -64=%11000000. When shifting to the right, the most significant bit
  is kept as it is.
```

Operator % (Modulo)

Returns the remainder of the RHS by LHS integer division.

Example

```
1 a=16 % 2 ; The value of a will be 0 as 16/2 = 8 (no remainder)
2 b=17 % 2 ; The value of a will be 1 as 17/2 = 8*2+1 (1 is remaining)
```

Operators shorthands

Every math operators can be used in a shorthand form.

Example

```
1 Value + 1 ; The same as: Value = Value + 1
2 Value * 2 ; The same as: Value = Value * 2
3 Value << 1 ; The same as: Value = Value << 1
```

Note: this can lead to 'unexpected' results in some rare cases, if the assignment is modified before the affection.

Example

```
1 Dim MyArray(10)
2 MyArray(Random(10)) + 1 ; The same as: MyArray(Random(10)) =
  MyArray(Random(10)) + 1, but here Random() won't return the same
  value for each call.
```

Operators priorities

Priority Level	Operators
8 (high)	~, - (negative)
7	<<, >>, %, !
6	!, &
5	*, /
4	+, - (subtraction)
3	>, >=, <=, <, >, <=, <=, <=, <=
2	Not
1 (low)	And, Or, XOr

Structured types

Build structured types, via the [Structure](#) keyword. More information can be found in the structures chapter .

Pointer types

Pointers are declared with a '*' in front of the variable name. More information can be found in the pointers chapter .

Special information about Floats and Doubles

A floating-point number is stored in a way that makes the binary point "float" around the number, so that it is possible to store very large numbers or very small numbers. However, you cannot store very large numbers with very high accuracy (big and small numbers at the same time, so to speak).

Another limitation of floating-point numbers is that they still work in binary, so they can only store numbers exactly which can be made up of multiples and divisions of 2. This is especially important to realize when you try to print a floating-point number in a human readable form (or when performing operations on that float) - storing numbers like 0.5 or 0.125 is easy because they are divisions of 2. Storing numbers such as 0.11 are more difficult and may be stored as a number such as 0.10999999. You can try to display to only a limited range of digits, but do not be surprised if the number displays different from what you would expect!

This applies to floating-point numbers in general, not just those in PureBasic.

Like the name says the doubles have double-precision (64-bit) compared to the single-precision of the floats (32-bit). So if you need more accurate results with floating-point numbers use doubles instead of floats.

The exact range of values, which can be used with floats and doubles to get correct results from arithmetic operations, looks as follows:

Float: +- 1.175494e-38 till +- 3.402823e+38

Double: +- 2.2250738585072013e-308 till +- 1.7976931348623157e+308

More information about the 'IEEE 754' standard you can get on [Wikipedia](#).

Chapter 96

While : Wend

Syntax

```
While <expression>  
    ...  
Wend
```

Description

Wend will loop until the <expression> becomes false. A good point to keep in mind with a **While** test is that if the first test is false, then the program will never enter the loop and will skip this part. A Repeat loop is executed at least once, (as the test is performed after each loop).

With the Break command it is possible to exit the **While : Wend** loop during any iteration, with the Continue command the end of the current iteration may be skipped.

Example

```
1  b = 0  
2  a = 10  
3  While a = 10  
4      b = b+1  
5      If b=10  
6          a=11  
7      EndIf  
8  Wend
```

This program loops until the 'a' value is <> 10. The value of 'a' becomes 11 when b=10, the program will loop 10 times.

Chapter 97

Windows Message Handling

Introduction

The messages for your program will be sent by Windows into a queue, which is worked off only if you want this. Windows sends thousand messages to your program without noticing this directly. For example if you change the status of a gadget (identical whether you add a new entry or change the image of an ImageGadget), a message is sent to the queue of your program. There are two possibilities to receive and to process the Windows messages in PureBasic: `WaitWindowEvent()` and `WindowEvent()`. The difference is, that `WaitWindowEvent()` waits till a message arrives and `WindowEvent()` continues to work also so. The messages in the queue will be processed however only, after you have called `WindowEvent()` or `WaitWindowEvent()`.

Specials of `WindowEvent()`

The command `WindowEvent()` don't wait, until a message arrives, but checks only whether one is in the queue. If yes, the message is processed and `WindowEvent()` gives back the number of the message. If no message lines up, then zero (0) is given back.

Example

```
1 While WindowEvent() : Wend
```

cause, that `WindowEvent()` becomes called as long till it gives back 0, i.e. until all messages of the queue are processed.

It doesn't reach, if you insert a simple '`WindowEvent()`' after a `SetGadgetState()` to process this 1 message. Firstly there still can be other messages in the queue, which have arrived before, and secondly Windows also sends quite a number of other messages, we don't have to take care of... which nevertheless are in the queue.

A simple call of

```
1 WindowEvent()
```

doesn't reach, the code then runs correct under circumstances on one Windows version, but on another version not. The different Windows versions are internally very different, so that one version only sends 1 message but another Windows version sends 5 messages for the same circumstance.

Because of this one always uses for updating:

```
1 While WindowEvent() : Wend
```

Of course there is also the alternative

```
1 Repeat : Until WindowEvent() = 0
```

possible, what isn't rather usual however.

The described method While : WindowEvent() : Wend is frequently useful in connection with the command Delay() , where the loop is inserted BEFORE the Delay(), e.g. to firstly wait for the update of an ImageGadget after changing an image with SetGadgetState().

Chapter 98

With : EndWith

Syntax

```
With <expression>
...
EndWith
```

Description

`With : EndWith` blocks may be used with structure fields in order to reduce the quantity of code and to improve its' readability. This is a compiler directive, and works similarly to a macro , i.e., the specified expression is automatically inserted before any backslash '\' character which have a space or an operator preceding it. The code behaves identically to its' expanded version. `With : EndWith` blocks may not be nested, as this could introduce bugs which are difficult to track under conditions where several statements have been replaced implicitly.

Example

```
1  Structure Person
2      Name$
3      Age.l
4      Size.l
5  EndStructure
6
7  Friend.Person
8
9  With Friend
10     \Name$ = "Yann"
11     \Age   = 30
12     \Size  = 196
13
14     Debug \Size+\Size
15 EndWith
```

Example: Complex example

```
1  Structure Body
2      Weight.l
3      Color.l
```



```
4      Texture.1
5  EndStructure
6
7  Structure Person
8      Name$
9      Age.1
10     Body.Body[10]
11 EndStructure
12
13 Friend.Person
14
15 For k = 0 To 9
16     With Friend\Body[k]
17         \Weight = 50
18         \Color  = 30
19         \Texture = \Color*k
20
21         Debug \Texture
22     EndWith
23 Next
```

Part IV

Library Reference

Chapter 99

2DDrawing

Overview

The 2D drawing library contains all the 2D rendering operations that can be performed on a visual area. Drawing a line, a box, a circle or even text is considered a 2D rendering operation.

The output of the drawing functions is possible on a window, a screen, a sprite, an image or the printer. More information can be found at `StartDrawing()` .

Note: The drawing process starts after calling `StartDrawing()` and must end with `StopDrawing()` .

99.1 Red

Syntax

```
Result = Red(Color)
```

Description

Returns the red component of a color value.

Parameters

Color The color value. This can be a 24-bit RGB or a 32-bit RGBA value.

Return value

Returns the value of the red component. The result will be between 0 and 255.

Remarks

To combine red, green and blue values in order to create a 24-bit RGB color, use the `RGB()` function. These functions are useful to perform Drawing operations .

See Also

Green() , Blue() , Alpha() , RGB() , RGBA()

99.2 Green

Syntax

```
Result = Green(Color)
```

Description

Returns the green component of a color value.

Parameters

Color The color value. This can be a 24-bit RGB or a 32-bit RGBA value.

Return value

Returns the value of the green component. The result will be between 0 and 255.

Remarks

To combine red, green and blue values in order to create a 24-bit RGB color, use the RGB() function. These functions are useful to perform Drawing operations .

See Also

Red() , Blue() , Alpha() , RGB() , RGBA()

99.3 Blue

Syntax

```
Result = Blue(Color)
```

Description

Returns the blue component of a color value.

Parameters

Color The color value. This can be a 24-bit RGB or a 32-bit RGBA value.

Return value

Returns the value of the blue component. The result will be between 0 and 255.

Remarks

To combine red, green and blue values in order to create a 24-bit RGB color, use the RGB() function. These functions are useful to perform Drawing operations .

See Also

Red() , Green() , Alpha() , RGB() , RGBA()

99.4 Alpha

Syntax

```
Result = Alpha(Color)
```

Description

Returns the alpha component of a color value.

Parameters

Color The color value. This must be a 32-bit RGBA value.

Return value

Returns the value of the alpha component. The result will be between 0 and 255. A value of 0 means fully transparent and a value of 255 means fully opaque.

Remarks

To combine red, green, blue and alpha values in order to create a 32-bit RGB color, use the RGBA() function. These functions are useful to perform Drawing operations .

See Also

Red() , Green() , Blue() , RGBA()

99.5 RGB

Syntax

```
Color = RGB(Red, Green, Blue)
```

Description

Returns the 24-bit color value corresponding to the Red, Green, Blue components.

Parameters

Red, Green, Blue The value of the red, green and blue components for the color. Each value must be between 0 and 255.

Return value

Returns the combined color value.

Remarks

To extract the red, green and blue values from a 24 Bit color value, use the Red() , Green() and Blue() functions. These functions are useful to perform Drawing operations .
A color table with common colors is available [here](#) .

See Also

Red() , Green() , Blue() , RGBA()

99.6 RGBA

Syntax

```
Color.q = RGBA(Red, Green, Blue, Alpha)
```

Description

Returns the 32-bit color value corresponding to the Red, Green, Blue and Alpha values.

Parameters

Red, Green, Blue The value of the red, green and blue components for the color. Each value must be between 0 and 255.

Alpha The alpha component of the color. The value must be between 0 and 255. A value of 0 means fully transparent and a value of 255 means fully opaque.

Return value

Returns the combined color value.

Remarks

To extract the red, green, blue or alpha values from a 32-bit color value, use the `Red()` , `Green()` , `Blue()` and `Alpha()` functions. These functions are useful to perform Drawing operations .

Result varies from 0 to 4 294 967 295 shades. It is therefore advisable to use a 'quad', (`Result.q`) and set unused bytes to zero. Indeed, on a 32-Bit operating system, Result is a Long integer (by default) with a range of use from - 2 147 483 648 to + 2 147 483 647, so comparing two colors is hazardous with a Long integer.

Example

```
1  Debug RGBA(0, 0, 0, 0)           ; Completely transparent black
2  Debug RGBA(255, 255, 255, 255) ; White totally opaque
```

Example: Color 24-bits to Color 32-bits

```
1  Alpha = 255
2
3  ; Use a Quad (see remarks)
4  Color24.q = ColorRequester()
5
6  Color32.q = RGBA(Red(Color24), Green(Color24), Blue(Color24), alpha)
7  Color32 = Color32 & $FFFFFFFF ; Zeroing unused bytes
8
9  ; It's also possible to replace the two lines above with:
10 ; Color32 = Color24 | Alpha << 24
11
12 Debug "Red " + Red(Color32)
13 Debug "Green " + Green(Color32)
14 Debug "Blue " + Blue(Color32)
15 Debug "Alpha " + Alpha(Color32)
```

See Also

`Red()` , `Green()` , `Blue()` , `Alpha()` , `RGB()`

99.7 AlphaBlend

Syntax

```
Color = AlphaBlend(Color1, Color2)
```

Description

Returns the resulting 32-bit color from blending the two 32-bit colors.

Parameters

Color1 The color that will be in front. It is blended on to 'Color2'.

Color2 The color that will be in the back.

Return value

Returns the combined color.

Remarks

The `RGBA()` function can be used to create 32-bit colors with alpha transparency. These functions are useful to perform Drawing operations .

See Also

`RGBA()`

99.8 BackColor

Syntax

```
BackColor(Color)
```

Description

Set the default background color for graphic functions and text display.

Parameters

Color The new color to be used as the background color. This color can be in RGB or RGBA format. Whether or not the alpha channel is used depends on the drawing mode .
A color table with common colors is available [here](#) .

Return value

None.

See Also

`FrontColor()` , `RGB()` , `RGBA()` , `DrawingMode()`

99.9 Box

Syntax

```
Box(x, y, Width, Height [, Color])
```


Description

Draw a box of given dimensions on the current output. The filling mode is determined by `DrawingMode()` . The current output is set with `StartDrawing()` .

Parameters

x, y, Width, Height The position and size of the box in the current drawing output.

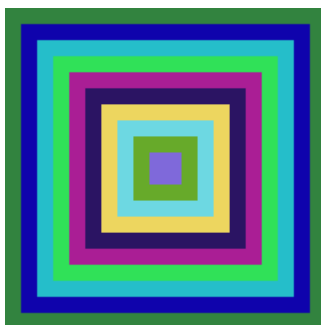
Color (optional) The color to be used for the box. If this parameter is not specified, the default color set with `FrontColor()` will be used. This color can be in RGB or RGBA format.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 200, 200, "2DDrawing Example",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      If CreateImage(0, 200, 200) And StartDrawing(ImageOutput(0))
4          y = 0
5          For x = 0 To 95 Step 10
6              Box(x, y, 200-2*x, 200-2*y, RGB(Random(255), Random(255),
7                  Random(255)))
8              y + 10          ; the same as y = y + 10
9          Next x
10         StopDrawing()
11         ImageGadget(0, 0, 0, 200, 200, ImageID(0))
12     EndIf
13
14     Repeat
15         Event = WaitWindowEvent()
16         Until Event = #PB_Event_CloseWindow
17     EndIf
```



See Also

`RoundBox()` , `Line()` , `Circle()` , `Ellipse()` `FrontColor()` , `RGB()` , `RGBA()` , `DrawingMode()`

99.10 RoundBox

Syntax

```
RoundBox(x, y, Width, Height, RoundX, RoundY [, Color])
```

Description

Draw a box of the given dimensions with rounded corners on the current output. The filling mode is determined by `DrawingMode()`. The current output is set with `StartDrawing()`.

Parameters

x, y, Width, Height The position and size of the box in the current drawing output.

RoundX, RoundY The radius of the rounded corners in the x and y direction.

Color (optional) The color to be used for the round box. If this parameter is not specified, the default color set with `FrontColor()` will be used. This color can be in RGB or RGBA format.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 200, 200, "2DDrawing Example",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      If CreateImage(0, 200, 200) And StartDrawing(ImageOutput(0))
4          y = 0
5          For x = 0 To 95 Step 10
6              RoundBox(x, y, 200-2*x, 200-2*y, 20, 20, RGB(Random(255),
7                  Random(255), Random(255)))
8              y + 10
9          Next x
10         StopDrawing()
11         ImageGadget(0, 0, 0, 200, 200, ImageID(0))
12     EndIf
13     Repeat
14         Event = WaitWindowEvent()
15     Until Event = #PB_Event_CloseWindow
16 EndIf
```



See Also

Box() , Line() , Circle() , Ellipse() FrontColor() , RGB() , RGBA() , DrawingMode()

99.11 Circle

Syntax

```
Circle(x, y, Radius [, Color])
```

Description

Draw a circle on the current output. The filling mode is determined by DrawingMode() . The current output is set with StartDrawing() .

Parameters

x, y The position of the center pixel of the circle.

Radius The radius of the circle. This radius does not include the center pixel.

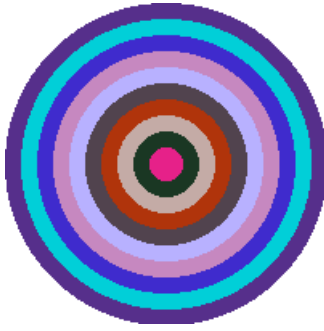
Color (optional) The color to be used for the circle. If this parameter is not specified, the default color set with FrontColor() will be used. This color can be in RGB or RGBA format.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 200, 200, "2DDrawing Example",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      If CreateImage(0, 200, 200) And StartDrawing(ImageOutput(0))
3          Box(0, 0, 200, 200, RGB(255, 255, 255))
4          For Radius = 100 To 10 Step -10
5              Circle(100, 100, Radius, RGB(Random(255), Random(255),
Random(255)))
6          Next
7          StopDrawing()
8          ImageGadget(0, 0, 0, 200, 200, ImageID(0))
9      EndIf
10
11     Repeat
12         Event = WaitWindowEvent()
13     Until Event = #PB_Event_CloseWindow
14 EndIf
```



See Also

`Box()` , `RoundBox()` , `Line()` , `Ellipse()` `FrontColor()` , `RGB()` , `RGBA()` , `DrawingMode()`

99.12 DrawImage

Syntax

```
DrawImage(ImageID, x, y [, Width, Height])
```

Description

Draws an image to the current drawing output. The filling mode is determined by `DrawingMode()` . The current output is set with `StartDrawing()` .

Parameters

ImageID The ID of the image to draw. This value can be obtained using the `ImageID()` function from the image library.

x, y The position of the top/left corner of the image in the drawing output.

Width, Height (optional) The image will be resized and drawn. If these parameters are not specified then the image will be drawn in its original size.

Return value

None.

Remarks

The image will be alpha-blended if the current `DrawingMode()` specifies one of the alpha-blending flags, otherwise the image is just copied to the output. To draw an image using alpha-blending in any case, use the `DrawAlphaImage()` command.

A number of backends (including `WindowOutput()` on Linux) can not ignore the alphachannel when drawing images. So `DrawImage()` is equal to `DrawAlphaImage()` there. If you want consistent results, just use `DrawAlphaImage()` which should produce the same output on all backends.

See Also

`DrawAlphaImage()` , `ImageID()`

99.13 DrawAlphaImage

Syntax

```
DrawAlphaImage(ImageID, x, y [, ConstAlpha])
```

Description

Draws an image to the current drawing output using alpha-blending. The filling mode is determined by `DrawingMode()` . The current output is set with `StartDrawing()` .

Parameters

ImageID The ID of the image to draw. This value can be obtained using the `ImageID()` function from the image library.

x, y The position of the top/left corner of the image in the drawing output.

ConstAlpha (optional) The amount of extra transparency to apply to the image when drawing. The value can range from 0 (fully transparent) to 255 (fully opaque). Even images that have no alpha channel themselves can be drawn with transparency this way.

Return value

None.

Remarks

This command works on all drawing outputs, even those that do not support the alpha-blending flags in `DrawingMode()` . The image will be drawn at its original size. `ResizeImage()` can be used to change the size of an image.

This command cannot be used to draw Icons (loaded from .ico files).

See Also

`DrawImage()` , `ImageID()`

99.14 DrawingBuffer

Syntax

```
*Buffer = DrawingBuffer()
```

Description

Returns the drawing buffer for direct pixel manipulation.

Parameters

None.

Return value

Returns the pixel data pointer if direct access is possible or zero if the pixel data cannot be accessed directly.

Remarks

This function has to be called again if other drawing commands of this library where used since the last pixel manipulation. Once `StopDrawing()` has been called, the buffer is invalidated and can no more be used.

This function is for advanced programmers only. To get more information about the buffer, the following functions are available: `DrawingBufferPixelFormat()` and `DrawingBufferPitch()` .

The returned address can be directly in video memory if the output is `ScreenOutput()` or `SpriteOutput()` and allows very fast pixel manipulation. With `ImageOutput()` this command allows direct access to the pixels of the target image.

Example

For an example of how the drawing buffer can be used to create a nice visual effect, take a look here:

See Also

`DrawingBufferPixelFormat()` , `DrawingBufferPitch()`

99.15 DrawingBufferPitch

Syntax

```
Result = DrawingBufferPitch()
```

Description

Returns the real length of one line of the current drawing buffer.

Parameters

None.

Return value

Returns the length in bytes of one line in the output, including any additional padding behind the pixel data of a line.

Remarks

DrawingBuffer() must be called before using this function.

See Also

DrawingBuffer() , DrawingBufferPixelFormat()

99.16 DrawingBufferPixelFormat

Syntax

```
Result = DrawingBufferPixelFormat()
```

Description

Returns the pixel format of the current output.

Parameters

None.

Return value

The result can be a combination (with bitwise or) of the following flags:

```
#PB_PixelFormat_8Bits      : 1 byte per pixel, palletised
#PB_PixelFormat_15Bits     : 2 bytes per pixel
#PB_PixelFormat_16Bits     : 2 bytes per pixel
#PB_PixelFormat_24Bits_RGB : 3 bytes per pixel (RRGGBB)
#PB_PixelFormat_24Bits_BGR : 3 bytes per pixel (BBGGRR)
#PB_PixelFormat_32Bits_RGB : 4 bytes per pixel (RRGGBB)
#PB_PixelFormat_32Bits_BGR : 4 bytes per pixel (BBGGRR)
#PB_PixelFormat_ReversedY  : The Y-Coordinate of the output is
    reversed in memory (the bottom row is stored first).
#PB_PixelFormat_NoAlpha    : The pixel format is 32-bits but the
    alpha component is not used.
```

Remarks

DrawingBuffer() must be called before using this function.

Example

The following examples show how to handle the result:

```
1  If DrawingBufferPixelFormat() = #PB_PixelFormat_32Bits_RGB |
   #PB_PixelFormat_ReversedY
2      ; 32-bit RGB with reversed Y coordinate
3  EndIf
4
5  If DrawingBufferPixelFormat() = #PB_PixelFormat_32Bits_RGB
6      ; 32-bit RGB without reversed Y coordinate
7  EndIf
8
9  If DrawingBufferPixelFormat() & #PB_PixelFormat_32Bits_RGB
10     ; 32-bit RGB, with or without reversed Y coordinate
11 EndIf
```

See Also

StartDrawing() , DrawingBufferPitch()

99.17 DrawingFont

Syntax

```
DrawingFont(FontID)
```

Description

Sets the font to be used for text rendering on the current output.

Parameters

FontID The font to be used. The FontID can be easily obtained with the FontID() function from the font library.

To restore the default system font, `#PB_Default` can be used as FontID.

Return value

None.

See Also

LoadFont() , FontID()

99.18 DrawingMode

Syntax

```
DrawingMode (Mode)
```

Description

Change the drawing mode for text and graphics output.

Parameters

Mode The behavior for further drawing operations. It can be a combination of the following flags:

`#PB_2DDrawing_Default`

This is the default drawing mode when the drawing starts. Text is displayed with a solid background and graphic shapes are filled. If the current output has an alpha channel, the drawing operations will only modify the color components and leave the alpha channel unchanged.



`#PB_2DDrawing_Transparent`

If this flag is set then the background will be transparent with the DrawText() command.



`#PB_2DDrawing_NativeText`

If this flag is set then the DrawText() and DrawRotatedText() commands will use native text output to have gadget-like text rendering. All special effects like XOR, AlphaBlend, CustomFilters, etc. are not available with this mode.

`#PB_2DDrawing_XOr`

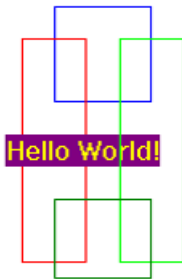
Enables the XOR mode. All graphics will be XOR'ed with the current background. This mode cannot be combined with the below alpha channel modes.

Note: This mode does not work with `PrinterOutput()` .



`#PB_2DDrawing_Outlined`

If this flag is set then shapes will be drawn as outlines only and not filled. This applies to commands such as `Circle` , `Box` , etc.



Note: The following modes only work with `ImageOutput()` , `CanvasOutput()` and `SpriteOutput()` . They are ignored for all other outputs:

`#PB_2DDrawing_AlphaBlend`

The drawing operations will be alpha-blended onto the background. The `RGBA()` command can be used to specify colors with alpha transparency in commands like `FrontColor()` , `Box()` , `DrawText()` etc.



`#PB_2DDrawing_AlphaClip`

The drawing operations will be alpha-blended onto the background like with the `#PB_2DDrawing_AlphaBlend` mode, with the addition that the alpha channel of the drawing output acts as a mask. This means that areas of the output that are transparent before the blending will also remain transparent afterwards. If the drawing output has no alpha channel then this mode acts just like the `#PB_2DDrawing_AlphaBlend` mode.

`#PB_2DDrawing_AlphaChannel`

The drawing operations will only modify the alpha channel of the drawing output. All color information is ignored. For example drawing a circle with a color value of RGBA (0, 0, 0, 0) will "cut" a hole into the drawing output by making the circle area fully transparent. If the drawing output has no alpha channel, as the CanvasGadget, then no drawing will have an effect in this mode.

#PB_2DDrawing_AllChannels

The drawing operations will modify the color channels and the alpha channel of the drawing output. The content of the channels is replaced by the drawing operation without any blending. Drawing in this mode has the same effect as drawing first using the #PB_2DDrawing_Default mode and then drawing the same operation using the #PB_2DDrawing_AlphaChannel mode. If the drawing output has no alpha channel, as the CanvasGadget, then this mode is equivalent to the #PB_2DDrawing_Default mode.



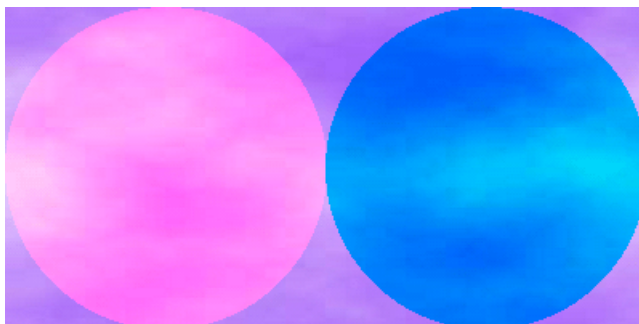
#PB_2DDrawing_Gradient

This mode allows drawing with a gradient instead of a solid color. The gradient shape can be defined with commands such as LinearGradient() , CircularGradient() etc. and the colors used in the gradient can be set with GradientColor() . The color parameters given to the individual drawing commands will be ignored in this mode. This mode can be combined with the above alpha channel modes to have gradients with semitransparent colors.

Alphachannel Gradient

#PB_2DDrawing_CustomFilter

In this mode, the drawing of the pixels can be defined by a custom procedure with the CustomFilterCallback() command. This allows the implementation of custom drawing effects while still using the default functions to do the actual drawing.



Return value

None.

Remarks

To use several modes at once, you have to use the '||' (OR) operator. The following is an example for XOR'ed outlined shapes:

```
1 DrawingMode(#PB_2DDrawing_Outlined | #PB_2DDrawing_XOr)
```

See Also

FrontColor() , BackColor()

99.19 DrawRotatedText

Syntax

```
DrawRotatedText(x.d, y.d, Text$, Angle.f [, Color])
```

Description

Displays the given text on the current output at the given angle. To have gadget-like text output, the drawing mode `#PB_2DDrawing_NativeText` can be used.

Parameters

x.d, y.d The location of the top/left corner of the text in the output. This is also the location around which the string will be rotated.

Text\$ The text to draw.

Angle.f The angle in degrees to rotate counterclockwise starting from the normal text orientation .

Color (optional) The color to be used for the text. If this parameter is not specified, the default color set with FrontColor() will be used. This color can be in RGB or RGBA format. The background of the rotated text is always transparent.

Return value

None.

Example

```
1 If OpenWindow(0, 0, 0, 200, 200, "2DDrawing Example",
2   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3   If CreateImage(0, 200, 200) And StartDrawing(ImageOutput(0))
4     Box(0, 0, 200, 200, RGB(255, 255, 255))
5     For Angle = 0 To 360 Step 45
6       DrawRotatedText(100, 100, "Hello World!", Angle, RGB(0, 0, 0))
7     Next Angle
8     StopDrawing()
9     ImageGadget(0, 0, 0, 200, 200, ImageID(0))
10  EndIf
```

```

10
11 Repeat
12     Event = WaitWindowEvent()
13 Until Event = #PB_Event_CloseWindow
14 EndIf

```



See Also

DrawText() , DrawingFont() , FrontColor() , DrawingMode()

99.20 FillArea

Syntax

```
FillArea(x, y, OutlineColor [, FillColor])
```

Description

Fill an arbitrary area starting from x,y position until the OutlineColor is encountered. This is useful for filling any kind of shape.

Parameters

x, y The location at which the filling should start.

OutlineColor The color which should be considered the border to stop the filling. If this parameter is set to -1 then the area defined by the color found at the (x,y) coordinates will be filled, and filling stops at any color that differs from the starting point.
On 32-bit images, the alpha channel is ignored when determining whether a pixel counts as a border pixel or not.

FillColor (optional) The color to be used for filling the pixels. If this parameter is not specified, the default color set with FrontColor() will be used. This color can be in RGB or RGBA format.

Return value

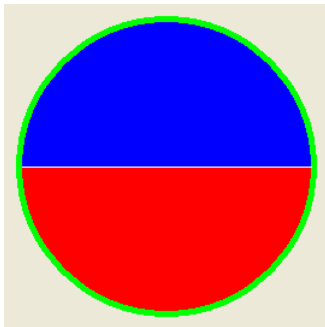
None.

Remarks

This command does not work with PrinterOutput() .

Example

```
1  If OpenWindow(0, 0, 0, 300, 300, "2DDrawing Example",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      If CreateImage(0, 300, 300) And StartDrawing(ImageOutput(0))
3          Box(0, 0, 300, 300, RGB(255, 255, 255))
4
5          Circle(150, 150, 125, $00FF00)
6          Circle(150, 150, 120, $FF0000)
7          LineXY(30, 150, 270, 150, $FFFFFF)
8          FillArea(150, 155, -1, $0000FF) ; Replace -1 by $00FF00, and
    compare the result
9
10         StopDrawing()
11         ImageGadget(0, 0, 0, 300, 300, ImageID(0))
12     EndIf
13
14     Repeat
15         Event = WaitWindowEvent()
16     Until Event = #PB_Event_CloseWindow
17 EndIf
```



See Also

FrontColor()

99.21 GrabDrawingImage

Syntax

```
Result = GrabDrawingImage(#Image, x, y, Width, Height)
```

Description

Create a new image with the content of the given area in the current output.

Parameters

#Image The number of the new image to create. #PB_Any can be used to select a number automatically.

x, y, Width, Height The location and size of the area to copy into the new image. The new image will be created with the specified width and height.

Return value

Returns nonzero on success and zero on failure. If `#PB_Any` was used as the '`#Image`' parameter then the number of the new image is returned.

Remarks

This command does not work with `PrinterOutput()` .

Any parts of the specified area that are outside of the drawing output will be undefined in the created image. Also if the current output is `WindowOutput()` , any part of the window that is currently not visible may be undefined in the resulting image.

See Also

`GrabImage()`

99.22 StartDrawing

Syntax

```
Result = StartDrawing(OutputID)
```

Description

Change the current drawing output to the specified output. After setting this, all drawing functions are rendered to this output.

Parameters

OutputID The output to draw on. It can be obtained with the following functions:

- `WindowOutput()` : Graphics will be rendered directly on the Window
- `ScreenOutput()` : Graphics will be rendered directly on the Screen (for games)
- `SpriteOutput()` : Graphics will be rendered directly on the Sprite (for games)
- `ImageOutput()` : Graphics will be rendered directly on the Image data (see `CreateImage()`)
- `PrinterOutput()` : Graphics will be rendered directly on the Printer
- `CanvasOutput()` : Graphics will be rendered directly on the CanvasGadget()
- `TextureOutput()` : Graphics will be rendered directly on the Texture (for 3D games)

Return value

Returns nonzero if drawing is possible or zero if the operation failed.

Remarks

Once all drawing operations are finished, `StopDrawing()` must be called.

If "Create thread-safe executable" is enabled in the compiler options then every thread has its own current drawing output, which means two threads can do drawing on separate outputs at the same time.

See Also

StopDrawing()

99.23 DrawText

Syntax

```
Result.d = DrawText(x.d, y.d, Text$ [, FrontColor [, BackColor]])
```

Description

Display the given string on the current output at the given x,y position. The current output is set with StartDrawing() . To have gadget-like text output, the drawing mode `#PB_2DDrawing_NativeText` can be used.

Parameters

x.d, y.d The location at which to draw the text.

Text\$ The text to draw.

FrontColor (optional) The color to be used for the text. If this parameter is not specified, the default color set with FrontColor() will be used. This color can be in RGB or RGBA format.

BackColor (optional) The color to be used for the background. If this parameter is not specified, the default color set with BackColor() will be used.

If the current DrawingMode() includes the `#PB_2DDrawing_Transparent` flag, then this parameter is ignored and the background is transparent.

Return value

Returns the new x position of the text cursor (ie the location just after the printed text).

Remarks

If DrawingMode() is set to non-transparent background and the current drawing mode uses the alpha channel then the text is first blended onto the background and then applied to the drawing output.

Example

```
1  If OpenWindow(0, 0, 0, 200, 200, "2DDrawing Example",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      If CreateImage(0, 200, 200) And StartDrawing(ImageOutput(0))
4          DrawingMode(#PB_2DDrawing_Transparent)
5          Box(0, 0, 200, 200, RGB(255, 255, 255))
6          For i = 1 To 30
7              DrawText(Random(200), Random(200), "Hello World!",
8                  RGB(Random(255), Random(255), Random(255)))
9          Next i
10         StopDrawing()
```



```

9      ImageGadget(0, 0, 0, 200, 200, ImageID(0))
10  EndIf
11
12  Repeat
13      Event = WaitWindowEvent()
14      Until Event = #PB_Event_CloseWindow
15  EndIf

```



See Also

DrawRotatedText() , DrawingFont() , FrontColor() , BackColor() , DrawingMode()

99.24 Ellipse

Syntax

```
Ellipse(x, y, RadiusX, RadiusY [, Color])
```

Description

Draw an ellipse in the current drawing output. The filling mode is determined by DrawingMode() . The current output is set with StartDrawing() .

Parameters

x, y The position of the center pixel of the ellipse.

RadiusX, RadiusY The radius of the ellipse in the x and y direction. The center pixel is not included in these values.

Color (optional) The color to be used for the ellipse. If this parameter is not specified, the default color set with FrontColor() will be used. This color can be in RGB or RGBA format.

Return value

None.

Example

```

1  If OpenWindow(0, 0, 0, 200, 200, "2DDrawing Example",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      If CreateImage(0, 200, 200) And StartDrawing(ImageOutput(0))
3          Box(0, 0, 200, 200, RGB(255, 255, 255))
4          For radius=50 To 10 Step -10
5              Ellipse(100, 100, radius*2, radius, RGB(Random(255),
Random(255), Random(255)))
6              Next radius
7              StopDrawing()
8              ImageGadget(0, 0, 0, 200, 200, ImageID(0))
9          EndIf
10
11      Repeat
12          Event = WaitWindowEvent()
13          Until Event = #PB_Event_CloseWindow
14      EndIf

```



See Also

Box() , RoundBox() , Line() , Circle() FrontColor() , RGB() , RGBA() , DrawingMode()

99.25 FrontColor

Syntax

```
ForeColor(Color)
```

Description

Set the default color for graphic functions and text display.

Parameters

Color The new color to be used as the foreground color. This color can be in RGB or RGBA format. Whether or not the alpha channel is used depends on the drawing mode .
A color table with common colors is available [here](#) .

Return value

None.

See Also

BackColor() , RGB() , RGBA() , DrawingMode()

99.26 Line

Syntax

```
Line(x, y, Width, Height [, Color])
```

Description

Draw a line of given dimensions on the current output. The current output is set with StartDrawing() .

Parameters

x, y The origin of the line to draw.

Width, Height The dimension of the line to draw. These values include the starting point so a Height of 1 draws a horizontal line while a Height of 0 draws nothing at all.

Color (optional) The color to be used for the line. If this parameter is not specified, the default color set with FrontColor() will be used. This color can be in RGB or RGBA format.

Return value

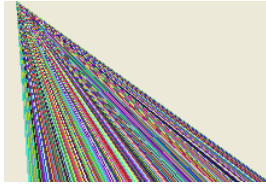
None.

Remarks

To draw a line given the coordinates of the start- and end point, use the LineXY() function.

Example

```
1  If OpenWindow(0, 0, 0, 200, 200, "2DDrawing Example",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      If CreateImage(0, 200, 200) And StartDrawing(ImageOutput(0))
4          Box(0, 0, 200, 200, RGB(255, 255, 255))
5          For Width = 1 To 180 Step 5
6              Line(10, 10, Width, 180, RGB(Random(255), Random(255),
7                  Random(255)))
8              Next Width
9              StopDrawing()
10             ImageGadget(0, 0, 0, 200, 200, ImageID(0))
11         EndIf
12     Repeat
13         Event = WaitWindowEvent()
14     Until Event = #PB_Event_CloseWindow
15 EndIf
```



See Also

LineXY() , Box() , RoundBox() , Ellipse() , Circle() FrontColor() , RGB() , RGBA()

99.27 LineXY

Syntax

```
LineXY(x1, y1, x2, y2 [, Color])
```

Description

Draw a line using the location of the start- and endpoint on the current output. The current output is set with StartDrawing() .

Parameters

x1, y1 The location of the startpoint of the line.

x2, y2 The location of the endpoint of the line.

Color (optional) The color to be used for the line. If this parameter is not specified, the default color set with FrontColor() will be used. This color can be in RGB or RGBA format.

Return value

None.

Remarks

To draw a line given the start coordinates and the dimensions, use the Line() function.

Example

```

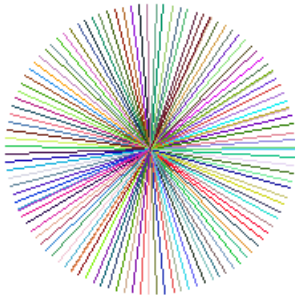
1  If OpenWindow(0, 0, 0, 200, 200, "2DDrawing Example",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      If CreateImage(0, 200, 200) And StartDrawing(ImageOutput(0))
3          Box(0, 0, 200, 200, RGB(255, 255, 255))
4          For Angle = 0 To 360 Step 3
5              LineXY(100, 100, 100+Cos(Radian(Angle))*90,
100+Sin(Radian(Angle))*90, RGB(Random(255), Random(255),
Random(255)))
6              Next Angle

```

```

7      StopDrawing()
8      ImageGadget(0, 0, 0, 200, 200, ImageID(0))
9  EndIf
10
11  Repeat
12      Event = WaitWindowEvent()
13      Until Event = #PB_Event_CloseWindow
14  EndIf

```



See Also

Line() , Box() , RoundBox() , Ellipse() , Circle() FrontColor() , RGB() , RGBA()

99.28 Plot

Syntax

```
Plot(x, y [, Color])
```

Description

Draw a single pixel at the given location in the current output. The current output is set with StartDrawing() .

Parameters

x, y The location of the pixel to set.

For performance reasons there are no bounds checks performed on these coordinates, the specified coordinates must be inside the current drawing area. OutputWidth() and OutputHeight() can be used to verify that. This command is also not affected by any clipping imposed by ClipOutput() .

Color (optional) The color to be used for the pixel. If this parameter is not specified, the default color set with FrontColor() will be used. This color can be in RGB or RGBA format.

Return value

None.

Example

```

1  If OpenWindow(0, 0, 0, 200, 200, "2DDrawing Example",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2  If CreateImage(0, 200, 200) And StartDrawing(ImageOutput(0))
3      For x = 0 To 199
4          For y = 0 To 199
5              Plot(x, y, RGB(Random(255), Random(255), Random(255)))
6          Next y
7      Next x
8      StopDrawing()
9      ImageGadget(0, 0, 0, 200, 200, ImageID(0))
10 EndIf
11
12 Repeat
13     Event = WaitWindowEvent()
14 Until Event = #PB_Event_CloseWindow
15 EndIf

```

See Also

Point() , FrontColor()

99.29 Point

Syntax

Color = Point(x, y)

Description

Return the color of a pixel in the current output.

Parameters

x, y The location of the pixel in the output.

For performance reasons there are no bounds checks performed on these coordinates, the specified coordinates must be inside the current drawing area. OutputWidth() and OutputHeight() can be used to verify that. This command is also not affected by calls to ClipOutput() .

Return value

Returns the color of the specified pixel.

This color will only contain alpha information if the output has a 32-bit color depth and the current DrawingMode() is set to one of the alpha channel modes. Otherwise the alpha component of the color is set to 0.

Remarks

This command does not work with PrinterOutput() .

See Also

Plot() , Red() , Green() , Blue() , Alpha()

99.30 StopDrawing

Syntax

```
StopDrawing()
```

Description

Once all the needed graphics operations (started with StartDrawing()) have been performed, this function must be called to finish the drawing and free all associated resources.

Parameters

None.

Return value

None.

Example

Typically, a normal drawing sequence would look like:

```
1  If StartDrawing(WindowOutput(0))
2      Box(10,10,20,20)
3      Line(30,50,100,100)
4      ....
5      StopDrawing()
6  EndIf
```

See Also

StartDrawing()

99.31 TextHeight

Syntax

```
Height.d = TextHeight(Text$)
```

Description

Return the height of the given string in the current output using the current font.

Parameters

Text\$ The text to measure.

Return value

Returns the height of the given text.

See Also

TextWidth() , DrawingFont()

99.32 TextWidth

Syntax

```
Width.d = TextWidth(Text$)
```

Description

Return the width of the given string in the current output using the current font.

Parameters

Text\$ The text to measure.

Return value

Returns the width of the given text.

See Also

TextHeight() , DrawingFont()

99.33 OutputDepth

Syntax

```
Result = OutputDepth()
```

Description

Returns the color depth of the current drawing output.

Parameters

None.

Return value

Returns the depth in bits per pixel.

See Also

`OutputWidth()` , `OutputHeight()`

99.34 OutputWidth

Syntax

```
Result = OutputWidth()
```

Description

Returns the width of the current drawing output.

Parameters

None.

Return value

Returns the width of the output in pixels.

See Also

`OutputHeight()` , `OutputDepth()`

99.35 OutputHeight

Syntax

```
Result = OutputHeight()
```

Description

Returns the height of the current drawing output.

Parameters

None.

Return value

Returns the height of the output in pixels.

See Also

OutputWidth() , OutputDepth()

99.36 CustomFilterCallback

Syntax

```
CustomFilterCallback(@FilterCallback())
```

Description

Specifies a callback that will be called for every pixel that is part of a drawing operation in `#PB_2DDrawing_CustomFilter` drawing mode .

Parameters

@FilterCallback() The address of a callback function to call. It must have the following form:

```
1  Procedure CustomCallback(x, y, SourceColor.1, TargetColor.1)
2      ;
3      ; Calculate ResultColor from the given input
4      ;
5      ProcedureReturn ResultColor
6  EndProcedure
```

The callback will be called for every pixel that is drawn as a result of a call to drawing functions like `Line()` , `Box()` or `DrawText()` . The `SourceColor` parameter specifies the color given in the drawing operation and the `TargetColor` parameter specifies the color of the target pixel in the drawing area. Both colors are always 32-bit with alpha channel independent of the color depth of the output. The callback has to calculate the color that the target pixel should have after the drawing and return that.

The `x` and `y` coordinate received in the callback are always relative to the upper left corner of the drawing output. The coordinates are not affected by any calls to `SetOrigin()` or `ClipOutput()` .

Return value

None.

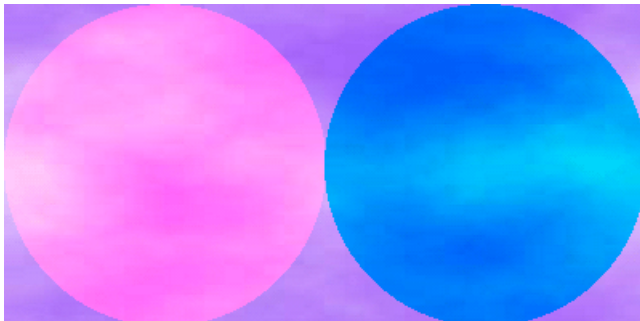
Remarks

This callback will be called many times (for every pixel to draw) so it should be very small and fast to not have a too big impact on the drawing performance.

Note: The `#PB_2DDrawing_CustomFilter` drawing mode only works on `ImageOutput()` and `CanvasOutput()` .

Example

```
1  Procedure FilterCallback(x, y, SourceColor.1, TargetColor.1)
2      ; Take only the Red component from the Source, do not modify the
      others
3      ProcedureReturn RGBA(Red(SourceColor), Green(TargetColor),
      Blue(TargetColor), Alpha(TargetColor))
4  EndProcedure
5
6  UseJPEGImageDecoder()
7
8  If OpenWindow(0, 0, 0, 400, 200, "2DDrawing Example",
      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
9      LoadImage(1, #PB_Compiler_Home +
      "examples/3d/Data/Textures/clouds.jpg")
10
11     If CreateImage(0, 400, 200) And StartDrawing(ImageOutput(0))
12         DrawImage(ImageID(1), 0, 0, 400, 200)
13
14         DrawingMode(#PB_2DDrawing_CustomFilter)
15         CustomFilterCallback(@FilterCallback())
16         Circle(100, 100, 100, $0000FF)
17         Circle(300, 100, 100, $000000)
18
19         StopDrawing()
20         ImageGadget(0, 0, 0, 400, 200, ImageID(0))
21     EndIf
22
23     Repeat
24         Event = WaitWindowEvent()
25         Until Event = #PB_Event_CloseWindow
26     EndIf
```



See Also

`DrawingMode()` , `CustomGradient()`

99.37 GradientColor

Syntax

```
GradientColor(Position.f, Color)
```

Description

Adds the given Color at the given Position to the spectrum of the drawing gradient.

Parameters

Position.f The position for the color in the gradient. It must be a float value between 0.0 and 1.0.

Color The color to be used. This color can be in RGB or RGBA format.

Return value

None.

Remarks

By default, the drawing gradient ranges from the current background color at position 0.0 to the current front color at position 1.0. With this command, additional colors can be added Inbetween, or the colors at 0.0 and 1.0 can be overwritten.

The ResetGradientColors() command can be used to revert back to the default gradient.

The following commands can be used to specify the shape of the drawing gradient:

- LinearGradient()
- CircularGradient()
- EllipticalGradient()
- BoxedGradient()
- ConicalGradient()
- CustomGradient()

Note: This command only has an effect with ImageOutput() and CanvasOutput() . The gradient only affects the drawing commands when #PB_2DDrawing_Gradient is set with the DrawingMode() command.

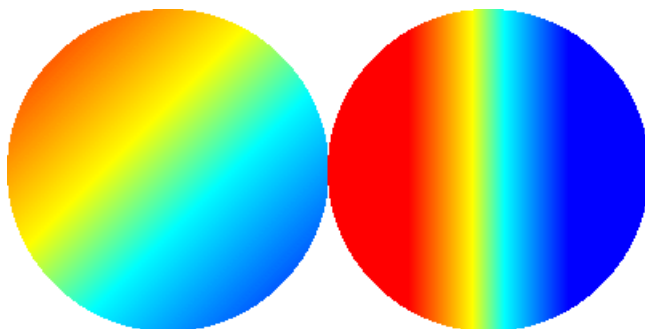
Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "2DDrawing Example",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      If CreateImage(0, 400, 200) And StartDrawing(ImageOutput(0))
4          Box(0, 0, 400, 200, $FFFFFF)
5
6          DrawingMode(#PB_2DDrawing_Gradient)
7          BackColor($0000FF)
8          GradientColor(0.4, $00FFFF)
9          GradientColor(0.6, $FFFF00)
10         FrontColor($FF0000)
11
12         LinearGradient(0, 0, 200, 200)
13         Circle(100, 100, 100)
```

```

13     LinearGradient(350, 100, 250, 100)
14     Circle(300, 100, 100)
15
16     StopDrawing()
17     ImageGadget(0, 0, 0, 400, 200, ImageID(0))
18 EndIf
19
20 Repeat
21     Event = WaitWindowEvent()
22 Until Event = #PB_Event_CloseWindow
23 EndIf

```



See Also

ResetGradientColors() , LinearGradient() , CircularGradient() , EllipticalGradient() , BoxedGradient() , ConicalGradient() , CustomGradient() , DrawingMode()

99.38 ResetGradientColors

Syntax

```
ResetGradientColors()
```

Description

Removes all colors from the drawing gradient and reverts back to a gradient from the current background color to the current front color .

Parameters

None.

Return value

None.

Remarks

The GradientColor() command can be used to add additional colors to the gradient.

Note: This command only has an effect with ImageOutput() and CanvasOutput() . The gradient only affects the drawing commands when #PB_2DDrawing_Gradient is set with the DrawingMode() command.

See Also

GradientColor() , LinearGradient() , CircularGradient() , EllipticalGradient() , BoxedGradient() , ConicalGradient() , CustomGradient() , DrawingMode()

99.39 LinearGradient

Syntax

```
LinearGradient(x1, y1, x2, y2)
```

Description

Sets the drawing gradient to have a linear shape defined by the two points x1,y1 and x2,y2.

Parameters

x1, y1 The position at which to apply the current background color .

x2, y2 The position at which to apply the current front color .

Return value

None.

Remarks

Additional colors can be added to the gradient with the GradientColor() command.

Note: This command only has an effect with ImageOutput() and CanvasOutput() . The gradient only affects the drawing commands when #PB_2DDrawing_Gradient is set with the DrawingMode() command.

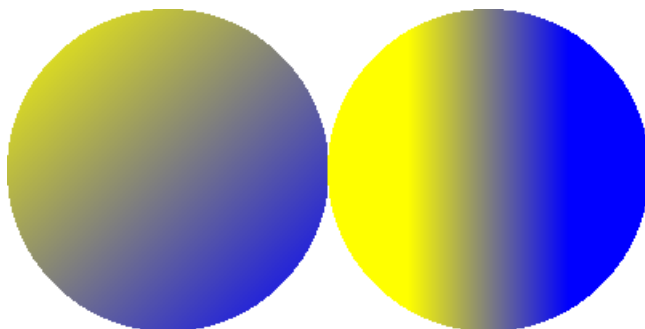
Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "2DDrawing Example",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      If CreateImage(0, 400, 200) And StartDrawing(ImageOutput(0))
4          Box(0, 0, 400, 200, $FFFFFF)
5
6          DrawingMode(#PB_2DDrawing_Gradient)
7          BackColor($00FFFF)
8          FrontColor($FF0000)
9
10         LinearGradient(0, 0, 200, 200)
11         Circle(100, 100, 100)
```

```

11     LinearGradient(350, 100, 250, 100)
12     Circle(300, 100, 100)
13
14     StopDrawing()
15     ImageGadget(0, 0, 0, 400, 200, ImageID(0))
16 EndIf
17
18 Repeat
19     Event = WaitWindowEvent()
20 Until Event = #PB_Event_CloseWindow
21 EndIf

```



See Also

GradientColor() , ResetGradientColors() , CircularGradient() , EllipticalGradient() , BoxedGradient() , ConicalGradient() , CustomGradient() , DrawingMode()

99.40 CircularGradient

Syntax

```
CircularGradient(x, y, Radius)
```

Description

Sets the drawing gradient to have a circular shape.

Parameters

x, y The location at which to apply the current background color .

Radius The radius around (x, y) at which to apply the current front color .

Return value

None.

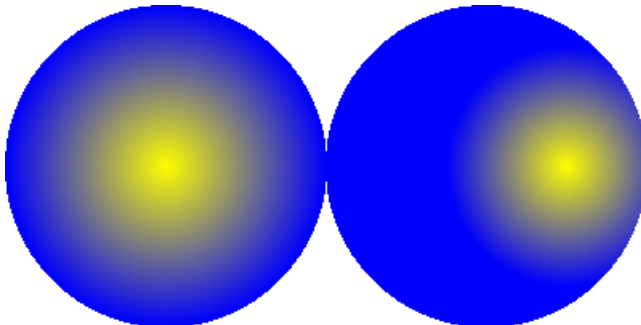
Remarks

Additional colors can be added to the gradient with the GradientColor() command.

Note: This command only has an effect with ImageOutput() and CanvasOutput() . The gradient only affects the drawing commands when #PB_2DDrawing_Gradient is set with the DrawingMode() command.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "2DDrawing Example",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      If CreateImage(0, 400, 200) And StartDrawing(ImageOutput(0))
4          Box(0, 0, 400, 200, $FFFFFF)
5
6          DrawingMode(#PB_2DDrawing_Gradient)
7          BackColor($00FFFF)
8          FrontColor($FF0000)
9
10         CircularGradient(100, 100, 100)
11         Circle(100, 100, 100)
12         CircularGradient(350, 100, 75)
13         Circle(300, 100, 100)
14
15         StopDrawing()
16         ImageGadget(0, 0, 0, 400, 200, ImageID(0))
17     EndIf
18
19     Repeat
20         Event = WaitWindowEvent()
21     Until Event = #PB_Event_CloseWindow
22 EndIf
```



See Also

GradientColor(), ResetGradientColors(), LinearGradient(), EllipticalGradient(), BoxedGradient(), ConicalGradient(), CustomGradient(), DrawingMode()

99.41 EllipticalGradient

Syntax

EllipticalGradient(x, y, RadiusX, RadiusY)

Description

Sets the drawing gradient to have an elliptical shape.

Parameters

x, y The location at which to apply the current background color .

RadiusX, RadiusY The radius around in the x and y direction at which to apply the current front color .

Return value

None.

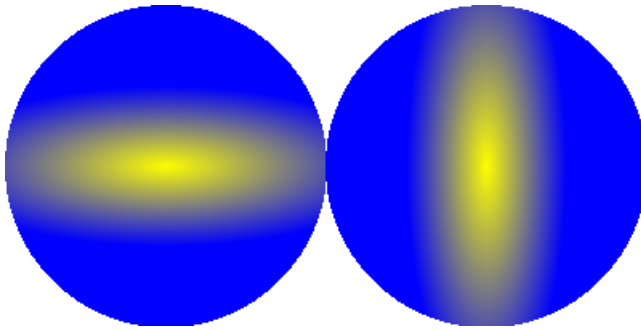
Remarks

Additional colors can be added to the gradient with the GradientColor() command.

Note: This command only has an effect with ImageOutput() and CanvasOutput() . The gradient only affects the drawing commands when #PB_2DDrawing_Gradient is set with the DrawingMode() command.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "2DDrawing Example",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      If CreateImage(0, 400, 200) And StartDrawing(ImageOutput(0))
4          Box(0, 0, 400, 200, $FFFFFF)
5
6          DrawingMode(#PB_2DDrawing_Gradient)
7          BackColor($00FFFF)
8          FrontColor($FF0000)
9
10         EllipticalGradient(100, 100, 150, 50)
11         Circle(100, 100, 100)
12         EllipticalGradient(300, 100, 50, 150)
13         Circle(300, 100, 100)
14
15         StopDrawing()
16         ImageGadget(0, 0, 0, 400, 200, ImageID(0))
17     EndIf
18
19     Repeat
20         Event = WaitWindowEvent()
21     Until Event = #PB_Event_CloseWindow
22 EndIf
```



See Also

GradientColor() , ResetGradientColors() , LinearGradient() , CircularGradient() , BoxedGradient() , ConicalGradient() , CustomGradient() , DrawingMode()

99.42 BoxedGradient

Syntax

```
BoxedGradient(x, y, Width, Height)
```

Description

Sets the drawing gradient to have a box shape.

Parameters

x, y, Width, Height The location of the gradient box. The gradient ranges from the current background color at the center of the box to the current front color at the edges of the box.

Return value

None.

Remarks

Additional colors can be added to the gradient with the GradientColor() command.

Note: This command only has an effect with ImageOutput() and CanvasOutput() . The gradient only affects the drawing commands when #PB_2DDrawing_Gradient is set with the DrawingMode() command.

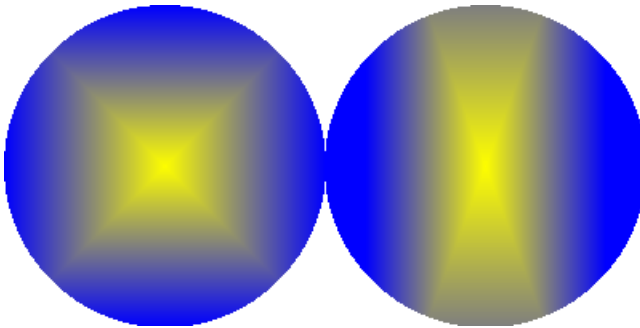
Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "2DDrawing Example",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      If CreateImage(0, 400, 200) And StartDrawing(ImageOutput(0))
4          Box(0, 0, 400, 200, $FFFFFF)
```

```

5      DrawingMode(#PB_2DDrawing_Gradient)
6      BackColor($00FFFF)
7      FrontColor($FF0000)
8
9      BoxedGradient(0, 0, 200, 200)
10     Circle(100, 100, 100)
11     BoxedGradient(225, -100, 150, 400)
12     Circle(300, 100, 100)
13
14     StopDrawing()
15     ImageGadget(0, 0, 0, 400, 200, ImageID(0))
16 EndIf
17
18 Repeat
19     Event = WaitWindowEvent()
20 Until Event = #PB_Event_CloseWindow
21 EndIf

```



See Also

GradientColor(), ResetGradientColors(), LinearGradient(), CircularGradient(), EllipticalGradient(), ConicalGradient(), CustomGradient(), DrawingMode()

99.43 ConicalGradient

Syntax

```
ConicalGradient(x, y, Angle.f)
```

Description

Sets the drawing gradient to have a conical shape.

Parameters

x, y The center position of the conical gradient.

Angle.f The angle (in degrees) at which to start the gradient. The gradient starts with the current background color at the given angle and then changes towards the current front color counterclockwise until the angle is reached again.

Return value

None.

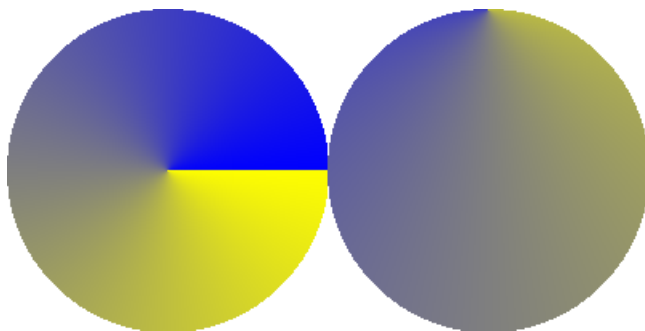
Remarks

Additional colors can be added to the gradient with the GradientColor() command.

Note: This command only has an effect with ImageOutput() and CanvasOutput() . The gradient only affects the drawing commands when #PB_2DDrawing_Gradient is set with the DrawingMode() command.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "2DDrawing Example",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      If CreateImage(0, 400, 200) And StartDrawing(ImageOutput(0))
3          Box(0, 0, 400, 200, $FFFFFF)
4
5          DrawingMode(#PB_2DDrawing_Gradient)
6          BackColor($00FFFF)
7          FrontColor($FF0000)
8
9          ConicalGradient(100, 100, 0.0)
10         Circle(100, 100, 100)
11         ConicalGradient(300, 0, 90.0)
12         Circle(300, 100, 100)
13
14         StopDrawing()
15         ImageGadget(0, 0, 0, 400, 200, ImageID(0))
16     EndIf
17
18     Repeat
19         Event = WaitWindowEvent()
20     Until Event = #PB_Event_CloseWindow
21 EndIf
```



See Also

GradientColor(), ResetGradientColors(), LinearGradient(), CircularGradient(), EllipticalGradient(), BoxedGradient(), CustomGradient(), DrawingMode()

99.44 CustomGradient

Syntax

```
CustomGradient(@GradientCallback())
```

Description

Sets the drawing gradient to have a custom shape, defined by the given callback procedure.

Parameters

@GradientCallback() The address of a callback procedure to define the gradient. The callback must have the following form:

```
1  Procedure.f GradientCallback(x, y)
2      ;
3      ; Return a value between 0.0 and 1.0 to define the gradient for
   the x/y position.
4      ;
5      ProcedureReturn 1.0
6  EndProcedure
```

The callback will be called for every pixel that is part of a drawing operation. The callback has to return a value between 0.0 and 1.0 (not a color value) to define the gradient value at the given position.

The x and y coordinate received in the callback are always relative to the upper left corner of the drawing output. The coordinates are not affected by any calls to `SetOrigin()` or `ClipOutput()`.

Return value

None.

Remarks

By default the value 0.0 represents the current background color and the value 1.0 represents the current front color. Additional colors can be added to the gradient with the `GradientColor()` command.

This callback will be called many times (for every pixel to draw) so it should be very small and fast to not have a too big impact on the drawing performance.

Note: This command only has an effect with `ImageOutput()` and `CanvasOutput()`. The gradient only affects the drawing commands when `#PB_2DDrawing_Gradient` is set with the `DrawingMode()` command.

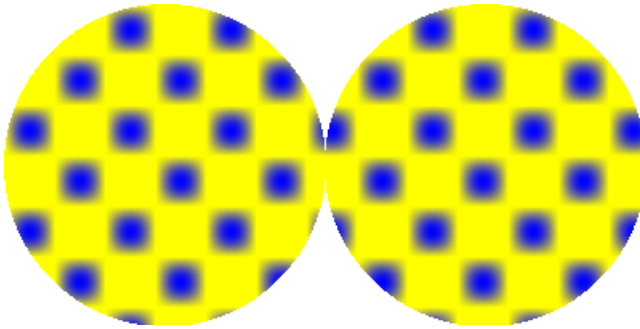
Example

```
1  Procedure.f GradientCallback(x, y)
2      ProcedureReturn Sin(x * 0.1) * Sin(y * 0.1) ; will be between 0 and
   1
3  EndProcedure
4
5  If OpenWindow(0, 0, 0, 400, 200, "2DDrawing Example",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
```

```

6      If CreateImage(0, 400, 200) And StartDrawing(ImageOutput(0))
7          Box(0, 0, 400, 200, $FFFFFF)
8
9          DrawingMode(#PB_2DDrawing_Gradient)
10         BackColor($00FFFF)
11         FrontColor($FF0000)
12
13         CustomGradient(@GradientCallback())
14         Circle(100, 100, 100)
15         Circle(300, 100, 100)
16
17         StopDrawing()
18         ImageGadget(0, 0, 0, 400, 200, ImageID(0))
19     EndIf
20
21     Repeat
22         Event = WaitWindowEvent()
23     Until Event = #PB_Event_CloseWindow
24 EndIf

```



See Also

GradientColor(), ResetGradientColors(), LinearGradient(), CircularGradient(), EllipticalGradient(), BoxedGradient(), ConicalGradient(), DrawingMode()

99.45 SetOrigin

Syntax

```
SetOrigin(x, y)
```

Description

Set an offset at which all drawing in the current output takes place. This defines the location of the coordinates (0, 0) within the output for every following drawing command. By default, the origin is located in the upper left corner of the drawing output.

Parameters

x, y The new position of the drawing origin. This is an absolute location and is not affected by any previous call to this function.

Return value

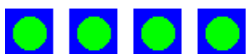
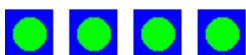
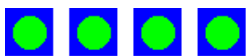
None.

Remarks

This command affects the location of all future drawing commands with the exception of the ClipOutput() command and the SetOrigin() command itself. Also, the coordinates received in a CustomGradient() or CustomFilterCallback() callback are always absolute regardless of any calls to this function.

Example

```
1  If OpenWindow(0, 0, 0, 200, 200, "2DDrawing Example",  
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
3      If CreateImage(0, 200, 200, 24, $FFFFFF) And  
4          StartDrawing(ImageOutput(0))  
5          ; Draw the same figure at different locations by moving the  
6          drawing origin  
7          For x = 0 To 120 Step 40  
8              For y = 0 To 120 Step 60  
9                  SetOrigin(x, y)  
10                 Box(0, 0, 30, 30, $FF0000)  
11                 Circle(15, 15, 10, $00FF00)  
12             Next y  
13         Next x  
14         StopDrawing()  
15         ImageGadget(0, 0, 0, 200, 200, ImageID(0))  
16     EndIf  
17     Repeat  
18         Event = WaitWindowEvent()  
19         Until Event = #PB_Event_CloseWindow  
20     EndIf
```



See Also

GetOriginX() , GetOriginY() , ClipOutput()

99.46 GetOriginX

Syntax

```
Result = GetOriginX()
```

Description

Get the X coordinate of the drawing origin that was set using SetOrigin() .

Parameters

None.

Return value

The X coordinate of the drawing origin.

See Also

GetOriginY() , SetOrigin()

99.47 GetOriginY

Syntax

```
Result = GetOriginY()
```

Description

Get the Y coordinate of the drawing origin that was set using SetOrigin() .

Parameters

None.

Return value

The Y coordinate of the drawing origin.

See Also

GetOriginX() , SetOrigin()

99.48 ClipOutput

Syntax

```
ClipOutput(x, y, Width, Height)
```

Description

Define a bounding box that restricts all drawing to the current drawing output. Any pixels drawn outside of this box will be clipped.

Parameters

x, y, Width, Height The position and size of the clipping box. The (x, y) coordinates are always absolute and not affected by calls to SetOrigin() .

Return value

None.

Remarks

This command only has an effect on drawing outputs created by ImageOutput() or CanvasOutput() . The drawing origin is not modified by a call to this function. To make all drawing relative to the upper left corner of the clipping box, a separate call to SetOrigin() must be made if this is desired. The Plot() and Point() commands perform no bounds checking for performance reasons and therefore are also not affected by this command.

Example

```
1  If OpenWindow(0, 0, 0, 200, 200, "2DDrawing Example",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     If CreateImage(0, 200, 200, 24, $FFFFFF) And
4     StartDrawing(ImageOutput(0))
5
6         ClipOutput(50, 50, 100, 100) ; restrict all drawing to this region
7         Circle( 50,  50, 50, $0000FF)
8         Circle( 50, 150, 50, $00FF00)
9         Circle(150,  50, 50, $FF0000)
10        Circle(150, 150, 50, $00FFFF)
11
12        DrawingMode(#PB_2DDrawing_Outlined)
13        Box(50, 50, 100, 100, $000000)
14
15        StopDrawing()
16        ImageGadget(0, 0, 0, 200, 200, ImageID(0))
17    EndIf
18
19    Repeat
20        Event = WaitWindowEvent()
21        Until Event = #PB_Event_CloseWindow
22    EndIf
```



See Also

`UnclipOutput()` , `SetOrigin()` , `OutputWidth()` , `OutputHeight()`

99.49 UnclipOutput

Syntax

```
UnclipOutput()
```

Description

Remove any clipping imposed by the `ClipOutput()` command. The following drawing commands will be able to draw to the entire drawing output again.

Parameters

None.

Return value

None.

Remarks

This command only has an effect by drawing outputs created by `ImageOutput()` or `CanvasOutput()` .

See Also

`ClipOutput()` , `SetOrigin()` , `OutputWidth()` , `OutputHeight()`

Chapter 100

Array

Overview

Arrays are structures to store indexed elements. Unlike a list or a map , the elements are allocated in a contiguous manner in memory. Therefore, it's not possible to easily insert or remove an element. On the other hand, it provides very fast direct access to a random element.

To work with Arrays, they have to be declared first. This could be done with the keyword `Dim` and can be resized with `ReDim` .

Arrays can be sorted using `SortArray()` or `SortStructuredArray()` , and can also be randomized using `RandomizeArray()` .

Note: These arrays are called dynamic because they can change size. However, there are also so-called static arrays, which can not be resized and are only used in structures. These arrays are written with square brackets. For example: `ArrayStatic[2]`. See [here](#) . The functions of this Array library can not be used with this type of array.

Sort & misc.

`SortArray()`
`SortStructuredArray()`
`RandomizeArray()`

100.1 ArraySize

Syntax

```
Result = ArraySize(Array() [, Dimension])
```

Description

Returns the size of the array, as specified with `Dim` or `ReDim`.

Parameters

Array() The array to get the size from.

Dimension (optional) For multidimensional arrays, this parameter can be specified to get a specific dimension size. The first dimension starts from 1.

Return value

Returns the size of the array dimension. If the array isn't yet declared (for example after FreeArray() or if its allocation has failed), it will return -1.

Remarks

As specified with Dim , the number of elements is equal to the size + 1. For example: Dim a(2) contains 3 elements from a(0) to a(2) for a size of 2.

Does not work with static arrays declared in Structures . Use SizeOf instead.

Example

```
1 Dim MyArray.1(10)
2 Debug ArraySize(MyArray()) ; will print '10'
3
4 Dim MultiArray.1(10, 20, 30)
5 Debug ArraySize(MultiArray(), 2) ; will print '20'
6
7 Dim MultiArray2.1(2,2,2)
8 For n = 0 To ArraySize(MultiArray2(),2)
9     MultiArray2(0,n,0) = n+1
10 Next n
11 Debug MultiArray2(0,0,0) ; will print '1'
12 Debug MultiArray2(0,1,0) ; will print '2'
13 Debug MultiArray2(0,2,0) ; will print '3'
14 Debug ArraySize(MultiArray2(),2) ; will print '2'
```

Example

```
1 Dim Test.q(999999999999999999)
2
3 If ArraySize(Test()) <> -1
4     Test(12345) = 123 ; everything fine
5 Else
6     Debug "Array 'Test()' couldn't be initialized."
7 EndIf
```

Example

```
1 Structure MyStructure
2     ArrayStatic.1[3] ; Static array, only in structures
3     Array ArrayDynamic.1(4) ; Dynamic array
4 EndStructure
5
6 Debug SizeOf(MyStructure\ArrayStatic) ; Display 12
7 Debug SizeOf(MyStructure\ArrayDynamic) ; Display 8
8
9 Ex.MyStructure
10 Debug ArraySize(Ex\ArrayDynamic()); Display 4
```

See Also

ListSize() , MapSize()

100.2 CompareArray

Syntax

```
Result = CompareArray(Array1(), Array2() [, Flags])
```

Description

Compare each elements of the two arrays for equality. Recursively compares also contents of structured arrays with dynamic elements (such as embedded arrays, lists or maps). The two arrays are considered the equal if they have the same type and dimensions and if each pair of elements is equal.

Parameters

Array1(), **Array2()** The arrays to compare.

Flags (optional) Can be a combination of the following values:

```
#PB_String_CaseSensitive : String comparison is case sensitive
(a=a). (default)
#PB_String_NoCase       : String comparison is case
insensitive(a=A).
#PB_Memory_FollowPointers: If a structure element is a pointer
that is not 0, recursively compare the pointer target.
                        The default is to compare only the
pointer value itself. See remarks below for details.
```

Return value

Returns nonzero if both arrays are the same or zero if they differ.

Remarks

The `#PB_Memory_FollowPointers` option is for advanced users and requires special care to avoid crashes. If this option is used then all pointer values must point to valid and initialized memory or have the value 0. It is also not allowed to have loops in the pointed elements (a chain of pointers that refers back to itself).

Example

```
1 Dim A(4)
2 A(0) = 10
3 A(1) = 20
4 A(2) = 30
5 A(3) = 40
6
7 Dim B(4)
```

```

8   B(0) = 10
9   B(1) = 20
10  B(2) = 50 ; Not equal to A(2)
11  B(3) = 40
12
13  Debug CompareArray(A(), B()) ; Not equal

```

See Also

CompareList() , CompareMap()

100.3 CopyArray

Syntax

```
Result = CopyArray(SourceArray(), DestinationArray())
```

Description

Copy every element of 'SourceArray()' into 'DestinationArray()'. After a successful copy, the two arrays are identical. The arrays must have the same number of dimensions.

Parameters

SourceArray() The array to copy from.

DestinationArray() The array to copy to. Every element previously found in this array will be deleted. This array must be of the same type (native or structured) and the same number of dimensions of the source array.

Return value

Returns nonzero if the copy succeeded or zero if it failed.

Example

```

1   Dim Numbers(5)
2   Dim NumbersCopy(10)
3
4   Numbers(0) = 128
5   Numbers(5) = 256
6
7   Debug "Array size before copy: "+Str(ArraySize(NumbersCopy())) ; will
   print 10
8
9   CopyArray(Numbers(), NumbersCopy())
10
11  Debug "Array size after copy: "+Str(ArraySize(NumbersCopy())) ; will
   print 5
12  Debug NumbersCopy(0)
13  Debug NumbersCopy(5)

```

See Also

`CopyList()` , `CopyMap()`

100.4 FreeArray

Syntax

```
FreeArray(Array())
```

Description

Free the specified 'Array()' and release all its associated memory. To access it again Dim has to be called.

Parameters

Array() The array to free.

Return value

None.

See Also

`FreeList()` , `FreeMap()`

Chapter 101

AudioCD

Overview

Audio CDs are an excellent way to play very high quality music during a game or an application, without consuming a lot of system resources. This library provides all the functions necessary to manage audio CD playback within PureBasic programs.

101.1 AudioCDLength

Syntax

```
Result = AudioCDLength()
```

Description

Returns the total length of an entire audio CD.

Parameters

None.

Return value

Returns the length in seconds.

See Also

AudioCDTrackLength() , UseAudioCD()

Supported OS

Windows, MacOS X

101.2 AudioCDName

Syntax

```
Result\$_ = AudioCDName()
```

Description

Returns the system-dependent name associated with the current audio CD drive.

Parameters

None.

Return value

Returns a string containing the name. For example, it would return 'D:\' on a Windows-based system, if the audio CD is associated as drive letter 'D'.

See Also

UseAudioCD()

101.3 AudioCDTrackLength

Syntax

```
Result = AudioCDTrackLength(TrackNumber)
```

Description

Returns the length of the specified track.

Parameters

TrackNumber The number of the requested track. The first track has an index of 1.

Return value

Returns the length (in seconds) of the specified track.

See Also

AudioCDTracks() , UseAudioCD()

101.4 AudioCDStatus

Syntax

```
Result = AudioCDStatus()
```

Description

Returns the actual state of the current audio CD drive.

Parameters

None.

Return value

The return-value may take the following values:

-1: CD drive not ready (it is either empty or opened)

0: CD drive stopped (but the CD is in the drive and has been detected)

>0: Audio track which is currently being played.

See Also

UseAudioCD()

101.5 AudioCDTracks

Syntax

```
Result = AudioCDTracks()
```

Description

Return the total number of tracks on the CD available to be played.

Parameters

None.

Return value

Returns the number of tracks.

See Also

UseAudioCD()

101.6 AudioCDTrackSeconds

Syntax

```
Result = AudioCDTrackSeconds()
```

Description

Return the number of seconds elapsed since the current track began to play.

Parameters

None.

Return value

Returns the number of seconds since the beginning of the track.

See Also

AudioCDTrackLength() , AudioCDTracks() , UseAudioCD()

101.7 EjectAudioCD

Syntax

```
EjectAudioCD(State)
```

Description

Eject (open) or close the current CD drive.

Parameters

State The action to take. If State = 1, the CD drive is opened, else it is closed.

Return value

None.

See Also

UseAudioCD()

Supported OS

Windows, Linux

101.8 InitAudioCD

Syntax

```
Result = InitAudioCD()
```

Description

Try to initialize all the necessary resources in order to handle audio CD playback.

Parameters

None.

Return value

If Result is 0, the initialization has failed or there are no CD drives available, else Result contains the number of CD drives attached to the system.

Remarks

This function must be called before any other audio CD functions.

101.9 PauseAudioCD

Syntax

```
PauseAudioCD()
```

Description

Pause audio CD playback. The playback may be resumed by using the ResumeAudioCD() function.

Parameters

None.

Return value

None.

See Also

UseAudioCD()

101.10 PlayAudioCD

Syntax

```
PlayAudioCD(StartTrack, EndTrack)
```

Description

Start to play the audio CD from 'StartTrack' until the end of 'EndTrack'.

Parameters

StartTrack The first track to play. (The first track on the CD has an index of 1)

EndTrack The last track to play.

Return value

None.

See Also

AudioCDTracks() , UseAudioCD()

101.11 ResumeAudioCD

Syntax

```
ResumeAudioCD()
```

Description

Resume audio CD playback, previously paused by the use of the PauseAudioCD() function.

Parameters

None.

Return value

None.

See Also

UseAudioCD()

101.12 StopAudioCD

Syntax

```
StopAudioCD()
```

Description

Stop audio CD playback.

Parameters

None.

Return value

None.

See Also

UseAudioCD()

101.13 UseAudioCD

Syntax

```
UseAudioCD(AudioCDDrive)
```

Description

Select the Drive to which the following AudioCD commands apply. It is possible to play several CDs at the same time.

Parameters

AudioCDDrive The drive to use. The first drive has index 0. The amount of drives available in the system is returned from the InitAudioCD() function.

Return value

None.

See Also

`InitAudioCD()`

Chapter 102

Billboard

Overview

Billboards are 3D planes (composed of 2 triangles) which always face the camera . Billboards are useful to quickly render a high number of elements , such as rain, snow, trees, vegetation or any particle-based effects. `InitEngine3D()` must be called successfully before using any billboard functions. Billboards are added to billboard groups, which are collections of similarly sized and positioned billboards, so that they may be displayed.

102.1 AddBillboard

Syntax

```
Result = AddBillboard(#BillboardGroup , x, y, z)
```

Description

Adds a billboard to the specified billboard group at the given position, relative to the position of the billboard group. The billboard group must have previously been created using the `CreateBillboardGroup()` function.

Parameters

#BillboardGroup The number of the billboard group you want to attach the new billboard to.

x, y, z The coordinates to create billboard at, relative to the coordinates of the billboard group.

Return value

The new billboard index. This value can be used to manipulate this specific billboard with the commands which needs one.

102.2 BillboardGroupID

Syntax

```
BillboardGroupID = BillboardGroupID(#BillboardGroup)
```

Description

Returns the unique ID which identifies the given '#BillboardGroup' in the operating system. This function is very useful when another library needs a billboard-group reference.

Parameters

#BillboardGroup The billboard group to use.

Return value

The unique ID which identifies the given '#BillboardGroup' in the operating system.

102.3 BillboardGroupMaterial

Syntax

```
BillboardGroupMaterial(#BillboardGroup, MaterialID)
```

Description

Assigns a material to the specified billboard group. This material will be used by all the billboards added to this group. A group can only have one material assigned at any point in time.

Parameters

#BillboardGroup The number of the billboard group to assign a new material to.

MaterialID The ID number of the material to assign to the billboard group (and billboards). A valid 'MaterialID' can be easily obtained by using the MaterialID() function.

Return value

None.

102.4 BillboardGroupX

Syntax

```
Result = BillboardGroupX(#BillboardGroup [, Mode])
```

Description

Determines the absolute position of the billboard group in the world.

Parameters

#BillboardGroup The number of the billboard group to get the position of.

Mode (optional) The mode to get the 'x' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'x' position of the billboard
               group in the world (default).
#PB_Relative: get the 'x' position of the billboard group
               relative to its parent.
```

Return value

The absolute 'x' position of the billboard group.

102.5 BillboardGroupY

Syntax

```
Result = BillboardGroupY(#BillboardGroup [, Mode])
```

Description

Determines the absolute position of the billboard group in the world.

Parameters

#BillboardGroup The number of the billboard group to get the position of.

Mode (optional) The mode to get the 'y' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'y' position of the billboard
               group in the world (default).
#PB_Relative: get the 'y' position of the billboard group
               relative to its parent.
```

Return value

The absolute 'y' position of the billboard group.

102.6 BillboardGroupZ

Syntax

```
Result = BillboardGroupZ(#BillboardGroup [, Mode])
```

Description

Determines the absolute position of the billboard group in the world.

Parameters

#BillboardGroup The number of the billboard group to get the position of.

Mode (optional) The mode to get the 'z' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'z' position of the billboard
               group in the world (default).
#PB_Relative: get the 'z' position of the billboard group
               relative to its parent.
```

Return value

The absolute 'z' position of the billboard group.

102.7 BillboardHeight

Syntax

```
Result = BillboardHeight(#Billboard, #BillboardGroup)
```

Description

Determines the height of a billboard which has been added to a billboard group. The height is measured in the units used by the world.

Parameters

#Billboard The number of the billboard to get the height of.

#BillboardGroup The number of the billboard group which contains the billboard.

Return value

The height of the billboard, in world units.

102.8 BillboardLocate

Syntax

```
BillboardLocate(#Billboard, #BillboardGroup, x, y, z)
```

Description

Moves a billboard to a new location within the billboard group which is it added to. The position to move to is relative to the position of the billboard group. Use the `MoveBillboard()` function to move a billboard relative to its own coordinates.

Parameters

#Billboard The number of the billboard to move.

#BillboardGroup The number of the billboard group which contains the billboard.

x, y, z The new position for the billboard.

Return value

None.

102.9 BillboardWidth

Syntax

```
Result = BillboardWidth(#Billboard, #BillboardGroup)
```

Description

Determines the width of a billboard which has been added to a billboard group. The width is measured in the units used by the world.

Parameters

#Billboard The billboard to use.

#BillboardGroup The number of the billboard group which contains the billboard.

Return value

The width of the billboard, in world units.

102.10 BillboardX

Syntax

```
Result = BillboardX(#Billboard, #BillboardGroup)
```

Description

Returns the position of the billboard, relative to the position of the group which it is in.

Parameters

#Billboard The billboard to use.

#BillboardGroup The number of the billboard group which contains the billboard.

Return value

The X position of the billboard.

102.11 BillboardY

Syntax

```
Result = BillboardY(#Billboard, #BillboardGroup)
```

Description

Returns the position of the billboard, relative to the position of the group which it is in.

Parameters

#Billboard The billboard to use.

#BillboardGroup The number of the billboard group which contains the billboard.

Return value

The Y position of the billboard.

102.12 BillboardZ

Syntax

```
Result = BillboardZ(#Billboard, #BillboardGroup)
```

Description

Returns the position of the billboard, relative to the position of the group which it is in.

Parameters

#Billboard The billboard to use.

#BillboardGroup The number of the billboard group which contains the billboard.

Return value

The Z position of the billboard.

102.13 ClearBillboards

Syntax

```
ClearBillboards(#BillboardGroup)
```

Description

Removes and destroys all billboards in the specified billboard group.

Parameters

#BillboardGroup The billboard group to clear.

Return value

None.

102.14 CountBillboards

Syntax

```
Result = CountBillboards(#BillboardGroup)
```

Description

Counts the number of billboards contained in a billboard group.

Parameters

#BillboardGroup The billboard group to use.

Return value

The number of billboards which have been added to the billboard group.

102.15 CreateBillboardGroup

Syntax

```
Result = CreateBillboardGroup(#BillboardGroup, MaterialID,
    DefaultBillboardWidth, DefaultBillboardHeight [, x, y, z [,
    VisibilityMask [, Type]])
```

Description

Creates a new empty billboard group.

Parameters

#BillboardGroup The number to identify the new billboard group. `#PB_Any` can be used to auto-generate this number.

MaterialID The material to use for all billboards added to this group, using the `AddBillboard()` function. To get a valid material id use `MaterialID()` .

DefaultBillboardWidth The default width (in world units) of billboards added to this group. Although the size of each billboard can be set separately with the `ResizeBillboard()` function, it is important to keep in mind that it will have a negative impact on performance if all billboards are not the same size.

DefaultBillboardHeight The default height (in world units) of billboards added to this group. Although the size of each billboard can be set separately with the `ResizeBillboard()` function, it is important to keep in mind that it will have a negative impact on performance if all billboards are not the same size.

x, y, z (optional) The absolute position of the new billboard group in the world.

VisibilityMask (optional) A mask to select on which camera the billboard group should be displayed. If this mask match the mask specified in `CreateCamera()` , the billboard group will be displayed on the camera. See `CreateEntity()` to build correct masks. If this parameter is omitted or set to `#PB_All`, then the billboard group will be visible on all cameras.

Type (optional) The billboard type. It can be one of the following values:

```
#PB_Billboard_Point: standard point billboard, always faces the
    camera completely and is always upright (default).
#PB_Billboard_Oriented: billboards are oriented around a shared
    direction vector (used as Y axis) and only rotate around this to
    face the camera.
#PB_Billboard_SelfOriented: billboards are oriented around their
    own direction vector (their own Y axis) and only rotate around
    this to face the camera.
#PB_Billboard_Perpendicular: billboards are perpendicular to a
    shared direction vector (used as Z axis, the facing direction)
    and X, Y axis are determined by a shared up-vector.
#PB_Billboard_SelfPerpendicular: billboards are perpendicular to
    their own direction vector (their own Z axis, the facing
    direction) and X, Y axis are determined by a shared up-vector.
```

Return value

Returns zero if the billboard group can't be created. If `#PB_Any` is used as '`#BillboardGroup`' parameter, the new billboard group number is returned.

See Also

FreeBillboardGroup() , AddBillboard() , BillboardGroupCommonDirection() ,
BillboardGroupCommonDirection()

102.16 BillboardGroupCommonDirection

Syntax

```
BillboardGroupCommonDirection(#BillboardGroup, x, y, z)
```

Description

Changes the billboard common direction.

Parameters

#BillboardGroup The billboard group to use.

x, y, z The common direction vector (usually values between -1.0 and 1.0, if not it will be automatically normalized).

Return value

None.

See Also

CreateBillboardGroup()

102.17 BillboardGroupCommonUpVector

Syntax

```
BillboardGroupCommonUpVector(#BillboardGroup, x, y, z)
```

Description

Changes the billboard common up vector.

Parameters

#BillboardGroup The billboard group to use.

x, y, z The common up vector (usually values between -1.0 and 1.0, if not it will be automatically normalized).

Return value

None.

See Also

CreateBillboardGroup()

102.18 FreeBillboardGroup

Syntax

```
FreeBillboardGroup(#BillboardGroup)
```

Description

Frees the specified billboard group and any billboards contained within it. All its associated memory is released and this object cannot be used anymore.

Parameters

#BillboardGroup The billboard group to free. If **#PB_All** is specified, all the remaining billboard groups are freed.

Return value

None.

Remarks

All remaining billboard groups are automatically freed when the program ends.

102.19 HideBillboardGroup

Syntax

```
HideBillboardGroup(#BillboardGroup, State)
```

Description

Changes the visibility of (hides or shows) a billboard group and any billboards it contains.

Parameters

#BillboardGroup The billboard group to hide or show.

State It can take the following values:

```
#True : the billboard group is hidden
#False: the billboard group is shown
```

Return value

None.

102.20 IsBillboardGroup

Syntax

```
Result = IsBillboardGroup(#BillboardGroup)
```

Description

Tests if the given billboard group number is a valid and correctly initialized billboard group.

Parameters

#BillboardGroup The billboard group to use.

Return value

Nonzero if the billboard group is valid, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a billboard group is ready to use.

102.21 MoveBillboard

Syntax

```
MoveBillboard(#Billboard, #BillboardGroup, x, y, z)
```

Description

Moves a billboard which is contained in a billboard group by the specified x, y and z values. This is a relative move based on the current location of the billboard. To perform an absolute move (actually, relative to the coordinates of the billboard group) use BillboardLocate() .

Parameters

#Billboard The billboard to move.

#BillboardGroup The billboard group which contains the billboard.

x, y, z The offset to add to the current position of the billboard.

Return value

None.

102.22 MoveBillboardGroup

Syntax

```
MoveBillboardGroup(#BillboardGroup, x, y, z [, Mode])
```

Description

Moves a billboard group by the specified x, y and z values. This is by default a relative move, based on the current location of the billboard group.

Parameters

#BillboardGroup The billboard group to move.

x, y, z The new position of the billboard group.

Mode (optional) It can be one of the following values:

```
#PB_Relative: relative move, from the current billboard group  
position (default).  
#PB_Absolute: absolute move to the specified position.
```

combined with one of the following values:

```
#PB_Local : local move.  
#PB_Parent: move relative to the parent position.  
#PB_World : move relative to the world.
```

Return value

None.

102.23 RemoveBillboard

Syntax

```
RemoveBillboard(#Billboard, #BillboardGroup)
```

Description

Removes a billboard from the specified billboard group.

Parameters

#Billboard The billboard to remove.

#BillboardGroup The billboard group which currently contains the billboard.

Return value

None.

102.24 ResizeBillboard

Syntax

```
ResizeBillboard(#Billboard, #BillboardGroup, Width, Height)
```

Description

Resizes a billboard, which is currently contained in a billboard group, to a new width and height, specified in world units. Note that although you can resize billboards independently with this function there will be some performance lost if the billboards within a billboard group are all different sizes.

Parameters

#Billboard The billboard to resize.

#BillboardGroup The billboard group which currently contains the billboard.

Width, Height The new size of the billboard, in world units.

Return value

None.

102.25 RotateBillboardGroup

Syntax

```
RotateBillboardGroup(#BillboardGroup, x, y, z [, Mode])
```

Description

Rotates the #BillboardGroup according to the specified x, y, z angle values.

Parameters

#BillboardGroup The billboard group to rotate.

x, y, z The new rotation to apply to the billboard group. The value are in degree, ranging from 0 to 360.

Mode (optional) It can be one of the following value:

```
#PB_Absolute: absolute rotation (default).  
#PB_Relative: relative rotation based on the previous billboard  
rotation.
```

Return value

None.

Chapter 103

CGI

Overview

CGI stands for "Common Gateway Interface" and allows creation of server side application. This library provides all commands needed to receive requests or files, answer it and do various other operations. Both CGI and FastCGI mode are supported.

The [Wikipedia article on CGI](#) provides a good starting point for people new to CGI.

103.1 CGICookieName

Syntax

```
Result\$$ = CGICookieName(Index)
```

Description

Returns the name of the specified cookie.

Parameters

Index The index of the cookie to get the name. The first index value starts from 0. To get the number of available cookie, use CountCGICookies() .

Return value

Returns the name of the specified cookie.

Example

```
1  If Not InitCGI() Or Not ReadCGI()  
2      End  
3  EndIf  
4  
5  WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html") ; Write the  
    headers to inform the browser of the content format
```

```

6 WriteCGIHeader(#PB_CGI_HeaderSetCookie , "mycookie=hello",
  #PB_CGI_LastHeader) ; Write one cookie named 'mycookie'
7
8 WriteCGIString("<html><title>PureBasic - cookies</title><body>" +
9   "NbCookies: " + CountCGICookies() + "<br><br>")
10
11 ; List the all cookies and display their name and value
12 ;
13 For k = 0 To CountCGICookies()-1
14   WriteCGIString(CGICookieName(k)+": " +
15     CGICookieValue(CGICookieName(k)) + "<br>")
16 Next
17 WriteCGIString("</body></html>")

```

See Also

CountCGICookies() , CGICookieValue()

103.2 CGICookieValue

Syntax

Result\\$_ = CGICookieValue(Name\$)

Description

Returns the value of the specified cookie.

Parameters

Name\$ The name of the cookie to get the value. The cookie name is case-sensitive. CGICookieName() can be used to get the name of a specified cookie. To get the number of available cookie, use CountCGICookies() .

Return value

Returns the value of the specified cookie.

Example

```

1 If Not InitCGI() Or Not ReadCGI()
2   End
3 EndIf
4
5 WriteCGIHeader(#PB_CGI_HeaderContentType , "text/html") ; Write the
  headers to inform the browser of the content format
6 WriteCGIHeader(#PB_CGI_HeaderSetCookie , "mycookie=hello",
  #PB_CGI_LastHeader) ; Write one cookie named 'mycookie'
7

```

```

8 WriteCGIString("<html><title>PureBasic - cookies</title><body>" +
9 "NbCookies: " + CountCGICookies() + "<br><br>")
10
11 ; List the all cookies and display their name and value
12 ;
13 For k = 0 To CountCGICookies()-1
14 WriteCGIString(CGICookieName(k)+": " +
15 CGICookieValue(CGICookieName(k)) + "<br>")
16 Next
17 WriteCGIString("</body></html>")

```

See Also

CountCGICookies() , CGICookieName()

103.3 CountCGICookies

Syntax

```
Result = CountCGICookies()
```

Description

Returns the number of available cookies. The cookies are small persistent files stored in the web browser to allow to remember a context and ease future navigation when loading the same page later on. Please note that European legislation now impose to inform users that cookies are not being used to gather information unnecessarily.

Parameters

None.

Return value

Returns the number of available cookies. The name and value of cookies can be get with CGICookieName() and CGICookieValue() .

Example

```

1 If Not InitCGI() Or Not ReadCGI()
2 End
3 EndIf
4
5 WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html") ; Write the
  headers to inform the browser of the content format
6 WriteCGIHeader(#PB_CGI_HeaderSetCookie, "mycookie=hello",
  #PB_CGI_LastHeader) ; Write one cookie named 'mycookie'
7

```



```

8 WriteCGIString("<html><title>PureBasic - cookies</title><body>" +
9   "NbCookies: " + CountCGICookies() + "<br><br>")
10
11 WriteCGIString("</body></html>")

```

See Also

CGICookieName() , CGICookieValue()

103.4 CountCGIParameters

Syntax

```
Result = CountCGIParameters()
```

Description

Returns the number of available parameters from GET or POST requests.

Parameters

None.

Return value

Returns the number of available parameters from GET or POST requests. Parameter information can be get with CGIParameterName() , CGIParameterValue() and CGIParameterType() .

Example

```

1  If Not InitCGI() Or Not ReadCGI()
2      End
3  EndIf
4
5  WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",
6    #PB_CGI_LastHeader) ; Write the headers to inform the browser of the
   content format
7
8  WriteCGIString("<html><title>PureBasic - parameters</title><body>" +
9    "NbParameters: " + CountCGIParameters() + "<br><br>")
10
   WriteCGIString("</body></html>")

```

See Also

CGIParameterName() , CGIParameterValue() , CGIParameterType() , CGIParameterData()

103.5 CGIParameterName

Syntax

`Result = CGIParameterName(Index)`

Description

Returns the name of the specified parameter.

Parameters

Index The index of the parameter to get the name. The first index value starts from 0. To get the number of available parameters, use `CountCGIParameters()` .

Return value

Returns the name of the specified parameter.

Example

```
1  If Not InitCGI() Or Not ReadCGI()
2      End
3  EndIf
4
5  WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",
6      #PB_CGI_LastHeader) ; Write the headers to inform the browser of the
7      content format
8
9  WriteCGIString("<html><title>PureBasic - parameters</title><body>" +
10      "NbParameters: " + CountCGIParameters() + "<br><br>")
11
12  ; List the all parameters and display their name
13  ;
14  For k = 0 To CountCGIParameters()-1
15      WriteCGIString(CGIParameterName(k)+"<br>")
16  Next
17
18  WriteCGIString("</body></html>")
```

See Also

`CountCGIParameters()` , `CGIParameterValue()` , `CGIParameterType()`

103.6 CGIParameterValue

Syntax

`Result = CGIParameterValue(Name$ [, Index])`

Description

Returns the value of the specified parameter.

Parameters

Name\$ The name of the parameter to get the value. The parameter name is case-sensitive. `CGIParaterName()` can be used to get the name of a specified parameter. To get the number of available parameters, use `CountCGIParameters()` . This parameter will be ignored if an 'Index' is specified.

Index (optional) The index of the parameter to get the value. The first index value starts from 0. If specified, the 'Name\$' parameter value is ignored (excepts if sets to `#PB_Ignore`).

Return value

Returns the value of the specified parameter.

Example

```
1  If Not InitCGI() Or Not ReadCGI()
2      End
3  EndIf
4
5  WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",
6      #PB_CGI_LastHeader) ; Write the headers to inform the browser of the
7      content format
8
9  WriteCGIString("<html><title>PureBasic - parameters</title><body>" +
10      "NbParameters: " + CountCGIParameters() + "<br><br>")
11
12 ; List the all parameters and display their name and value
13 ;
14 For k = 0 To CountCGIParameters()-1
15     WriteCGIString(CGIParaterName(k) + ": " + CGIParaterValue("",
16         k) + "<br>")
17 Next
18
19 WriteCGIString("</body></html>")
```

See Also

`CountCGIParameters()` , `CGIParaterName()` , `CGIParaterType()`

103.7 CGIParaterType

Syntax

`Result = CGIParaterType(Name$ [, Index])`

Description

Returns the type of the specified parameter.

Parameters

Name\$ The name of the parameter to get the type. The parameter name is case-sensitive.
CGIParameName() can be used to get the name of a specified parameter. To get the number of available parameters, use CountCGIParameters() .

Index (optional) The index of the parameter to get the type. The first index value starts from 0. If specified, the 'Name\$' parameter value is ignored (excepts if sets to #PB_Ignore).

Return value

Returns the type of the specified parameter. It can be one of the following value:

#PB_CGI_Text: the parameter is a string
#PB_CGI_File: the parameter is a binary file. CGIParameValue()
will returns the original filename
and CGIParameData()
can be used to retrieve the binary data.

Example

```
1  If Not InitCGI() Or Not ReadCGI()
2      End
3  EndIf
4
5  WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",
6      #PB_CGI_LastHeader) ; Write the headers to inform the browser of the
7      content format
8
9  WriteCGIString("<html><title>PureBasic - parameters</title><body>" +
10      "NbParameters: " + CountCGIParameters() + "<br><br>")
11
12 ; List the all parameters and display their name
13 ;
14 For k = 0 To CountCGIParameters()-1
15     If CGIParameType("", k) = #PB_CGI_File
16         WriteCGIString("[File] "+CGIParameName(k)+" (filename:
17         "+CGIParameValue("", k)+"<br>")
18     Else
19         WriteCGIString("[String] "+CGIParameName(k)+" (value:
20         "+CGIParameValue("", k)+"<br>")
21     EndIf
22 Next
23
24 WriteCGIString("</body></html>")
```

See Also

CGIParameName() , CGIParameValue() , CGIParameData() , CGIParameDataSize()

103.8 CGIParameterData

Syntax

```
*Result = CGIParameterData(Name$ [, Index])
```

Description

Returns the memory buffer address of the specified parameter data.

Parameters

Name\$ The name of the parameter to get the type. The parameter name is case-sensitive.

CGIParameterName() can be used to get the name of a specified parameter. To get the number of available parameters, use CountCGIParameters() .

Index (optional) The index of the parameter to get the type. The first index value starts from 0. If specified, the 'Name\$' parameter value is ignored (excepts if sets to #PB_Ignore).

Return value

Returns the memory buffer address of the specified parameter data. The parameter type has to be #PB_CGI_File. CGIParameterDataSize() can be used to get the size of the memory buffer.

Example

```
1  If Not InitCGI() Or Not ReadCGI()
2      End
3  EndIf
4
5  WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",
6      #PB_CGI_LastHeader) ; Write the headers to inform the browser of the
7      content format
8
9  WriteCGIString("<html><title>PureBasic - parameters</title><body>" +
10      "NbParameters: " + CountCGIParameters() + "<br><br>")
11
12  ; List the all parameters and display their name
13  ;
14  For k = 0 To CountCGIParameters()-1
15      If CGIParameterType("", k) = #PB_CGI_File
16          WriteCGIString("[File] "+CGIParameterName(k)+" (filename:
17              '"+CGIParameterValue("", k) +
18              " - size: " +
19              CGIParameterDataSize("", k) +
20              " bytes - *buffer: "
21              + CGIParameterData("", k) + ")<br>")
22          EndIf
23      Next
24
25  WriteCGIString("</body></html>")
```

See Also

CGIPParameterName() , CGIPParameterValue() , CGIPParameterType() , CGIPParameterDataSize()

103.9 CGIPParameterDataSize

Syntax

```
Result = CGIPParameterDataSize(Name$ [, Index])
```

Description

Returns the size of the specified parameter data size.

Parameters

Name\$ The name of the parameter to get the type. The parameter name is case-sensitive.

CGIPParameterName() can be used to get the name of a specified parameter. To get the number of available parameters, use CountCGIPParameters() .

Index (optional) The index of the parameter to get the type. The first index value starts from 0. If specified, the 'Name\$' parameter value is ignored (excepts if sets to `#PB_Ignore`).

Return value

Returns the data size (in bytes) of the specified parameter. CGIPParameterData() can be used to get the memory buffer address of the parameter. The parameter type has to be `#PB_CGI_File`.

Example

```
1  If Not InitCGI() Or Not ReadCGI()
2      End
3  EndIf
4
5  WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",
6      #PB_CGI_LastHeader) ; Write the headers to inform the browser of the
7      content format
8
9  WriteCGIString("<html><title>PureBasic - parameters</title><body>" +
10      "NbParameters: " + CountCGIPParameters() + "<br><br>")
11
12 ; List the all parameters and display their name
13 ;
14 For k = 0 To CountCGIPParameters()-1
15     If CGIPParameterType("", k) = #PB_CGI_File
16         WriteCGIString("[File] "+CGIPParameterName(k)+" (filename:
17             '"+CGIPParameterValue("", k) +
18                 "' - size: " +
19                 CGIPParameterDataSize("", k) +
20                 " bytes - *buffer: "
21             + CGIPParameterData("", k) + ")<br>")
22     EndIf
```

```

18   Next
19
20   WriteCGIString("</body></html>")

```

See Also

CGIParameName() , CGIParameValue() , CGIParameType() , CGIParameData()

103.10 CGIBuffer

Syntax

```
*Result = CGIBuffer()
```

Description

For advanced users. Returns the memory buffer address of the raw CGI input (only useful for POST request type). It can be useful to do extra parsing not supported by this library while still using other commands. The size of the buffer is the value returned by ReadCGI() .

Parameters

None.

Return value

Returns the memory buffer address of the raw CGI input, or zero if an error occurred.

Example

```

1   If Not InitCGI()
2       End
3   EndIf
4
5   BufferSize = ReadCGI()
6
7   WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",
8       #PB_CGI_LastHeader) ; Write the headers to inform the browser of the
9       content format
10
11  WriteCGIString("<html><title>PureBasic - raw buffer</title><body>")
12  If CGIBuffer()
13      WriteCGIString("Raw buffer content: <br><pre>" + PeekS(CGIBuffer(),
14          BufferSize, #PB_Ascii) + "</pre>")
15  EndIf
16
17  WriteCGIString("</body></html>")

```

See Also

ReadCGI()

103.11 CGIVariable

Syntax

```
Result\$ = CGIVariable(Name$)
```

Description

Gets the specified CGI environment variable content. When the CGI is loaded, many information are sent from the web server to the CGI application through environment variables.

Parameters

Name\$ The name of the variable to get. It can be a custom value or one of following predefined constants:

```
#PB_CGI_AuthType      : the authentication method used by the  
web browser if any authentication  
method was used. This is not set  
unless the script is protected.  
  
#PB_CGI_ContentLength : used for scripts that are receiving  
form data using the POST method.  
This variable tells the byte length  
of the CGI input data stream. This is  
required to read the data from the  
standard input with the POST method.  
  
#PB_CGI_HeaderContentType : tells the media type of data being  
received from the user.  
This is used for scripts called using  
the POST method.  
#PB_CGI_DocumentRoot   : the root path to the home HTML page  
for the server.  
#PB_CGI_GatewayInterface : the version of the common gateway  
Interface (CGI) specification  
being used to exchange the data  
between the client and server.  
This is usually CGI/1.1 for the  
current revision level of 1.1.  
#PB_CGI_PathInfo        : extra path information added to the  
end of the URL that  
accessed the server side script  
program.  
#PB_CGI_PathTranslated  : a translated version of the PATH_INFO  
variable translated  
by the webserver from virtual to  
physical path information.  
#PB_CGI_QueryString     : this string contains any information  
at the end of the server side script
```


path that followed a question mark.
Used to Return data if the GET method
was used by a form. There are length
restrictions to the QUERY_STRING.

#PB_CGI_RemoteAddr : the IP address of the client computer.
#PB_CGI_RemoteHost : the fully qualified domain name of
the client machine making the
HTTP request. It may not be possible
to determine this name since
many client computers names are not
recorded in the DNS system.

#PB_CGI_RemoteIdent : the ability to use this variable is
limited to servers that support RFC 931.
This variable may contain the client
machine's username, but it is intended
to be used for logging purposes only,
when it is available.

#PB_CGI_RemotePort : the clients requesting port.
#PB_CGI_RemoteUser : if the CGI script was protected and
the user had to be logged in to
get access to the script, this value
will contain the user's log in name.

#PB_CGI_RequestURI : the path to the requested file by the
client.

#PB_CGI_RequestMethod : describes the request method used by
the browser
which is usually GET, POST, Or HEAD.

#PB_CGI_ScriptName : the virtual path of the CGI script
being executed.

#PB_CGI_ScriptFilename : the local filename of the script
being executed.

#PB_CGI_ServerAdmin : the e-mail address of the server
administrator.

#PB_CGI_ServerName : the server hostname, IP address Or
DNS alias name shown as a
self referencing URL. This does not
include the protocol identifier
such as "HTTP:", the machine name, or
port number.

#PB_CGI_ServerPort : the port number the HTTP requests and
responses are being sent on.

#PB_CGI_ServerProtocol : this value is usually HTTP which
describes the protocol
being used between the client and
server computers.

#PB_CGI_ServerSignature : server information specifying the
name and version
of the web server and the port being
serviced.

#PB_CGI_ServerSoftware : the name and version of the web
server.

#PB_CGI_HttpAccept : the media types of data that the
client browser can accept.
These data types are separated by
commas.

```

#PB_CGI_HttpAcceptEncoding: the encoding type the client browser
                           accepts.
#PB_CGI_HttpAcceptLanguage: the language the client browser
                           accepts.
#PB_CGI_HttpCookie          : used as an environment variable that
                           contains cookies
                           associated with the server domain
                           from the browser.
#PB_CGI_HttpForwarded       : the forwarded page URL.
#PB_CGI_HttpHost            : hostname from where the HTTP requests
                           comes from.
#PB_CGI_HttpPragma          : HTTP pragmas
#PB_CGI_HttpReferer         : the page address where the HTTP
                           request originated.
#PB_CGI_HttpUserAgent       : the name of the client web browser
                           being used to make the request.

```

Return value

Returns the value of the specified CGI environment variable.

Example

```

1  If Not InitCGI() Or Not ReadCGI()
2      End
3  EndIf
4
5  WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",
6      #PB_CGI_LastHeader) ; Write the headers to inform the browser of the
7      content format
8
9  WriteCGIString("<html><title>PureBasic - variables</title><body>")
10
11 Procedure WriteCGIConstant(Constant$)
12     WriteCGIString(Constant$ + ": " + CGIVariable(Constant$)+"<br>")
13 EndProcedure
14
15 WriteCGIConstant(#PB_CGI_AuthType)
16 WriteCGIConstant(#PB_CGI_ContentLength)
17 WriteCGIConstant(#PB_CGI_HeaderContentType)
18 WriteCGIConstant(#PB_CGI_DocumentRoot)
19 WriteCGIConstant(#PB_CGI_GatewayInterface)
20 WriteCGIConstant(#PB_CGI_PathInfo)
21 WriteCGIConstant(#PB_CGI_PathTranslated)
22 WriteCGIConstant(#PB_CGI_QueryString)
23 WriteCGIConstant(#PB_CGI_RemoteAddr)
24 WriteCGIConstant(#PB_CGI_RemoteHost)
25 WriteCGIConstant(#PB_CGI_RemoteIdent)
26 WriteCGIConstant(#PB_CGI_RemotePort)
27 WriteCGIConstant(#PB_CGI_RemoteUser)
28 WriteCGIConstant(#PB_CGI_RequestURI)
29 WriteCGIConstant(#PB_CGI_RequestMethod)
30 WriteCGIConstant(#PB_CGI_ScriptName)
31 WriteCGIConstant(#PB_CGI_ScriptFilename)
32 WriteCGIConstant(#PB_CGI_ServerAdmin)
33 WriteCGIConstant(#PB_CGI_ServerName)

```

```

32 WriteCGIConstant(#PB_CGI_ServerPort)
33 WriteCGIConstant(#PB_CGI_ServerProtocol)
34 WriteCGIConstant(#PB_CGI_ServerSignature)
35 WriteCGIConstant(#PB_CGI_ServerSoftware)
36 WriteCGIConstant(#PB_CGI_HttpAccept)
37 WriteCGIConstant(#PB_CGI_HttpAcceptEncoding)
38 WriteCGIConstant(#PB_CGI_HttpAcceptLanguage)
39 WriteCGIConstant(#PB_CGI_HttpCookie)
40 WriteCGIConstant(#PB_CGI_HttpForwarded)
41 WriteCGIConstant(#PB_CGI_HttpHost)
42 WriteCGIConstant(#PB_CGI_HttpPragma)
43 WriteCGIConstant(#PB_CGI_HttpReferer)
44 WriteCGIConstant(#PB_CGI_HttpUserAgent)
45
46 WriteCGIString("</body></html>")

```

See Also

ReadCGI()

103.12 FinishFastCGIRequest

Syntax

```
FinishFastCGIRequest()
```

Description

Finish the current FastCGI request and free all resources associated to it. It's not mandatory to use this command, as the request will be automatically finished when WaitFastCGIRequest() () is called again, or when the thread ends. It can still be useful in some special case where resources are light before doing other processing.

Parameters

None.

Return value

Returns non-zero if a new request has been processed.

Example

```

1  If Not InitCGI()
2      End
3  EndIf
4
5  If Not InitFastCGI(5600) ; Create the FastCGI program on port 5600
6      End

```

```

7   EndIf
8
9   While WaitFastCGIRequest()
10
11       If ReadCGI()
12           WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",
#PB_CGI_LastHeader) ; Write the headers to inform the browser of the
content format
13
14           WriteCGIString("<html><title>PureBasic - FastCGI</title><body>" +
15                           "Hello from PureBasic FastCGI !<br>" +
16                           "Actual time: <b>" + FormatDate("%hh:%ii", Date()) +
" </b>" +
17                           "</body></html>")
18
19           FinishFastCGIRequest()
20
21           ; Do some processing
22           ;
23           Delay(1000) ; Simulate large processing
24
25       EndIf
26   Wend

```

See Also

InitCGI() , InitFastCGI() , WaitFastCGIRequest()

103.13 InitCGI

Syntax

```
Result = InitCGI([MaxRequestSize])
```

Description

Initializes the CGI environment. This function has to be called successfully before using any other commands of this library.

Parameters

MaxRequestSize (optional) The maximum request size, in bytes (default is 50 MB). When sending big data through CGI (like binary files), it could be useful to specify a larger value.

Return value

Returns non-zero if the CGI environment has been correctly initialized.

Example

```

1  If Not InitCGI() Or Not ReadCGI()
2      End
3  EndIf
4
5  WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",
6      #PB_CGI_LastHeader) ; Write the headers to inform the browser of the
7      content format
8
9  WriteCGIString("<html><title>PureBasic - variables</title><body>" +
    "Hello from PureBasic CGI !" +
    "</body></html>")

```

See Also

ReadCGI()

103.14 InitFastCGI

Syntax

```
Result = InitFastCGI(LocalPort [, BoundIP$])
```

Description

Initializes FastCGI support. Once called, all the CGI commands switch automatically to FastCGI support. This library support threaded FastCGI processing, when enabling the 'thread-mode' in PureBasic. FastCGI support is only supported through a local socket. InitCGI() needs to be called before using this command.

Unlike a regular CGI program which is launched at every request, the FastCGI program stays in memory once launched and can handle any number of requests. It can be very useful if the CGI initialization is time consuming (for example connecting a database), so it's only performed once at start.

Parameters

LocalPort The local port to bind the FastCGI application. The web-server needs to be configured to use this port.

BoundIP\$ (optional) The IP address to bind the FastCGI application. For now, only "localhost" or "127.0.0.1" is accepted.

Return value

Returns non-zero if the FastCGI environment has been correctly initialized.

Remarks

Using FastCGI can be much easier for development than standard CGI, as the program can stay in memory and be debugged as usual PureBasic application.

To configure FastCGI support on Apache, you need to activate the 'mod_proxy' and 'mod_proxy_fcgi' modules, and then add a 'ProxyPass' declaration in the configuration:

```
ProxyPass /myfastcgiapp/ fcgi://localhost:5600/
```

Here, the url '/myfastcgiapp' will redirect to the FastCGI program bound on the port 5600. It's also possible to run the FastCGI program on distant server.

Example

```
1  If Not InitCGI()
2      End
3  EndIf
4
5  If Not InitFastCGI(5600) ; Create the FastCGI program on port 5600
6      End
7  EndIf
8
9  While WaitFastCGIRequest()
10
11      If ReadCGI()
12          WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",
13          #PB_CGI_LastHeader) ; Write the headers to inform the browser of the
14          content format
15
16          WriteCGIString("<html><title>PureBasic - FastCGI</title><body>" +
17          "Hello from PureBasic FastCGI !<br>" +
18          "Actual time: <b>" + FormatDate("%hh:%ii", Date()) +
19          "</b>" +
20          "</body></html>")
21      EndIf
22  Wend
```

See Also

InitCGI() , WaitFastCGIRequest()

103.15 ReadCGI

Syntax

```
Result = ReadCGI()
```

Description

Reads the CGI request input. InitCGI() has to be called successfully before trying to read the CGI input.

Parameters

None.

Return value

Returns non-zero if the CGI request has been successfully read. If the request was too big or another error occurred, zero is returned and the CGI program should be ended.

Example

```
1  If Not InitCGI() Or Not ReadCGI()
2      End
3  EndIf
4
5  WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",
6      #PB_CGI_LastHeader) ; Write the headers to inform the browser of the
    content format
7
8  WriteCGIString("<html><title>PureBasic - variables</title><body>" +
9      "Hello from PureBasic CGI !" +
    "</body></html>")
```

See Also

InitCGI()

103.16 WriteCGIFile

Syntax

```
Result = WriteCGIFile(Filename$)
```

Description

Writes a whole file to the CGI output. When sending binary data, the 'content-type' header should be set to 'application/octet-stream'.

Parameters

Filename\$ The file to write to the CGI output.

Return value

Returns non-zero if the file has been successfully written to the CGI output.

Example

```
1  If Not InitCGI() Or Not ReadCGI()
2      End
3  EndIf
4
```

```

5 WriteCGIHeader(#PB_CGI_HeaderContentType, "application/octet-stream")
6 WriteCGIHeader(#PB_CGI_HeaderContentDisposition, "attachment;
  filename=test.bmp", #PB_CGI_LastHeader)
7
8 WriteCGIFile(#PB_Compiler_Home +
  "examples/sources/data/PureBasic.bmp")

```

See Also

InitCGI() , WriteCGIHeader()

103.17 WriteCGIData

Syntax

```
Result = WriteCGIData(*Buffer, Size)
```

Description

Writes binary data to the CGI output. When sending binary data, the 'content-type' header should be set to 'application/octet-stream'.

Parameters

***Buffer** The memory buffer to write.

Size The size (in bytes) to write.

Return value

Returns non-zero if the data has been successfully written to the CGI output.

Example

```

1 If Not InitCGI() Or Not ReadCGI()
2   End
3 EndIf
4
5 WriteCGIHeader(#PB_CGI_HeaderContentType, "application/octet-stream")
6 WriteCGIHeader(#PB_CGI_HeaderContentDisposition, "attachment;
  filename=image.png", #PB_CGI_LastHeader)
7
8 If ReadFile(0, #PB_Compiler_Home + "examples/sources/data/world.png")
9   Size = Lof(0)
10  *Buffer = AllocateMemory(Size)
11  ReadData(0, *Buffer, Size) ; Read the whole file into the new
    allocated buffer
12
13  WriteCGIData(*Buffer, Size) ; Write the whole buffer to the CGI
    output

```



```

14
15     CloseFile(0)
16 EndIf

```

See Also

InitCGI() , WriteCGIHeader() , WriteCGIFile()

103.18 WriteCGIHeader

Syntax

```
Result = WriteCGIHeader(Header$, Value$ [, Flags])
```

Description

Writes a header to the CGI output. The headers needs to be written before any other data.

Parameters

Header\$ The header to write. It can be a custom or one of the following value:

```

#PB_CGI_HeaderContentLength : the length (in bytes) of the output
                             stream (implies binary data).
#PB_CGI_HeaderContentType   : the MIME content type of the output
                             stream.
#PB_CGI_HeaderExpires       : date and time when the document is
                             no longer valid
                             and should be reloaded by the
                             browser.
#PB_CGI_HeaderLocation      : server redirection (cannot be sent
                             as part of a complete header).
#PB_CGI_HeaderPragma        : turns document caching on and off.
#PB_CGI_HeaderStatus        : status of the request (cannot be
                             sent as part of a complete header).
#PB_CGI_HeaderContentDisposition : allows to specify a default
                             filename when sending out a file.
#PB_CGI_HeaderRefresh       : client reloads specified document.
#PB_CGI_HeaderSetCookie     : client stores specified data,
                             useful for keeping track of data
                             between requests.

```

Value\$ The header value to write.

Flags (optional) The string encoding to use. It can be one of the following value:

```

#PB_Ascii (default)
#PB_UTF8

```

Combined with one of the following value:

```

#PB_CGI_LastHeader : This is the last header written, meaning no
                    more headers can be sent.
                    This flag is mandatory for the last header
                    written.

```

Return value

Returns non-zero if the header has been successfully written to the CGI output.

Example

```
1  If Not InitCGI() Or Not ReadCGI()  
2      End  
3  EndIf  
4  
5  WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",  
6      #PB_CGI_LastHeader)  
7  
8  WriteCGIString("<html><title>PureBasic - test</title><body>" +  
9      "Hello from PureBasic CGI<br>" +  
      "</body></html>")
```

See Also

InitCGI() , WriteCGIFile() , WriteCGIStringN()

103.19 WriteCGIString

Syntax

```
Result = WriteCGIString(String$ [, Encoding])
```

Description

Write a string to the CGI output.

Parameters

String\$ The string to write.

Encoding (optional) The string encoding to use. It can be one of the following value:

```
#PB_UTF8 (default)  
#PB_Ascii
```

Return value

Returns non-zero if the string has been successfully written to the CGI output.

Example

```

1  If Not InitCGI() Or Not ReadCGI()
2      End
3  EndIf
4
5  WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",
6      #PB_CGI_LastHeader)
7
8  WriteCGIString("<html><title>PureBasic - test</title><body>" +
9      "Hello from PureBasic CGI<br>" +
10     "Actual time: <b>" + FormatDate("%hh:%ii", Date()) +
        "</b>" +
        "</body></html>")

```

See Also

InitCGI() , WriteCGIHeader() , WriteCGIStringN()

103.20 WriteCGIStringN

Syntax

Result = WriteCGIStringN(String\$ [, Encoding])

Description

Write a string to the CGI output, including a carriage return.

Parameters

String\$ The string to write.

Encoding (optional) The string encoding to use. It can be one of the following value:

```

#PB_UTF8 (default)
#PB_Ascii

```

Return value

Returns non-zero if the string has been successfully written to the CGI output.

Example

```

1  If Not InitCGI() Or Not ReadCGI()
2      End
3  EndIf
4
5  WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",
6      #PB_CGI_LastHeader)

```

```

7 | ; Using carriage return will make the page more readable when using
   | "Show page source" in the browser
8 | ;
9 | WriteCGIStringN("<html><title>PureBasic - test</title><body>")
10 | WriteCGIStringN("Hello from PureBasic CGI<br>")
11 | WriteCGIStringN("Actual time: <b>" + FormatDate("%hh:%ii", Date()) +
   | " </b>")
12 | WriteCGIStringN("</body></html>")

```

See Also

InitCGI() , WriteCGIHeader() , WriteCGIString()

103.21 WaitFastCGIRequest

Syntax

```
Result = WaitFastCGIRequest()
```

Description

Waits for a new incoming request. This command will halt the program execution until a new request is available. InitFastCGI() needs to be called successfully before using this command.

Parameters

None.

Return value

Returns non-zero if a new request has been processed.

Example

```

1 | If Not InitCGI()
2 |     End
3 | EndIf
4 |
5 | If Not InitFastCGI(5600) ; Create the FastCGI program on port 5600
6 |     End
7 | EndIf
8 |
9 | While WaitFastCGIRequest()
10 |
11 |     If ReadCGI()
12 |         WriteCGIHeader(#PB_CGI_HeaderContentType, "text/html",
   | #PB_CGI_LastHeader) ; Write the headers to inform the browser of the
   | content format
13 |

```

```
14     WriteCGIString("<html><title>PureBasic - FastCGI</title><body>" +
15                   "Hello from PureBasic FastCGI !<br>" +
16                   "Actual time: <b>" + FormatDate("%hh:%ii", Date()) +
    "</b>" +
17                   "</body></html>")
18     EndIf
19 Wend
```

See Also

InitCGI() , InitFastCGI() , FinishFastCGIRequest()

Chapter 104

Camera

Overview

Cameras are used to display the 3D world. You can manage them like real life cameras, which means you can rotate, move, change the field of vision and more. At least one camera is absolutely required in order to render the world to the screen. Many cameras can be used at the same time with different positions and views to allow cool effects like: split-screen, rear view, etc. `InitEngine3D()` must be called successfully before using the camera functions.

104.1 CameraBackColor

Syntax

```
CameraBackColor(#Camera, Color)
```

Description

Changes the camera background color. When a new camera is created, the default background color is set to black.

Parameters

#Camera The camera to use.

Color New color of the camera background. `RGB()` can be used to get a valid color value.

Return value

None.

104.2 CameraFollow

Syntax

```
CameraFollow(#Camera, ObjectID, Angle, Height, Distance,
             RotationPercent, PositionPercent [, Mode])
```

Description

Follow the specified object in a smooth manner, using interpolation.

Parameters

#Camera The camera to use.

ObjectID The object to follow. It can be one of the following type:

- Entity : use `EntityID()`
to get a valid ID.
- Light : use `LightID()`
to get a valid ID.
- Node : use `NodeID()`
to get a valid ID.
- ParticleEmitter: use `ParticleEmitterID()`
to get a valid ID.
- BillboardGroup : use `BillboardGroupID()`
to get a valid ID.
- Text3D : use `Text3DID()`
to get a valid ID.

Angle The angle of the camera relative to the followed object.

Height The absolute height of the camera.

Distance The distance of the camera relative to the followed object.

RotationPercent Value to apply when the camera rotate to get it again in the correct angle. Valid values are from 0 to 1.

PositionPercent Value to apply when the camera moves to get it again in the correct position. Valid values are from 0 to 1. When value is 0, the camera doesn't move. When value is 1, the camera is set to the final position, without interpolation.

Mode (optional) It can be one of the following value:

- `#True` : the camera will automatically look at the followed entity (default).
- `#False`: the camera will **not** automatically look at the followed entity.

Return value

None.

See Also

CreateCamera()

104.3 CameraFOV

Syntax

```
CameraFOV(#Camera, Angle)
```

Description

Changes a camera field of vision (FOV) which allows you to view a larger or smaller area of the scene. Angles above 90 degrees result in a wide-angle (fish-eye like) view. Angles lower than 30 degrees result in a stretched (telescopic) view. Typical values are between 45 and 60 degrees.

Parameters

#Camera The camera to use.

Angle The new view angle for the camera, in degrees.

Return value

None.

104.4 CameraID

Syntax

```
CameraID = CameraID(#Camera)
```

Description

Returns the unique ID which identifies the given '#Camera' in the operating system. This function is very useful when another library needs a camera reference.

104.5 CameraCustomParameter

Syntax

```
CameraCustomParameter(#Camera, ParameterIndex, Value1, Value2, Value3,  
    Value4)
```

Description

Sets a custom parameter value to the camera shader script (either GLSL or HLSL).

Parameters

#Camera The camera to use.

ParameterIndex The parameter index in the shader script.

Value1 The first parameter value.

Value2 The second parameter value (if the parameter only accept one value, this value will be ignored).

Value3 The third parameter value (if the parameter only accept two values, this value will be ignored).

Value4 The fourth parameter value (if the parameter only accept three values, this value will be ignored).

Return value

None.

104.6 CheckObjectVisibility

Syntax

```
Result = CheckObjectVisibility(#Camera, ObjectID)
```

Description

Checks if an object is visible within a camera view.

Parameters

#Camera The camera to use.

ObjectID The object to check. It can be one of the following type:

```
- Entity          : use EntityID()
to get a valid ID.
- Light           : use LightID()
to get a valid ID.
- Node            : use NodeID()
to get a valid ID.
- ParticleEmitter: use ParticleEmitterID()
to get a valid ID.
- BillboardGroup : use BillboardGroupID()
to get a valid ID.
- Text3D          : use Text3DID()
to get a valid ID.
```

Return value

Returns **#True** if object is within the selected camera view or **#False** if outside the visible area.

Example

```

1  InitEngine3D()
2  InitSprite()
3  InitKeyboard()
4
5  OpenWindow(0, 0, 0, 800, 600, "Object Visibility (Use arrows <-- -->
   and ESC to quit)", #PB_Window_ScreenCentered)
6  OpenWindowedScreen(WindowID(0), 0, 0, 800, 600)
7  CreateCamera(0, 0, 0, 100, 100)
8
9  CreateCube(0, 1)
10 CreateTexture(0, 100, 100)
11 If StartDrawing(TextureOutput(0))
12     DrawingMode(#PB_2DDrawing_Gradient)
13     CircularGradient(50, 50, 48)
14     Circle(50, 50, 48)
15     StopDrawing()
16 EndIf
17
18 CreateMaterial(0, TextureID(0))
19 ScaleMaterial(0, 0.1, 0.1)
20 CreateEntity(0, MeshID(0), MaterialID(0), 0, 0, -2)
21
22 Repeat
23 Repeat : Event = WindowEvent() : Until Event = 0
24
25 RenderWorld()
26 RotateEntity(0, 0.1, -0.3, 0.3, #PB_Relative)
27 ExamineKeyboard()
28
29 If KeyboardPushed(#PB_Key_Left)
30     MoveEntity(0, -0.1, 0, 0, #PB_Relative)
31 ElseIf KeyboardPushed(#PB_Key_Right)
32     MoveEntity(0, 0.1, 0, 0, #PB_Relative)
33 EndIf
34
35 If CheckObjectVisibility(0, EntityID(0)) = #False
36     Debug "Can't see the object anymore"
37 Else
38     Debug "I see it !"
39 EndIf
40
41 FlipBuffers()
42 Until KeyboardPushed(#PB_Key_Escape)

```

104.7 CameraDirection

Syntax

```
CameraDirection(#Camera, x, y, z)
```

Description

Changes the direction of a camera. The position of the camera is not changed.

Parameters

#Camera The camera to use.

x, y, z The direction vector (usually values between -1.0 and 1.0, if not it will be automatically normalized).

Return value

None.

104.8 CameraDirectionX

Syntax

```
Result = CameraDirectionX(#Camera [, Mode])
```

Description

Returns the 'x' direction vector of the camera.

Parameters

#Camera The camera to use.

Mode (optional) The mode to get the 'x' direction vector. It can be one of the following value:

```
#PB_Absolute: get the absolute 'x' direction vector of the camera  
in the world (default).  
#PB_Relative: get the 'x' direction vector of the camera relative  
to its parent.
```

Return value

Returns the 'x' direction vector of the camera. This value is always between -1.0 and 1.0.

104.9 CameraDirectionY

Syntax

```
Result = CameraDirectionY(#Camera [, Mode])
```

Description

Returns the 'y' direction vector of the camera.

Parameters

#Camera The camera to use.

Mode (optional) The mode to get the 'y' direction vector. It can be one of the following value:

```
#PB_Absolute: get the absolute 'y' direction vector of the camera
               in the world (default).
#PB_Relative: get the 'y' direction vector of the camera relative
               to its parent.
```

Return value

Returns the 'y' direction vector of the camera. . This value is always between -1.0 and 1.0.

104.10 CameraDirectionZ

Syntax

```
Result = CameraDirectionZ(#Camera [, Mode])
```

Description

Returns the 'z' direction vector of the camera.

Parameters

#Camera The camera to use.

Mode (optional) The mode to get the 'z' direction vector. It can be one of the following value:

```
#PB_Absolute: get the absolute 'z' direction vector of the camera
               in the world (default).
#PB_Relative: get the 'z' direction vector of the camera relative
               to its parent.
```

Return value

Returns the 'z' direction vector of the camera. This value is always between -1.0 and 1.0.

104.11 CameraFixedYawAxis

Syntax

```
CameraFixedYawAxis(#Camera, Enable [, VectorX, VectorY, VectorZ])
```

Description

Change the fixed yaw axis of the camera. The default behaviour of a camera is to yaw around its own Y axis.

Parameters

#Camera The camera to use.

Enable Enable or disable the use of a custom yaw axis. If set to **#True**, a new axis vector has to be specified. If set to **#False**, the camera will yaw around its own Y axis.

VectorX (optional) 'x' vector direction of the new yaw axis (usually a value between -1.0 and 1.0, if not it will be automatically normalized). 'Enable' parameter has to be set to have any effect.

VectorY (optional) 'y' vector direction of the new yaw axis (usually a value between -1.0 and 1.0, if not it will be automatically normalized). 'Enable' parameter has to be set to have any effect.

VectorZ (optional) 'z' vector direction of the new yaw axis (usually a value between -1.0 and 1.0, if not it will be automatically normalized). 'Enable' parameter has to be set to have any effect.

Return value

None.

104.12 CameraLookAt

Syntax

```
CameraLookAt(#Camera, x, y, z)
```

Description

The point (in world unit) that a camera will face.

Parameters

#Camera The camera to use.

x, y, z The position (in world unit) to point the camera at.

Return value

None.

104.13 CameraProjectionMode

Syntax

```
CameraProjectionMode(#Camera, Mode [, Width, Height])
```

Description

Change the **#Camera** projection mode.

Parameters

#Camera The camera to use.

Mode The value representing the way in which the world should be projected. This should be one of the following values:

```
#PB_Camera_Perspective : Renders the whole scene with perspective
#PB_Camera_Orthographic: Renders the whole scene in Orthographic
mode (no 3D depth)
```

Width (optional) Width of the plotting x range.

Height (optional) Height of the plotting y range.

Return value

None.

104.14 CameraProjectionX

Syntax

```
Result = CameraProjectionX(#Camera, x, y, z)
```

Description

Returns the 'x' position, in pixels, of a 3D point on the specified #Camera. If the point is outside of the camera current view, it returns -1. This is very useful to map 3D points to 2D screen.

104.15 CameraProjectionY

Syntax

```
Result = CameraProjectionY(#Camera, x, y, z)
```

Description

Returns the 'y' position, in pixels, of a 3D point on the specified #Camera. If the point is outside of the camera current view, it returns -1. This is very useful to map 3D points to 2D screen.

104.16 CameraRange

Syntax

```
CameraRange(#Camera, Near, Far)
```

Description

Changes camera near and far range.

Parameters

#Camera The camera to use.

Near The closest distance to the camera where the world should be rendered.

Far The largest distance to the camera where the world should be rendered.

Return value

None.

104.17 CameraRenderMode

Syntax

```
CameraRenderMode(#Camera, RenderMode)
```

Description

Changes the mode in which the world is displayed through a camera. When you create a new camera, using the `CreateCamera()` function, the default render mode is to display the world with full details and textures.

Parameters

#Camera The camera to use.

RenderMode The value representing the way in which the world should be rendered. This should be one of the following values:

```
#PB_Camera_Plot      : Renders the whole scene by showing only the  
                        vertices (plots)  
#PB_Camera_Wireframe: Renders the whole scene by showing only the  
                        triangles (lines)  
#PB_Camera_Textured  : Renders the whole scene at full detail,  
                        with textures.
```

Return value

None.

104.18 CameraReflection

Syntax

```
CameraReflection(#Camera, #MainCamera, EntityID)
```

Description

Set the `#Camera` as a reflective camera, using `#MainCamera` and the `EntityID` as source. A RTT Texture has to be created from `#Camera` using `CreateRenderTexture()` . The material which will use this RTT texture has to be defined with `SetMaterialAttribute(Material, #PB_Material_ProjectiveTexturing, #Camera)`.

This command has to be used in the rendering loop.

Parameters

#Camera The camera to use for the reflection.

#MainCamera The camera to use for the reflection source.

EntityID The `EntityID()` to use as source for the reflection.

Return value

None.

See Also

`SetMaterialAttribute()`

104.19 CameraRoll

Syntax

```
Result = CameraRoll(#Camera [, Mode])
```

Description

Get the roll of the `#Camera`.

Parameters

#Camera The camera to use.

Mode (optional) The mode to get the roll. It can be one of the following value:

```
#True : the roll is the raw value, but it can't be used in
        RotateCamera()
to get back the same orientation (default).
#False: the roll is adjusted, so it can be put back in
        RotateCamera()
to get back the same orientation.
```

Return value

The current roll value of the camera. This value is always between -180.0 and 180.0 degrees.

See Also

CameraYaw() , CameraPitch()

104.20 CameraPitch

Syntax

```
Result = CameraPitch(#Camera [, Mode])
```

Description

Get the pitch of the #Camera.

Parameters

#Camera The camera to use.

Mode (optional) The mode to get the pitch. It can be one of the following value:

```
#True : the pitch is the raw value, but it can't be used in
        RotateCamera()
to get back the same orientation (default).
#False: the pitch is adjusted, so it can be put back in
        RotateCamera()
to get back the same orientation.
```

Return value

The current pitch value of the camera. This value is always between -180.0 and 180.0 degrees.

See Also

CameraYaw() , CameraRoll()

104.21 CameraYaw

Syntax

```
Result = CameraYaw(#Camera [, Mode])
```

Description

Get the yaw of the #Camera.

Parameters

#Camera The camera to use.

Mode (optional) The mode to get the yaw. It can be one of the following value:

```
#True : the yaw is the raw value, but it can't be used in
RotateCamera()
to get back the same orientation (default).
#False: the yaw is adjusted, so it can be put back in
RotateCamera()
to get back the same orientation.
```

Return value

The current yaw value of the camera. This value is always between -180.0 and 180.0 degrees.

See Also

CameraPitch() , CameraRoll()

104.22 CameraViewX

Syntax

```
Result = CameraViewX(#Camera)
```

Description

Returns the 'x' position (in pixels) of the camera frame in the screen.

Parameters

#Camera The camera to use.

Return value

Returns the 'x' position (in pixels) of the camera frame in the screen.

104.23 CameraViewY

Syntax

```
Result = CameraViewY(#Camera)
```

Description

Returns the 'y' position (in pixels) of the camera frame in the screen.

Parameters

#Camera The camera to use.

Return value

Returns the 'y' position (in pixels) of the camera frame in the screen.

104.24 CameraViewWidth

Syntax

```
Result = CameraViewWidth(#Camera)
```

Description

Returns the width (in pixels) of the camera frame in the screen.

Parameters

#Camera The camera to use.

Return value

Returns the width (in pixels) of the camera frame in the screen.

104.25 CameraViewHeight

Syntax

```
Result = CameraViewHeight(#Camera)
```

Description

Returns the height (in pixels) of the camera frame in the screen.

Parameters

#Camera The camera to use.

Return value

Returns the height (in pixels) of the camera frame in the screen.

104.26 CameraX

Syntax

```
Result = CameraX(#Camera [, Mode])
```

Description

Returns the current 'x' position of the camera in the world.

Parameters

#Camera The camera to use.

Mode (optional) The mode to get the 'x' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'x' position of the camera in the
               world (default).
#PB_Relative: get the 'x' position of the camera relative to its
               parent.
```

Return value

Returns the 'x' position of the camera.

See Also

CameraY() , CameraZ() , MoveCamera()

104.27 CameraY

Syntax

```
Result = CameraY(#Camera [, Mode])
```

Description

Returns the current 'y' position of the camera in the world.

Parameters

#Camera The camera to use.

Mode (optional) The mode to get the 'y' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'y' position of the camera in the
               world (default).
#PB_Relative: get the 'y' position of the camera relative to its
               parent.
```

Return value

Returns the 'y' position of the camera.

See Also

CameraX() , CameraZ() , MoveCamera()

104.28 CameraZ

Syntax

```
Result = CameraZ(#Camera [, Mode])
```

Description

Returns the current position of the camera in the world.

Parameters

#Camera The camera to use.

Mode (optional) The mode to get the 'z' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'z' position of the camera in the  
world (default).  
#PB_Relative: get the 'z' position of the camera relative to its  
parent.
```

Return value

Returns the 'z' position of the camera.

See Also

CameraX() , CameraY() , MoveCamera()

104.29 CreateCamera

Syntax

```
Result = CreateCamera(#Camera, x, y, Width, Height [, VisibilityMask [,  
Lod]])
```

Description

Creates a new camera in the current world, at the position x,y with the specified dimensions. Note that these positions and sizes are the position and sizes of the display on the screen, not the position and size of the camera in the world.

Unlike other graphics related functions, these coordinates and dimensions are in percents, from 0 to 100 (float numbers can be used for more precise placement). This could seems odd, but 3D applications should be resolution independent and this is a good way of achieving that. If a camera is created with a height of 50% then it will always fill 50% of the height of the screen, irrespective of whether you use a screen which is 640*480 or 1600*1200. If a camera has already been created with the same number, the previous camera is automatically freed and replaced by the new one.

Cameras can be overlapped. This means that it is possible to display a camera which fills the whole screen and put a little camera inside which shows another place in the world. To achieve this, the order of creation is very important: the camera which will be on top should be created last.

Parameters

#Camera The number to identify the new camera. #PB_Any can be used to auto-generate this number.

x, y The position (in percentages) of the left edge of the start of the display for this camera. 0 means at the left of the screen, 100 means at the right of the screen. See the picture below for a better overview.

Width, Height The size (in percentages) the display for this camera should take up. See the picture below for a better overview.

VisibilityMask (optional) A mask to select which entities and billboards to display on this camera. The camera defines its own mask, and if the entity or billboard mask match, then it will be displayed. By default the entities and billboards have no mask, meaning they will be always displayed on all cameras.

Lod (optional) A Lod bias (Level Of Details) to apply to the entities which supports it. If Lod bias is < 1, a more detailed object will be display. If Lod bias is > 1, a less detailed entity will be displayed. The entity distance is divided by the Lod bias (ie: a Lod bias of 0.5 double the distance).

Return value

Returns zero if the camera can't be created. If #PB_Any is used as '#Camera' parameter, the new camera number will be returned as 'Result'.

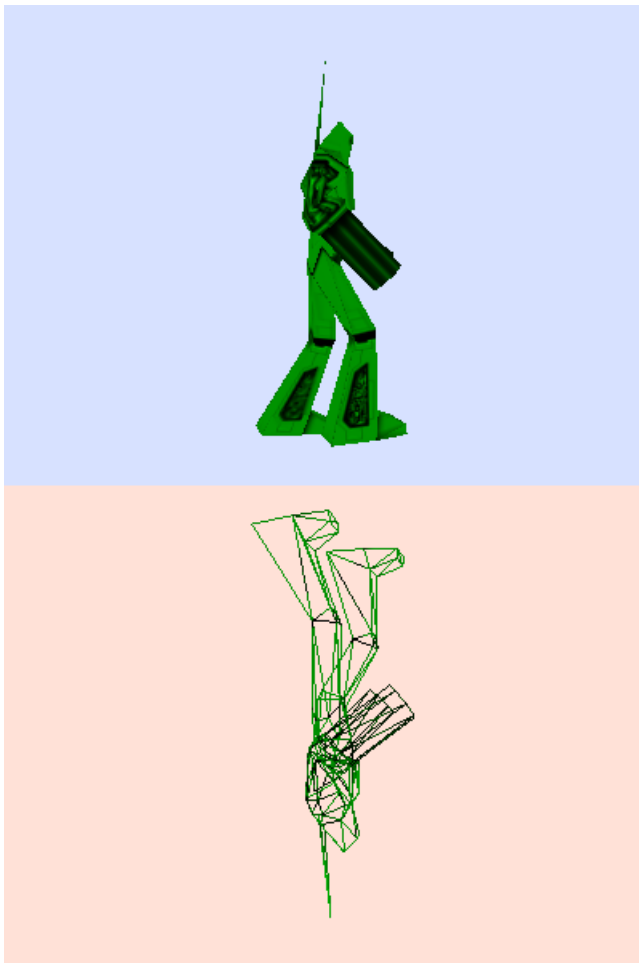
Remarks

Example: Camera creation position and dimension:

```
1  CreateCamera(0, 0, 0, 100, 100) ; Creates a camera which fill the
   whole screen
2
3  CreateCamera(0, 0, 0, 100, 50) ; Creates an horizontal split screen
   effect, for 2 players
4  CreateCamera(1, 0, 50, 100, 50) ; mode on the same screen
5
6  CreateCamera(0, 0, 0, 100, 100) ; Create a full screen camera
7  CreateCamera(1, 25, 0, 50, 10) ; And a rearview mirror effect.
8                                ; Note the rearview is last so that
   it is displayed on top of the full screen camera
```

Example: Two cameras separating screen into two halves:

```
1 ; upper camera
2 CreateCamera(0, 0, 0, 100, 50)
3 MoveCamera(0, 0, 50, 150, #PB_Absolute)
4 CameraBackColor(0, RGB(215, 225, 255))
5
6 ; camera below
7 CreateCamera(1, 0, 50, 100, 50)
8 MoveCamera(1, 0, 50, -150, #PB_Absolute)
9 CameraBackColor(1, RGB(255, 225, 215))
10 RotateCamera(1, 180, 0, 0)
```



See Also

FreeCamera() , ResizeCamera()

104.30 FreeCamera

Syntax

```
FreeCamera(#Camera)
```

Description

Frees a camera and releases all its associated memory. This camera must not be used (by using its number with the other functions in this library) after calling this function, unless you create it again.

Parameters

#Camera The camera to free. If **#PB_All** is specified, all the remaining cameras are freed.

Return value

None.

Remarks

All remaining cameras are automatically freed when the program ends.

See Also

CreateCamera()

104.31 IsCamera

Syntax

```
Result = IsCamera(#Camera)
```

Description

Tests if the given #Camera is a valid and correctly initialized camera.

This function is bulletproof and can be used with any value. If the 'Result' is not zero then the object is valid and initialized, else it returns zero. This is the correct way to ensure a camera is ready to use.

104.32 MoveCamera

Syntax

```
MoveCamera(#Camera, x, y, z [, Mode])
```

Description

Move the specified camera.

Parameters

#Camera The camera to move.

x, y, z The new position of the camera.

Mode (optional) The move mode. It can be one of the following values:

```
#PB_Relative: relative move, from the current camera position
              (default).
#PB_Absolute: absolute move to the specified position.
```

combined with one of the following values:

```
#PB_Local   : local move.
#PB_Parent  : move relative to the parent position.
#PB_World   : move relative to the world.
```

Return value

None.

See Also

RotateCamera()

104.33 ResizeCamera

Syntax

```
ResizeCamera(#Camera, x, y, Width, Height)
```

Description

Resizes the camera according to the specified dimension. All values are in percentages, like with CreateCamera() .

Parameters

#Camera The camera to resize.

x, y The new position of the camera display. This values are in percentages, from to 0 to 100.

Width, Height The new size of the camera display. This values are in percentages, from to 0 to 100.

Return value

None.

See Also

CreateCamera()

104.34 RotateCamera

Syntax

```
RotateCamera(#Camera, x, y, z [, Mode])
```

Description

Rotates the camera according to the specified x,y,z angle values.

Parameters

#Camera The camera to rotate.

x, y, z The new rotation to apply to the camera. Angles are in degree, with values ranging from 0 to 360.

Mode (optional) It can be one of the following value:

```
#PB_Absolute: absolute rotation (default).  
#PB_Relative: relative rotation based on the previous camera  
rotation.
```

Return value

None.

Chapter 105

Cipher

Overview

The cipher library is a set of functions useful to cipher or encode data. For example the SHA-2 is a very popular fingerprint routine, used in many areas due to its strong resistance to attacks.

105.1 AddCipherBuffer

Syntax

```
AddCipherBuffer(#Cipher, *Input, *Output, Size)
```

Description

Add new data to the cipher started with StartAESCipher() and copy the ciphered data into the output buffer.

Parameters

#Cipher The cipher to which the data should be added.

***Input** The input buffer.

***Output** The output buffer.

Size The size of the data to be ciphered. This is the amount of bytes which will be read from the input buffer and also written to the output buffer.

Return value

None.

See Also

StartAESCipher() , FinishCipher() , AESDecoder() , AESEncoder()

105.2 AESEncoder

Syntax

```
Result = AESEncoder(*Input, *Output, Size, *Key, Bits,  
    *InitializationVector [, Mode])
```

Description

Encodes the specified input buffer using the AES algorithm into the output buffer.

Parameters

***Input** The input buffer with the plain data.

***Output** The output buffer which will receive the encoded data. It has to be different than the input buffer.

Size The amount of bytes to encode. It has to be at least 16 bytes. To encode something smaller, padding has to be added before the encoding.

***Key** A buffer containing the key for encoding. Its size depends of the 'Bits' parameter: 16 bytes for 128-bit encryption, 24 bytes for 192 bit and 32 bytes for 256-bit. DeriveCipherKey() can be used to create this key from a password.

Bits The size of the key used by the ciphering. Valid values are 128, 192 and 256.

***InitializationVector** The InitializationVector is a random data block, used to initialize the ciphering to avoid breach in decoding (only needed when using the #PB_Cipher_CBC mode). The initialization vector is always 16 bytes long.

Mode (optional) This can be one of the following value:

```
#PB_Cipher_CBC: Default mode of encoding (Cipher Block Chaining).  
    Needs an '*InitializationVector'.  
                Recommended as more secure than ECB mode.  
#PB_Cipher_ECB: Alternative mode (Electronic CodeBook). It  
    doesn't uses random value nor chaining  
                (each block is ciphered independently) making it  
    very weak compared to CBC, and shouldn't be used for  
                serious ciphering.
```

Return value

Returns nonzero if the encoding was successful, zero otherwise.

Remarks

AES is an industry class cipher algorithm and is good balanced between speed and security. Here is the Wikipedia introduction about AES: 'In cryptography, the Advanced Encryption Standard (AES) is an encryption standard adopted by the U.S. government. The standard comprises three block ciphers, AES-128, AES-192 and AES-256, adopted from a larger collection originally published as Rijndael. Each AES cipher has a 128-bit block size, with key sizes of 128, 192 and 256-bit, respectively. The AES ciphers have been analyzed extensively and are now used worldwide.'

PureBasic uses a RFC compliant implementation of AES. More information can be found in the RFC 3602: <http://www.ietf.org/rfc/rfc3602.txt>.

Example: CBC

```
1 ; Encrypt some string
2 ;
3 String$ = "Hello this is a test for AES"
4
5 StringMemorySize = StringByteLength(String$) + SizeOf(Character) ;
   Space for the string and its null terminating character
6 *CipheredString = AllocateMemory(StringMemorySize)
7 *DecipheredString = AllocateMemory(StringMemorySize)
8
9 If AESEncoder(@String$, *CipheredString, StringByteLength(String$),
   ?Key, 128, ?InitializationVector)
10     Debug "Ciphered: "+PeekS(*CipheredString) ; warning, it will stop
   on the first null byte, only for demo purpose
11
12     AESDecoder(*CipheredString, *DecipheredString,
   StringByteLength(String$), ?Key, 128, ?InitializationVector)
13     Debug "Deciphered: "+PeekS(*DecipheredString)
14 EndIf
15
16 DataSection
17     Key:
18         Data.b $06, $a9, $21, $40, $36, $b8, $a1, $5b, $51, $2e, $03,
   $d5, $34, $12, $00, $06
19
20     InitializationVector:
21         Data.b $3d, $af, $ba, $42, $9d, $9e, $b4, $30, $b4, $22, $da,
   $80, $2c, $9f, $ac, $41
22 EndDataSection
```

Example: ECB

```
1 String$ = "Hello this is a test for AES"
2
3 *CipheredString = AllocateMemory(StringByteLength(String$) +
   SizeOf(Character)) ; Space for the string and its
4 *DecipheredString = AllocateMemory(StringByteLength(String$) +
   SizeOf(Character)) ; null terminating character
5
6 If AESEncoder(@String$, *CipheredString, MemorySize(*CipheredString),
   ?Key, 128, 0, #PB_Cipher_ECB)
7     Debug "Ciphered: " + PeekS(*CipheredString)
8
9     AESDecoder(*CipheredString, *DecipheredString,
   MemorySize(*DecipheredString), ?Key, 128, 0, #PB_Cipher_ECB)
10     Debug "Deciphered: " + PeekS(*DecipheredString)
11 EndIf
12
13 DataSection
14     Key:
15         Data.a $06, $a9, $21, $40, $36, $b8, $a1, $5b, $51, $2e, $03, $d5,
   $34, $12, $00, $06
16 EndDataSection
```

See Also

AESDecoder() , StartAESCipher() , DeriveCipherKey()

105.3 AESDecoder

Syntax

```
Result = AESDecoder(*Input, *Output, Size, *Key, Bits,  
    *InitializationVector [, Mode])
```

Description

Decodes the specified input buffer using the AES algorithm into the output buffer.

Parameters

***Input** The input buffer with the encoded data.

***Output** The output buffer which will receive the plain data. It has to be different than the input buffer.

Size The amount of bytes to decode. It has to be at least 16 bytes. To decode something smaller, padding has to be added before the encoding.

***Key** A buffer containing the key for decoding. Its size depends of the 'Bits' parameter: 16 bytes for 128-bit encryption, 24 bytes for 192 bit and 32 bytes for 256-bit. DeriveCipherKey() can be used to create this key from a password.

Bits The size of the key used by the ciphering. Valid values are 128, 192 and 256.

***InitializationVector** The InitializationVector is a random data block, used to initialize the ciphering to avoid breach in decoding (only needed when using the `#PB_Cipher_CBC` mode). Its size is always 16 bytes long. The contents of this data block must match the one which was used when encoding the data.

Mode (optional) This can be one of the following value:

```
#PB_Cipher_CBC: Default mode of encoding (Cipher Block Chaining).  
    Needs an '*InitializationVector'.  
                Recommended as more secure than ECB mode.  
#PB_Cipher_ECB: Alternative mode (Electronic CodeBook). It  
    doesn't uses random value nor chaining  
                (each block is ciphered independently) making it  
    very weak compared to CBC, and shouldn't be used for  
                serious ciphering.
```

Return value

Returns nonzero if the decoding was successful, zero otherwise.

Remarks

For more information about AES and source examples, see AESEncoder() .

See Also

AESEncoder() , StartAESCipher() , DeriveCipherKey()

105.4 CreatePasswordHash

Syntax

```
Result\$ = CreatePasswordHash(Password$ [, WorkFactor])
```

Description

Creates a hash digest of a password for storage and later verification of a password. It is not possible to recover the input password from the hash value, but passwords can be verified to see if they match the hash using VerifyPasswordHash() later.

Parameters

Password\$ The password to hash.

WorkFactor (optional) A parameter to influence the amount of work the hash function should require to create or verify a password hash. Larger values for this factor make it more costly to try to break a password by brute force but it will also make creating and verifying the password more costly for the application itself.
Allowed values are 4 to 31. Increasing the WorkFactor by one doubles the amount of work that the function does to create or later verify a password. The default value is 10.

Return value

Returns the hash of the password.

Remarks

The returned password hash includes a random component (a salt value) to ensure that multiple hashes created from the same input password do not create the same hash value. It is therefore not possible to compare the result from this function directly to check a password. The VerifyPasswordHash() function must be called to check a password instead.

This function implements the bcrypt password hashing algorithm.

Example

```
1 ; Create a hash digest for secure storage
2 Hash$ = CreatePasswordHash("secret")
3
4 ; Check passwords against the hash
5 Debug VerifyPasswordHash("secret", Hash$) ; correct
6 Debug VerifyPasswordHash("othersecret", Hash$) ; incorrect
```

See Also

VerifyPasswordHash()

105.5 VerifyPasswordHash

Syntax

```
Result = VerifyPasswordHash(Password$, Hash$)
```

Description

Checks if a given password matches the hash value previously created with CreatePasswordHash() .

Parameters

Password\$ The password to check.

Hash\$ The result from a previous call to CreatePasswordHash() .

Return value

Returns non-zero if the password matches the hash value or zero if it does not match or if the hash value was not valid.

Example

See CreatePasswordHash() for an example.

See Also

CreatePasswordHash()

105.6 DESFingerprint

Syntax

```
Result\$_ = DESFingerprint(Password$, Key$)
```

Description

Warning

This function is deprecated, it may be removed in a future version of PureBasic. It should not be used in newly written code.

Returns a DES encrypted version of the given Password\$. This command is deprecated because it is no longer secure enough for modern uses. Use the CreatePasswordHash() function instead.

See Also

CreatePasswordHash()

105.7 DeriveCipherKey

Syntax

```
Result = DeriveCipherKey(Password$, Salt$, Iterations, *Key, KeyBits,  
    Plugin [, PluginBits])
```

Description

Creates a cypher key with the specified number of bits from an input password for use in other cypher functions like encryption or decryption. This function implements the PBKDF2 key derivation algorithm.

Parameters

Password\$ The password for the key generation.

Salt\$ A "salt" string to make the result of the function unique even if the same password is used. This makes brute force attacks harder and prevents information disclosure in case of matching passwords.

This value does not have to be secret but it should be unique if possible.

Iterations The number of iterations of the PBKDF2 key generation loop. A larger number increases the time/computations needed to compute the key and therefore makes brute force attacks harder.

***Key** A pointer to the memory location that receives the generated key. The memory area must be at least KeyBits/8 bytes in size.

KeyBits The size of the key to generate in bits.

Plugin The plugin for the fingerprint function to use in the key computation. It can be one of the following values:

```
#PB_Cipher_MD5 : uses MD5 algorithm. UseMD5Fingerprint()  
needs to be called before to register this plugin.  
#PB_Cipher_SHA1 : uses SHA1 algorithm. UseSHA1Fingerprint()  
needs to be called before to register this plugin.  
#PB_Cipher_SHA2 : uses SHA2 algorithm. UseSHA2Fingerprint()  
needs to be called before to register this plugin.  
#PB_Cipher_SHA3 : uses SHA3 algorithm. UseSHA3Fingerprint()  
needs to be called before to register this plugin.
```

PluginBits (optional) The bits number to use for the fingerprint function. It is only supported for the following plugin:

```
#PB_Cipher_SHA2 : can be 224, 256 (default), 384 or 512.  
#PB_Cipher_SHA3 : can be 224, 256 (default), 384 or 512.
```

The number of bits for the fingerprint function is independent of the key size to generate. Any combination is allowed.

Return value

Returns nonzero if the key was generated. Returns zero only if the plugin parameters were invalid.

Example

```
1  UseSHA2Fingerprint()
2  *Key = AllocateMemory(32)
3
4  ; Create a 256bit key using SHA-512 hash function and 500000
   iterations
5  DeriveCipherKey("SecretPassword", "NonSecretSalt", 500000, *Key, 256,
   #PB_Cipher_SHA2, 512)
6
7  ; Show the key
8  ShowMemoryViewer(*Key, 32)
```

See Also

AESEncoder() , AESDecoder() , StartAESCipher()

105.8 StartFingerprint

Syntax

```
Result = StartFingerprint(#Fingerprint, Plugin [, Bits [, HmacKey$ [,
   HmacKeyFormat]])
```

Description

Initializes the calculation of a fingerprint in several steps. Unlike Fingerprint() function this allows to calculate the fingerprint of large data without the need to load it all into one continuous memory buffer.

Parameters

#Fingerprint The number to refer to this checksum calculation in later calls. #PB_Any can be used to auto-generate this number.

Plugin The plugin to use. It can be one of the following values:

```
#PB_Cipher_CRC32: uses CRC32 algorithm. UseCRC32Fingerprint()
needs to be called before to register this plugin.
#PB_Cipher_MD5   : uses MD5 algorithm. UseMD5Fingerprint()
needs to be called before to register this plugin.
#PB_Cipher_SHA1  : uses SHA1 algorithm. UseSHA1Fingerprint()
needs to be called before to register this plugin.
#PB_Cipher_SHA2  : uses SHA2 algorithm. UseSHA2Fingerprint()
needs to be called before to register this plugin.
#PB_Cipher_SHA3  : uses SHA3 algorithm. UseSHA3Fingerprint()
needs to be called before to register this plugin.
```

The plugin to use can be combined with the `#PB_Cipher_HMAC` flag to compute a HMAC (hash based message authentication code) instead of a regular fingerprint. In this case the "HmacKey\$" parameter is needed as well. See `StringFingerprint()` for example usage of the HMAC flag.

Bits (optional) The bits number to use for the fingerprint. It is only supported for the following plugin:

```
#PB_Cipher_SHA2 : can be 224, 256 (default), 384 or 512.  
#PB_Cipher_SHA3 : can be 224, 256 (default), 384 or 512.
```

HmacKey\$ (optional) The secret key to use in the HMAC computation. This parameter is only used if the `#PB_Cipher_HMAC` is added to the plugin parameter.

HmacKeyFormat (optional) The string format to use for the HMAC key. This can be one of the following values:

```
#PB_Ascii   : encode the key in ASCII  
#PB_UTF8    : encode the key on UTF-8 (default).  
#PB_Unicode : encode the key in Unicode
```

Return value

Returns the `#Fingerprint` value if `#PB_Any` was used for that parameter.

Remarks

`AddFingerprintBuffer()` can be used to add memory blocks into the calculation and `FinishFingerprint()` to finish the calculation and read the resulting hash.

Example

```
1  UseMD5Fingerprint()  
2  
3  *Buffer = AllocateMemory(200) ; Prepare a buffer with data  
4  If *Buffer  
5      PokeS(*Buffer, "The quick brown fox jumps over the lazy dog.", -1,  
6      #PB_Ascii)  
7      Length = MemoryStringLength(*Buffer, #PB_Ascii)  
8  
9      If StartFingerprint(0, #PB_Cipher_MD5) ; start the  
10         calculation  
11         AddFingerprintBuffer(0, *Buffer, Length/2) ; calculate  
12         part 1  
13         AddFingerprintBuffer(0, *Buffer+Length/2, Length/2) ; calculate  
14         part 2  
15  
16         MD5$ = FinishFingerprint(0) ; finish calculation  
17         Debug "MD5 checksum = " + MD5$  
18  
19         MD5$ = Fingerprint(*Buffer, Length, #PB_Cipher_MD5) ; compare to  
20         a calculation in 1 step  
21         Debug "MD5 checksum = " + MD5$  
22     EndIf  
23  
24     FreeMemory(*Buffer)  
25 EndIf
```

See Also

Fingerprint() , FileFingerprint() , StringFingerprint()

105.9 FinishCipher

Syntax

```
FinishCipher(#Cipher)
```

Description

Finish a cipher stream previously started with StartAESCipher() .

Parameters

#Cipher The cipher to finish.

Return value

None.

Remarks

This command should be called to finish a cipher calculation, even if the cipher is actually no longer needed as it does free any data associated with the cipher calculation.

See Also

StartAESCipher() , AddCipherBuffer()

105.10 IsCipher

Syntax

```
Result = IsCipher(#Cipher)
```

Description

Tests if the given #Cipher number is a valid and correctly initialized cipher.

Parameters

#Cipher The cipher to use.

Return value

Returns nonzero if #Cipher is a valid cipher and zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a cipher is ready to use.

See Also

StartAESCipher()

105.11 AddFingerprintBuffer

Syntax

```
AddFingerprintBuffer(#Fingerprint, *Buffer, Size)
```

Description

Add a new memory buffer into the calculation of a checksum started by StartFingerprint() . The checksum returned at the end of the calculation will include all the added buffers as if the checksum was calculated with all of them in one continuous memory buffer.

Parameters

#Fingerprint The fingerprint to which the data should be added.

***Buffer** The buffer to be added to the fingerprint.

Size The amount of bytes to be added to the fingerprint.

Return value

None.

Remarks

See StartFingerprint() for a code example and more information.

See Also

StartFingerprint() , FinishFingerprint()

105.12 FinishFingerprint

Syntax

```
Result\$$ = FinishFingerprint(#Fingerprint)
```

Description

Finishes the calculation of a fingerprint started by StartFingerprint() and returns it as an hexadecimal string.

Parameters

#Fingerprint The fingerprint to finish.

Return value

Returns the fingerprint as an hexadecimal string.

Remarks

This command should be called to finish a fingerprint calculation, even if the fingerprint is actually no longer needed as it frees up any data associated with the calculation as well.
See StartFingerprint() for a code example and more information.

See Also

StartFingerprint() , AddFingerprintBuffer()

105.13 IsFingerprint

Syntax

```
Result = IsFingerprint(#Fingerprint)
```

Description

Tests if the given #Fingerprint is a valid fingerprint calculation created by StartFingerprint() .

Parameters

#Fingerprint The fingerprint to test.

Return value

Returns nonzero if the given fingerprint is valid and zero otherwise.

Remarks

This function is bulletproof and can be used with any value.

See Also

StartFingerprint()

105.14 FileFingerprint

Syntax

```
Result\$ = FileFingerprint(Filename$, Plugin [, Bits [, Offset [,  
    Length [, HmacKey$ [, HmacKeyFormat]]]])
```

Description

Returns a fingerprint for the specified file.

Parameters

FileName\$ The file of which the fingerprint should be calculated.

Plugin The plugin to use. It can be one of the following value:

```
#PB_Cipher_CRC32: uses CRC32 algorithm. UseCRC32Fingerprint()  
needs to be called before to register this plugin.  
#PB_Cipher_MD5 : uses MD5 algorithm. UseMD5Fingerprint()  
needs to be called before to register this plugin.  
#PB_Cipher_SHA1 : uses SHA1 algorithm. UseSHA1Fingerprint()  
needs to be called before to register this plugin.  
#PB_Cipher_SHA2 : uses SHA2 algorithm. UseSHA2Fingerprint()  
needs to be called before to register this plugin.  
#PB_Cipher_SHA3 : uses SHA3 algorithm. UseSHA3Fingerprint()  
needs to be called before to register this plugin.
```

The plugin to use can be combined with the `#PB_Cipher_HMAC` flag to compute a HMAC (hash based message authentication code) instead of a regular fingerprint. In this case the "HmacKey\$" parameter is needed as well. See StringFingerprint() for example usage of the HMAC flag.

Bits (optional) The bits number to use for the fingerprint. It is only supported for the following plugin:

```
#PB_Cipher_SHA2 : can be 224, 256 (default), 384 or 512.  
#PB_Cipher_SHA3 : can be 224, 256 (default), 384 or 512.
```

Offset (optional) The offset (in bytes) from the start of the file to begin the checksum calculation.

Length (optional) The length (in bytes) to use for the checksum calculation.

HmacKey\$ (optional) The secret key to use in the HMAC computation. This parameter is only used if the `#PB_Cipher_HMAC` is added to the plugin parameter.

HmacKeyFormat (optional) The string format to use for the HMAC key. This can be one of the following values:

```
#PB_Ascii   : encode the key in ASCII
#PB_UTF8    : encode the key on UTF-8 (default).
#PB_Unicode : encode the key in Unicode
```

Return value

Returns the fingerprint if the calculation was successful. If the file isn't found or an error has happened, the result will be an empty string.

See Also

Fingerprint() , StartFingerprint() , StringFingerprint()

105.15 Fingerprint

Syntax

```
Result\$ = Fingerprint(*Buffer, Size, Plugin [, Bits [, HmacKey$ [,
    HmacKeyFormat]])
```

Description

Returns a fingerprint for the given buffer.

Parameters

***Buffer** The buffer containing the data.

Size The size of the given buffer.

Plugin The plugin to use. It can be one of the following value:

```
#PB_Cipher_CRC32: uses CRC32 algorithm. UseCRC32Fingerprint()
needs to be called before to register this plugin.
#PB_Cipher_MD5   : uses MD5 algorithm. UseMD5Fingerprint()
needs to be called before to register this plugin.
#PB_Cipher_SHA1  : uses SHA1 algorithm. UseSHA1Fingerprint()
needs to be called before to register this plugin.
#PB_Cipher_SHA2  : uses SHA2 algorithm. UseSHA2Fingerprint()
needs to be called before to register this plugin.
#PB_Cipher_SHA3  : uses SHA3 algorithm. UseSHA3Fingerprint()
needs to be called before to register this plugin.
```

The plugin to use can be combined with the `#PB_Cipher_HMAC` flag to compute a HMAC (hash based message authentication code) instead of a regular fingerprint. In this case the "HmacKey\$" parameter is needed as well. See StringFingerprint() for example usage of the HMAC flag.

Bits (optional) The bits number to use for the fingerprint. It is only supported for the following plugin:

```
#PB_Cipher_SHA2 : can be 224, 256 (default), 384 or 512.
#PB_Cipher_SHA3 : can be 224, 256 (default), 384 or 512.
```


HmacKey\$ (optional) The secret key to use in the HMAC computation. This parameter is only used if the `#PB_Cipher_HMAC` is added to the plugin parameter.

HmacKeyFormat (optional) The string format to use for the HMAC key. This can be one of the following values:

```
#PB_Ascii    : encode the key in ASCII
#PB_UTF8     : encode the key on UTF-8 (default).
#PB_Unicode  : encode the key in Unicode
```

Return value

Returns the fingerprint as an hexadecimal string.

Example

```
1  UseMD5Fingerprint()
2
3  *Buffer = AllocateMemory(500)
4  If *Buffer
5      PokeS(*Buffer, "The quick brown fox jumps over the lazy dog.", -1,
        #PB_Ascii)
6      MD5$ = Fingerprint(*Buffer, MemoryStringLength(*Buffer, #PB_Ascii),
        #PB_Cipher_MD5)
7      Debug "MD5 Fingerprint = " + MD5$
8      FreeMemory(*Buffer) ; would also be done automatically at the end
        of the program
9  EndIf
```

See Also

`FileFingerprint()` , `StartFingerprint()` , `StringFingerprint()`

105.16 StringFingerprint

Syntax

```
Result\$ = StringFingerprint(String$, Plugin [, Bits [, Format[,
    HmacKey$ [, HmacKeyFormat]]]])
```

Description

Returns a fingerprint for the given string.

Parameters

String\$ The string to hash.

Plugin The plugin to use. It can be one of the following value:

```

#PB_Cipher_CRC32: uses CRC32 algorithm. UseCRC32Fingerprint()
needs to be called before to register this plugin.
#PB_Cipher_MD5 : uses MD5 algorithm. UseMD5Fingerprint()
needs to be called before to register this plugin.
#PB_Cipher_SHA1 : uses SHA1 algorithm. UseSHA1Fingerprint()
needs to be called before to register this plugin.
#PB_Cipher_SHA2 : uses SHA2 algorithm. UseSHA2Fingerprint()
needs to be called before to register this plugin.
#PB_Cipher_SHA3 : uses SHA3 algorithm. UseSHA3Fingerprint()
needs to be called before to register this plugin.

```

The plugin to use can be combined with the `#PB_Cipher_HMAC` flag to compute a HMAC (hash based message authentication code) instead of a regular fingerprint. In this case the "HmacKey\$" parameter is needed as well.

Bits (optional) The bits number to use for the fingerprint. It is only supported for the following plugin:

```

#PB_Cipher_SHA2 : can be 224, 256 (default), 384 or 512.
#PB_Cipher_SHA3 : can be 224, 256 (default), 384 or 512.

```

Format (optional) The string format to use before hashing it. It can be one of the following value:

```

#PB_UTF8      : the string will be hashed in UTF8 format (default).
#PB_Ascii     : the string will be hashed in ASCII format.
#PB_Unicode   : the string will be hashed in Unicode (UTF16) format.

```

HmacKey\$ (optional) The secret key to use in the HMAC computation. This parameter is only used if the `#PB_Cipher_HMAC` is added to the plugin parameter.

HmacKeyFormat (optional) The string format to use for the HMAC key. This can be one of the following values:

```

#PB_Ascii    : encode the key in ASCII
#PB_UTF8     : encode the key on UTF-8 (default).
#PB_Unicode  : encode the key in Unicode

```

Return value

Returns the fingerprint as an hexadecimal string.

Example

```

1  UseMD5Fingerprint()
2
3  Debug StringFingerprint("yourpassword", #PB_Cipher_MD5)

```

Example: HMAC

```

1  UseSHA2Fingerprint()
2
3  ; Use hmacSha256 with both key and message in ascii
4  ; Should be
   4643978965ffcec6e6d73b36a39ae43ceb15f7ef8131b8307862ebc560e7f988
5  Debug StringFingerprint("the message to hash here",
   #PB_Cipher_SHA2|#PB_Cipher_HMAC, 256, #PB_Ascii, "the shared secret
   key here", #PB_Ascii)

```

See Also

FileFingerprint() , StartFingerprint() , Fingerprint()

105.17 UseMD5Fingerprint

Syntax

```
UseMD5Fingerprint()
```

Description

Register the MD5 fingerprint plugin for future use.

Parameters

None.

Return value

None.

Remarks

Here is a quick explanation taken from the RFC 1321 on MD5:

'The algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest. The MD5 algorithm is intended for digital signature applications.'

MD5 hashes are often used for password encryption, but it should be avoided as it has been found to be vulnerable to several attacks. More information about MD5 can be found in the RFC 1321:

<http://www.ietf.org/rfc/rfc1321.txt>.

Example

```
1  UseMD5Fingerprint()
2
3  Debug StringFingerprint("yourpassword", #PB_Cipher_MD5)
```

See Also

UseSHA1Fingerprint() (), UseSHA2Fingerprint() (), UseSHA3Fingerprint() (), UseCRC32Fingerprint()
()

105.18 UseSHA1Fingerprint

Syntax

```
UseSHA1Fingerprint()
```

Description

Register the SHA1 fingerprint plugin for future use.

Parameters

None.

Return value

None.

Remarks

SHA1 can be used to calculate a checksum to verify that a 'message' has not been altered. Unlike CRC32 it is nearly impossible to modify the original message and still produce the same SHA1 fingerprint.

Here is a quick explanation taken from the RFC 3174 on SHA1:

'The SHA-1 is called secure because it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest. Any change to a message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify.'

More information can be found in the RFC 3174: <http://www.ietf.org/rfc/rfc3174.txt>.

Example

```
1  UseSHA1Fingerprint()
2
3  Debug StringFingerprint("yourpassword", #PB_Cipher_SHA1)
```

See Also

UseMD5Fingerprint() (), UseSHA2Fingerprint() (), UseSHA3Fingerprint() (), UseCRC32Fingerprint() ()

105.19 UseSHA2Fingerprint

Syntax

```
UseSHA2Fingerprint()
```

Description

Register the SHA2 fingerprint plugin for future use. The standard 224-bit, 256-bit, 384-bit and 512-bit variants are supported.

Parameters

None.

Return value

None.

Remarks

From [Wikipedia](#): SHA-2 includes significant changes from its predecessor, SHA-1. In 2005, an algorithm emerged for finding SHA-1 collisions in about 2000-times fewer steps than was previously thought possible. Although (as of 2015) no example of a SHA-1 collision has been published yet, the security margin left by SHA-1 is weaker than intended, and its use is therefore no longer recommended for applications that depend on collision resistance, such as digital signatures. Although SHA-2 bears some similarity to the SHA-1 algorithm, these attacks have not been successfully extended to SHA-2.

Example

```
1 UseSHA2Fingerprint()
2
3 Debug StringFingerprint("yourpassword", #PB_Cipher_SHA2, 512) ; Use
   SHA2-512 variant
```

See Also

UseMD5Fingerprint() (), UseSHA1Fingerprint() (), UseSHA3Fingerprint() (), UseCRC32Fingerprint() ()

105.20 UseSHA3Fingerprint

Syntax

```
UseSHA3Fingerprint()
```

Description

Register the SHA3 fingerprint plugin for future use. The standard 224-bit, 256-bit, 384-bit and 512-bit variants are supported.

Parameters

None.

Return value

None.

Example

```
1 UseSHA3Fingerprint()
2
3 Debug StringFingerprint("yourpassword", #PB_Cipher_SHA3, 512) ; Use
   SHA3-512 variant
```

See Also

UseMD5Fingerprint() (), UseSHA1Fingerprint() (), UseSHA2Fingerprint() (), UseCRC32Fingerprint() ()

105.21 UseCRC32Fingerprint

Syntax

```
UseCRC32Fingerprint()
```

Description

Register the CRC32 fingerprint plugin for future use.

Parameters

None.

Return value

None.

Remarks

CRC32 is a 32-bit fingerprint not intended for password storage as it's easily crackable, but for quick data integrity check. For example, zip files have a CRC32 checksum at the end of each file to be sure that the zip is not corrupted. The main advantage of CRC32 over MD5 or other fingerprint algorithm is its very high speed.

Example

```
1 UseCRC32Fingerprint()
2
3 Debug StringFingerprint("any text", #PB_Cipher_CRC32)
```

See Also

UseMD5Fingerprint() (), UseSHA1Fingerprint() (), UseSHA2Fingerprint() (), UseSHA3Fingerprint() ()

105.22 Base64DecoderBuffer

Syntax

```
Result = Base64DecoderBuffer(*InputBuffer, InputSize, *OutputBuffer,
                             OutputSize)
```

Description

Decodes the specified Base64 encoded buffer.

Parameters

***InputBuffer** The buffer containing the encoded data.

InputSize The size of the input buffer.

***OutputBuffer** The output buffer where the plain data will be copied.

OutputSize The size of the output buffer.

The output buffer can be up to 25% smaller than the input buffer, with a minimum size of 64 bytes. It's recommended to get a slightly larger buffer, like 20% smaller to avoid overflows.

Return value

Returns the length of the decoded data in bytes.

Example

```
1  Example$ = "This is a test string!"
2  Decoded$ = Space(1024)
3  Encoded$ = Space(1024)
4
5  Debug Base64EncoderBuffer(@Example$, StringByteLength(Example$),
6  @Encoded$, StringByteLength(Encoded$))
7  Debug Encoded$
8
9  Debug Base64DecoderBuffer(@Encoded$, StringByteLength(Encoded$),
  @Decoded$, StringByteLength(Decoded$))
  Debug Decoded$
```

Example: Encoding & Decoding from a DataSection

```
1  DataSection
2  ?? Test:
3  ?? Data.a $00, $01, $02, $03, $04, $05, $06, $07
4  ?? Data.a $08, $09, $0A, $0B, $0C, $0D, $0E, $0F
```

```

5  ?? TestEnd:
6  EndDataSection
7
8  Size = (?TestEnd - ?Test) * 1.35
9  If Size < 64
10 ?? Size = 64
11 EndIf
12
13 *EncodeBuffer = AllocateMemory(Size)
14 Size = Base64EncoderBuffer(?Test, ?TestEnd - ?Test, *EncodeBuffer,
    MemorySize(*EncodeBuffer))
15 Encoded$ = PeekS(*EncodeBuffer, Size, #PB_Ascii)
16 Debug Encoded$
17
18 *DecodeBuffer = AllocateMemory(Size)
19 Size = PokeS(*EncodeBuffer, Encoded$, StringByteLength(Encoded$,
    #PB_Ascii), #PB_Ascii|#PB_String_NoZero)
20 Size = Base64DecoderBuffer(*EncodeBuffer, Size, *DecodeBuffer,
    MemorySize(*DecodeBuffer))
21 ShowMemoryViewer(*DecodeBuffer, Size)

```

See Also

Base64EncoderBuffer()

105.23 Base64EncoderBuffer

Syntax

```
Result = Base64EncoderBuffer(*InputBuffer, InputSize, *OutputBuffer,
    OutputSize [, Flags])
```

Description

Encodes the specified buffer using the Base64 algorithm. This is widely used in e-mail programs but can be useful for any other programs which need an ASCII only (7 bit, only from 32 to 127 characters) encoding for raw binary files.

Parameters

***InputBuffer** The buffer containing the plain data.

InputSize The size of the input buffer.

***OutputBuffer** The output buffer where the encoded data will be copied.

OutputSize The size of the output buffer.

The output buffer should be at least 33% bigger than the input buffer, with a minimum size of 64 bytes. It's recommended to get a slightly larger buffer, like 35% bigger to avoid overflows.

Flags (optional) It can be a combination of the following values:

#PB_Cipher_NoPadding: it will **not** insert additional '=' at the end of the encoded buffer to pad it to 3 bytes boundary.

`#PB_Cipher_URL` : it will use a slightly different encoding, mainly used in URL. The usual '+' and '/' encoded characters will be respectively encoded to '-' and '_'

Return value

Returns the length of the encoded data in bytes.

Example

```
1 Example$ = "This is a test string!"
2 Decoded$ = Space(1024)
3 Encoded$ = Space(1024)
4
5 Debug Base64EncoderBuffer(@Example$, StringByteLength(Example$),
6   @Encoded$, StringByteLength(Encoded$))
7 Debug Encoded$
8
9 Debug Base64DecoderBuffer(@Encoded$, StringByteLength(Encoded$),
10  @Decoded$, StringByteLength(Decoded$))
11 Debug Decoded$
```

See Also

Base64DecoderBuffer()

105.24 Base64Decoder

Syntax

```
Result = Base64Decoder(Input$, *OutputBuffer, OutputSize)
```

Description

Decodes the specified Base64 encoded string.

Parameters

Input\$ A string containing the encoded data.

***OutputBuffer** The output buffer where the plain data will be copied.

OutputSize The size of the output buffer.

The output buffer can be up to 25% smaller than the input buffer, with a minimum size of 64 bytes. It's recommended to get a slightly larger buffer, like 20% smaller to avoid overflows.

Return value

Returns the length of the decoded data in bytes.

Example

```
1 String$ = "This is a test string!"
2
3 ; Encoding an Ascii string
4 *Text = Ascii(String$)
5 Encoded$ = Base64Encoder(*Text, StringByteLength(String$, #PB_Ascii))
6 Debug "Encoded: " + Encoded$
7
8 *DecodedBuffer = AllocateMemory(1024)
9 Base64Decoder(Encoded$, *DecodedBuffer, 1024)
10 Debug "Decoded: '" + PeekS(*DecodedBuffer, -1, #PB_Ascii) + "'"
11
12 FreeMemory(*Text)
13 FreeMemory(*DecodedBuffer)
```

See Also

Base64Encoder()

105.25 Base64Encoder

Syntax

```
Result\$ = Base64Encoder(*InputBuffer, InputSize [, Flags])
```

Description

Encodes the specified buffer using the Base64 algorithm. This is widely used in e-mail programs but can be useful for any other programs which need an ASCII only (7 bit, only from 32 to 127 characters) encoding for raw binary files.

Parameters

***InputBuffer** The buffer containing the plain data.

InputSize The size of the input buffer.

Flags (optional) It can be a combination of the following values:

```
#PB_Cipher_NoPadding: it will not insert additional '=' at the
                      end of the encoded buffer to pad it to 3 bytes boundary.
#PB_Cipher_URL       : it will use a slightly different encoding,
                      mainly used in URL. The usual '+' and '/' encoded characters
                      will be respectively encoded to '-' and '_'
```

Return value

Returns the encoded data as a string.

Example

```
1 String$ = "This is a test string!"
2
3 ; Encoding an UTF-8 string
4 *Text = UTF8(String$)
5 Encoded$ = Base64Encoder(*Text, StringByteLength(String$, #PB_UTF8))
6 Debug "Encoded: " + Encoded$
7
8 *DecodedBuffer = AllocateMemory(1024)
9 Base64Decoder(Encoded$, *DecodedBuffer, 1024)
10 Debug "Decoded: '" + PeekS(*DecodedBuffer, -1, #PB_UTF8) + "'"
11
12 FreeMemory(*Text)
13 FreeMemory(*DecodedBuffer)
```

See Also

Base64Decoder() , Base64DecoderBuffer() , Base64EncoderBuffer()

105.26 StartAESCipher

Syntax

```
Result = StartAESCipher(#Cipher, *Key, Bits, *InitializationVector,
    Mode)
```

Description

Initializes a new AES cipher stream where data can be added using AddCipherBuffer() .

Parameters

#Cipher The number which identifies this new cipher. #PB_Any can be used to auto-generate this number.

***Key** A buffer containing the key for decoding. Its size depends of the 'Bits' parameter: 16 bytes for 128-bit encryption, 24 bytes for 196-bit and 32 bytes for 256-bit. DeriveCipherKey() can be used to create this key from a password.

Bits The size of the key used by the ciphering. Valid values are 128, 192 and 256.

***InitializationVector** The InitializationVector is a random data block, used to initialize the ciphering to avoid breach in decoding (only needed when using the #PB_Cipher_CBC mode). Its size depends of the 'Bits' parameter: 16 bytes for 128-bit encryption, 24 bytes for 196-bit and 32 bytes for 256-bit.

Mode This parameter can be a combination of one the following values:

```
#PB_Cipher_Decode: The stream is used to decode data.
#PB_Cipher_Encode: The stream is used to encode data.
```

with

```

#PB_Cipher_CBC: Default mode of encoding (Cipher Block Chaining).
Needs an '*InitializationVector'.
                Recommended as more secure than ECB mode.
#PB_Cipher_ECB: Alternative mode (Electronic CodeBook). It
                doesn't uses random value nor chaining
                (each block is ciphered independently) making it
                very weak compared to CBC, and shouldn't be used for
                serious ciphering.

```

Return value

If `#PB_Any` was used as the `#Cipher` parameter then the auto-generated `#Cipher` number is returned.

Remarks

New buffers to be encoded or decoded can be added with `AddCipherBuffer()` . Once a cipher is finished, `FinishCipher()` has to be called.

For more information about AES, see `AESEncoder()` .

See Also

`AddCipherBuffer()` , `FinishCipher()` , `AESEncoder()` , `AESDecoder()` , `DeriveCipherKey()`

105.27 OpenCryptRandom

Syntax

```
Result = OpenCryptRandom()
```

Description

Opens the cryptographic safe pseudorandom number generator. The `CryptRandom()` and `CryptRandomData()` commands can be used to read data from the opened generator.

Parameters

None.

Return value

Returns non-zero if the random number generator could be successfully opened. If the result is zero then there is no cryptographic safe random number generator available on the system.

Remarks

This generator provides random data which is strong enough for cryptographic purposes such as generating keys for the `AESEncoder()` function. The source for the random data is the `"/dev/urandom"`

device on Linux or Mac OSX and the "Microsoft Cryptography API" on Windows. See the `CryptRandomData()` command for an example.

See Also

`CryptRandom()` , `CryptRandomData()` , `CloseCryptRandom()`

105.28 CloseCryptRandom

Syntax

```
CloseCryptRandom()
```

Description

Closes the cryptographic safe pseudorandom number generator that was opened with `OpenCryptRandom()` and frees its resources.

Parameters

None.

Return value

None.

See Also

`OpenCryptRandom()` , `CryptRandom()` , `CryptRandomData()`

105.29 CryptRandom

Syntax

```
Result = CryptRandom(Maximum)
```

Description

Returns a random number (integer) which lies between (and including) 0 and the Maximum value from the cryptographic safe pseudorandom number generator.

Parameters

Maximum The maximum value to be returned by the function. 'Maximum' may not exceed the positive long value: 2147483647.

Return value

Returns the generated random number.

Remarks

The generator has to be opened first with the `OpenCryptRandom()` command.

Important: Using a 'Maximum' value which is not one less than a power of two will cause certain numbers to be more likely than others which could be used for a statistical attack. This is the result of dividing the generated random number to fit the specified range.

To generate larger amounts of random data, use the `CryptRandomData()` function. To generate random numbers from the faster but not cryptographic safe pseudorandom number generator, use the `Random()` function.

See Also

`OpenCryptRandom()` , `CryptRandomData()` , `CloseCryptRandom()` , `Random()`

105.30 CryptRandomData

Syntax

```
Result = CryptRandomData(*Buffer, Length)
```

Description

Fills the specified memory buffer with random data from the cryptographic safe pseudorandom number generator.

Parameters

***Buffer** The buffer to fill.

Length The size of the buffer in bytes.

Return value

Returns nonzero if the random data was generated successfully and zero otherwise.

Remarks

The generator has to be opened first with the `OpenCryptRandom()` command.

To generate random data from the faster but not cryptographic safe pseudorandom number generator, use the `RandomData()` function.

Example

```
1  *Key = AllocateMemory(16)
2
3  If OpenCryptRandom() And *Key
4      CryptRandomData(*Key, 16)
5
6      Text$ = "Generated Key:"
7      For i = 0 To 15
8          Text$ + " " + RSet(Hex(PeekB(*Key+i), #PB_Byte), 2, "0")
9      Next i
10
11     CloseCryptRandom()
12 Else
13     Text$ = "Key generation is not available"
14 EndIf
15
16 MessageRequester("Example", Text$)
```

See Also

OpenCryptRandom() , CryptRandom() , CloseCryptRandom() RandomData()

Chapter 106

Clipboard

Overview

The clipboard is the standard way to share information and data between applications, which are currently running on the OS. It also gives the user a temporary location where information and data may be stored and accessed quickly and easily. For instance, when text is cut within an editor, that text goes to the clipboard, where it may be retrieved later, by pasting it into another application. PureBasic gives the programmer the ability to develop applications capable of cutting, copying and pasting text or images via the standard clipboard.

106.1 ClearClipboard

Syntax

```
ClearClipboard()
```

Description

Clears the clipboard. This means that any data contained within the clipboard is flushed and no longer available from the clipboard.

Parameters

None.

Return value

None.

See Also

SetClipboardText() , SetClipboardImage()

106.2 GetClipboardImage

Syntax

```
Result = GetClipboardImage(#Image [, Depth])
```

Description

Creates a new image from the clipboard image data (if any).

Parameters

#Image The number for the new image. **#PB_Any** can be used to auto-generate this number.

Depth (optional) The depth of the new image. Valid values are 24 (default) or 32-bit.

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Image** parameter, then the new image number is returned.

Remarks

On Windows, transparency may not be detected after pasting the clipboard image into third-party software. (Saving this image as a .png preserves transparency.) This is a problem of the BMP format and how Windows handles it. It is possible to have a 32bit bitmap without alpha channel. In this case all alpha values are 0 but the image is supposed to be opaque. Since images in `GetClipboardImage()` can come from outside of PB we cannot know whether this is supposed to be a fully transparent or fully opaque image. This is why we only use the alpha channel if there is at least one pixel with a value other than 0. Otherwise we treat it as an opaque image.

The image may be freed by using the `FreeImage()` function.

For further use of the image, there are functions available such as `DrawImage()` or `ImageGadget()` .

See Also

`GetClipboardText()`

106.3 GetClipboardText

Syntax

```
Text\[extract_itex] = GetClipboardText()
```

Description

Returns the text string currently contained within the clipboard.

Parameters

None.

Return value

Returns the content of the clipboard if it contains text. If the clipboard contains no data or no text data, then an empty string is returned.

See Also

GetClipboardImage()

106.4 SetClipboardImage

Syntax

```
SetClipboardImage(#Image)
```

Description

Places a copy of the given image onto the clipboard. If the clipboard currently contains an image, then it will be overwritten.

Parameters

#Image The image to put onto the clipboard.

Return value

None.

Example

```
1  If CreateImage(0, 26, 20)
2      StartDrawing(ImageOutput(0))
3          Box(0, 0, 26, 20, RGB(255, 255, 255))
4          Circle(13, 10, 5, RGB(255, 0, 0))
5      StopDrawing()
6      SetClipboardImage(0)
7  EndIf
8
9  ; now if you open a graphics application and paste the clipboard
10 ; contents,
    ; you will see a Japanese flag ;-)
```

See Also

SetClipboardText() , ClearClipboard()

106.5 SetClipboardText

Syntax

```
SetClipboardText (Text$)
```

Description

Stores a string into the clipboard. If the clipboard already contains text, it will be overwritten.

Parameters

Text\$ The string you want to store in the clipboard.

Return value

None.

See Also

SetClipboardImage() , ClearClipboard()

Chapter 107

Console

Overview

This library allow the programmer to create console mode applications. This is used to create small programs that don't require a user interface or be executed from the function line. Also, console functions can be really helpful to debug a program by printing out some information to the console, without stopping the program flow.

If your program is intended to be a pure console application (i.e. not a GUI application which sometimes opens a console) then you must remember to set the executable format to 'Console' when you are compiling your programs.

You should start with the `OpenConsole()` function, since you must use that function to open a console (character mode) display window before you use any other console-related functions.

Please note, that if you create console programs with PureBasic you will still need Windows 95 or later to run them. These programs must be executed only from the Windows function prompt. These programs are not real MS-DOS programs!

107.1 ClearConsole

Syntax

```
ClearConsole()
```

Description

Clears the whole console content using the current background color. The background color is set with `ConsoleColor()` . The console has to be in graphical mode, see `EnableGraphicalConsole()` .

Parameters

None.

Return value

None.

Example

```
1  If OpenConsole()
2      EnableGraphicalConsole(1)
3
4      PrintN("You will never see me")
5      ClearConsole()
6
7      PrintN("Press return to exit")
8      Input()
9  EndIf
```

See Also

EnableGraphicalConsole() , ConsoleColor()

Supported OS

Windows

107.2 CloseConsole

Syntax

```
CloseConsole()
```

Description

Close the console previously opened with OpenConsole() . Once the console has been closed, it's not possible to use any console-related functions unless you open the console again. The console will automatically be closed when your program ends. No effects with Linux and MacOS.

Parameters

None.

Return value

None.

Example

```
1  For i = 0 To 4
2      If OpenConsole()
3          PrintN("This is console #" + Str(i))
4          PrintN("Press return to close console")
5          Input()
6          CloseConsole()
```

```
7   EndIf
8   Next
```

See Also

OpenConsole()

107.3 ConsoleError

Syntax

```
ConsoleError(Message$)
```

Description

Writes the Message string (plus a newline) to the standard error output of the program. This output can be read for example with the ReadProgramError() function of the Process library.

Parameters

Message\$ The string to send.

Return value

None.

See Also

Print() , PrintN()

107.4 ConsoleTitle

Syntax

```
ConsoleTitle(Title$)
```

Description

Changes the console title to the specified string. The console title is typically shown in the title bar of the window which the console is in (when you are viewing the console in a graphical environment such as your desktop).

Under Windows, the console title is also the text shown in the start bar and in the task manager representing your console application.

Parameters

Title\$ The new title for the console.

Return value

None.

Example

```
1  If OpenConsole()  
2      ConsoleTitle("The ConsoleTitle example program")  
3      PrintN("Press return to exit")  
4      Input()  
5  EndIf
```

See Also

EnableGraphicalConsole()

Supported OS

Windows

107.5 ConsoleColor

Syntax

```
ConsoleColor(CharacterColor, BackgroundColor)
```

Description

Change the colors used by the text display. Any characters printed after calling this function will use the new colors.

Parameters

CharacterColor The color to be used for the characters. The color values range from 0 to 15, which are the 16 colors available in console mode:

- 0 - Black (default background)
- 1 - Blue
- 2 - Green
- 3 - Cyan
- 4 - Red
- 5 - Magenta
- 6 - Brown
- 7 - Light grey (default foreground)
- 8 - Dark grey

```

9 - Bright blue
10 - Bright green
11 - Bright cyan
12 - Bright red
13 - Bright magenta
14 - Yellow
15 - White

```

BackgroundColor The color for the character background. The values are the same as for the CharacterColor.

Return value

None.

Example

```

1  If OpenConsole()
2      For Foreground = 0 To 15
3          For Background = 0 To 15
4              ConsoleColor(Foreground, Background)
5              Print(Right(Hex(Background), 1))
6          Next
7      Next
8      PrintN(" ")
9  Next
10
11  ConsoleColor(7, 0)
12  PrintN("Press return to exit")
13  Input()
14 EndIf

```



See Also

EnableGraphicalConsole()

107.6 EnableGraphicalConsole

Syntax

```
EnableGraphicalConsole(State)
```

Description

Changes the way the characters are drawn on the console between a graphical and a text-only mode.

Parameters

State If 'State' is 1, then console will be switched to graphical mode, else if 'State' is 0, it will switch back to text mode.

Return value

None.

Remarks

The default mode of the console is the text mode, which means the text can't be positioned anywhere in the console, but redirections (through pipes) work correctly.

When being in the graphical mode, functions like `ClearConsole()` or `ConsoleLocate()` are available and the text can be positioned anywhere in the console screen, which allow to do console games or text only applications (which can be useful when accessed remotely through telnet or ssh). The redirection (pipes) doesn't work correctly when being in graphical mode.

Example

```
1  If OpenConsole()
2      EnableGraphicalConsole(1)
3      ConsoleLocate(7, 8)
4      PrintN("Press return to exit")
5      Input()
6  EndIf
```

See Also

`ConsoleLocate()` , `ConsoleColor()` , `ClearConsole()`

Supported OS

Windows

107.7 Inkey

Syntax

```
String\$ = Inkey()
```

Description

Returns a character string if a key is pressed during the call of Inkey(). It doesn't interrupt (halt) the program flow. If special keys (non-ASCII) have to be handled, RawKey() should be called after Inkey().

Parameters

None.

Return value

Returns a string containing the pressed character if it is a non-special key, and an empty string otherwise.

Example

```
1  If OpenConsole()
2      PrintN("Press Escape to exit")
3
4      Repeat
5          KeyPressed$ = Inkey()
6
7          If KeyPressed$ <> ""
8
9              PrintN("You pressed: " + KeyPressed$)
10             PrintN("It has a raw code of: "+Str(RawKey()))
11
12             ElseIf RawKey()
13
14                 PrintN("You pressed a non ASCII key.")
15                 PrintN("It has a raw code of: "+Str(RawKey()))
16
17             Else
18                 Delay(20) ; Don't eat all the CPU time, we're on a multitask OS
19             EndIf
20
21         Until KeyPressed$ = Chr(27) ; Wait until escape is pressed
22     EndIf
```

Remarks

The ASCII codes and numeric values reported in this description may change depending on the code page you have configured at boot time for keyboard input. However, the concepts should be the same and you can use the above example to figure out the real values for your system.

A table with ASCII codes is available [here](#) .

See Also

RawKey() , Input()

107.8 Input

Syntax

```
String\$ = Input()
```

Description

Allows the program to catch an entire line of characters. This function locks the program execution and waits until the user presses the return key.

Parameters

None.

Return value

Returns the string that the user entered before pressing the return key.

Remarks

If the console is in graphical mode (see EnableGraphicalConsole()), the line can't be longer than the console width (no more keys are accepted when the line gets too long).

In non-graphical mode, a special return-value of #PB_Input_Eof (equals Chr(4)) will be received if the user enters a single Ctrl+D in the console, or a file that has been redirected to the programs input has reached its end. For compatibility with other console applications on Windows, #PB_Input_Eof is also received when Ctrl+Z is entered in the console.

If not line based or raw input is needed, ReadConsoleData() can be used in non-graphical mode.

Example

```
1  If OpenConsole()
2      Print("Enter your name and press return: ")
3      Name$ = Input()
4
5      PrintN("Hello " + Name$ + ", nice to meet you.")
6      PrintN("Press return to exit")
7      Input()
8  EndIf
```

See Also

Inkey() , RawKey()

107.9 ConsoleLocate

Syntax

```
ConsoleLocate(x, y)
```

Description

Moves the cursor to the given position, in character coordinates. Any text you print after calling this function will start from the specified coordinates.

Parameters

x The horizontal position in the console to move to (starting from 0)

y The vertical position in the console to move to (starting from 0)

Return value

None.

Remarks

The console has to be in graphical mode, see `EnableGraphicalConsole()` .

Example

```
1  If OpenConsole()  
2      EnableGraphicalConsole(1)  
3  
4      For i = 0 To 200  
5          ConsoleLocate(Random(79), Random(24))  
6          Print("*")  
7      Next  
8  
9      ConsoleLocate(30, 10)  
10     PrintN("Press return to exit")  
11     Input()  
12 EndIf
```

See Also

`EnableGraphicalConsole()`

Supported OS

Windows

107.10 ConsoleCursor

Syntax

`ConsoleCursor(Height)`

Description

Changes the display of the cursor, which is the indicator used to show where the next displayed character will be drawn. This function allows you to change the height of the cursor.

Parameters

Height The new height of the cursor indicator. This value can be zero (for an invisible cursor) or range from 1 to 10 for the height of the cursor. Currently, 3 values are available:

```
1 : Underline cursor (default)
5 : Mid-height cursor
10: Full-height cursor
```

Return value

None.

Remarks

By default the cursor is a flashing underline in consoles under the Windows OS. Note that you might need to make the console window full-screen to see the true effect of this function. The console has to be in graphical mode, see `EnableGraphicalConsole()` .

Example

```
1  If OpenConsole()
2      EnableGraphicalConsole(1)
3
4      For CursorHeight = 0 To 10
5          ConsoleCursor(CursorHeight)
6          PrintN("Press return to increase cursor size")
7          Input()
8      Next
9
10     PrintN("Press return to exit")
11     Input()
12 EndIf
```

See Also

`EnableGraphicalConsole()`

Supported OS

Windows

107.11 Print

Syntax

```
Print(Text$)
```

Description

Displays the specified 'Text\$' in the console.

Parameters

Text\$ The text to display. In graphical mode the length of the string can't exceed the width of the console, otherwise the string will be truncated (have the end cut off).

Return value

None.

Remarks

In graphical mode it's possible to change the position with the function `ConsoleLocate()` . To change the appearance of the string when it is printed, the function `ConsoleColor()` should be used.

The cursor will be moved to the next character after the end of the string that is printed. If you print over the right edge of the console the text will wrap around to the left edge on the next line down. If you print off the bottom of the console window, the console window will scroll its contents up.

To output raw data on the non-graphical console (for pipe communication) `WriteConsoleData()` can be used.

Example

```
1  If OpenConsole()  
2      Print("This is quite a long string.")  
3      Print("You see how this one joins onto the end of the previous  
   one?")  
4      Print("That is because the cursor moves to the end of the string  
   and not onto a new line.")  
5      Print("Hopefully the text will also have been wrapped by now.")  
6  
7      PrintN(" ")  
8      PrintN(" ")  
9      PrintN("Press return to exit")  
10     Input()  
11 EndIf
```

See Also

PrintN() , Input()

107.12 PrintN

Syntax

```
PrintN(Text$)
```

Description

Displays the specified 'Text\$' in the console and adds a new line.

Parameters

Text\$ The text to display before the newline. In graphical mode the length of the string can't exceed the width of the console, otherwise the string will be truncated (have the end cut off).

Return value

None.

Remarks

In graphical mode it's possible to change the position with the function ConsoleLocate() . To change the appearance of the string when it is printed, the function ConsoleColor() should be used.

The cursor will be moved to the start of the next line after the end of the string. If you print over the right edge of the console the text will wrap around to the left edge on the next line down. If you print off the bottom of the console window, the console window will scroll its contents up. You can use Print() instead, if you want to continue the output directly after last character.

To output raw data on the non-graphical console (for pipe communication) WriteConsoleData() can be used.

Example

```
1  If OpenConsole()
2      PrintN("This is quite a long string.")
3      PrintN("You see how this one does not join onto the end of the
   previous one?")
4      PrintN("That is because the cursor moves to the start of the next
   line after the end of the string.")
5
6      PrintN("Press return to exit")
7      Input()
8  EndIf
```

See Also

Print() , Input()

107.13 OpenConsole

Syntax

```
Result = OpenConsole([Title$ [, Mode]])
```

Description

Open a console window. This function must be called before any other function of this library. Only one console can be opened at the same time in a PureBasic program.

Parameters

Title\$ (optional) The title for the new console window. On Windows, specifying a title allow the saving of custom console parameter, like font, color etc. It has no effect on other OS.

Mode (optional) The mode to use for console output. It can one of the following value:

```
#PB_UTF8: string will use UTF-8 format when printed to the
          console (default).
#PB_Ascii: string will use ASCII format when printed to the
           console.
#PB_Unicode: string will use UTF-16 format when printed to the
             console. Can be useful on Windows
              when using string redirection, and the target
              program expect UTF-16 input. Has no effect
              on Linux or OS X (will use UTF-8).
```

Return value

If Result is 0, it has failed and all further call to console functions must be disabled.

Remarks

The console can be closed using the CloseConsole() function.

With EnableGraphicalConsole() the console can be switched between text and graphics mode.

On Microsoft Windows, there are two different executable formats: Win32 and Console. If you want to create a standard console application, like 'dir', 'del' etc. you must compile the executable using the 'Console' format (Compiler Option menu in the PureBasic IDE). On Linux and OS X, there is no special Console format however setting the Compiler option to 'Console' will launch a terminal window automatically when you run your program from the IDE.

Example

```
1  OpenConsole()
2  PrintN("Waiting 5 secs before quit...")
3  Delay(5000)
```


See Also

CloseConsole() , EnableGraphicalConsole()

107.14 ReadConsoleData

Syntax

```
Result = ReadConsoleData(*Buffer, Size)
```

Description

Reads raw input from the console. This function is only supported in non-graphical mode. It can be used to read not line-based data, or text like files redirected to the program through a pipe.

Parameters

***Buffer** The memory buffer to which the data should be read.

Size The maximum amount of data (in bytes) to be read.

Return value

Returns the number of bytes actually read from the input. If zero is returned, this means that there is no more input to read. (an end of file was received)

Remarks

This function waits until there is some input to read. It will only return without reading data if there was an error or an EOF (End Of File) condition.

Example

```
1  ; This reads a passed image from the console and displays it in a
   window
2  ; Compile this to an exe and run it like "myexe < image.bmp"
3  ;
4  ; (set "Executable format" To "Console" in the compiler options!)
5  ; (works only with Bitmaps and icons unless you use an Image Decoder)
6  ;
7  OpenConsole()
8  TotalSize = 0
9  BufferFree = 10000
10 *Buffer = AllocateMemory(BufferFree)
11
12 Repeat
13     ReadSize = ReadConsoleData(*Buffer+TotalSize, BufferFree) ; read a
   block of data
14     TotalSize + ReadSize
15     BufferFree - ReadSize
16     If BufferFree < 100 ; resize the buffer if it is not large enough
```

```

17     BufferFree = 10000
18     *Buffer = ReAllocateMemory(*Buffer, TotalSize+10000)
19 EndIf
20 Until ReadSize = 0 ; once 0 is returned, there is nothing else to read
21
22 If TotalSize > 0 ; display the image if successful
23     If CatchImage(0, *Buffer, TotalSize)
24         If OpenWindow(0, 0, 0, ImageWidth(0), ImageHeight(0), "Image",
25             #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
26             ImageGadget(0, 0, 0, ImageWidth(0), ImageHeight(0), ImageID(0))
27             Repeat
28                 Until WaitWindowEvent() = #PB_Event_CloseWindow
29             End
30         EndIf
31     EndIf
32 EndIf
33 MessageRequester("Error", "Not a valid image.")

```

See Also

WriteConsoleData() , AllocateMemory()

107.15 RawKey

Syntax

Result = RawKey()

Description

Returns the raw key code of the last Inkey() function call. It's useful for extended (non-ASCII) keys (for example, function keys, arrows, etc).

Parameters

None.

Return value

Returns the key code of the last pressed key.

Remarks

It is not only alphanumeric keys that have an ASCII value. The escape key (27), return key (13) tab key (9) and backspace key (8) are just four examples. A table with ASCII codes you find here .

Example

```

1  If OpenConsole()
2      PrintN("Press Escape to exit")
3
4      Repeat
5          KeyPressed$ = Inkey()
6
7          If KeyPressed$ <> ""
8
9              PrintN("You pressed: " + KeyPressed$)
10             PrintN("It has a raw code of: "+Str(RawKey()))
11
12             ElseIf RawKey()
13
14                 PrintN("You pressed a non ASCII key.")
15                 PrintN("It has a raw code of: "+Str(RawKey()))
16
17             Else
18                 Delay(20) ; Don't eat all the CPU time, we're on a multitask OS
19             EndIf
20
21         Until KeyPressed$ = Chr(27) ; Wait until escape is pressed
22     EndIf

```

See Also

Inkey() , Input()

Supported OS

Windows

107.16 WriteConsoleData

Syntax

```
Result = WriteConsoleData(*Buffer, Size)
```

Description

Writes raw data to the console output. This function is only supported in non-graphical mode. It can be used to output data other than text to the console that can then be redirected to a file or another program.

Parameters

***Buffer** The memory buffer from which the data is read.

Size The amount of data (in bytes) to write.

Return value

Returns the number of bytes actually written to the output.

See Also

`ReadConsoleData()`

Chapter 108

Database

Overview

The database library is an easy set of functions to access SQLite, PostgreSQL, MySQL, DBMaria or any database type (Oracle, MySQL, Access, etc) via ODBC. Accessing and updating data is done using SQL queries, therefore it is necessary to have an understanding of SQL syntax.

Here are some links about SQL syntax:

[W3Schools SQL Tutorial](#)

[SQLite SQL functions](#)

[PostgreSQL manual](#)

Database programming starts by initializing the database environment using `UseODBCDatabase()` , `UseSQLiteDatabase()` , `UseMySQLDatabase()` and `UsePostgreSQLDatabase()` .

Note: Under Windows, prior to using an ODBC database, it is necessary to establish an ODBC "User Data Source" which makes your database available via ODBC and usable with this Database library. For more information, refer to Windows's ODBC help document.

108.1 AffectedDatabaseRows

Syntax

```
Result = AffectedDatabaseRows(#Database)
```

Description

Returns the number of rows affected by the last `DatabaseUpdate()` operation.

Parameters

`#Database` The database to use.

Return value

Returns the number of rows affected by the last `DatabaseUpdate()` operation.

See Also

DatabaseUpdate()

108.2 CloseDatabase

Syntax

```
CloseDatabase( #Database )
```

Description

Close the specified #Database (and connections/transactions if any). No further operations are allowed on this database.

Parameters

#Database The database to close. If **#PB_All** is specified, all remaining databases are closed.

Return value

None.

Remarks

All remaining opened databases are automatically closed when the program ends.

See Also

OpenDatabase() , OpenDatabaseRequester()

108.3 DatabaseColumns

Syntax

```
Result = DatabaseColumns( #Database )
```

Description

Returns the numbers of columns (fields) from the last executed database query with DatabaseQuery() .

Parameters

#Database The database to use.

Return value

Returns the number of columns from the last database query.

See Also

DatabaseColumnName() , DatabaseColumnType() , DatabaseColumnSize()

108.4 DatabaseColumnIndex

Syntax

```
Result = DatabaseColumnIndex(#Database, ColumnName$)
```

Description

Returns the index of the column after executing a query with DatabaseQuery() in the opened #Database. This can be useful for use with commands like GetDatabaseLong() which require a column index.

Parameters

#Database The database to use.

#ColumnName\$ The name of the column to get the index of.

Return value

Returns the index of the specified column, -1 if #ColumnName\$ is invalid. This is only valid after having executed a query with DatabaseQuery() .

See Also

DatabaseQuery() , GetDatabaseBlob() , GetDatabaseDouble() , GetDatabaseFloat() , GetDatabaseString() , GetDatabaseQuad()

108.5 DatabaseColumnName

Syntax

```
Text\$$ = DatabaseColumnName(#Database, Column)
```

Description

Return the name of the specified column in the #Database.

Parameters

#Database The database to use.

Column The column to use.

Return value

Returns the name of the column.

See Also

DatabaseColumns() , DatabaseColumnType() , DatabaseColumnSize()

108.6 DatabaseColumnSize

Syntax

```
Result = DatabaseColumnSize(#Database , Column)
```

Description

Return the size of the specified column in the #Database. It is especially useful when the size of the column can change depending of the records, like a blob or string column.

Parameters

#Database The database to use.

Column The column to use.

Return value

Returns the size of the column in bytes.

See Also

DatabaseColumns() , DatabaseColumnType() , DatabaseColumnName()

108.7 DatabaseColumnType

Syntax

```
Result = DatabaseColumnType(#Database , Column)
```

Description

Return the type of the specified column in the #Database.

Parameters

#Database The database to use.

Column The column to use.

Return value

Returns the type of the given column. If Result is 0, the type is undefined or the function has failed (e.g. it was not possible to determine the data type).

Type values can be:

```
#PB_Database_Long   : Numeric format (a Long (.l) in PureBasic)
#PB_Database_String: String format (a String (.s) in PureBasic)
#PB_Database_Float  : Numeric float format (a Float (.f) in PureBasic)
#PB_Database_Double: Numeric double format (a Double (.d) in
    PureBasic)
#PB_Database_Quad   : Numeric quad format (a Quad (.q) in PureBasic)
#PB_Database_Blob    : Blob format
```

See Also

DatabaseColumns() , DatabaseColumnName() , DatabaseColumnSize()

108.8 DatabaseDriverDescription

Syntax

```
Text\$ = DatabaseDriverDescription()
```

Description

Returns the description of the current database driver. Drivers are listed using the ExamineDatabaseDrivers() and NextDatabaseDriver() functions.

Parameters

None.

Return value

Returns the description string.

Remarks

This is an ODBC database specific command.

See Also

ExamineDatabaseDrivers() , NextDatabaseDriver() , DatabaseDriverName()

108.9 DatabaseDriverName

Syntax

```
Text\$ = DatabaseDriverName()
```

Description

Return the name of the current database driver. Drivers are listed using the ExamineDatabaseDrivers() and NextDatabaseDriver() functions.

Parameters

None.

Return value

Returns the name of the driver.

Remarks

This is an ODBC database specific command.

See Also

ExamineDatabaseDrivers() , NextDatabaseDriver() , DatabaseDriverDescription()

108.10 DatabaseError

Syntax

```
Error\$ = DatabaseError()
```

Description

Returns a description of the last database error in text format. This is especially useful with the following functions: OpenDatabase() , DatabaseQuery() and DatabaseUpdate() .

Parameters

None.

Return value

Returns the error description.

Example

```
1 ; First, connect to a database with an employee table
2 ;
3 If DatabaseQuery(#Database, "SELECT * FROM employee") ; Get all the
   records in the 'employee' table
4 ; ...
5 FinishDatabaseQuery(#Database)
6 Else
7     MessageRequester("Error", "Can't execute the query:
   "+DatabaseError())
8 EndIf
```

See Also

DatabaseQuery() , DatabaseUpdate()

108.11 DatabaseID

Syntax

```
DatabaseID = DatabaseID(#Database)
```

Description

Returns the unique ID which identifies the given '#Database' in the operating system. This function is useful when another library needs a database reference.

Parameters

#Database The database to use.

Return value

Returns the ID for this database connection.

108.12 DatabaseQuery

Syntax

```
Result = DatabaseQuery(#Database, Request$ [, Flags])
```

Description

Executes a SQL query on the given database. Only queries which doesn't change the database records are accepted ('SELECT' like queries). To performs database modification, use DatabaseUpdate() .

Parameters

#Database The database to use.

Request\$ The SQL query to execute.

Flags (optional) The flags to use. It can be one of the following value:

```
#PB_Database_StaticCursor : performs the query to access the
                           result in a sequential manner. It's not possible to rewind
                           with PreviousDatabaseRow()
or FirstDatabaseRow()
on some drivers, but it is the faster way to get the data
(default).
#PB_Database_DynamicCursor: performs the query to access the
                           result in a random manner using PreviousDatabaseRow()
or FirstDatabaseRow()
.
                           It can be slower, or even unsupported
                           on some drivers.
```

Return value

Returns nonzero if the query was successful or zero if it failed (due to a SQL error or a badly-formatted query).

Remarks

If the query has succeeded then NextDatabaseRow() can be used to list returned records (see the example below). In the event of an error, the error text can be retrieved with DatabaseError() . It is safe to use NextDatabaseRow() even if the request doesn't return any records. To get the number of columns returned by the query, use DatabaseColumns() .

Once the query results aren't needed anymore, FinishDatabaseQuery() has to be called to release all the query resources.

The query can contain place holders for bind variables. Such variables must be set before calling the function using SetDatabaseString() , SetDatabaseLong() etc. After executing the query, the bound variables are cleared and have to be set again for future calls. The syntax for specifying bind variables in SQL is dependent on the database. The example below demonstrate the syntax.

Example

```
1 ; First, connect to a database with an employee table
2 ;
3 If DatabaseQuery(#Database, "SELECT * FROM employee") ; Get all the
  records in the 'employee' table
4
5 While NextDatabaseRow(#Database) ; Loop for each records
6   Debug GetDatabaseString(#Database, 0) ; Display the content of
  the first field
```

```

7      Wend
8
9      FinishDatabaseQuery(#Database)
10     EndIf

```

Example: Bind variables with SQLite, MySQL and ODBC

```

1      ; SQLite, MySQL and ODBC shares the same syntax for bind variables.
      It is indicated by the '?' character
2      ;
3      SetDatabaseString(#Database, 0, "test")
4      If DatabaseQuery(#Database, "SELECT * FROM employee WHERE id=?")
5          ; ...
6      EndIf

```

Example: PostgreSQL

```

1      ; PostgreSQL uses another syntax: $1, $2.. into the statement to
      indicate the undefined parameter
2      ;
3      SetDatabaseString(#Database, 0, "test")
4      If DatabaseQuery(#Database, "SELECT * FROM employee WHERE id=$1")
5          ; ...
6      EndIf

```

See Also

DatabaseUpdate(), NextDatabaseRow() SetDatabaseString(), SetDatabaseLong(), SetDatabaseQuad()
, SetDatabaseFloat(), SetDatabaseDouble() SetDatabaseBlob(), SetDatabaseNull()

108.13 DatabaseUpdate

Syntax

```
Result = DatabaseUpdate(#Database, Request$)
```

Description

Executes a modification query on the given database. This command doesn't return any record. To perform a 'SELECT' like query, use DatabaseQuery().

Parameters

#Database The database to use.

Request\$ The query to execute.

Return value

Returns nonzero if the query was successful or zero if it failed (due to a SQL error or a badly-formatted query).

Remarks

This function is similar to DatabaseQuery() but is independent from the NextDatabaseRow() function. Therefore it's not possible to do a 'SELECT' like query with this function. This function is useful for updating records in the database. In the event of an error, the error text can be retrieved with DatabaseError() .

The update request can contain place holders for bind variables. Such variables must be set before calling the function using SetDatabaseString() , SetDatabaseLong() etc. After executing the update, the bound variables are cleared and have to be set again for future calls. The syntax for specifying bind variables in SQL is dependent on the database. The example below demonstrate the syntax.

Example

```
1 ; First, connect to a database with an employee table
2 ;
3 If DatabaseQuery(#Database, "SELECT * FROM employee") ; Get all the
   records in the 'employee' table
4
5 While NextDatabaseRow(#Database) ; Loop for each records
6
7     ; Update the 'checked' field for each records, assuming the 'id'
   field is
8     ; the first one in the 'employee' table
9     ;
10    DatabaseUpdate(#Database, "UPDATE employee SET checked=1 WHERE
   id="+GetDatabaseString(#Database, 0))
11    Wend
12
13    FinishDatabaseQuery(#Database)
14 EndIf
```

Example: Bind variables with SQLite, MySQL and ODBC

```
1 ; SQLite, MySQL and ODBC shares the same syntax for bind variables.
   It is indicated by the '?' character
2 ;
3 SetDatabaseLong(0, 0, 1)
4 SetDatabaseString(0, 1, "test")
5 DatabaseUpdate(0, "UPDATE employee SET checked=? WHERE id=?")
```

Example: PostgreSQL

```
1 ; PostgreSQL uses another syntax: $1, $2.. into the statement to
   indicate the undefined parameter
2 ;
3 SetDatabaseLong(0, 0, 1)
4 SetDatabaseString(0, 1, "test")
```

5 | `DatabaseUpdate(0, "UPDATE employee SET checked=$1 WHERE id=$2")`

See Also

`DatabaseQuery()` `SetDatabaseString()` , `SetDatabaseLong()` , `SetDatabaseQuad()` , `SetDatabaseFloat()` , `SetDatabaseDouble()` `SetDatabaseBlob()` , `SetDatabaseNull()`

108.14 ExamineDatabaseDrivers

Syntax

```
Result = ExamineDatabaseDrivers()
```

Description

Examines the database drivers available on the system.

Parameters

None.

Return value

If ODBC isn't installed or no drivers are available, it returns 0, otherwise `NextDatabaseDriver()` can be used to list all the drivers.

Remarks

This is an ODBC database specific command.

See Also

`NextDatabaseDriver()` , `DatabaseDriverName()` , `DatabaseDriverDescription()`

108.15 FinishDatabaseQuery

Syntax

```
FinishDatabaseQuery(#Database)
```

Description

Finish the current database SQL query and release its associated resources. Query related functions like `FirstDatabaseRow()` or `NextDatabaseRow()` can't be used anymore.

Parameters

#Database The database to use.

Return value

None.

Example

```
1 ; First, connect to a database with an employee table
2 ;
3 If DatabaseQuery(#Database, "SELECT * FROM employee") ; Get all the
   records in the 'employee' table
4
5 While NextDatabaseRow(#Database) ; Loop for each records
6     Debug GetDatabaseString(#Database, 0) ; Display the content of
   the first field
7 Wend
8
9 FinishDatabaseQuery(#Database)
10 EndIf
```

See Also

DatabaseQuery()

108.16 FirstDatabaseRow

Syntax

```
Result = FirstDatabaseRow(#Database)
```

Description

Retrieves information about the first #Database row. The flag **#PB_Database_DynamicCursor** has to be specified to DatabaseQuery() to have this command working.

Parameters

#Database The database to use.

Return value

If Result is zero, then no row is available

Remarks

To access fields within a row, `GetDatabaseLong()` , `GetDatabaseFloat()` , `GetDatabaseString()` can be used.

See Also

`NextDatabaseRow()` , `PreviousDatabaseRow()` , `GetDatabaseLong()`

108.17 GetDatabaseBlob

Syntax

```
Result = GetDatabaseBlob(#Database, Column, *Buffer, BufferLength)
```

Description

Returns the content of the specified database column in the specified buffer as a pointer to the blob memory. This command is only valid after a successful `FirstDatabaseRow()` , `PreviousDatabaseRow()` or `NextDatabaseRow()` .

Parameters

#Database The database to use.

Column The column to use. `DatabaseColumnIndex()` is available to get the index of a named column.

***Buffer** The address of the blob data.

BufferLength The size of the blob data in bytes.

Return value

If 'Result' is 0, then the blob can't be retrieved or its content is empty.

Remarks

To determine the type of a column, `DatabaseColumnType()` can be used. To determine the size of the blob, `DatabaseColumnSize()` can be used.

Note: This function can be called only once for each column. Therefore if this value needs to be used more than once, the data has to be stored in a variable, since all subsequent calls will return the wrong value. This is an ODBC limitation.

See Also

`GetDatabaseDouble()` , `GetDatabaseFloat()` , `GetDatabaseLong()` , `GetDatabaseString()` , `GetDatabaseQuad()`

108.18 GetDatabaseDouble

Syntax

```
Result.d = GetDatabaseDouble(#Database , Column)
```

Description

Returns the content of the specified database column as a double precision floating-point number. This command is only valid after a successful FirstDatabaseRow() , PreviousDatabaseRow() or NextDatabaseRow() .

Parameters

#Database The database to use.

Column The column to use. DatabaseColumnIndex() is available to get the index of a named column.

Return value

Returns a double precision floating-point value.

Remarks

To determine the type of a column, DatabaseColumnType() can be used.

Note: This function can be called only once for each column. Therefore if this value needs to be used more than once, the data has to be stored in a variable, since all subsequent calls will return the wrong value. This is an ODBC limitation.

See Also

GetDatabaseBlob() , GetDatabaseFloat() , GetDatabaseLong() , GetDatabaseString() , GetDatabaseQuad()

108.19 GetDatabaseFloat

Syntax

```
Result.f = GetDatabaseFloat(#Database , Column)
```

Description

Returns the content of the specified database column as a floating-point number. This command is only valid after a successful FirstDatabaseRow() , PreviousDatabaseRow() or NextDatabaseRow() .

Parameters

#Database The database to use.

Column The column to use. DatabaseColumnIndex() is available to get the index of a named column.

Return value

Returns a single precision floating-point value.

Remarks

To determine the type of a column, DatabaseColumnType() can be used.

Note: This function can be called only once for each column. Therefore if this value needs to be used more than once, the data has to be stored in a variable, since all subsequent calls will return the wrong value. This is an ODBC limitation.

See Also

GetDatabaseBlob() , GetDatabaseDouble() , GetDatabaseLong() , GetDatabaseString() ,
GetDatabaseQuad()

108.20 GetDatabaseLong

Syntax

```
Result = GetDatabaseLong(#Database , Column)
```

Description

Returns the content of the specified #Database column as an integer number. This command is only valid after a successful FirstDatabaseRow() , PreviousDatabaseRow() or NextDatabaseRow() .

Parameters

#Database The database to use.

Column The column to use. DatabaseColumnIndex() is available to get the index of a named column.

Return value

Returns the content of the column as an integer value.

Remarks

To determine the type of a column, DatabaseColumnType() can be used.

Note: This function can be called only once for each column. Therefore if this value needs to be used more than once, the data has to be stored in a variable, since all subsequent calls will return the wrong value. This is an ODBC limitation.

See Also

GetDatabaseBlob() , GetDatabaseDouble() , GetDatabaseFloat() , GetDatabaseString() ,
GetDatabaseQuad()

108.21 GetDatabaseQuad

Syntax

```
Result.q = GetDatabaseQuad(#Database, Column)
```

Description

Returns the content of the specified #Database column as a quad number. This command is only valid after a successful FirstDatabaseRow() , PreviousDatabaseRow() or NextDatabaseRow() .

Parameters

#Database The database to use.

Column The column to use. DatabaseColumnIndex() is available to get the index of a named column.

Return value

Returns the content of the column as a quad value.

Remarks

To determine the type of a column, DatabaseColumnType() can be used.

Note: This function can be called only once for each column. Therefore if this value needs to be used more than once, the data has to be stored in a variable, since all subsequent calls will return the wrong value. This is an ODBC limitation.

See Also

GetDatabaseBlob() , GetDatabaseDouble() , GetDatabaseFloat() , GetDatabaseString() ,
GetDatabaseLong()

108.22 GetDatabaseString

Syntax

```
Text\$_ = GetDatabaseString(#Database, Column)
```

Description

Returns the content of the specified #Database column as a string. This command is only valid after a successful FirstDatabaseRow() , PreviousDatabaseRow() or NextDatabaseRow() .

Parameters

#Database The database to use.

Column The column to use. DatabaseColumnIndex() is available to get the index of a named column.

Return value

Returns the content of the column as a string.

Remarks

To determine the type of a column, DatabaseColumnType() can be used.

Note: This function can be called only once for each column. Therefore if this value needs to be used more than once, the data has to be stored in a variable, since all subsequent calls will return the wrong value. This is an ODBC limitation.

See Also

GetDatabaseBlob() , GetDatabaseDouble() , GetDatabaseFloat() , GetDatabaseLong() ,
GetDatabaseQuad()

108.23 CheckDatabaseNull

Syntax

```
Result = CheckDatabaseNull(#Database , Column)
```

Description

Checks if the content of the specified database column is null. This command is only valid after a successful FirstDatabaseRow() , PreviousDatabaseRow() or NextDatabaseRow() .

Parameters

#Database The database to use.

Column The column to use. DatabaseColumnIndex() is available to get the index of a named column.

Return value

Returns **#True** if the data is null, **#False** otherwise.

Remarks

To determine the type of a column, DatabaseColumnType() can be used.

Note: This function can be called only once for each column. Therefore if this value needs to be used more than once, the data has to be stored in a variable, since all subsequent calls will return the wrong value. This is an ODBC limitation.

See Also

GetDatabaseBlob() , GetDatabaseDouble() , GetDatabaseFloat() , GetDatabaseLong() ,
GetDatabaseQuad()

108.24 IsDatabase

Syntax

```
Result = IsDatabase (#Database)
```

Description

This function evaluates if the given #Database number is a valid and correctly-initialized database.

Parameters

#Database The database to use.

Return value

Returns nonzero if #Database is a valid database connection and zero otherwise.

Remarks

This function is bulletproof and can be used with any value. If Result is not zero then the object is valid and initialized, otherwise it returns zero. This is a good way to check that a database is ready to use.

See Also

OpenDatabase() , OpenDatabaseRequester()

108.25 NextDatabaseDriver

Syntax

```
Result = NextDatabaseDriver()
```

Description

Retrieves information about the next available database driver. This function must be called after ExamineDatabaseDrivers() . To get information about the current driver, DatabaseDriverName() and DatabaseDriverDescription() can be used.

Parameters

None.

Return value

If Result is 0, no more drivers are available.

Remarks

This is an ODBC database specific command.

See Also

ExamineDatabaseDrivers() , DatabaseDriverName() , DatabaseDriverDescription()

108.26 NextDatabaseRow

Syntax

```
Result = NextDatabaseRow(#Database)
```

Description

Retrieves information about the next database row in the #Database. To access fields within a row, GetDatabaseLong() , GetDatabaseFloat() , GetDatabaseString() can be used.

Parameters

#Database The database to use.

Return value

If Result is 0, then no more rows are available (i.e. reached the end of the table).

See Also

GetDatabaseBlob() , GetDatabaseDouble() , GetDatabaseFloat() , GetDatabaseLong() , GetDatabaseQuad() , GetDatabaseString()

108.27 OpenDatabase

Syntax

```
Result = OpenDatabase(#Database, DatabaseName$, User$, Password$ [,
    Plugin])
```

Description

Opens a new database connection.

Parameters

#Database A number to identify the new database. **#PB_Any** can be used to auto-generate this number.

DatabaseName\$ The name of the database to open.

User\$ The user name for the connection.

Password\$ The password for the connection. This can be an empty string if no password is required.

Plugin (optional) Specifies the database plug-in to use. It can be one of the following value:

```
#PB_Database_ODBC          : The database will use ODBC backend
    (UseODBCDatabase()
has to be called).
#PB_Database_SQLite        : The database will use SQLite backend
    (UseSQLiteDatabase()
has to be called).
#PB_Database_PostgreSQL    : The database will use PostgreSQL backend
    (UsePostgreSQLDatabase()
has to be called).
#PB_Database_MySQL         : The database will use MySQL backend
    (UseMySQLDatabase()
has to be called).
```

If 'Plugin' isn't specified, then the first registered database plug-in will be used.

Return value

Returns nonzero if the database connection was established successfully and zero if not. Error information can be received with the DatabaseError() command. If **#PB_Any** was used for the **#Database** parameter, then the generated number is returned.

See Also

OpenDatabaseRequester() , CloseDatabase() , UseODBCDatabase() , UseSQLiteDatabase() , UsePostgreSQLDatabase() , UseMySQLDatabase()

108.28 OpenDatabaseRequester

Syntax

```
Result = OpenDatabaseRequester(#Database [, Plugin])
```

Description

Open the standard ODBC requester to choose which database to open.

Parameters

#Database A number to identify the new database. **#PB_Any** can be used to auto-generate this number.

Plugin (optional) Specifies the database plug-in to use. It can be one of the following value:

```
#PB_Database_ODBC      : The database will use ODBC backend
(UseODBCDatabase()
has to be called).
```

If 'Plugin' isn't specified, then the first registered database plug-in will be used.

Return value

Returns nonzero if the database connection was established successfully and zero if not. Error information can be received with the `DatabaseError()` command. If **#PB_Any** was used for the **#Database** parameter, then the generated number is returned.

Remarks

This is an ODBC database specific command.

Note: This command is not supported on Linux and MacOS X and will return 0.

See Also

`OpenDatabase()` , `CloseDatabase()`

108.29 PreviousDatabaseRow

Syntax

```
Result = PreviousDatabaseRow(#Database)
```

Description

Retrieves information about the previous database row in the **#Database**. The flag **#PB_Database_DynamicCursor** has to be specified to `DatabaseQuery()` to have this command working. To access to fields inside a row, `GetDatabaseLong()` , `GetDatabaseFloat()` , `GetDatabaseString()` can be used.

Parameters

#Database The database to use.

Return value

If Result is 0, then no more rows are available (i.e. reached the start of the table).

Remarks

If this function returns zero despite additional rows being available before the current one, then the ODBC driver does not support data retrieval in a backwards direction. It is not mandatory for an ODBC driver to support this function (unlike `NextDatabaseRow()`). Of course, if this function works, it will work on every computer using the same driver. SQLite databases don't support this command.

See Also

`GetDatabaseBlob()`, `GetDatabaseDouble()`, `GetDatabaseFloat()`, `GetDatabaseLong()`, `GetDatabaseQuad()`, `GetDatabaseString()`

108.30 SetDatabaseBlob

Syntax

```
SetDatabaseBlob(#Database, StatementIndex, *Buffer, BufferLength)
```

Description

Set the blob for future use with `DatabaseUpdate()`.

Parameters

#Database The database to use.

StatementIndex Undefined query parameter index the blob should be inserted for. The first undefined parameter index starts from zero. The SQL syntax to specify undefined parameter is database manager dependent. See the following examples to see how to proceed.

***Buffer** The address of the blob data.

BufferLength The size of the blob data in bytes.

Return value

None.

Example: SQLite, MySQL and ODBC

```
1 ; SQLite, MySQL and ODBC shares the same syntax to insert blob. It is
   indicated by the '?' character
2 ;
3 ; The database should be opened and a table PHOTOS with 3 column
   (BLOB, VARCHAR(255), BLOB)
4 ;
5 SetDatabaseBlob(0, 0, ?Picture, PictureLength)
6 SetDatabaseBlob(0, 1, ?SmallPicture, SmallPictureLength)
7 DatabaseUpdate(0, "INSERT INTO PHOTOS (picture, name, small_picture)
   values (?, 'my description', ?);")
```

Example: PostgreSQL

```
1 ; PostgreSQL uses another syntax: $1, $2.. into the statement to
   indicate the undefined parameter
2 ;
3 ; The database should be opened and a table PHOTOS with 3 column
   (BYTEA, VARCHAR(255), BYTEA)
4 ;
5 SetDatabaseBlob(0, 0, ?Picture, PictureLength)
6 SetDatabaseBlob(0, 1, ?SmallPicture, SmallPictureLength)
7 DatabaseUpdate(0, "INSERT INTO PHOTOS (picture, name, small_picture)
   values ($1, 'my description', $2);")
```

Note: PostgreSQL uses BYTEA to store large objects. The escaping needed to store binary data into such a column make it often bigger than expected. A good way to store binary data is to encode it with Base64Encoder() before submitting to the database manager.

See Also

DatabaseUpdate() , GetDatabaseBlob()

108.31 UseMySQLDatabase

Syntax

```
Result = UseMySQLDatabase([LibraryName$])
```

Description

Initialize the MySQL and MariaDB database environment for future use.

Parameters

LibraryName\$ (optional) Filename (and path if needed) of the dynamic library to use. As most Linux distribution ship with packaged libmysql.so, it can be set to the correct name, so the libmaria.so doesn't have to be package with the executable. If this parameter is not specified, 'libmariadb.dll' (Windows), 'libmariadb.so' (Linux) or 'libmariadb.dylib' (OSX) will be used.

Return value

Returns **#True** if the MySQL environment is successfully initialized, otherwise returns **#False**.

Remarks

MySQL and MariaDB (an opensource fork of MySQL) are powerful, server based database managers which support very large database and high concurrency. PureBasic uses opensource MariaDB library to connect MySQL and MariaDB databases seamlessly, which can be used in commercial application without additional licenses. When shipping your PureBasic program, you will need to add 'libmariadb.dll' (Windows), 'libmariadb.so' (Linux) or 'libmariadb.dylib' (OSX) found in the 'PureBasic/Compilers' directory to your package.

There is no additional driver to install, all is ready to connect a MySQL or MariaDB server. For more information about MariaDB: <https://mariadb.org/>.

A MySQL or MariaDB database has to be connected using `OpenDatabase()` before using any other database functions. MySQL specific parameters have to be passed in the 'DatabaseName\$' parameter of `OpenDatabase()` :

- `host`: Name of host or IP address to connect to.
- `port`: Port number to connect to at the server host.
- `dbname`: The database name. The name can be surrounded with single quotes if it contains spaces: `dbname='movies and musics'`.

Example

```
1 UseMySQLDatabase()
2
3 ; You should have a server running on localhost
4 ;
5 If OpenDatabase(0, "host=localhost port=3306 dbname='test'", "user",
6   "password")
7   Debug "Connected to MySQL"
8 Else
9   Debug "Connection failed: "+DatabaseError()
EndIf
```

See Also

`OpenDatabase()` , `UseSQLiteDatabase()` , `UseODBCDatabase()` , `UsePostgreSQLDatabase()`

108.32 UsePostgreSQLDatabase

Syntax

```
Result = UsePostgreSQLDatabase([LibraryName$])
```

Description

Initialize the PostgreSQL database environment for future use.

Parameters

LibraryName\$ (optional) Filename (and path if needed) of the dynamic library to use. It could be useful to use a different version of the library than the one included with PureBasic. If this parameter is not specified, the internal postgresql client library shipped with PureBasic will be used (no external dynamic library needed).

Return value

When using the optional 'LibraryName\$' parameter, it will return `#True` if the dynamic library has been correctly loaded or `#False` otherwise. If the optional 'LibraryName\$' parameter is not specified, it will always return `#True`.

Remarks

PostgreSQL is a powerful, server based database manager which support very large database and high concurrency. It is free to use in commercial projects, unlike MySQL which requires a licence to use it in a non-GPL program. There is no additional driver to install, all is ready to connect a PostgreSQL server. For more information about PostgreSQL: <http://www.postgresql.org>.

A PostgreSQL database has to be connected using OpenDatabase() before using any other database functions. PostgreSQL specific parameters can be passed in the 'DatabaseName\$' parameter of OpenDatabase() :

- host: Name of host to connect to.
- hostaddr: Numeric IP address of host to connect to.
- port: Port number to connect to at the server host.
- dbname: The database name. Defaults to be the same as the user name.
- connect_timeout: Maximum wait for connection, in seconds (write as a decimal integer string).
Zero or not specified means wait indefinitely.
It is not recommended to use a timeout of less than 2 seconds.

Example

```
1  UsePostgreSQLDatabase()  
2  
3  ; You should have a server running on localhost  
4  ;  
5  If OpenDatabase(0, "host=localhost port=5432", "user", "password")  
6      Debug "Connected to PostgreSQL"  
7  Else  
8      Debug "Connection failed: "+DatabaseError()  
9  EndIf
```

See Also

OpenDatabase() , UseSQLiteDatabase() , UseODBCDatabase() , UseMySQLDatabase()

108.33 UseSQLiteDatabase

Syntax

```
Result = UseSQLiteDatabase([LibraryName$])
```

Description

Initialize the SQLite database environment for future use.

Parameters

LibraryName\$ (optional) Filename (and path if needed) of the dynamic library to use.

You will be able to use the latest version of the 'dll' file (so, dylib) without waiting for a PB update.

Without this file, the static library will be used as usual and the executable will be larger.

Return value

When using the optional 'LibraryName\$' parameter, it will return `#True` if the dynamic library has been correctly loaded or `#False` otherwise. If the optional 'LibraryName\$' parameter is not specified, it will always return `#True`.

Remarks

SQLite is a file based, serverless database manager. There is no driver or additional files to install, all is ready to use. SQLite is widely spread across the industry and is considered to be one of the best embedded database manager available. For more information about SQLite: <http://www.sqlite.org>.

To create a new empty database, create a new file with `CreateFile()`. Database commands can now be used to create tables and add records.

A SQLite database has to be opened using `OpenDatabase()` before using any other database functions.

Example

```
1  UseSQLiteDatabase()
2
3  Filename$ = OpenFileDialog("Choose a file name",
4      "PureBasic.sqlite", "*.sqlite|*.sqlite", 0)
5
6  If CreateFile(0, Filename$)
7      Debug "Database file created"
8      CloseFile(0)
9  EndIf
10
11 If OpenDatabase(0, Filename$, "", "")
12     Debug "Connected to PureBasic.sqlite"
13     If DatabaseUpdate(0, "CREATE TABLE info (test VARCHAR(255));")
14         Debug "Table created"
15     EndIf
16 EndIf
```

See Also

`OpenDatabase()` , `UsePostgreSQLDatabase()` , `UseODBCDatabase()` , `UseMySQLDatabase()`

108.34 UseODBCDatabase

Syntax

```
Result = UseODBCDatabase()
```

Description

Initialize the ODBC database environment for future use. It attempts to load the ODBC driver and allocate the required resources.

Parameters

None.

Return value

If Result is 0, then the ODBC driver is not available or is too old (ODBC 3.0 or higher is needed) and the Database functions should not be used.

Remarks

After calling UseODBCDatabase(), a database has to be opened using OpenDatabase() using a registered ODBC database name as database or OpenDatabaseRequester() before using any other Database functions.

It is possible to obtain a list of available drivers by calling the function ExamineDatabaseDrivers() .

Example

```
1  UseODBCDatabase ()
2
3  If OpenDatabase (0, "MySQL-ODBC", "user", "password")
4      Debug "Connected to MySQL"
5  Else
6      Debug "Connection failed: "+DatabaseError ()
7  EndIf
```

See Also

OpenDatabase() , UseSQLiteDatabase() , UsePostgreSQLDatabase() , UseMySQLDatabase()

108.35 SetDatabaseString

Syntax

```
SetDatabaseString (#Database , StatementIndex , Value$)
```

Description

Set a string as a bind variable for the next call to DatabaseQuery() or DatabaseUpdate() .

Parameters

#Database The database to use.

StatementIndex The index of the bind variable within the statement. The first variable has index 0.

Value\$ The value to use for the bind variable.

Return value

None.

Remarks

Bind variables make constructing statements with variable data easier, because there is no need to add the data into the string. The statement string can contain the placeholders and the data is bound before executing the statement. This method also avoids vulnerabilities due to possible SQL injection which can be done if data (such as strings) is directly inserted in the statement text. Since the statement only contains the placeholder, there is no danger.

See `DatabaseQuery()` and `DatabaseUpdate()` for examples how to specify bind variables in an SQL statement.

See Also

`SetDatabaseLong()` , `SetDatabaseQuad()` , `SetDatabaseFloat()` , `SetDatabaseDouble()`
`SetDatabaseBlob()` , `SetDatabaseNull()` , `DatabaseQuery()` , `DatabaseUpdate()`

108.36 SetDatabaseLong

Syntax

```
SetDatabaseLong(#Database , StatementIndex , Value)
```

Description

Set a long value as a bind variable for the next call to `DatabaseQuery()` or `DatabaseUpdate()` .

Parameters

#Database The database to use.

StatementIndex The index of the bind variable within the statement. The first variable has index 0.

Value The value to use for the bind variable.

Return value

None.

Remarks

Bind variables make constructing statements with variable data easier, because there is no need to add the data into the string. The statement string can contain the placeholders and the data is bound before executing the statement. This method also avoids vulnerabilities due to possible SQL injection which can be done if data (such as strings) is directly inserted in the statement text. Since the statement only contains the placeholder, there is no danger.

See `DatabaseQuery()` and `DatabaseUpdate()` for examples how to specify bind variables in an SQL statement.

See Also

SetDatabaseString() , SetDatabaseQuad() , SetDatabaseFloat() , SetDatabaseDouble()
SetDatabaseBlob() , SetDatabaseNull() , DatabaseQuery() , DatabaseUpdate()

108.37 SetDatabaseQuad

Syntax

```
SetDatabaseQuad(#Database , StatementIndex , Value.q)
```

Description

Set a quad value as a bind variable for the next call to DatabaseQuery() or DatabaseUpdate() .

Parameters

#Database The database to use.

StatementIndex The index of the bind variable within the statement. The first variable has index 0.

Value.q The value to use for the bind variable.

Return value

None.

Remarks

Bind variables make constructing statements with variable data easier, because there is no need to add the data into the string. The statement string can contain the placeholders and the data is bound before executing the statement. This method also avoids vulnerabilities due to possible SQL injection which can be done if data (such as strings) is directly inserted in the statement text. Since the statement only contains the placeholder, there is no danger.

See DatabaseQuery() and DatabaseUpdate() for examples how to specify bind variables in an SQL statement.

See Also

SetDatabaseString() , SetDatabaseLong() , SetDatabaseFloat() , SetDatabaseDouble()
SetDatabaseBlob() , SetDatabaseNull() , DatabaseQuery() , DatabaseUpdate()

108.38 SetDatabaseFloat

Syntax

```
SetDatabaseFloat(#Database , StatementIndex , Value.f)
```

Description

Set a float as a bind variable for the next call to DatabaseQuery() or DatabaseUpdate() .

Parameters

#Database The database to use.

StatementIndex The index of the bind variable within the statement. The first variable has index 0.

Value.f The value to use for the bind variable.

Return value

None.

Remarks

Bind variables make constructing statements with variable data easier, because there is no need to add the data into the string. The statement string can contain the placeholders and the data is bound before executing the statement. This method also avoids vulnerabilities due to possible SQL injection which can be done if data (such as strings) is directly inserted in the statement text. Since the statement only contains the placeholder, there is no danger.

See DatabaseQuery() and DatabaseUpdate() for examples how to specify bind variables in an SQL statement.

See Also

SetDatabaseString() , SetDatabaseLong() , SetDatabaseQuad() , SetDatabaseDouble()
SetDatabaseBlob() , SetDatabaseNull() , DatabaseQuery() , DatabaseUpdate()

108.39 SetDatabaseDouble

Syntax

```
SetDatabaseDouble( #Database , StatementIndex , Value.d )
```

Description

Set a double value as a bind variable for the next call to DatabaseQuery() or DatabaseUpdate() .

Parameters

#Database The database to use.

StatementIndex The index of the bind variable within the statement. The first variable has index 0.

Value.d The value to use for the bind variable.

Return value

None.

Remarks

Bind variables make constructing statements with variable data easier, because there is no need to add the data into the string. The statement string can contain the placeholders and the data is bound before executing the statement. This method also avoids vulnerabilities due to possible SQL injection which can be done if data (such as strings) is directly inserted in the statement text. Since the statement only contains the placeholder, there is no danger.

See `DatabaseQuery()` and `DatabaseUpdate()` for examples how to specify bind variables in an SQL statement.

See Also

`SetDatabaseString()` , `SetDatabaseLong()` , `SetDatabaseQuad()` , `SetDatabaseFloat()` `SetDatabaseBlob()` , `SetDatabaseNull()` , `DatabaseQuery()` , `DatabaseUpdate()`

108.40 SetDatabaseNull

Syntax

```
SetDatabaseNull(#Database, StatementIndex)
```

Description

Set a bind variable to a NULL value for the next call to `DatabaseQuery()` or `DatabaseUpdate()` .

Parameters

#Database The database to use.

StatementIndex The index of the bind variable within the statement. The first variable has index 0.

Return value

None.

Remarks

See `DatabaseQuery()` and `DatabaseUpdate()` for examples how to specify bind variables in an SQL statement.

See Also

`SetDatabaseString()` , `SetDatabaseLong()` , `SetDatabaseQuad()` , `SetDatabaseFloat()` , `SetDatabaseDouble()` `SetDatabaseBlob()` , `DatabaseQuery()` , `DatabaseUpdate()`

; ; Date library documentation ; ; (c) 2020 - Fantaisie Software ;

Chapter 109

Date

Overview

The Date library allows for the manipulation of Date and Time from the year 1601 up to 9999 using the Unix method (i.e. the number of seconds elapsed since the 1st of January 1970). Dates before the year 1970 are represented as negative values.

Note: In order to use the full available date range, values returned by this library must be stored in Quad type variables (.q). Using a 32bit data type (Long or Integer on 32bit systems) results in invalid values starting with the year 2038.

109.1 AddDate

Syntax

```
Date.q = AddDate(Date.q, Type, Value)
```

Description

Add an amount of time to a date.

Parameters

Date.q The date value to which the value should be added.

Type The value type. It can be one of the following constants:

```
#PB_Date_Year      : Will add 'Value' Years to the date
#PB_Date_Month     : Will add 'Value' Months to the date
#PB_Date_Week      : Will add 'Value' Weeks to the date
#PB_Date_Day       : Will add 'Value' Days to the date
#PB_Date_Hour      : Will add 'Value' Hours to the date
#PB_Date_Minute    : Will add 'Value' Minutes to the date
#PB_Date_Second    : Will add 'Value' Seconds to the date
```

Note: when `#PB_Date_Month` is used, it will automatically account for the fact that the numbers of days per month varies, for example: if a month is added to '31 march 2008' the result will be '30 april 2008', since april does not have 31 days.

Value The value to add to the date. A negative value can be used to subtract a date.

Return value

Returns the new date, -1 otherwise.

Example

```
1
2  Debug FormatDate("%yyyy/%mm/%dd", AddDate(Date(), #PB_Date_Year, 2))
   ; Returns the current date + 2 years
3
4  Debug FormatDate("%mm/%dd/%yyyy", AddDate(Date(), #PB_Date_Year, 2))
   ; Returns the current date + 2 years
```

See Also

Date() , FormatDate()

109.2 ConvertDate

Syntax

```
Date.q = ConvertDate(Date.q, Format)
```

Description

Converts a date between local time and UTC time.

Parameters

Date.q The date to convert.

Format The new date format. It can be one of the following constants:

```
#PB_Date_LocalTime: Will convert an UTC date into local time date.
#PB_Date_UTC       : Will convert a local time date into an UTC
                    date.
```

Return value

Returns converted date, or -1 if the conversion failed.

Example

```
1  Debug FormatDate("Local time: %hh:%ii:%ss", ConvertDate(DateUTC(),
   #PB_Date_LocalTime))
2  Debug FormatDate("  UTC time: %hh:%ii:%ss", ConvertDate(Date(),
   #PB_Date_UTC))
```

See Also

`FormatDate()` , `Date()` , `DateUTC()`

109.3 Date

Syntax

```
Date.q = Date([Year, Month, Day, Hour, Minute, Second])
```

Description

Returns the date value created from the given parameters, or the local system time if no parameters are specified.

Parameters

Year, Month, Day, Hour, Minute, Second (optional) The components used to create the date. If these parameters are not specified, the local system time is returned. The allowed range for the Year value is 1601 to 9999.

Return value

Returns the specified date value. If the given parameters are not valid (or outside the supported date range) then -1 will be returned.

Example

```
1  Debug Date() / (3600*24*365) ; will print the number of years since
   01/01/1970 and now
2  Debug Date(1999, 12, 31, 23, 59, 59) ; will print '946684799'
   (number of seconds between 01/01/1970 0:00:00 and 12/31/1999
   23:59:59)
```

See Also

`FormatDate()` , `DateUTC()` `Year()` , `Month()` , `Day()` , `Hour()` , `Minute()` , `Second()`

109.4 DateUTC

Syntax

```
Date.q = DateUTC()
```

Description

Returns the system date as UTC time.

Return value

Returns the system date as UTC time.

Example

```
1  Debug FormatDate("UTC time: %hh:%ii:%ss", DateUTC())
```

See Also

FormatDate() , Date()

109.5 Day

Syntax

```
Result = Day(Date.q)
```

Description

Returns the day component of the specified date.

Parameters

Date.q The date value from which to extract the day.

Return value

Returns the day component, -1 otherwise. The result is always between 1 and 31.

Example

```
1  Debug Day(Date(2002, 10, 3, 0, 0, 0)) ; Outputs '3'.
```

See Also

Date() , Year() , Month() , Hour() , Minute() , Second()

109.6 DayOfWeek

Syntax

```
Result = DayOfWeek(Date.q)
```


Description

Returns the weekday of the specified date.

Parameters

Date.q The date value from which to extract the weekday.

Return value

Returns a number between 0 and 6 representing the day of the week, -1 otherwise:

```
0 : Sunday
1 : Monday
2 : Tuesday
3 : Wednesday
4 : Thursday
5 : Friday
6 : Saturday
```

Example

```
1  Debug DayOfWeek(Date(2006, 10, 30, 0, 0, 0)) ; Outputs '1' for
    Monday.
```

See Also

FormatDate() , Year() , Month() , Day() , Hour() , Minute() , Second()

109.7 DayOfYear

Syntax

```
Result = DayOfYear(Date.q)
```

Description

Returns the number of days elapsed since the beginning of the year of the specified date.

Parameters

Date.q The date value from which to extract the number of days.

Return value

Returns the number of days since the beginning of the year, -1 otherwise. The result is always between 1 and 366.

Example

```
1  Debug DayOfYear(Date(2002, 2, 1, 0, 0, 0)) ; Outputs '32'. (31 days
    for January + 1)
```

See Also

FormatDate() , Year() , Month() , Day() , Hour() , Minute() , Second()

109.8 Month

Syntax

```
Result = Month(Date.q)
```

Description

Returns the month value of the specified date.

Parameters

Date.q The date from which to extract the month.

Return value

Returns the month component of the date, -1 otherwise. The result is always between 1 and 12.

Example

```
1  Debug Month(Date(2002, 10, 3, 0, 0, 0)) ; Outputs '10'.
```

See Also

FormatDate() , Year() , Day() , Hour() , Minute() , Second()

109.9 Year

Syntax

```
Result = Year(Date.q)
```

Description

Returns the year value of the specified date.

Parameters

Date.q The date from which to extract the year.

Return value

Returns the year component of the date, -1 otherwise.

Example

```
1  Debug Year(Date(2002, 10, 3, 0, 0, 0)) ; Outputs '2002'.
```

See Also

FormatDate() , Month() , Day() , Hour() , Minute() , Second()

109.10 Hour

Syntax

```
Result = Hour(Date.q)
```

Description

Returns the hour value of the specified date.

Parameters

Date.q The date from which to extract the hour.

Return value

Returns the hour component of the date, -1 otherwise. The result is always between 0 and 23.

Example

```
1  Debug Hour(Date(1970, 1, 1, 11, 3, 45)) ; Outputs '11'.
```

See Also

FormatDate() , Year() , Month() , Day() , Minute() , Second()

109.11 Minute

Syntax

```
Result = Minute(Date.q)
```

Description

Returns the minute value of the specified date.

Parameters

Date.q The date from which to extract the minutes.

Return value

Returns the minute component of the date, -1 otherwise. The result is always between 0 and 59.

Example

```
1  Debug Minute(Date(1970, 1, 1, 11, 3, 45)) ; Outputs '3'.
```

See Also

FormatDate() , Year() , Month() , Day() , Hour() , Second()

109.12 Second

Syntax

```
Result = Second(Date.q)
```

Description

Returns the second value of the specified date.

Parameters

Date.q The date from which to extract the seconds.

Return value

Returns the second component of the date, -1 otherwise. The result is always between 0 and 59.

Example

```
1  Debug Second(Date(1970, 1, 1, 11, 3, 45)) ; Outputs '45'.
```

See Also

FormatDate() , Year() , Month() , Day() , Hour() , Minute()

109.13 FormatDate

Syntax

```
Text\$ = FormatDate(Mask$, Date.q)
```

Description

Returns a string representation of the given Date.

Parameters

Mask\$ The mask used to format the date. The following tokens in the mask string will be replaced according to the given date:

- %yyyy: Will be replaced by the **year** value, on 4 digits.
- %yy: Will be replaced by the **year** value, on 2 digits.
- %mm: Will be replaced by the **month** value, on 2 digits.
- %dd: Will be replaced by the **day** value, on 2 digits.
- %hh: Will be replaced by the **hour** value, on 2 digits.
- %ii: Will be replaced by the **minute** value, on 2 digits.
- %ss: Will be replaced by the **second** value, on 2 digits.

Date.q The date value to use.

Return value

Returns the mask string with all tokens replaced by the date values if the date was valid, an empty string otherwise.

Example

```
1
2  Debug FormatDate("Y=%yyyy, M= %mm, D=%dd", Date()) ; Will display the
   actual date with
3
   ; the form
   "Y=2010, M=01, D=07"
4
5  Debug FormatDate("%mm/%dd/%yyyy", Date()) ; Will display the actual
   date with
6
   ; the form "01/07/2010"
7
```

```
8 | Debug FormatDate("%hh:%ii:%ss", Date()) ; Will display the time
    using the 00:00:00 format
```

See Also

Date() , ParseDate()

109.14 ParseDate

Syntax

```
Date.q = ParseDate(Mask$, String$)
```

Description

Transforms a string date into a regular date value which then can be used by other date functions.

Parameters

Mask\$ A mask string which defines how the date string is formatted. Possible tokens are:

- %yyyy: Will be replaced by the **year** value, on 4 digits.
- %yy: Will be replaced by the **year** value, on 2 digits.
- %mm: Will be replaced by the **month** value, on 2 digits.
- %dd: Will be replaced by the **day** value, on 2 digits.
- %hh: Will be replaced by the **hour** value, on 2 digits.
- %ii: Will be replaced by the **minute** value, on 2 digits.
- %ss: Will be replaced by the **second** value, on 2 digits.

String\$ The string with the date to be parsed.

Return value

Returns the date representing the parsed string. If the input string did not match the mask then the result is -1.

Example

```
1 |
2 | Debug ParseDate("%yy/%mm/%dd", "10/01/07") ; Returns the date
    value of "10/01/07"
3 |
4 | Debug ParseDate("%mm/%dd/%yyyy", "01/07/2010") ; Returns the date
    value of "01/07/2010"
```

See Also

Date() , FormatDate()

Chapter 110

Debugger

Overview

The Debugger library provides functions for controlling the debugger , for example to empty the debug output window or to open the memory viewer with a specific memory area to display. All these debugger tools are described in the debugger tools chapter.

The functions in this library are only compiled into the executable if the debugger is enabled on compilation. If the debugger is disabled then the entire call to these functions will be ignored.

There are also a number of special keywords to control the debugger from code.

110.1 CopyDebugOutput

Syntax

```
CopyDebugOutput()
```

Description

Copy the debug output window content to the clipboard.

Parameters

None.

Return value

None.

See Also

Included debugging tools , Debug , ShowDebugOutput() , ClearDebugOutput() , SaveDebugOutput()

110.2 ShowDebugOutput

Syntax

```
ShowDebugOutput()
```

Description

Open the debug output window or bring it to the front if it is already open.

Parameters

None.

Return value

None.

See Also

Included debugging tools , Debug , ClearDebugOutput() , SaveDebugOutput() , CopyDebugOutput() , CloseDebugOutput()

110.3 CloseDebugOutput

Syntax

```
CloseDebugOutput()
```

Description

Close the debug output .

Parameters

None.

Return value

None.

See Also

Included debugging tools , Debug , CopyDebugOutput() , ShowDebugOutput() , ClearDebugOutput() , SaveDebugOutput()

110.4 ClearDebugOutput

Syntax

```
ClearDebugOutput()
```

Description

Clear the content of the debug output window.

Parameters

None.

Return value

None.

Example

```
1 ; Show 10 debug values only, not a continuous list
2 Repeat
3     ClearDebugOutput()
4     For i = 1 To 10
5         Debug x
6         x + 1
7     Next i
8
9     Delay(500)
10 ForEver
```

See Also

Included debugging tools , Debug , ShowDebugOutput() , SaveDebugOutput() , CopyDebugOutput()

110.5 DebuggerError

Syntax

```
DebuggerError(Message$)
```

Description

Generates a runtime debugger error. The program execution will be stopped if the debugger is activated. Can be useful when creating reusable modules meant to be shared.

Parameters

Message\$ The error message to display.

Return value

None.

See Also

DebuggerWarning()

110.6 DebuggerWarning

Syntax

```
DebuggerWarning(Message$)
```

Description

Generates a runtime debugger warning. Can be useful when creating reusable modules meant to be shared.

Parameters

Message\$ The warning message to display.

Return value

None.

See Also

DebuggerError()

110.7 SaveDebugOutput

Syntax

```
SaveDebugOutput(Filename$)
```

Description

Save the content of the debug output window to the given file.

Parameters

Filename\$ The filename to save the output to.

Return value

None.

Remarks

An error occurs if the file cannot be saved.

Example

```
1 For i = 1 To 100
2     Debug Random(i)
3 Next i
4 SaveDebugOutput("C:\log.txt")
```

See Also

Included debugging tools , Debug , ShowDebugOutput() , ClearDebugOutput() , CopyDebugOutput()

110.8 ShowProfiler

Syntax

```
ShowProfiler()
```

Description

Open the profiler window or bring it to the front if it is already open.

Parameters

None.

Return value

None.

See Also

Included debugging tools , ResetProfiler() , StartProfiler() , StopProfiler()

110.9 ResetProfiler

Syntax

```
ResetProfiler()
```

Description

Reset the line counters for the profiler.

Parameters

None.

Return value

None.

See Also

Included debugging tools , ShowProfiler() , StartProfiler() , StopProfiler()

110.10 StartProfiler

Syntax

```
StartProfiler()
```

Description

Start the counting of executed lines by the profiler.

Parameters

None.

Return value

None.

See Also

Included debugging tools , ShowProfiler() , ResetProfiler() , StopProfiler()

110.11 StopProfiler

Syntax

```
StopProfiler()
```

Description

Stop the counting of executed lines by the profiler.

Parameters

None.

Return value

None.

See Also

Included debugging tools , ShowProfiler() , ResetProfiler() , StartProfiler()

110.12 ShowMemoryViewer

Syntax

```
ShowMemoryViewer([*Buffer, Length])
```

Description

Open the memory viewer window or bring it to the front if it is already open.

Parameters

***Buffer, Length (optional)** The memory area to be displayed in the memory viewer. If these parameters are not specified, then the memory viewer will just be opened without displaying a specific memory area.

Return value

None.

Example

```

1  *Memory = AllocateMemory(1000)
2  If *Memory
3      RandomData(*Memory, 1000)           ; Fill memory with some data
4
5      ShowMemoryViewer(*Memory, 1000) ; Open memory viewer
6      CallDebugger                       ; Halt the program, so it does not end
7      right away
8  EndIf

```

See Also

Included debugging tools

110.13 ShowLibraryViewer

Syntax

```
ShowLibraryViewer([Library$ [, #Object]])
```

Description

Open the library viewer window or bring it to the front if it is already open. If Library\$ is specified then the viewer will show the objects of that library. If an #Object number is specified, then the viewer will display the specified object of that library.

Parameters

Library\$ (optional) The library that should be displayed in the viewer. If this parameter is not specified then the viewer will be opened without displaying a specific library.

#Object (optional) An object of the library to display in the viewer. If this parameter is not specified, then no object is displayed.

Return value

None.

Example

```

1  If CreateImage(0, 200, 200) And StartDrawing(ImageOutput(0))
2      DrawingMode(#PB_2DDrawing_Transparent)
3      Box(0, 0, 200, 200, RGB(255, 255, 255))
4      For i = 1 To 30
5          DrawText(Random(200), Random(200), "Hello World!",
6          RGB(Random(255), Random(255), Random(255)))
7      Next i
8      StopDrawing()
9      ShowLibraryViewer("Image", 0) ; Show the image

```

10	<code>CallDebugger</code>	<code>; Halt the program so it does not end</code>
	<code>right away</code>	
11	<code>EndIf</code>	

See Also

Included debugging tools

110.14 ShowWatchlist

Syntax

`ShowWatchlist()`

Description

Open the watchlist window or bring it to the front if it is already open.

Parameters

None.

Return value

None.

See Also

Included debugging tools

110.15 ShowVariableViewer

Syntax

`ShowVariableViewer()`

Description

Open the variable viewer window or bring it to the front if it is already open.

Parameters

None.

Return value

None.

See Also

Included debugging tools

110.16 ShowCallstack

Syntax

```
ShowCallstack()
```

Description

Open the callstack window or bring it to the front if it is already open.

Parameters

None.

Return value

None.

See Also

Included debugging tools

110.17 ShowAssemblyViewer

Syntax

```
ShowAssemblyViewer()
```

Description

Open the assembly viewer window or bring it to the front if it is already open.

Parameters

None.

Return value

None.

See Also

Included debugging tools

110.18 PurifierGranularity

Syntax

```
PurifierGranularity(GlobalGranularity, LocalGranularity,  
    StringGranularity, DynamicGranularity)
```

Description

Change the interval in which the purifier checks the different areas for memory corruption.

Parameters

GlobalGranularity The number of source lines to execute between checks on global variables. Using `#PB_Ignore` will keep the existing interval value. A value of 0 disables the check.

LocalGranularity The number of source lines to execute between checks on local variables. Using `#PB_Ignore` will keep the existing interval value. A value of 0 disables the check.

StringGranularity The number of source lines to execute between checks on string memory. Using `#PB_Ignore` will keep the existing interval value. A value of 0 disables the check.

DynamicGranularity The number of source lines to execute between checks on allocated memory. Using `#PB_Ignore` will keep the existing interval value. A value of 0 disables the check.

Return value

None.

Example

```
1 ; Disable check for string memory and check allocated memory every 10  
  lines  
2 PurifierGranularity(#PB_Ignore, #PB_Ignore, 0, 10)
```

See Also

Included debugging tools

Chapter 111

Desktop

Overview

The desktop library allows access to information about the user's desktop environment, such as screen width, height, depth, mouse position etc.

111.1 ExamineDesktops

Syntax

```
Result = ExamineDesktops()
```

Description

Retrieves information about all the desktops connected to the local computer. This function must be called before using the functions of this library the following functions: DesktopDepth() , DesktopFrequency() , DesktopHeight() , DesktopName() and DesktopWidth() .

Parameters

None.

Return value

The number of desktops on success, zero otherwise.

Remarks

Typically, a standard user has only one desktop, but multi-screen users may have several desktops, each with their own resolution.

Example

```
1  MessageRequester("Desktop Information", "You have  
   "+Str(ExamineDesktops())+" desktops")
```

See Also

DesktopDepth() , DesktopFrequency() , DesktopHeight() , DesktopName() , DesktopWidth()

111.2 DesktopDepth

Syntax

```
Result = DesktopDepth(#Desktop)
```

Description

Returns the color depth of the specified desktop.

Parameters

#Desktop The index of the desktop. The first index always specifies the primary monitor. The first index value is zero.

Return value

Returns the depth in bits-per-pixel: 1, 2, 4, 8, 15, 16, 24 or 32-bit

Remarks

ExamineDesktops() must be called before using this function to retrieve information about the available desktops.

Example

```
1  ExamineDesktops()  
2  MessageRequester("Display Information", "Current resolution =  
   "+Str(DesktopWidth(0))+"x"+Str(DesktopHeight(0))+"x"+Str(DesktopDepth(0)))
```

See Also

ExamineDesktops() , DesktopFrequency() , DesktopHeight() , DesktopName() , DesktopWidth()

111.3 DesktopResolutionX

Syntax

```
Result.d = DesktopResolutionX()
```

Description

Returns the desktop DPI resolution factor on the 'x' axis.

Parameters

None.

Return value

Returns the desktop DPI resolution factor on the 'x' axis. If the value is '1', then no DPI factor has been applied to the display on the 'x' axis. If the value is '1.25', a 125% factor has been applied to the display on the 'x' axis.

Remarks

The application needs to be compiled with the 'DPI Aware' switch to have this command returning real DPI resolution factor. If not, the result will be always be '1'.

Example

```
1  Debug "Desktop DPI 'x' factor: " + DesktopResolutionX()
```

See Also

DesktopResolutionY() , DesktopScaledX() , DesktopScaledY() , DesktopUnscaledX() , DesktopUnscaledY()

111.4 DesktopResolutionY

Syntax

```
Result.d = DesktopResolutionY()
```

Description

Returns the desktop DPI resolution factor on the 'y' axis.

Parameters

None.

Return value

Returns the desktop DPI resolution factor on the 'y' axis. If the value is '1', then no DPI factor has been applied to the display on the 'y' axis. If the value is '1.25', a 125% factor has been applied to the display on the 'y' axis.

Remarks

The application needs to be compiled with the 'DPI Aware' switch to have this command returning real DPI resolution factor. If not, the result will be always be '1'.

Example

```
1  Debug "Desktop DPI 'y' factor: " + DesktopResolutionY()
```

See Also

DesktopResolutionX() , DesktopScaledX() , DesktopScaledY() , DesktopUnscaledX() , DesktopUnscaledY()

111.5 DesktopScaledX

Syntax

```
Result = DesktopScaledX(Value)
```

Description

Returns the scaled value according to current display DPI on 'x' axis. This is mostly useful to calculate real pixel position independently of the display DPI.

Parameters

Value The value to use.

Return value

Returns the scaled value according to current display DPI on 'x' axis. For example, on a display with a 125% DPI applied, a value of 100 will result to 125.

Remarks

The application needs to be compiled with the 'DPI Aware' switch to have this command returning scaled DPI value. If not, the result will be always be the same as 'Value' parameter.

Example

```
1  Debug "Desktop DPI 'x' scaled value of 100: " + DesktopScaledX(100)
```

See Also

DesktopResolutionX() , DesktopResolutionY() , DesktopScaledY() , DesktopUnscaledX() , DesktopUnscaledY()

111.6 DesktopScaledY

Syntax

```
Result = DesktopScaledY(Value)
```

Description

Returns the scaled value according to current display DPI on 'y' axis. This is mostly useful to calculate real pixel position independently of the display DPI.

Parameters

Value The value to use.

Return value

Returns the scaled value according to current display DPI on 'y' axis. For example, on a display with a 125% DPI applied, a value of 100 will result to 125.

Remarks

The application needs to be compiled with the 'DPI Aware' switch to have this command returning scaled DPI value. If not, the result will be always be the same as 'Value' parameter.

Example

```
1  Debug "Desktop DPI 'y' scaled value of 100: " + DesktopScaledY(100)
```

See Also

DesktopResolutionX() , DesktopResolutionY() , DesktopScaledX() , DesktopUnscaledX() , DesktopUnscaledY()

111.7 DesktopUnscaledX

Syntax

```
Result = DesktopUnscaledX(Value)
```

Description

Returns the unscaled value according to current display DPI on 'x' axis. This is mostly useful to calculate real pixel position independently of the display DPI.

Parameters

Value The value to use.

Return value

Returns the unscaled value according to current display DPI on 'x' axis. For example, on a display with a 125% DPI applied, a value of 125 will result to 100.

Remarks

The application needs to be compiled with the 'DPI Aware' switch to have this command returning scaled DPI value. If not, the result will be always be the same as 'Value' parameter.

Example

```
1  Debug "Desktop DPI 'x' unscaled value of 125: " +  
    DesktopUnscaledX(125)
```

See Also

DesktopResolutionX() , DesktopResolutionY() , DesktopScaledX() , DesktopScaledY() , DesktopUnscaledY()

111.8 DesktopUnscaledY

Syntax

```
Result = DesktopUnscaledY(Value)
```

Description

Returns the unscaled value according to current display DPI on 'y' axis. This is mostly useful to calculate real pixel position independently of the display DPI.

Parameters

Value The value to use.

Return value

Returns the unscaled value according to current display DPI on 'y' axis. For example, on a display with a 125% DPI applied, a value of 125 will result to 100.

Remarks

The application needs to be compiled with the 'DPI Aware' switch to have this command returning scaled DPI value. If not, the result will be always be the same as 'Value' parameter.

Example

```
1  Debug "Desktop DPI 'y' unscaled value of 125: " +  
    DesktopUnscaledY(125)
```

See Also

DesktopResolutionX() , DesktopResolutionY() , DesktopScaledX() , DesktopScaledY() ,
DesktopUnscaledX()

111.9 DesktopFrequency

Syntax

```
Result = DesktopFrequency(#Desktop)
```

Description

Returns the frequency of the specified desktop.

Parameters

#Desktop The index of the desktop. The first index always specifies the primary monitor. The first index value is zero.

Return value

Returns the frequency of the specified desktop in Hertz. If the return-value is 0 then the default hardware frequency is being used, or the actual frequency could not be determined.

Remarks

ExamineDesktops() must be called before using this function to retrieve information about the available desktops. **Note:** on Linux, this function always returns 0.

Example

```
1  ExamineDesktops()
2  f = DesktopFrequency(0)
3  If f = 0
4      MessageRequester("Display Information", "There isn't set any
      desktop frequency, the standard hardware frequency is used.")
5  Else
6      MessageRequester("Display Information", "Frequency of desktop:
      "+Str(f)+" Hz.")
7  EndIf
```

See Also

ExamineDesktops() , DesktopDepth() , DesktopHeight() , DesktopName() , DesktopWidth()

Supported OS

Windows, MacOS X

111.10 DesktopHeight

Syntax

```
Result = DesktopHeight(#Desktop)
```

Description

Returns the height of the specified desktop.

Parameters

#Desktop The index of the desktop. The first index always specifies the primary monitor. The first index value is zero.

Return value

Returns the height in pixels.

Remarks

ExamineDesktops() must be called before using this function to retrieve information about the available desktops.

Example

```
1 ExamineDesktops()
2 MessageRequester("Display Information", "Current resolution =
  "+Str(DesktopWidth(0))+"x"+Str(DesktopHeight(0))+"x"+Str(DesktopDepth(0)))
```

See Also

ExamineDesktops() , DesktopDepth() , DesktopX() , DesktopY() , DesktopWidth()

111.11 DesktopX

Syntax

```
Result = DesktopX(#Desktop)
```

Description

Returns the x coordinate of the specified desktop.

Parameters

#Desktop The index of the desktop. The first index always specifies the primary monitor. The first index value is zero.

Return value

Returns the x coordinate (in pixel) of the top left corner of the desktop. The coordinate is relative to the top left corner of the primary monitor. It is negative if the specified monitor is to the left of the primary monitor.

Remarks

ExamineDesktops() must be called before using this function to retrieve information about the available desktops.

See Also

ExamineDesktops() , DesktopDepth() , DesktopY() , DesktopHeight() , DesktopWidth()

111.12 DesktopY

Syntax

```
Result = DesktopY(#Desktop)
```

Description

Returns the y coordinate of the specified desktop.

Parameters

#Desktop The index of the desktop. The first index always specifies the primary monitor. The first index value is zero.

Return value

Returns the y coordinate (in pixel) of the top left corner of the desktop. The coordinate is relative to the top left corner of the primary monitor. It is negative if the specified monitor is above the primary monitor.

Remarks

ExamineDesktops() must be called before using this function to retrieve information about the available desktops.

See Also

ExamineDesktops() , DesktopDepth() , DesktopX() , DesktopHeight() , DesktopWidth()

111.13 DesktopMouseX

Syntax

```
Result = DesktopMouseX()
```

Description

Returns the absolute x position of the mouse on the desktop.

Parameters

None.

Return value

Returns the x coordinate (in pixel) of the mouse relative to the top left corner of the primary monitor. The coordinate is negative if the mouse is on a monitor to the left of the primary monitor.

Example

```
1  If OpenWindow(0, 0, 0, 300, 30, "Desktop mouse monitor",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      TextGadget(0, 10, 6, 200, 20, "")
4
5      Repeat
6          Event = WindowEvent()
7
8          If Event = 0 ; No events are in queue anymore, so halt the
9              process for a few milliseconds for multitasking
10             SetGadgetText(0, "Desktop mouse position:
11                 "+Str(DesktopMouseX())+" , "+Str(DesktopMouseY()))
12             Delay(20)
13         EndIf
14
15     Until Event = #PB_Event_CloseWindow
16 EndIf
```

See Also

DesktopMouseY() , DesktopX() , DesktopWidth() , WindowMouseX()

111.14 DesktopMouseY

Syntax

```
Result = DesktopMouseY()
```

Description

Returns the absolute y position of the mouse on the desktop.

Parameters

None.

Return value

Returns the y coordinate (in pixel) of the mouse relative to the top left corner of the primary monitor. The coordinate is negative if the mouse is on a monitor above the primary monitor.

Example

```
1  If OpenWindow(0, 0, 0, 300, 30, "Desktop mouse monitor",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      TextGadget(0, 10, 6, 200, 20, "")
4
5      Repeat
6          Event = WindowEvent()
7
8          If Event = 0 ; No events are in queue anymore, so halt the
           process for a few milliseconds for multitasking
9              SetGadgetText(0, "Desktop mouse position:
10             "+Str(DesktopMouseX())+", "+Str(DesktopMouseY()))
11             Delay(20)
12         EndIf
13
14     Until Event = #PB_Event_CloseWindow
15 EndIf
```

See Also

DesktopMouseX() , DesktopY() , DesktopHeight() , WindowMouseY()

111.15 DesktopName

Syntax

```
Result\$_ = DesktopName(#Desktop)
```

Description

Returns the name (if any) for the specified desktop.

Parameters

#Desktop The index of the desktop. The first index always specifies the primary monitor. The first index value is zero.

Return value

Returns a string with the desktop name. If there is no name then an empty string is returned.

Remarks

ExamineDesktops() must be called before using this function to retrieve information about the available desktops.

Example

```
1 ExamineDesktops()
2 MessageRequester("Display Information", "Primary desktop name =
  "+DesktopName(0))
```

See Also

ExamineDesktops() , DesktopDepth() , DesktopFrequency() , DesktopHeight() , DesktopWidth()

111.16 DesktopWidth

Syntax

```
Result = DesktopWidth(#Desktop)
```

Description

Returns the width for the specified desktop.

Parameters

#Desktop The index of the desktop. The first index always specifies the primary monitor. The first index value is zero.

Return value

Returns the width in pixels.

Remarks

ExamineDesktops() must be called before using this function to retrieve information about the available desktops.

Example

```
1 ExamineDesktops()
2 MessageRequester("Display Information", "Current resolution =
  "+Str(DesktopWidth(0))+"x"+Str(DesktopHeight(0))+"x"+Str(DesktopDepth(0)))
```

See Also

ExamineDesktops() , DesktopDepth() , DesktopX() , DesktopY() , DesktopHeight()

Chapter 112

Dialog

Overview

The dialog library allow to easily create complex user interface (GUI) based on an XML definition. It features automatic gadget layout, which is very useful when creating interface which needs to work on different operating systems or working with different font size.

The XML definition can be file based, or created on the fly in memory using the XML library. To find a list of all supported keywords and attributes of this library, see [here](#) .

112.1 CreateDialog

Syntax

```
Result = CreateDialog(#Dialog)
```

Description

Create a new uninitialized dialog. To initialize the dialog, use `OpenXMLDialog()` .

Parameters

#Dialog A number to identify the new dialog. `#PB_Any` can be used to auto-generate this number.

Return value

Returns nonzero if the dialog was created successfully and zero if not. If `#PB_Any` was used as the `#Dialog` parameter, then the auto-generated number is returned in case of success.

See Also

`OpenXMLDialog()` , `FreeDialog()`

112.2 DialogError

Syntax

```
Result\$$ = DialogError(#Dialog)
```

Description

Returns the last error message (in english) to get more information about dialog creation failure after OpenXMLDialog() .

Parameters

#Dialog The dialog to use.

Return value

Returns the error message. If no additional information is available, then the error message can be empty.

See Also

CreateDialog() , OpenXMLDialog()

112.3 DialogGadget

Syntax

```
Result = DialogGadget(#Dialog, Name$)
```

Description

Returns the gadget number of the specified gadget name.

Parameters

#Dialog The dialog to use.

Name\$ The name of the gadget, as specified in the XML file (using the 'name' attribute).

Return value

Returns the gadget number of the specified gadget name, or -1 if the gadget isn't found in the dialog.

See Also

CreateDialog() , OpenXMLDialog()

112.4 DialogWindow

Syntax

```
Result = DialogWindow(#Dialog)
```

Description

Returns the window number of the dialog. It allows to use any window related commands with the dialog. The dialog has to be initialized successfully with `OpenXMLDialog()` before using this command.

Parameters

#Dialog The dialog to use.

Return value

Returns the window number of the specified dialog.

See Also

`CreateDialog()` , `OpenXMLDialog()`

112.5 DialogID

Syntax

```
Result = DialogID(#Dialog)
```

Description

Returns the unique ID which identifies the dialog in the operating system.

Parameters

#Dialog The dialog to use.

Return value

Returns the unique ID which identifies the dialog in the operating system.

See Also

`CreateDialog()` , `OpenXMLDialog()`

112.6 FreeDialog

Syntax

```
FreeDialog(#Dialog)
```

Description

Free the specified dialog and release its associated memory. If the dialog window was still opened, it will be automatically closed.

Parameters

#Dialog The dialog to free. If **#PB_All** is specified, all the remaining dialogs are freed.

Return value

None.

Remarks

All remaining dialogs are automatically freed when the program ends.

See Also

CreateDialog()

112.7 IsDialog

Syntax

```
Result = IsDialog(#Dialog)
```

Description

Tests if the given dialog number is a valid dialog.

Parameters

#Dialog The dialog to test.

Return value

Returns nonzero if **#Dialog** is a valid dialog and zero if not.

Remarks

This function is bulletproof and can be used with any value. This is the correct way to ensure a dialog is ready to use.

See Also

CreateDialog()

112.8 OpenXMLDialog

Syntax

```
Result = OpenXMLDialog(#Dialog, #XML, Name$ [, x, y [, Width, Height [, ParentID]])
```

Description

Open the specified dialog and display it on the screen. To access the dialog gadgets use DialogGadget() . To get the window number of this dialog use DialogWindow() . UseDialogScintillaGadget() , UseDialogOpenGLGadget() , UseDialogWebGadget() and UseDialogWebViewGadget() can be used to enable support for these gadgets in the dialog lib. They are not enabled by default to reduce the dialog library size footprint.

Parameters

#Dialog The dialog to use. It has to be previously created with CreateDialog() .

#XML The xml to use. It has to be previously created with LoadXML() , CreateXML() , CatchXML() or ParseXML() . That means it's possible to create dialogs on the fly with CreateXML() , CatchXML() or ParseXML() . See below for the supported XML attributes. When including XML in the code, it may be easier to use single quote in XML for attribute (it's perfectly legal XML syntax).

Name\$ The name of the dialog to open. An XML file can have several dialogs defined.

x, y (optional) The x, y coordinate (in pixels) of the #Dialog.

Width, Height (optional) The size (in pixels) of the #Dialog. If the size is smaller than the required size as defined in the XML (after layout calculation), then the required size will be used. If omitted, the size of the dialog will be the smallest size required.

ParentID (optional) The parent window identifier. A valid window identifier can be retrieved with WindowID() .

Return value

Returns nonzero if the dialog has been successfully opened, returns zero otherwise. To get more information about the error which has occurred, use DialogError() .

Remarks

Dialog XML format

I. Common attributes

width - positive integer value or 0 (default="0") (set the
 "minimum size" of a control)
height

id - #Number identifier for a gadget or a window (default is
 #PB_Any if not specified). It can be a runtime constant.
name - a string identifying the object (for DialogGadget()
mainly, case insensitive) (default="")
text - text string for the object (default="")

flags - gadget/window flags in the form "#PB_Window_Borderless |
 #PB_Window_ScreenCentered" (default="")

min - minimum value
max - maximum value
value - current value

invisible - if set to "yes", creates the object invisible
 (default="no")
disabled - if "yes", creates the object disabled (gadgets only)
 (default="no")

colspan - inside the <gridbox> element only, allows an element to
 span multiple rows/columns
rowspan (default="1")

All these attributes are optional.

II. Root element

<window> for a single window definition in the same XML file
</window>

or

<dialogs> for a multiple window definition in the same XML file
 <window name="FirstWindow">
 </window>
 <window name="SecondWindow">
 </window>
 ...
</dialogs>

III. Window element

<window>

</window>

Accepted keys in the XML:

All common attributes and the following:

```
minwidth  = 'auto' or a numeric value
maxwidth  = 'auto' or a numeric value
minheight = 'auto' or a numeric value
maxheight = 'auto' or a numeric value
```

It allows to set the window bounds. If set to 'auto', then the size is calculated depending of the children size requirement.

- Creates the a window
- Can have all common attributes.
- Is a single-element container.
- If more than one <window> element is present, the 'name' attribute is used to identify them
- all gui elements can only be placed in here

IV. Layout elements

```
*****
hbox and vbox
*****
```

Arrange the elements horizontally or vertically. Can contain any number of children.

Accepted keys in the XML:

All common attributes and the following:

```
spacing    = space to add between the packed childs (default=5)
```

```
expand     = yes          - items get bigger to fill all space
(default)
```

```
no          - do not expand to fill all space
```

```
equal       - force equal sized items
```

```
item:<number> - expand only one item if space is
```

available

```
align       = top/center/bottom - only applied when expand="no"
for the vbox and top is the default
```

```
            = left/center/right - only applied when expand="no"
for the hbox et left is the default
```

- An assembly of vbox and hbox

allows a complex alignment top/left/center/bottom/right

```
*****
gridbox
*****
```

Align elements in a table. Can contain any number of children.

Accepted keys in the XML:

All common attributes and the following:

columns = number of columns (default = 2)

colspacing = space to add between columns/rows (default = 5)
rowspacing

colexpand = yes - items get bigger to fill all space
rowexpand no - do not expand to fill all space
 equal - force equal sized items
 item:<number> - expand only one item if space is
available

for colexpand, Default=yes, For rowexpand, Default=no

Any child within a gridbox can have these keys:

colspan = number of columns to span (default = 1)
rowspan = number of rows to span

multibox

A box with multiple childs in the same position. Used to put
multiple containers
inside and show only one of them at a time. Can contain
any number of children.

Accepted keys in the XML:

All common attributes.

singlebox

A box with just one child. Used only to apply extra
margin/alignment properties to a child.
Its called a box (as all virtual containers are called that).

Accepted keys in the XML:

All common attributes and the following:

margin = margin around the content (default = 10)
 can be a single number (= all margin), or a combination of
 top:<num>,left:<num>,right:<num>,bottom:<num>,vertical:<num>,horizontal
 example: "vertical:5,left:10,right:0"

expand = yes - expand child to fill all space (default)
 no - no expanding
 vertical - expand vertically only
 horizontal - expand horizontally only

expandwidth = max size to expand the children to. If the requested
size is larger than

expandheight this setting then the request size is used (ie the content does **not** get smaller)
default=0

align = combination of top, left, bottom, right and center. (only effective when expand <> yes)
example: "top, center" or "top, left" (default)

V. Gadget elements

All common XML attributes are supported. To bind an event procedure directly in the xml, the following attributes are available for the gadgets:

onevent = EventProcedure() - generic event binding, for all event type
onchange = EventProcedure() - #PB_EventType_Change binding (only for gadget which support this event type)
onfocus = EventProcedure() - #PB_EventType_Focus binding (only for gadget which support this event type)
onlostfocus = EventProcedure() - #PB_EventType_LostFocus binding (only for gadget which support this event type)
ondragstart = EventProcedure() - #PB_EventType_DragStart binding (only for gadget which support this event type)
onrightclick = EventProcedure() - #PB_EventType_RightClick binding (only for gadget which support this event type)
onleftclick = EventProcedure() - #PB_EventType_LeftClick binding (only for gadget which support this event type)
onrightdoubleclick = EventProcedure() - #PB_EventType_RightDoubleClick binding (only for gadget which support this event type)
onleftdoubleclick = EventProcedure() - #PB_EventType_LeftDoubleClick binding (only for gadget which support this event type)

The 'EventProcedure()' has to be declared as 'Runtime' in the main code, and has to respect the BindEvent() procedure format. Under the hood, BindGadgetEvent() is called with the specified procedure.

Supported gadgets:

<button>
<buttonimage>
<calendar>
<canvas>
<checkbox>
<combobox>
<container> - single element container
<date>
<editor>
<explorercombo>
<explorerlist>
<exploretree>
<frame> - single element container (with standard borders only)
<hyperlink>


```

<ipaddress>
<image>
<listicon>
<listview>
<option group> - use the same 'group' number to create linked
  OptionGadget()

<opengl> - UseDialogOpenGLGadget() needs to be called to enable
  <opengl> support.
<panel> - can contain <tab> items only
<progressbar min max value>
<scintilla> - callback remains empty. UseDialogScintillaGadget()
  needs to be called to enable <scintilla> support.
<scrollarea scrolling="vertical, horizontal or both (default)"
  innerheight="value or auto (default)" innerwidth="value or auto
  (default)" step> - single element container, scrolling value
  determines growth behavior.
<scrollbar min max page value> - page = page length.
<spin min max value>
<splitter firstmin="value or auto" secondmin> - must contain 2
  subitems, so its a 2 item container basically, minimum size is
  determined by contained gadgets. If "auto" is specified, the min
  value will be the minimum size of the child.
<string>
<text>
<trackbar min max value>
<tree>
<web> - UseDialogWebGadget() needs to be called to enable <web>
  support.
<webview> - UseDialogWebViewGadget() needs to be called to enable
  <webview> support.

```

Gadget related elements:

```

<tab> - single element container, for panel tabs (attribute 'text' is
  supported).

```

Special elements:

```

<empty> - an empty element, useful when it's needed to have space
  between element, to align them to borders for example.

```

Example: Simple resizable dialog

```

1  #Dialog = 0
2  #Xml = 0
3
4  XML$ = "<window id='PB_Any' name='test' text='test' minwidth='auto'
      minheight='auto' flags='PB_Window_ScreenCentered |
      PB_Window_SystemMenu | PB_Window_SizeGadget'>" +
5      "    <panel>" +
6      "      <tab text='First tab'>" +
7      "        <vbox expand='item:2'>" +
8      "          <hbox>" +
9      "            <button text='button 1'>" +
10     "            <checkbox text='checkbox 1'>" +
11     "            <button text='button 2'>" +
12     "          </hbox>" +
13     "          <editor text='content' height='150'>" +
14     "        </vbox>" +
15     "      </tab>" +

```

```

16         "        <tab text='Second tab'>" +
17         "        </tab>" +
18         "    </panel>" +
19         "</window>"
20
21 If ParseXML(#Xml, XML$) And XMLStatus(#Xml) = #PB_XML_Success
22
23     If CreateDialog(#Dialog) And OpenXMLDialog(#Dialog, #Xml, "test")
24
25         Repeat
26             Event = WaitWindowEvent()
27             Until Event = #PB_Event_CloseWindow
28
29         Else
30             Debug "Dialog error: " + DialogError(#Dialog)
31         EndIf
32     Else
33         Debug "XML error: " + XMLError(#Xml) + " (Line: " +
34             XMLErrorLine(#Xml) + ")"
35     EndIf

```

Example: Multibox example

```

1     #Dialog = 0
2     #Xml = 0
3
4     Runtime Enumeration Gadget
5         #ListView
6         #GeneralContainer
7         #EditorContainer
8         #BackupContainer
9     EndEnumeration
10
11     Procedure ShowPanels()
12
13         HideGadget(#GeneralContainer, #True)
14         HideGadget(#EditorContainer, #True)
15         HideGadget(#BackupContainer, #True)
16
17         Select GetGadgetState(#ListView)
18             Case 0
19                 HideGadget(#GeneralContainer, #False)
20
21             Case 1
22                 HideGadget(#EditorContainer, #False)
23
24             Case 2
25                 HideGadget(#BackupContainer, #False)
26         EndSelect
27     EndProcedure
28
29     Runtime Procedure OnListViewEvent()
30         ShowPanels()
31     EndProcedure
32
33     XML$ = "<window id='#PB_Any' name='test' text='Preferences'
34         minwidth='auto' minheight='auto' flags='#PB_Window_ScreenCentered |

```

```

#PB_Window_SystemMenu | #PB_Window_SizeGadget '>' +
34     "    <hbox expand='item:2'>" +
35     "        <listview id='#ListView' width='100'
onEvent='OnListViewEvent()'>" +
36     "        <multibox>" +
37     "            " +
38     "                <container id='#GeneralContainer' invisible='yes'>" +
39     "                    <frame text='General'>" +
40     "                        <vbox expand='no'>" +
41     "                            <checkbox text='Enable red light'>" +
42     "                            <checkbox text='Enable green light'>" +
43     "                        </vbox>" +
44     "                    </frame>" +
45     "                </container>" +
46     "            " +
47     "                <container id='#EditorContainer' invisible='yes'>" +
48     "                    <frame text='Editor'>" +
49     "                        <vbox expand='no'>" +
50     "                            <checkbox text='Set read only mode'>" +
51     "                            <checkbox text='Duplicate line automatically'>" +
52     "                            <checkbox text='Enable monospace font'>" +
53     "                        </vbox>" +
54     "                    </frame>" +
55     "                </container>" +
56     "            " +
57     "                <container id='#BackupContainer' invisible='yes'>" +
58     "                    <frame text='Backup'>" +
59     "                        <vbox expand='no'>" +
60     "                            <checkbox text='Activate backup'>" +
61     "                        </vbox>" +
62     "                    </frame>" +
63     "                </container>" +
64     "            " +
65     "        </multibox>" +
66     "    </hbox>" +
67     "</window>"

68
69 If ParseXML(#Xml, XML$) And XMLStatus(#Xml) = #PB_XML_Success
70
71     If CreateDialog(#Dialog) And OpenXMLDialog(#Dialog, #Xml, "test")
72
73         AddGadgetItem(#ListView, -1, "General")
74         AddGadgetItem(#ListView, -1, "Editor")
75         AddGadgetItem(#ListView, -1, "Backup")
76
77         SetGadgetState(#ListView, 0)
78
79         ShowPanels()
80
81         Repeat
82             Event = WaitWindowEvent()
83             Until Event = #PB_Event_CloseWindow
84
85         Else
86             Debug "Dialog error: " + DialogError(#Dialog)
87         EndIf
88     Else
89         Debug "XML error: " + XMLError(#Xml) + " (Line: " +
XMLErrorLine(#Xml) + ")"

```

Example: Gridbox example

```

1  #Dialog = 0
2  #Xml = 0
3
4  XML$ = "<window id='#PB_Any' name='test' text='Gridbox'
      minwidth='auto' minheight='auto' flags='#PB_Window_ScreenCentered |
      #PB_Window_SystemMenu | #PB_Window_SizeGadget'>" +
5      "    <gridbox columns='6'>" +
6      "        <button text='Button 1' />" +
7      "        <button text='Button 2' />" +
8      "        <button text='Button 3' colspan='3' />" +
9      "        <button text='Button 4' />" +
10     "        <button text='Button 5' rowspan='2' />" +
11     "        <button text='Button 6' />" +
12     "        <button text='Button 7' />" +
13     "        <button text='Button 8' />" +
14     "        <button text='Button 9' />" +
15     "        <button text='Button 10' />" +
16     "        <button text='Button 11' />" +
17     "        <button text='Button 12' />" +
18     "    </gridbox>" +
19     "</window>"
20
21 If ParseXML(#Xml, XML$) And XMLStatus(#Xml) = #PB_XML_Success
22
23     If CreateDialog(#Dialog) And OpenXMLDialog(#Dialog, #Xml, "test")
24
25         Repeat
26             Event = WaitWindowEvent()
27             Until Event = #PB_Event_CloseWindow
28
29         Else
30             Debug "Dialog error: " + DialogError(#Dialog)
31         EndIf
32     Else
33         Debug "XML error: " + XMLError(#Xml) + " (Line: " +
34             XMLErrorLine(#Xml) + ")"
35     EndIf

```

See Also

CreateDialog()

112.9 RefreshDialog

Syntax

```
RefreshDialog(#Dialog)
```

Description

Refresh the dialog size to adjust it to any change. For example, when changing the text content of gadgets, the dialog size will may be need adjustments.

Parameters

#Dialog The dialog to refresh.

Return value

None.

See Also

CreateDialog()

112.10 UseDialogOpenGLGadget

Syntax

```
UseDialogOpenGLGadget()
```

Description

Enable OpenGLGadget() support in the dialog library. This is not enabled by default to reduce the dialog library size footprint if OpenGLGadget() is not needed.

Return value

None.

See Also

OpenXMLDialog()

112.11 UseDialogScintillaGadget

Syntax

```
UseDialogScintillaGadget()
```

Description

Enable ScintillaGadget() support in the dialog library. This is not enabled by default to reduce the dialog library size footprint if ScintillaGadget() is not needed.

Return value

None.

See Also

OpenXMLDialog()

112.12 UseDialogWebGadget

Syntax

```
UseDialogWebGadget ()
```

Description

Enable WebGadget() support in the dialog library. This is not enabled by default to reduce the dialog library size footprint if WebGadget() is not needed.

Return value

None.

See Also

OpenXMLDialog()

112.13 UseDialogWebViewGadget

Syntax

```
UseDialogWebViewGadget ()
```

Description

Enable WebViewGadget() support in the dialog library. This is not enabled by default to reduce the dialog library size footprint if WebViewGadget() is not needed.

Return value

None.

See Also

OpenXMLDialog()

Chapter 113

DragDrop

Overview

Drag & Drop is a widely used technology today as it provides an easy and very intuitive way to move data around between windows and applications. This library provides cross-platform functions to add this functionality to gadgets and windows with just a few extra lines of code.

Furthermore this library provides some functions to extend its default possibilities through platform specific APIs, giving the experienced programmer all the freedom to work with custom formats while still taking advantage of the framework provided by this library.

There are limitations with OSX, only pictures can be drag and dropped.

113.1 DragText

Syntax

```
Result = DragText(Text$ [, Actions])
```

Description

Starts a Drag & Drop operation with text data.

Parameters

Text\$ The text to drag.

Actions (optional) A combination of the Drag & Drop actions that should be allowed for the data. If the parameter is not specified, `#PB_Drag_Copy` will be the only allowed action. Possible actions are: (they can be combined with '|')

`#PB_Drag_Copy`: The text can be copied

`#PB_Drag_Move`: The text can be moved

`#PB_Drag_Link`: The text can be linked

The user can decide which of these actions to take by pressing modifier keys like Ctrl or Shift. The actions that can really be taken also depend on the actions allowed by the drop target. (On MacOSX, the actions are only treated as a suggestion. The drop target can still choose another action.)

Return value

Returns one of the above Drag & Drop action values to indicate what action the user took, or `#PB_Drag_None` if the user aborted the Drag & Drop operation.

Note that if `#PB_Drag_Move` is returned, it is your responsibility to remove the dragged text data from your application.

Remarks

Drag & Drop can basically be started any time, but the left mouse button should be currently pressed as otherwise the operation will end immediately without success. The usual time to start a Drag & Drop operation is when a Gadget reported an event with `EventType()` of `#PB_EventType_DragStart`. On MacOS X the `#PB_EventType_DragStart` event has to be handled using `BindEvent()` or `BindGadgetEvent()`.

Example

```
1  Procedure DragStartHandler()
2      ExamineDraggedItems()
3      While NextDraggedItem()
4          Text$ + GetGadgetItemText(0, DraggedItemIndex()) + Chr(10)
5      Wend
6
7      Debug "Dragging text: " + Text$
8
9      DragText(Text$)
10 EndProcedure
11
12 ; Select some files or folders and drag them to another application
13 ;
14 If OpenWindow(0, 200, 200, 400, 400, "Drag & Drop")
15     ListViewGadget(0, 10, 10, 380, 380, #PB_ListView_MultiSelect)
16     AddGadgetItem(0, -1, "Item 1")
17     AddGadgetItem(0, -1, "Item 2")
18     AddGadgetItem(0, -1, "Item 3")
19
20     ; BindGadgetEvent() is required to handle the drag start event
21     BindGadgetEvent(0, @DragStartHandler(), #PB_EventType_DragStart)
22
23     Repeat
24         Event = WaitWindowEvent()
25     Until Event = #PB_Event_CloseWindow
26 EndIf
```

See Also

`DragFiles()` , `DragImage()` , `DragPrivate()` , `DragOSFormats()` , `SetDragCallback()`

Supported OS

Windows, Linux

113.2 DragImage

Syntax

```
Result = DragImage(ImageID [, Actions])
```

Description

Starts a Drag & Drop operation with image data.

Parameters

ImageID The image to drag. ImageID() can be used to get this ID for an image.

Actions (optional) A combination of the Drag & Drop actions that should be allowed for the image. If the parameter is not specified, `#PB_Drag_Copy` will be the only allowed action. Possible actions are: (they can be combined with '|')

```
#PB_Drag_Copy: The image can be copied
#PB_Drag_Move: The image can be moved
#PB_Drag_Link: The image can be linked
```

The user can decide which of these actions to take by pressing modifier keys like Ctrl or Shift. The actions that can really be taken also depend on the actions allowed by the drop target. (On MacOSX, the actions are only treated as a suggestion. The drop target can still choose another action.)

Return value

Returns one of the above Drag & Drop action values to indicate what action the user took, or `#PB_Drag_None` if the user aborted the Drag & Drop operation.

Note that if `#PB_Drag_Move` is returned, it is your responsibility to remove the dragged image from your application.

Remarks

Drag & Drop can basically be started any time, but the left mouse button should be currently pressed as otherwise the operation will end immediately without success. The usual time to start a Drag & Drop operation is when a Gadget reported an event with EventType() of `#PB_EventType_DragStart`. On MacOS X the `#PB_EventType_DragStart` event has to be handled using BindEvent() or BindGadgetEvent() .

Example

```
1 ; Drag the image to an application that can accept images like an
  ; office or graphic program.
2 ;
3 If LoadImage(1, #PB_Compiler_Home +
  "examples/sources/data/PureBasicLogo.bmp")
4   If OpenWindow(1, 200, 200, 400, 90, "Drag & Drop",
  #PB_Window_SystemMenu)
5     ImageGadget(1, 10, 10, 380, 70, ImageID(1))
6
```

```

7      Repeat
8          Event = WaitWindowEvent()
9          If Event = #PB_Event_Gadget And EventGadget() = 1 And
EventType() = #PB_EventType_DragStart
10             DragImage(ImageID(1))
11         EndIf
12     Until Event = #PB_Event_CloseWindow
13 EndIf
14 EndIf

```

See Also

DragFiles() , DragText() , DragPrivate() , DragOSFormats() , SetDragCallback()

113.3 DragFiles

Syntax

```
Result = DragFiles(Files$ [, Actions])
```

Description

Starts a Drag & Drop operation with a list of filenames.

Parameters

Files\$ The list of filenames or directories to drag. Multiple filenames should be separated with a Chr(10) (linefeed) character. Each filename must include its full path, as the target application will not know how to resolve relative names. The filenames must refer to existing files, so the target application can access them.

Actions (optional) A combination of the Drag & Drop actions that should be allowed for the files. If the parameter is not specified, #PB_Drag_Copy will be the only allowed action. Possible actions are: (they can be combined with '|')

```

#PB_Drag_Copy: The files can be copied
#PB_Drag_Move: The files can be moved
#PB_Drag_Link: The files can be linked

```

The user can decide which of these actions to take by pressing modifier keys like Ctrl or Shift. The actions that can really be taken also depend on the actions allowed by the drop target. (On MacOSX, the actions are only treated as a suggestion. The drop target can still choose another action.)

Return value

Returns one of the above Drag & Drop action values to indicate what action the user took, or

#PB_Drag_None if the user aborted the Drag & Drop operation.

Note that unlike with the other functions that start Drag & Drop, nothing should be done when #PB_Drag_Move is returned. As the dragged data is only the filename and not the file itself, any action that is taken on the file must be done by the drag target.

Remarks

Drag & Drop can basically be started any time, but the left mouse button should be currently pressed as otherwise the operation will end immediately without success. The usual time to start a Drag & Drop operation is when a Gadget reported an event with EventType() of #PB_EventType_DragStart. On MacOS X the #PB_EventType_DragStart event has to be handled using BindEvent() or BindGadgetEvent() .

Example

```
1  Procedure DragStartHandler()
2      ExamineDraggedItems()
3      While NextDraggedItem()
4          Files$ + GetGadgetText(0) + GetGadgetItemText(0,
5              DraggedItemIndex()) + Chr(10)
6      Wend
7
8      Debug "Dragging files: "+Files$
9
10     DragFiles(Files$)
11 EndProcedure
12
13 ; Select some files or folders and drag them to another application
14 ;
15 If OpenWindow(0, 200, 200, 400, 400, "Drag & Drop")
16     ExplorerListGadget(0, 10, 10, 380, 380, GetHomeDirectory(),
17         #PB_Explorer_MultiSelect)
18
19     ; BindGadgetEvent() is required to handle the drag start event
20     BindGadgetEvent(0, @DragStartHandler(), #PB_EventType_DragStart)
21
22     Repeat
23         Event = WaitWindowEvent()
24     Until Event = #PB_Event_CloseWindow
25 EndIf
```

See Also

DragText() , DragImage() , DragPrivate() , DragOSFormats() , SetDragCallback()

Supported OS

Windows, Linux

113.4 DragPrivate

Syntax

```
Result = DragPrivate(Type [, Actions])
```

Description

Starts a "private" Drag & Drop operation. Unlike the other functions that start Drag & Drop, this data can only be dropped inside the application (Data dragged with functions like DragText() or DragImage() can be accepted by other applications as well). This function should be used to add Drag & Drop functionality between Gadgets or Windows with data that would not be understood by other applications.

Parameters

Type This parameter can be any integer value that identifies the data to be dragged in the application. The same value must be specified for EnableGadgetDrop() or EnableWindowDrop() for those Gadget /Windows that should accept this data. This way it can be exactly defined which private drag operation will be accepted by Gadget/Window, which allows complex Drag & Drop schemes to be realized.

Actions (optional) A combination of the Drag & Drop actions that should be allowed for the data. If the parameter is not specified, `#PB_Drag_Copy` will be the only allowed action. Possible actions are: (they can be combined with '|')

```
#PB_Drag_Copy: The data can be copied
#PB_Drag_Move: The data can be moved
#PB_Drag_Link: The data can be linked
```

The user can decide which of these actions to take by pressing modifier keys like Ctrl or Shift. The actions that can really be taken also depend on the actions allowed by the drop target.

Return value

Returns one of the above Drag & Drop action values to indicate what action the user took, or `#PB_Drag_None` if the user aborted the Drag & Drop operation.

Remarks

Drag & Drop can basically be started any time, but the left mouse button should be currently pressed as otherwise the operation will end immediately without success. The usual time to start a Drag & Drop operation is when a Gadget reported an event with EventType() of `#PB_EventType_DragStart`. If the operation was not aborted, the event loop will receive a `#PB_Event_WindowDrop` or `#PB_Event_GadgetDrop` event of type `#PB_Drop_Private`.

See Also

DragText() , DragImage() , DragFiles() , DragOSFormats() , SetDragCallback()

Supported OS

Windows, Linux

113.5 DragOSFormats

Syntax

```
Result = DragOSFormats(Formats(), Count [, Actions])
```

Description

Starts a Drag & Drop operation with a list of custom data formats. The types of formats available and the way in which they are represented depends on the Operating system. This function offers the possibility to work with formats not natively supported by PureBasic, while still using the simple mechanisms provided by this library to carry out the Drag & Drop operation.

Parameters

Formats() An array of DragDataFormat structures containing one or more formats to drag. The structure has the following form:

```
1  Structure DragDataFormat
2      Format.i      ; The OS specific ID for the format to drag (see
                        below for more information)
3      *Buffer      ; The memory buffer containing the data in this
                        format
4      Size.i       ; The size of the data in the buffer
5  EndStructure
```

Windows:

On Windows, the 'Format' field specifies a CLIPBOARDFORMAT value. It can be any standard clipboard format (found in the Windows SDK), or a format registered with the RegisterClipboardFormat_() API.

Linux:

On Linux, the 'Format' field specifies a GdkAtom value. It can be created with the gdk function gdk_atom_intern_(). Widely understood atoms are the common mime types (ie "text/html" for html data). The atom can also be created with any string that is understood by the target application.

MacOSX:

On MacOSX, the 'Format' field specifies a clipboard scrap type. These are 4 letter numeric character constants, for example 'TEXT'. There are a number of predefined scrap types, but also custom values can be used if the target program understands them.

Count The number of formats in the array.

If multiple formats are dragged, the target will accept the first one it recognizes. So the format that provides the most information (i.e. represents the data most accurately) should be first in the array, with less descriptive but more common formats later. This way each application will get the best representation of the data that it understands.

Actions (optional) A combination of the Drag & Drop actions that should be allowed for the data. If the parameter is not specified, #PB_Drag_Copy will be the only allowed action. Possible actions are: (they can be combined with '|')

```
#PB_Drag_Copy: The data can be copied
#PB_Drag_Move: The data can be moved
#PB_Drag_Link: The data can be linked
```

The user can decide which of these actions to take by pressing modifier keys like Ctrl or Shift. The actions that can really be taken also depend on the actions allowed by the drop target. (On MacOSX, the actions are only treated as a suggestion. The drop target can still choose another action.)

Return value

Returns one of the above Drag & Drop action values to indicate what action the user took, or `#PB_Drag_None` if the user aborted the Drag & Drop operation.

Note that if `#PB_Drag_Move` is returned, it is your responsibility to remove the dragged data from your application.

Remarks

Drag & Drop can basically be started any time, but the left mouse button should be currently pressed as otherwise the operation will end immediately without success. The usual time to start a Drag & Drop operation is when a Gadget reported an event with `EventType()` of `#PB_EventType_DragStart`.

See Also

`DragText()` , `DragImage()` , `DragFiles()` , `DragPrivate()` , `SetDragCallback()`

113.6 EnableGadgetDrop

Syntax

```
EnableGadgetDrop(#Gadget, Format, Actions [, PrivateType])
```

Description

Enables a gadget to be a target for Drag & Drop operations of a specific format. When the user drags data of this format over the gadget, the cursor will indicate that the data can be dropped there.

Parameters

#Gadget The PureBasic gadget number for the gadget in question.

Format The data format, which can be one of the following values, or an OS specific ID for a custom format (see `DragOSFormats()` for more information).

```
#PB_Drop_Text      : Accept text on this gadget
#PB_Drop_Image     : Accept images on this gadget
#PB_Drop_Files    : Accept filenames on this gadget
#PB_Drop_Private   : Accept a "private" Drag & Drop on this gadget
```

Actions A combination of the Drag & Drop actions that should be allowed for the data. The user can decide which of these actions to take by pressing modifier keys like Ctrl or Shift. The actions that can really be taken also depend on the actions allowed by the drag source. Possible actions are: (they can be combined with '|')

#PB_Drag_None: The data format will **not** be accepted on the gadget
#PB_Drag_Copy: The data can be copied
#PB_Drag_Move: The data can be moved
#PB_Drag_Link: The data can be linked

PrivateType (optional) The type of private Drag & Drop to accept if 'Format' is **#PB_Drop_Private**. See DragPrivate() for more information. This parameter is ignored for other formats.

Return value

None.

Remarks

Multiple formats can be allowed on the same gadget. If the drag source provides multiple formats that match the list of accepted formats, the one that was added last will be accepted. So the preferred format in which to receive data should be enabled last.

If data was dropped on the gadget, the program will receive a **#PB_Event_GadgetDrop** event. EventGadget() will indicate the target gadget and the Event functions of this library can be used to get the dropped data.

Example

```

1  ; Drag a file to one of the gadgets and you will receive the file path
2  ;
3  If OpenWindow(0, 0, 0, 500, 200, "Drop File Here",
4     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
5
6     StringGadget(0, 10, 10, 500 - 20, 30, "")
7     EditorGadget(1, 10, 50, 500 - 20, 140)
8
9     EnableGadgetDrop(0, #PB_Drop_Files, #PB_Drag_Copy)
10    EnableGadgetDrop(1, #PB_Drop_Files, #PB_Drag_Copy)
11
12    Repeat
13        Select WaitWindowEvent()
14        Case #PB_Event_GadgetDrop
15            Select EventGadget()
16                Case 0 ; gadgets that received a file/folder drag
17                    and drop event
18                        If Not FindString(EventDropFiles(), Chr(10))
19                            SetGadgetText(0, EventDropFiles())
20                        EndIf
21                    Case 1
22                        SetGadgetText(1, EventDropFiles())
23                    EndSelect
24                Case #PB_Event_CloseWindow
25                    CloseWindow(0)
26            End
27        EndSelect
28    ForEver
29 EndIf
  
```

See Also

EnableWindowDrop() , EventDropType() , EventDropAction() , SetDropCallback()

113.7 EnableWindowDrop

Syntax

```
EnableWindowDrop(#Window, Format, Actions [, PrivateType])
```

Description

Enables a window to be a target for Drag & Drop operations of a specific format. Only the area not covered by any gadgets will be the target area. When the user drags data of this format over the window, the cursor will indicate that the data can be dropped there.

Parameters

#Window The PureBasic window number for the window in question.

Format The data format, which can be one of the following values, or an OS specific ID for a custom format (see DragOSFormats() for more information).

```
#PB_Drop_Text      : Accept text on this window
#PB_Drop_Image     : Accept images on this window
#PB_Drop_Files    : Accept filenames on this window
#PB_Drop_Private   : Accept a "private" Drag & Drop on this window
```

Actions A combination of the Drag & Drop actions that should be allowed for the data. The user can decide which of these actions to take by pressing modifier keys like Ctrl or Shift. The actions that can really be taken also depend on the actions allowed by the drag source. Possible actions are: (they can be combined with '|')

```
#PB_Drag_None: The data format will not be accepted on the gadget
#PB_Drag_Copy: The data can be copied
#PB_Drag_Move: The data can be moved
#PB_Drag_Link: The data can be linked
```

PrivateType (optional) The type of private Drag & Drop to accept if 'Format' is

`#PB_Drop_Private`. See DragPrivate() for more information. This parameter is ignored for other formats.

Return value

None.

Remarks

Multiple formats can be allowed on the same window. If the drag source provides multiple formats that match the list of accepted formats, the one that was added last will be accepted. So the preferred format in which to receive data should be enabled last.

If data was dropped on the window, the program will receive a `#PB_Event_WindowDrop` event. EventWindow() will indicate the target window and the Event functions of this library can be used to get the dropped data.

See Also

EnableGadgetDrop() , EventDropType() , EventDropAction() , SetDropCallback()

113.8 EventDropAction

Syntax

```
Result = EventDropAction()
```

Description

After a `#PB_Event_GadgetDrop` or `#PB_Event_WindowDrop` is received by `WaitWindowEvent()` or `WindowEvent()` , this function returns the action that should be taken with the data.

Parameters

None.

Return value

Returns one of the following values:

```
#PB_Drag_Copy: The data should be copied
#PB_Drag_Move: The data should be moved (The drag source is
               responsible for removing the original data)
#PB_Drag_Link: The data should be linked
```

See Also

EnableGadgetDrop() , EnableWindowDrop() , EventDropType() , EventDropX() , EventDropY()

113.9 EventDropType

Syntax

```
Result = EventDropType()
```

Description

After a `#PB_Event_GadgetDrop` or `#PB_Event_WindowDrop` is received by `WaitWindowEvent()` or `WindowEvent()` , this function returns the format of the dropped data.

Parameters

None.

Return value

Returns one of the following values, or an OS specific ID for a custom format (see DragOSFormats() for more information)

```
#PB_Drop_Text      : Text was dropped.          (use EventDropText()
to retrieve it)
#PB_Drop_Image     : An image was dropped.       (use EventDropImage()
to retrieve it)
#PB_Drop_Files    : Filenames were dropped. (use EventDropFiles()
to retrieve them)
#PB_Drop_Private   : A "private" operation finished. (use
EventDropPrivate()
to know its type)
```

Remarks

To handle OS specific formats, EventDropBuffer() and EventDropSize() can be used.

See Also

EnableGadgetDrop() , EnableWindowDrop() , EventDropAction() , EventDropText() ,
EventDropImage() , EventDropFiles() , EventDropPrivate() , EventDropBuffer() , EventDropSize() ,
EventDropX() , EventDropY()

113.10 EventDropText

Syntax

```
Result\$$ = EventDropText()
```

Description

After a #PB_Event_GadgetDrop or #PB_Event_WindowDrop is received by WaitWindowEvent() or WindowEvent() with a format (can be get with EventDropType()) of #PB_Drop_Text, this function returns the text that was dropped.

Parameters

None.

Return value

Returns the dropped text.

See Also

EnableGadgetDrop() , EnableWindowDrop() , EventDropType() , EventDropAction() , EventDropX() ,
EventDropY()

113.11 EventDropImage

Syntax

```
Result = EventDropImage(#Image [, Depth])
```

Description

After a `#PB_Event_GadgetDrop` or `#PB_Event_WindowDrop` is received by `WaitWindowEvent()` or `WindowEvent()` with a format (can be get with `EventDropType()`) of `#PB_Drop_Image`, this function can be used to retrieve the dropped image.

Parameters

#Image The number for the new image to create. `#PB_Any` can be used to auto-generate this number.

Depth (optional) The color depth for the new image. Supported values are 24 and 32-bit. 24-bit is the default.

Return value

Returns nonzero if the image was successfully created and zero if not. If `#PB_Any` was used for the `#Image` parameter then the generated number is returned on success.

See Also

`EnableGadgetDrop()` , `EnableWindowDrop()` , `EventDropType()` , `EventDropAction()` , `EventDropX()` , `EventDropY()`

113.12 EventDropFiles

Syntax

```
Result\[extract_itex] = EventDropFiles()
```

Description

After a `#PB_Event_GadgetDrop` or `#PB_Event_WindowDrop` is received by `WaitWindowEvent()` or `WindowEvent()` with a format (can be get with `EventDropType()`) of `#PB_Drop_Files`, this function returns the dropped filenames.

Parameters

None.

Return value

Returns a list of filenames or directories, all with full path. The individual names are separated by a Chr(10) (linefeed) character.

See Also

EnableGadgetDrop() , EnableWindowDrop() , EventDropType() , EventDropAction() , EventDropX() , EventDropY()

113.13 EventDropPrivate

Syntax

```
Result = EventDropPrivate()
```

Description

After a #PB_Event_GadgetDrop or #PB_Event_WindowDrop is received by WaitWindowEvent() or WindowEvent() with a format (can be get with EventDropType()) of #PB_Drop_Private, this function returns the 'PrivateType' that was dropped.

Parameters

None.

Return value

Returns the private type value used when starting the Drag & Drop operation. (See DragPrivate() for more information.)

See Also

EnableGadgetDrop() , EnableWindowDrop() , EventDropType() , EventDropAction() , EventDropX() , EventDropY()

113.14 EventDropBuffer

Syntax

```
*Result = EventDropBuffer()
```

Description

After a #PB_Event_GadgetDrop or #PB_Event_WindowDrop is received by WaitWindowEvent() or WindowEvent() with an OS specific format, this function can be used to access the data.

Parameters

None.

Return value

Returns the memory address where the dropped data is located. The `EventDropSize()` command returns the size of this buffer.

Remarks

The returned memory buffer is internal to the library and should not be freed. It is also only valid until the next call to `WaitWindowEvent()` or `WindowEvent()` , so the data should be copied if it is required for longer than that.

See Also

`EventDropSize()` , `EnableGadgetDrop()` , `EnableWindowDrop()` , `EventDropType()` , `EventDropAction()` , `EventDropX()` , `EventDropY()`

113.15 EventDropSize

Syntax

```
Result = EventDropSize()
```

Description

After a `#PB_Event_GadgetDrop` or `#PB_Event_WindowDrop` is received by `WaitWindowEvent()` or `WindowEvent()` with an OS specific format, this function returns the size of the dropped data.

Parameters

None.

Return value

Returns the size in bytes of the memory buffer returned by `EventDropBuffer()` .

See Also

`EventDropBuffer()` , `EnableGadgetDrop()` , `EnableWindowDrop()` , `EventDropType()` , `EventDropAction()` , `EventDropX()` , `EventDropY()`

113.16 EventDropX

Syntax

```
Result = EventDropX()
```

Description

After a `#PB_Event_GadgetDrop` or `#PB_Event_WindowDrop` is received by `WaitWindowEvent()` or `WindowEvent()` , this function returns the X position at which the data was dropped.

Parameters

None.

Return value

Returns the x coordinate of the drop location relative to the gadget or window in which the data was dropped.

See Also

`EnableGadgetDrop()` , `EnableWindowDrop()` , `EventDropType()` , `EventDropAction()` , `EventDropY()`

113.17 EventDropY

Syntax

```
Result = EventDropY()
```

Description

After a `#PB_Event_GadgetDrop` or `#PB_Event_WindowDrop` is received by `WaitWindowEvent()` or `WindowEvent()` , this function returns the Y position at which the data was dropped.

Parameters

None.

Return value

Returns the y coordinate of the drop location relative to the gadget or window in which the data was dropped.

See Also

EnableGadgetDrop() , EnableWindowDrop() , EventDropType() , EventDropAction() , EventDropX()

113.18 SetDragCallback

Syntax

```
SetDragCallback(@DragCallback())
```

Description

A callback function to be called during a Drag & Drop operation initiated from this application. The callback allows to modify the Drag & Drop process provided by PureBasic, for example by providing a custom cursor through the API of the Operating system.

Parameters

@DragCallback() The address of a function to call during a drag operation. The form and purpose of the callback is dependent on the OS. It must take the following form:

Windows:

```
1 Procedure DragCallback(Action.i)
2
3     ProcedureReturn #True
4 EndProcedure
```

The callback is called during the Drag & Drop operation. Action specifies the action that would be taken if the user released the mouse at this point. It can be one of these values:

```
#PB_Drag_None: The data will not be accepted if dropped here
#PB_Drag_Copy: The data will be copied
#PB_Drag_Move: The data will be moved
#PB_Drag_Link: The data will be linked
```

The callback can provide a custom cursor or drag image. If it does so, it should return **#False**. Returning **#True** will cause the default cursor to be used.

Linux:

```
1 Procedure DragCallback(*Context.GdkDragContext, isStart.i)
2
3 EndProcedure
```

The callback is only called at the start and the end of a Drag & Drop operation. The **'*Context'** parameter specifies the gdk drag context of this operation, **'isStart'** specifies whether this is the start or the end of the operation. The return-value of the callback is ignored.

Gtk functions like `gtk_drag_set_icon_pixbuf()` can be used in the callback to set a different drag image for the operation.

MacOSX:

Not supported.

Return value

None.

See Also

SetDropCallback()

113.19 SetDropCallback

Syntax

```
SetDropCallback(@DropCallback())
```

Description

A callback function to be called when data is dragged over a gadget or window that allows data to be dropped (see EnableGadgetDrop() / EnableWindowDrop()). The callback allows to modify the Drag & Drop process provided by PureBasic, for example by providing extra visual notification on the target gadget or window.

Parameters

@DropCallback() A callback to be called during a drop operation.

The callback is called as the mouse enters, moves and leaves the target gadget or window and allows to provide additional feedback to the user, for example by highlighting the target item or area. Furthermore the callback can deny the currently intended action and this way specify more detailed where within a gadget or window the data can be dropped. The cursor should not be modified here, as the drag source is responsible for this.

The form of the callback is described below:

```
1  Procedure DropCallback(TargetHandle, State, Format, Action,
2      x, y)
3      ProcedureReturn #True
4  EndProcedure
```

The first parameter specifies the OS specific handle for the target gadget or window. On Windows this is a HWND value, on Linux a GtkWidget pointer and on MacOSX it is a ControlRef or WindowRef value. These are the same values as returned by GadgetID() or WindowID() for the target gadget or window.

'State' specifies the current state of the Drag & Drop operation and is one of the following values:

```
#PB_Drag_Enter : The mouse entered the gadget or window
#PB_Drag_Update: The mouse was moved inside the gadget or
    window, or the intended action changed
#PB_Drag_Leave : The mouse left the gadget or window (Format,
    Action, x, y are 0 here)
#PB_Drag_Finish: The Drag & Drop finished
```

'Format' specifies the data format and can be one of the following values, or an OS specific ID for a custom format (see DragOSFormats() for more information).

```
#PB_Drop_Text    : Accept text on this gadget or window
#PB_Drop_Image   : Accept images on this gadget or window
#PB_Drop_Files   : Accept filenames on this gadget or window
#PB_Drop_Private : Accept a "private" Drag & Drop on this
    gadget or window
```


'Action' specifies the action that would be taken if the user released the mouse at this point. It can be one of these values:

```
#PB_Drag_None: The data will not be accepted if dropped here
#PB_Drag_Copy: The data will be copied
#PB_Drag_Move: The data will be moved
#PB_Drag_Link: The data will be linked
```

By returning `#True`, the callback allows the action to take place at this point. By returning `#False`, the callback denies the action (the cursor will be changed to a "forbidden" cursor by the drag source, except on Linux GTK). Especially if 'State' is `#PB_Drag_Finish`, returning `#False` will cause the whole Drag & Drop operation to fail.

Return value

None.

See Also

SetDragCallback()

113.20 ExamineDraggedItems

Syntax

```
ExamineDraggedItems()
```

Description

Start to examine the dragged items with the functions `NextDraggedItem()` and `DraggedItemIndex()`. It has to be used after a `#PB_Event_DragStart`. This function is supported for the following gadgets: `ExplorerListGadget()`, `ExplorerTreeGadget()`, `ListIconGadget()`, `ListViewGadget()` and `TreeGadget()`.

Return value

None.

Example

```
1  Procedure DragStartHandler()
2
3      ; Start to examine the dragged items. Have to be after a
      #PB_EventType_DragStart event
4
5      ExamineDraggedItems()
6
7      ; Iterate over the dragged items
8
9      While NextDraggedItem()
10         Debug "Dragged item: " + GetGadgetText(0) +
            GetGadgetItemText(0, DraggedItemIndex())
11     Wend
12 EndProcedure
13
14 ; Select some files or folders and drag them to another
    application
```

```

15 ;
16 If OpenWindow(0, 200, 200, 400, 400, "Drag & Drop")
17     ExplorerListGadget(0, 10, 10, 380, 380, GetHomeDirectory(),
18         #PB_Explorer_MultiSelect)
19     ; BindGadgetEvent() is required to handle the drag start event
20     BindGadgetEvent(0, @DragStartHandler(), #PB_EventType_DragStart)
21
22     Repeat
23         Event = WaitWindowEvent()
24     Until Event = #PB_Event_CloseWindow
25 EndIf

```

See Also

NextDraggedItem() , DraggedItemIndex()

113.21 NextDraggedItem

Syntax

```
Result = NextDraggedItem()
```

Description

This function must be called after ExamineDraggedItems() . It will iterate over dragged items.

Return value

Returns nonzero if another dragged item is available or zero if there are no more dragged item.

See Also

ExamineDraggedItems() , DraggedItemIndex()

113.22 DraggedItemIndex

Syntax

```
Result = DraggedItemIndex()
```

Description

Return the current dragged item index. This function has to be called after NextDraggedItem() .

Return value

Returns the index of the current dragged item (first index is zero).

See Also

ExamineDraggedItems() , NextDraggedItem()

Chapter 114

Engine3D

Overview

PureBasic provides easy access to a very powerful OpenSource 3D Engine called OGRE. This choice has been made because we don't like to reinvent the wheel every time and when we saw the OGRE code quality, we felt that it would make a considerable contribution to the PureBasic 3D library.

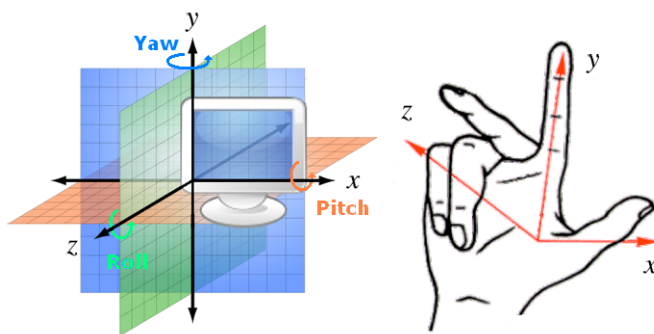
OGRE is still under heavy development and will probably become a very advanced engine soon. Its community is growing more and more and many tools are available to make the most of this engine (e.g. Blender, Lightwave and 3DS Max exporters, Terrain manager etc.). More information about OGRE is available at [OGRE web site](#). The OGRE license file can be consulted here (read more in [Licensing FAQ](#)).

Note: If you use the 3D engine of PureBasic in your projects and you intent to distribute your executable, you will have to copy the Engine3D.dll from the PureBasic/Compilers directory to your main project directory.

Spatial conventions

OGRE uses the following spatial conventions:

The spatial reference of OGRE is direct and the rule of the 'three fingers' of the right hand allows us to find easily the position of the axes X, Y and Z in the 3D World.



Attention, the Z axis is facing you.

Definition of: Roll, Pitch and Yaw:

- Pitch is a rotation around the x axis.
- Yaw is a rotation around the y axis.
- Roll is a rotation around the z axis.

Positive direction:

The positive direction for a rotation around an axis.

Thumb in the direction of the axis as indicated, the direction of the fingers gives the positive direction of the rotation.

(Use your right hand).



Important note:

In all 3D libraries supplied with PureBasic, all variables and returned values are Float-type except for IDs and indexes, even if not indicated by a '.f'.

114.1 Add3DArchive

Syntax

```
Add3DArchive(Path$ , Type)
```

Description

Add a new absolute or relative path to the current 3D path list. All the 3D functions which need to load data (e.g. texture , mesh , sky , world) will use this path. It needs to be put after `OpenScreen()` or `OpenWindowedScreen()` .

You may wonder why there is a special way to handle the files for 3D functions? Because it makes the data access more flexible and now, you can access the data using the standard filesystem or directly via the archive file like a .zip file. It's a very cool feature because you can pack all your data in one big compressed file, and then access the assets as if these were contained a standard directory.

Parameters

Path\$ The path to add to the 3D path list. It can be a real filesystem path, or a ZIP archive.

Type It can be one of the following value:

```
#PB_3DArchive_FileSystem : Standard directory
#PB_3DArchive_Zip       : Compressed zip file
```

Return value

None.

Example

```
1 Add3DArchive("MyData.zip", #PB_3DArchive_Zip) ; Add the .zip as a
   directory in the path
2 LoadTexture(0, "MyTexture.jpg")              ; load the
   MyTexture.jpg from the zip
3 LoadTexture(1, "World/Grass.jpg")             ; load the
   Grass.jpg from the zip, in the World\ path
```

114.2 AmbientColor

Syntax

```
AmbientColor (Color)
```

Description

Changes the ambient color of the world.

Parameters

Color The new ambient color to set. RGB() can be used to get a valid Color value.

Return value

None.

114.3 AntialiasingMode

Syntax

```
AntialiasingMode (Mode)
```

Description

Changes the fullscreen antialiasing mode. This function has to be called before OpenScreen() to have any effect.

Parameters

Mode It can be one of the following constants:

```
#PB_AntialiasingMode_None: No antialiasing (default).  
#PB_AntialiasingMode_x2  : x2 fullscreen antialiasing (FSAA).  
#PB_AntialiasingMode_x4  : x4 fullscreen antialiasing (FSAA).  
#PB_AntialiasingMode_x6  : x6 fullscreen antialiasing (FSAA).
```

Remarks

Depending of the graphic card, this command can have a severe impact on rendering performance.

Return value

None.

114.4 ConvertLocalToWorldPosition

Syntax

```
ConvertLocalToWorldPosition (ObjectID, x, y, z)
```

Description

Converts the local x,y,z coordinates into world coordinates. GetX() , GetY() and GetZ() will be used to get the converted coordinates.

Parameters

ObjectID The object ID to convert the coordinate from. It can be one of the following type:

```
- Camera          : use CameraID()
to get a valid ID.
- Entity          : use EntityID()
to get a valid ID.
- Light           : use LightID()
to get a valid ID.
- Node            : use NodeID()
to get a valid ID.
- ParticleEmitter: use ParticleEmitterID()
to get a valid ID.
- BillboardGroup  : use BillboardGroupID()
to get a valid ID.
- Text3D          : use Text3DID()
to get a valid ID.
```

x, y, z The local coordinates to convert.

Return value

None.

See Also

ConvertWorldToLocalPosition()

114.5 ConvertWorldToLocalPosition

Syntax

```
ConvertWorldToLocalPosition(ObjectID, x, y, z)
```

Description

Converts the world x,y,z coordinates into local coordinates. GetX() , GetY() and GetZ() will be used to get the converted coordinates.

Parameters

ObjectID The object ID to convert the coordinate from. It can be one of the following type:

```
- Camera          : use CameraID()
to get a valid ID.
- Entity          : use EntityID()
to get a valid ID.
- Light           : use LightID()
to get a valid ID.
- Node            : use NodeID()
to get a valid ID.
- ParticleEmitter: use ParticleEmitterID()
to get a valid ID.
- BillboardGroup  : use BillboardGroupID()
to get a valid ID.
- Text3D          : use Text3DID()
to get a valid ID.
```

x, y, z The world coordinates to convert.

Return value

None.

See Also

`ConvertLocalToWorldPosition()`

114.6 Engine3DStatus

Syntax

```
Result = Engine3DStatus(Type)
```

Description

Gets the 3D engine current status.

Parameters

Type This can be one of the following values:

```
#PB_Engine3D_NbRenderedTriangles: Nb rendered triangles in
the last frame.
#PB_Engine3D_NbRenderedBatches  : Nb rendered batches in the
last frame.
#PB_Engine3D_CurrentFPS         : Current frame rate.
#PB_Engine3D_AverageFPS         : Average frame rate achieved
since the engine started running.
#PB_Engine3D_MaximumFPS         : Best frame rate achieved
since the engine started running.
#PB_Engine3D_MinimumFPS         : Worst frame rate achieved
since the engine started running.
#PB_Engine3D_ResetFPS           : Reset all the frame rate
related statistics.
```

Return value

The value depending of the specified 'Type'.

114.7 EnableWorldCollisions

Syntax

```
EnableWorldCollisions(State)
```

Description

Enable or disable the collisions in the world.

Parameters

State If nonzero, the collisions will be enabled (default). The collisions is applied to all the entities which have a body set with `CreateEntityBody()` . Collisions only work if the physics engine is enabled with `EnableWorldPhysics()` .

Return value

None.

See Also

`EnableWorldPhysics()`

114.8 EnableWorldPhysics

Syntax

```
EnableWorldPhysics(State)
```

Description

Enable or disable the physics engine in the world.

Parameters

State If nonzero, the physics engine will be enabled (default). The physics engine is applied to all the entities which have a body set with `CreateEntityBody()` . Collisions can be enabled with `EnableWorldCollisions()` . To have working collisions, the physics engine has to be enabled.

Return value

None.

See Also

`EnableWorldCollisions()`

114.9 ExamineWorldCollisions

Syntax

```
Result = ExamineWorldCollisions(Contacts)
```

Description

Examine the collisions which have occurred in the world since the last call. Collisions have to be enabled with `EnableWorldCollisions()` before using this command. To step through the collisions, use `NextWorldCollision()` .

Parameters

Contacts If set the `#True`, contact information about the colliding objects are collected and can be retrieved with `WorldCollisionContact()` , `WorldCollisionNormal()` and `WorldCollisionAppliedImpulse()` . If set to `#False`, no contact information are collected (faster).

Return value

Returns nonzero if the collisions can be examined, returns zero otherwise.

See Also

`EnableWorldCollisions()` , `NextWorldCollision()`

114.10 NextWorldCollision

Syntax

```
Result = NextWorldCollision()
```

Description

Go to the next collision. `ExamineWorldCollisions()` needs to be called successfully before using this command. To get more information about the current collision, use `FirstWorldCollisionEntity()` , `SecondWorldCollisionEntity()` , `WorldCollisionContact()` , `WorldCollisionNormal()` and `WorldCollisionAppliedImpulse()` .

Return value

Returns nonzero if there is another collision to be examined, or zero if there is no more collisions.

See Also

`ExamineWorldCollisions()` , `FirstWorldCollisionEntity()` , `SecondWorldCollisionEntity()` , `WorldCollisionContact()` , `WorldCollisionNormal()` , `WorldCollisionAppliedImpulse()`

114.11 FirstWorldCollisionEntity

Syntax

```
Result = FirstWorldCollisionEntity()
```

Description

Returns the `#Entity` number of the first object in the collision being examined with `ExamineWorldCollisions()` .

Return value

Returns the `#Entity` number of the first object in the current collision.

See Also

`ExamineWorldCollisions()` , `SecondWorldCollisionEntity()`

114.12 SecondWorldCollisionEntity

Syntax

```
Result = SecondWorldCollisionEntity()
```

Description

Returns the #Entity number of the second object in the collision being examined with ExamineWorldCollisions() ().

Return value

Returns the #Entity number of the second object in the current collision.

See Also

ExamineWorldCollisions() , FirstWorldCollisionEntity()

114.13 WorldCollisionContact

Syntax

```
WorldCollisionContact()
```

Description

Fetch contact information about the collision being examined with ExamineWorldCollisions() (). ExamineWorldCollisions() 'Contacts' parameter has to be set to #True to have this command working. The contact vector values can be retrieved with GetX() , GetY() and GetZ() .

Return value

None.

See Also

ExamineWorldCollisions() , GetX() , GetY() , GetZ()

114.14 WorldCollisionNormal

Syntax

```
WorldCollisionNormal()
```

Description

Fetch contact normal information about the collision being examined with ExamineWorldCollisions() (). ExamineWorldCollisions() 'Contacts' parameter has to be set to #True to have this command working. The contact normal vector values can be retrieved with GetX() , GetY() and GetZ() .

Return value

None.

See Also

ExamineWorldCollisions() , GetX() , GetY() , GetZ()

114.15 WorldCollisionAppliedImpulse

Syntax

```
Result.f = WorldCollisionAppliedImpulse()
```

Description

Returns the applied impulse about the collision being examined with ExamineWorldCollisions() . ExamineWorldCollisions() 'Contacts' parameter has to be set to `#True` to have this command working.

Return value

The applied impulse about the collision being examined with ExamineWorldCollisions() .

See Also

ExamineWorldCollisions()

114.16 FetchOrientation

Syntax

```
FetchOrientation(ObjectID [, Mode])
```

Description

Get the orientation of the specified object. GetX() , GetY() , GetZ() and GetW() will be used to get orientation values.

Parameters

ObjectID The object ID to get the orientation. It can be one of the following type:

```
- Camera          : use CameraID()
to get a valid ID.
- Entity          : use EntityID()
to get a valid ID.
- Light           : use LightID()
to get a valid ID.
- Mesh            : use MeshID()
to get a valid ID.
- Node            : use NodeID()
to get a valid ID.
- ParticleEmitter : use ParticleEmitterID()
to get a valid ID.
- BillboardGroup  : use BillboardGroupID()
to get a valid ID.
- Text3D          : use Text3DID()
to get a valid ID.
```

Mode (optional) The mode to get the orientation. It can be one of the following value:

```
#PB_Absolute: get the absolute orientation of the object in
               the world (default).
#PB_Relative: get the orientation of the object relative to
               its parent.
```

Return value

None.

See Also

GetX() , GetY() , GetZ() , GetW() , SetOrientation()

114.17 SetOrientation

Syntax

```
SetOrientation(ObjectID, x, y, z, w)
```

Description

Set the orientation of the specified object.

Parameters

ObjectID The object ID to set the orientation. It can be one of the following type:

```
- Camera          : use CameraID()
to get a valid ID.
- Entity          : use EntityID()
to get a valid ID.
- Light           : use LightID()
to get a valid ID.
- Mesh            : use MeshID()
to get a valid ID.
- Node            : use NodeID()
to get a valid ID.
- ParticleEmitter : use ParticleEmitterID()
to get a valid ID.
- BillboardGroup  : use BillboardGroupID()
to get a valid ID.
- Text3D          : use Text3DID()
to get a valid ID.
```

x, y, z, w The new orientation value.

Return value

None.

See Also

FetchOrientation()

114.18 GetX

Syntax

```
Result = GetX()
```

Description

Returns the x value of the last called command. Supported commands are FetchOrientation() , ConvertLocalToWorldPosition() and ConvertWorldToLocalPosition() .

Return value

Returns the x value of the last called command.

See Also

GetY() , GetZ() , GetW() , FetchOrientation() , ConvertLocalToWorldPosition() , ConvertWorldToLocalPosition()

114.19 GetY

Syntax

```
Result = GetY()
```

Description

Returns the y value of the last called command. Supported commands are FetchOrientation() , ConvertLocalToWorldPosition() and ConvertWorldToLocalPosition() .

Return value

Returns the y value of the last called command.

See Also

GetX() , GetZ() , GetW() , FetchOrientation() , ConvertLocalToWorldPosition() , ConvertWorldToLocalPosition()

114.20 GetZ

Syntax

```
Result = GetZ()
```

Description

Returns the z value of the last called command. Supported commands are FetchOrientation() , ConvertLocalToWorldPosition() and ConvertWorldToLocalPosition() .

Return value

Returns the z value of the last called command.

See Also

GetX() , GetY() , GetW() , FetchOrientation() , ConvertLocalToWorldPosition() ,
ConvertWorldToLocalPosition()

114.21 GetW

Syntax

```
Result = GetW()
```

Description

Returns the w value of the last called command. The only supported command is
FetchOrientation() .

Return value

Returns the w value of the last called command.

See Also

GetX() , GetY() , GetZ() , FetchOrientation()

114.22 Fog

Syntax

```
Fog(Color , Intensity , StartDistance , EndDistance)
```

Description

Creates a fog at the specified distance of the camera. The fog effect is applied to all cameras. The fog is also applied automatically to the SkyBox() and SkyDome() commands if called before them.

Parameters

Color The fog color. RGB() can be used to get a valid color value.

Intensity The fog intensity. If sets to zero, the fog effect is disable.

StartDistance The distance from the camera (in world units) where the fog should start.

EndDistance The distance from the camera where the fog is fully opaque.

Return value

None.

See Also

SkyDome() , SkyBox()

114.23 InitEngine3D

Syntax

```
Result = InitEngine3D([Flags [, LibraryName$])
```

Description

Initializes the 3D environment for later use. You must put this function at the top of your source code if you want to use any of the 3D functions.

Parameters

Flags (optional) It can be a combination of the following values:

```
#PB_Engine3D_NoLog      : No log will be written to disk or  
                        printed to the console (default).  
#PB_Engine3D_DebugLog   : A debug log file named 'Ogre.log'  
                        will be created in the current directory, to help  
                        debugging or user reports. A lot of  
                        actions are logged in this file, but it shouldn't  
                        affect performances so it can even  
                        be activated for a release product.  
#PB_Engine3D_DebugOutput: The actions are printed to the  
                        console. You must remember to set the executable format  
                        to 'Console' when you are compiling  
                        your programs.
```

LibraryName\$ (optional) Name of the Engine3D file which should be loaded. If it is put to another place than the current directory, it could be specified here.

Return value

Returns non-zero if the library has been successfully loaded, zero otherwise. If initialization fails, the program should quit or all 3D function calls should be disabled.

Remarks

InitEngine3D() tries to load the Engine3D library (named 'Engine3D.dll' on Windows, 'engine3d.so' on Linux and 'engine3d.dylib' on OS X, found in the PureBasic/compilers/ directory). If it fails, it's probably because the library is not found or doesn't match the current PureBasic version. On Windows, a recent version of OpenGL drivers needs to be installed. After calling this function, you will not be able to use the ScreenOutput() function.

See Also

OpenScreen() , OpenWindowedScreen() , Add3DArchive()

114.24 LoadWorld

Syntax

```
Result = LoadWorld(Filename$)
```

Description

This function loads an entire world. Currently, the Quake3 BSP format is the only one supported but more formats will follow. The Filename\$ must be accessible in the 3D path, so the Add3DArchive() function should be used before calling this function. A world can be easily created using third party tools like 'Quark'. A world can contain a sky, building, lights and more. Actually, all Quake 3 maps can be loaded out of the box with full detail. If the 'Result' is 0, the world could not be loaded.

The BSP format is owned by iD Software and can only be used in freeware programs. Commercial software must acquire a license from iD to use this format. The license scheme is beyond PureBasic and Fantaisie Software can't be responsible for the incorrect use of this function.

114.25 MousePick

Syntax

```
Result = MousePick(#Camera, x, y [, PickMask])
```

Description

Simulates a mouse click and returns which object is under the specified 2D point on the specified camera.

Parameters

#Camera The camera to use.

x, y The coordinates, in pixels, of the point.

PickMask (optional) The entity mask used while performing the ray cast. Only the entities with a mask matching the PickMask value will be reported. If this parameter is omitted, all the entities are valid for the ray detection. The mask can be a combination, to select more than one entity group. To have more information about pick mask, see CreateEntity() .

Return value

The returned value can be:

```
-1 : Nothing has been detected
#PB_World_WaterPick: the click occurred on the water.
0 and above: the click occurred on the #Entity. The entity pick
is based on its bounding box, not on its mesh.
```

To get more information about the picked object position, use PickX() , PickY() and PickZ() .

114.26 PointPick

Syntax

```
Result = PointPick(#Camera, x, y)
```

Description

Allows to get the direction the specified 2D point on the specified camera.

Parameters

#Camera The camera to use.

x, y The coordinates, in pixels, of the point.

Return value

Returns non-zero if the pick was successful. To get the point direction, use `PickX()` , `PickY()` and `PickZ()` .

114.27 BodyPick

Syntax

```
Result = BodyPick(#Camera, Picked, x, y, Locked)
```

Description

Simulates a mouse click and starts the manipulate the entity at the specified coordinate.

Parameters

#Camera The camera to use.

Picked If **#True**, the picked body won't be released until a new function call with this parameter sets to **#False**. It can only have one picked body at the same time.

x, y The coordinates, in pixels, of the point.

Locked If set to **#True**, the picked body won't be able to freely rotate when moving it. If set to **#False**, the body will rotate freely according to body move.

Return value

The **#Entity** picked, or -1 if no entity are found at these coordinates.

114.28 PickX

Syntax

```
Result.f = PickX()
```

Description

After `MousePick()` or `RayPick()` , it returns the 'x' position of the picked object in world coordinates.

After `PointPick()` , it returns the 'x' direction of the picked point, between -1 and 1.

114.29 PickY

Syntax

```
Result.f = PickY()
```

Description

After MousePick() or RayPick() , it returns the 'y' position of the picked object in world coordinates.

After PointPick() , it returns the 'y' direction of the picked point, between -1 and 1.

114.30 PickZ

Syntax

```
Result.f = PickZ()
```

Description

After MousePick() or RayPick() , it returns the 'z' position of the picked object in world coordinates.

After PointPick() , it returns the 'z' direction of the picked point, between -1 and 1.

114.31 RayCollide

Syntax

```
Result = RayCollide(x, y, z, DestinationX, DestinationY,  
    DestinationZ [, CollisionGroup , CollisionMask])
```

Description

Casts a ray between the first point and the second point, and checks if an entity is colliding the ray. This function relies on the physic engine, which needs to be activated with EnableWorldPhysics() before using this command. Only entities with bodies will react to the ray. To get the position about the collide point, use PickX() , PickY() and PickZ() . The normals values at the collide point are available with NormalX() , NormalY() and NormalZ() .

Parameters

x, y, z The first point coordinates, in world unit.

DestinationX, DestinationY, DestinationZ The second point coordinates, in world unit.

CollisionGroup, CollisionMask (optional) The collision group and collision mask to use. Can be useful to filter which entity should collide to the ray. Collision group and mask can be changed with SetEntityCollisionFilter() .

Return value

Returns the entity number if the ray has collided with one, or -1 if no collision has occurred.

See Also

NormalX() , NormalY() , NormalZ() , SetEntityCollisionFilter()

114.32 RayCast

Syntax

```
Result = RayCast(x, y, z, DestinationX, DestinationY, DestinationZ,  
                PickMask)
```

Description

Casts a ray between the first point and the second point, and checks if an object is crossing the ray. This function doesn't rely on the physic engine. The normals value at the impact point are available with NormalX() , NormalY() and NormalZ() .

Parameters

x, y, z The first point coordinates, in world unit.

DestinationX, DestinationY, DestinationZ The second point coordinates, in world unit.

PickMask The entity mask used while performing the ray cast. Only the entities with a mask matching the PickMask value will be reported. If this parameter is omitted, all the entities are valid for the ray detection. The mask can be a combination, to select more than one entity group. To have more information about pick mask, see CreateEntity() .

Return value

Returns -1 if the ray has collided with any object.

See Also

NormalX() , NormalY() , NormalZ()

114.33 MouseRayCast

Syntax

```
Result = MouseRayCast(#Camera, x, y, PickMask)
```

Description

Casts a ray from the 2D point through the scene, and checks if an object is crossing the ray. This function doesn't rely on the physic engine. The normals value at the impact point are available with NormalX() , NormalY() and NormalZ() .

Parameters

#Camera The camera to use.

x, y The coordinates, in pixels, of the point.

PickMask The entity mask used while performing the ray cast. Only the entities with a mask matching the PickMask value will be reported. If this parameter is omitted, all the entities are valid for the ray detection. The mask can be a combination, to select more than one entity group. To have more information about pick mask, see CreateEntity() .

Return value

Returns the number of the object in contact with the ray, -1 otherwise.

See Also

NormalX() , NormalY() , NormalZ()

114.34 NormalX

Syntax

```
Result.f = NormalX()
```

Description

Returns the 'x' normal value at the crossed point, after RayCast() , RayCollide() or MouseRayCast() .

Return value

Returns the 'x' normal value at the crossed point.

See Also

RayCast() , RayCollide() , MouseRayCast() , NormalY() , NormalZ()

114.35 NormalY

Syntax

```
Result.f = NormalY()
```

Description

Returns the 'y' normal value at the crossed point, after RayCast() , RayCollide() or MouseRayCast() .

Return value

Returns the 'y' normal value at the crossed point.

See Also

RayCast() , RayCollide() , MouseRayCast() , NormalX() , NormalZ()

114.36 NormalZ

Syntax

```
Result.f = NormalZ()
```

Description

Returns the 'z' normal value at the crossed point, after RayCast() , RayCollide() or MouseRayCast() .

Return value

Returns the 'z' normal value at the crossed point.

See Also

RayCast() , RayCollide() , MouseRayCast() , NormalX() , NormalY()

114.37 RayPick

Syntax

```
Result = RayPick(x, y, z, DestinationX, DestinationY, DestinationZ  
[, PickMask])
```

Description

Casts a ray between the first point and the second point, and checks if an object is crossing the ray.

Parameters

x, y, z The first point coordinates, in world unit.

DestinationX, DestinationY, DestinationZ The second point coordinates, in world unit.

PickMask (optional) The entity mask used while performing the ray pick. Only the entities with a mask matching the PickMask value will be reported. If this parameter is omitted, all the entities are valid for the ray detection. The mask can be a combination, to select more than one entity group. To have more information about pick mask, see CreateEntity() .

Return value

The returned value can be:

```
-1 : Nothing has been crossed.  
#PB_World_WaterPick: the ray crossed the water.  
0 and above: the ray crossed an #Entity. The entity detection is  
based on its bounding box, not on its mesh.
```

To get more information about the picked object position, use PickX() , PickY() and PickZ() .

114.38 ShowGUI

Syntax

```
ShowGUI(Transparency, ShowMouse [, #Camera, Enable])
```

Description

Shows or hides the whole GUI elements, which are composed of 3d windows and 3d gadgets .

Parameters

Transparency Transparency level of the GUI. Valid values are between 0 (hidden) and 255 (fully opaque).

ShowMouse Change the mouse cursor visibility. If ShowMouse = 1, the mouse cursor will be displayed, if ShowMouse = 0 it will be hidden.

#Camera (optional) If set the GUI will be displayed or not on this camera. Works together with the following 'Enable' parameter.

Enable (optional) Enable or disable the GUI display on the previously selected camera. Works together with the previous '#Camera' parameter.

Return value

None.

114.39 SetGUITheme3D

Syntax

```
SetGUITheme3D(ThemeName$, FontName$)
```

Description

As CEGUI support skinning, this command allow to select which theme and which font to use for the GUI. This command has to be called before any other GUI commands to have an effect.

Parameters

ThemeName\$ The new theme to use, specified without the '.scheme' extension.

FontName\$ The new font to use, specified without the '.font' extension.

Return value

None.

Remarks

For more information about skins, visit the [CEGUI web site](#).

114.40 Parse3DScripts

Syntax

```
Parse3DScripts()
```

Description

Parses all the .materials OGRE scripts found in the paths set with Add3DArchive(). This allows the use of meshes with complex materials scripts directly in PureBasic. When creating the entity the constant `#PB_Material_None` has to be specified, so all the material information will be taken from the script, if it was correctly loaded.

Return value

None.

Remarks

More information about OGRE material scripts can be found at [OGRE online manual](#).

114.41 RenderWorld

Syntax

```
Result = RenderWorld([ElapsedPhysicTime])
```

Description

Renders the whole world on the screen. This function should be called once all 3D operations are finished and only one time per frame.

Parameters

ElapsedPhysicTime (optional) If set it will force the physic engine to use this value, in milliseconds, as elapsed time since the last call of RenderWorld. It can be useful to makes the physic simulation faster or slower than real-time.

Return value

Returns the elapsed time since the last frame in milliseconds. It can be useful to have an accurate time based simulation if the frame render time isn't stable.

Remarks

Once the RenderWorld() function has been performed, it's possible to use regular 2D functions like DisplaySprite() to display 2D sprites over the 3D world.

114.42 SetRenderQueue

Syntax

```
SetRenderQueue(ObjectID, Queue [, Priority])
```

Description

Change the render order of the specified object.

Parameters

ObjectID The object ID to convert the coordinate from. It can be one of the following type:

```
- Entity          : use EntityID()
to get a valid ID.
- Light           : use LightID()
to get a valid ID.
- Mesh            : use MeshID()
to get a valid ID.
```

```

- ParticleEmitter: use ParticleEmitterID()
to get a valid ID.
- BillboardGroup : use BillboardGroupID()
to get a valid ID.
- Text3D          : use Text3DID()
to get a valid ID.

```

Queue The queue number to use to render the object. Queue number can be from 0 (background) to 100 (foreground). The default queue is 0, and is render behind all the other render queues.

Priority (optional) The priority to use within the queue. Valid values are from 0 (background) to 10000 (foreground).

Return value

None.

114.43 SkyBox

Syntax

```

Result = SkyBox(TextureName$ [, FogColor, FogIntensity,
    FogStartDistance, FogEndDistance])

```

Description

Loads a 6 face cube and creates an artificial box which is far away from the camera but close to the world. This is a very useful function to easily make a closed world.

Parameters

TextureName\$ The textures must be in a path previously declared with the Add3DArchive() function. The base name of the textures, which should be named using the following rule:

```

TextureName_BK ; Back face
TextureName_FR ; Front face
TextureName_DN ; Down face
TextureName_UP ; Up face
TextureName_LF ; Left face
TextureName_RT ; Right face

```

FogColor (optional) The fog color. RGB() can be used to get a valid color value. If not specified, the fog settings are inherited from the Fog() command.

FogIntensity (optional) The fog intensity. If sets to zero, the fog effect is disable.

FogStartDistance (optional) The distance from the camera (in world units) where the fog should start.

FogEndDistance (optional) The distance from the camera where the fog is fully opaque.

Return value

Returns non-zero if the SkyBox has been successfully created. If the textures can't be loaded, the skybox is created with blank textures.

See Also

SkyDome()

114.44 SkyDome

Syntax

```
Result = SkyDome(TextureID, SkyColor, RiseColor [, NbCloudLayers,
            Height.f, SpeedX.f, SpeedZ.f])
```

Description

Creates a new SkyDome which is a curved moving sky displayed using the specified cloud texture.

Parameters

TextureID The cloud texture id to use. TextureID() can be used to get a valid value.

SkyColor The sky color as RGBA() .

RiseColor The sun rising color as RGBA() .

NbCloudLayers (optional) The number of clouds layer, starting from 1.

Height (optional) The clouds height in world unit.

SpeedX, SpeedZ (optional) The clouds moving speed

Return value

Returns non-zero if the SkyDome has been successfully created.

Example

```
1  #EndDistance = 1024*4
2
3  InitEngine3D() : InitSprite() : InitKeyboard()
4
5  ExamineDesktops() : dx = DesktopWidth(0)*0.9 : dy =
    DesktopHeight(0)*0.9
6  OpenWindow(0, 0, 0, DesktopUnscaledX(dx), DesktopUnscaledY(dy),
    "SkyDome - Press ESC to quit", #PB_Window_ScreenCentered)
7  OpenWindowedScreen(WindowID(0), 0, 0, dx, dy, 0, 0, 0)
8
9  Add3DArchive(#PB_Compiler_Home + "examples/3d/Data/Textures",
    #PB_3DArchive_FileSystem)
10 Parse3DScripts()
11
12 ; Camera
13 CreateCamera(0, 0, 0, 100, 100)
14 MoveCamera(0,0,5,0)
15 CameraLookAt(0,2,5,10)
16
17 ; Sky
18 TextureSky = LoadTexture(#PB_Any, "sky.png")
19 SkyDome(TextureID(TextureSky), $cc6600, $0088ff, 3, 400, -0.5, 0)
20
21 ; Sun
22 CreateLight(0, $ff88ff, 20000, 40000, 20000)
23 AmbientColor($010101)
24 Fog($554488,1, 0, #EndDistance)
25
```

```

26 Repeat
27     While WindowEvent() : Wend
28     ExamineKeyboard()
29
30     RenderWorld()
31     FlipBuffers()
32 Until KeyboardReleased(#PB_Key_Escape)

```

See Also

SkyBox()

114.45 CreateWater

Syntax

```
CreateWater(WaveTextureID, FoamTextureID, WaterColor, SkyColor,
            Distance.f, WaveBig.f, WaveLittle.f, Swell.f, Foam.f)
```

Description

Creates a world wide water plane.

Return value

None.

See Also

SkyDome() , SkyBox()

114.46 WorldShadows

Syntax

```
WorldShadows(Type [, Distance.f [, Color [, TextureSize]])
```

Description

Sets how shadows will be rendered in the world.

Parameters

Type Type can be one of the following values:

```

#PB_Shadow_None      : No shadows will be displayed in the
                        world. This can save a lot of CPU
                        power if shadows aren't needed
                        (default)
#PB_Shadow_Modulative: shadows will be displayed for entities
                        that have the cast
                        shadow set with EntityRenderMode()
and #PB_Entity_CastShadow.

```

This shadow mode is the fastest available, but is not very good-looking as the shadows are not translucent.

`#PB_Shadow_Additive` : shadows will be displayed for the entities which have the cast shadow set with `EntityRenderMode()` and `#PB_Entity_CastShadow`.

This mode is slower than the modulative mode, but it looks much better as the shadows are translucent.

Also if some shadows overlap, the shadows will be added resulting in a more realistic darker result.

`#PB_Shadow_TextureAdditive`: shadows will be displayed for the entities which have the cast shadow set with `EntityRenderMode()` and `#PB_Entity_CastShadow`. This mode is average in speed and quality between modulative and additive shadow, and will work even with water.

`#PB_Shadow_TextureModulative`: shadows will be displayed for the entities which have the cast shadow set with `EntityRenderMode()` and `#PB_Entity_CastShadow`.

This shadow mode is the faster than `#PB_Shadow_TextureAdditive` but is not very good-looking as the shadows are not translucent.

Distance (optional) Maximum distance from the camera, in world unit, above which the shadows won't be calculated anymore.

Color (optional) Color of the shadows. `RGB()` can be used to get a valid Color value.

TextureSize (optional) Pixel size of the texture used to render the shadow. The bigger it is it, the better the shadow will be looking, but the slower it will be. The default value is 512, and this value shouldn't be bigger than 4096.

Return value

None.

114.47 WorldGravity

Syntax

```
WorldGravity(Gravity.f [, x, y, z])
```

Description

Changes the gravity of the world when the physics engine is enabled with `EnableWorldPhysics()` .

Parameters

Gravity The new gravity to set. The default gravity value is set to -9.806 (which is Earth's gravitational value).

x, y, z (optional) The gravity force vector to use, to change the gravity direction.

Return value

None.

See Also

EnableWorldPhysics()

114.48 WorldDebug

Syntax

```
WorldDebug(Mode)
```

Description

Changes the world debug mode. This can be very useful to help finding issues with collisions or picking for example.

Parameters

Mode Can be a combination of the following constants:

```
#PB_World_DebugNone    : no debug information (default).
#PB_World_DebugEntity : shows the entities bounding boxes.
#PB_World_DebugBody    : shows the physics bodies, both static
                        and dynamic.
```

Return value

None.

114.49 Pitch

Syntax

```
Pitch(ObjectID, Value.f, Mode)
```

Description

Pitch the specified object.

Parameters

ObjectID The entity to pitch. It can be one of the following type:

```
- Camera          : use CameraID()
to get a valid ID.
- Entity          : use EntityID()
to get a valid ID.
- Light           : use LightID()
to get a valid ID.
- Mesh            : use MeshID()
```

```

to get a valid ID.
- Node : use NodeID()
to get a valid ID.
- ParticleEmitter: use ParticleEmitterID()
to get a valid ID.
- BillboardGroup : use BillboardGroupID()
to get a valid ID.
- Text3D : use Text3DID()
to get a valid ID.

```

Value The pitch value (in degree).

Mode The pitch mode. It can be one of the following values:

```

#PB_Relative: relative pitch, from the current pitch.
#PB_Absolute: absolute pitch (default).

```

combined with one of the following values:

```

#PB_Local : local pitch.
#PB_Parent: pitch relative to the parent.
#PB_World : pitch relative to the world.

```

Return value

None.

See Also

Roll() , Yaw()

114.50 Roll

Syntax

```
Roll(ObjectID, Value.f, Mode)
```

Description

Roll the specified object.

Parameters

ObjectID The entity to roll. It can be one of the following type:

```

- Camera : use CameraID()
to get a valid ID.
- Entity : use EntityID()
to get a valid ID.
- Light : use LightID()
to get a valid ID.
- Mesh : use MeshID()
to get a valid ID.
- Node : use NodeID()
to get a valid ID.
- ParticleEmitter: use ParticleEmitterID()
to get a valid ID.
- BillboardGroup : use BillboardGroupID()

```

```

to get a valid ID.
- Text3D          : use Text3DID()
to get a valid ID.

```

Value The roll value (in degree).

Mode The roll mode. It can be one of the following values:

```

#PB_Relative: relative roll, from the current roll.
#PB_Absolute: absolute roll (default).

```

combined with one of the following values:

```

#PB_Local  : local roll.
#PB_Parent: roll relative to the parent.
#PB_World  : roll relative to the world.

```

Return value

None.

See Also

Pitch() , Yaw()

114.51 Yaw

Syntax

```
Yaw(ObjectID, Value.f, Mode)
```

Description

Yaw the specified object.

Parameters

ObjectID The entity to yaw. It can be one of the following type:

```

- Camera          : use CameraID()
to get a valid ID.
- Entity          : use EntityID()
to get a valid ID.
- Light           : use LightID()
to get a valid ID.
- Mesh            : use MeshID()
to get a valid ID.
- Node            : use NodeID()
to get a valid ID.
- ParticleEmitter: use ParticleEmitterID()
to get a valid ID.
- BillboardGroup : use BillboardGroupID()
to get a valid ID.
- Text3D          : use Text3DID()
to get a valid ID.

```

Value The yaw value (in degree).

Mode The yaw mode. It can be one of the following values:

```
#PB_Relative: relative yaw, from the current yaw.  
#PB_Absolute: absolute yaw (default).
```

combined with one of the following values:

```
#PB_Local : local yaw.  
#PB_Parent: yaw relative to the parent.  
#PB_World : yaw relative to the world.
```

Return value

None.

See Also

Pitch() , Roll()

114.52 GetWorldAttribute

Syntax

```
Result.f = GetWorldAttribute(Attribute)
```

Description

Get the specified attribute of the world.

Parameters

Attribute The attribute value to get. The following attributes are available:

```
#PB_Shadow_FarDistance    : get shadow far distance.  
#PB_Shadow_Color          : get the current shadow color as  
    RGBA()  
.  
#PB_SkyDome_SkyColor      : get the color of the sky as RGBA()  
.  
#PB_SkyDome_RiseColor     : get the sun rise color as RGBA()  
.  
#PB_SkyDome_NbCloudLayers : get the number of cloud layer.  
#PB_SkyDome_CloudsHeight  : get the height of the clouds in  
    world unit.  
#PB_Water_WaterColor      : get the water color as RGBA()  
.  
#PB_Water_SkyColor        : get the sky color as reflected in  
    the water as RGBA()  
.  
#PB_Water_WaveHeight      : get the wave height.  
#PB_Water_WaveSmall       : get the wave riddle amplitude.  
#PB_Water_Swell           : get the water swell intensity.  
#PB_Water_Foam            : get the foam intensity.
```

Return value

Returns the value of the specified attribute.

See Also

SetWorldAttribute()

114.53 SetWorldAttribute

Syntax

```
SetWorldAttribute(Attribute, Value.f)
```

Description

Set the specified attribute of the world.

Parameters

Attribute The attribute value to set. The following attributes are available:

<code>#PB_Shadow_FarDistance</code>	: set shadow far distance
<code>#PB_Shadow_Color</code>	: change the current shadow color
<code>#PB_SkyDome_Free</code>	: free the sky dome previously created by <code>SkyDome()</code>
<code>#PB_SkyDome_SkyColor</code>	: the color of the sky as <code>RGBA()</code>
<code>#PB_SkyDome_RiseColor</code>	: the sun rise color as <code>RGBA()</code>
<code>#PB_SkyDome_NbCloudLayers</code>	: set the number of cloud layer (value between X and X)
<code>#PB_SkyDome_CloudsHeight</code>	: set the height of the clouds in world coordinates
<code>#PB_Water_WaterColor</code>	: set the water color as <code>RGBA()</code>
<code>#PB_Water_SkyColor</code>	: set the sky color as reflected in the water as <code>RGBA()</code>
<code>#PB_Water_WaveHeight</code>	: set the wave height (value between 0 to 3).
<code>#PB_Water_WaveSmall</code>	: set the wave riddle amplitude (value between 0 to 3).
<code>#PB_Water_Swell</code>	: set the swell intensity (value between 0 to 2).
<code>#PB_Water_Foam</code>	: set the foam intensity (value between 0 to 1).
<code>#PB_Water_Free</code>	: free the world water plane previously created with <code>CreateWater()</code>

Return value

Change the value of the specified attribute.

See Also

SetWorldAttribute()

Chapter 115

Entity

Overview

Entities are objects composed of one mesh object and one material which can be freely moved and transformed in real-time. The library entity animation is available to animate an entity, based on a skeleton.

It's possible to share a mesh or a material between several entities, reducing memory consumption and saving CPU clocks.

InitEngine3D() should be called successfully before using the entity functions.

115.1 ApplyEntityForce

Syntax

```
ApplyEntityForce(#Entity, x, y, z [, PositionX, PositionY,  
    PositionZ [, Mode [, PositionMode]])
```

Description

Apply the specified force to the entity. The new force value replace any previous force previously applied to the entity.

Parameters

#Entity The entity to use.

x, y, z The force values.

PositionX, PositionY, PositionZ (optional) The position relative to the entity center where the force should be applied.

Mode (optional) The applied force mode. It can be one of the following values:

```
#PB_Local : local force.  
#PB_Parent: force relative to the parent position.  
#PB_World : force relative to the world.
```

PositionMode (optional) The applied position mode. It can be one of the following values:

```
#PB_Local : local position.  
#PB_Parent: position relative to the parent position.  
#PB_World : position relative to the world.
```

Return value

None.

See Also

`ApplyEntityImpulse()` , `ApplyEntityTorque()` , `ApplyEntityTorqueImpulse()`

115.2 ApplyEntityImpulse

Syntax

```
ApplyEntityImpulse(#Entity, x, y, z [, PositionX, PositionY,  
                    PositionZ [, Mode]])
```

Description

Apply an impulse to the entity. The new impulse is added to the current force of the entity.

Parameters

#Entity The entity to use.

x, y, z The impulse values.

PositionX, PositionY, PositionZ (optional) The position relative to the entity center where the impulse should be applied.

Mode (optional) The applied impulse mode. It can be one of the following values:

```
#PB_Local : local impulse.  
#PB_Parent: impulse relative to the parent position.  
#PB_World : impulse relative to the world.
```

Return value

None.

See Also

`ApplyEntityForce()` , `ApplyEntityTorque()` , `ApplyEntityTorqueImpulse()`

115.3 ApplyEntityTorque

Syntax

```
ApplyEntityTorque(#Entity, x, y, z [, Mode])
```

Description

Apply a rotation force to the entity. The new rotation force replace any previous force previously applied to the entity.

Parameters

#Entity The entity to use.

x, y, z The rotation force values.

Mode (optional) The applied rotation force mode. It can be one of the following values:

```
#PB_Local : local rotation force.
#PB_Parent: rotation force relative to the parent position.
#PB_World : rotation force relative to the world.
```

Return value

None.

See Also

ApplyEntityImpulse() , ApplyEntityForce() , ApplyEntityTorqueImpulse()

115.4 ApplyEntityTorqueImpulse

Syntax

```
ApplyEntityTorqueImpulse(#Entity, x, y, z [, Mode])
```

Description

Apply a rotation impulse to the entity. The new impulse is added to the rotation force previously applied to the entity.

Parameters

#Entity The entity to use.

x, y, z The rotation impulse values.

Mode (optional) The applied rotation impulse mode. It can be one of the following values:

```
#PB_Local : local rotation impulse.
#PB_Parent: rotation impulse relative to the parent position.
#PB_World : rotation impulse relative to the world.
```

Return value

None.

See Also

ApplyEntityImpulse() , ApplyEntityForce() , ApplyEntityTorque()

115.5 CopyEntity

Syntax

```
Result = CopyEntity(#Entity, #NewEntity)
```

Description

Creates a #NewEntity which is the exact copy of the specified #Entity. If #PB_Any is used as '#NewEntity' parameter, the new entity number will be returned as 'Result'.

If the 'Result' is 0, the entity copy has failed. If #NewEntity was already created, it will be automatically freed and replaced by the new one.

115.6 CreateEntity

Syntax

```
Result = CreateEntity(#Entity, MeshID, MaterialID, [x, y, z [,  
PickMask [, VisibilityMask]])
```

Description

Creates a new #Entity using the specified Mesh and Material.

Parameters

#Entity The number to identify the new entity. #PB_Any can be used to auto-generate this number.

MeshID The mesh to use to create the entity. To get a valid mesh id, use MeshID() . Dynamic meshes are not supported (meshes created with the #PB_Mesh_Dynamic flag).

MaterialID The material to use to create the entity. To get a valid material id, use MaterialID() . #PB_Material_None can be used as parameter value, to use the .material script associated to the mesh. Parse3DScripts() should be called before using #PB_Material_None.

x, y, z (optional) The position of the new entity in the world.

PickMask (optional) A special value used by RayPick() and MousePick() to select which entity group will be handled. As it's a mask, each value should be a power of two. 31 different masks are available. To create a mask value easily, the '«' operator can be used:

```
- 1 << 1 : First valid mask value  
- 1 << 2 : Second valid mask value  
- 1 << 3 : Third valid mask value  
- ...  
- 1 << 31 : Last valid mask value
```

To ease the use, constants should be used to store the mask value and used later. When calling the pick functions, masks can be combined with the '||' operator to select more than one type of entity.

VisibilityMask (optional) A mask to select on which camera the entity should be displayed. If this mask match the mask specified in CreateCamera() , the entity will be displayed on the camera. See 'PickMask' to build correct masks. If this parameter is omitted, then the entity will be visible on all cameras.

Return value

Returns zero if the entity can't be created. If #PB_Any is used as '#Entity' parameter, the new entity number is returned.

See Also

FreeEntity()

115.7 EntityFixedYawAxis

Syntax

```
EntityFixedYawAxis(#Entity, Enable [, VectorX, VectorY, VectorZ])
```

Description

Change the fixed yaw axis of the entity. The default behaviour of a entity is to yaw around its own Y axis.

Parameters

#Entity The entity to use.

Enable Enable or disable the use of a custom yaw axis. If set to **#True**, a new axis vector has to be specified. If set to **#False**, the entity will yaw around its own Y axis.

VectorX (optional) X vector direction of the new yaw axis (value between -1.0 and 1.0). 'Enable' parameter has to be set to have any effect.

VectorY (optional) Y vector direction of the new yaw axis (value between -1.0 and 1.0). 'Enable' parameter has to be set to have any effect.

VectorZ (optional) Z vector direction of the new yaw axis (value between -1.0 and 1.0). 'Enable' parameter has to be set to have any effect.

Return value

None.

115.8 EntityID

Syntax

```
EntityID = EntityID(#Entity)
```

Description

Returns the unique ID which identifies the given '#Entity' in the operating system. This function is very useful when another library needs a entity reference.

115.9 EntityLookAt

Syntax

```
EntityLookAt(#Entity, x, y, z [, DirectionX, DirectionY, DirectionZ])
```

Description

The point (in world unit) that an entity is facing. The position of the entity is not changed.

Parameters

#Entity The entity to use.

x, y, z The position (in world unit) to point the entity at.

DirectionX, DirectionY, DirectionZ (optional) The vector direction of the entity (values between -1.0 and 1.0).

Return value

None.

115.10 EntityVelocity

Syntax

```
EntityVelocity(#Entity, x, y, z)
```

Description

Changes the linear velocity of the #Entity. The linear factor is applied to the entity before any move. To get the final value, see EntityLinearFactor() for more information. The entity needs a physic body to support linear velocity. To get the current entity velocity, use GetEntityAttribute().

Parameters

#Entity The entity to use.

x, y, z The velocity vector force.

Return value

None.

115.11 EntityAngularFactor

Syntax

```
EntityAngularFactor(#Entity, x, y, z)
```

Description

Changes the angular factor of the #Entity.

Parameters

#Entity The entity to use.

x, y, z The angular factor vector.

Return value

None.

115.12 EntityLinearFactor

Syntax

```
EntityLinearFactor(#Entity, x, y, z)
```

Description

Changes the linear factor for the #Entity. When moved, the entity linear velocity is multiplied by the linear factor to get the final velocity. This is very useful to constraint an entity move on one or several axis. By default, the linear factor is 1 for all axis meaning no impact on the velocity. The entity needs a physic body to support linear velocity constraint.

Parameters

#Entity The entity to use.

x The 'x' linear factor value. If set to the 0, the entity won't be able to move on the 'x' axis anymore.

y The 'y' linear factor vector. If set to the 0, the entity won't be able to move on the 'y' axis anymore.

z The 'z' linear factor vector. If set to the 0, the entity won't be able to move on the 'z' axis anymore.

Return value

None.

115.13 EntityCustomParameter

Syntax

```
EntityCustomParameter(#Entity, SubEntity, ParameterIndex, Value1.f,  
    Value2.f, Value3.f, Value4.f)
```

Description

Set a custom parameter value to the #Entity material shader script. To have any effect, the material associated to the entity should have a shader script (either GLSL or HLSL).

Parameters

#Entity The entity to use.

SubEntity The sub-entity to use. First sub-entity index starts at 0 (representing the main entity).

ParameterIndex The parameter index in the shader script.

Value1 The first parameter value.

Value2 The second parameter value (if the parameter only accept one value, this value will be ignored).

Value3 The third parameter value (if the parameter only accept two values, this value will be ignored).

Value4 The fourth parameter value (if the parameter only accept three values, this value will be ignored).

Return value

None.

Remarks

To set parameters in a material, you need to define them in the material script as `param_named_auto 'VariableName' custom 'ParameterIndex'`, for example:

```
param_named_auto foo custom 1
param_named_auto bar custom 2
```

The definition and size of the parameter is defined in the shaders script, for example:

```
uniform float foo;
uniform vec3 bar;
```

115.14 EntityBoundingBox

Syntax

```
Result = EntityBoundingBox(#Entity, Flags)
```

Description

Returns the position of the bounding box, either in local or world coordinate.

Parameters

#Entity The entity to use.

Flags Flags can be one of the following values:

```
#PB_Entity_MinBoundingBoxX: Min 'x' position of the bounding
box
#PB_Entity_MaxBoundingBoxX: Max 'x' position of the bounding
box
#PB_Entity_MinBoundingBoxY: Min 'y' position of the bounding
box
#PB_Entity_MaxBoundingBoxY: Max 'y' position of the bounding
box
#PB_Entity_MinBoundingBoxZ: Min 'z' position of the bounding
box
#PB_Entity_MaxBoundingBoxZ: Max 'z' position of the bounding
box
```

combined with one of the following values:

```
#PB_Entity_WorldBoundingBox: Positions are returned in world
coordinates (default)
#PB_Entity_LocalBoundingBox: Positions are returned in local
coordinates
```

Return value

Returns the position of the bounding box, either in local or world coordinate.

115.15 DisableEntityBody

Syntax

```
DisableEntityBody(#Entity, Disable)
```

Description

Disable an entity body. The physic engine doesn't affect the entity anymore when an entity body is disabled.

Parameters

#Entity The entity to disable.

Disable If set to **#True**, the entity body is disabled. If set to **#False** the body is enabled.

Return value

None.

115.16 EntityParentNode

Syntax

```
Result = EntityParentNode(#Entity)
```

Description

Returns the parent NodeID() .

Parameters

#Entity The entity to use.

Return value

Returns the parent NodeID() , if any. This can be either a real node, or a bone if the entity is attached to a bone. If the entity has no parent node, it will returns 0.

115.17 FetchEntityMaterial

Syntax

```
Result = FetchEntityMaterial(#Entity, #Material [, SubEntity])
```

Description

Fetch the material associated to the specified #Entity with SetEntityMaterial() .

Parameters

#Entity The entity to use.

#Material The new create material number.

SubEntity (optional) The sub-entity to get the material from. First sub-entity index starts at 0 (representing the main entity).

Return value

Returns nonzero on success and zero on failure.

115.18 SetEntityMaterial

Syntax

```
SetEntityMaterial(#Entity, MaterialID [, SubEntity])
```

Description

Assign a material to the specified #Entity. An entity can only have one material assigned at once.

Parameters

#Entity The entity to use.

#Material The new created material number.

SubEntity (optional) The sub-entity to set the material. First sub-entity index starts at 0 (representing the main entity).

Return value

None.

See Also

FetchEntityMaterial() , GetEntityAttribute() , SetEntityAttribute()

115.19 EntityCollide

Syntax

```
Result = EntityCollide(#Entity, #Entity2)
```

Description

Checks if the two specified entities are colliding.

To have its collisions managed by the physic engine, an entity needs a body created with CreateEntityBody() .

To have any effect, the physic engine needs to be activated with the EnableWorldPhysics() .

Parameters

#Entity The first entity to test.

#Entity2 The second entity to test.

Return value

Return non-zero if the two entities are colliding.

115.20 CreateEntityBody

Syntax

```
CreateEntityBody(#Entity, Type [, Mass [, Restitution, Friction [,  
Axis [, SizeX, SizeY, SizeZ]]]])
```

Description

Changes the type of the body associated with the #Entity.

To have its collisions managed by the physic engine, an entity has to set a body. In fact, only the body is known by the physic engine, which will do all the calculation about the entity, check the mass, friction and if it collides will move back the real entity.

To have any effect, the physic engine needs to be activated with the EnableWorldPhysics() .

Parameters

Type Type defines how the physic engine will handle this entity. It can be one of the following constants:

```
#PB_Entity_None          : No body is associated to the entity  
                          (default)  
#PB_Entity_StaticBody   : The body is a static only, which  
                          means the mesh can't be animated.  
                          This mode allows very precise  
                          collisions, as it's done against triangles  
                          (also known as tri-mesh collision).  
                          It's also fast when colliding with  
                          a box or sphere entity body type.  
                          It's perfect when using a mesh for  
                          a ground or static world.  
#PB_Entity_PlaneBody    : A 'virtual' plane is set on the  
                          entity (with the same dimensions)  
                          and is used to manage the collision  
                          against the other entities.  
#PB_Entity_ConeBody     : A 'virtual' cone is set around the  
                          entity (with the same dimensions)  
                          and is used to manage the collision  
                          against the other entities.  
#PB_Entity_BoxBody      : A 'virtual' box is set around the  
                          entity (with the same dimensions)  
                          and is used to manage the collision  
                          against the other entities.  
#PB_Entity_SphereBody   : A 'virtual' sphere is set around the  
                          entity and is used to manage the collision  
                          against the other entities.  
#PB_Entity_CylinderBody : A 'virtual' cylinder is set around  
                          the entity and is used to manage the collision
```

```

                                against the other entities.
#PB_Entity_CapsuleBody : A 'virtual' capsule is set around
                        the entity and is used to manage the collision
                                against the other entities.
#PB_Entity_ConvexHullBody : A 'virtual' complex form deduced
                        from the real mesh is set around the entity
                                and is used to manage the collision
                        against the other entities. This mode is
                                slower than basic collide forms.
#PB_Entity_CompoundBody : A 'virtual' Compound body is set
                        around the entity and is used to manage the collision
                                against the other entities.

```

Mass (optional) Mass of the object. Don't use too big value or it could produce physic incoherencies (1 is the preferred value).

Restitution (optional) Restitution of the object. This value can also be get or set via `GetEntityAttribute()` and `SetEntityAttribute()`

Friction (optional) Friction of the object. This value can also be get or set via `GetEntityAttribute()` and `SetEntityAttribute()`

Axis (optional) The axis of the body. It only applies to the following body type:

```

#PB_Entity_PlaneBody
#PB_Entity_CylinderBody
#PB_Entity_CapsuleBody

```

SizeX, SizeY, SizeZ (optional) The bounding box size of the body. It only applies to the following body type:

```

#PB_Entity_BoxBody      : SizeX, SizeY and SizeZ are available.
#PB_Entity_SphereBody   : SizeX is available.
#PB_Entity_ConeBody     : SizeX and SizeY are available.
#PB_Entity_CylinderBody : SizeX and SizeY are available.
#PB_Entity_CapsuleBody  : SizeX, SizeY and SizeZ are available.

```

Return value

None.

See Also

`FreeEntityBody()`

115.21 EntityRenderMode

Syntax

```
EntityRenderMode(#Entity, Mode)
```

Description

Changes the render mode of the specified entity.

Parameters

#Entity The entity to use.

Mode Can be a combination (using the '||' operator) of the following values:

```
#PB_Entity_CastShadow      : casts entity shadow if the
WorldShadows()
has been activated (default)
#PB_Entity_DisplaySkeleton : displays the entity skeleton
#PB_Shadow_None           : turns off shadow casting for
individual mesh/object entity (useful with "grounds")
```

Return value

None.

115.22 AttachEntityObject

Syntax

```
AttachEntityObject(#Entity, Bone$, ObjectID[, x, y, z, Pitch,
Roll, Yaw])
```

Description

Attach an existing object to an entity bone. An object can be detached from an entity with DetachEntityObject() .

Parameters

#Entity The entity to use.

Bone\$ The bone name in the OGRE mesh, or as specified in CreateBone() . If the bone name is empty, then the object is not attached to a bone but directly to the entity.

ObjectID The object to attach. It can be one of the following types:

```
- Entity : use EntityID()
as 'ObjectID'.
- Camera : use CameraID()
as 'ObjectID'.
- Light  : use LightID()
as 'ObjectID'.
- BillboardGroup : use BillboardGroupID()
as 'ObjectID'.
- ParticleEmitter: use ParticleEmitterID()
as 'ObjectID'.
- Node    : use NodeID()
as 'ObjectID'.
- Text3D  : use Text3DID()
as 'ObjectID'.
```

x, y, z (optional) The relative offset position of the attached object.

Pitch (optional) The pitch to apply to the attached object.

Roll (optional) The roll to apply to the attached object.

Yaw (optional) The yaw to apply to the attached object.

Return value

None.

See Also

DetachEntityObject()

115.23 DetachEntityObject

Syntax

```
DetachEntityObject(#Node, ObjectID)
```

Description

Detach a previously attached object from an #Entity bone. The supported objects are the following:

```
- Entity : use EntityID()
as 'ObjectID'.
- Camera : use CameraID()
as 'ObjectID'.
- Light   : use LightID()
as 'ObjectID'.
- BillboardGroup : use BillboardGroupID()
as 'ObjectID'.
- ParticleEmitter: use ParticleEmitterID()
as 'ObjectID'.
```

An object can be attached to an entity bone with AttachEntityObject() .

115.24 EnableManualEntityBoneControl

Syntax

```
EnableManualEntityBoneControl(#Entity, Bone$, State,
                               InheritOrientation)
```

Description

Enable the manual control of a bone. It can be manually moved with MoveEntityBone() , rotated with RotateEntityBone() or scaled with ScaleEntityBone() .

Parameters

#Entity The entity to use.

Bone\$ The bone name in the OGRE mesh, or as specified in CreateBone() .

State It can be one of the following value:

```
#True : the manual bone control is enabled.
#False: the manual bone control is disabled.
```

InheritOrientation (optional) Tells if the bone orientation should inherit from the entity orientation. It can be one of the following value:

`#True` : the manual bone orientation inherit from the entity.
`#False`: the manual bone orientation doesn't take in account the entity orientation.

Return value

None.

115.25 MoveEntityBone

Syntax

```
MoveEntityBone(#Entity, Bone$, x, y, z, Mode)
```

Description

Move the specified entity bone. The bone has to be in manual mode, set with `EnableManualEntityBoneControl()` .

Parameters

#Entity The entity to use.

Bone\$ The bone name in the OGRE mesh, or as specified in `CreateBone()` .

x, y, z The new position of the bone.

Mode The move mode. It can be one of the following values:

`#PB_Relative`: relative move, from the current bone position (default).
`#PB_Absolute`: absolute move to the specified position.

combined with one of the following values:

`#PB_Local` : local move.
`#PB_Parent`: move relative to the parent position.
`#PB_World` : move relative to the world.

Return value

None.

See Also

`RotateEntityBone()` , `ScaleEntityBone()`

115.26 ScaleEntityBone

Syntax

```
ScaleEntityBone(#Entity, Bone$, x, y, z, Mode)
```

Description

Scale the specified entity bone. The bone has to be in manual mode, set with `EnableManualEntityBoneControl()` .

Parameters

#Entity The entity to use.

Bone\$ The bone name in the OGRE mesh, or as specified in `CreateBone()` .

x, y, z The new scale of the bone.

Mode The scale mode. It can be one of the following value:

```
#PB_Relative: relative scale, based on the previous size
               (default). Using 1.0 for scale value will let this value
               unchanged.
#PB_Absolute: absolute scale, in world unit.
```

Return value

None.

See Also

`RotateEntityBone()` , `MoveEntityBone()`

115.27 FreeEntityBody

Syntax

```
FreeEntityBody(#Entity)
```

Description

Free the body associated with the entity.

Parameters

#Entity The entity to use.

Return value

None.

See Also

`CreateEntityBody()`

115.28 FreeEntityJoints

Syntax

```
FreeEntityJoints(#Entity)
```

Description

Free all joints associated with the entity.

Parameters

#Entity The entity to use.

Return value

None.

115.29 EntityBoneX

Syntax

```
Result = EntityBoneX(#Entity, Bone$ [, OffsetX, OffsetY, OffsetZ])
```

Description

Returns the 'x' position of the bone in the world.

Parameters

#Entity The entity to use.

Bone\$ The bone name in the OGRE mesh, or as specified in CreateBone() .

OffsetX (optional) The 'x' offset relative to the bone.

OffsetY (optional) The 'y' offset relative to the bone.

OffsetX (optional) The 'z' offset relative to the bone.

Return value

Returns the 'x' position of the #Entity bone in the world.

115.30 EntityBoneY

Syntax

```
Result = EntityBoneY(#Entity, Bone$ [, OffsetX, OffsetY, OffsetZ])
```

Description

Returns the 'y' position of the bone in the world.

Parameters

#Entity The entity to use.

Bone\$ The bone name in the OGRE mesh, or as specified in CreateBone() .

OffsetX (optional) The 'x' offset relative to the bone.

OffsetY (optional) The 'y' offset relative to the bone.

OffsetX (optional) The 'z' offset relative to the bone.

Return value

Returns the 'y' position of the #Entity bone in the world.

115.31 EntityBoneZ

Syntax

```
Result = EntityBoneZ(#Entity, Bone$ [, OffsetX, OffsetY, OffsetZ])
```

Description

Returns the 'z' position of the bone in the world.

Parameters

#Entity The entity to use.

Bone\$ The bone name in the OGRE mesh, or as specified in CreateBone() .

OffsetX (optional) The 'x' offset relative to the bone.

OffsetY (optional) The 'y' offset relative to the bone.

OffsetZ (optional) The 'z' offset relative to the bone.

Return value

Returns the 'z' position of the #Entity bone in the world.

115.32 EntityBonePitch

Syntax

```
Result = EntityBonePitch(#Entity, Bone$)
```

Description

Returns the pitch of the entity bone.

Parameters

#Entity The entity to use.

Bone\$ The bone name in the OGRE mesh, or as specified in CreateBone() .

Return value

The current pitch value of the bone. This value is always between -180.0 and 180.0 degrees.

See Also

EntityBoneYaw() , EntityBoneRoll()

115.33 EntityBoneYaw

Syntax

```
Result = EntityBoneYaw(#Entity, Bone$)
```

Description

Returns the yaw of the entity bone.

Parameters

#Entity The entity to use.

Bone\$ The bone name in the OGRE mesh, or as specified in CreateBone() .

Return value

The current yaw value of the bone. This value is always between -180.0 and 180.0 degrees.

See Also

EntityBonePitch() , EntityBoneRoll()

115.34 EntityBoneRoll

Syntax

```
Result = EntityBoneRoll(#Entity, Bone$)
```

Description

Returns the roll of the entity bone.

Parameters

#Entity The entity to use.

Bone\$ The bone name in the OGRE mesh, or as specified in CreateBone() .

Return value

The current roll value of the bone. This value is always between -180.0 and 180.0 degrees.

See Also

EntityBonePitch() , EntityBoneYaw()

115.35 EntityX

Syntax

```
Result = EntityX(#Entity [, Mode])
```

Description

Returns the current position of the entity in the world.

Parameters

#Entity The entity to use.

Mode (optional) The mode to get the 'x' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'x' position of the entity in
               the world (default).
#PB_Relative: get the 'x' position of the entity relative to
               its parent.
```

Return value

Returns the 'x' position of the entity.

See Also

EntityY() , EntityZ() , MoveEntity()

115.36 EntityY

Syntax

```
Result = EntityY(#Entity [, Mode])
```

Description

Returns the current position of the entity in the world.

Parameters

#Entity The entity to use.

Mode (optional) The mode to get the 'y' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'y' position of the entity in
               the world (default).
#PB_Relative: get the 'y' position of the entity relative to
               its parent.
```

Return value

Returns the 'y' position of the entity.

See Also

EntityX() , EntityZ() , MoveEntity()

115.37 EntityZ

Syntax

```
Result = EntityZ(#Entity [, Mode])
```

Description

Returns the current position of the entity in the world.

Parameters

#Entity The entity to use.

Mode (optional) The mode to get the 'z' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'z' position of the entity in
               the world (default).
#PB_Relative: get the 'z' position of the entity relative to
               its parent.
```

Return value

Returns the 'z' position of the entity.

See Also

EntityX() , EntityY() , MoveEntity()

115.38 FreeEntity

Syntax

```
FreeEntity(#Entity)
```

Description

Free the specified #Entity created with CreateEntity() before. All its associated memory is released and this object can't be used anymore.

Parameters

#Entity The entity to free. If **#PB_All** is specified, all the remaining entities are freed.

Return value

None.

Remarks

All remaining entities are automatically freed when the program ends.

115.39 HideEntity

Syntax

```
HideEntity(#Entity, State)
```

Description

Hides or shows the specified #Entity.
'State' can take the following values:

```
1: the #Entity is hidden
0: the #Entity is shown
```

115.40 IsEntity

Syntax

```
Result = IsEntity(#Entity)
```

Description

Tests if the given #Entity is a valid and correctly initialized entity.

This function is bulletproof and can be used with any value. If the 'Result' is not zero then the object is valid and initialized, else it returns zero. This is the correct way to ensure an entity is ready to use.

115.41 MoveEntity

Syntax

```
MoveEntity(#Entity, x, y, z [, Mode])
```

Description

Move the specified entity.

Parameters

#Entity The entity to use.

x, y, z The new position of the entity.

Mode (optional) The move mode. It can be one of the following values:

```
#PB_Relative: relative move, from the current entity position  
(default).  
#PB_Absolute: absolute move to the specified position.
```

combined with one of the following values:

```
#PB_Local : local move.  
#PB_Parent: move relative to the parent position.  
#PB_World : move relative to the world.
```

Return value

None.

See Also

RotateEntity()

115.42 RotateEntity

Syntax

```
RotateEntity(#Entity, x, y, z [, Mode])
```

Description

Rotates the entity according to the specified x,y,z angle values.

Parameters

#Entity The entity to use.

x, y, z The rotation to apply, in degree. Valid values are from 0 to 359.

Mode (optional) The rotation mode. It can be one of the following value:

```
#PB_Absolute: absolute rotation (default).
#PB_Relative: relative rotation based on the previous entity
rotation.
```

Return value

None.

See Also

MoveEntity()

115.43 RotateEntityBone

Syntax

```
RotateEntityBone(#Entity, Bone$, x, y, z, Mode)
```

Description

Rotates the entity bone according to the specified x,y,z angle values. The bone has to be in manual mode, set with EnableManualEntityBoneControl() .

Parameters

#Entity The entity to use.

Bone\$ The bone name in the OGRE mesh, or as specified in CreateBone() .

x, y, z The rotation to apply, in degree. Valid values are from 0 to 359.

Mode (optional) The rotation mode. It can be one of the following value:

```
#PB_Absolute: absolute rotation (default).
#PB_Relative: relative rotation based on the previous entity
bone rotation.
```

Return value

None.

See Also

MoveEntityBone() , ScaleEntityBone()

115.44 ScaleEntity

Syntax

```
ScaleEntity(#Entity, x, y, z [, Mode])
```

Description

Scales the entity according to the specified x,y,z values. When using `#PB_Relative` mode, this is a factor based scale which means the entity size will be multiplied with the given value to obtain the new size.

Parameters

`#Entity` The entity to use.

`x, y, z` The scaling to apply.

Mode (optional) The scale mode. It can be one of the following value:

```
#PB_Relative: relative scale, based on the previous size
               (default). Using 1.0 for scale value will let this value
               unchanged.
#PB_Absolute: absolute scale, in world unit.
```

Return value

None.

Example

```
1  ScaleEntity(0, 2, 2, 2) ; Double the current size of the entity
2  ScaleEntity(0, 1, 1, 1) ; Don't change the size of the entity
   (multiply by 1 don't change anything)
3  ScaleEntity(0, 3, 1, 1) ; Make the width of the entity 3 times
   larger
4  ScaleEntity(0, 1, 1, 1, #PB_Absolute) ; Reset the entity size to
   1,1,1.
```

115.45 EntityRoll

Syntax

```
Result = EntityRoll(#Entity [, Mode])
```

Description

Get the roll of the `#Entity`.

Parameters

#Entity The entity to use.

Mode (optional) The mode to get the roll. It can be one of the following value:

```
#PB_Absolute: get the absolute roll value, ignoring the
               current roll of the parent (default).
#PB_Relative: get the roll value relative to the current roll
               of the parent.
```

combined with one of the following value:

```
#PB_Engine3D_Raw      : the roll is the raw value, but it
                       can't be used in RotateEntity()
to get back the same orientation (default).
#PB_Engine3D_Adjusted: the roll is adjusted, so it can be put
                       back in RotateEntity()
to get back the same orientation.
```

Return value

The current roll value of the specified entity. This value is always between -180.0 and 180.0 degrees.

See Also

EntityYaw() , EntityPitch()

115.46 EntityPitch

Syntax

```
Result = EntityPitch(#Entity [, Mode])
```

Description

Get the pitch of the #Entity.

Parameters

#Entity The entity to use.

Mode (optional) The mode to get the pitch. It can be one of the following value:

```
#PB_Absolute: get the absolute pitch value, ignoring the
               current pitch of the parent (default).
#PB_Relative: get the pitch value relative to the current
               pitch of the parent.
```

combined with one of the following value:

```
#PB_Engine3D_Raw      : the pitch is the raw value, but it
                       can't be used in RotateEntity()
to get back the same orientation (default).
#PB_Engine3D_Adjusted: the pitch is adjusted, so it can be
                       put back in RotateEntity()
to get back the same orientation.
```

Return value

The current pitch value of the specified entity. This value is always between -180.0 and 180.0 degrees.

See Also

EntityYaw() , EntityRoll()

115.47 EntityYaw

Syntax

```
Result = EntityYaw(#Entity [, Mode])
```

Description

Get the yaw of the #Entity.

Parameters

#Entity The entity to use.

Mode (optional) The mode to get the yaw. It can be one of the following value:

- #PB_Absolute**: get the absolute yaw value, ignoring the current yaw of the parent (default).
- #PB_Relative**: get the yaw value relative to the current yaw of the parent.

combined with one of the following value:

- #PB_Engine3D_Raw** : the yaw is the raw value, but it can't be used in RotateEntity() to get back the same orientation (default).
- #PB_Engine3D_Adjusted**: the yaw is adjusted, so it can be put back in RotateEntity() to get back the same orientation.

Return value

The current yaw value of the specified entity. This value is always between -180.0 and 180.0 degrees.

See Also

EntityPitch() , EntityRoll()

115.48 GetEntityAttribute

Syntax

```
Result.f = GetEntityAttribute(#Entity, Attribute)
```

Description

Get the specified attribute of the given entity.

Parameters

#Entity The entity to use.

Attribute The attribute value to get. The following attributes are available:

```
#PB_Entity_Friction          : Get the friction value.
#PB_Entity_Restitution       : Get the restitution value.
#PB_Entity_LinearVelocity    : Get the current linear velocity
    (all axis).
#PB_Entity_LinearVelocityX   : Get the current linear velocity
    on 'x' axis.
#PB_Entity_LinearVelocityY   : Get the current linear velocity
    on 'y' axis.
#PB_Entity_LinearVelocityZ   : Get the current linear velocity
    on 'z' axis.
#PB_Entity_MassCenterX       : Get the mass center 'x' position.
#PB_Entity_MassCenterY       : Get the mass center 'y' position.
#PB_Entity_MassCenterZ       : Get the mass center 'z' position.
#PB_Entity_NbSubEntities     : Get the number of sub-entities.
#PB_Entity_LinearSleeping    : Get the minimum linear velocity
    value under which the entity will be sleeping.
#PB_Entity_AngularSleeping   : Get the minimum angular velocity
    value under which the entity will be sleeping.
#PB_Entity_DeactivationTime : Get the time to wait (in
    milliseconds) before putting the entity in sleep mode when
    the above conditions are met.
#PB_Entity_IsActive          : Get if an entity body is active
    (not sleeping).
#PB_Entity_AngularVelocityX  : Get the current angular velocity
    on 'x' axis.
#PB_Entity_AngularVelocityY  : Get the current angular velocity
    on 'y' axis.
#PB_Entity_AngularVelocityZ  : Get the current angular velocity
    on 'z' axis.
#PB_Entity_AngularVelocity   : Get the current angular velocity
    (all axis).
#PB_Entity_HasContactResponse : Check if the entity body has
    contacts.
#PB_Entity_ScaleX            : Get the current entity scale on
    'x' axis.
#PB_Entity_ScaleY            : Get the current entity scale on
    'y' axis.
#PB_Entity_ScaleZ            : Get the current entity scale on
    'z' axis.
#PB_Entity_InheritScale      : Get if the entity has inherit
    scale.
```

Return value

Returns the value of the specified attribute or 0 if the entity does not support the attribute.

See Also

SetEntityAttribute()

115.49 SetEntityAttribute

Syntax

```
SetEntityAttribute(#Entity, Attribute, Value.f)
```

Description

Set the specified attribute value to the given entity.

Parameters

#Entity The entity to use.

Attribute The attribute to set. The following attributes are available:

```
#PB_Entity_Friction          : Change the friction value.
#PB_Entity_Restitution       : Change the restitution value.
#PB_Entity_MinVelocity       : Set the minimum linear velocity
    of the entity. As this value isn't stored,
                                it needs to be called every
    time the entity is moved.
#PB_Entity_MaxVelocity       : Set the maximum linear velocity
    of the entity. As this value isn't stored,
                                it needs to be called every
    time the entity is moved.
#PB_Entity_ForceVelocity     : Set the linear velocity of the
    entity. As this value isn't stored,
                                it needs to be called every
    time the entity is moved.
#PB_Entity_LinearSleeping    : Change the minimum linear
    velocity value under which the entity will be sleeping.
#PB_Entity_AngularSleeping   : Change the minimum angular
    velocity value under which the entity will be sleeping.
#PB_Entity_DeactivationTime  : Time to wait (in milliseconds)
    before putting the entity in sleep mode when the above
    conditions are met.
#PB_Entity_DisableContactResponse: Disable or enable the
    physic contacts for this entity. Value can be #True or
    #False.
#PB_Entity_InheritScale      : Disable or enable if the entity
    has inherit scale. Value can be #True or #False
```

Value Value of the attribute to set.

Return value

None.

See Also

GetEntityAttribute()

115.50 GetEntityCollisionMask

Syntax

```
Result = GetEntityCollisionMask(#Entity)
```

Description

Get the current entity collision mask, as set with SetEntityCollisionFilter() .

Parameters

#Entity The entity to use.

Return value

The current entity collision mask.

See Also

SetEntityCollisionFilter() , GetEntityCollisionGroup()

115.51 GetEntityCollisionGroup

Syntax

```
Result = GetEntityCollisionGroup(#Entity)
```

Description

Get the current entity collision group, as set with SetEntityCollisionFilter() .

Parameters

#Entity The entity to use.

Return value

The current entity collision group.

See Also

SetEntityCollisionFilter() , GetEntityCollisionMask()

115.52 SetEntityCollisionFilter

Syntax

```
SetEntityCollisionFilter(#Entity, CollisionGroup, CollisionMask)
```

Description

Set the entity collision group and mask.

Parameters

#Entity The entity to use.

CollisionGroup The new collision group.

CollisionMask The new collision mask.

Return value

None.

See Also

GetEntityCollisionGroup() , GetEntityCollisionMask() , RayCollide()

115.53 AddSubEntity

Syntax

```
Result = AddSubEntity(#Entity, #SubEntity, Type, [OffsetX, OffsetY,
    OffsetZ [, SizeX, SizeY, SizeZ [, AxisX, AxisY, AxisZ]]])
```

Description

Add a sub entity to an entity.

Parameters

#Entity The entity to use.

#SubEntity The entity to add.

Type Type defines how the physic engine will handle this entity. It can be one of the following constants:

```
#PB_Entity_StaticBody : The body is a static only, which
    means the mesh can't be animated.
    This mode allows very precise
    collisions, as it's done against triangles
    (also known as tri-mesh collision).
    It's also fast when colliding with
    a box or sphere entity body type.
    It's perfect when using a mesh for
    a ground or static world.
#PB_Entity_PlaneBody : A 'virtual' plane is set on the
    entity (with the same dimensions)
    and is used to manage the collision
    against the other entities.
#PB_Entity_ConeBody : A 'virtual' cone is set around the
    entity (with the same dimensions)
    and is used to manage the collision
    against the other entities.
#PB_Entity_BoxBody : A 'virtual' box is set around the
    entity (with the same dimensions)
    and is used to manage the collision
    against the other entities.
#PB_Entity_SphereBody : A 'virtual' sphere is set around the
    entity and is used to manage the collision
```

against the other entities.

`#PB_Entity_CylinderBody` : A 'virtual' cylinder is set around the entity and is used to manage the collision against the other entities.

`#PB_Entity_CapsuleBody` : A 'virtual' capsule is set around the entity and is used to manage the collision against the other entities.

`#PB_Entity_ConvexHullBody` : A 'virtual' complex form deduced from the real mesh is set around the entity and is used to manage the collision against the other entities. This mode is slower than basic collide forms.

`#PB_Entity_CompoundBody` : A 'virtual' Compound body is set around the entity and this is the fastest kind of arbitrary shape. It is defined by a cloud of vertices but the shape formed is the smallest convex shape that encloses the vertices.

OffsetX, OffsetY, OffsetZ (optional) Offset of the translation of the body.

SizeX, SizeY, SizeZ (optional) The bounding box size of the body. It only applies to the following body type:

`#PB_Entity_BoxBody` : SizeX, SizeY and SizeZ are available.

`#PB_Entity_SphereBody` : SizeX is available.

`#PB_Entity_ConeBody` : SizeX and SizeY are available.

`#PB_Entity_CylinderBody`: SizeX and SizeY are available.

`#PB_Entity_CapsuleBody` : SizeX, SizeY and SizeZ are available.

If SizeX = -1 then it uses the BoundingBox.

AxisX, AxisY, AxisZ (optional) The axis of the body' orientation. It only applies to the following body type:

`#PB_Entity_PlaneBody`

`#PB_Entity_CylinderBody`

`#PB_Entity_CapsuleBody`

Return value

Returns zero if the entity can't be created.

Remarks

It is necessary to use the `CreateEntityBody()` function with the `#PB_Entity_CompoundBody` option after adding all sub-entities.

See Also

`CreateEntity()`

115.54 EntityDirection

Syntax

`EntityDirection(#Entity, x, y, z [, Mode, LocalDirectionVector])`

Description

Set the direction for the entity.

Parameters

#Entity The entity to use.

x, y, z The direction vector (value between -1.0 and 1.0).

Mode (optional) The direction mode. It can be one of the following values:

```
#PB_Local : local move.
#PB_Parent: move relative to the parent position.
#PB_World : move relative to the world.
```

LocalDirectionVector (optional) The local direction vector. It can be one of the following values:

```
#PB_Vector_X
#PB_Vector_Y
#PB_Vector_Z
#PB_Vector_NegativeX
#PB_Vector_NegativeY
#PB_Vector_NegativeZ
```

Return value

None.

See Also

EntityDirectionX() , EntityDirectionY() , EntityDirectionZ()

115.55 EntityDirectionX

Syntax

```
Result = EntityDirectionX(#Entity)
```

Description

Get the 'x' direction of the entity.

Parameters

#Entity The entity to use.

Return value

Returns the 'x' direction of the entity.

See Also

EntityDirection() , EntityDirectionY() , EntityDirectionZ()

115.56 EntityDirectionY

Syntax

```
Result = EntityDirectionY(#Entity)
```

Description

Get the 'y' direction of the entity.

Parameters

#Entity The entity to use.

Return value

Returns the 'y' direction of the entity.

See Also

EntityDirection() , EntityDirectionX() , EntityDirectionZ()

115.57 EntityDirectionZ

Syntax

```
Result = EntityDirectionZ(#Entity)
```

Description

Get the 'z' direction of the entity.

Parameters

#Entity The entity to use.

Return value

Returns the 'z' direction of the entity.

See Also

EntityDirection() , EntityDirectionX() , EntityDirectionY()

115.58 GetEntityMesh

Syntax

```
Result = GetEntityMesh(#Entity)
```

Description

Returns the #Mesh used by the entity.

Parameters

#Entity The entity to use.

Return value

Returns the #Mesh used by the entity.

See Also

CreateEntity()

Chapter 116

EntityAnimation

Overview

Entities are objects composed of one mesh object and one material which can be freely moved and transformed in real-time. This library is provided to control the entity animation. The mesh associated to the entity needs to have a skeleton with some predefined animations. `InitEngine3D()` should be called successfully before using the entity animation functions.

116.1 AddEntityAnimationTime

Syntax

```
AddEntityAnimationTime(#Entity, Animation$, Time)
```

Description

Add time to the specified `#Entity` animation.

Parameters

#Entity The entity to use.

Animation\$ The animation name. The animations are stored in the mesh object in a case-sensitive manner (ie: "Walk" will be a different animation than "walk"). If the animation isn't found or the mesh doesn't have a skeleton, this function will have no effect.

Time The time to add (in milliseconds) to the specified animation, relative to the current animation time.

Return value

None.

See Also

`StartEntityAnimation()`

116.2 StartEntityAnimation

Syntax

```
StartEntityAnimation(#Entity, Animation$ [, Flags])
```

Description

Start the specified #Entity animation. The animation is always started from the beginning.

Parameters

#Entity The entity to use.

Animation\$ The animation name. The animations are stored in the mesh object in a case-sensitive manner (ie: "Walk" will be a different animation than "walk"). If the animation isn't found or the mesh doesn't have a skeleton, this function will have no effect.

Flag Flags can be a combination of the following values:

```
#PB_EntityAnimation_Once:   Play the animation only once. By  
default the animation loops automatically when its end is  
reached.
```

```
EntityAnimationStatus()  
can be used to detect the animation end.
```

```
#PB_EntityAnimation_Manual: Start the animation in manual  
mode, the time won't be automatically added after each  
RenderWorld()
```

```
AddEntityAnimationTime()  
needs to be called to update the animation time manually.
```

Return value

None.

See Also

StopEntityAnimation() , EntityAnimationStatus() , AddEntityAnimationTime()

116.3 StopEntityAnimation

Syntax

```
StopEntityAnimation(#Entity, Animation$)
```

Description

Stop the specified #Entity animation.

Parameters

#Entity The entity to use.

Animation\$ The animation name. The animations are stored in the mesh object in a case-sensitive manner (ie: "Walk" will be a different animation than "walk"). If the animation isn't found or the mesh doesn't have a skeleton, this function will have no effect.

Return value

None.

See Also

StartEntityAnimation()

116.4 EntityAnimationStatus

Syntax

```
Result = EntityAnimationStatus(#Entity, Animation$)
```

Description

Return the specified #Entity animation status.

Parameters

#Entity The entity to use.

Animation\$ The animation name. The animations are stored in the mesh object in a case-sensitive manner (ie: "Walk" will be a different animation than "walk"). If the animation isn't found or the mesh doesn't have a skeleton, this function will have no effect.

Return value

The return value can be one of the following constants:

```
#PB_EntityAnimation_Stopped: The animation is stopped (or has ended).  
#PB_EntityAnimation_Started: The animation is running.  
#PB_EntityAnimation_Unknown: The animation doesn't exist in the mesh object.
```

See Also

StartEntityAnimation() , StopEntityAnimation()

116.5 EntityAnimationBlendMode

Syntax

```
EntityAnimationBlendMode(#Entity, Mode)
```

Description

Changes the #Entity animation blendmode.

Parameters

#Entity The entity to use.

Mode The blend mode can be one of the following value:

#PB_EntityAnimation_Average: The blend will result of the average of the two animations (default). For example, if the first animation rotates an arm 40 degrees and the second animation rotates the arm 90 degrees, the arm will rotate $(40+90)/2 = 65$ degrees (if the both animation are played with full weight).

#PB_EntityAnimation_Cumulative: The blend will sum the two animations. For example, if the first animation rotates an arm 40 degrees and the second animation rotates the arm 90 degrees, the arm will rotate $40+90 = 130$ degrees (if the both animation are played with full weight).

Return value

None.

Remarks

When switching from an animation to another with `SetEntityAnimationWeight()`, a blend is applied to have a smooth transition between the animations.

See Also

`StartEntityAnimation()`, `SetEntityAnimationWeight()`

116.6 GetEntityAnimationTime

Syntax

```
Result = GetEntityAnimationTime(#Entity, Animation$)
```

Description

Returns the current #Entity animation time.

Parameters

#Entity The entity to use.

Animation\$ The animation name. The animations are stored in the mesh object in a case-sensitive manner (ie: "Walk" will be a different animation than "walk"). If the animation isn't found or the mesh doesn't have a skeleton, this function will have no effect.

Return value

The current entity animation time (in milliseconds) or 0 if the animation isn't running.

See Also

StartEntityAnimation() , AddEntityAnimationTime() , SetEntityAnimationTime()

116.7 SetEntityAnimationTime

Syntax

```
SetEntityAnimationTime(#Entity, Animation$, Time)
```

Description

Changes the current #Entity animation time. This is an absolute time position. To change the time relative to the current time, use AddEntityAnimationTime() .

Parameters

#Entity The entity to use.

Animation\$ The animation name. The animations are stored in the mesh object in a case-sensitive manner (ie: "Walk" will be a different animation than "walk"). If the animation isn't found or the mesh doesn't have a skeleton, this function will have no effect.

Time The absolute time to set (in milliseconds).

Return value

None.

See Also

StartEntityAnimation() , AddEntityAnimationTime() , GetEntityAnimationTime()

116.8 GetEntityAnimationLength

Syntax

```
Result = GetEntityAnimationLength(#Entity, Animation$)
```

Description

Returns the #Entity animation length.

Parameters

#Entity The entity to use.

Animation\$ The animation name. The animations are stored in the mesh object in a case-sensitive manner (ie: "Walk" will be a different animation than "walk"). If the animation isn't found or the mesh doesn't have a skeleton, this function will have no effect.

Return value

The entity animation length (in milliseconds).

See Also

StartEntityAnimation() , SetEntityAnimationLength()

116.9 SetEntityAnimationLength

Syntax

```
SetEntityAnimationLength(#Entity, Animation$, Length)
```

Description

Change the #Entity animation length.

Parameters

#Entity The entity to use.

Animation\$ The animation name. The animations are stored in the mesh object in a case-sensitive manner (ie: "Walk" will be a different animation than "walk"). If the animation isn't found or the mesh doesn't have a skeleton, this function will have no effect.

Length The new entity animation length (in milliseconds).

See Also

StartEntityAnimation() , GetEntityAnimationLength()

116.10 GetEntityAnimationWeight

Syntax

```
Result = GetEntityAnimationWeight(#Entity, Animation$)
```

Description

Returns the #Entity animation weight. The weight is useful when playing several animations at once. For example to do a smooth transition from one animation to another, it is possible to reduce progressively the weight of the first animation and increase the weight of the second animation.

Parameters

#Entity The entity to use.

Animation\$ The animation name. The animations are stored in the mesh object in a case-sensitive manner (ie: "Walk" will be a different animation than "walk"). If the animation isn't found or the mesh doesn't have a skeleton, this function will have no effect.

Return value

The current entity animation weight (value between 0.0 and 1.0). If the weight is 0, then the animation has no effect. If the weight is 1, then animation is fully playing.

Remarks

The EntityAnimationBlendMode() also affects how animations are mixed.

See Also

StartEntityAnimation() , EntityAnimationBlendMode()

116.11 SetEntityAnimationWeight

Syntax

```
SetEntityAnimationWeight(#Entity, Animation$, Weight)
```

Description

Changes the #Entity animation weight. The weight is useful when playing several animations at once. For example to do a smooth transition from one animation to another, it is possible to reduce progressively the weight of the first animation and increase the weight of the second animation.

Parameters

#Entity The entity to use.

Animation\$ The animation name. The animations are stored in the mesh object in a case-sensitive manner (ie: "Walk" will be a different animation than "walk"). If the animation isn't found or the mesh doesn't have a skeleton, this function will have no effect.

Weight The new entity animation weight (value between 0.0 and 1.0). If the weight is 0, then the animation has no effect. If the weight is 1, then animation is fully playing.

Return value

None.

Remarks

The EntityAnimationBlendMode() also affects how animations are mixed.

See Also

StartEntityAnimation() , EntityAnimationBlendMode()

116.12 UpdateEntityAnimation

Syntax

```
UpdateEntityAnimation(#Entity, Animation$)
```

Description

Update the #Entity animation. For example, if the vertices of the mesh have been modified the animation cache needs to be recalculated.

Parameters

#Entity The entity to use.

Animation\$ The animation name. The animations are stored in the mesh object in a case-sensitive manner (ie: "Walk" will be a different animation than "walk"). If the animation isn't found or the mesh doesn't have a skeleton, this function will have no effect.

Return value

None.

See Also

`StartEntityAnimation()`

Chapter 117

File

Overview

Files are the main method for storing data on computers. PureBasic allows the programmer to create applications in such a manner that the methods used to manage these files are simple to use, and yet still optimized. Any number of files may be handled at the same time. This library uses buffered functions to increase the reading/writing speed. All the file functions can handle huge files, all the way up to: 2⁶⁴ bytes, (i.e. if the file-system supports it).

For large amounts of data it may be useful to load the data into an array , a list or a Map , using a memory block may also be a good idea.

To get valid file paths for reading/saving data, take a look at the FileSystem and the Requester libraries.

117.1 CloseFile

Syntax

```
CloseFile(#File)
```

Description

Close the specified file.

Parameters

#File The file to close. If **#PB_All** is specified, all the remaining files are closed.

Return value

None.

Remarks

Once the file is closed, it may not be used anymore. Closing a file ensures the buffer will effectively be put to the disk.

All remaining opened files are automatically closed when the program ends.

For an example see the ReadFile() or the CreateFile() functions.

See Also

CreateFile() , OpenFile() , ReadFile()

117.2 CreateFile

Syntax

```
Result = CreateFile(#File, Filename$ [, Flags])
```

Description

Create an empty file.

Parameters

#File The number to identify the new file. **#PB_Any** can be used to auto-generate this number.

Filename\$ The filename and path to the new file. If the filename does not include a full path, it is interpreted relative to the current directory .

Flags (optional) It can be a combination (using the '||' operand) of the following values:

```
#PB_File_SharedRead : the opened file can be read by another
process (Windows only).
#PB_File_SharedWrite: the opened file can be written by
another process (Windows only).
#PB_File_NoBuffering: the internal PureBasic file buffering
system will be disabled for this file.
                        FileBuffersSize()
can not be used on this file.
```

combined with one of the following values (the following flags affect the WriteString() (), WriteStringN() , ReadString() , ReadCharacter() and WriteCharacter() behaviour):

```
#PB_Ascii   : all read/write string operation will use ASCII
if not specified otherwise.
#PB_UTF8    : all read/write string operation will use UTF-8
if not specified otherwise (default).
#PB_Unicode: all read/write string operation will use Unicode
if not specified otherwise.
```

Return value

Returns nonzero if the file was created successfully and zero if there was an error. If **#PB_Any** was used as the **#File** parameter then the new generated number is returned on success.

Remarks

If the file already exists, it will be overwritten by the new empty file. The FileSize() function can be used to determine whether a file exists so the user can be prompted before overwriting a file. To open an existing file for reading/writing, use the OpenFile() function. To open a file for reading only, use ReadFile() .

Example

```
1
2 If CreateFile(0, "Text.txt")           ; we create a new text
   file...
3   For a=1 To 10
4     WriteStringN(0, "Line "+Str(a))    ; we write 10 lines (each
     with 'end of line' character)
```

```

5      Next
6      For a=1 To 10
7          WriteString(0, "String"+Str(a)) ; and now we add 10 more
          strings on the same line (because there is no 'end of line'
          character)
8      Next
9      CloseFile(0) ; close the previously
          opened file and store the written data this way
10     Else
11         MessageRequester("Information","may not create the file!")
12     EndIf

```

See Also

OpenFile() , ReadFile() , CloseFile()

117.3 Eof

Syntax

```
Result = Eof(#File)
```

Description

Checks whether the end of the file has been reached.

Parameters

#File The file to use.

Return value

Returns nonzero if the read-pointer is at the end of the file or zero if not.

Remarks

For an example see the ReadFile() function.

See Also

Lof() , Loc() , CreateFile() , OpenFile() , ReadFile()

117.4 FileBuffersSize

Syntax

```
FileBuffersSize(#File, Size)
```

Description

Changes the size of the memory buffer used for file operations.

Parameters

#File The file to change. If **#PB_Default** is used as this parameter, then the new buffer size will apply to all newly opened files with `OpenFile()` , `CreateFile()` or `ReadFile()` .

Size The new size (in bytes) for the memory buffer. A size of 0 disables memory buffering on the file.

Return value

None.

Remarks

For performance reasons, the buffer size should be kept large enough (1028 seems to be ok as a minimum). When buffers are used, the information is really written to the disk once the cache buffer is full or when the file is closed. The `FlushFileBuffers()` function forces the cache buffer to be written at the time the function is called. The default buffer size is 4096 bytes per file.

See Also

`FlushFileBuffers()`

117.5 FileID

Syntax

```
Result = FileID(#File)
```

Description

Returns the operating system handle of the file.

Parameters

#File The file to use.

Return value

Returns the file handle.

See Also

`CreateFile()` , `OpenFile()` , `ReadFile()`

117.6 FileSeek

Syntax

```
FileSeek(#File, NewPosition.q [, Mode])
```

Description

Change the read/write pointer position in the file.

Parameters

#File The file to use.

NewPosition.q The new position relative to the beginning of the file in bytes.

Mode (optional) The seek mode. It can be one of the following values:

```
#PB_Absolute: the 'NewPosition' parameter will be an absolute
               position with the file (default).
#PB_Relative: the 'NewPosition' parameter will be an offset
               (positive or negative) relative to the current file pointer
               position.
```

Return value

None.

Example

```
1  File$ = OpenFileRequester("Select a file","", "All files
   (*.*)|*.*",0)
2  If File$
3      If ReadFile(0, File$)
4
5          ; Read length of file
6          Length = Lof(0)
7          Debug "File length: "+FormatNumber(Length, 0)+" bytes"
8
9          ; Set the file pointer 10 bytes from end of file
10         FileSeek(0, Length - 10)
11         Debug "Position after seek: "+FormatNumber(Loc(0), 0)
12
13         CloseFile(0)
14     Else
15         Debug "Can't read the file: "+File$
16     EndIf
17 EndIf
```

See Also

Loc() , Lof()

117.7 FlushFileBuffers

Syntax

```
Result = FlushFileBuffers(#File)
```

Description

Ensures that all buffered operations are written to disk.

Parameters

#File The file to use.

Return value

Nonzero if the buffer has been successfully written the disk. If an error occurred (ie: disk full, disk error), it will return zero.

Remarks

See FileBuffersSize() for more information about file buffer management.

See Also

FileBuffersSize()

117.8 IsFile

Syntax

```
Result = IsFile(#File)
```

Description

Tests if the given #File number is a valid and correctly initialized file.

Parameters

#File The file to use.

Return value

Returns nonzero if #File is a valid file and zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a file is ready to use.

See Also

CreateFile() , OpenFile() , ReadFile()

117.9 Loc

Syntax

```
Position.q = Loc(#File)
```

Description

Returns the read/write pointer position in the file.

Parameters

#File The file to use.

Return value

Returns the file pointer position relative to the start of the file in bytes.

Remarks

For an example look at the FileSeek() function.

See Also

FileSeek() , Lof()

117.10 Lof

Syntax

```
Length.q = Lof(#File)
```

Description

Returns the length of the specified file.

Parameters

#File The file to use.

Return value

Returns the length of the file in bytes.

Example

```
1
2  file$ = OpenFileRequester("Select a file","", "Text
   (.txt)|*.txt|All files (*.*)|*.*",0)
3  If file$
4      If ReadFile(0, file$)
5          length = Lof(0)                                ; get the length
   of opened file
6          *MemoryID = AllocateMemory(length)              ; allocate the
   needed memory
7          If *MemoryID
8              bytes = ReadData(0, *MemoryID, length)      ; read all data
   into the memory block
9              Debug "Number of bytes read: " + Str(bytes)
10             EndIf
11             CloseFile(0)
12         EndIf
13     EndIf
```

See Also

Loc() , FileSeek() , FileSize()

117.11 OpenFile

Syntax

```
Result = OpenFile(#File, Filename$ [, Flags])
```

Description

Opens a file for reading/writing or creates a new file if it does not exist.

Parameters

#File The number to identify the file. **#PB_Any** can be used to auto-generate this number.

Filename\$ The filename and path to the file. If the filename does not include a full path, it is interpreted relative to the current directory .

Flags (optional) It can be a combination (using the '||' operand) of the following values:

```
#PB_File_SharedRead : the opened file can be read by another
process (Windows only).
#PB_File_SharedWrite: the opened file can be written by
another process (Windows only).
#PB_File_Append      : the file pointer position will be set
at the end of file.
#PB_File_NoBuffering: the internal PureBasic file buffering
system will be disabled for this file.
                        FileBuffersSize()
can not be used on this file.
```

combined with one of the following values (the following flags affect the WriteString() , WriteStringN() , ReadString() , ReadCharacter() and WriteCharacter() behaviour):

```
#PB_Ascii   : all read/write string operation will use ASCII
if not specified otherwise.
#PB_UTF8    : all read/write string operation will use UTF-8
if not specified otherwise (default).
#PB_Unicode: all read/write string operation will use Unicode
if not specified otherwise.
```

Return value

Returns nonzero if the file was opened successfully and zero if there was an error. If **#PB_Any** was used as the **#File** parameter then the new generated number is returned on success.

Remarks

This function fails if the file cannot be opened with write permission, for example if the file is located on a read-only file-system like a CD. To open a file for reading only, use the ReadFile() function. To overwrite an existing file with a new and empty file, use the CreateFile() function. The file pointer will be positioned at the beginning of the file. To append data to the end of the file, use the **#PB_File_Append** flag to set the pointer to the end of the file.

Example

```

1
2  If OpenFile(0, "Test.txt")      ; opens an existing file or creates
   one, if it does not exist yet
3      FileSeek(0, Lof(0))        ; jump to the end of the file
   (result of Lof() is used)
4      WriteStringN(0, "... another line at the end.")
5      CloseFile(0)
6  EndIf

```

See Also

CreateFile() , ReadFile() , CloseFile()

117.12 TruncateFile

Syntax

```
TruncateFile(#File)
```

Description

Cuts the file at the current file position and discards all data that follows.

Parameters

#File The file to use.

Return value

None.

Remarks

This function may be used to shorten a file without the necessity of recreating it entirely. To make a file longer, simply append more data with the write commands of this library.

See Also

FileSeek() , Loc()

117.13 ReadAsciiCharacter

Syntax

```
Number.a = ReadAsciiCharacter(#File)
```

Description

Read an ASCII character (1 byte) from a file, starting at the current file position.

Parameters

#File The file to read from.

Return value

Returns the read ASCII character or zero if there was an error.

Remarks

For an example of how to read from a file, see the `ReadFile()` function - with `ReadAsciiCharacter()` only an ASCII character is read, instead of a complete line (string).

See Also

`WriteAsciiCharacter()` , `ReadUnicodeCharacter()` , `ReadCharacter()` , `OpenFile()` , `ReadFile()` , `Loc()`

117.14 ReadByte

Syntax

```
Number.b = ReadByte(#File)
```

Description

Read a byte (1 byte) from a file, starting at the current file position.

Parameters

#File The file to read from.

Return value

Returns the read byte or zero if there was an error.

Remarks

For an example of how read from a file see the `ReadFile()` function - with `ReadByte()` only a byte value is read, instead of a complete line (string).

See Also

`WriteByte()` , `OpenFile()` , `ReadFile()` , `Loc()`

117.15 ReadCharacter

Syntax

```
Result.c = ReadCharacter(#File [, Format])
```

Description

Read a character from a file, starting at the current file position.

Parameters

#File The file to read from.

Format (optional) The format of the character to read. It can be one of the following value:

```
#PB_Ascii    : 1 byte character.  
#PB_Unicode  : 2 bytes character (default in unicode  
mode).  
#PB_UTF8     : multi-bytes character (from 1 to 4 bytes).
```

If this flag isn't set, then the format for reading the character depends on the related setting at the previously used `CreateFile()` , `OpenFile()` or `ReadFile()` command.

Return value

Returns the read character or zero if there was an error.

Remarks

For an example of how to read from a file, see the `ReadFile()` function - with `ReadCharacter()` only a character is read, instead of a complete line (string).

See Also

`WriteCharacter()` , `ReadAsciiCharacter()` , `ReadUnicodeCharacter()` , `OpenFile()` , `ReadFile()` , `Loc()`

117.16 ReadDouble

Syntax

```
Number.d = ReadDouble(#File)
```

Description

Read a double (8 bytes) from a file, starting at the current file position.

Parameters

#File The file to read from.

Return value

Returns the read double value or zero if there was an error.

Remarks

For an example of how to read from a file, see the `ReadFile()` function - with `ReadDouble()` only a double value is read, instead of a complete line (string).

See Also

`WriteDouble()` , `OpenFile()` , `ReadFile()` , `Loc()`

117.17 ReadFile

Syntax

```
Result = ReadFile(#File, Filename$ [, Flags])
```

Description

Open an existing file for read-only operations.

Parameters

#File The number to identify the file. **#PB_Any** can be used to auto-generate this number.

Filename\$ The filename and path to the file. If the filename does not include a full path, it is interpreted relative to the current directory .

Flags (optional) It can be a combination (using the '||' operand) of the following values:

#PB_File_SharedRead : if the file has been already opened by another process or thread for read operation, this flag is needed to access it

(Windows only).

#PB_File_SharedWrite: if the file has been already opened by another process or thread for write operation, this flag is needed to access it

(Windows only).

#PB_File_NoBuffering: the internal PureBasic file buffering system will be disabled for this file.

FileBuffersSize()

can not be used on this file.

combined with one of the following values (the following flags affect the behaviour of ReadString() and ReadCharacter()):

#PB_Ascii : all read string operation will use ASCII if not specified otherwise.

#PB_UTF8 : all read string operation will use UTF-8 if not specified otherwise (default).

#PB_Unicode: all read string operation will use Unicode if not specified otherwise.

Return value

Returns nonzero if the file was opened successfully and zero if there was an error. If **#PB_Any** was used as the **#File** parameter then the new generated number is returned on success.

Remarks

To open a file for reading and writing, use the OpenFile() function. To create a new and empty file, use the CreateFile() function.

Example

```
1 If ReadFile(0, "Text.txt") ; if the file could be read, we
  continue ...
2 Format = ReadStringFormat(0)
```

```

3      While Eof(0) = 0                                ; loop as long the 'end of
      file' isn't reached
4      Debug ReadString(0, Format)                      ; display line by line in the
      debug window
5      Wend
6      CloseFile(0)                                    ; close the previously opened
      file
7  Else
8      MessageRequester("Information", "Couldn't open the file!")
9  EndIf

```

See Also

OpenFile() , CreateFile() , CloseFile()

117.18 ReadFloat

Syntax

```
Number.f = ReadFloat(#File)
```

Description

Read a float (4 bytes) from a file, starting at the current file position.

Parameters

#File The file to read from.

Return value

Returns the read float value or zero if there was an error.

Remarks

For an example of how to read from a file, see the ReadFile() function - with ReadFloat() only a float value is read, instead of a complete line (string).

See Also

WriteFloat() , ReadDouble() , OpenFile() , ReadFile() , Loc()

117.19 ReadInteger

Syntax

```
Number.i = ReadInteger(#File)
```

Description

Read an integer (4 bytes in 32-bit executable, 8 bytes in 64-bit executable) from a file, starting at the current file position.

Parameters

#File The file to read from.

Return value

Returns the read value or zero if there was an error.

Remarks

For an example of how to read from a file, see the `ReadFile()` function - with `ReadInteger()` only a integer value is read, instead of a complete line (string).

See Also

`WriteInteger()` , `OpenFile()` , `ReadFile()` , `Loc()`

117.20 ReadLong

Syntax

```
Number.l = ReadLong(#File)
```

Description

Read a long (4 bytes) from a file, starting at the current file position.

Parameters

#File The file to read from.

Return value

Returns the read value or zero if there was an error.

Remarks

For an example of how to read from a file, see the `ReadFile()` function - with `ReadLong()` only a long value is read, instead of a complete line (string).

See Also

`WriteLong()` , `OpenFile()` , `ReadFile()` , `Loc()`

117.21 ReadQuad

Syntax

```
Number.q = ReadQuad(#File)
```

Description

Read a quad (8 bytes) from a file, starting at the current file position.

Parameters

#File The file to read from.

Return value

Returns the read number or zero if there was an error.

Remarks

For an example of how to read from a file, see the ReadFile() function - with ReadQuad() only a quad value is read, instead of a complete line (string).

See Also

WriteQuad() , OpenFile() , ReadFile() , Loc()

117.22 ReadData

Syntax

```
Result = ReadData(#File, *MemoryBuffer, LengthToRead)
```

Description

Read the content from the file to the specified memory buffer, starting at the current file position.

Parameters

#File The file to read from.

***MemoryBuffer** The address to write the read data to.

LengthToRead The number of bytes to read. The maximum length is 2 GB.

Return value

Returns the number of bytes actually read from the file. If there is an error, the return value is zero.

Remarks

For a code example look at the Lof() function.

See Also

WriteData() , OpenFile() , ReadFile() , Loc()

117.23 ReadString

Syntax

```
Text\$ = ReadString(#File [, Flags [, Length]])
```

Description

Read a string from a file until an 'End Of Line' or a 'Null' character is found (Unix, DOS and Macintosh file formats are supported).

Parameters

#File The file to read from.

Flags (optional) The flags to apply while reading the string. It may be one of the following values:

```
#PB_Ascii    : reads the string as ASCII
#PB_UTF8     : reads the string as UTF8
#PB_Unicode  : reads the string as UTF16
```

combined with:

```
#PB_File_IgnoreEOL: ignores the end of line (but the
                    resulting string will still contain them) until the specified
                    length or the end of file.
```

If this flag isn't set, then the format for reading the string depends on the related setting at the previously used `CreateFile()` , `OpenFile()` or `ReadFile()` command.

Length (optional) Read the file until the length (in characters) have been reached. If an end of line is encountered before the length is reached, the read will stop (unless the flag `#PB_File_IgnoreEOL` has been set).

Return value

Returns the read string, or an empty string if the read has failed.

Remarks

For detecting the string encoding format (byte order mark) used in a file there is the `ReadStringFormat()` function available.

For an example see the `ReadFile()` .

See Also

`WriteString()` , `ReadStringFormat()` , `OpenFile()` , `ReadFile()`

117.24 ReadStringFormat

Syntax

```
Result = ReadStringFormat(#File)
```

Description

Checks if the current file position contains a BOM (Byte Order Mark) and tries to identify the String encoding used in the file.

Parameters

#File The file to use.

Return value

Returns one of the following values:

```
#PB_Ascii    : No BOM detected. This usually means a plain text
               file.
#PB_UTF8     : UTF-8 BOM detected.
#PB_Unicode  : UTF-16 (little endian) BOM detected.

#PB_UTF16BE  : UTF-16 (big endian) BOM detected.
#PB_UTF32    : UTF-32 (little endian) BOM detected.
#PB_UTF32BE  : UTF-32 (big endian) BOM detected.
```

The `#PB_Ascii`, `#PB_UTF8` and `#PB_Unicode` results may be used directly in further calls to `ReadString()` to read the file. The other results represent string formats that cannot be directly read with PureBasic string functions. They are included for completeness so that an application can display a proper error-message.

Remarks

If a BOM is detected, the file pointer will be placed at the end of the BOM. If no BOM is detected, the file pointer remains unchanged.

The Byte Order Mark is a commonly used way to indicate the encoding for a textfile. It is usually placed at the beginning of the file. It is however not a standard, just a commonly used practice. So if no BOM is detected at the start of a file, it does not necessarily mean that it is a plain text file. It could also just mean that the program that created the file did not use this practice.

`WriteStringFormat()` may be used to place such a BOM in a file.

For more information, see this [Wikipedia Article](#).

More information about using unicode in a PureBasic program can also be found [here](#) .

See Also

`WriteStringFormat()` , `ReadString()` , `OpenFile()` , `ReadFile()`

117.25 ReadUnicodeCharacter

Syntax

```
Number.u = ReadUnicodeCharacter(#File)
```

Description

Read a unicode character (2 bytes) from a file, starting at the current file position.

Parameters

#File The file to read from.

Return value

Returns the read character or zero if there was an error.

Remarks

For an example of how to read from a file, see the `ReadFile()` function - with `ReadUnicodeCharacter()` only a unicode character is read, instead of a complete line (string).

See Also

`WriteUnicodeCharacter()` , `ReadAsciiCharacter()` , `ReadCharacter()` , `OpenFile()` , `ReadFile()` , `Loc()`

117.26 ReadWord

Syntax

```
Number.w = ReadWord(#File)
```

Description

Read a word (2 bytes) from a file, starting at the current file position.

Parameters

#File The file to read from.

Return value

Returns the read number or zero if there was an error.

Remarks

For an example of how to read from a file, see the `ReadFile()` function - with `ReadWord()` only a word value is read, instead of a complete line (string).

See Also

`WriteWord()` , `OpenFile()` , `ReadFile()` , `Loc()`

117.27 WriteAsciiCharacter

Syntax

```
Result = WriteAsciiCharacter(#File, Number.a)
```

Description

Write an ASCII character (1 byte) to a file.

Parameters

#File The file to write to.

Number The ASCII character value to write.

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

Because of file buffering , this function may return successful even if there is not enough space left on the output device for the write operation. The file must be opened using a write-capable function (i.e. not with `ReadFile()`).

For an example see the `CreateFile()` function - with `WriteAsciiCharacter()` only an ASCII character is written, instead of a string.

See Also

`ReadAsciiCharacter()` , `WriteUnicodeCharacter()` , `WriteCharacter()` , `CreateFile()` , `OpenFile()`

117.28 WriteByte

Syntax

```
Result = WriteByte(#File, Number.b)
```

Description

Write a byte number (1 byte) to a file.

Parameters

#File The file to write to.

Number The value to write.

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

Because of file buffering , this function may return successful even if there is not enough space left on the output device for the write operation. The file must be opened using a write-capable function (i.e. not with `ReadFile()`).

For an example see the `CreateFile()` function - with `WriteByte()` only a byte number is written, instead of a string.

See Also

`ReadByte()` , `CreateFile()` , `OpenFile()`

117.29 WriteCharacter

Syntax

```
Result = WriteCharacter(#File, Character.c [, Format])
```

Description

Write a character number (1 byte in ASCII, 2 bytes in unicode) to a file.

Parameters

#File The file to write to.

Character The character value to write.

Format (optional) The format of the character to write. It can be one of the following value:

```
#PB_Ascii    : 1 byte character.
#PB_Unicode  : 2 bytes character (default, see unicode
mode).
#PB_UTF8     : multi-bytes character (from 1 to 4 bytes).
```

If this flag isn't set, then the format for writing the character depends on the related setting at the previously used `CreateFile()` or `OpenFile()` command.

Return value

Returns nonzero if the operation was successful or zero if it failed.

Remarks

Because of file buffering, this function may return successful even if there is not enough space left on the output device for the write operation. The file must be opened using a write-capable function (i.e. not with `ReadFile()`).

For an example see the `CreateFile()` function - with `WriteCharacter()` only a character number is written, instead of a string.

See Also

`ReadCharacter()` , `WriteAsciiCharacter()` , `WriteUnicodeCharacter()` , `CreateFile()` , `OpenFile()`

117.30 WriteDouble

Syntax

```
Result = WriteDouble(#File, Number.d)
```

Description

Write a double number (8 bytes) to a file.

Parameters

#File The file to write to.

Number.d The value to write.

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

Because of file buffering, this function may return successful even if there is not enough space left on the output device for the write operation. The file must be opened using a write-capable function (i.e. not with `ReadFile()`).

For an example see the `CreateFile()` function - with `WriteDouble()` only a double number is written, instead of a string.

See Also

ReadDouble() , WriteFloat() , CreateFile() , OpenFile()

117.31 WriteFloat

Syntax

```
Result = WriteFloat(#File, Number.f)
```

Description

Write a float number (4 bytes) to a file.

Parameters

#File The file to write to.

Number.f The float value to write.

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

Because of file buffering , this function may return successful even if there is not enough space left on the output device for the write operation. The file must be opened using a write-capable function (i.e. not with ReadFile()).

For an example see the CreateFile() function - with WriteFloat() only a float number is written, instead of a string.

See Also

ReadFloat() , WriteDouble() , CreateFile() , OpenFile()

117.32 WriteInteger

Syntax

```
Result = WriteInteger(#File, Number)
```

Description

Write an integer number (4 bytes in 32-bit executable, 8 bytes in 64-bit executable) to a file.

Parameters

#File The file to write to.

Number The value to write.

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

Because of file buffering , this function may return successful even if there is not enough space left on the output device for the write operation. The file must be opened using a write-capable function (i.e. not with `ReadFile()`).

For an example see the `CreateFile()` function - with `WriteInteger()` only a integer variable is written, instead of a string.

See Also

`ReadInteger()` , `CreateFile()` , `OpenFile()`

117.33 WriteLong

Syntax

```
Result = WriteLong(#File, Number)
```

Description

Write a long number (4 bytes) to a file.

Parameters

#File The file to write to.

Number The value to write.

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

Because of file buffering , this function may return successful even if there is not enough space left on the output device for the write operation. The file must be opened using a write-capable function (i.e. not with `ReadFile()`).

For an example see the `CreateFile()` function - with `WriteLong()` only a long variable is written, instead of a string.

See Also

`ReadLong()` , `CreateFile()` , `OpenFile()`

117.34 WriteData

Syntax

```
Result = WriteData(#File, *MemoryBuffer, Length)
```

Description

Write the content of the specified memory buffer to a file.

Parameters

#File The file to write to.

***MemoryBuffer** The memory address of the data to write to the file.

Length The number of bytes to write to the file. The maximum length is 2 GB.

Return value

Returns the number of bytes actually written to the file. If there is an error, the return-value is zero.

Remarks

Because of file buffering , this function may return successful even if there is not enough space left on the output device for the write operation. The file must be opened using a write-capable function (i.e. not with ReadFile()).

Example

```
1
2  *MemoryID = AllocateMemory(1000)           ; allocating a memory block
3  If *MemoryID
4      PokeS(*MemoryID, "Store this string in the memory area")    ; we
        write a string into the memory block
5  EndIf
6  If CreateFile(0, GetHomeDirectory()+"Text.txt")           ; we
        create a new text file...
7      WriteData(0, *MemoryID, SizeOf(Character)*Len("Store this
        string in the memory area"))           ; write the text from the
        memory block into the file
8      CloseFile(0)                                           ; close the previously
        opened file and so store the written data
9  Else
10     Debug "may not create the file!"
11 EndIf
```

See Also

ReadData() , CreateFile() , OpenFile()

117.35 WriteQuad

Syntax

Result = WriteQuad(#File, Number.q)

Description

Write a quad number (8 bytes) to a file.

Parameters

#File The file to write to.

Number.q The value to write.

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

Because of file buffering , this function may return successful even if there is not enough space left on the output device for the write operation. The file must be opened using a write-capable function (i.e. not with ReadFile()).

For an example see the CreateFile() function - with WriteQuad() only a quad number is written, instead of a string.

See Also

ReadQuad() , CreateFile() , OpenFile()

117.36 WriteString

Syntax

```
Result = WriteString(#File, Text$ [, Format])
```

Description

Write a string to a file.

Parameters

#File The file to write to.

Text\$ The string to write.

Format (optional) The format in which to write the string. It can be one of the following values:

```
#PB_Ascii    : Writes the string in ASCII format
#PB_UTF8     : Writes the string in UTF8 format
#PB_Unicode  : Writes the string in UTF16 format
```

If this flag isn't set, then the format for writing the string depends on the related setting at the previously used CreateFile() or OpenFile() command.

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

Because of file buffering , this function may return successful even if there is not enough space left on the output device for the write operation. The file must be opened using a write-capable function (i.e. not with ReadFile()). The null ending string character is not written to the file.

To place a BOM (byte order mark) to later identify the string encoding format of a file use the WriteStringFormat() function. To write a string including a newline sequence, use the WriteStringN() function.

For an example see the CreateFile() function.

See Also

ReadString() , WriteStringN() , WriteStringFormat() , CreateFile() , OpenFile()

117.37 WriteStringFormat

Syntax

```
Result = WriteStringFormat(#File, Format)
```

Description

Writes a BOM (Byte Order Mark) at the current position in the file.

Parameters

#File The file to write to.

Format The format for which the mark should be written. It can be one of the following values:

```
#PB_Ascii    : Writes no BOM at all (this is usually
               interpreted as a plain ASCII file.)
#PB_UTF8     : UTF-8 BOM
#PB_Unicode  : UTF-16 (little endian) BOM

#PB_UTF16BE  : UTF-16 (big endian) BOM
#PB_UTF32    : UTF-32 (little endian) BOM
#PB_UTF32BE  : UTF-32 (big endian) BOM
```

The `#PB_Ascii`, `#PB_UTF8` and `#PB_Unicode` correspond to the flags supported by `WriteString()` and `WriteStringN()` . After placing such a BOM, the strings which follow should all be written with this flag. The other formats represent string formats that can not be written directly with the PureBasic string functions. They are included only for completeness.

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

Because of file buffering , this function may return successful even if there is not enough space left on the output device for the write operation. The file must be opened using a write-capable function (i.e. not with `ReadFile()`).

The Byte Order Mark is a commonly used method with which to indicate the encoding of a textfile. It is usually placed at the beginning of the file. It is however not a standard, just a commonly used practice. So if no BOM is detected at the start of a file, it does not necessarily mean that it is a plain text file. It could also just mean that the program that created the file did not use this practice. `ReadStringFormat()` may be used detect a BOM within a file.

For more information, see this [Wikipedia Article](#).

More information about using unicode in PureBasic program can also be found here .

See Also

ReadStringFormat() , WriteString() , WriteStringN() , CreateFile() , OpenFile()

117.38 WriteStringN

Syntax

```
Result = WriteStringN(#File, Text$ [, Format])
```

Description

Write a string to a file and add an 'end of line' character.

Parameters

#File The file to write to.

Text\$ The string to write.

Format (optional) The format in which to write the string. It can be one of the following values:

```
#PB_Ascii    : Writes the string in ASCII format
#PB_UTF8     : Writes the string in UTF8 format
#PB_Unicode  : Writes the string in UTF16 format
```

If this flag isn't set, then the format for writing the string depends on the related setting at the previously used CreateFile() or OpenFile() command.

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

Because of file buffering, this function may return successful even if there is not enough space left on the output device for the write operation. The file must be opened using a write-capable function (i.e. not with ReadFile()).

To place a BOM (byte order mark) to later identify the string encoding format of a file use the WriteStringFormat() function. To write a string without a newline sequence, use the WriteString() function.

For an example see the CreateFile() function.

See Also

ReadString() , WriteString() , WriteStringFormat() , CreateFile() , OpenFile()

117.39 WriteUnicodeCharacter

Syntax

```
Result = WriteUnicodeCharacter(#File, Number)
```

Description

Write a unicode character (2 bytes) to a file.

Parameters

#File The file to write to.

Number The unicode character value to write.

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

Because of file buffering , this function may return successful even if there is not enough space left on the output device for the write operation. The file must be opened using a write-capable function (i.e. not with ReadFile()).

For an example see the CreateFile() function - with WriteUnicodeCharacter() only a unicode character is written, instead of a string.

See Also

ReadUnicodeCharacter() , WriteAsciiCharacter() , WriteCharacter() , CreateFile() , OpenFile()

117.40 WriteWord

Syntax

```
Result = WriteWord(#File, Number)
```

Description

Write a word number (2 bytes) to a file.

Parameters

#File The file to write to.

Number The value to write.

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

Because of file buffering , this function may return successful even if there is not enough space left on the output device for the write operation. The file must be opened using a write-capable function (i.e. not with ReadFile()).

For an example see the CreateFile() function - with WriteWord() only a word variable is written, instead of a string.

See Also

ReadWord() , CreateFile() , OpenFile()

Chapter 118

FileSystem

Overview

FileSystem is a generic term which deals with all advanced file related manipulations. For example, it's possible to read a directory content, create a new directory and more...

If you want to examine the contents of a directory start with the `ExamineDirectory()` function. All file and directory names in this library can be specified either with a full or a relative path. Relative paths are interpreted relative to the current directory of the program. The `GetCurrentDirectory()` and `SetCurrentDirectory()` functions can be used to modify the current directory.

Specific OS path separator characters are available `#PS`, `#NPS`, `#PS$` ('\\') and `#NPS$` ('/'). In addition to the functions in this library you will also find functions for manipulating file contents in the File library. To get the name of a running program use the function `ProgramFilename()` from the Process library.

118.1 CopyDirectory

Syntax

```
Result = CopyDirectory(SourceDirectory$, DestinationDirectory$,  
    Pattern$ [, Mode])
```

Description

Copy the contents of the source directory to the destination.

Parameters

SourceDirectory\$ The directory to copy.

DestinationDirectory\$ The destination to copy the directory to.

Pattern\$ A pattern identifying the files to copy. For example: `"*.exe"` will copy any files in the directory. `"*.exe"` will copy only the .exe files. By default, a null `Pattern$` ("") will copy all the files.

Mode (optional) Options for the copy operation. It can be a combination of the following values:

```
#PB_FileSystem_Recursive: Copy the directory with all  
    subdirectories.  
#PB_FileSystem_Force    : Overwrites the files which are  
    protected (read-only).
```

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

If the target directory already exists, its content will be overwritten automatically. The `FileSize()` function can be used to determine if the target exists or not.

Example

```
1  Debug CopyDirectory("D:\Games\MyGame\","D:\Games\Backup\","",  
    #PB_FileSystem_Recursive)
```

See Also

`CreateDirectory()` , `ExamineDirectory()` , `DeleteDirectory()`

118.2 CopyFile

Syntax

```
Result = CopyFile(SourceFilename$, DestinationFilename$)
```

Description

Copy the source file to the destination.

Parameters

SourceFilename\$ The file to copy.

DestinationFilename\$ The location to copy the file to.

Return value

Returns nonzero if the operation was successful or zero if it failed.

Remarks

If the destination file already exists, it will be overwritten. The `FileSize()` function can be used to determine if the target exists or not. If source file and destination file are the same, a copy will not occur and zero will be returned.

See Also

`RenameFile()` , `DeleteFile()` , `FileSize()` , `CreateFile()` , `OpenFile()`

118.3 CreateDirectory

Syntax

```
Result = CreateDirectory(DirectoryName$)
```

Description

Creates a new directory.

Parameters

DirectoryName\$ The name of the new directory.

Return value

Returns nonzero if the operation was successful or zero if it failed.

Remarks

This function fails if the parent directory of the new directory does not exist. To create multiple levels of directories, this function has to be called for each directory level to create separately.

See Also

CopyDirectory() , ExamineDirectory() , DeleteDirectory()

118.4 DeleteDirectory

Syntax

```
Result = DeleteDirectory(Directory$, Pattern$ [, Mode])
```

Description

Deletes the specified Directory\$ or files in that Directory\$ matching the provided pattern.

Parameters

Directory\$ The directory to delete.

Pattern\$ A pattern for deleting files within the directory. For example: "*.*)" will delete any files in the directory. "*.exe" will delete only the .exe files. By default, a null Pattern\$ ("") will delete all the files.

Mode (optional) Options for the delete operation. It can be a combination of the following values:

```
#PB_FileSystem_Recursive: Deletes the directory with all
                        subdirectories.
#PB_FileSystem_Force    : Deletes even the files which are
                        protected (read-only).
```

Return value

Returns nonzero if the operation was successful or zero if it failed.

See Also

CreateDirectory() , ExamineDirectory() , CopyDirectory()

118.5 DeleteFile

Syntax

```
Result = DeleteFile(Filename$ [, Mode])
```

Description

Deletes the specified file.

Parameters

Filename\$ The file to delete.

Mode (optional) Options for the delete operation. It can be one of the following values:

```
#PB_FileSystem_Force: Also deletes the files which are  
protected (read-only).
```

Return value

Returns nonzero if the operation was successful or zero if it failed.

See Also

CopyFile() , RenameFile() , FileSize() , CreateFile() , OpenFile()

118.6 DirectoryEntryAttributes

Syntax

```
Attributes = DirectoryEntryAttributes(#Directory)
```

Description

Returns the attributes of the current entry in the directory being listed with ExamineDirectory() and NextDirectoryEntry() functions.

Parameters

#Directory The directory examined with ExamineDirectory() .

Return value

Returns the attributes of the current file.

On Windows, Attributes is a combination of the following values:

```

#PB_FileSystem_Hidden      : File is hidden
#PB_FileSystem_Archive    : File has been changed and not archived
                           since the last time
#PB_FileSystem_Compressed : File is compressed
#PB_FileSystem_Normal     : Normal attributes
#PB_FileSystem_ReadOnly   : File is in read-only mode
#PB_FileSystem_System     : File is a system file

```

On Linux or MacOSX, Attributes is a combination of these values:

```

#PB_FileSystem_Link       : The file is a symbolic link
#PB_FileSystem_ReadUser   : Access flags for the owning user
#PB_FileSystem_WriteUser
#PB_FileSystem_ExecUser
#PB_FileSystem_ReadGroup  : Access flags for the owning user's
                           group
#PB_FileSystem_WriteGroup
#PB_FileSystem_ExecGroup
#PB_FileSystem_ReadAll    : Access flags for all other users
#PB_FileSystem_WriteAll
#PB_FileSystem_ExecAll

```

Remarks

To check if one attribute is actually set, just use the '&' (binary AND) and the attribute constant value:

```

1  [...]
2
3  FileAttributes = DirectoryEntryAttributes(#Directory)
4  If FileAttributes & #PB_FileSystem_Hidden
5      Debug "This file is hidden !"
6  EndIf

```

See Also

ExamineDirectory() , NextDirectoryEntry() , DirectoryEntryType() , DirectoryEntryName() ,
DirectoryEntrySize() , DirectoryEntryDate()

118.7 DirectoryEntryDate

Syntax

```
Result = DirectoryEntryDate(#Directory, DateType)
```

Description

Returns the date of the current entry in the directory being listed with ExamineDirectory() and NextDirectoryEntry() functions.

Parameters

#Directory The directory examined with ExamineDirectory() .

DateType The kind of date to return. It can be one of the following values:

```

#PB_Date_Created : returns the file creation date.
#PB_Date_Accessed: returns the last file access date.
#PB_Date_Modified: returns the last file modification date.

```

Return value

Returns the specified date of the current directory entry in the format of the PureBasic Date library.

Remarks

On Linux and Mac OSX, the date returned for `#PB_Date_Created` is the same as the date for `#PB_Date_Modified`, because most file systems do not store a file creation date.

See Also

`ExamineDirectory()` , `NextDirectoryEntry()` , `DirectoryEntryType()` , `DirectoryEntryName()` , `DirectoryEntrySize()` , `DirectoryEntryAttributes()`

118.8 DirectoryEntryName

Syntax

```
Filename\$ = DirectoryEntryName(#Directory)
```

Description

Returns the name of the current entry in the directory being listed with `ExamineDirectory()` and `NextDirectoryEntry()` functions.

Parameters

`#Directory` The directory examined with `ExamineDirectory()` .

Return value

Returns the name of the current directory entry.

Remarks

The pseudo-directories `."` and `.."` can be returned in a directory enumeration, so they have to be filtered if they should not be included in the program output.

See Also

`ExamineDirectory()` , `NextDirectoryEntry()` , `DirectoryEntryType()` , `DirectoryEntrySize()` , `DirectoryEntryAttributes()` , `DirectoryEntryDate()`

118.9 DirectoryEntryType

Syntax

```
Result = DirectoryEntryType(#Directory)
```

Description

Returns the type of the current entry in the directory being listed with `ExamineDirectory()` and `NextDirectoryEntry()` functions.

Parameters

#Directory The directory examined with `ExamineDirectory()` .

Return value

Returns one of the following values:

```
#PB_DirectoryEntry_File      : This entry is a file.  
#PB_DirectoryEntry_Directory: This entry is a directory.
```

See Also

`ExamineDirectory()` , `NextDirectoryEntry()` , `DirectoryEntryName()` , `DirectoryEntrySize()` , `DirectoryEntryAttributes()` , `DirectoryEntryDate()`

118.10 DirectoryEntrySize

Syntax

```
Size.q = DirectoryEntrySize(#Directory)
```

Description

Returns the size of the current entry in the directory being listed with `ExamineDirectory()` and `NextDirectoryEntry()` functions.

Parameters

#Directory The directory examined with `ExamineDirectory()` .

Return value

Returns the size of the current directory entry in bytes.

See Also

`ExamineDirectory()` , `NextDirectoryEntry()` , `DirectoryEntryType()` , `DirectoryEntryName()` , `DirectoryEntryAttributes()` , `DirectoryEntryDate()`

118.11 ExamineDirectory

Syntax

```
Result = ExamineDirectory(#Directory, DirectoryName$, Pattern$)
```

Description

Start to examine a directory for listing with the functions `NextDirectoryEntry()` , `DirectoryEntryName()` and `DirectoryEntryType()` .

Parameters

#Directory A number to identify the new directory listing. **#PB_Any** can be used as a parameter to auto-generate this number.

DirectoryName\$ The directory to examine.

Pattern\$ A pattern to filter the returned entries by. For example: A 'Pattern\$' like `"*.**"` or `"**"` will list all the files (and sub-directories) in the directory. A 'Pattern\$' like `"*.exe"` will list only .exe files (and sub-directories ending with .exe if any). Please be aware of a MS Windows specific behaviour: When using a 3-char file name extension any extension starting with those 3 chars will match, e.g. `"*.log"` will also match a file name/extension `"test.log_1"`.

Return value

Returns nonzero if the directory can be enumerated or zero if there was an error. If **#PB_Any** was used as the **#Directory** parameter then the generated directory number is returned.

Remarks

Once the enumeration is done, `FinishDirectory()` must be called to free the resources associated to the listing.

Specific OS path separator characters are available **#PS**, **#NPS**, **#PS\$** (`'\'`) and **#NPS\$** (`'/'`).

Example

```
1  Directory$ = GetHomeDirectory() ; Lists all files and folder in
   the home directory
2  If ExamineDirectory(0, Directory$, "**.*")
3      While NextDirectoryEntry(0)
4          If DirectoryEntryType(0) = #PB_DirectoryEntry_File
5              Type$ = "[File] "
6              Size$ = " (Size: " + DirectoryEntrySize(0) + ")"
7          Else
8              Type$ = "[Directory] "
9              Size$ = "" ; A directory doesn't have a size
10         EndIf
11
12         Debug Type$ + DirectoryEntryName(0) + Size$
13     Wend
14     FinishDirectory(0)
15 EndIf
```

See Also

`FinishDirectory()` , `NextDirectoryEntry()` , `DirectoryEntryType()` , `DirectoryEntryName()` ,
`DirectoryEntrySize()` , `DirectoryEntryAttributes()` , `DirectoryEntryDate()`

118.12 FinishDirectory

Syntax

```
FinishDirectory(#Directory)
```

Description

Finish the enumeration started with `ExamineDirectory()` . This frees the resources associated with the `#Directory` listing.

Parameters

#Directory The directory examined with `ExamineDirectory()` .

Return value

None.

See Also

`ExamineDirectory()`

118.13 GetExtensionPart

Syntax

```
Extension\$$ = GetExtensionPart(FullPathName$)
```

Description

Retrieves the file extension part of a full path.

Parameters

FullPathName\$ The full path to get the extension from.

Return value

Returns the file extension. For example, if the full path is "C:\PureBasic\PB.exe", the result will be "exe".

Remarks

To retrieve the file or the path part from a full path, look at the `GetFilePart()` and `GetPathPart()` functions. If the path does not contain a file name then the result may be random. For example `GetExtensionPart(".htaccess")` or `GetExtensionPart(".access.")` are not valid.

See Also

`GetFilePart()` , `GetPathPart()`

118.14 GetFilePart

Syntax

```
Filename\$$ = GetFilePart(FullPathName$ [, Mode])
```

Description

Retrieves the file part of a full path.

Parameters

FullPathName\$ The full path to get the filename from.

Mode (optional) It can be one of the following values:

```
#PB_FileSystem_NoExtension: Get the filename without its  
extension (if any).
```

Return value

Returns the file name. For example, if the full path is "C:\PureBasic\PB.exe", the result will be "PB.exe".

Remarks

To retrieve the extension or the path part from a full path, look at the `GetExtensionPart()` and `GetPathPart()` functions.

See Also

`GetExtensionPart()` , `GetPathPart()`

118.15 GetPathPart

Syntax

```
Path\$ = GetPathPart(FullPathName$)
```

Description

Retrieves the path part of a full path.

Parameters

FullPathName\$ The full path to get the path part from.

Return value

Returns the path part. For example, if the full path is "C:\PureBasic\PB.exe", the result will be "C:\PureBasic\".

Remarks

To retrieve the extension or the file part from a full path, look at the `GetExtensionPart()` and `GetFilePart()` functions.

Example

```
1  Debug GetPathPart(ProgramFilename())
```

See Also

`GetExtensionPart()` , `GetFilePart()` , `ProgramFilename()`

118.16 IsDirectory

Syntax

```
Result = IsDirectory(#Directory)
```

Description

Tests if the given directory number is a valid and correctly initialized directory enumeration.

Parameters

#Directory The directory enumeration to test.

Return value

Returns nonzero if the given input is a valid enumeration and zero otherwise.

Remarks

This function is bulletproof and can be used with any value. This is the correct way to ensure a directory is ready to use.

See Also

ExamineDirectory()

118.17 CheckFilename

Syntax

```
Result = CheckFilename(Filename$)
```

Description

Checks if the specified Filename\$ doesn't contain invalid characters for the file-system. For example, on Windows '/' and '\' characters are not allowed in the filename.

Parameters

Filename\$ The filename to check without a path.

Return value

Returns nonzero if the filename does not contain invalid characters and zero if it does.

Remarks

Even if the syntax-check of this function doesn't complain, there are different 'forbidden' filenames on different OS. For example on Windows filenames containing "COM1" till "COM9", "LPT1" till "LPT9" or "aux" are not allowed. For more information see [here](#).

118.18 FileSize

Syntax

```
Result.q = FileSize(Filename$)
```

Description

Returns the size of the specified file. This function can also be used to check if a file or directory exists or not.

Parameters

Filename\$ The filename to get the size from.

Return value

Returns the size of the file in bytes, or one of the following values:

```
-1: File not found.  
-2: File is a directory.
```

See Also

DirectoryEntrySize()

118.19 GetCurrentDirectory

Syntax

```
Result\$ = GetCurrentDirectory()
```

Description

Returns the current directory for the program.

Parameters

None.

Return value

Returns the full path of the current directory. It will end with a path separator **#PS**, **#PS\$** ('\' for Windows or '/' otherwise).

It's very unlikely, but if this function fails, it will return an empty string.

Remarks

All files accesses are relative to this directory, when no absolute path is specified. To change the current directory, use SetCurrentDirectory() .

See Also

SetCurrentDirectory() , GetHomeDirectory() , GetUserDirectory() , GetTemporaryDirectory()

118.20 GetHomeDirectory

Syntax

```
Result\$ = GetHomeDirectory()
```

Description

Returns the home directory path of the currently logged user.

Parameters

None.

Return value

Returns the full path of the home directory. It will end with a path separator `#PS`, `#PS$` (`'\'` for Windows or `'/'` otherwise).

It's very unlikely, but if this function fails, it will return an empty string.

Remarks

The home directory is the directory for the user data (preferences, plugins etc.) of the current user. Read and write access should be possible in this directory.

See Also

`GetCurrentDirectory()` , `GetTemporaryDirectory()` , `GetUserDirectory()`

118.21 GetUserDirectory

Syntax

```
Result\$ = GetUserDirectory(Type)
```

Description

Returns the directory path of the specified directory type.

Parameters

Type The type of directory to get the path. It can be one of the following value:

<code>#PB_Directory_Desktop</code>	: desktop directory of the current logged user
<code>#PB_Directory_Downloads</code>	: downloads directory of the current logged user
<code>#PB_Directory_Documents</code>	: documents directory path of the current logged user
<code>#PB_Directory_Videos</code>	: videos directory path of the current logged user
<code>#PB_Directory_Musics</code>	: musics directory path of the current logged user
<code>#PB_Directory_Pictures</code>	: pictures directory path of the current logged user

```

#PB_Directory_Public      : public directory of the current
                           logged user
#PB_Directory_ProgramData: program data directory of the
                           current logged user.

                           On Linux and OSX, it is the home
                           directory followed by './' to be able to create
                           hidden config directory in the
                           home user.
#PB_Directory_AllUserData: common program data directory
                           (accessible to all users)
#PB_Directory_Programs    : global program files path (can be
                           read only)

```

Return value

Returns the full path of the specified directory. It will end with a path separator `#PS`, `#PS$` (`'\'` for Windows or `'/'` otherwise). If the type is not found it will return an empty string.

Example

```

1 ; List some of the user directories:
2 Debug "#PB_Directory_Downloads = " +
   GetUserDirectory(#PB_Directory_Downloads)
3 Debug "#PB_Directory_Documents = " +
   GetUserDirectory(#PB_Directory_Documents)
4 Debug "#PB_Directory_Programs = " +
   GetUserDirectory(#PB_Directory_Programs)

```

See Also

`GetCurrentDirectory()` , `GetHomeDirectory()` , `GetTemporaryDirectory()`

118.22 GetTemporaryDirectory

Syntax

```
Result\$$ = GetTemporaryDirectory()
```

Description

Returns the temporary directory name.

Parameters

None.

Return value

Returns the full path to the temporary directory. It will end with a path separator `#PS`, `#PS$` (`'\'` for Windows or `'/'` otherwise). It's very unlikely, but if this function fails, it will return an empty string.

Remarks

This directory provides read and write access and should be used to store temporary files. Depending on the operating system and settings, the contents of this directory may be deleted on a regular basis. It is however better to delete temporary files when they are no longer needed and not rely on such automatic cleanup.

See Also

GetCurrentDirectory() , GetHomeDirectory() , GetUserDirectory()

118.23 GetFileDate

Syntax

```
Result = GetFileDate(Filename$, DateType)
```

Description

Returns the date of the specified file.

Parameters

Filename\$ The file to get the date from.

DateType The kind of date to return. It can be one of the following values:

```
#PB_Date_Created : returns the file creation date.  
#PB_Date_Accessed: returns the last file access date.  
#PB_Date_Modified: returns the last file modification date.
```

Return value

Returns the requested date in the format of the PureBasic Date library.

Remarks

On Linux and Mac OSX, the date returned for `#PB_Date_Created` is the same as the date for `#PB_Date_Modified`, because most file systems do not store a file creation date.

See Also

SetFileDate() , DirectoryEntryDate()

118.24 GetFileAttributes

Syntax

```
Attributes = GetFileAttributes(Filename$)
```

Description

Returns the attributes of the given file.

Parameters

Filename\$ The file to read the attributes from. This can also specify the name of a directory.

Return value

Returns the attributes of the file. If the file does not exist or the attributes cannot be read, the result is the value -1.

On Windows, Attributes is a combination of the following values:

```
#PB_FileSystem_Hidden      : File is hidden
#PB_FileSystem_Archive     : File has been changed and not archived
                           since the last time
#PB_FileSystem_Compressed : File is compressed
#PB_FileSystem_Normal      : Normal attributes
#PB_FileSystem_ReadOnly    : File is in read-only mode
#PB_FileSystem_System      : File is a system file
```

On Linux or MacOSX, Attributes is a combination of these values:

```
#PB_FileSystem_Link        : The file is a symbolic link
#PB_FileSystem_ReadUser    : Access flags for the owning user
#PB_FileSystem_WriteUser
#PB_FileSystem_ExecUser
#PB_FileSystem_ReadGroup   : Access flags for the owning user's
                           group
#PB_FileSystem_WriteGroup
#PB_FileSystem_ExecGroup
#PB_FileSystem_ReadAll     : Access flags for all other users
#PB_FileSystem_WriteAll
#PB_FileSystem_ExecAll
```

Remarks

To check if one attribute is actually set, just use the '&' (binary AND) and the attribute constant value:

```
1  FileAttributes = GetFileAttributes("C:\Text.txt")
2  If FileAttributes & #PB_FileSystem_Hidden
3      Debug "This file is hidden !"
4  EndIf
```

Example

```
1  Value = GetFileAttributes("c:\autoexec.bat")
2
3  If Value = -1
4      Debug "Error reading file attributes!"
5  Else
6      If Value & #PB_FileSystem_Hidden      : txt$ + "H" : Else :
txt$+"-" : EndIf
7      If Value & #PB_FileSystem_Archive     : txt$ + "A" : Else :
txt$+"-" : EndIf
8      If Value & #PB_FileSystem_Compressed : txt$ + "C" : Else :
txt$+"-" : EndIf
9      If Value & #PB_FileSystem_Normal      : txt$ + "N" : Else :
txt$+"-" : EndIf
```

```

10     If Value & #PB_FileSystem_ReadOnly      : txt$ + "R" : Else :
    txt$+"-" : EndIf
11     If Value & #PB_FileSystem_System        : txt$ + "S" : Else :
    txt$+"-" : EndIf
12     Debug txt$
13 EndIf

```

See Also

SetFileAttributes() , DirectoryEntryAttributes()

118.25 NextDirectoryEntry

Syntax

```
Result = NextDirectoryEntry(#Directory)
```

Description

This function must be called after an ExamineDirectory() . It will go step-by-step into the directory and list its contents.

Parameters

#Directory The directory examined with ExamineDirectory()

Return value

Returns nonzero if a new entry was read from the directory and zero if there are no more entries.

Remarks

The entry name can be read with the DirectoryEntryName() function. If you want to know whether an entry is a subdirectory or a file, use the DirectoryEntryType() function.

See Also

ExamineDirectory() , DirectoryEntryType() , DirectoryEntryName() , DirectoryEntrySize() , DirectoryEntryAttributes() , DirectoryEntryDate()

118.26 RenameFile

Syntax

```
Result = RenameFile(OldFilename$, NewFilename$)
```

Description

Rename the old file to the new file.

Parameters

OldFilename\$ The old name of the file.

NewFilename\$ The new name of the file.

Return value

Returns nonzero if the operation succeeded and zero if it failed.

Remarks

The old and new filename do not have to be in the same directory, so this function can be used to move a file to a different directory. This function can also be used to rename/move directories.

Example

```
1  If RenameFile("C:\test.txt", "D:\temp\test_backup.txt")
2      Debug "Moving and renaming successful." ; Moving and
    renaming of the file was successful
3  Else
4      Debug "Moving and renaming failed." ; Moving and
    renaming failed, e.g. because of a not found source file
5  EndIf
```

See Also

CopyFile() , DeleteFile() , CreateFile() , CopyDirectory()

118.27 SetFileDate

Syntax

```
Result = SetFileDate(Filename$, DateType, Date)
```

Description

Changes the date of the specified file.

Parameters

Filename\$ The name of the file to modify.

DateType The kind of date to modify. It can be one of the following values:

```
#PB_Date_Created : change the file creation date.
#PB_Date_Accessed: change the last file access date.
#PB_Date_Modified: change the last file modification date.
```

Date The date to set. This has to be a value from the Date library.

Return value

Returns nonzero if the operation was successful and zero otherwise.

Remarks

On Linux and Mac OSX, the date used for `#PB_Date_Created` is the same as the date for `#PB_Date_Modified`, because most file systems do not store a file creation date.

See Also

GetFileDate()

118.28 SetFileAttributes

Syntax

```
Result = SetFileAttributes(Filename$, Attributes)
```

Description

Set the attributes of the given Filename\$.

Parameters

Filename\$ The name of the file to modify. This can also specify the name of a directory.

Attributes The new attributes.

On Windows, 'Attributes' is a combination of the following values:

```
#PB_FileSystem_Hidden      : File is hidden
#PB_FileSystem_Archive     : File has been changed and not
    archived since the last time
#PB_FileSystem_Normal      : Normal attributes
#PB_FileSystem_ReadOnly    : File is in read-only mode
#PB_FileSystem_System      : File is a system file
```

On Linux and MacOSX, the following values can be used:

```
#PB_FileSystem_ReadUser    : Access flags for the owning user
#PB_FileSystem_WriteUser
#PB_FileSystem_ExecUser
#PB_FileSystem_ReadGroup   : Access flags for the owning user's
    group
#PB_FileSystem_WriteGroup
#PB_FileSystem_ExecGroup
#PB_FileSystem_ReadAll     : Access flags for all other users
#PB_FileSystem_WriteAll
#PB_FileSystem_ExecAll
```

To combine several attributes, just use the '||' (binary OR) operand:

```
1 SetFileAttributes("C:\Text.txt", #PB_FileSystem_Hidden |
    #PB_FileSystem_ReadOnly)
```

Return value

Returns nonzero if the operation was successful and zero otherwise.

See Also

GetFileAttributes()

118.29 SetCurrentDirectory

Syntax

`Result = SetCurrentDirectory(Directory$)`

Description

Changes the current directory for the program.

Parameters

Directory\$ The full path to the new current directory, or a path relative to the existing current directory.

Return value

Returns nonzero if the current directory has been successfully changed, return zero otherwise.

Remarks

All files accesses are relative to this directory, when no absolute path is specified. To get the current directory, use `GetCurrentDirectory()` .
Specific OS path separator characters are available `#PS`, `#NPS`, `#PS$` (`'\'`) and `#NPS$` (`'/'`).

See Also

`GetCurrentDirectory()`

Chapter 119

Font

Overview

Fonts are widely used on computers since it is the only way to render text in different sizes and forms.

Note: With PureBasic using a colored font is not yet possible. This may be achieved by using the `StartDrawing()` function and then drawing directly on the visual area.

119.1 FreeFont

Syntax

```
FreeFont (#Font)
```

Description

Free the given Font, previously loaded with `LoadFont()` .

Parameters

`#Font` The font to free. If `#PB_All` is specified, all the remaining fonts are freed.

Return value

None.

Remarks

All remaining fonts are automatically freed when the program ends.

See Also

`LoadFont()`

119.2 FontID

Syntax

```
FontID = FontID (#Font)
```

Description

Returns the unique system identifier of the #Font.

Parameters

#Font The font to use.

Return value

Returns the unique system ID of the font. This result is sometimes also referred to as a 'Handle'. Take a look at the extra chapter Handles and Numbers for more information.

Example

An example of using FontID() in combination with SetGadgetFont() :

```
1  If OpenWindow(0, 0, 0, 222, 130, "FontID()",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      ButtonGadget(0, 10, 10, 200, 30, "Click to change the font...")
4      Font1 = LoadFont(#PB_Any, "Arial" , 8, #PB_Font_Bold)
5      Font2 = LoadFont(#PB_Any, "Verdana", 12, #PB_Font_StrikeOut)
6      UsedFont = 1
7  EndIf
8
9  Repeat
10     Event = WaitWindowEvent()
11
12     If Event = #PB_Event_Gadget
13         If EventGadget() = 0
14             If UsedFont = 1
15                 SetGadgetFont(0, FontID(Font2))
16                 UsedFont = 2
17             Else
18                 SetGadgetFont(0, FontID(Font1))
19                 UsedFont = 1
20             EndIf
21         EndIf
22     EndIf
23 Until Event = #PB_Event_CloseWindow
```

See Also

LoadFont() , SetGadgetFont() , DrawingFont()

119.3 IsFont

Syntax

Result = **IsFont**(#Font)

Description

Tests if the given #Font is a valid and correctly initialized font.

Parameters

#Font The font to use.

Return value

Returns nonzero if #Font is a valid font and zero otherwise.

Remarks

This function is bulletproof and can be used with any value. If Result is not zero then the object is valid and initialized, otherwise it returns zero. This is a good way to check that a font is ready to use.

See Also

LoadFont()

119.4 LoadFont

Syntax

```
Result = LoadFont(#Font, Name$, YSize [, Flags])
```

Description

Tries to open the specified font.

Parameters

#Font A number to identify the new font. #PB_Any can be used to auto-generate this number.

Name\$ The name of the font to load.

YSize The vertical size of the font in points.

Flags (optional) A combination of the following constants:

```
#PB_Font_Bold      : The font will be bold
#PB_Font_Italic    : The font will be italic
#PB_Font_Underline : The font will be underlined (Windows
only)
#PB_Font_StrikeOut  : The font will be strikeout (Windows
only)
#PB_Font_HighQuality: The font will be in high-quality mode
(slower) (Windows only)
```

Return value

Returns nonzero if the font was loaded successfully and zero if not. If #PB_Any was used for the #Font parameter then the generated number is returned on success.

Remarks

If another font was previously loaded with the same #Font number, then the older font will be automatically freed when loading the new one.

On Windows and OS X the font mapper will always attempt to locate a font. If a font name which does not exist such as: "Tim Now Ronin", is used, then the font mapper will attempt to find the closest match. This will be based upon such criteria as: the font name, the font height, the style and so forth. Therefore, the assumption can not be made that a font will not be loaded, due only to an incorrect font name, size, etc.

On Windows and macOS, you can change the size and style of the application's system font without specifying its name like this: LoadFont(1, "", 24)

Example

```
1  If OpenWindow(0, 0, 0, 270, 160, "Loading font...",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      If LoadFont(1, "Arial", 24)
3          SetGadgetFont(#PB_Default, FontID(1))
4          TextGadget(0, 10, 10, 250, 40, "Arial 24")
5      EndIf
6      Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
7  EndIf
```

See Also

FontID() , FreeFont()

119.5 RegisterFontFile

Syntax

```
Result = RegisterFontFile(FileName$)
```

Description

Register a font file for use with the LoadFont() command. All fonts contained in the file are then available.

Parameters

FileName\$ The file containing the font. The file must be in TrueType format.

Return value

Returns nonzero if the file was successfully registered, zero otherwise.

Remarks

The font file is registered for the current program only. This means that the font(s) are not accessible by other programs and will be unregistered when the program ends. No system-wide changes are made by this command.

You cannot use a registered font with the VectorDrawing library.

Example

```
1   ; Now let's use a new font, which was downloaded from the
    internet to the temporary directory...
2   ; The font name is "ascii" and the font file is "ascii.ttf"
3   If RegisterFontFile(GetTemporaryDirectory() + "ascii.ttf") ; We
    have to register it before to use it
4       LoadFont(0, "ascii", 12) ; Now we can load the font, the
    operating system knows it
5       SetGadgetFont(0, FontID(0))
6   ...
```

See Also

LoadFont()

Chapter 120

Ftp

Overview

FTP (File Transfer Protocol) is a way to share files based amongst users on a client-server model. This library implement the client side of FTP and SFTP, and allows to connect to a remote server and manipulate files (like download, upload, list available files, navigate through the directories and more).

120.1 AbortFTPFile

Syntax

```
AbortFTPFile (#Ftp)
```

Description

Aborts the current background file transfer previously started with SendFTPFile() or ReceiveFTPFile() . If no file transfer is in progress, this function has no effect.

Parameters

#Ftp The FTP connection to use.

Return value

None.

See Also

SendFTPFile() , ReceiveFTPFile() , FTPProgress()

120.2 CheckFTPConnection

Syntax

```
Result = CheckFTPConnection (#Ftp)
```

Description

Checks if the specified **#Ftp** connection is still connected to the server.

Parameters

#Ftp The FTP connection to use.

Return value

Returns nonzero if the connection is still open and zero if the server closed the connection.

See Also

OpenFTP()

120.3 CloseFTP

Syntax

```
CloseFTP(#Ftp)
```

Description

Closes the specified #Ftp client connection previously opened with OpenFTP() and free any associated resources.

Parameters

#Ftp The connection to close. If **#PB_All** is specified, all the remaining FTP connection are closed.

Return value

None.

Remarks

All remaining ftp connection are automatically closed when the program ends.

See Also

OpenFTP() , IsFtp()

120.4 CreateFTPDirectory

Syntax

```
Result = CreateFTPDirectory(#Ftp, Directory$)
```

Description

Creates a new directory on the FTP server.

Parameters

#Ftp The FTP connection to use.

Directory\$ The name of the directory to create. The new directory will be created in the current directory (see `GetFTPDDirectory()` and `SetFTPDDirectory()`). It is not possible to specify a subpath in the `Directory$`.

Return value

Returns nonzero if the operation was successful and zero if it failed.

See Also

`DeleteFTPDDirectory()` , `SetFTPDDirectory()` , `GetFTPDDirectory()`

120.5 DeleteFTPDDirectory

Syntax

```
Result = DeleteFTPDDirectory(#Ftp, Directory$)
```

Description

Deletes a directory on the FTP server.

Parameters

#Ftp The connection to use.

Directory\$ The directory to delete. The directory has to be in the current directory (see `GetFTPDDirectory()` and `SetFTPDDirectory()`). It is not possible to specify a subpath in the `Directory$`. The directory has to be empty, or the delete with fail.

Return value

Returns nonzero if the directory was deleted or zero if the operation failed.

See Also

`CreateFTPDDirectory()` , `SetFTPDDirectory()` , `GetFTPDDirectory()`

120.6 DeleteFTPFile

Syntax

```
Result = DeleteFTPFile(#Ftp, Filename$)
```

Description

Deletes a file on the FTP server.

Parameters

#Ftp The connection to use.

Filename\$ The file to delete. The file has to be in the current directory (see GetFTPDirectory() and SetFTPDirectory()). It is not possible to specify a subpath in the Filename\$.

Return value

Returns nonzero if the file was deleted or zero if the operation failed.

See Also

SendFTPFile() , SetFTPDirectory() , GetFTPDirectory()

120.7 ExamineFTPDirectory

Syntax

```
Result = ExamineFTPDirectory(#Ftp)
```

Description

Starts to examine the content of the current FTP directory.

Parameters

#Ftp The connection to use.

Return value

Returns nonzero if the operation succeeded or zero if it failed.

Remarks

For now, only Unix-like servers are supported for directory listing. The FTP protocol doesn't specify how a directory listing has to be returned, so every server has its own way to return the directory information. Fortunately, most of the world servers are running under Unix/Linux and uses the same way to return the data. That said, there is no warranty that the listing works on every servers. This command will be updated on demand to automatically handle more servers type. If the server isn't supported, please use FTPDirectoryEntryRaw() to get back the raw information about each entry.

The functions NextFTPDirectoryEntry() , FTPDirectoryEntryName() , FTPDirectoryEntryType() , FTPDirectoryEntryAttributes() , FTPDirectoryEntryDate() and FTPDirectoryEntrySize() can be used to read the directory entries. To change the current directory, use SetFTPDirectory() .

Example

```
1  If OpenFTP(0, "ftp.free.fr", "anonymous", "")
2      If ExamineFTPDirectory(0)
3          While NextFTPDirectoryEntry(0)
4              Debug FTPDirectoryEntryName(0)
5          Wend
6          FinishFTPDirectory(0)
7      EndIf
```

```
8 Else
9     Debug "Can't connect to ftp.free.fr"
10 EndIf
```

See Also

NextFTPDirectoryEntry() , FinishFTPDirectory()

120.8 GetFTPDirectory

Syntax

```
Result\$$ = GetFTPDirectory(#Ftp)
```

Description

Returns the current FTP directory.

Parameters

#Ftp The connection to use.

Return value

Return the current FTP directory, relative to the FTP root.

Remarks

To change the current directory, use SetFTPDirectory() .

See Also

SetFTPDirectory() , ExamineFTPDirectory() , SendFTPFile() , ReceiveFTPFile()

120.9 FinishFTPDirectory

Syntax

```
FinishFTPDirectory(#Ftp)
```

Description

Finishes the enumeration started with ExamineFTPDirectory() . This allows to free the resources associated with the FTP listing.

Parameters

#Ftp The connection to use.

Return value

None.

See Also

ExamineFTPDirectory()

120.10 FTPDirectoryEntryAttributes

Syntax

```
Attributes = FTPDirectoryEntryAttributes(#Ftp)
```

Description

Returns the attributes of the current entry in the FTP enumeration being listed with the ExamineFTPDirectory() and NextFTPDirectoryEntry() functions.

Parameters

#Ftp The connection to use.

Return value

Returns a combination of the following values:

```
#PB_FTP_ReadUser    : Access flags for the owning user
#PB_FTP_WriteUser   :
#PB_FTP_ExecuteUser :
#PB_FTP_ReadGroup   : Access flags for the owning user's group
#PB_FTP_WriteGroup  :
#PB_FTP_ExecuteGroup:
#PB_FTP_ReadAll     : Access flags for all other users
#PB_FTP_WriteAll    :
#PB_FTP_ExecuteAll  :
```

Example

To check if one attribute is actually set, just use the '&' (binary AND) and the attribute constant value:

```
1  [...]
2
3  FileAttributes = FTPDirectoryEntryAttributes(#Ftp)
4  If FileAttributes & #PB_FTP_ReadUser
5      Debug "This file has the ReadUser flag"
6  EndIf
```

See Also

ExamineFTPDirectory() , NextFTPDirectoryEntry() , FTPDirectoryEntryType() ,
FTPDirectoryEntryName() , FTPDirectoryEntryDate() , FTPDirectoryEntrySize() ,
FTPDirectoryEntryRaw()

120.11 FTPDirectoryEntryDate

Syntax

```
Result = FTPDirectoryEntryDate(#Ftp)
```

Description

Returns the date of the current entry in the FTP enumeration being listed with ExamineFTPDirectory() and NextFTPDirectoryEntry() functions.

Parameters

#Ftp The connection to use.

Return value

Returns the date of the entry in the format of the Date library.

See Also

ExamineFTPDirectory() , NextFTPDirectoryEntry() , FTPDirectoryEntryType() , FTPDirectoryEntryName() , FTPDirectoryEntrySize() , FTPDirectoryEntryRaw() , FTPDirectoryEntryAttributes()

120.12 FTPDirectoryEntryName

Syntax

```
Filename\[extract_itex] = FTPDirectoryEntryName(#Ftp)
```

Description

Returns the name of the current entry in the FTP enumeration being listed with ExamineFTPDirectory() and NextFTPDirectoryEntry() functions.

Parameters

#Ftp The connection to use.

Return value

Returns the entry name.

See Also

ExamineFTPDirectory() , NextFTPDirectoryEntry() , FTPDirectoryEntryType() , FTPDirectoryEntryDate() , FTPDirectoryEntrySize() , FTPDirectoryEntryRaw() , FTPDirectoryEntryAttributes()

120.13 FTPDirectoryEntryType

Syntax

```
Result = FTPDirectoryEntryType(#Ftp)
```

Description

Returns the type of the current entry in the FTP enumeration being listed with `ExamineFTPDirectory()` and `NextFTPDirectoryEntry()` functions.

Parameters

#Ftp The connection to use.

Return value

Returns one of the following constants:

```
#PB_FTP_File      : This entry is a file.  
#PB_FTP_Directory: This entry is a directory.
```

See Also

`ExamineFTPDirectory()` , `NextFTPDirectoryEntry()` , `FTPDirectoryEntryName()` , `FTPDirectoryEntryDate()` , `FTPDirectoryEntrySize()` , `FTPDirectoryEntryRaw()` , `FTPDirectoryEntryAttributes()`

120.14 FTPDirectoryEntryRaw

Syntax

```
Entry\[extract_itex] = FTPDirectoryEntryRaw(#Ftp)
```

Description

Returns the raw line of the current entry in the FTP enumeration being listed with `ExamineFTPDirectory()` and `NextFTPDirectoryEntry()` functions, as it has been sent by the FTP server. It can be useful when the server isn't supported by `ExamineFTPDirectory()` , so the information about the directory listing can still be retrieved and parsed accordingly.

Parameters

#Ftp The connection to use.

Return value

Returns the entry line as sent by the FTP server.

Example

```
1  If OpenFTP(0, "ftp.free.fr", "anonymous", "")
2      If ExamineFTPDirectory(0)
3          While NextFTPDirectoryEntry(0)
4              Debug FTPDirectoryEntryRaw(0)
5          Wend
6          FinishFTPDirectory(0)
7      EndIf
8  Else
9      Debug "Can't connect to ftp.free.fr"
10 EndIf
```

See Also

ExamineFTPDirectory() , NextFTPDirectoryEntry() , FTPDirectoryEntryType() ,
FTPDirectoryEntryName() , FTPDirectoryEntryDate() , FTPDirectoryEntrySize() ,
FTPDirectoryEntryAttributes()

120.15 FTPDirectoryEntrySize

Syntax

```
Size = FTPDirectoryEntrySize(#Ftp)
```

Description

Returns the size of the current entry in the FTP enumeration being listed with
ExamineFTPDirectory() and NextFTPDirectoryEntry() functions.

Parameters

#Ftp The connection to use.

Return value

Returns the size in bytes.

See Also

ExamineFTPDirectory() , NextFTPDirectoryEntry() , FTPDirectoryEntryType() ,
FTPDirectoryEntryName() , FTPDirectoryEntryDate() , FTPDirectoryEntryRaw() ,
FTPDirectoryEntryAttributes()

120.16 FTPProgress

Syntax

```
Result.q = FTPProgress(#Ftp)
```

Description

Returns the progress of the current file transfer, started either with ReceiveFTPFile() or
SendFTPFile() .

Parameters

#Ftp The connection to use.

Return value

Returns the actual number of bytes which has been received/sent or one of the following values:

```
#PB_FTP_Started : The file transfer is in the initialization
phase.
#PB_FTP_Finished: The file transfer is finished correctly.
#PB_FTP_Error   : The file transfer is finished but an error
occurred.
```

See Also

SendFTPFile() , ReceiveFTPFile() , AbortFTPFile()

120.17 IsFtp

Syntax

```
Result = IsFtp(#Ftp)
```

Description

Tests if the given **#Ftp** number is a valid and correctly initialized ftp client.

Parameters

#Ftp The connection to use.

Return value

Returns nonzero if the given number is a valid and correctly initialized client connection.

Remarks

This function is bulletproof and can be used with any value. If the 'Result' is not zero then the object is valid and initialized, else it returns zero. This is the correct way to ensure an ftp handle is ready to use.

See Also

OpenFTP() , CloseFTP() , CheckFTPConnection()

120.18 NextFTPDirectoryEntry

Syntax

```
Result = NextFTPDirectoryEntry(#Ftp)
```

Description

Moves to the next entry in an enumeration started with ExamineFTPDirectory() .

Parameters

#Ftp The connection to use.

Return value

Returns nonzero if the next entry is available or zero if there are no more entries to examine.

Remarks

The entry name can be read with `FTPDirectoryEntryName()` . To know whether an entry is a file or directory, use `FTPDirectoryEntryType()` .

See Also

`ExamineFTPDirectory()` , `FTPDirectoryEntryType()` , `FTPDirectoryEntryName()` , `FTPDirectoryEntryDate()` , `FTPDirectoryEntrySize()` , `FTPDirectoryEntryRaw()` , `FTPDirectoryEntryAttributes()`

120.19 OpenFTP

Syntax

```
Result = OpenFTP(#Ftp, ServerName$, User$, Password$ [, Passive [, Port]])
```

Description

Tries to open a connection on the specified FTP or SFTP server.

Parameters

#Ftp The number to identify the new connection. **#PB_Any** can be used to auto-generate this number.

ServerName\$ The URL or address of the server to connect to. To connect to a SFTP server the prefix "sftp://" has to be added before the server address.

User\$ The user name to authenticate on the server.

Password\$ The password to authenticate on the server.

Passive (optional) Enables or disables passive mode for the connection (valid values are **#True** or **#False**). If this parameter is not specified, passive mode will be used.

Port (optional) The port to use for the connection. The default is port 21 for FTP and 22 for SFTP.

Return value

Returns nonzero if the connection was established correctly and zero if there was an error. If **#PB_Any** is used as the **#Ftp** parameter then the generated number for the connection is returned on success.

Example: For FTP

```
1  If OpenFTP(0, "ftp.free.fr", "anonymous", "")
2      Debug "Successfully connected"
3  Else
4      Debug "Can't connect to ftp.free.fr"
5  EndIf
```

Example: For SFTP

```
1  If OpenFTP(0, "sftp://test.rebex.net", "demo", "password") ; Use
    a public SFTP test server
2      Debug "Successfully connected"
3  Else
4      Debug "Can't connect to test.rebex.net"
5  EndIf
```

See Also

CloseFTP() , SetFTPDDirectory() , ReceiveFTPFile() , SendFTPFile() , ExamineFTPDDirectory()

120.20 ReceiveFTPFile

Syntax

```
Result = ReceiveFTPFile(#Ftp, RemoteFilename$, Filename$ [,
    Asynchronous])
```

Description

Receives a file from a FTP server.

Parameters

#Ftp The connection to use.

RemoteFilename\$ The name of the file to download. It has to be in the current ftp directory (see GetFTPDDirectory() and SetFTPDDirectory()).

Filename\$ The location to save the file on the local system. If the filename does not include a full path, it is interpreted relative to the current directory . If the file exists, it will be overwritten.

Asynchronous (optional) If set to a **#True**, the transfer will be made in the background. The default is to block the program until the file has been received. The progress of an asynchronous transfer can be received with the FTPProgress() command and it can be aborted using the AbortFTPFile() command.

Return value

Returns nonzero if the file was downloaded correctly, or the asynchronous transfer was initialized correctly. On failure, the return-value is zero.

Remarks

Only one file can be downloaded or uploaded (see SendFTPFile()) in the background at once.

See Also

SendFTPFile() , SetFTPDDirectory() , GetFTPDDirectory() , FTPProgress() , AbortFTPFile()

120.21 RenameFTPFile

Syntax

```
Result = RenameFTPFile(#Ftp, Filename$, NewFilename$)
```

Description

Renames a file on the FTP server.

Parameters

#Ftp The connection to use.

Filename\$ The name of the file on the server to be renamed. The renamed file has to be in the current directory (see GetFTPDDirectory() and SetFTPDDirectory()). It is not possible to specify a subpath in the Filename\$.

NewFilename\$ The new name for the file on the server. The renamed file has to be in the current directory (see GetFTPDDirectory() and SetFTPDDirectory()). It is not possible to specify a subpath in the NewFilename\$.

Return value

Returns nonzero if the file was renamed successfully and zero otherwise.

See Also

SendFTPFile() , SetFTPDDirectory() , GetFTPDDirectory() ,

120.22 SendFTPFile

Syntax

```
Result = SendFTPFile(#Ftp, Filename$, RemoteFilename$ [,
    Asynchronous])
```

Description

Sends a file to a FTP server.

Parameters

#Ftp The connection to use.

Filename\$ The name of the file to send. If the filename does not include a full path, it is interpreted relative to the current directory .

RemoteFilename\$ The remote filename. It has to be in the current ftp directory (see GetFTPDDirectory() and SetFTPDDirectory()).

Asynchronous (optional) If set to a **#True**, the transfer will be made in the background. The default is to block the program until the file has been fully sent. The progress of an asynchronous transfer can be received with the FTPProgress() command and it can be aborted using the AbortFTPFile() command.

Return value

Returns nonzero if the file was uploaded correctly, or the asynchronous transfer was initialized correctly. On failure, the return-value is zero.

Remarks

Only one file can be downloaded or uploaded (see `ReceiveFTPFile()`) in the background at once.

See Also

`ReceiveFTPFile()` , `SetFTPDDirectory()` , `GetFTPDDirectory()` , `FTPProgress()` , `AbortFTPFile()`

120.23 SetFTPDDirectory

Syntax

```
Result = SetFTPDDirectory(#Ftp, Directory$)
```

Description

Changes the current `#Ftp` directory, relative to the current directory.

Parameters

#Ftp The connection to use.

Directory\$ The directory to change to. This parameter must refer to a directory inside the current FTP directory. Nested paths are not allowed. Use the value `".."` to return to the parent directory of the current directory.

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

To get the current FTP directory, use the `GetFTPDDirectory()` function.

Example

```
1  If OpenFTP(0, "ftp.free.fr", "anonymous", "")
2
3      Debug "Successfully connected"
4
5      If SetFTPDDirectory(0, "pub")
6          If SetFTPDDirectory(0, "Linux")
7              Debug "Cool, changed to '/pub/Linux'"
8              Debug GetFTPDDirectory(0)
9          Else
10             Debug "Can't change to '/pub/Linux'"
11         EndIf
12     Else
13         Debug "Can't change to 'pub'"
14     EndIf
```

```
15  
16     Else  
17         Debug "Can't connect to ftp.free.fr"  
18     EndIf
```

See Also

GetFTPDirectory() , ExamineFTPDirectory() , SendFTPFile() , ReceiveFTPFile()

Chapter 121

Gadget

Overview

The Gadgets in PureBasic (in other languages also called "controls" or "widgets") are a generic name for all the interface components: button, combobox, listview, panels, ... This library is OS independent and uses the real OS Graphical User Interface (GUI) components.

Before using gadgets there will be normally opened a window first, furthermore there will be often used menus, toolbars and statusbars when creating graphical user interfaces.

The functions that create a new gadget return the new gadget number (called `#Gadget` in this library) if `#PB_Any` was used to create the gadget. If a static number was given to identify the gadget instead of `#PB_Any`, then the functions return the OS identifier for the created Gadget. These OS identifiers (or also called handles) can be used for other PureBasic functions, as well for WinAPI functions like `SendMessage_()` etc. Look at the chapter Handles and Numbers in the reference manual for more information. If you want to create GUI dialogs with automatic layout support take a look at the dialog library.

121.1 AddGadgetColumn

Syntax

```
AddGadgetColumn(#Gadget, Position, Title$, Width)
```

Description

Adds a new column to the specified gadget.

Parameters

#Gadget The gadget to use.

Position The column index where the new item should be inserted. Column indexes start from 0, which is the leftmost column, and increase by 1 for each column to the right. When you add a column, all the old columns which are on the right of the new column will have a position which is one more than they previously had.

Title\$ The text for the column header.

Width The initial width of the new column.

Return value

None.

Remarks

This command can be used with the following types of gadgets:

- ListIconGadget()
- ExplorerListGadget()

For ExplorerListGadget() you can use this function to completely customize the information that the gadget displays, by first removing the standard columns with RemoveGadgetColumn() and adding new ones of your choice. Note that the 'Name' column doesn't need to be the first one in the gadget.

Note: To update the gadget contents after you have added new columns, use SetGadgetText() . To fill in a custom column, use SetGadgetItemText() for each item, after you have received a #PB_EventType_Change event for the gadget.

When adding a column, use one of the following constants in the Title\$ field to create a column whose contents will be automatically updated by the gadget, or use any title string you wish to create an empty column which you can then fill with SetGadgetItemText() .

#PB_Explorer_Name	: displays the name of the file/directory
#PB_Explorer_Size	: displays the filesize in Kb
#PB_Explorer_Type	: displays a string describing the filetype
#PB_Explorer_Attributes	: displays the attributes of the file/directory
#PB_Explorer_Created	: displays the time the file/directory was created
#PB_Explorer_Modified	: displays the time the file/directory was last modified
#PB_Explorer_Accessed	: displays the time the file/directory was last accessed

Example

```
1  If OpenWindow(0, 0, 0, 400, 150, "ListIcon - Add Columns",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      ListIconGadget(0, 10, 10, 380, 100, "Standard Column", 150,
    #PB_ListIcon_GridLines)
3      ButtonGadget(1, 10, 120, 150, 20, "Add new column")
4      index = 1 ; "Standard column" has already index 0
5      Repeat
6          Event = WaitWindowEvent()
7          If Event = #PB_Event_Gadget
8              If EventGadget() = 1
9                  AddGadgetColumn(0, index, "Column "+Str(index), 80)
10                 index + 1
11             EndIf
12         EndIf
13     Until Event = #PB_Event_CloseWindow
14 EndIf
```

See Also

RemoveGadgetColumn() , ListIconGadget() , ExplorerListGadget()

121.2 AddGadgetItem

Syntax

```
Result = AddGadgetItem(#Gadget, Position, Text$ [, ImageID [,
Flags]])
```

Description

Add a new item to the specified gadget.

Parameters

#Gadget The gadget to use.

Position The item index where the new item should be inserted. To add the item at the start, use an index of 0. To add this item to the end of the current item list, use a value of -1.

Remember that when you add an item that all current items which come after the new one will have their positions increased by 1.

For the MDIGadget() this parameter specifies the #Window number for the new MDI childwindow. #PB_Any can be used, in which case the return-value will be the new number assigned by PB.

Text\$ The text for the new item.

When adding an item to a ListIconGadget() , this parameter can contain the text for multiple columns separated by a Chr(10) character.

ImageID (optional) An optional image to use for the item in gadgets that support it. The used image should be in the standard 16x16 size. Use the ImageID() command to get the ID for this parameter.

Flags (optional) This parameter has a meaning only for the following gadget types:

TreeGadget()

This parameter specifies the new sublevel for the item. If the sublevel is greater than that of the previous item, the new one will become this item's child. If it is lower, it will be added after the parent of the previous item.

MDIGadget()

This parameter can be used to specify the flags for the new window (see OpenWindow()). The #PB_Window_Borderless, #PB_Window_Screencentered and #PB_Window_WindowCentered flags are not supported for MDI windows.

Return value

The return-value is only valid with the MDIGadget() . If #PB_Any was used as the 'Position' parameter when adding an item to the MDIGadget() , the return-value is the number that identifies the new MDI window.

Remarks

The following gadgets are supported:

- ComboBoxGadget() : supports the ImageID if the #PB_ComboBox_Image is set.
- EditorGadget()
- ListIconGadget() : supports the ImageID.
- ListViewGadget()
- MDIGadget() : ImageID can contain an icon for the childwindow titlebar. Flags can specify the new window flags.
- PanelGadget() : supports the ImageID.
- TreeGadget() : supports the ImageID. 'Flags' is required and specifies the new sublevel.

See Also

`RemoveGadgetItem()` , `ClearGadgetItems()` , `CountGadgetItems()` , `ComboBoxGadget()` ,
`ListIconGadget()` , `ListViewGadget()` , `MDIGadget()` , `PanelGadget()` , `TreeGadget()`

121.3 ButtonImageGadget

Syntax

```
Result = ButtonImageGadget(#Gadget, x, y, Width, Height, ImageID [,  
    Flags])
```

Description

Create a button gadget with an image in the current GadgetList.

Parameters

#Gadget A number to identify the new gadget. `#PB_Any` can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

ImageID The image for the gadget. Use the `ImageID()` function to get this ID from an image. This parameter can be zero to create a button without an image. The `SetGadgetAttribute()` function can be used to change the image later.

Flags (optional) This parameter can be `#PB_Button_Toggle` to create a toggle-button (one which has an on/off state). A push-button is created by default.

Return value

Returns nonzero on success and zero on failure. If `#PB_Any` was used as the `#Gadget` parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using `GadgetToolTip()` .

The following functions can be used to control the gadget:

- `GetGadgetState()` can be used to get the toggle state of the gadget.
- `SetGadgetState()` can be used to set the toggle state of the gadget.
- `GetGadgetAttribute()` with the following values:

```
#PB_Button_Image           : Get the displayed image ID, e.g.  
    ImageID(#MyImage) .  
#PB_Button_PressedImage : Get the displayed image ID when the  
    button is pressed, e.g. ImageID(#MyImagePressed) .
```

- `SetGadgetAttribute()` with the following values:

```
#PB_Button_Image           : Set the displayed image.  
#PB_Button_PressedImage : Set the image displayed when the button  
    is pressed .
```

Example

```
1  If OpenWindow(0, 0, 0, 200, 60, "ButtonImageGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2  If LoadImage(0, #PB_Compiler_Home +
    "examples/sources/Data/PureBasic.bmp") ; change 2nd parameter
    to the path/filename of your image
3  ButtonImageGadget(0, 10, 10, 180, 40, ImageID(0))
4  EndIf
5  Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
6  EndIf
```



See Also

GetGadgetState() , SetGadgetState() , GetGadgetAttribute() , SetGadgetAttribute() ,
ButtonGadget() , ImageID() , EventGadget()

121.4 ButtonGadget

Syntax

```
Result = ButtonGadget(#Gadget, x, y, Width, Height, Text$ [, Flags])
```

Description

Create a button gadget in the current GadgetList.

Parameters

#Gadget A number to identify the new gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget. **Note:** on OS X, using an height of 25 will enable the fixed height button type, which is commonly used in OS X applications. It will also make the #PB_Button_Default flags available.

Text\$ The text to display on the button.

Flags (optional) A combination (using the bitwise OR operator '||') of the following constants:

```
#PB_Button_Right      : Aligns the button text at the right.
    (not supported on Mac OSX)
#PB_Button_Left       : Aligns the button text at the left.
    (not supported on Mac OSX)
#PB_Button_Default    : Makes the button look as if it is the
    default button in the window (on OS X, the height of the
    button needs to be 25).
#PB_Button_MultiLine  : If the text is too long, it will be
    displayed on several lines. (not supported on OSX)
#PB_Button_Toggle     : Creates a toggle button: one click
    pushes it, another will release it.
```

Return value

Returns nonzero on success and zero on failure. If `#PB_Any` was used as the `#Gadget` parameter then the return-value is the auto-generated gadget number on success.

Remarks

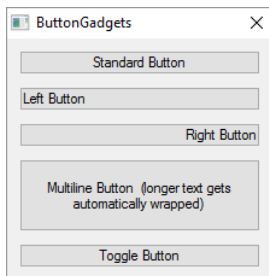
A 'mini help' can be added to this gadget using `GadgetToolTip()` .

The following functions can be used to control the gadget:

- `SetGadgetText()` : Changes the text of the `ButtonGadget`.
- `GetGadgetText()` : Returns the text of the `ButtonGadget`.
- `SetGadgetState()` : Used with `#PB_Button_Toggle` buttons to set the actual state (1 = toggled, 0 = normal).
- `GetGadgetState()` : Used with `#PB_Button_Toggle` buttons to get the actual state of the button (1 = toggled, 0 = normal).

Example

```
1 ; Shows possible flags of ButtonGadget in action...
2 If OpenWindow(0, 0, 0, 222, 200, "ButtonGadgets",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3   ButtonGadget(0, 10, 10, 200, 20, "Standard Button")
4   ButtonGadget(1, 10, 40, 200, 20, "Left Button", #PB_Button_Left)
5   ButtonGadget(2, 10, 70, 200, 20, "Right Button",
   #PB_Button_Right)
6   ButtonGadget(3, 10, 100, 200, 60, "Multiline Button (longer
   text gets automatically wrapped)", #PB_Button_MultiLine)
7   ButtonGadget(4, 10, 170, 200, 20, "Toggle Button",
   #PB_Button_Toggle)
8   Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
9 EndIf
```



See Also

`SetGadgetText()` , `GetGadgetText()` , `SetGadgetState()` , `GetGadgetState()` ,
`ButtonImageGadget()`

121.5 CalendarGadget

Syntax

```
Result = CalendarGadget(#Gadget, x, y, Width, Height [, Date [,
    Flags]])
```

Description

Create a calendar gadget in the current GadgetList. This gadget displays a month calendar and lets the user select a date.

Parameters

#Gadget A number to identify the new gadget. `#PB_Any` can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Date (optional) The initial date to set. The default is the current date.

Flags (optional) This parameter can be set to `#PB_Calendar_Borderless` to create a gadget without a border (not supported on Linux).

Return value

Returns nonzero on success and zero on failure. If `#PB_Any` was used as the `#Gadget` parameter then the return-value is the auto-generated gadget number on success.

Remarks

The dates used by this gadget and the functions for it use the same date format as the date library .

A 'mini help' can be added to this gadget using `GadgetToolTip()` .

The following functions can be used for this gadget:

- `SetGadgetState()` : Set the currently displayed date.
- `GetGadgetState()` : Get the currently displayed date.
- `SetGadgetAttribute()` : With the following attributes:

```
#PB_Calendar_Minimum: Set the minimum selectable date
#PB_Calendar_Maximum: Set the maximum selectable date in this
gadget.
                        (Note: Limitation of the selectable date
                        isn't supported on Linux.)
```

- `GetGadgetAttribute()` : With the following attributes:

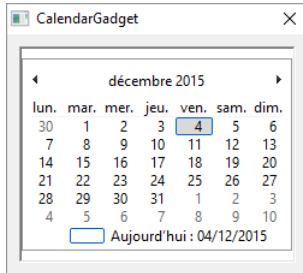
```
#PB_Calendar_Minimum: Get the minimum selectable date
#PB_Calendar_Maximum: Get the maximum selectable date of this
gadget.
                        (Note: Limitation of the selectable date
                        isn't supported on Linux.)
```

This gadget supports the `SetGadgetColor()` and `GetGadgetColor()` functions with the following values as 'ColorType':

```
#PB_Gadget_BackColor      : backgroundcolor
#PB_Gadget_FrontColor     : textcolor for displayed days (not
supported on Windows Vista+)
#PB_Gadget_TitleBackColor : backgroundcolor of the month title
(not supported on Windows Vista+)
#PB_Gadget_TitleFrontColor: textcolor of the month title (not
supported on Windows Vista+)
#PB_Gadget_GrayTextColor  : textcolor for days not of the current
month (not supported on Windows Vista+)
```

Example

```
1  If OpenWindow(0, 0, 0, 250, 200, "CalendarGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      CalendarGadget(0, 10, 10, 230, 180)
3      Repeat
4      Until WaitWindowEvent() = #PB_Event_CloseWindow
5  EndIf
```



See Also

SetGadgetState(), GetGadgetState(), SetGadgetItemState(), GetGadgetItemState(), SetGadgetAttribute(), GetGadgetAttribute(), SetGadgetColor(), GetGadgetColor(), DateGadget(), Date(), FormatDate()

121.6 CanvasGadget

Syntax

```
Result = CanvasGadget(#Gadget, x, y, Width, Height [, Flags])
```

Description

Create a canvas gadget in the current GadgetList. This gadget provides a drawing surface without alpha channel and events for mouse and keyboard interaction to easily create custom views.

Parameters

#Gadget A number to identify the new gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget (in pixels). The maximum width and height is 16000 pixels.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following constants:

```
#PB_Canvas_Border      : Draws a border around the gadget.
#PB_Canvas_ClipMouse: Automatically clips the mouse to the
    gadget area while a mouse button is down (Not supported on
    OS X and Linux Gtk3).
#PB_Canvas_Keyboard  : Allows the gadget to receive the
    keyboard focus and keyboard events.
#PB_Canvas_DrawFocus: Draws a focus rectangle on the gadget
    if it has keyboard focus.
#PB_Canvas_Container: Enables container support so gadget can
    be added to it. CloseGadgetList()
needs to be called
```

to go back to previous gadget list like any other container gadget.

On Windows, the transparency of the gadgets does not work so the text of the following gadgets will be

displayed on an opaque background:
CheckBoxGadget, FrameGadget, HyperlinkGadget, OptionGadget,
TextGadget and TrackBarGadget.

The #PB_Canvas_Keyboard flag is required to receive any keyboard events in the gadget. If you include this flag, you should check for the #PB_EventType_Focus and #PB_EventType_LostFocus events and draw a visual indication on the gadget when it has the focus so it is clear to the user which gadget currently has the focus. Alternatively, the #PB_Canvas_DrawFocus flag can be included to let the gadget draw a standard focus rectangle whenever it has the focus.

Return value

Returns nonzero on success and zero on failure. If #PB_Any was used as the #Gadget parameter then the return-value is the auto-generated gadget number on success.

Remarks

The CanvasGadget() does not have an alpha channel so the #PB_2DDrawing_AlphaChannel modes of the DrawingMode() function will have no effect and the #PB_2DDrawing_AllChannels mode will be equivalent to #PB_2DDrawing_Default.

The created gadget starts out with just a white background. Use the CanvasOutput() or CanvasVectorOutput() command to draw to the gadget. The drawn content stays persistent until it is erased by a further drawing operation. There is no need to redraw the content every time a #PB_Event_Repaint event is received.

The following events are reported by the gadget. The EventType() function reports the type of the current gadget event:

#PB_EventType_MouseEnter gadget	: The mouse cursor entered the gadget
#PB_EventType_MouseLeave	: The mouse cursor left the gadget
#PB_EventType_MouseMove	: The mouse cursor moved
#PB_EventType_MouseWheel	: The mouse wheel was moved
#PB_EventType_LeftButtonDown	: The left mouse button was pressed
#PB_EventType_LeftButtonUp	: The left mouse button was released
#PB_EventType_LeftClick	: A click with the left mouse button
#PB_EventType_LeftDoubleClick	: A double-click with the left mouse button
#PB_EventType_RightButtonDown	: The right mouse button was pressed
#PB_EventType_RightButtonUp	: The right mouse button was released
#PB_EventType_RightClick	: A click with the right mouse button
#PB_EventType_RightDoubleClick	: A double-click with the right mouse button
#PB_EventType_MiddleButtonDown	: The middle mouse button was pressed
#PB_EventType_MiddleButtonUp	: The middle mouse button was released
#PB_EventType_Focus	: The gadget gained keyboard focus
#PB_EventType_LostFocus	: The gadget lost keyboard focus
#PB_EventType_KeyDown	: A key was pressed
#PB_EventType_KeyUp	: A key was released
#PB_EventType_Input	: Text input was generated

`#PB_EventType_Resize` : The gadget has been resized

Note that the events `#PB_EventType_KeyDown`, `#PB_EventType_KeyUp` and `#PB_EventType_Input` are only reported when the gadget has the keyboard focus. This means that the `#PB_Canvas_Keyboard` flag has to be set on gadget creation to allow keyboard events. On Windows, the `#PB_EventType_MouseWheel` event is also only reported if the gadget has keyboard focus. On the other OS, this event is reported to the gadget under the cursor, regardless of keyboard focus.

Further information about the current event can be received with the `GetGadgetAttribute()` function. This information is only available if the current event received by `WaitWindowEvent()` or `WindowEvent()` is an event for this gadget. The following attributes can be used:

`#PB_Canvas_MouseX`, `#PB_Canvas_MouseY`

Returns the given mouse coordinate relative to the drawing area of the gadget. This returns the mouse location at the time that the event was generated, so the result can differ from the coordinates reported by `WindowMouseX()` and `WindowMouseY()` which return the current mouse location regardless of the state of the processed events. The coordinates returned using these attributes should be used for this gadget to ensure that the mouse coordinates are in sync with the current event.

`#PB_Canvas_Buttons`

Returns the state of the mouse buttons for the event. The result is a combination (using bitwise or) of the following values:

`#PB_Canvas_LeftButton` : The left button is currently down.
`#PB_Canvas_RightButton` : The right button is currently down.
`#PB_Canvas_MiddleButton`: The middle button is currently down.

`#PB_Canvas_Modifiers`

Returns the state of the keyboard modifiers for the event. The result is a combination (using bitwise or) of the following values:

`#PB_Canvas_Shift` : The 'shift' key is currently pressed.
`#PB_Canvas_Alt` : The 'alt' key is currently pressed.
`#PB_Canvas_Control`: The 'control' key is currently pressed.
`#PB_Canvas_Command`: The 'command' (or "apple") key is currently pressed. (Mac OSX only)

`#PB_Canvas_WheelDelta`

Returns the movement of the mouse wheel in the current event in multiples of 1 or -1. A positive value indicates that the wheel was moved up (away from the user) and a negative value indicates that the wheel was moved down (towards the user). This attribute is 0 if the current event is not a `#PB_EventType_MouseWheel` event.

`#PB_Canvas_Key`

Returns the key that was pressed or released in a `#PB_EventType_KeyDown` or `#PB_EventType_KeyUp` event. The returned value is one of the `#PB_Shortcut_...` values used by the `AddKeyboardShortcut()` function. This attribute returns raw key presses. To get text input for the gadget, it is better to watch for `#PB_EventType_Input` events and use the `#PB_Canvas_Input` attribute because it contains the text input from multiple key presses such as shift or dead keys combined.

#PB_Canvas_Input

Returns the input character that was generated by one or more key presses. This attribute is only present after a #PB_EventType_Input event. The returned character value can be converted into a string using the Chr() function.

In addition to this event information, GetGadgetAttribute() can also be used to read the following attributes:

#PB_Canvas_Image

Returns an ImageID value that represents an image with the current content of the CanvasGadget. This value can be used to draw the content of the gadget to another drawing output using the DrawImage() function.
Note: The returned value is only valid until changes are made to the gadget by resizing it or drawing to it, so it should only be used directly in a command like DrawImage() and not stored for future use.

#PB_Canvas_Clip

Returns non-zero if the mouse is currently clipped to the gadget area or zero if not.

#PB_Canvas_Cursor

Returns the cursor that is currently used in the gadget. See below for a list of possible values. If the gadget is using a custom cursor handle, the return-value is -1.

#PB_Canvas_CustomCursor

Returns the custom cursor handle that was set using SetGadgetAttribute() . If the gadget uses a standard cursor, the return-value is 0.

The SetGadgetAttribute() function can be used to modify the following gadget attributes

#PB_Canvas_Image

Applies the given ImageID to the CanvasGadget. The gadget makes a copy of the input image so it can be freed or reused after this call. Setting this attribute is the same as using StartDrawing() , CanvasOutput() and DrawImage() to draw the image onto the CanvasGadget.

#PB_Canvas_Clip

If the value is set to a non-zero value, the mouse cursor will be confined to the area of the canvas gadget. Setting the value to zero removes the clipping.
Note: Mouse clipping should only be done as a direct result of user interaction with the gadget such as a mouse click and care must be taken to properly remove the clipping again or the user's mouse will be trapped inside the gadget. The #PB_Canvas_ClipMouse gadget flag can be used to automatically clip/unclip the mouse when the user presses or releases a mouse button in the gadget.

#PB_Canvas_Cursor

Changes the cursor that is displayed when the mouse is over the gadget. The following values are possible:

#PB_Cursor_Default	: default arrow cursor
#PB_Cursor_Cross	: crosshair cursor
#PB_Cursor_IBeam	: I-cursor used for text selection
#PB_Cursor_Hand	: hand cursor
#PB_Cursor_Busy	: hourglass or watch cursor
#PB_Cursor_Denied	: slashed circle or X cursor


```

#PB_Cursor_Arrows      : arrows in all direction (not
    available on Mac OSX)
#PB_Cursor_LeftRight   : left and right arrows
#PB_Cursor_UpDown      : up and down arrows
#PB_Cursor_LeftUpRightDown: diagonal arrows (Windows only)
#PB_Cursor_LeftDownRightUp: diagonal arrows (Windows only)
#PB_Cursor_Invisible    : hides the cursor

```

#PB_Canvas_CustomCursor

Changes the cursor that is displayed when the mouse is over the gadget to a custom cursor handle created using the corresponding OS API. This attribute expects the following kind of input:

Windows: a HCURSOR handle
Linux: a GtkCursor pointer
Mac OSX: a pointer to a Cursor structure

A 'mini help' can be added to this gadget using GadgetToolTip() .

Example

```

1  If OpenWindow(0, 0, 0, 220, 220, "CanvasGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      CanvasGadget(0, 10, 10, 200, 200)
3
4      Repeat
5          Event = WaitWindowEvent()
6
7          If Event = #PB_Event_Gadget And EventGadget() = 0
8              If EventType() = #PB_EventType_LeftButtonDown Or
                (EventType() = #PB_EventType_MouseMove And GetGadgetAttribute(0,
                #PB_Canvas_Buttons) & #PB_Canvas_LeftButton)
9                  If StartDrawing(CanvasOutput(0))
10                     x = GetGadgetAttribute(0, #PB_Canvas_MouseX)
11                     y = GetGadgetAttribute(0, #PB_Canvas_MouseY)
12                     Circle(x, y, 10, RGB(Random(255), Random(255),
                Random(255)))
13                     StopDrawing()
14                     EndIf
15                 EndIf
16             EndIf
17
18         Until Event = #PB_Event_CloseWindow
19     EndIf

```

Example: Canvas container

```

1  If OpenWindow(0, 0, 0, 220, 220, "Canvas container example",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2
3      CanvasGadget(0, 10, 10, 200, 200, #PB_Canvas_Container)
4      ButtonGadget(1, 10, 10, 80, 30, "Clean up")
5      CloseGadgetList()
6
7      Repeat
8          Event = WaitWindowEvent()

```

```

9
10     If Event = #PB_Event_Gadget
11         Select EventGadget()
12             Case 0
13                 If EventType() = #PB_EventType_LeftButtonDown Or
14 (EventType() = #PB_EventType_MouseMove And GetGadgetAttribute(0,
15 #PB_Canvas_Buttons) & #PB_Canvas_LeftButton)
16                     If StartDrawing(CanvasOutput(0))
17                         x = GetGadgetAttribute(0, #PB_Canvas_MouseX)
18                         y = GetGadgetAttribute(0, #PB_Canvas_MouseY)
19                         Circle(x, y, 10, RGB(Random(255), Random(255),
20 Random(255)))
21                     StopDrawing()
22                 EndIf
23             EndIf
24         EndIf
25
26         Case 1
27             If StartDrawing(CanvasOutput(0))
28                 Box(0, 0, 200, 200, #White)
29                 StopDrawing()
30             EndIf
31         EndSelect
32     EndSelect
33 EndIf
34
35 Until Event = #PB_Event_CloseWindow
36 EndIf

```



See Also

CanvasOutput() , GetGadgetAttribute() , SetGadgetAttribute() , EventType() , StartDrawing()

121.7 CanvasOutput

Syntax

```
OutputID = CanvasOutput(#Gadget)
```

Description

Returns the OutputID of a CanvasGadget to perform 2D rendering operation on it.

Parameters

#Gadget The gadget to draw on. This must be a CanvasGadget() .

Return value

Returns the output ID or zero if drawing is not possible. This value should be passed directly to the StartDrawing() function to start the drawing operation. The return-value is valid only for one drawing operation and cannot be reused.

Example

```
1  ...
2  StartDrawing(CanvasOutput(#Gadget))
3  ; do some drawing stuff here...
4  StopDrawing()
```

Remarks

The CanvasGadget() does not have an alpha channel so the #PB_2DDrawing_AlphaChannel modes of the DrawingMode() function will have no effect and the #PB_2DDrawing_AllChannels mode will be equivalent to #PB_2DDrawing_Default.

Drawing on a CanvasGadget() is double-buffered. This means that the drawing operations only become visible at the StopDrawing() command to avoid visible flicker during the drawing.

See Also

StartDrawing() , CanvasGadget() , CanvasVectorOutput()

121.8 CanvasVectorOutput

Syntax

```
VectorOutputID = CanvasVectorOutput(#Gadget [, Unit])
```

Description

Returns the OutputID of a CanvasGadget to perform vector drawing operations on it.

Parameters

#Gadget The gadget to draw on. This must be a CanvasGadget() .

Unit (optional) Specifies the unit used for measuring distances on the drawing output. The default unit for canvas gadgets is #PB_Unit_Pixel.

```
#PB_Unit_Pixel      : Values are measured in pixels (or dots
                     in case of a printer)
#PB_Unit_Point      : Values are measured in points (1/72 inch)
#PB_Unit_Inch       : Values are measured in inches
#PB_Unit_Millimeter : Values are measured in millimeters
```

Return value

Returns the output ID or zero if drawing is not possible. This value should be passed directly to the StartVectorDrawing() function to start the drawing operation. The return-value is valid only for one drawing operation and cannot be reused.

Example

```
1 ...  
2 StartVectorDrawing(CanvasVectorOutput(#Gadget))  
3 ; do some drawing stuff here...  
4 StopVectorDrawing()
```

Remarks

Drawing on a CanvasGadget() is double-buffered. This means that the drawing operations only become visible at the StopVectorDrawing() command to avoid visible flicker during the drawing.

See Also

StartVectorDrawing() , CanvasGadget() , CanvasOutput()

121.9 OpenGLGadget

Syntax

```
Result = OpenGLGadget(#Gadget, x, y, Width, Height [, Flags])
```

Description

Creates an OpenGL gadget in the current GadgetList. This gadget provides an OpenGL drawing context and events for mouse and keyboard interaction to easily create 3D opengl view. Most of the OpenGL commands are directly available in PureBasic using the underscore API notation (command ending with an underscore, like 'glBegin_()').

Parameters

#Gadget A number to identify the new gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget (in pixels). The maximum width and height is 16000 pixels.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following constants:

#PB_OpenGL_Keyboard	: allows the gadget to receive the keyboard focus and keyboard events.
#PB_OpenGL_NoFlipSynchronization	: disables the vsync synchronization.
#PB_OpenGL_FlipSynchronization	: enables the vsync synchronization (default).
#PB_OpenGL_NoDepthBuffer	: disables the depth buffer.
#PB_OpenGL_16BitDepthBuffer	: creates a 16-bit depth buffer (default).
#PB_OpenGL_24BitDepthBuffer	: creates a 24-bit depth buffer.
#PB_OpenGL_NoStencilBuffer	: disables the stencil buffer (default).
#PB_OpenGL_8BitStencilBuffer	: creates a 8-bit stencil buffer.
#PB_OpenGL_NoAccumulationBuffer	: disables the accumulation buffer (default).

```
#PB_OpenGL_32BitAccumulationBuffer: creates a 32-bit
    accumulation buffer.
#PB_OpenGL_64BitAccumulationBuffer: creates a 64-bit
    accumulation buffer.
```

The `#PB_OpenGL_Keyboard` flag is required to receive any keyboard events in the gadget. If you include this flag, you should check for the `#PB_EventType_Focus` and `#PB_EventType_LostFocus` events and draw a visual indication on the gadget when it has the focus so it is clear to the user which gadget currently has the focus.

Return value

Returns nonzero on success and zero on failure. If `#PB_Any` was used as the `#Gadget` parameter then the return-value is the auto-generated gadget number on success.

Remarks

The created gadget starts out with just a black background. The current OpenGL context is set to this new created gadget. To change the current OpenGL context, use the `#PB_OpenGL_SetContext` attribute. Once drawn, a frame can be displayed using the `#PB_OpenGL_FlipBuffers` attribute.

As the OpenGL gadget is meant to be a realtime rendering gadget, it doesn't store its current rendering in a separate buffer: it needs to be rendered again when the window is resized, minimized, maximized or if the gadget itself is resized or moved.

The following events are reported by the gadget. The `EventType()` function reports the type of the current gadget event:

```
#PB_EventType_MouseEnter      : The mouse cursor entered the
    gadget
#PB_EventType_MouseLeave      : The mouse cursor left the gadget
#PB_EventType_MouseMove      : The mouse cursor moved
#PB_EventType_MouseWheel     : The mouse wheel was moved
#PB_EventType_LeftButtonDown : The left mouse button was pressed
#PB_EventType_LeftButtonUp   : The left mouse button was released
#PB_EventType_LeftClick      : A click with the left mouse button
#PB_EventType_LeftDoubleClick: A double-click with the left
    mouse button
#PB_EventType_RightButtonDown: The right mouse button was pressed
#PB_EventType_RightButtonUp   : The right mouse button was
    released
#PB_EventType_RightClick     : A click with the right mouse
    button
#PB_EventType_RightDoubleClick: A double-click with the right
    mouse button
#PB_EventType_MiddleButtonDown: The middle mouse button was
    pressed
#PB_EventType_MiddleButtonUp  : The middle mouse button was
    released
#PB_EventType_Focus          : The gadget gained keyboard focus
#PB_EventType_LostFocus      : The gadget lost keyboard focus
#PB_EventType_KeyDown        : A key was pressed
#PB_EventType_KeyUp          : A key was released
#PB_EventType_Input          : Text input was generated
```

Note that the events `#PB_EventType_KeyDown`, `#PB_EventType_KeyUp` and `#PB_EventType_Input` are only reported when the gadget has the keyboard focus. This means that the `#PB_OpenGL_Keyboard` flag has to be set on gadget creation to allow keyboard events. On Windows, the `#PB_EventType_MouseWheel` event is also only reported if the gadget has

keyboard focus. On the other OS, this event is reported to the gadget under the cursor, regardless of keyboard focus.

Further information about the current event can be received with the `GetGadgetAttribute()` function. This information is only available if the current event received by `WaitWindowEvent()` or `WindowEvent()` is an event for this gadget. The following attributes can be used:

`#PB_OpenGL_MouseX`, `#PB_OpenGL_MouseY`

Returns the given mouse coordinate relative to the drawing area of the gadget. This returns the mouse location at the time that the event was generated, so the result can differ from the coordinates reported by `WindowMouseX()` and `WindowMouseY()` which return the current mouse location regardless of the state of the processed events. The coordinates returned using these attributes should be used for this gadget to ensure that the mouse coordinates are in sync with the current event.

`#PB_OpenGL_Buttons`

Returns the state of the mouse buttons for the event. The result is a combination (using bitwise or) of the following values:

`#PB_OpenGL_LeftButton` : The `left` button is currently down.
`#PB_OpenGL_RightButton` : The `right` button is currently down.
`#PB_OpenGL_MiddleButton`: The `middle` button is currently down.

`#PB_OpenGL_Modifiers`

Returns the state of the keyboard modifiers for the event. The result is a combination (using bitwise or) of the following values:

`#PB_OpenGL_Shift` : The 'shift' key is currently pressed.
`#PB_OpenGL_Alt` : The 'alt' key is currently pressed.
`#PB_OpenGL_Control`: The 'control' key is currently pressed.
`#PB_OpenGL_Command`: The 'command' (or "apple") key is currently pressed. (Mac OSX only)

`#PB_OpenGL_WheelDelta`

Returns the movement of the mouse wheel in the current event in multiples of 1 or -1. A positive value indicates that the wheel was moved up (away from the user) and a negative value indicates that the wheel was moved down (towards the user). This attribute is 0 if the current event is not a `#PB_EventType_MouseWheel` event.

`#PB_OpenGL_Key`

Returns the key that was pressed or released in a `#PB_EventType_KeyDown` or `#PB_EventType_KeyUp` event. The returned value is one of the `#PB_Shortcut_...` values used by the `AddKeyboardShortcut()` function. This attribute returns raw key presses. To get text input for the gadget, it is better to watch for `#PB_EventType_Input` events and use the `#PB_OpenGL_Input` attribute because it contains the text input from multiple key presses such as shift or dead keys combined.

`#PB_OpenGL_Input`

Returns the input character that was generated by one or more key presses. This attribute is only present after a `#PB_EventType_Input` event. The returned character value can be converted into a string using the `Chr()` function.

In addition to this event information, `GetGadgetAttribute()` can also be used to read the following attributes:

`#PB_OpenGL_Cursor`

Returns the cursor that is currently used in the gadget. See below for a list of possible values. If the gadget is using a custom cursor handle, the return-value is -1.

`#PB_OpenGL_CustomCursor`

Returns the custom cursor handle that was set using `SetGadgetAttribute()`. If the gadget uses a standard cursor, the return-value is 0.

The `SetGadgetAttribute()` function can be used to modify the following gadget attributes

`#PB_OpenGL_SetContext`

Changes the current OpenGL context to this gadget context.

`#True` : use the OpenGL context from this gadget.
`#False` : remove the current OpenGL context. No more context are available.

`#PB_OpenGL_FlipBuffers`

Flip the back and front buffers. All drawing are done in the back buffer. To be visible, the buffers needs to be flipped, so the back becomes the front one (the one which is displayed).

`#True` : flips the buffers
`#False` : don't flip the buffers (no use for now).

`#PB_OpenGL_Cursor`

Changes the cursor that is displayed when the mouse is over the gadget. The following values are possible:

<code>#PB_Cursor_Default</code>	: default arrow cursor
<code>#PB_Cursor_Cross</code>	: crosshair cursor
<code>#PB_Cursor_IBeam</code>	: I-cursor used for text selection
<code>#PB_Cursor_Hand</code>	: hand cursor
<code>#PB_Cursor_Busy</code>	: hourglass or watch cursor
<code>#PB_Cursor_Denied</code>	: slashed circle or X cursor
<code>#PB_Cursor_Arrows</code>	: arrows in all direction (not available on Mac OSX)
<code>#PB_Cursor_LeftRight</code>	: left and right arrows
<code>#PB_Cursor_UpDown</code>	: up and down arrows
<code>#PB_Cursor_LeftUpRightDown</code>	: diagonal arrows (Windows only)
<code>#PB_Cursor_LeftDownRightUp</code>	: diagonal arrows (Windows only)
<code>#PB_Cursor_Invisible</code>	: hides the cursor

`#PB_OpenGL_CustomCursor`

Changes the cursor that is displayed when the mouse is over the gadget to a custom cursor handle created using the corresponding OS API. This attribute expects the following kind of input:

Windows: a `HCURSOR` handle
Linux: a `GtkCursor` pointer
Mac OSX: a pointer to a `Cursor` structure

A 'mini help' can be added to this gadget using `GadgetToolTip()`.

See Also

GetGadgetAttribute() , SetGadgetAttribute() , EventType()

121.10 CheckBoxGadget

Syntax

```
Result = CheckBoxGadget(#Gadget, x, y, Width, Height, Text$ [,
    Flags])
```

Description

Create a checkbox gadget in the current GadgetList.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Text\$ The text to display next to the checkbox.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following constants:

```
#PB_CheckBox_Right      : Aligns the text to right.
#PB_CheckBox_Center     : Centers the text.
#PB_CheckBox_ThreeState: Create a checkbox that can have a
    third "in between" state.
```

The **#PB_CheckBox_ThreeState** flag can be used for a checkbox that represents the state of multiple items. The "in between" state can then be used to indicate that the setting is not the same for all items. By clicking on the checkbox, the user can bring it back to either the "on" or "off" state to apply this to all the items. Therefore the "in between" state can only be set by the program via SetGadgetState() and not by the user by clicking on the checkbox.

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using GadgetToolTip() .

- GetGadgetState() can be used to get the current gadget state.

- SetGadgetState() can be used to change the gadget state.

Example

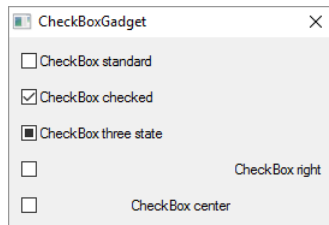
```
1  If OpenWindow(0, 0, 0, 270, 160, "CheckBoxGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      CheckBoxGadget(0, 10, 10, 250, 20, "CheckBox standard")
3      CheckBoxGadget(1, 10, 40, 250, 20, "CheckBox checked"):
    SetGadgetState(1, #PB_Checkbox_Checked)
```



```

4      CheckBoxGadget(2, 10, 70, 250, 20, "CheckBox three state",
#PB_CheckBox_ThreeState): SetGadgetState(2,
#PB_Checkbox_Inbetween)
5      CheckBoxGadget(3, 10, 100, 250, 20, "CheckBox right",
#PB_CheckBox_Right)
6      CheckBoxGadget(4, 10, 130, 250, 20, "CheckBox center",
#PB_CheckBox_Center)
7      Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
8      EndIf

```



See Also

GetGadgetState() , SetGadgetState() , OptionGadget()

121.11 ClearGadgetItems

Syntax

```
ClearGadgetItems (#Gadget)
```

Description

Clears all the items from the specified gadget.

Parameters

#Gadget The gadget to clear.

Return value

None.

Remarks

The Gadget must be one of the following types:

- ComboBoxGadget()
- EditorGadget()
- ListIconGadget()
- ListViewGadget()
- MDIGadget()
- PanelGadget()
- TreeGadget()

Example

```
1  If OpenWindow(0, 0, 0, 220, 150, "ClearGadgetItems",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      ListViewGadget(0,10,10,200,100)
3      For a = 1 To 10 : AddGadgetItem (0, -1, "Item "+Str(a)) : Next
        ; add 10 items
4      ButtonGadget(1, 10, 120, 150, 20, "Clear Listview contents")
5      Repeat
6          Event = WaitWindowEvent()
7          If Event = #PB_Event_Gadget
8              If EventGadget() = 1
9                  ClearGadgetItems(0)
10             EndIf
11         EndIf
12     Until Event = #PB_Event_CloseWindow
13 EndIf
```

See Also

AddGadgetItem() , RemoveGadgetItem() , CountGadgetItems()

121.12 CloseGadgetList

Syntax

```
CloseGadgetList()
```

Description

Terminate the current gadget list creation and go back to the previous GadgetList.

Parameters

None.

Return value

None.

Remarks

This is especially useful for the following gadgets:

- CanvasGadget()
- ContainerGadget()
- PanelGadget()
- ScrollAreaGadget()

See Also

OpenGadgetList() , ContainerGadget() , PanelGadget() , ScrollAreaGadget()

121.13 ComboBoxGadget

Syntax

```
Result = ComboBoxGadget(#Gadget, x, y, Width, Height [, Flags])
```

Description

Create a ComboBox gadget in the current GadgetList.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget. **Note:** on OS X, the height of a combobox can't be changed and this parameter will be ignored.

Flags (optional) Flags to modify the gadget behavior. It can be composed (using the '||' operator) of one of the following constants:

```
#PB_ComboBox_Editable    : Makes the ComboBox editable
#PB_ComboBox_LowerCase   : All text entered in the ComboBox
                           will be converted to lower case.
#PB_ComboBox_UpperCase   : All text entered in the ComboBox
                           will be converted to upper case.
#PB_ComboBox_Image       : Enable support for images in items
                           (not supported for editable ComboBox on OSX)
```

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using `GadgetToolTip()` .

The following functions can be used to act on the list contents:

- `AddGadgetItem()` : Add an item.
- `GetGadgetItemText()` : Returns the gadget item text.
- `CountGadgetItems()` : Count the items in the current combobox.
- `ClearGadgetItems()` : Remove all the items.
- `RemoveGadgetItem()` : Remove an item.
- `SetGadgetItemText()` : Changes the gadget item text.
- `SetGadgetItemImage()` : Changes the gadget item image (need to be created with the **#PB_ComboBox_Image** flag).
- `GetGadgetState()` : Get the index (starting from 0) of the current element, -1 if no element added or not selected.
- `GetGadgetText()` : Get the (text) content of the current element.
- `SetGadgetState()` : Change the selected element.
- `SetGadgetText()` : Set the displayed text. If the `ComboBoxGadget` is not editable, the text must be in the dropdown list.
- `GetGadgetItemData()` : Returns the value that was stored with item.
- `SetGadgetItemData()` : Stores a value with the item.

`ComboBoxGadget()` supports the following events reported by `EventType()` :

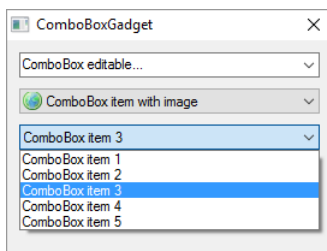
#PB_EventType_Change : The current selection of the text in the edit field changed.
#PB_EventType_Focus : The edit field received the keyboard focus (editable ComboBox only).
#PB_EventType_LostFocus: The edit field lost the keyboard focus (editable ComboBox only).

Example

```

1  UsePNGImageDecoder()
2  LoadImage(0, #PB_Compiler_Home +
   "examples/sources/Data/world.png")
3
4  If OpenWindow(0, 0, 0, 270, 180, "ComboBoxGadget",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
5      ComboBoxGadget(0, 10, 10, 250, 21, #PB_ComboBox_Editable)
6      AddGadgetItem(0, -1, "ComboBox editable...")
7
8      ComboBoxGadget(1, 10, 40, 250, 21, #PB_ComboBox_Image)
9      AddGadgetItem(1, -1, "ComboBox item with image", ImageID(0))
10
11     ComboBoxGadget(2, 10, 70, 250, 21)
12     For a = 1 To 5
13         AddGadgetItem(2, -1, "ComboBox item " + Str(a))
14     Next
15
16     SetGadgetState(0, 0)
17     SetGadgetState(1, 0)
18     SetGadgetState(2, 2) ; set (beginning with 0) the third item
   as active one
19
20     Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
21 EndIf

```



See Also

AddGadgetItem() , RemoveGadgetItem() , CountGadgetItems() , ClearGadgetItems() ,
 GetGadgetState() , SetGadgetState() , GetGadgetText() , SetGadgetText() GetGadgetItemText()
 , SetGadgetItemText() , SetGadgetItemImage() GetGadgetItemData() , SetGadgetItemData() ,
 ExplorerComboGadget()

121.14 ContainerGadget

Syntax

Result = ContainerGadget(#Gadget, x, y, Width, Height [, Flags])

Description

Creates a container gadget in the current GadgetList. It's a simple panel gadget which can contain other gadgets.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Flags (optional) Flags to modify the gadget behavior. It can be composed of one of the following constants:

```
#PB_Container_BorderLess : Without any border (Default).
#PB_Container_Flat       : Flat frame.
#PB_Container_Raised     : Raised frame.
#PB_Container_Single     : Single sunken frame.
#PB_Container_Double     : Double sunken frame.
```

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

Remarks

Once the gadget is created, all future created gadgets will be created inside the container. When all the needed gadgets have been created, `CloseGadgetList()` must be called to return to the previous GadgetList. `OpenGadgetList()` can be used later to add others gadgets on the fly in the container area.

The following event is supported through `EventType()` :

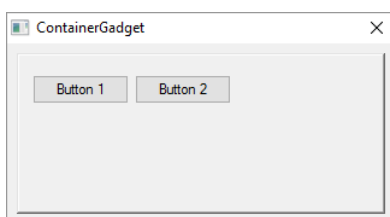
#PB_EventType_Resize: The gadget has been resized.

A 'mini help' can be added to this gadget using `GadgetToolTip()` .

This gadget supports the `SetGadgetColor()` and `GetGadgetColor()` functions with **#PB_Gadget_BackColor** as type to change the gadget background.

Example

```
1  If OpenWindow(0, 0, 0, 322, 150, "ContainerGadget",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      ContainerGadget(0, 8, 8, 306, 133, #PB_Container_Raised)
4      ButtonGadget(1, 10, 15, 80, 24, "Button 1")
5      ButtonGadget(2, 95, 15, 80, 24, "Button 2")
6      CloseGadgetList()
7      Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
      EndIf
```



See Also

OpenGadgetList() , CloseGadgetList() , SetGadgetColor() , GetGadgetColor()

121.15 CountGadgetItems

Syntax

```
Result = CountGadgetItems(#Gadget)
```

Description

Returns the number of items in the specified gadget.

Parameters

#Gadget The gadget to use.

Return value

Returns the number of items in the gadget.

Remarks

This is a universal function which works for all gadgets which handle several items:

- ComboBoxGadget()
- EditorGadget()
- ExplorerListGadget()
- ListIconGadget()
- ListViewGadget()
- MDIGadget()
- PanelGadget()
- TreeGadget()

See Also

AddGadgetItem() , RemoveGadgetItem() , ClearGadgetItems()

121.16 DateGadget

Syntax

```
Result = DateGadget(#Gadget, x, y, Width, Height [, Mask$ [, Date  
[, Flags]])
```

Description

Creates a String gadget in the current GadgetList, in which a date and/or time can be entered.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Mask\$ (optional) The format in which the date can be entered. See `FormatDate()` for the format of this mask. Important note: The gadget does not support the display of seconds, so if you specify "%ss" in the `Mask$` parameter, it will simply be ignored! If you don't specify the mask or specify an empty string, a default mask will be chosen. The mask can be modified with the `SetGadgetText()` function.

Date (optional) The initial date for the gadget. Not specifying this parameter or specifying a 0 value will display the current date.

Flags (optional) Flags to modify the gadget behavior:

By default, the gadget has a button to display a calendar in which the user can choose a date (see image below). You can change this by specifying `#PB_Date_UpDown` in the `Flags` parameter. This will make the gadget display an up/down button that lets the user change the current selected part of the gadget. This option is only available on Windows.

If you specify `#PB_Date_Checkbox` in the `Flags` parameter, the Gadget will have a checkbox with which the user can set the Gadget to 'no date' (if the checkbox is unchecked). While the checkbox is unchecked, `GetGadgetState()` will return 0. To change the state of the checkbox, use `SetGadgetState()` with either 0 (=checkbox unchecked) or any valid date (=checkbox checked).

Return value

Returns nonzero on success and zero on failure. If `#PB_Any` was used as the `#Gadget` parameter then the return-value is the auto-generated gadget number on success.

Remarks

This gadget uses the same date format for its functions as used by the `Date` library . So you can use for example `FormatDate()` to display the results you get from `GetGadgetState()` in a proper format.

A 'mini help' can be added to this gadget using `GadgetToolTip()` .

The following functions can be used for this gadget:

- `SetGadgetState()` : Set the currently displayed date.
- `SetGadgetText()` : Change the input mask of the gadget.
- `GetGadgetState()` : Get the currently displayed date.
- `GetGadgetText()` : Get the current displayed date as a string, as it is displayed in the gadget.
- `GetGadgetAttribute()` : With the following attributes:

```
#PB_Date_Minimum: Get the minimum date that can be entered
#PB_Date_Maximum: Get the maximum date that can be entered
                  (Note: Limitation of the selectable date isn't
supported on Linux.)
```

- `SetGadgetAttribute()` : With the following attributes:

```
#PB_Date_Minimum: Set the minimum date that can be entered
#PB_Date_Maximum: Set the maximum date that can be entered
                  (Note: Limitation of the selectable date isn't
supported on Linux.)
```

The following events are supported through `EventType()` :

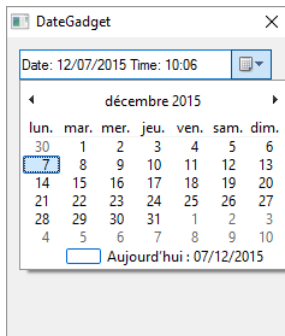
```
#PB_EventType_Change: The date has been modified by the user.
```

This gadget supports the SetGadgetColor() and GetGadgetColor() functions with the following values as 'ColorType' to color the dropdown calendar (the edit area cannot be colored):

```
#PB_Gadget_BackColor      : backgroundcolor
#PB_Gadget_FrontColor     : textcolor for displayed days
#PB_Gadget_TitleBackColor : backgroundcolor of the month title
#PB_Gadget_TitleFrontColor : textcolor of the month title
#PB_Gadget_GrayTextColor  : textcolor for days not of the current
                             month
```

Example

```
1  If OpenWindow(0, 0, 0, 235, 250, "DateGadget",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      DateGadget(0, 10, 10, 210, 25, "Date: %mm/%dd/%yyyy Time:
4      %hh:%ii")
5      Repeat
6          Until WaitWindowEvent() = #PB_Event_CloseWindow
7      EndIf
```



See Also

GetGadgetState(), SetGadgetState(), GetGadgetText(), SetGadgetText(),
 GetGadgetAttribute(), SetGadgetAttribute(), GetGadgetColor(), SetGadgetColor(),
 CalendarGadget(), Date(), FormatDate()

121.17 DisableGadget

Syntax

```
DisableGadget(#Gadget, State)
```

Description

Disable or enable the gadget.

Parameters

#Gadget The gadget to enable or disable.

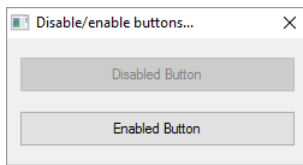
State **#False**: The gadget will be enabled.
#True : The gadget will be disabled.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 250, 105, "Disable/enable buttons...",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      ButtonGadget(0, 10, 15, 230, 30, "Disabled Button") :
    DisableGadget(0, #True)
3      ButtonGadget(1, 10, 60, 230, 30, "Enabled Button") :
    DisableGadget(1, #False)
4      Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
5  EndIf
```



See Also

HideGadget()

121.18 EditorGadget

Syntax

```
Result = EditorGadget(#Gadget, x, y, Width, Height [, Flags])
```

Description

Creates an Editor gadget in the current GadgetList.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Flags (optional) Flags to modify the gadget behavior. This can be a combination of the following values:

```
#PB_Editor_ReadOnly      : the user cannot edit the text in
    the gadget.
#PB_Editor_WordWrap      : the lines too long to be displayed
    will be wrapped still displayed completely
#PB_Editor_TabNavigation: when using TAB in the editor gadget
    it will cycle to the next gadget instead of inserting a TAB
    character.
```

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using `GadgetToolTip()` .
The following events are supported through `EventType()` :

```
#PB_EventType_Change      : the text has been modified by the user.
#PB_EventType_Focus       : the editor has got the focus.
#PB_EventType_LostFocus   : the editor has lost the focus.
```

The following functions can be used to act on the editor content:

- `AddGadgetItem()` : Add a text line.
- `CountGadgetItems()` : Return the number of lines in the editor gadget.
- `GetGadgetItemText()` : Get the specified line text.
- `GetGadgetText()` : Get the text content of the editor gadget. Please note, that several lines of text are separated with "Chr(13)+Chr(10)" on Windows and "Chr(10)" on Linux and OS X.
- `RemoveGadgetItem()` : Remove a line in the editor.
- `ClearGadgetItems()` : Clear the text content.
- `SetGadgetItemText()` : Set the specified line text.
- `SetGadgetText()` : Change the text content of the editor gadget.
- `SetGadgetAttribute()` : With the following attribute:

```
#PB_Editor_ReadOnly: set the read-only state (zero means
                     editable, nonzero means read-only).
#PB_Editor_WordWrap: set the word-wrap state
```

- `GetGadgetAttribute()` : With the following attribute:

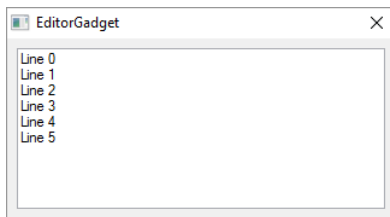
```
#PB_Editor_ReadOnly: get the read-only state (zero means
                     editable, nonzero means read-only).
#PB_Editor_WordWrap: get the word-wrap state
```

This gadget supports the `SetGadgetColor()` and `GetGadgetColor()` functions with the following values as 'ColorType':

```
#PB_Gadget_BackColor      : backgroundcolor (not supported on
                             MacOSX)
#PB_Gadget_FrontColor     : textcolor
```

Example

```
1  If OpenWindow(0, 0, 0, 322, 150, "EditorGadget",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     EditorGadget(0, 8, 8, 306, 133)
4     For a = 0 To 5
5         AddGadgetItem(0, a, "Line "+Str(a))
6     Next
7     Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
8     EndIf
```



See Also

`AddGadgetItem()` , `RemoveGadgetItem()` , `CountGadgetItems()` , `ClearGadgetItems()` ,
`GetGadgetText()` , `SetGadgetText()` , `GetGadgetItemText()` , `SetGadgetItemText()` ,

GetGadgetAttribute() , SetGadgetAttribute() , GetGadgetColor() , SetGadgetColor() ,
StringGadget()

121.19 ExplorerComboGadget

Syntax

```
Result = ExplorerComboGadget(#Gadget, x, y, Width, Height,  
    Directory$, [, Flags])
```

Description

Creates a ComboBox that lets you display a path and all its parent folders, so the user can choose one of them. You can find such a ComboBox, for example, in the OpenFileRequester() .

Parameters

#Gadget A number to identify the new gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Directory\$ The initial displayed directory (must be set as full path), an empty string specifies the root folder.

If the #PB_Explorer_DrivesOnly flag is set, Directory\$ may only be a drive letter. Everything that follows the drive letter in this case will be ignored.

Flags (optional) Flags to modify the gadget behavior. This can be a combination of the following values:

```
#PB_Explorer_DrivesOnly    : The gadget will only display  
    drives to choose from.  
#PB_Explorer_Editable      : The gadget will be editable with  
    an autocomplete feature. With this flag set, it acts exactly  
    like the one in Windows Explorer.  
#PB_Explorer_NoMyDocuments: The 'My Documents' Folder will  
    not be displayed as a separate item.
```

Return value

Returns nonzero on success and zero on failure. If #PB_Any was used as the #Gadget parameter then the return-value is the auto-generated gadget number on success.

Remarks

The following functions can be used to control the gadget:

- GetGadgetText() : Get the currently displayed directory. Use this to check what the user has selected after you get an event for this Gadget.
- SetGadgetText() : Changes the currently displayed directory.

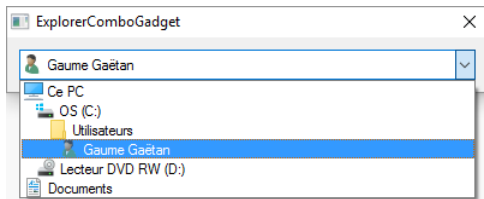
Example

```
1  If OpenWindow(0, 0, 0, 400, 45, "ExplorerComboGadget",  
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
2      ExplorerComboGadget(0, 10, 10, 380, 25, GetHomeDirectory(),  
    #PB_Explorer_Editable)  
3      Repeat
```

```

4      Event = WaitWindowEvent()
5      Until Event = #PB_Event_CloseWindow
6      EndIf

```



See Also

GetGadgetText() , SetGadgetText() , ExplorerListGadget() , ExplorerTreeGadget() ,
 ComboBoxGadget()

121.20 ExplorerListGadget

Syntax

```

Result = ExplorerListGadget(#Gadget, x, y, Width, Height,
    Directory$, [, Flags])

```

Description

Creates a listing of a directory just as Explorer does. It lets the user choose a file or a folder and (if you do not prevent it by a flag) navigate through the whole directory tree.

Parameters

#Gadget A number to identify the new gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Directory\$ The initial displayed directory, it can include one or multiple patterns, like "C:*.pb;*.pbi". If no pattern is included, the directory must end with a '\'. Including no directory will display the root containing the drives. Including no pattern defaults to '*.*'. So a Directory\$ of "" will display the root and set '*.*' as pattern.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

#PB_Explorer_BorderLess	: Create Gadget without borders.
#PB_Explorer_AlwaysShowSelection	: The selection is visible, even when the gadget is not activated.
#PB_Explorer_MultiSelect	: Enable multiple selection of items in the gadget.
#PB_Explorer_GridLines	: Display separator lines between rows and columns.
#PB_Explorer_HeaderDragDrop	: In report view, the headers can be changed by Drag'n'Drop.
#PB_Explorer_FullRowSelect	: The selection covers the full row instead of the first column.
#PB_Explorer_NoFiles	: No files will be displayed.

```

#PB_Explorer_NoFolders          : No folders will be
    displayed.
#PB_Explorer_NoParentFolder     : There will be no [...] link
    to the parent folder.
#PB_Explorer_NoDirectoryChange  : The directory cannot be
    changed by the user.
#PB_Explorer_NoDriveRequester   : There will be no 'please
    insert drive X:' displayed.
#PB_Explorer_NoSort            : The user cannot sort the
    content by clicking on a column header.
#PB_Explorer_NoMyDocuments      : The 'My Documents' Folder
    will not be displayed as a separate item.
#PB_Explorer_AutoSort           : The content will be sorted
    automatically by name.
#PB_Explorer_HiddenFiles        : Will display hidden files
    as well (supported on Linux and OS X only).

```

Return value

Returns nonzero on success and zero on failure. If `#PB_Any` was used as the `#Gadget` parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using `GadgetToolTip()` .

The following functions can be used to control the gadget:

- `AddGadgetColumn()` : Add a new automatically or custom filled column to the gadget. See the `AddGadgetColumn()` function help for more details.
- `RemoveGadgetColumn()` : Remove a column from the gadget.
- `GetGadgetText()` : Get the currently displayed directory.
- `SetGadgetText()` : Changes the currently displayed directory, or the current pattern for files.
- `GetGadgetState()` : Get the first selected item (-1 if none selected).
- `GetGadgetItemText()` : Get the name of an item (or column header, if item = -1).
- `SetGadgetItemText()` : Alter the contents of any items text (or column header, if item = -1), or fill a custom column with data.
- `GetGadgetItemState()` : Check if an item is a directory or a file, and if it is currently selected.
- `SetGadgetItemState()` : Change selected state of the specified item.
- `CountGadgetItems()` : Count the items in the current directory.
- `GetGadgetAttribute()` / `SetGadgetAttribute()` : With the following attribute:

```

#PB_Explorer_DisplayMode : Changes the display of the gadget. Can
    be one of the following constants (Windows only):
    #PB_Explorer_LargeIcon: Large icon mode
    #PB_Explorer_SmallIcon: Small icon mode
    #PB_Explorer_List      : List icon mode
    #PB_Explorer_Report    : Report mode (columns, default mode)

```

- `GetGadgetItemAttribute()` / `SetGadgetItemAttribute()` : With the following attribute:

```

#PB_Explorer_ColumnWidth : Returns/Changes the width of the given
    'Column'. The 'Item' parameter is ignored.

```

This gadget supports the `SetGadgetColor()` and `GetGadgetColor()` functions with the following values as 'ColorType':

```

#PB_Gadget_FrontColor: Textcolor
#PB_Gadget_BackColor  : Backgroundcolor
#PB_Gadget_LineColor  : Color for the gridlines if the
    #PB_Explorer_GridLines flag is used.

```

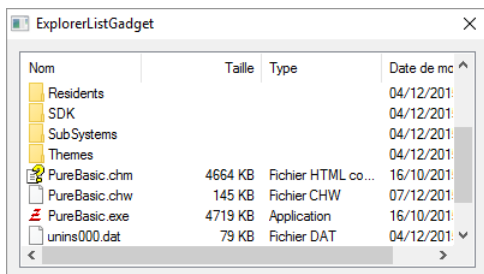
ExplorerListGadget() supports the following events reported by EventType() :

```
#PB_EventType_Change           : The selection or the current
    displayed directory has changed.
#PB_EventType_LeftClick        : User clicked on an item with the
    left mouse button.
#PB_EventType_RightClick       : User clicked on an item with the
    right mouse button.
#PB_EventType_LeftDoubleClick  : User double-clicked on an item
    with the left mouse button.
#PB_EventType_RightDoubleClick : User double-clicked on an item
    with the right mouse button.
#PB_EventType_DragStart        : User tried to start a Drag &
    Drop operation. ExamineDraggedItems()
can be used to get the dragged items information.
#PB_EventType_Refresh          : The gadget content has been
    refreshed due to a filesystem change in the displayed directory
    (Windows only).
```

After a #PB_EventType_DragStart event, the Drag & Drop library can be used to start a Drag & Drop operation. On MacOS X the #PB_EventType_DragStart event has to be handled using BindEvent() or BindGadgetEvent() .

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "ExplorerListGadget",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      ExplorerListGadget(0, 10, 10, 380, 180, " *.*",
4      #PB_Explorer_MultiSelect)
      Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
      EndIf
```



See Also

AddGadgetColumn() , RemoveGadgetColumn() , GetGadgetText() , SetGadgetText() ,
GetGadgetState() , GetGadgetItemState() , GetGadgetItemText() , SetGadgetItemText() ,
CountGadgetItems() , GetGadgetAttribute() , SetGadgetAttribute() , GetGadgetItemAttribute() ,
SetGadgetItemAttribute() ,SetGadgetColor() , SetGadgetItemState() , GetGadgetColor() ,
ExplorerComboGadget() , ExplorerTreeGadget() , ListIconGadget()

121.21 ExplorerTreeGadget

Syntax

```
Result = ExplorerTreeGadget(#Gadget, x, y, Width, Height,  
    Directory$, [, Flags])
```

Description

Creates a tree listing of the directory tree just as Explorer does. It lets the user navigate through his file-system, and choose a file or folder.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Directory\$ The directory that will be initially selected. It can include one or multiple patterns, like "C:*.pb;*.pbi". If no pattern is included, the directory must end with a '\'. Including no directory will display the root containing the drives. Including no pattern defaults to '*.*'. So a Directory\$ of "" will display the root and set '*.*' as pattern.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

```
#PB_Explorer_BorderLess      : Create Gadget without
                                borders.
#PB_Explorer_AlwaysShowSelection : The selection is still
                                visible, even when the gadget is not
                                activated.
#PB_Explorer_NoLines        : Hide the little lines
                                between each node.
#PB_Explorer_NoButtons      : Hide the '+' node buttons.

#PB_Explorer_NoFiles        : No files will be displayed.
#PB_Explorer_NoDriveRequester : There will be no 'please
                                insert drive X:' displayed.
#PB_Explorer_NoMyDocuments  : The 'My Documents' Folder
                                will not be displayed as a separate
                                item.
#PB_Explorer_AutoSort       : The content will be sorted
                                automatically by name.
```

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using GadgetToolTip() .

The following functions can be used to control the gadget:

- GetGadgetText() : Get the full path of the currently selected directory/file.
- SetGadgetText() : Set the currently selected file/directory.
- GetGadgetState() : Check if the selected item is a file or a directory.
- GetGadgetItemText() : Returns the full path of the specified item.

This gadget supports the SetGadgetColor() and GetGadgetColor() functions with the following values as 'ColorType':

```
#PB_Gadget_FrontColor : Textcolor
#PB_Gadget_BackColor  : Backgroundcolor
#PB_Gadget_LineColor  : Color for the lines and buttons if they
                        are displayed
```

ExplorerTreeGadget() supports the following Events reported by EventType() :

```

#PB_EventType_Change           : The selection has changed.
#PB_EventType_LeftClick       : User clicked on an item with the
    left mouse button.
#PB_EventType_RightClick      : User clicked on an item with the
    right mouse button.
#PB_EventType_LeftDoubleClick  : User double-clicked on an item
    with the left mouse button.
#PB_EventType_RightDoubleClick : User double-clicked on an item
    with the right mouse button.
#PB_EventType_DragStart       : User tried to start a Drag &
    Drop operation. ExamineDraggedItems()
    can be used to get the dragged items information.

```

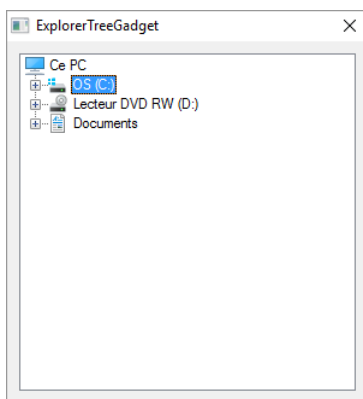
After a `#PB_EventType_DragStart` event, the Drag & Drop library can be used to start a Drag & Drop operation. On MacOS X the `#PB_EventType_DragStart` event has to be handled using `BindEvent()` or `BindGadgetEvent()` .

Example

```

1  If OpenWindow(0, 0, 0, 300, 300, "ExplorerTreeGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      ExplorerTreeGadget(0, 10, 10, 280, 280, "*.pb;*.pbi")
3      Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
4      EndIf

```



See Also

`GetGadgetText()` , `SetGadgetText()` , `GetGadgetState()` , `GetGadgetColor()` , `SetGadgetColor()` , `ExplorerComboGadget()` , `ExplorerListGadget()` , `TreeGadget()`

121.22 FrameGadget

Syntax

```
Result = FrameGadget(#Gadget, x, y, Width, Height, Text$ [, Flags])
```

Description

Creates a Frame gadget in the current GadgetList. This kind of gadget is decorative only.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Text\$ A text to display in the frame. This parameter is only valid if no borders are specified, else it will be ignored.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

#PB_Frame_Container: Enables container support so gadgets can be added to it (strongly recommended to avoid various issues).

CloseGadgetList()

needs to be called to go back to previous gadget list like any other container gadget.

#PB_Frame_Single : Single sunken frame (Windows only).

#PB_Frame_Double : Double sunken frame (Windows only).

#PB_Frame_Flat : Flat frame (Windows only).

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

Remarks

As this Gadget is decorative only, **GadgetToolTip()** cannot be used. This Gadget also receives no events.

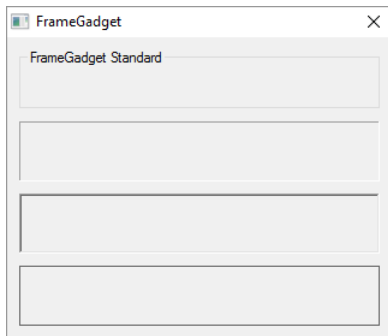
Example: With Container

```
1  If OpenWindow(0, 0, 0, 320, 250, "FrameGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      FrameGadget(0, 10, 10, 300, 200, "Options",
    #PB_Frame_Container)
3          CheckBoxGadget(1, 10, 40, 200, 30, "Enable Optimizer")
4          CheckBoxGadget(2, 10, 75, 200, 30, "Validate Output")
5          CloseGadgetList()
6
7      Repeat
8          Until WaitWindowEvent() = #PB_Event_CloseWindow
9  EndIf
```

Example

```
1  If OpenWindow(0, 0, 0, 320, 250, "FrameGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      FrameGadget(0, 10, 10, 300, 50, "FrameGadget Standard")
3      FrameGadget(1, 10, 70, 300, 50, "", #PB_Frame_Single)
4      FrameGadget(2, 10, 130, 300, 50, "", #PB_Frame_Double)
5      FrameGadget(3, 10, 190, 300, 50, "", #PB_Frame_Flat)
6
7      Repeat
```

```
8     Until WaitWindowEvent() = #PB_Event_CloseWindow
9     EndIf
```



See Also

GetGadgetText() , SetGadgetText() , ContainerGadget()

121.23 FreeGadget

Syntax

```
FreeGadget( #Gadget )
```

Description

Free and remove the gadget from the display (and free its gadgetlist if the gadget was a container).

Parameters

#Gadget The gadget to free. If **#PB_All** is specified, all the remaining gadgets are freed.

Return value

None.

Remarks

A gadget is freed automatically if one of the following happens:

- The window that contains the gadget is closed .
- The parent of the gadget (ContainerGadget() , PanelGadget() etc.) is freed.
- The program ends.

Bound events (using BindEvent()) are automatically resolved when freeing the gadget.

See Also

IsGadget() , CloseWindow()

121.24 GadgetID

Syntax

```
GadgetID = GadgetID(#Gadget)
```

Description

Returns the unique system identifier of the #Gadget.

Parameters

#Gadget The gadget to use.

Return value

Returns the system identifier. This result is sometimes also known as 'Handle'. Look at the extra chapter Handles and Numbers for more information.

121.25 GadgetItemID

Syntax

```
Result = GadgetItemID(#Gadget, Item)
```

Description

Returns the OS handle for the given item in a gadget. This is especially useful for use with the OS API.

Parameters

#Gadget The gadget to use.

Item The item of which the handle should be returned. The first item in the gadget has index 0.

Return value

Returns the OS handle for the item or zero on failure.

Remarks

This function is currently only supported by TreeGadget() and PanelGadget() on Windows. It returns zero in other cases.

See Also

TreeGadget()

Supported OS

Windows

121.26 GadgetToolTip

Syntax

```
GadgetToolTip(#Gadget , Text$)
```

Description

Associate the specified **Text\$** with the **#Gadget**. A tooltip text is text which is displayed when the mouse cursor is over the gadget for a small amount of time (typically a yellow floating box).

Parameters

#Gadget The gadget to use.

Text\$ The tooltip text.

Return value

None.

Remarks

The following gadgets are supported:

- ButtonGadget()
- ButtonImageGadget()
- CalendarGadget()
- CanvasGadget()
- CheckBoxGadget()
- ComboBoxGadget()
- ContainerGadget()
- DateGadget()
- EditorGadget()
- ExplorerListGadget()
- ExplorerTreeGadget()
- HyperLinkGadget()
- ImageGadget()
- IPAddressGadget()
- ListIconGadget()
- ListViewGadget()
- MDIGadget()
- OpenGLGadget()
- OptionGadget()
- PanelGadget()
- ProgressBarGadget()
- ScrollBarGadget()
- SpinGadget()
- SplitterGadget()
- StringGadget()
- TrackBarGadget()
- TreeGadget()

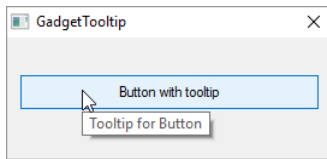
Example

```
1  If OpenWindow(0, 0, 0, 270, 100, "GadgetTooltip",  
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
    ButtonGadget(0, 10, 30, 250, 30, "Button with tooltip")
```

```

3   GadgetToolTip(0, "Tooltip for Button")
4   Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
5   EndIf

```



121.27 GadgetX

Syntax

```
Result = GadgetX(#Gadget [, Mode])
```

Description

Returns the X position of the specified gadget.

Parameters

#Gadget The gadget to use.

Mode (optional) Mode can be one of the following value:

```

#PB_Gadget_ContainerCoordinate: the gadget X position (in
pixels) within its container, or window (default)
#PB_Gadget_WindowCoordinate  : the absolute gadget X
position (in pixels) within the window.
#PB_Gadget_ScreenCoordinate  : the absolute gadget X
position (in pixels) in the screen.

```

It can be useful to display something over the gadget like a floating window or a popup menu.

Return value

Returns the X position relative to the gadgets parent or window (in pixels).

See Also

GadgetY() , GadgetWidth() , GadgetHeight() , ResizeGadget()

121.28 GadgetY

Syntax

```
Result = GadgetY(#Gadget [, Mode])
```

Description

Returns the Y position of the specified gadget.

Parameters

#Gadget The gadget to use.

Mode (optional) Mode can be one of the following value:

```
#PB_Gadget_ContainerCoordinate: the gadget Y position (in
pixels) within its container, or window (default)
#PB_Gadget_WindowCoordinate   : the absolute gadget Y
position (in pixels) within the window.
#PB_Gadget_ScreenCoordinate   : the absolute gadget Y
position (in pixels) in the screen.

It can be useful to display
something over the gadget like a floating window or a popup
menu.
```

Return value

Returns the Y position relative to the gadgets parent or window in pixels.

See Also

GadgetX() , GadgetWidth() , GadgetHeight() , ResizeGadget()

121.29 GadgetHeight

Syntax

```
Result = GadgetHeight(#Gadget [, Mode])
```

Description

Returns the height of the specified gadget.

Parameters

#Gadget The gadget to use.

Mode (optional) Can be one of the following value:

```
#PB_Gadget_ActualSize   : returns the current height of the
gadget, in pixels (default).
#PB_Gadget_RequiredSize: returns the height needed to fully
display the gadget, in pixels.
```

Return value

Returns the height of the gadget in pixels.

See Also

GadgetWidth() , GadgetX() , GadgetY() , ResizeGadget()

121.30 GadgetType

Syntax

```
Result = GadgetType(#Gadget)
```

Description

Returns the type of gadget that is represented by the specified gadget number. It can be useful to write generic functions that work with more than one type of gadget.

Parameters

#Gadget The gadget to use.

Return value

Returns one of the following values:

<code>#PB_GadgetType_Button</code>	: <code>ButtonGadget()</code>
<code>#PB_GadgetType_ButtonImage</code>	: <code>ButtonImageGadget()</code>
<code>#PB_GadgetType_Calendar</code>	: <code>CalendarGadget()</code>
<code>#PB_GadgetType_Canvas</code>	: <code>CanvasGadget()</code>
<code>#PB_GadgetType_CheckBox</code>	: <code>CheckBoxGadget()</code>
<code>#PB_GadgetType_ComboBox</code>	: <code>ComboBoxGadget()</code>
<code>#PB_GadgetType_Container</code>	: <code>ContainerGadget()</code>
<code>#PB_GadgetType_Date</code>	: <code>DateGadget()</code>
<code>#PB_GadgetType_Editor</code>	: <code>EditorGadget()</code>
<code>#PB_GadgetType_ExplorerCombo</code>	: <code>ExplorerComboGadget()</code>
<code>#PB_GadgetType_ExplorerList</code>	: <code>ExplorerListGadget()</code>
<code>#PB_GadgetType_ExplorerTree</code>	: <code>ExplorerTreeGadget()</code>
<code>#PB_GadgetType_Frame</code>	: <code>FrameGadget()</code>
<code>#PB_GadgetType_HyperLink</code>	: <code>HyperLinkGadget()</code>
<code>#PB_GadgetType_Image</code>	: <code>ImageGadget()</code>
<code>#PB_GadgetType_IPAddress</code>	: <code>IPAddressGadget()</code>
<code>#PB_GadgetType_ListIcon</code>	: <code>ListIconGadget()</code>
<code>#PB_GadgetType_ListView</code>	: <code>ListViewGadget()</code>
<code>#PB_GadgetType_MDI</code>	: <code>MDIGadget()</code>
<code>#PB_GadgetType_OpenGL</code>	: <code>OpenGLGadget()</code>

```

#PB_GadgetType_Option      : OptionGadget()
#PB_GadgetType_Panel       : PanelGadget()
#PB_GadgetType_ProgressBar : ProgressBarGadget()
#PB_GadgetType_Scintilla   : ScintillaGadget()
#PB_GadgetType_ScrollArea  : ScrollAreaGadget()
#PB_GadgetType_ScrollBar   : ScrollBarGadget()
#PB_GadgetType_Shortcut    : ShortcutGadget()
#PB_GadgetType_Spin        : SpinGadget()
#PB_GadgetType_Splitter    : SplitterGadget()
#PB_GadgetType_String      : StringGadget()
#PB_GadgetType_Text        : TextGadget()
#PB_GadgetType_TrackBar    : TrackBarGadget()
#PB_GadgetType_Tree        : TreeGadget()
#PB_GadgetType_Web         : WebGadget()
#PB_GadgetType_WebView     : WebViewGadget()

#PB_GadgetType_Unknown     : The type is unknown. Most likely
    it is not a PB gadget at all.

```

121.31 GadgetWidth

Syntax

```
Result = GadgetWidth(#Gadget [, Mode])
```

Description

Returns the width of the specified gadget.

Parameters

#Gadget The gadget to use.

Mode (optional) Can be one of the following value:

```

#PB_Gadget_ActualSize : returns the current width of the
    gadget, in pixels (default).
#PB_Gadget_RequiredSize: returns the width needed to fully
    display the gadget, in pixels.

```

Return value

Returns the width of the gadget in pixels.

See Also

GadgetHeight() , GadgetX() , GadgetY() , ResizeGadget()

121.32 GetActiveGadget

Syntax

```
Result = GetActiveGadget()
```

Description

Returns the gadget number of the Gadget that currently has the keyboard focus.

Parameters

None.

Return value

Returns the #Gadget number of the Gadget with the focus. If no Gadget has the focus, -1 is returned.

Example

```
1  If OpenWindow(0, 0, 0, 270, 70, "GetActiveGadget",  
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
2      StringGadget (0, 10, 10, 250, 20, "Press escape...")  
3      StringGadget (1, 10, 40, 250, 20, "Press escape...")  
4      AddKeyboardShortcut(0, #PB_Shortcut_Escape, 1)  
5      SetActiveGadget(0)  
6      Repeat  
7          Event = WaitWindowEvent()  
8          If Event = #PB_Event_Menu And EventMenu() = 1  
9              MessageRequester("Test", "Escape pressed in Gadget " +  
                Str(GetActiveGadget()))  
10             EndIf  
11         Until Event = #PB_Event_CloseWindow  
12     EndIf
```

121.33 GetGadgetAttribute

Syntax

```
Value = GetGadgetAttribute(#Gadget, Attribute)
```

Description

Gets an attribute value of the specified gadget.

Parameters

#Gadget The gadget to use.

Attribute The attribute to get.

Return value

Returns the value of the specified gadget attribute or 0 if the gadget does not support the attribute.

Remarks

This function is available for all gadgets which support attributes. See the individual gadget for the supported attributes:

- ButtonImageGadget()
- CalendarGadget()
- CanvasGadget()
- DateGadget()
- EditorGadget()
- ExplorerListGadget()
- ListIconGadget()
- OpenGLGadget()
- PanelGadget()
- ProgressBarGadget()
- ScrollAreaGadget()
- ScrollBarGadget()
- SpinGadget()
- SplitterGadget()
- StringGadget()
- TrackBarGadget()
- WebGadget()

See Also

SetGadgetAttribute() , GetGadgetItemAttribute() , SetGadgetItemAttribute()

121.34 GetGadgetColor

Syntax

```
Color = GetGadgetColor(#Gadget , ColorType)
```

Description

Returns a color setting from the specified gadget.

Parameters

#Gadget The gadget to use.

ColorType The setting to get. This can be one of the following values:

```
#PB_Gadget_FrontColor      : Gadget text
#PB_Gadget_BackColor       : Gadget background
#PB_Gadget_LineColor       : Color for gridlines
#PB_Gadget_TitleFrontColor: Text color in the title
    (for CalendarGadget())
)
#PB_Gadget_TitleBackColor : Background color in the title
    (for CalendarGadget())
)
#PB_Gadget_GrayTextColor   : Color for grayed out text
    (for CalendarGadget())
)
```

Return value

Returns the current color setting. This function returns the color that was previously set by `SetGadgetColor()` . If no custom color is set for the `#Gadget` and `ColorType`, the function returns `#PB_Default`.

Remarks

This function is supported by the following gadgets: (See each gadget description for the supported `ColorType` values.)

- `CalendarGadget()`
- `ContainerGadget()`
- `DateGadget()`
- `EditorGadget()`
- `ExplorerListGadget()`
- `ExplorerTreeGadget()`
- `HyperLinkGadget()`
- `ListViewGadget()`
- `ListIconGadget()`
- `MDIGadget()`
- `ProgressBarGadget()`
- `ScrollAreaGadget()`
- `SpinGadget()`
- `StringGadget()`
- `TextGadget()`
- `TreeGadget()`

Note: With activated Windows style the color will probably be overwritten by the style.

See Also

`SetGadgetColor()` , `GetGadgetItemColor()` , `SetGadgetItemColor()`

121.35 GetGadgetData

Syntax

```
Result = GetGadgetData(#Gadget)
```

Description

Returns the 'Data' value that has been stored for this gadget with the `SetGadgetData()` function. This allows to associate a custom value with any gadget.

Parameters

#Gadget The gadget to use.

Return value

Returns the current data value. If there was never a data value set for this gadget, the return-value will be 0.

Remarks

This function works with all PureBasic gadgets. See `SetGadgetData()` for an example.

See Also

SetGadgetData() , GetGadgetItemData() , SetGadgetItemData()

121.36 GetGadgetFont

Syntax

```
FontID = GetGadgetFont(#Gadget)
```

Description

Get the FontID associated with the specified gadget.

Parameters

#Gadget The gadget to use. If the **#PB_Default** constant is used as the gadget then the default FontID used for new created gadgets is returned.

Return value

Returns the FontID for the gadget or the one used for newly created gadgets.

Note: On Mac OSX, this function returns 0 if the gadget does not have a specific font associated with it.

See Also

SetGadgetFont() , FontID()

121.37 GetGadgetItemAttribute

Syntax

```
Value = GetGadgetItemAttribute(#Gadget, Item, Attribute [, Column])
```

Description

Gets an attribute value of the specified gadget item.

Parameters

#Gadget The gadget to use.

Item The item from which to get the attribute. The first item in the gadget has index 0.

Attribute The attribute to get.

Column (optional) The column from which to get the attribute on gadgets that support multiple columns. The first column has index 0. The default is column 0.

Return value

Returns the attribute value or zero if the item or attribute does not exist.

Remarks

This function is available for all gadgets which support item attributes:

- ExplorerListGadget() :

`#PB_Explorer_ColumnWidth` : Returns the width of the given 'Column'. The 'Item' parameter is ignored.

- ListIconGadget() :

`#PB_ListIcon_ColumnWidth` : Returns the width of the given 'Column'. The 'Item' parameter is ignored.

- TreeGadget() :

`#PB_Tree_SubLevel` : Returns the sublevel of the given item

The sublevel value of the tree item can be used to determine relations between items in the tree. For example the first item with a sublevel smaller than the current item, if the list is checked backwards, is the parent of the current item.

See Also

SetGadgetItemAttribute() , GetGadgetAttribute() , SetGadgetAttribute()

121.38 GetGadgetItemColor

Syntax

`Color = GetGadgetItemColor(#Gadget, Item, ColorType [, Column])`

Description

Returns a color setting from the specified gadget item.

Parameters

#Gadget The gadget to use.

Item The item from which to get the color. The first item in the gadget has index 0.

ColorType The setting to get. This can be one of the following values:

`#PB_Gadget_FrontColor` : Item text
`#PB_Gadget_BackColor` : Item background

Column (optional) The column from which to get the color for gadgets that support multiple columns. The first column has index 0. The default is column 0.

Return value

Returns the current color setting. This function returns the color that was previously set by SetGadgetItemColor() . If no custom color is set for the #Gadget and ColorType, the function returns #PB_Default.

Remarks

This function is supported by the following gadgets:

- ListIconGadget()

- TreeGadget()

Note: With activated Windows style the color will probably be overwritten by the style.

See Also

SetGadgetItemColor() , GetGadgetColor() , SetGadgetColor()

121.39 GetGadgetItemData

Syntax

```
Result = GetGadgetItemData(#Gadget, Item)
```

Description

Returns the value that was previously stored with this gadget item with the SetGadgetItemData() function. This allows to associate a custom value with the items of a gadget.

Parameters

#Gadget The gadget to use.

Item The item from which the data should be read. The first item in the gadget has index 0.

Return value

Returns the stored data. If no value has been stored with the item yet, the return-value will be 0.

Remarks

This function works with the following gadgets:

- ComboBoxGadget()
- ListIconGadget()
- ListViewGadget()
- PanelGadget()
- TreeGadget()

See SetGadgetItemData() for an example.

See Also

SetGadgetItemData() , GetGadgetData() , SetGadgetData()

121.40 GetGadgetState

Syntax

```
Result = GetGadgetState(#Gadget)
```

Description

Returns the current state of the gadget.

Parameters

#Gadget The gadget to use.

Return value

Returns the state of the gadget. See below for its meaning depending on the gadget type.

Remarks

This is a universal function which works for almost all gadgets:

- ButtonImageGadget() : returns 1 if a #PB_Button_Toggle button is toggled, else 0.
- ButtonGadget() : returns 1 if a #PB_Button_Toggle button is toggled, else 0.
- CalendarGadget() : returns the currently selected date.
- CheckBoxGadget() : Returns one of the following values:


```
#PB_CheckBox_Checked    : The check mark is set.  
#PB_CheckBox_Unchecked: The check mark is not set.  
#PB_CheckBox_Inbetween: The "in between" state is set. (Only for  
#PB_CheckBox_ThreeState checkboxes)
```
- ComboBoxGadget() : returns the currently selected item index, -1 if none is selected.
- DateGadget() : returns the currently selected date/time. IF #PB_Date_CheckBox was used, and the checkbox is unchecked, 0 is returned.
- ExplorerListGadget() : returns the index of the first selected item in the Gadget, -1 if none is selected.
- ExplorerTreeGadget() : returns the type of the currently selected item (#PB_Explorer_File or #PB_Explorer_Directory).
- ImageGadget() : returns the ImageID of the currently displayed image.
- IPAddressGadget() : returns the current IP address.
- ListIconGadget() : returns the first selected item index, -1 if none is selected.
- ListViewGadget() : returns the currently selected item index, -1 if none is selected.
- MDIGadget() : returns the currently focused child window, -1 if none has the focus.
- OptionGadget() : returns 1 if activated, 0 otherwise.
- PanelGadget() : returns the current panel index, -1 if no panel.
- ProgressBarGadget() : returns the current value of the ProgressBar.
- ScrollBarGadget() : returns the current slider position.
- ShortcutGadget() : returns the currently selected keyboard shortcut.
- SpinGadget() : returns the current value of the SpinGadget.
- SplitterGadget() : returns the current splitter position, in pixels.
- TrackBarGadget() : returns the current position of the TrackBar (value inside the minimum - maximum range).
- TreeGadget() : returns the currently selected item index, -1 if none is selected.

See Also

SetGadgetState() , GetGadgetItemState() , SetGadgetItemState()

121.41 GetGadgetItemText

Syntax

```
Result\$_ = GetGadgetItemText(#Gadget, Item [, Column])
```

Description

Returns the item text of the specified gadget.

Parameters

#Gadget The gadget to use.

Item The item to get the text. The first item in the gadget has index 0.

Column (optional) The column to use for gadgets that support multiple columns. The first column has index 0. The default is column 0.

Return value

Returns the text of the gadget item or an empty string if there is an error.

Remarks

This is a universal function which works for almost all gadgets which handle several items:

- `ComboBoxGadget()` - 'Item' is the index of the item in the ComboBox list. 'Column' will be ignored.
- `EditorGadget()` - 'Item' is the text line in the EditorGadget. 'Column' will be ignored.
- `ExplorerListGadget()` - returns the name of the 'Item', without the path. If 'Item' = -1, the 'Column' header is returned.
- `ExplorerTreeGadget()` - returns the full path of the 'Item'.
- `ListIconGadget()` - returns the entry of the given 'Item' and 'Column'. If 'Item' = -1, the 'Column' header is returned.
- `ListViewGadget()` - 'Item' is the index of the entry from which you want to receive the content. 'Column' will be ignored.
- `MDIGadget()` - 'Item' is the index of the child window of which you want to get the title.
- `PanelGadget()` - 'Item' is the panel from which you want to receive the header text.
- `TreeGadget()` - 'Item' is the index of the entry from which you want to receive the content. 'Column' will be ignored.
- `WebGadget()` - Get the html code, page title, status message or current selection.

See Also

`SetGadgetItemText()` , `GetGadgetText()` , `SetGadgetText()`

121.42 GetGadgetItemState

Syntax

```
Result = GetGadgetItemState(#Gadget, Item)
```

Description

Returns the item state of the specified gadget.

Parameters

#Gadget The gadget to use.

Item The item to get the state. The first item in the gadget has index 0.

Return value

Returns the state of the gadget item or 0 if there is an error. See below for the meaning of this value depending on the gadget type.

Remarks

This is a universal function that works for almost all gadgets which handle several items:

- ExplorerListGadget() : returns a combination of the following values

```
#PB_Explorer_File      : The item is a file.
#PB_Explorer_Directory : The item is a Directory (or drive).
#PB_Explorer_Selected  : The item is currently selected.
```

- ListViewGadget() : returns 1 if the item is selected, 0 otherwise.

- ListIconGadget() : returns a combination of the following values:

```
#PB_ListIcon_Selected : The item is selected.
#PB_ListIcon_Checked  : The item has its checkbox checked
                      (#PB_ListIcon_CheckBoxes flag).
#PB_ListIcon_Inbetween: The item's checkbox is in the "in
                      between" state (#PB_ListIcon_ThreeState flag).
```

- TreeGadget() : returns a combination of the following values:

```
#PB_Tree_Selected : The item is selected, 0 otherwise.
#PB_Tree_Expanded : The item is expanded (a tree branch opened).
#PB_Tree_Collapsed: The item is collapsed (the tree branch
                      closed).
#PB_Tree_Checked  : The Checkbox of the item is checked
                      (#PB_Tree_CheckBoxes flag).
#PB_Tree_Inbetween: The Checkbox of the item is in the "in
                      between" state (#PB_Tree_ThreeState flag)
```

Check for these states like this:

```
1  If Result & #PB_Tree_Checked
2      ; Item is checked
3  EndIf
```

Example

Below an example with the ListIconGadget() , how you can check for a combination of several results:

```
1  ; ... here a code snippet from a WaitWindowEvent() - event loop:
2  If GetGadgetItemState(#Listicon, n) & #PB_ListIcon_Checked
3      ; Item n is checked (no matter if selected)
4  EndIf
5
6  If GetGadgetItemState(#Listicon, n) & #PB_ListIcon_Selected
7      ; Item n is selected (no matter if checked)
8  EndIf
9
10 If GetGadgetItemState(#Listicon, n) = #PB_ListIcon_Checked |
    #PB_ListIcon_Selected
11     ; Item n is checked AND selected
12 EndIf
13
14 If GetGadgetItemState(#Listicon, n) & (#PB_ListIcon_Checked |
    #PB_ListIcon_Selected)
15     ; Item n is checked OR selected OR both
16 EndIf
```

See Also

SetGadgetItemState() , GetGadgetState() , SetGadgetState()

121.43 GetGadgetText

Syntax

```
String\$ = GetGadgetText(#Gadget)
```

Description

Returns the gadget text content of the specified gadget.

Parameters

#Gadget The gadget to use.

Return value

Returns the gadget text, or an empty string if the gadget does not support text content.

Remarks

This function is especially useful for:

- ButtonGadget() : return the text of the ButtonGadget.
- ComboBoxGadget() - return the content of the current item.
- DateGadget() - return the currently displayed date, in the form it is displayed in the gadget.
- EditorGadget() - return the text content of the editor gadget. Please note, that several lines of text are separated with "Chr(13)+Chr(10)" on Windows and "Chr(10)" on Linux and OS X.
- ExplorerComboGadget() - return the currently selected and displayed directory.
- ExplorerListGadget() - return the currently displayed directory.
- ExplorerTreeGadget() - return the full path of the currently selected file/directory.
- FrameGadget() - return the title of the FrameGadget.
- HyperLinkGadget() - return the text of the HyperLinkGadget.
- ListIconGadget() - return the content of first column of the currently selected item.
- ListViewGadget() - return the content of the current item.
- StringGadget() - return contents of the StringGadget.
- TextGadget() - return contents of the TextGadget.
- TreeGadget() - return the text of the currently selected Item in the TreeGadget.
- WebGadget() - return the URL of the displayed website.

See Also

SetGadgetText() , GetGadgetItemText() , SetGadgetItemText()

121.44 HideGadget

Syntax

```
HideGadget(#Gadget, State)
```

Description

Hide or show a gadget.

Parameters

#Gadget The gadget to use.

State The new state of the gadget. If State = 1, the gadget will be hidden, if State = 0 it will be shown.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 180, 120, "HideGadget",  
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
2  ButtonGadget(0, 10, 10, 160, 50, "Button 1") : button =  
    #True ; Button is displayed  
3  ButtonGadget(1, 10, 80, 160, 30, "Hide Button 1")  
4  Repeat  
5      Event = WaitWindowEvent()  
6      If Event = #PB_Event_Gadget  
7          If EventGadget() = 1  
8              If button = #True ; ButtonGadget is displayed  
9                  HideGadget(0, 1) ; => hide it  
10                 button = #False  
11                 SetGadgetText(1, "Show Button 1")  
12             Else ; ButtonGadget is hidden  
13                 HideGadget(0, 0) ; => show it  
14                 button = #True  
15                 SetGadgetText(1, "Hide Button 1")  
16             EndIf  
17         EndIf  
18     EndIf  
19     Until Event = #PB_Event_CloseWindow  
20 EndIf
```

See Also

DisableGadget()

121.45 HyperLinkGadget

Syntax

```
Result = HyperLinkGadget(#Gadget, x, y, Width, Height, Text$, Color  
    [, Flags])
```

Description

Creates an HyperLink gadget in the current GadgetList. A hyperlink gadget is a text area which reacts to the mouse pointer by changing its color and the cursor shape.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Text\$ The text to display on the link.

Color The color for the text when the mouse is over the gadget. The text color for the non-highlighted state can be changed with `SetGadgetColor()` .

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

#PB_Hyperlink_Underline: Draw a line under the text without the need to use an underlined font.

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

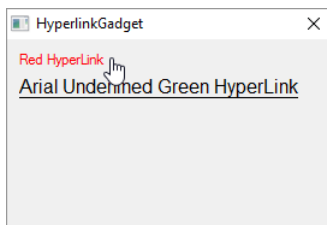
Remarks

A 'mini help' can be added to this gadget using `GadgetToolTip()` .

This gadget supports the `SetGadgetColor()` and `GetGadgetColor()` functions with the **#PB_Gadget_FrontColor** and **#PB_Gadget_BackColor** types to change the color of the text and background (if the window's background have changed).

Example

```
1  If OpenWindow(0, 0, 0, 270, 160, "HyperlinkGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      HyperLinkGadget(0, 10, 10, 250,20,"Red HyperLink", RGB(255,0,0))
3      HyperLinkGadget(1, 10, 30, 250,40,"Arial Underlined Green
    HyperLink", RGB(0,255,0), #PB_Hyperlink_Underline)
4      SetGadgetFont(1, LoadFont(0, "Arial", 12))
5      Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
6      EndIf
```



See Also

`GetGadgetText()` , `SetGadgetText()` , `GetGadgetColor()` , `SetGadgetColor()`

121.46 ImageGadget

Syntax

```
Result = ImageGadget(#Gadget, x, y, Width, Height, ImageID [,  
    Flags])
```

Description

Creates an Image gadget in the current GadgetList.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.
The gadget adjusts its width and height to fit the displayed image. The specified width and height are only used when no image is displayed.

ImageID The image to display. Use the ImageID() function to get the ID from an image. If this parameter is 0, then no image will be displayed.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

```
#PB_Image_Border: display a sunken border around the image.  
#PB_Image_Raised: display a raised border around the image.
```

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using GadgetToolTip() .

- SetGadgetState() : Change the current Image of the gadget. A valid ImageID can be easily obtained with the ImageID() function. If the ImageID is 0, then the image is removed from the gadget.

The following events are supported through EventType() :

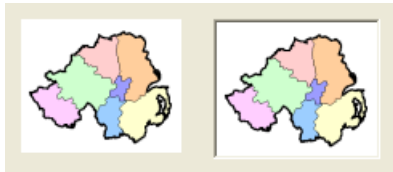
```
#PB_EventType_LeftClick  
#PB_EventType_RightClick  
#PB_EventType_LeftDoubleClick  
#PB_EventType_RightDoubleClick  
#PB_EventType_DragStart
```

After a **#PB_EventType_DragStart** event, the Drag & Drop library can be used to start a Drag & Drop operation. On MacOS X the **#PB_EventType_DragStart** event has to be handled using BindEvent() or BindGadgetEvent() .

If the support of more event types or double-buffering for regularly updated image contents is needed, then take a look at the CanvasGadget() .

Example

```
1  If OpenWindow(0, 0, 0, 245, 105, "ImageGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2  If LoadImage(0, #PB_Compiler_Home +
    "examples/sources/Data/Map.bmp")
3      ImageGadget(0, 10, 10, 100, 83, ImageID(0))
                                     ; imagegadget standard
4      ImageGadget(1, 130, 10, 100, 83, ImageID(0),
    #PB_Image_Border) ; imagegadget with border
5  EndIf
6  Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
7  EndIf
```



See Also

GetGadgetState() , SetGadgetState() , ButtonImageGadget() , ImageID() , CanvasGadget()

121.47 IPAddressGadget

Syntax

```
Result = IPAddressGadget(#Gadget, x, y, Width, Height)
```

Description

Creates an IPAddress gadget in the current GadgetList. It allows you to easily enter a full IPv4 address.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using GadgetToolTip() .

The following functions can be used to act on this gadget:

- GetGadgetState() : Returns the current IP address (Use IPAddressField() to get the value of each field).
- SetGadgetState() : Changes the current IP address (Use MakeIPAddress() to build a valid IP address).

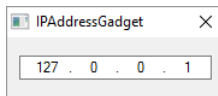
- GetGadgetText() : Returns the current IP address as text, in decimal dotted form ("127.0.0.1", for example).
- SetGadgetText() : Only used to clear the IP address contents, by passing an empty string.

Example

```

1  If OpenWindow(0, 0, 0, 180, 50, "IPAddressGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      IPAddressGadget(0, 10, 15, 160, 20)
3      SetGadgetState(0, MakeIPAddress(127, 0, 0, 1))    ; set a valid
    ip address
4      Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
5  EndIf

```



See Also

GetGadgetState() , SetGadgetState() , GetGadgetText() , SetGadgetText() , IPAddressField() , IPString() , MakeIPAddress()

121.48 IsGadget

Syntax

```
Result = IsGadget(#Gadget)
```

Description

Tests if the given gadget number is a valid and correctly initialized gadget.

Parameters

#Gadget The gadget to use.

Return value

Returns nonzero if the input is a valid gadget and zero otherwise.

Remarks

This function is bulletproof and can be used with any value. This is the correct way to ensure a gadget is ready to use.

See Also

FreeGadget()

121.49 ListIconGadget

Syntax

```
Result = ListIconGadget(#Gadget, x, y, Width, Height,  
    FirstColumnTitle$, FirstColumnWidth [, Flags])
```

Description

Creates a ListIcon gadget in the current GadgetList.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

FirstColumnTitle\$ The title for the first column in the gadget. The gadget is created with one initial column.

FirstColumnWidth The width of the first column in the gadget.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

#PB_ListIcon_CheckBoxes	: Display checkboxes in the first column.
#PB_ListIcon_ThreeState	: The checkboxes can have an "in between" state.
#PB_ListIcon_MultiSelect	: Enable multiple selection.
#PB_ListIcon_GridLines	: Display separator lines between rows and columns.
#PB_ListIcon_FullRowSelect	: The selection covers the full row instead of the first column (Windows and QT only).
#PB_ListIcon_HeaderDragDrop	: The order of columns can be changed using drag'n'drop.
#PB_ListIcon_AlwaysShowSelection	: The selection is still visible, even when the gadget is not activated (Windows only).

The **#PB_ListIcon_ThreeState** flag can be used in combination with the **#PB_ListIcon_CheckBoxes** flag to get checkboxes that can have an "on", "off" and "in between" state. The user can only select the "on" or "off" states. The "in between" state can be set programmatically using the `SetGadgetItemState()` function.

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using `GadgetToolTip()` .

The following functions can be used to act on the list content:

- `AddGadgetColumn()` : Add a column to the gadget.
- `RemoveGadgetColumn()` : Remove a column from the gadget.
- `AddGadgetItem()` : Add an item (with an optional image in the standard 16x16 icon size).
- `RemoveGadgetItem()` : Remove an item.
- `ClearGadgetItems()` : Remove all the items.

- CountGadgetItems() : Returns the number of items currently in the #Gadget.
- GetGadgetItemColor() : Returns front or backcolor of the item.
- SetGadgetItemColor() : Changes front or backcolor of the item (backcolor not supported on MacOS X).
- GetGadgetItemData() : Returns the value that was stored with item.
- SetGadgetItemData() : Stores a value with the item.
- GetGadgetItemState() : Returns the current state of the specified item.
- SetGadgetItemState() : Changes the current state of the specified item.
- GetGadgetItemText() : Returns the current text of the specified item. (or column header, if item = -1)
- SetGadgetItemText() : Changes the current text of the specified item. (or column header, if item = -1). Like with AddGadgetItem() , it is possible to set the text for several columns at once, with the Chr(10) separator.
- SetGadgetItemImage() : Changes the current image of the specified item.
- GetGadgetState() : Returns the first selected item or -1 if there is no item selected.
- SetGadgetState() : Change the selected item (all other selected items will be deselected). If -1 is specified, all selection will be removed.
- GetGadgetAttribute() with the following attribute:

```
#PB_ListIcon_ColumnCount : Returns the number of columns in the
    gadget
#PB_ListIcon_DisplayMode : Returns the current display mode of
    the gadget (Windows only)
#PB_ListIcon_ClickedColumn: Returns the last clicked column
    index. The first column has index 0.
```

- SetGadgetAttribute() with the following attribute:

```
#PB_ListIcon_DisplayMode : Changes the display of the gadget
    (Windows only). It can be one of the following constants
    (Windows only):
    #PB_ListIcon_LargeIcon: Large icon mode
    #PB_ListIcon_SmallIcon: Small icon mode
    #PB_ListIcon_List      : List icon mode
    #PB_ListIcon_Report    : Report mode (columns, default mode)
```

- GetGadgetItemAttribute() : With the following attributes:

```
#PB_ListIcon_ColumnWidth: Returns the width of the given
    'Column'. The 'Item' parameter is ignored.
```

SetGadgetItemAttribute() : With the following attribute:

```
#PB_ListIcon_ColumnWidth : Changes the width of the given
    'Column'. The 'Item' parameter is ignored.
#PB_ListIcon_ColumnAlignment: Changes the text alignment of the
    given 'Column'. The 'Value' parameter can be one of the
    following value:
    - #PB_ListIcon_Left : Text is left aligned (default).
    - #PB_ListIcon_Right : Text is right aligned.
    - #PB_ListIcon_Center: Text is centered.
```

This gadget supports the SetGadgetColor() and GetGadgetColor() functions with the following values as 'ColorType':

```
#PB_Gadget_FrontColor: Textcolor
#PB_Gadget_BackColor : Backgroundcolor
#PB_Gadget_LineColor : Color for the gridlines if the
    #PB_ListIcon_GridLines flag is used.
```

The following events are supported through EventType() :

`#PB_EventType_LeftClick`: left click on an item, or a checkbox was checked/unchecked
`#PB_EventType_LeftDoubleClick`
`#PB_EventType_RightClick`
`#PB_EventType_RightDoubleClick`
`#PB_EventType_Change`: the current item changed
`#PB_EventType_DragStart`: the user tried to start a Drag & Drop operation. `ExamineDraggedItems()` can be used to get the dragged items information.
`#PB_EventType_ColumnClick`: a column title has been clicked. Use `GetGadgetAttribute()` with `#PB_ListIcon_ClickedColumn` to know which one has been clicked.

After a `#PB_EventType_DragStart` event, the Drag & Drop library can be used to start a Drag & Drop operation. On MacOS X the `#PB_EventType_DragStart` event has to be handled using `BindEvent()` or `BindGadgetEvent()`.

Example

```

1  If OpenWindow(0, 100, 100, 300, 100, "ListIcon Example",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2  ListIconGadget(0, 5, 5, 290, 90, "Name", 100,
    #PB_ListIcon_FullRowSelect | #PB_ListIcon_AlwaysShowSelection)
3  AddGadgetColumn(0, 1, "Address", 250)
4  AddGadgetItem(0, -1, "Harry Rannit"+Chr(10)+"12 Parliament Way,
    Battle Street, By the Bay")
5  AddGadgetItem(0, -1, "Ginger Brokeit"+Chr(10)+"130 PureBasic
    Road, BigTown, CodeCity")
6  Repeat
7    Event = WaitWindowEvent()
8  Until Event = #PB_Event_CloseWindow
9  EndIf
  
```

Example

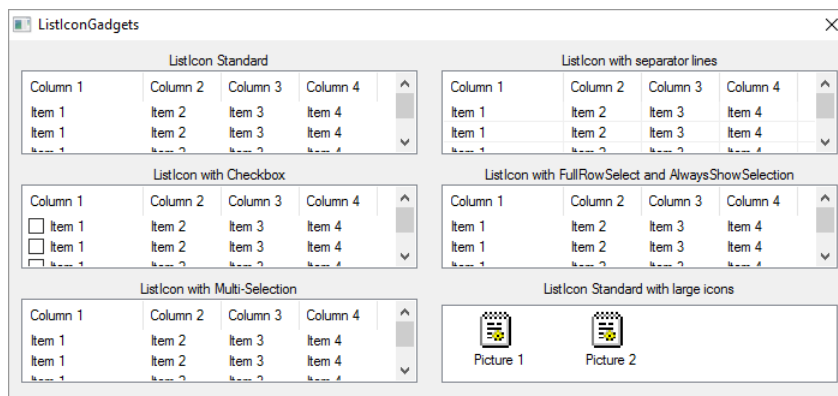
```

1  ; Shows possible flags of ListIconGadget in action...
2  If OpenWindow(0, 0, 0, 700, 300, "ListIconGadgets",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3    ; left column
4    TextGadget(6, 10, 10, 330, 20, "ListIcon Standard",
    #PB_Text_Center)
5    ListIconGadget(0, 10, 25, 330, 70, "Column 1", 100)
6    TextGadget(7, 10, 105, 330, 20, "ListIcon with Checkbox",
    #PB_Text_Center)
7    ListIconGadget(1, 10, 120, 330, 70, "Column 1", 100,
    #PB_ListIcon_CheckBoxes) ; ListIcon with checkbox
8    TextGadget(8, 10, 200, 330, 20, "ListIcon with
    Multi-Selection", #PB_Text_Center)
9    ListIconGadget(2, 10, 215, 330, 70, "Column 1", 100,
    #PB_ListIcon_MultiSelect) ; ListIcon with multi-selection
10   ; right column
11   TextGadget(9, 360, 10, 330, 20, "ListIcon with separator
    lines", #PB_Text_Center)
12   ListIconGadget(3, 360, 25, 330, 70, "Column 1", 100,
    #PB_ListIcon_GridLines)
13   TextGadget(10, 360, 105, 330, 20, "ListIcon with
    FullRowSelect and AlwaysShowSelection", #PB_Text_Center)
  
```

```

14     ListIconGadget(4, 360, 120, 330, 70, "Column 1", 100,
#PB_ListIcon_FullRowSelect | #PB_ListIcon_AlwaysShowSelection)
15     TextGadget (11, 360, 200, 330, 20, "ListIcon Standard with
large icons", #PB_Text_Center)
16     ListIconGadget(5, 360, 220, 330, 65, "",
200, #PB_ListIcon_GridLines)
17     For a = 0 To 4 ; add columns to each of the first 5
listicons
18         For b = 2 To 4 ; add 3 more columns to each listicon
19             AddGadgetColumn(a, b, "Column " + Str(b), 65)
20         Next
21         For b = 0 To 2 ; add 4 items to each line of the
listicons
22             AddGadgetItem(a, b, "Item 1"+Chr(10)+"Item 2"+Chr(10)+"Item
3"+Chr(10)+"Item 4")
23         Next
24     Next
25     ; Here we change the ListIcon display to large icons and show
an image
26     If LoadImage(0,
#PB_Compiler_Home+"examples/sources/Data/File.bmp") ; change
path/filename to your own 32x32 pixel image
27         SetGadgetAttribute(5, #PB_ListIcon_DisplayMode,
#PB_ListIcon_LargeIcon)
28         AddGadgetItem(5, 0, "Picture 1", ImageID(0))
29         AddGadgetItem(5, 1, "Picture 2", ImageID(0))
30     EndIf
31     Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
32 EndIf

```



See Also

AddGadgetColumn(), RemoveGadgetColumn(), AddGadgetItem(), RemoveGadgetItem(), ClearGadgetItems(), CountGadgetItems(), GetGadgetState(), SetGadgetState(), GetGadgetAttribute(), SetGadgetAttribute(), GetGadgetItemText(), SetGadgetItemText(), SetGadgetItemImage(), GetGadgetItemImage(), SetGadgetItemImage(), GetGadgetItemData(), SetGadgetItemData(), GetGadgetItemAttribute(), SetGadgetItemAttribute(), GetGadgetColor(), SetGadgetColor(), GetGadgetItemColor(), SetGadgetItemColor(), ExplorerListGadget(), ListViewGadget()

121.50 ListViewGadget

Syntax

```
Result = ListViewGadget(#Gadget, x, y, Width, Height [, Flags])
```

Description

Creates a ListView gadget in the current GadgetList.

Parameters

#Gadget A number to identify the new gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

```
#PB_ListView_Multiselect: allows multiple items to be selected
#PB_ListView_ClickSelect: allows multiple items to be
                          selected. clicking on one item selects/deselects it (on OS
                          X, same behaviour as #PB_ListView_Multiselect)
```

Return value

Returns nonzero on success and zero on failure. If #PB_Any was used as the #Gadget parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using GadgetToolTip() .

The following functions can be used to act on the list content:

- AddGadgetItem() : Add an item. A ListViewGadget() is limited to 65536 items.
- RemoveGadgetItem() : Remove an item.
- ClearGadgetItems() : Remove all the items
- CountGadgetItems() : Returns the number of items currently in the #Gadget.
- GetGadgetItemData() : Get the value that was stored with the gadget item.
- GetGadgetItemState() : Returns nonzero if the item is selected, zero otherwise.
- GetGadgetItemText() : Get the content of the given item.
- GetGadgetState() : Get the index of the selected item or -1 if there is no selected item.
- GetGadgetText() : Get the content of the selected item.
- SetGadgetItemData() : store a value with the given item.
- SetGadgetItemState() : Selects or deselects the given item.
- SetGadgetItemText() : Set the text of the given item.
- SetGadgetState() : Change the selected item. If -1 is specified, all selections will be removed.
- SetGadgetText() : Selects the item with the given text (the text must match exactly).

This gadget supports the SetGadgetColor() and GetGadgetColor() functions with the following values as 'ColorType':

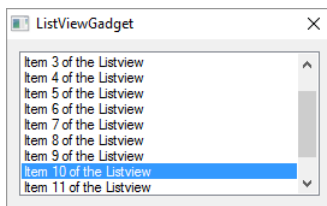
```
#PB_Gadget_FrontColor: Textcolor
#PB_Gadget_BackColor : Backgroundcolor
```

The following events are supported through EventType() :

```
#PB_EventType_LeftClick (also fired when the selection change)
#PB_EventType_LeftDoubleClick
#PB_EventType_RightClick
```

Example

```
1  If OpenWindow(0, 0, 0, 270, 140, "ListViewGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      ListViewGadget(0, 10, 10, 250, 120)
3      For a = 1 To 12
4          AddGadgetItem (0, -1, "Item " + Str(a) + " of the Listview")
        ; define listview content
5      Next
6      SetGadgetState(0, 9) ; set (beginning with 0) the tenth item as
        the active one
7      Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
8      EndIf
```



See Also

AddGadgetItem() , RemoveGadgetItem() , ClearGadgetItems() , CountGadgetItems() ,
GetGadgetState() , SetGadgetState() , GetGadgetText() , SetGadgetText() ,
GetGadgetItemState() , SetGadgetItemState() , GetGadgetItemText() , SetGadgetItemText() ,
GetGadgetItemData() , SetGadgetItemData() , GetGadgetColor() , SetGadgetColor() ,
ListItemIconGadget()

121.51 MDIGadget

Syntax

```
Result = MDIGadget(#Gadget, x, y, Width, Height, SubMenu, MenuItem  
[, Flags])
```

Description

Creates a client area, in which child windows can be displayed. These child windows are fully movable and sizable by the user in this area.

Parameters

#Gadget A number to identify the new gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

SubMenu The menu index to which the MDI window items should be added.

An MDIGadget() is always connected to a window Menu (see CreateMenu()). Therefore a MDIGadget() can only be put on a window that has a menu attached to it. The gadget will give the user the opportunity to select the child windows through one of the windows submenus. In the 'SubMenu' parameter, you have to specify the submenu index (created with MenuItem()) where these items will be attached to (the first submenu has the index 0). The Gadget will add a separator at the end of this menu, and then add one item for each currently displayed child window.

MenuItem The first menu item index to use for MDI windows.

The gadget will require a set of Menu item identifiers (see the 'MenuID' parameter of MenuItem()) to add these menuitems. In the 'MenuItem' parameter of MDIGadget() you have to specify the lowest number that the gadget can use for that purpose. It will use numbers above that, when new child windows are added, so you need to reserve at least as much numbers as you plan to add items. It is recommended to use a number above all menu identifiers of your program, to make sure there is never a collision.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

```
#PB_MDI_AutoSize      : The gadget will automatically resize
                        itself to fit the parent window.
                        If you have no other gadgets on the
                        window, this is a helpful option.
#PB_MDI_BorderLess    : There will be no border drawn around
                        the client area.
#PB_MDI_NoScrollBars  : When the user drags a childwindow
                        outside of the displayed area, there will be no scrollbars.
```

Return value

Returns nonzero on success and zero on failure. If #PB_Any was used as the #Gadget parameter then the return-value is the auto-generated gadget number on success.

Remarks

Because of the connection with the window menu, there can only be one MDIGadget() on a window, however you can put another one in a second window if you wish. You can only put this gadget directly on a window, you can NOT put it inside a ContainerGadget() , SplitterGadget() or PanelGadget() .

As the whole point of this gadget is to dynamically display data, it is recommended to use the #PB_Any feature to populate a child window with gadgets.

When using AddGadgetItem() with this gadget, there is actually a new PB window created. All the functions of the Window library can be used with this new window (except StickyWindow()). Of course the number chosen for the new window may not overlap with another open window, otherwise the other window will be closed. The MDI Gadget does not return any events. Events concerning the childwindows will be received as normal window events (#PB_Event_SizeWindow, #PB_Event_CloseWindow, ...) instead.

A 'mini help' can be added to this gadget using GadgetToolTip() .

The following functions can be used to manage the gadget contents:

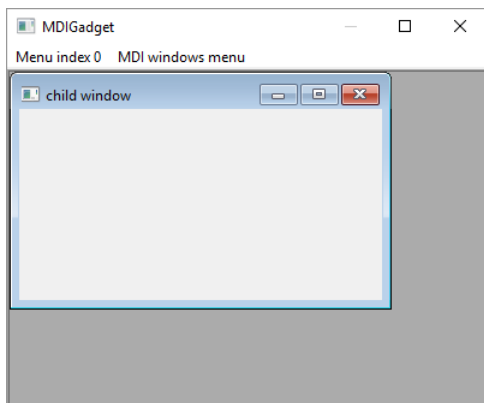
- CountGadgetItems() : Return the number of child windows.
- AddGadgetItem() : Add a new child window to the gadget.
- ClearGadgetItems() : Close all child windows.
- GetGadgetState() : Get the currently focused child window.
- SetGadgetState() : Set the currently focused window, or arrange the child windows. (see GetGadgetState() for more info.)
- SetGadgetAttribute() : With the following attributes:

```
#PB_MDI_Image      : Set a background image. An ImageID value must
                    be passed as value. (see ImageID()
)
#PB_MDI_TileImage  : Set the tile mode. 0 draws the image just once
                    in the top/left corner, 1 repeats the image to fill the whole
                    area
```

This gadget supports the SetGadgetColor() and GetGadgetColor functions with the #PB_Gadget_BackColor type to change the background color of the MDI area.

Example

```
1  #Main = 0
2  #MDIChild = 1
3  If OpenWindow(#Main, 0, 0, 400, 300, "MDIGadget",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered |
   #PB_Window_SizeGadget | #PB_Window_MaximizeGadget)
4      If CreateMenu(#Main, WindowID(#Main))
5          MenuTitle("Menu index 0")
6          MenuTitle("MDI windows menu")
7              MenuItem(0, "self created item")
8              MenuItem(1, "self created item")
9
10         MDIGadget(0, 0, 0, 0, 0, 1, 2, #PB_MDI_AutoSize)
11         AddGadgetItem(0, #MDIChild, "child window")
12             ; add gadgets here...
13         UseGadgetList(WindowID(#Main)) ; go back to the main window
   gadgetlist
14     EndIf
15     Repeat : Until WaitWindowEvent()=#PB_Event_CloseWindow
16 EndIf
```



See Also

AddGadgetItem(), CloseWindow(), CountGadgetItems(), ClearGadgetItems(),
GetGadgetState(), SetGadgetState(), GetGadgetAttribute(), SetGadgetAttribute(),
GetGadgetColor(), SetGadgetColor(), Window library

Supported OS

Windows

121.52 OpenGadgetList

Syntax

```
OpenGadgetList(#Gadget [, Item])
```

Description

Use the specified gadget as a GadgetList, to dynamically add new gadgets to it.

Parameters

#Gadget The gadget in which new gadgets should be created.

Item (optional) For the PanelGadget() : Specifies the panel to which the gadgets should be added. To add a new panel dynamically to the PanelGadget() the 'Item' parameter must be omitted.

Return value

None.

Remarks

The following gadgets are supported by OpenGadgetList():

- ContainerGadget()
- PanelGadget()
- ScrollAreaGadget()

Once the all the needed changes are done, CloseGadgetList() should be called.

See Also

CloseGadgetList() , ContainerGadget() , PanelGadget() , ScrollAreaGadget()

121.53 OptionGadget

Syntax

```
Result = OptionGadget(#Gadget, x, y, Width, Height, Text$)
```

Description

Creates an OptionGadget in the current GadgetList.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Text\$ The text to display.

Return value

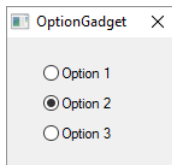
Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

Remarks

The first time this function is called, a group is created and all following calls of OptionGadget() will add a gadget to this group. To finish the group, just create a gadget of another type. These kind of gadgets are very useful as only one gadget from the group can be selected at any time. A 'mini help' can be added to this gadget using GadgetToolTip() .

Example

```
1  If OpenWindow(0, 0, 0, 140, 110, "OptionGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      OptionGadget(0, 30, 20, 60, 20, "Option 1")
3      OptionGadget(1, 30, 45, 60, 20, "Option 2")
4      OptionGadget(2, 30, 70, 60, 20, "Option 3")
5      SetGadgetState(1, 1) ; set second option as active one
6      Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
7  EndIf
```



See Also

GetGadgetText() , SetGadgetText() , GetGadgetState() , SetGadgetState() , CheckBoxGadget()

121.54 PanelGadget

Syntax

```
Result = PanelGadget(#Gadget, x, y, Width, Height)
```

Description

Creates a Panel gadget in the current GadgetList.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using GadgetToolTip() .

The following functions can be used to act on the panel content:

- AddGadgetItem() : Add a panel.
- RemoveGadgetItem() : Remove a panel.
- CountGadgetItems() : Count the number of panels.
- ClearGadgetItems() : Remove all panels.
- GetGadgetItemText() : Retrieve the text of the specified item.
- SetGadgetItemText() : Change the text of the specified item.
- SetGadgetItemImage() : Change the image of the specified item. (not supported on OS X)
- GetGadgetItemData() : Retrieve the value associated with the specified item.

- SetGadgetItemData() : Associate a value with the specified item.
- GadgetItemID() : Return the OS handle of the specified item (Windows only, useful for API functions).
- SetGadgetState() : Change the active panel.
- GetGadgetState() : Get the index of the current panel.
- GetGadgetAttribute() : With one of the following attributes: (there must be at least 1 item for this to work) (not supported on Linux GTK)

`#PB_Panel_ItemWidth` : Returns the width of the inner area where the gadgets are displayed.
`#PB_Panel_ItemHeight` : Returns the height of the inner area where the gadgets are displayed.
`#PB_Panel_TabHeight` : Returns height of the panel tabs on top of the gadget.

The following events are supported through EventType() :

`#PB_EventType_Change` : The current displayed tab has been changed.
`#PB_EventType_Resize` : The gadget has been resized.

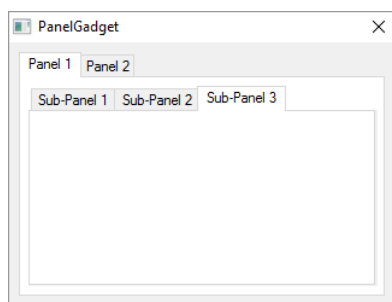
Once a Panel is created, its list of panels is empty. You must call AddGadgetItem() to add a panel before you can create other gadgets inside the panel gadget. The next gadgets will then be created automatically in the new panel. This is very convenient. When the PanelGadget item has been filled with all the needed gadgets, CloseGadgetList() must be called to return to the previous GadgetList. This means that a PanelGadget can be created inside another PanelGadget...

Example

```

1 ; Shows using of several panels...
2 If OpenWindow(0, 0, 0, 322, 220, "PanelGadget",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     PanelGadget (0, 8, 8, 306, 203)
4     AddGadgetItem (0, -1, "Panel 1")
5     PanelGadget (1, 5, 5, 290, 166)
6     AddGadgetItem(1, -1, "Sub-Panel 1")
7     AddGadgetItem(1, -1, "Sub-Panel 2")
8     AddGadgetItem(1, -1, "Sub-Panel 3")
9     CloseGadgetList()
10    AddGadgetItem (0, -1, "Panel 2")
11    ButtonGadget(2, 10, 15, 80, 24, "Button 1")
12    ButtonGadget(3, 95, 15, 80, 24, "Button 2")
13    CloseGadgetList()
14    Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
15 EndIf

```



See Also

AddGadgetItem() , RemoveGadgetItem() , CountGadgetItems() , ClearGadgetItems() ,
 GetGadgetItemText() , SetGadgetItemText() , GetGadgetState() , SetGadgetState() ,

GetGadgetAttribute() , CloseGadgetList() , OpenGadgetList() , SetGadgetItemImage()

121.55 ProgressBarGadget

Syntax

```
Result = ProgressBarGadget(#Gadget, x, y, Width, Height, Minimum,
    Maximum [, Flags])
```

Description

Creates a ProgressBar gadget in the current GadgetList.

Parameters

#Gadget A number to identify the new gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Minimum, Maximum The minimum and maximum values that the progress bar can take. The values must be between 0 and 65536 to work on all OS.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

```
#PB_ProgressBar_Smooth    : The progress bar is smooth instead
    of using blocks
                                (Note: On Windows with enabled
    skins and on OS X, this flag has no effect).
#PB_ProgressBar_Vertical  : The progress bar is in vertical
    mode.
```

Return value

Returns nonzero on success and zero on failure. If #PB_Any was used as the #Gadget parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using GadgetToolTip() .

The following functions can be used to act on the gadget:

- SetGadgetState() : Change progress bar value. A value of #PB_ProgressBar_Unknown can be used to indicate that the progress is unknown.

- GetGadgetState() : Get the current progress bar value.

- SetGadgetAttribute() : With the following attributes:

```
#PB_ProgressBar_Minimum    : Changes the minimum value.
#PB_ProgressBar_Maximum    : Changes the maximum value.
```

- GetGadgetAttribute() : With the following attributes:

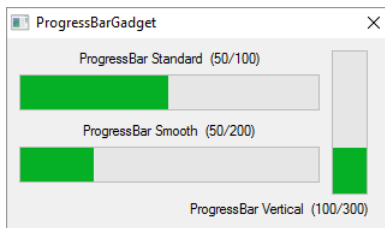
```
#PB_ProgressBar_Minimum    : Returns the minimum value.
#PB_ProgressBar_Maximum    : Returns the maximum value.
```

This gadget supports the SetGadgetColor() and GetGadgetColor() functions with the following values as 'ColorType'. (Note: Windows only. On Windows with enabled skins, this coloring has no effect.)

```
#PB_Gadget_FrontColor: Progressbarcolor
#PB_Gadget_BackColor : Backgroundcolor
```

Example

```
1  If OpenWindow(0, 0, 0, 320, 160, "ProgressBarGadget",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      TextGadget      (3, 10, 10, 250, 20, "ProgressBar Standard
   (50/100)", #PB_Text_Center)
3      ProgressBarGadget(0, 10, 30, 250, 30, 0, 100)
4      SetGadgetState   (0, 50)    ; set 1st progressbar (ID = 0) to
   50 of 100
5      TextGadget      (4, 10, 70, 250, 20, "ProgressBar Smooth
   (50/200)", #PB_Text_Center)
6      ProgressBarGadget(1, 10, 90, 250, 30, 0, 200,
   #PB_ProgressBar_Smooth)
7      SetGadgetState   (1, 50)    ; set 2nd progressbar (ID = 1) to
   50 of 200
8      TextGadget      (5, 100, 135, 200, 20, "ProgressBar Vertical
   (100/300)", #PB_Text_Right)
9      ProgressBarGadget(2, 270, 10, 30, 120, 0, 300,
   #PB_ProgressBar_Vertical)
10     SetGadgetState   (2, 100)    ; set 3rd progressbar (ID = 2) to
   100 of 300
11     Repeat : Until WaitWindowEvent()=#PB_Event_CloseWindow
12 EndIf
```



See Also

GetGadgetState() , SetGadgetState() , GetGadgetAttribute() , SetGadgetAttribute() ,
GetGadgetColor() , SetGadgetColor()

121.56 RemoveGadgetColumn

Syntax

```
RemoveGadgetColumn(#Gadget , Column)
```

Description

Removes a column of the specified gadget.

Parameters

#Gadget The gadget to use.

Column The column to remove (the first column has index 0). **#PB_All** can be used to remove all the columns.

Return value

None.

Remarks

The gadget type can be one of the following:

- ListIconGadget()
- ExplorerListGadget()

Example

```
1  If OpenWindow(0, 0, 0, 320, 160, "RemoveGadgetColumn",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3
4      ListIconGadget(0, 10, 10, 300, 140, "Hello", 100)
5      AddGadgetColumn(0, 1, "Column 2", 70)
6      AddGadgetColumn(0, 2, "Column 3", 70)
7
8      RemoveGadgetColumn(0, 1) ; Remove the 'Column 2'
9
10     Repeat
11         Until WaitWindowEvent() = #PB_Event_CloseWindow
12     EndIf
```

See Also

AddGadgetColumn() , ListIconGadget() , ExplorerListGadget()

121.57 RemoveGadgetItem

Syntax

```
RemoveGadgetItem(#Gadget, Position)
```

Description

Removes an item of the specified gadget.

Parameters

#Gadget The gadget to use.

Position The item to remove. The first item has the index 0.

Return value

None.

Remarks

The gadget type can be one of the following:

- ComboBoxGadget()
- EditorGadget()
- PanelGadget()
- ListViewGadget()
- ListIconGadget()
- MDIGadget()
- TreeGadget() - If the removed item is a node, all child-items will be removed too.

See Also

AddGadgetItem() , ClearGadgetItems() , CountGadgetItems()

121.58 ResizeGadget

Syntax

```
ResizeGadget(#Gadget, x, y, Width, Height)
```

Description

Resize the specified gadget to the given position and dimensions.

Parameters

#Gadget The gadget to resize.

x, y, Width, Height The new position and dimensions of the gadget. To ease the building of real-time resizable Graphical User Interfaces (GUIs), **#PB_Ignore** can be passed as any parameter (x, y, Width or Height) and this parameter will not be changed.

Return value

None.

Example: Resize gadgets according to new window size

```
1  Procedure SizeHandler()
2      ResizeGadget(0, #PB_Ignore, #PB_Ignore, WindowWidth(0) - 20,
3      WindowHeight(0) - 60)
4      ResizeGadget(1, #PB_Ignore, WindowHeight(0) - 40,
5      WindowWidth(0) - 20, #PB_Ignore)
6  EndProcedure
7
8  If OpenWindow(0, 0, 0, 220, 100, "Resize the window",
9  #PB_Window_SystemMenu | #PB_Window_ScreenCentered |
10 #PB_Window_SizeGadget)
11      WindowBounds(0, WindowWidth(0), WindowHeight(0), #PB_Ignore,
12      #PB_Ignore)
13
14      EditorGadget(0, 10, 10, WindowWidth(0) - 20, WindowHeight(0) -
15      60)
16      ButtonGadget(1, 10, WindowHeight(0) - 40, WindowWidth(0) - 20,
17      30, "Button")
18
19      ; Use BindEvent() to have a realtime gadget resize
20      BindEvent(#PB_Event_SizeWindow, @SizeHandler())
21
22      Repeat
23          Select WaitWindowEvent()
24              Case #PB_Event_CloseWindow
25                  End
26          EndSelect
27      ForEver
28  EndIf
```

See Also

GadgetX() , GadgetY() , GadgetWidth() , GadgetHeight() , ResizeWindow() , BindEvent()

121.59 ScrollBarGadget

Syntax

```
Result = ScrollBarGadget(#Gadget, x, y, Width, Height, Minimum,  
                          Maximum, PageLength [, Flags])
```

Description

Creates a scrollbar gadget in the current GadgetList.

Parameters

#Gadget A number to identify the new gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Minimum, Maximum The range of values that the scrollbar can take. These values must be between 0 and 10,000 to avoid limitations on some operating systems.

PageLength The amount of values that are part of the current displayed "page".

For example you can have a picture which is 100 pixels width and you only see 25 pixels.

What you see is called a 'page', in this example, the page length will be 25, the Minimum will be 0 and the Maximum will be 100.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

```
#PB_ScrollBar_Vertical : The scrollbar is vertical (instead  
of horizontal, which is the default).
```

Return value

Returns nonzero on success and zero on failure. If #PB_Any was used as the #Gadget parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using GadgetToolTip() .

The following functions can be used to act on this gadget:

- GetGadgetState() : Returns the current slider position (value between 'Minimum' and 'Maximum - PageLength + 1' range).

- SetGadgetState() : Changes the current slider position.

- GetGadgetAttribute() : With one of the following attributes:

```
#PB_ScrollBar_Minimum : Returns the minimum scroll position.  
#PB_ScrollBar_Maximum : Returns the maximum scroll position.  
#PB_ScrollBar_PageLength: Returns the PageLength value.
```

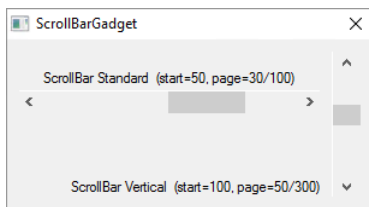
- SetGadgetAttribute() : With one of the following attributes:

```
#PB_ScrollBar_Minimum : Changes the minimum scroll position.  
#PB_ScrollBar_Maximum : Changes the maximum scroll position.  
#PB_ScrollBar_PageLength: Changes the PageLength value.
```

An event is triggered when it's used. See examples below.

Example

```
1  If OpenWindow(0, 0, 0, 305, 140, "ScrollBarGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      TextGadget      (2, 10, 25, 250, 20, "ScrollBar Standard
    (start=50, page=30/100)", #PB_Text_Center)
3      ScrollBarGadget (0, 10, 42, 250, 20, 0, 100, 30)
4      SetGadgetState  (0, 50)    ; set 1st scrollbar (ID = 0) to 50
    of 100
5      TextGadget      (3, 10, 115, 250, 20, "ScrollBar Vertical
    (start=100, page=50/300)", #PB_Text_Right)
6      ScrollBarGadget (1, 270, 10, 25, 120, 0, 300, 50,
    #PB_ScrollBar_Vertical)
7      SetGadgetState  (1, 100)   ; set 2nd scrollbar (ID = 1) to 100
    of 300
8      Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
9      EndIf
```



Example: ScrollBar events

```
1  Procedure BindHScrollDatas()
2      SetWindowTitle(0, "ScrollBarGadget (" + GetGadgetState(0) + ")")
3  EndProcedure
4
5  Procedure BindVScrollDatas()
6      SetWindowTitle(0, "ScrollBarGadget (" + GetGadgetState(1) + ")")
7  EndProcedure
8
9
10 If OpenWindow(0, 0, 0, 400, 400, "ScrollBarGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
11     TextGadget      (2, 10, 25, 350, 30, "ScrollBar Standard
    (start = 50, page = 30/100)")
12     ScrollBarGadget (0, 10, 50, 350, 20, 0, 100, 30)
13     SetGadgetState  (0, 50)    ; set 1st scrollbar (ID = 0) to 50
    of 100
14     TextGadget      (3, 10, 120, 350, 30, "ScrollBar vertical
    (start = 100, page = 50/300)")
15     ScrollBarGadget (1, 175, 160, 25, 120, 0, 300, 50,
    #PB_ScrollBar_Vertical)
16     SetGadgetState  (1, 100)   ; set 2nd scrollbar (ID = 1) to 100
    of 300
17
18     BindGadgetEvent(0, @ BindHScrollDatas())
19     BindGadgetEvent(1, @ BindVScrollDatas())
20
21     Repeat
22         Select WaitWindowEvent()
```



```

23         Case #PB_Event_CloseWindow
24         End
25         Case #PB_Event_Gadget
26         Select EventGadget()
27             Case 0
28                 MessageRequester("Info","The ScrollBar 0 has been
used ! (" + GetGadgetState(0) +
29                                     ")" ,#PB_MessageRequester_Ok)
30             Case 1
31                 MessageRequester("Info","The ScrollBar 1 has been
used ! (" + GetGadgetState(1) +
32                                     ")" ,#PB_MessageRequester_Ok)
33
34         EndSelect
35     EndSelect
36     ForEver
37 EndIf

```

See Also

GetGadgetState() , SetGadgetState() , GetGadgetAttribute() , SetGadgetAttribute() ,
ScrollAreaGadget()

121.60 ScrollAreaGadget

Syntax

```
Result = ScrollAreaGadget(#Gadget, x, y, Width, Height,
    ScrollAreaWidth, ScrollAreaHeight [, ScrollStep [, Flags]])
```

Description

Creates a ScrollArea gadget in the current GadgetList. It is a container for other gadgets with a scrollable area.

Parameters

#Gadget A number to identify the new gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

ScrollAreaWidth, ScrollAreaHeight The dimensions of the scrollable area inside the gadget. These can also be smaller than the outer dimensions, in this case scrolling will be disabled. (MS Windows only: These are limited to 32 000 pixels)

ScrollStep (optional) The amount of pixels to scroll when the user presses the scroll bar arrows.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

```

#PB_ScrollArea_Flat           : Flat frame
#PB_ScrollArea_Raised        : Raised frame
#PB_ScrollArea_Single        : Single sunken frame
#PB_ScrollArea_BorderLess    : Without any border
#PB_ScrollArea_Center        : If the inner size is smaller than
the outer, the inner area is automatically centered.

```

Return value

Returns nonzero on success and zero on failure. If `#PB_Any` was used as the `#Gadget` parameter then the return-value is the auto-generated gadget number on success.

Remarks

Once the gadget is created, all future created gadgets will be created inside the scroll area. When all the needed gadgets have been created, `CloseGadgetList()` must be called to return to the previous `GadgetList`. `OpenGadgetList()` can be used later to add others gadgets on the fly in the scroll area.

The following functions can be used to act on a `ScrollAreaGadget`:

`GetGadgetAttribute()` : With one of the following attribute:

```
#PB_ScrollArea_InnerWidth : Returns the width (in pixels) of the
                           contained scrollable area.
#PB_ScrollArea_InnerHeight : Returns the height (in pixels) of
                           the contained scrollable area.
#PB_ScrollArea_X           : Returns the current horizontal
                           scrolling position (in pixels).
#PB_ScrollArea_Y           : Returns the current vertical
                           scrolling position (in pixels).
#PB_ScrollArea_ScrollStep : Returns the current scroll step
                           value (in pixels).
```

`SetGadgetAttribute()` : With one of the following attribute:

```
#PB_ScrollArea_InnerWidth : Changes the width (in pixels) of the
                           contained scrollable area.
#PB_ScrollArea_InnerHeight : Changes the height (in pixels) of
                           the contained scrollable area.
#PB_ScrollArea_X           : Changes the current horizontal
                           scrolling position (in pixels).
#PB_ScrollArea_Y           : Changes the current vertical
                           scrolling position (in pixels).
#PB_ScrollArea_ScrollStep : Changes the current scroll step
                           value (in pixels).
```

The following event is supported through `EventType()` :

```
#PB_EventType_Resize: The gadget has been resized.
```

This gadget supports the `SetGadgetColor()` and `GetGadgetColor()` functions with the `#PB_Gadget_BackColor` type to change the background color.

An event is triggered when it's used. See example below.

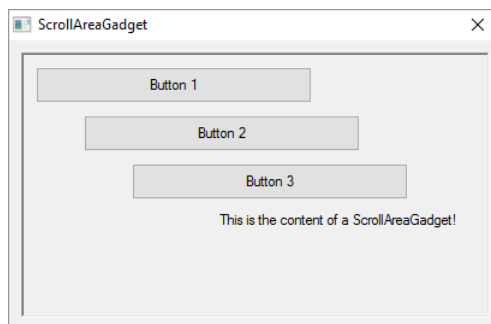
Example

```
1  Procedure BindScrollDatas()
2      SetWindowTitle(0, "ScrollAreaGadget " +
3          "(" +
4              GetGadgetAttribute(0, #PB_ScrollArea_X) +
5              ", " +
6              GetGadgetAttribute(0, #PB_ScrollArea_Y) +
7              ")" )
8  EndProcedure
9
10 If OpenWindow(0, 0, 0, 405, 240, "ScrollAreaGadget",
11     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
12     ScrollAreaGadget(0, 10, 10, 390, 220, 575, 555, 30)
13     ButtonGadget (1, 10, 10, 230, 30, "Button 1")
```

```

13     ButtonGadget (2, 50, 50, 230, 30,"Button 2")
14     ButtonGadget (3, 90, 90, 230, 30,"Button 3")
15     TextGadget (4,130,130, 230, 20,"This is the content of a
ScrollAreaGadget!",#PB_Text_Right)
16     CloseGadgetList()
17
18     BindGadgetEvent(0, @ BindScrollDatas())
19
20     Repeat
21         Select WaitWindowEvent()
22             Case #PB_Event_CloseWindow
23                 End
24             Case #PB_Event_Gadget
25                 Select EventGadget()
26                     Case 0
27                     MessageRequester("Info","A Scroll has been used ! (" +
28                                     GetGadgetAttribute(0,
#PB_ScrollArea_X) +
29                                     "," +
30                                     GetGadgetAttribute(0,
#PB_ScrollArea_Y) +
31                                     ")" ,#PB_MessageRequester_Ok)
32                     Case 1
33                     MessageRequester("Info","Button 1 was
pressed!",#PB_MessageRequester_Ok)
34                     Case 2
35                     MessageRequester("Info","Button 2 was
pressed!",#PB_MessageRequester_Ok)
36                     Case 3
37                     MessageRequester("Info","Button 3 was
pressed!",#PB_MessageRequester_Ok)
38                 EndSelect
39             EndSelect
40         ForEver
41     EndIf

```



See Also

GetGadgetAttribute() , SetGadgetAttribute() , ScrollBarGadget()

121.61 SetActiveGadget

Syntax

```
SetActiveGadget (#Gadget)
```

Description

Activates (sets the keyboard focus on) the gadget specified by the given gadget number. Activating a gadget allows it to become the current object to receive messages and handle keystrokes.

Parameters

#Gadget The gadget to activate. Can be set to '-1' to remove the keyboard focus from the current active window.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 270, 140, "SetActiveGadget",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      StringGadget (0, 10, 10, 250, 20, "bla bla...")
4      ComboBoxGadget(1, 10, 40, 250, 21)
5      For a = 1 To 5 : AddGadgetItem(1, -1, "ComboBox item " +
6          Str(a)) : Next
7      SetGadgetState(1, 2) ; set (beginning with 0)
8      the third item as active one
9      ButtonGadget (2, 10, 90, 250, 20, "Activate StringGadget")
10     ButtonGadget (3, 10, 115, 250, 20, "Activate ComboBox")
11     Repeat
12         Event = WaitWindowEvent()
13         If Event = #PB_Event_Gadget
14             Select EventGadget()
15                 Case 2 : SetActiveGadget(0) ; Activate StringGadget
16                 Case 3 : SetActiveGadget(1) ; Activate ComboBoxGadget
17             EndSelect
18         EndIf
19     Until Event = #PB_Event_CloseWindow
20 EndIf
```

See Also

GetActiveGadget() , SetActiveWindow()

121.62 SetGadgetAttribute

Syntax

```
SetGadgetAttribute(#Gadget, Attribute, Value)
```

Description

Changes an attribute value of the specified gadget.

Parameters

#Gadget The gadget to use.

Attribute The attribute to set. See the documentation of each gadget for the supported attributes and their meaning.

Value The value to set for the attribute.

Return value

None.

Remarks

This function is available for all gadgets which support attributes:

- ButtonImageGadget()
- CalendarGadget()
- CanvasGadget()
- DateGadget()
- EditorGadget()
- ExplorerListGadget()
- ListIconGadget()
- MDIGadget()
- OpenGLGadget()
- ProgressBarGadget()
- ScrollAreaGadget()
- ScrollBarGadget()
- SpinGadget()
- SplitterGadget()
- StringGadget()
- TrackBarGadget()
- WebGadget()

See Also

GetGadgetAttribute() , GetGadgetItemAttribute() , SetGadgetItemAttribute()

121.63 SetGadgetColor

Syntax

```
SetGadgetColor(#Gadget, ColorType, Color)
```

Description

Changes a color attribute on the given gadget.

Parameters

#Gadget The gadget to use.

ColorType The kind of color attribute to change. This can be one of the following values. See the documentation of each gadget for the supported color attributes:

```

#PB_Gadget_FrontColor      : Gadget text
#PB_Gadget_BackColor       : Gadget background
#PB_Gadget_LineColor       : Color for gridlines
#PB_Gadget_TitleFrontColor: Text color in the title
    (for CalendarGadget())
)
#PB_Gadget_TitleBackColor : Background color in the title
    (for CalendarGadget())
)
#PB_Gadget_GrayTextColor   : Color for grayed out text
    (for CalendarGadget())
)

```

Color The new color for the attribute. RGB() can be used to get a valid color value. To remove the custom color and go back to the default system color, set the 'Color' parameter to #PB_Default.

Return value

None.

Remarks

This function is supported by the following gadgets:

- CalendarGadget()
- ContainerGadget()
- DateGadget()
- EditorGadget()
- ExplorerListGadget()
- ExplorerTreeGadget()
- HyperLinkGadget()
- ListViewGadget()
- ListIconGadget()
- MDIGadget()
- ProgressBarGadget() (Windows only)
- ScrollAreaGadget()
- SpinGadget()
- StringGadget()
- TextGadget()
- TreeGadget()

Note: With activated Windows style the color settings will probably be ignored or overwritten by the style.

Example

```

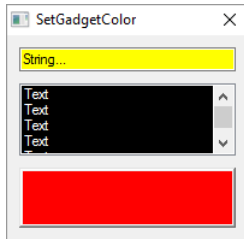
1  If OpenWindow(0, 0, 0, 200, 170, "SetGadgetColor",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      StringGadget(0, 10, 10, 180, 20, "String...")
3      ListViewGadget(1, 10, 40, 180, 60)
4      For i = 0 To 4
5          AddGadgetItem(1, -1, "Text")
6      Next i
7      ContainerGadget(2, 10, 110, 180, 50, #PB_Container_Raised)
8
9      SetGadgetColor(0, #PB_Gadget_BackColor, $00FFFF)
10     SetGadgetColor(1, #PB_Gadget_FrontColor, $FFFFFF)
11     SetGadgetColor(1, #PB_Gadget_BackColor, $000000)

```

```

12     SetGadgetColor(2, #PB_Gadget_BackColor, $0000FF)
13
14     Repeat
15     Until WaitWindowEvent() = #PB_Event_CloseWindow
16 EndIf

```



See Also

GetGadgetColor(), GetGadgetItemColor(), SetGadgetItemColor()

121.64 SetGadgetData

Syntax

```
SetGadgetData(#Gadget, Value)
```

Description

Stores the given value with the specified gadget. This value can later be read with the GetGadgetData() function. This allows to associate a custom value with any gadget.

Parameters

#Gadget The gadget to use.

Value The value to set.

Return value

None.

Example

```

1  ; This code uses SetGadgetData to associate an index for the
   Messages()
2  ; array with each button. This makes the event loop simpler as
   not every
3  ; gadget needs to be handled separately.
4  ;
5  Dim Messages.s(2)
6  Messages(0) = "Good morning"
7  Messages(1) = "Hello World"
8  Messages(2) = "Nothing to say"
9  If OpenWindow(0, 0, 0, 190, 100, "SetGadgetData",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
10     ButtonGadget(0, 10, 10, 80, 20, "Button"): SetGadgetData(0, 1)
11     ButtonGadget(1, 10, 40, 80, 20, "Button"): SetGadgetData(1, 2)

```

```

12     ButtonGadget(2, 10, 70, 80, 20, "Button"): SetGadgetData(2, 1)
13     ButtonGadget(3, 100, 10, 80, 20, "Button"): SetGadgetData(3, 2)
14     ButtonGadget(4, 100, 40, 80, 20, "Button") ; will have the
value 0 because nothing was set yet
15     ButtonGadget(5, 100, 70, 80, 20, "Button")
16     Repeat
17         Event = WaitWindowEvent()
18         If Event = #PB_Event_Gadget
19             Value = GetGadgetData(EventGadget())
20             MessageRequester("Message", Messages(Value))
21         EndIf
22     Until Event = #PB_Event_CloseWindow
23 EndIf

```

See Also

GetGadgetData() , GetGadgetItemData() , SetGadgetItemData()

121.65 SetGadgetFont

Syntax

```
SetGadgetFont(#Gadget, FontID)
```

Description

Changes the font of the specified gadget.

Parameters

#Gadget The gadget to use. If this parameter is set to **#PB_Default**, the font used by newly created gadgets is changed.

FontID The font to set. The FontID() function can be used to easily obtain a valid FontID. If this parameter is set to **#PB_Default**, then the system default font will be used.

Return value

None.

Example

```

1  If OpenWindow(0, 0, 0, 222, 130, "SetGadgetFont",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      If LoadFont(0, "Arial", 16)
3          SetGadgetFont(#PB_Default, FontID(0)) ; Set the loaded
Arial 16 font as new standard
4      EndIf
5      ButtonGadget(0, 10, 10, 200, 30, "Button - Arial 16")
6      SetGadgetFont(#PB_Default, #PB_Default) ; Set the font
settings back to original standard font
7      ButtonGadget(1, 10, 50, 200, 30, "Button - standard")
8      If LoadFont(1, "Courier", 10, #PB_Font_Bold | #PB_Font_Underline)
9          SetGadgetFont(#PB_Default, FontID(1)) ; Set the loaded
Courier 10 font as new standard

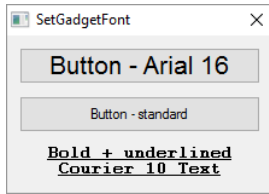
```



```

10     EndIf
11     TextGadget(2, 10, 90, 200, 40, "Bold + underlined Courier 10
    Text", #PB_Text_Center)
12     Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
13 EndIf

```



See Also

GetGadgetFont() , FontID() , LoadFont()

121.66 SetGadgetItemAttribute

Syntax

```
SetGadgetItemAttribute(#Gadget, Item, Attribute, Value [, Column])
```

Description

Changes an attribute value of the specified gadget item.

Parameters

#Gadget The gadget to use.

Item The item to use. The first item in the gadget has index 0.

Attribute The attribute to set. See below for the supported values.

Value The value to set for the attribute.

Column (optional) The column to use for gadgets that support multiple columns. The first column has index 0. The default is column 0.

Remarks

This function is available for all gadgets which support item attributes:

- ExplorerListGadget() :

```
#PB_Explorer_ColumnWidth : Changes the width of the given
'Column'. The 'Item' parameter is ignored.
```

- ListIconGadget() :

```
#PB_ListIcon_ColumnWidth : Changes the width of the given
'Column'. The 'Item' parameter is ignored.
```

See Also

GetGadgetItemAttribute() , GetGadgetAttribute() , SetGadgetAttribute()

121.67 SetGadgetItemColor

Syntax

```
SetGadgetItemColor(#Gadget, Item, ColorType, Color [, Column])
```

Description

Changes a color attribute of the given gadget item.

Parameters

#Gadget The gadget to use.

Item The item to use. The first item in the gadget has index 0. If this parameter is set to `#PB_All`, the color will be applied to the given 'Column' in all gadget items.

ColorType The kind of color attribute to change. This can be one of the following values:

```
#PB_Gadget_FrontColor      : Item text
#PB_Gadget_BackColor       : Item background
```

Color The new color for the attribute. `RGB()` can be used to get a valid color value. To remove the custom color and go back to the default system color, set the 'Color' parameter to `#PB_Default`.

Column (optional) The column to use for gadgets that support multiple columns. The first column has index 0. The default is column 0. If the this parameter is set to `#PB_All`, the color will be applied to all columns of the given Item.

Return value

None.

Remarks

This function is supported by the following gadgets:

- ListIconGadget()
- TreeGadget()

Note: With activated Windows style the color settings will probably be ignored or overwritten by the style.

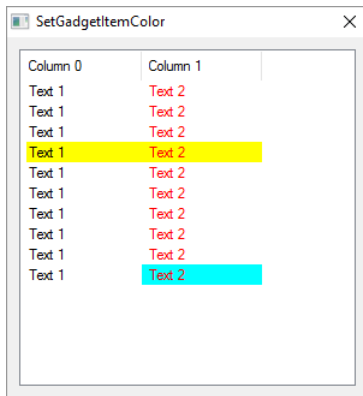
Example

```
1  If OpenWindow(0, 0, 0, 300, 300, "SetGadgetItemColor",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     ListIconGadget(0, 10, 10, 280, 280, "Column 0", 100)
4     AddGadgetColumn(0, 1, "Column 1", 100)
5     For i = 1 To 10
6         AddGadgetItem(0, -1, "Text 1"+Chr(10)+"Text 2")
7     Next
8     SetGadgetItemColor(0, #PB_All, #PB_Gadget_FrontColor, $0000FF,
9         1)
10    SetGadgetItemColor(0, 3, #PB_Gadget_BackColor, $00FFFF,
11        #PB_All)
12    SetGadgetItemColor(0, 9, #PB_Gadget_BackColor, $FFFF00, 1)
13    Repeat
```

```

12     Until WaitWindowEvent() = #PB_Event_CloseWindow
13 EndIf

```



See Also

GetGadgetItemColor() , GetGadgetColor() , SetGadgetColor()

121.68 SetGadgetItemData

Syntax

```
SetGadgetItemData(#Gadget, Item, Value)
```

Description

Stores the given value with the specified gadget item. This value can later be read with the GetGadgetItemData() function. This allows to associate a custom value with the items of a gadget.

Parameters

#Gadget The gadget to use.

Item The item to use. The first item in the gadget has index 0.

Value The value to set.

Return value

None.

Remarks

The set value will remain with the item, even if the item changes its index (for example because other items were deleted).

This function works with the following gadgets:

- ComboBoxGadget()
- ListIconGadget()
- ListViewGadget()
- PanelGadget()
- TreeGadget()

Example

```
1 ; This code uses SetGadgetItemData to store the original position
2 ; of each item to later know it, even if the items index changed.
3 ;
4 If OpenWindow(0, 0, 0, 280, 250, "SetGadgetItemData",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
5     ButtonGadget(0, 10, 10, 80, 20, "Add")
6     ButtonGadget(1, 100, 10, 80, 20, "Remove")
7     ButtonGadget(2, 190, 10, 80, 20, "Test")
8     ListViewGadget(3, 10, 40, 260, 200)
9     For i = 0 To 10
10         AddGadgetItem(3, i, "Old Item "+Str(i))
11         SetGadgetItemData(3, i, i)
12     Next i
13
14 Repeat
15     Event = WaitWindowEvent()
16     If Event = #PB_Event_Gadget
17         item = GetGadgetState(3)
18
19         Select EventGadget()
20             Case 0 ; Add
21                 AddGadgetItem(3, item, "New Item")
22                 If item <> -1
23                     SetGadgetItemData(3, item, -1)
24                 Else
25                     SetGadgetItemData(3, CountGadgetItems(3)-1, -1)
26                 EndIf
27
28             Case 1 ; Remove
29                 If item <> -1
30                     RemoveGadgetItem(3, item)
31                 EndIf
32
33             Case 2 ; Test
34                 If item <> -1
35                     value = GetGadgetItemData(3, item)
36                     If value = -1
37                         MessageRequester("", "Its a new item.")
38                     Else
39                         MessageRequester("", "It was item number
40 "+Str(value))
41                     EndIf
42                 EndIf
43             EndSelect
44         EndIf
45     Until Event = #PB_Event_CloseWindow
46 EndIf
```

See Also

GetGadgetItemData() , GetGadgetData() , SetGadgetData()

121.69 SetGadgetItemImage

Syntax

```
SetGadgetItemImage(#Gadget, Item, ImageID)
```

Description

Changes the image of the specified gadget item.

Parameters

#Gadget The gadget to use.

Item The item to change the image. The first item in the gadget has index 0.

ImageID The new image to use for the gadget item. The used image should be in the standard 16x16 size. Use the ImageID() command to get the ID for this parameter.

Return value

None.

Remarks

This is a universal function which works for the following gadgets:

- ComboBoxGadget()
- ListIconGadget()
- PanelGadget() (not supported on OS X)
- TreeGadget()

See Also

AddGadgetItem()

121.70 SetGadgetItemState

Syntax

```
SetGadgetItemState(#Gadget, Item, State)
```

Description

Changes the item state of the specified gadget.

Parameters

#Gadget The gadget to use.

Item The item to use. The first item in the gadget has index 0.

State The new state for the item. See below for the meaning of this parameter.

Return value

None.

Remarks

This is a universal function which works for almost all gadgets which handle several items. 'State' can take the following values:

- ExplorerListGadget() : with the following value:

`#PB_Explorer_Selected` : The item should be selected.

- ListViewGadget() : 1 if the item should be selected, 0 otherwise.

- ListIconGadget() : Combination of the following values:

`#PB_ListIcon_Selected` : The item should be selected.

`#PB_ListIcon_Checked` : The item should have its checkbox checked (`#PB_ListIcon_CheckBoxes` flag).

`#PB_ListIcon_Inbetween`: The item should have its checkbox in the in between state (`#PB_ListIcon_ThreeState` flag).

- TreeGadget() : Combination of the following values:

`#PB_Tree_Selected` : The item should be selected.

`#PB_Tree_Expanded` : The item should be expanded.

`#PB_Tree_Collapsed`: The item should be collapsed. If neither `#PB_Tree_Expanded` nor `#PB_Tree_Collapsed` is set, this state will not be changed.

`#PB_Tree_Checked` : The items checkbox should be checked.

`#PB_Tree_Inbetween`: The items checkbox should be in the in between state.

```
1 SetGadgetItemState(0, 1, #PB_Tree_Expanded | #PB_Tree_Selected) ;  
   The 2nd item is selected and expanded.
```

See Also

GetGadgetItemState() , GetGadgetState() , SetGadgetState()

121.71 SetGadgetItemText

Syntax

```
SetGadgetItemText(#Gadget, Item, Text$ [, Column])
```

Description

Changes the item text of the specified gadget.

Parameters

#Gadget The gadget to use.

Item The item to use. The first item in the gadget has index 0.

Text\$ The new text to set.

Column (optional) The column to use for gadgets that support multiple columns. The first column has index 0. The default is column 0.

Return value

None.

Remarks

This is a universal function which works for almost all gadgets which handle several items:

- ComboBoxGadget()
- EditorGadget()
- ExplorerListGadget() : If Item = -1, the header text of the given column is changed.
- ListIconGadget() : If Item = -1, the header text of the given column is changed.
- ListViewGadget()
- MDIGadget()
- PanelGadget()
- TreeGadget()
- WebGadget() : Change the html code in the gadget with `#PB_Web_HtmlCode` as 'Item'.
- WebViewGadget() : Change the html code in the gadget with `#PB_Web_HtmlCode` as 'Item'.

See Also

GetGadgetItemText() , GetGadgetText() , SetGadgetText()

121.72 SetGadgetState

Syntax

```
SetGadgetState(#Gadget , State)
```

Description

Change the current state of the specified gadget.

Parameters

#Gadget The gadget to use.

State The new state for the gadget. See below for the meaning of this value depending on the gadget type.

Return value

None.

Remarks

This is a universal function which works for almost all gadgets:

- ButtonImageGadget() : change the current state of a `#PB_Button_Toggle` gadget (1 = toggled, 0 = normal).
- ButtonGadget() : change the current state of a `#PB_Button_Toggle` gadget (1 = toggled, 0 = normal).
- CalendarGadget() : set the currently selected date.
- CheckBoxGadget() : Change the state of the checkbox. The following values are possible:

```
#PB_CheckBox_Checked   : Set the check mark.
#PB_CheckBox_Unchecked: Remove the check mark.
#PB_CheckBox_Inbetween: Set the "in between" state. (Only for
#PB_CheckBox_ThreeState checkboxes)
```

- `ComboBoxGadget()` : change the currently selected item.
- `DateGadget()` : set the currently displayed date/time. If `#PB_Date_CheckBox` was used, set 'State' to 0 to uncheck the checkbox.
- `ImageGadget()` : change the current image of the gadget.
- `IPAddressGadget()` : change the current IP address.
- `ListIconGadget()` : change the currently selected item. If -1 is specified, all items will be deselected.
- `ListViewGadget()` : change the currently selected item. . If -1 is specified, it will remove the selection.
- `MDIGadget()` : Change the currently focused childwindow (by giving the related `#Window` number), or use one of the following values:

```
#PB_MDI_Cascade           : Cascade the child windows
#PB_MDI_TileVertically    : Tile the childwindows vertically
#PB_MDI_TileHorizontally : Tile the childwindows horizontally
#PB_MDI_Next              : Give focus to the next childwindow
#PB_MDI_Previous          : Give focus to the previous childwindow
#PB_MDI_Arrange           : Arrange the iconic (minimized) windows
```

- `OptionGadget()` : 1 to activate it, 0 otherwise.
- `PanelGadget()` : change the current panel.
- `ProgressBarGadget()` : change progress bar value. A value of `#PB_ProgressBar_Unknown` can be used to indicate that the progress is unknown.
- `ScrollBarGadget()` : change the current slider position.
- `ShortcutGadget()` : Change the current shortcut.
- `SpinGadget()` : change the current value.
- `SplitterGadget()` : change the current splitter position, in pixels.
- `TrackBarGadget()` : change the current cursor position.
- `TreeGadget()` : change the currently selected item, -1 selects no item.
- `WebGadget()` : perform some action on the gadget. See `WebGadget` for further descriptions.

See Also

`GetGadgetState()` , `GetGadgetItemState()` , `SetGadgetItemState()`

121.73 SetGadgetText

Syntax

```
SetGadgetText(#Gadget, Text$)
```

Description

Change the gadget text content of the specified gadget.

Parameters

#Gadget The gadget to use.

Text\$ The new text to set.

Return value

None.

Remarks

This function is especially useful for:

- `ButtonGadget()` : change the text of the `ButtonGadget`.
- `ComboBoxGadget()` : Set the displayed text. If the `ComboBoxGadget` is not editable, the text must be in the dropdown list.
- `DateGadget()` : change the input mask for the dates in the gadget. See `FormatDate()` for the format of the `Text$` parameter.
- `EditorGadget()` : change the text content of the editor gadget. If you want to add several lines of text at once, separate them with `"Chr(13)+Chr(10)"` on Windows and `"Chr(10)"` on Linux and OS X.
- `ExplorerComboGadget()` : change the current displayed directory.
- `ExplorerListGadget()` : change the current displayed directory and/or the pattern for files (see `ExplorerListGadget()` for more details).
- `ExplorerTreeGadget()` : change the current selected directory and/or the pattern for files (see `ExplorerTreeGadget()` for more details).
- `FrameGadget()` : change the title of the `FrameGadget`.
- `HyperLinkGadget()` : change the text of the `HyperLinkGadget`.
- `ListViewGadget()` : selects the item that exactly matches the given text.
- `StringGadget()` : change the content of the `StringGadget`.
- `TextGadget()` : change the content of the `TextGadget`.
- `TreeGadget()` : change the text of the currently selected item.
- `WebGadget()` : change the current URL.
- `WebViewGadget()` : change the current URL.

See Also

`GetGadgetText()` , `GetGadgetItemText()` , `SetGadgetItemText()`

121.74 ShortcutGadget

Syntax

```
Result = ShortcutGadget(#Gadget, x, y, Width, Height, Shortcut)
```

Description

Creates a gadget for keyboard shortcut selection in the current `GadgetList`. The user can select it and hold down a keyboard combination to select a new shortcut.

Parameters

#Gadget A number to identify the new gadget. `#PB_Any` can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Shortcut The initial shortcut to display. The possible values are the same as in the `AddKeyboardShortcut()` function. A value of 0 can be used to indicate that no shortcut is currently set.

Return value

Returns nonzero on success and zero on failure. If `#PB_Any` was used as the `#Gadget` parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using `GadgetToolTip()` .

The following functions can be used to act on a `ShortcutGadget`:

- `GetGadgetState()` : Get the currently selected keyboard shortcut.
- `SetGadgetState()` : Change the currently selected keyboard shortcut.

Example

```
1  If OpenWindow(0, 0, 0, 240, 70, "ShortcutGadget",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      ShortcutGadget(0, 20, 20, 200, 25, #PB_Shortcut_Control |
4          #PB_Shortcut_A)
5      Repeat
6          Event = WaitWindowEvent()
7          Until Event = #PB_Event_CloseWindow
8      EndIf
```

See Also

`GetGadgetState()` , `SetGadgetState()` , `AddKeyboardShortcut()`

121.75 SpinGadget

Syntax

```
Result = SpinGadget(#Gadget, x, y, Width, Height, Minimum, Maximum
    [, Flags])
```

Description

Creates a spin gadget in the current `GadgetList`.

Parameters

#Gadget A number to identify the new gadget. `#PB_Any` can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Minimum, Maximum The minimum and maximum values for the gadget.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

```
#PB_Spin_ReadOnly : The StringGadget is not editable, the
                    number is only changeable by the arrows
#PB_Spin_Numeric  : The SpinGadget will automatically update
                    the text with value of the state, so SetGadgetText()
                    is not needed.
```

Return value

Returns nonzero on success and zero on failure. If `#PB_Any` was used as the `#Gadget` parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using `GadgetToolTip()` .

The following functions can be used to act on a `SpinGadget`:

`GetGadgetState()` : Get the current gadget value.

`SetGadgetState()` : Change the gadget value. For displaying the new value you still must use

`SetGadgetText()` !

`GetGadgetText()` : Get the text contained in the gadget.

`SetGadgetText()` : Change the text contained in the gadget.

`GetGadgetAttribute()` : With one of the following attributes:

```
#PB_Spin_Minimum      : Returns the minimum value.
#PB_Spin_Maximum      : Returns the maximum value.
```

`SetGadgetAttribute()` : With one of the following attributes:

```
#PB_Spin_Minimum      : Changes the minimum value.
#PB_Spin_Maximum      : Changes the maximum value.
```

The following events are supported through `EventType()` :

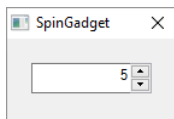
```
#PB_EventType_Change: The text in the edit area has been modified
                    by the user.
#PB_EventType_Up     : The 'Up' button was pressed.
#PB_EventType_Down   : The 'Down' button was pressed.
```

This gadget supports the `SetGadgetColor()` and `GetGadgetColor()` functions with the following values as 'ColorType':

```
#PB_Gadget_FrontColor: Textcolor
#PB_Gadget_BackColor  : Backgroundcolor
```

Example

```
1  If OpenWindow(0, 0, 0, 140, 70, "SpinGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      SpinGadget      (0, 20, 20, 100, 25, 0, 1000)
3      SetGadgetState (0, 5) : SetGadgetText(0, "5")    ; set initial
    value
4      Repeat
5          Event = WaitWindowEvent()
6          If Event = #PB_Event_Gadget
7              If EventGadget() = 0
8                  SetGadgetText(0, Str(GetGadgetState(0)))
9              EndIf
10         EndIf
11     Until Event = #PB_Event_CloseWindow
12 EndIf
```



See Also

`GetGadgetState()` , `SetGadgetState()` , `GetGadgetText()` , `SetGadgetText()` ,
`GetGadgetAttribute()` , `SetGadgetAttribute()` , `GetGadgetColor()` , `SetGadgetColor()`

121.76 SplitterGadget

Syntax

```
Result = SplitterGadget(#Gadget, x, y, Width, Height, #Gadget1,
                        #Gadget2 [, Flags])
```

Description

Creates a Splitter gadget in the current GadgetList. This gadget allows two child gadgets to be resized by the user with a separator bar.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

#Gadget1, #Gadget2 The gadgets to be placed in the splitter.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

```
#PB_Splitter_Vertical      : The gadget is split vertically
                             (instead of horizontally which is the default).
#PB_Splitter_Separator     : A 3D-looking separator is drawn in
                             the splitter.
#PB_Splitter_FirstFixed    : When the splitter gadget is
                             resized, the first gadget will keep its size
#PB_Splitter_SecondFixed   : When the splitter gadget is
                             resized, the second gadget will keep its size
```

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using `GadgetToolTip()`.

The following functions can be used to act on a `SplitterGadget`:

`GetGadgetState()` : Get the current splitter position, in pixels.

`SetGadgetState()` : Change the current splitter position, in pixels.

`GetGadgetAttribute()` : With one of the following attribute:

```
#PB_Splitter_FirstMinimumSize : Gets the minimum size (in pixels)
                               than the first gadget can have.
#PB_Splitter_SecondMinimumSize: Gets the minimum size (in pixels)
                               than the second gadget can have.
#PB_Splitter_FirstGadget      : Gets the gadget number of the
                               first gadget.
#PB_Splitter_SecondGadget     : Gets the gadget number of the
                               second gadget.
```

`SetGadgetAttribute()` : With one of the following attribute:

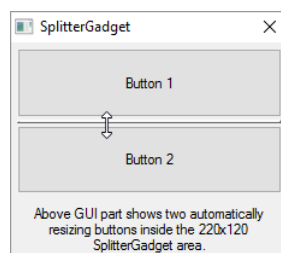
#PB_Splitter_FirstMinimumSize : Sets the minimum size (in pixels) than the first gadget can have.
#PB_Splitter_SecondMinimumSize: Sets the minimum size (in pixels) than the second gadget can have.
#PB_Splitter_FirstGadget : Replaces the first gadget with a new one.
#PB_Splitter_SecondGadget : Replaces the second gadget with a new one.

Note: When replacing a gadget with `SetGadgetAttribute()`, the old gadget will not be automatically freed. It will instead be put back on the parent window of the Splitter. This allows to switch gadgets between splitters without the need to recreate any of them. If the old gadget should be freed, its number can first be retrieved with `GetGadgetAttribute()`, and the gadget freed with `FreeGadget()` after it has been replaced. Note that a gadget cannot be in two splitters at once. So to move a gadget from one splitter to another, it first needs to be replaced in the first splitter so it is on the main window and then it can be put into the second splitter.

Example

```

1  If OpenWindow(0, 0, 0, 230, 180, "SplitterGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      #Button1 = 0
3      #Button2 = 1
4      #Splitter = 2
5
6      ButtonGadget(#Button1, 0, 0, 0, 0, "Button 1") ; No need to
    specify size or coordinates
7      ButtonGadget(#Button2, 0, 0, 0, 0, "Button 2") ; as they will
    be sized automatically
8      SplitterGadget(#Splitter, 5, 5, 220, 120, #Button1, #Button2,
    #PB_Splitter_Separator)
9
10     TextGadget(3, 10, 135, 210, 40, "Above GUI part shows two
    automatically resizing buttons inside the 220x120 SplitterGadget
    area.", #PB_Text_Center )
11
12     Repeat
13     Until WaitWindowEvent() = #PB_Event_CloseWindow
14 EndIf
  
```



See Also

`GetGadgetState()`, `SetGadgetState()`, `GetGadgetAttribute()`, `SetGadgetAttribute()`

121.77 StringGadget

Syntax

```
Result = StringGadget(#Gadget, x, y, Width, Height, Content$,
    Flags)
```

Description

Creates a String gadget in the current GadgetList. It allows the user to enter a single line of text.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Content\$ The initial content of this StringGadget. This gadget accepts only one line of text. To get multi-line input, use the EditorGadget() function.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

```
#PB_String_Numeric      : Only (positive) integer numbers are
    accepted.
#PB_String_Password     : Password mode, displaying only '*'
    instead of normal characters.
#PB_String_ReadOnly     : Read-only mode. No text can be
    entered.
#PB_String_LowerCase    : All characters are converted to lower
    case automatically.
#PB_String_UpperCase    : All characters are converted to upper
    case automatically.
#PB_String_BorderLess   : No borders are drawn around the
    gadget.
```

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

Remarks

Later the content can be changed with SetGadgetText() and received with GetGadgetText() . The following events are supported through EventType() :

```
#PB_EventType_Change    : The text has been modified by the user.
#PB_EventType_Focus     : The StringGadget got the focus.
#PB_EventType_LostFocus : The StringGadget lost the focus.
```

The following functions can be used to act on this gadget:

- SetGadgetColor() and GetGadgetColor() functions with the following values as 'ColorType':

```
#PB_Gadget_FrontColor : Textcolor
#PB_Gadget_BackColor  : Backgroundcolor
```

- GetGadgetAttribute() with the following attribute:

```
#PB_String_MaximumLength: Returns the maximum number of
    characters which can be entered.
```

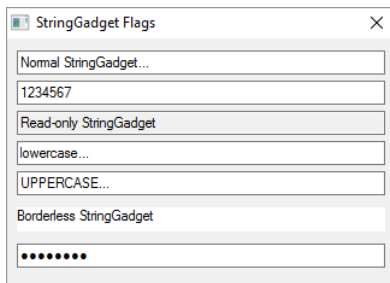
- SetGadgetAttribute() with the following attribute:

```
#PB_String_MaximumLength: Set the maximum number of characters
    which can be entered.
```

A 'mini help' can be added to this gadget using GadgetToolTip() .

Example

```
1 ; Shows possible flags of StringGadget in action...
2 If OpenWindow(0, 0, 0, 322, 205, "StringGadget Flags",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     StringGadget(0, 8, 10, 306, 20, "Normal StringGadget...")
4     StringGadget(1, 8, 35, 306, 20, "1234567", #PB_String_Numeric)
5     StringGadget(2, 8, 60, 306, 20, "Read-only StringGadget",
   #PB_String_ReadOnly)
6     StringGadget(3, 8, 85, 306, 20, "lowercase...",
   #PB_String_LowerCase)
7     StringGadget(4, 8, 110, 306, 20, "uppercase...",
   #PB_String_UpperCase)
8     StringGadget(5, 8, 140, 306, 20, "Borderless StringGadget",
   #PB_String_BorderLess)
9     StringGadget(6, 8, 170, 306, 20, "Password",
   #PB_String_Password)
10 Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
11 EndIf
```



See Also

GetGadgetText() , SetGadgetText() , GetGadgetColor() , SetGadgetColor() , EditorGadget()

121.78 TextGadget

Syntax

```
Result = TextGadget(#Gadget, x, y, Width, Height, Text$ [, Flags])
```

Description

Creates a Text gadget in the current GadgetList. A TextGadget is a basic text area for displaying, not entering, text.

Parameters

#Gadget A number to identify the new gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Text\$ The text to display.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

```
#PB_Text_Center : The text is centered in the gadget.
#PB_Text_Right  : The text is right aligned.
#PB_Text_Border : A sunken border is drawn around the gadget.
```

Return value

Returns nonzero on success and zero on failure. If `#PB_Any` was used as the `#Gadget` parameter then the return-value is the auto-generated gadget number on success.

Remarks

The content can be changed with the function `SetGadgetText()` and can be obtained with `GetGadgetText()`. The font of a `TextGadget()` you easily change with `SetGadgetFont()`. This gadget supports the `SetGadgetColor()` and `GetGadgetColor()` functions with the following values as 'ColorType':

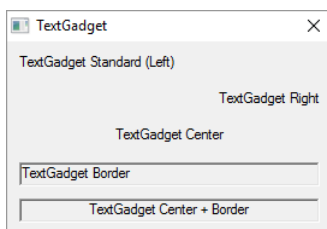
```
#PB_Gadget_FrontColor: Textcolor
#PB_Gadget_BackColor : Backgroundcolor
```

Note: This Gadget doesn't receive any user events.

Note: `GadgetToolTip()` only works with Linux.

Example

```
1  If OpenWindow(0, 0, 0, 270, 160, "TextGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      TextGadget(0, 10, 10, 250, 20, "TextGadget Standard (Left)")
3      TextGadget(1, 10, 70, 250, 20, "TextGadget Center",
    #PB_Text_Center)
4      TextGadget(2, 10, 40, 250, 20, "TextGadget Right",
    #PB_Text_Right)
5      TextGadget(3, 10, 100, 250, 20, "TextGadget Border",
    #PB_Text_Border)
6      TextGadget(4, 10, 130, 250, 20, "TextGadget Center + Border",
    #PB_Text_Center | #PB_Text_Border)
7      Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
8      EndIf
```



See Also

`GetGadgetText()`, `SetGadgetText()`, `GetGadgetColor()`, `SetGadgetColor()`

121.79 TrackBarGadget

Syntax


```
Result = TrackBarGadget(#Gadget, x, y, Width, Height, Minimum,
    Maximum [, Flags])
```

Description

Creates a TrackBar gadget in the current GadgetList. It allows you to select a range of values with a slide bar, like ones found in several multimedia players.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Minimum, Maximum The range of values used by the gadget. These values should be between 0 and 10,000 to avoid limitations on some operating systems.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

```
#PB_TrackBar_Ticks      : Display a 'tick' marker at each step.
#PB_TrackBar_Vertical  : The trackbar is vertical (instead of
    horizontal which is the default).
```

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

Remarks

A 'mini help' can be added to this gadget using `GadgetToolTip()` .

The following functions can be used to act on this gadget:

- `GetGadgetState()` : Returns the current cursor position (value between the Minimum-Maximum range).
- `SetGadgetState()` : Change the current cursor position.
- `GetGadgetAttribute()` with one of the following attributes:

```
#PB_TrackBar_Minimum    : Returns the minimum value.
#PB_TrackBar_Maximum    : Returns the maximum value.
```

- `SetGadgetAttribute()` with one of the following attributes:

```
#PB_TrackBar_Minimum    : Changes the minimum value.
#PB_TrackBar_Maximum    : Changes the maximum value.
```

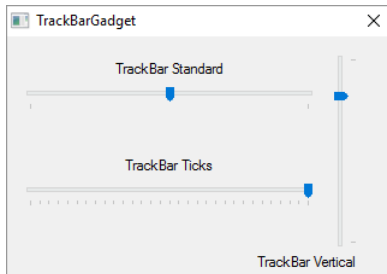
Example

```
1  If OpenWindow(0, 0, 0, 320, 200, "TrackBarGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      TextGadget      (3, 10, 20, 250, 20, "TrackBar Standard",
    #PB_Text_Center)
3      TrackBarGadget(0, 10, 40, 250, 20, 0, 10000)
4      SetGadgetState(0, 5000)
5      TextGadget      (4, 10, 100, 250, 20, "TrackBar Ticks",
    #PB_Text_Center)
6      TrackBarGadget(1, 10, 120, 250, 20, 0, 30, #PB_TrackBar_Ticks)
7      SetGadgetState(1, 3000)
```

```

8      TextGadget      (5,  90, 180, 200, 20, "TrackBar Vertical",
#PB_Text_Right)
9      TrackBarGadget(2, 270, 10, 20, 170, 0, 10000,
#PB_TrackBar_Vertical)
10     SetGadgetState(2, 8000)
11     Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
12 EndIf

```



See Also

GetGadgetState() , SetGadgetState() , GetGadgetAttribute() , SetGadgetAttribute()

121.80 TreeGadget

Syntax

Result = TreeGadget(#Gadget , x, y, Width, Height [, Flags])

Description

Creates a Tree gadget in the current GadgetList.

Parameters

#Gadget A number to identify the new gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Flags (optional) Flags to modify the gadget behavior. It can be a combination of the following values:

#PB_Tree_AlwaysShowSelection	: Even if the gadget isn't activated, the selection is still visible.
#PB_Tree_NoLines	: Hide the little lines between each nodes.
#PB_Tree_NoButtons	: Hide the '+' node buttons.
#PB_Tree_CheckBoxes	: Add a checkbox before each item.
#PB_Tree_ThreeState	: The checkboxes can have an "in between" state.

The #PB_Tree_ThreeState flag can be used in combination with the #PB_Tree_CheckBoxes flag to get checkboxes that can have an "on", "off" and "in between" state. The user can only select the "on" or "off" states. The "in between" state can be set programmatically using the SetGadgetItemState() function.

Return value

Returns nonzero on success and zero on failure. If `#PB_Any` was used as the `#Gadget` parameter then the return-value is the auto-generated gadget number on success.

Remarks

Each item in the tree has a sublevel value assigned to it, that determines its relation with the item above and below it. Items with the same sublevel belong to the same node, items with a higher sublevel are child-items and so on. This sublevel value can be used to determine the relation between two items by comparing their sublevel values. The 'Flags' parameter of `AddGadgetItem()` is always required for `TreeGadget` items and is used to set the sublevel at which the new item should be added. Note that if the function is called with a sublevel at which the item cannot be added, the item will be added at the sublevel where it is possible to add it.

A 'mini help' can be added to this gadget using `GadgetToolTip()`.

The following functions can be used to act on the tree content:

- `AddGadgetItem()` : Add an item (with an optional picture in the standard 16x16 icon size).
- `RemoveGadgetItem()` : Remove an item (and all its child-items).
- `ClearGadgetItems()` : Remove all the items.
- `CountGadgetItems()` : Return the number of items currently in the `#Gadget`.
- `GetGadgetItemState()` : Return the current state of the specified item.
- `SetGadgetItemState()` : Change the current state of the specified item.
- `GetGadgetItemText()` : Return the current text of the specified item.
- `SetGadgetItemText()` : Change the current text of the specified item.
- `SetGadgetItemImage()` : Change the current image of the specified item.
- `GetGadgetItemData()` : Returns the value that was stored with item.
- `SetGadgetItemData()` : Stores a value with the item.
- `GetGadgetItemColor()` : Returns front or backcolor of the item.
- `SetGadgetItemColor()` : Changes front or backcolor of the item. (not supported on MacOS X)
- `GetGadgetState()` : Return the current selected item.
- `SetGadgetState()` : Change the currently selected item.
- `GetGadgetText()` : Return the text of the currently selected item.
- `SetGadgetText()` : Change the text of the currently selected item.
- `GetGadgetItemAttribute()` : With the following attribute:

`#PB_Tree_SubLevel`: Returns the sublevel value of the given item.

- `GadgetItemID()` : Return the OS handle of the specified item (useful for API functions).

This gadget supports the `SetGadgetColor()` and `GetGadgetColor()` functions with the following values as 'ColorType':

`#PB_Gadget_FrontColor`: `Textcolor`
`#PB_Gadget_BackColor` : `Backgroundcolor`

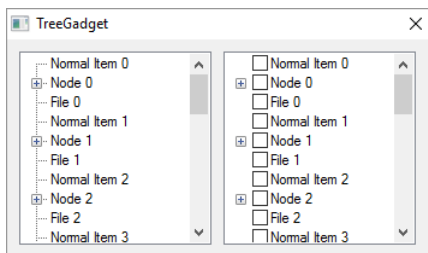
The following events are supported through `EventType()` :

`#PB_EventType_LeftClick`: left click on an item, or a checkbox was checked/unchecked
`#PB_EventType_LeftDoubleClick`
`#PB_EventType_RightClick`
`#PB_EventType_RightDoubleClick`
`#PB_EventType_Change`: the current item changed
`#PB_EventType_DragStart`: the user tried to start a Drag & Drop operation. `ExamineDraggedItems()` can be used to get the dragged items information.

After a `#PB_EventType_DragStart` event, the Drag & Drop library can be used to start a Drag & Drop operation. On MacOS X the `#PB_EventType_DragStart` event has to be handled using `BindEvent()` or `BindGadgetEvent()`.

Example

```
1  If OpenWindow(0, 0, 0, 355, 180, "TreeGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      TreeGadget(0, 10, 10, 160, 160)
                                     ; TreeGadget standard
3      TreeGadget(1, 180, 10, 160, 160, #PB_Tree_CheckBoxes |
    #PB_Tree_NoLines) ; TreeGadget with Checkboxes + NoLines
4      For ID = 0 To 1
5          For a = 0 To 10
6              AddGadgetItem (ID, -1, "Normal Item "+Str(a), 0, 0) ; if
you want to add an image, use
7              AddGadgetItem (ID, -1, "Node "+Str(a), 0, 0)          ;
ImageID(x) as 4th parameter
8              AddGadgetItem(ID, -1, "Sub-Item 1", 0, 1)          ; These are on
the 1st sublevel
9              AddGadgetItem(ID, -1, "Sub-Item 2", 0, 1)
10             AddGadgetItem(ID, -1, "Sub-Item 3", 0, 1)
11             AddGadgetItem(ID, -1, "Sub-Item 4", 0, 1)
12             AddGadgetItem (ID, -1, "File "+Str(a), 0, 0) ; sublevel 0
again
13         Next
14     Next
15     Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
16 EndIf
```



See Also

AddGadgetItem() , RemoveGadgetItem() , ClearGadgetItems() , CountGadgetItems() ,
GetGadgetItemState() , SetGadgetItemState() , GetGadgetItemText() , SetGadgetItemText() ,
SetGadgetItemImage() , GetGadgetItemData() , SetGadgetItemData() , GetGadgetState() ,
SetGadgetState() , GetGadgetText() , SetGadgetText() , GetGadgetItemAttribute() ,
GadgetItemID() , GetGadgetColor() , SetGadgetColor() , ExplorerTreeGadget()

121.81 UseGadgetList

Syntax

```
Result = UseGadgetList(WindowID)
```

Description

Selects the GadgetList window to which gadgets will be added. If there is no GadgetList on this window yet (because it was created with the #PB_Window_NoGadgets flag in OpenWindow()) or because it is not a PB window) it will be created.

Parameters

WindowID The new window to add gadgets to. It can be obtained with the WindowID() function. If 'WindowID' is 0, the current GadgetList window will be returned and nothing will be changed.

Return value

Returns the WindowID of the previous GadgetList window, or 0 if there was none. This value can be used to go back to the previous GadgetList later.

Example

This example shows how to use this command to create a new window with gadgets without interrupting the gadget creation on the current window:

```
1  If OpenWindow(0, 0, 0, 500, 500, "Main Window",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     ButtonGadget(0, 10, 10, 150, 25, "Button 1")
4
5     ; Create Window with #PB_Window_NoGadgets to prevent automatic
6     GadgetList creation
7     If OpenWindow(1, 0, 0, 300, 200, "Child Window",
8         #PB_Window_TitleBar | #PB_Window_WindowCentered |
9         #PB_Window_NoGadgets, WindowID(0))
10        OldGadgetList = UseGadgetList(WindowID(1)) ; Create
11        GadgetList and store old GadgetList
12
13        ButtonGadget(10, 10, 10, 150, 25, "Child Window Button")
14
15        UseGadgetList(OldGadgetList)                ; Return to
16        previous GadgetList
17    EndIf
18
19    ButtonGadget(1, 10, 45, 150, 25, "Button 2") ; This will be on
20    the main window again
21
22    Repeat
23    Until WaitWindowEvent() = #PB_Event_CloseWindow
24    EndIf
```

See Also

OpenWindow(), WindowID()

121.82 WebGadget

Syntax

```
Result = WebGadget(#Gadget, x, y, Width, Height, URL$ [, Flags])
```

Description

Creates a Web gadget in the current GadgetList. It can display html pages.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

URL\$ The url to load after the gadget is created.

Flags (optional) Can be following constant:

```
#PB_Web_Edge: Uses the WebView2 engine to render the web page
               (Windows only). WebView2 is the Edge based component
               to have modern web page display in Windows
               application. It's built-in for up-to-date Windows 10 and
               Windows 11. For older Windows version, it needs
               to be installed (if not already): distribution guide.
```

Return value

Returns nonzero on success and zero on failure. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

This function fails if the required components for the WebGadget cannot be loaded. See below for the requirements for the WebGadget on each OS.

Remarks

The following components are required to use the WebGadget on each OS. These components are required to use the WebGadget, not only for the compilation of the program.

Windows

The WebGadget uses the Internet Explorer 11 ActiveX object on Windows. If the flag **#PB_Web_Edge** is used, then it uses the modern Edge based component instead. Remember to replace `"\"` with `"/` in your codes.

Linux

The WebGadget uses the WebKitGtk library on Linux. The package with this library is named 'libwebkit'. Some distributions may include an old version of this package which is named 'WebKitGtk'. If your distribution does not include a package for this library, the sources can be downloaded from the [WebKitGtk home page](#).

Mac OSX

The WebGadget uses the WebKit component on Mac OSX. This component comes with the operating system. There are no further requirements.

The following functions can be used to act on a WebGadget:

- SetGadgetText() : Change the current URL.
- GetGadgetText() : Get the current URL.
- SetGadgetState() : Perform an action on the gadget. The following constants are valid:

```
#PB_Web_Back    : One step back in the navigation history.
#PB_Web_Forward : One step forward in the navigation history.
#PB_Web_Stop    : Stop loading the current page.
#PB_Web_Refresh : Refresh the current page.
```

- SetGadgetItemText() : With **#PB_Web_HtmlCode** as 'Item' html code can be streamed into the Gadget. (Windows only)
- GetGadgetItemText() : The following constants can be used to get information (Windows only):

#PB_Web_HtmlCode : Get the html code from the gadget.
#PB_Web_PageTitle : Get the current title for the displayed page.
#PB_Web_StatusMessage: Get the current statusbar message.
#PB_Web_SelectedText : Get the currently selected text inside the gadget.

- **SetGadgetAttribute()** : Set the following attributes:

#PB_Web_NavigationCallback: Set a callback for monitoring (and disabling) navigation.
#PB_Web_ScrollX : Set the horizontal scrolling position (Windows only).
#PB_Web_ScrollY : Set the vertical scrolling position (Windows only).
#PB_Web_BlockPopups : Block popup windows.
#PB_EventType_PopupWindow is fired if this setting is enabled (Windows only).
#PB_Web_BlockPopupMenu : Block standard the popup menu.
#PB_EventType_PopupMenu is fired if this setting is enabled (Windows only).

The Navigation callback must have the following format: (Windows only)

```

1  Procedure NavigationCallback(Gadget, Url$)
2      ;
3      ; Return #True to allow this navigation or #False to deny it.
4      ;
5      ProcedureReturn #True
6  EndProcedure
  
```

- **GetGadgetAttribute()** : Get the following attributes (Windows only):

#PB_Web_ScrollX : Get the horizontal scrolling position.
#PB_Web_ScrollY : Get the vertical scrolling position.
#PB_Web_Busy : Returns nonzero if the gadget is busy loading a page.
#PB_Web_Progress : Returns the current (sometimes estimated) progress after a **#PB_EventType_DownloadProgress** event.
#PB_Web_ProgressMax : Returns the current (sometimes estimated) maximum progress after a **#PB_EventType_DownloadProgress** event.
#PB_Web_BlockPopups : Get the current popupwindow blocking setting.
#PB_Web_BlockPopupMenu: Get the current popupmenu blocking setting.
#PB_Web_NavigationCallback: Get the current navigation callback (if any).

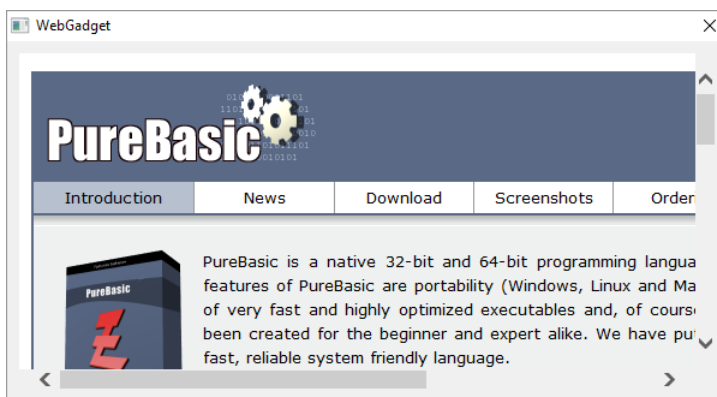
The following types of events can happen for this gadget:

#PB_EventType_TitleChange : The page title changed (Windows only).
#PB_EventType_StatusChange : The status message changed (Windows only).
#PB_EventType_DownloadStart : A page download started (Windows, OS X).
#PB_EventType_DownloadProgress: Progress info is available with **GetGadgetAttribute()** (Windows only).
#PB_EventType_DownloadEnd : A page download ended or aborted (Windows, OS X).

```
#PB_EventType_PopupWindow      : A popup window was blocked
                                (Windows only).
#PB_EventType_PopupMenu        : The popup menu was blocked
                                (display a custom menu here) (Windows only).
```

Example

```
1  If OpenWindow(0, 0, 0, 600, 300, "WebGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      WebGadget(0, 10, 10, 580, 280, "https://www.purebasic.com")
3      ; Note: if you want to use a local file, change last parameter
    to "file://" + path + filename
4      Repeat
5          Until WaitWindowEvent() = #PB_Event_CloseWindow
6      EndIf
```



Example 2: (Windows only, with navigation callback)

```
1  ; This example does display the PureBasic.com website. Inside the
    callback procedure
2  ; the navigation to the 'News' site will be avoided (#False
    returned), but allowed
3  ; for all other sites (#True returned).
4
5  Procedure NavigationCallback(Gadget, Url$) ;Windows only
6      If Url$= "https://www.purebasic.com/news.php"
7          MessageRequester("", "No news today!")
8          ProcedureReturn #False
9      Else
10         ProcedureReturn #True
11     EndIf
12 EndProcedure
13
14 If OpenWindow(0, 0, 0, 600, 300, "WebGadget",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
15     WebGadget(0, 10, 10, 580, 280, "https://www.purebasic.com")
16     SetGadgetAttribute(0, #PB_Web_NavigationCallback,
        @NavigationCallback())
17     Repeat
18         Until WaitWindowEvent() = #PB_Event_CloseWindow
19     EndIf
```


See Also

GetGadgetText() , SetGadgetText() , GetGadgetItemText() , SetGadgetItemText() ,
SetGadgetState() , GetGadgetAttribute() , SetGadgetAttribute()

121.83 BindGadgetEvent

Syntax

```
BindGadgetEvent(#Gadget, @Callback() [, EventType])
```

Description

Bind a gadget event to a callback. It's an additional way to handle events in PureBasic, which works without problem with the regulars WindowEvent() / WaitWindowEvent() commands. It also allows to have real-time event notifications as the callback can be invoked as soon as the event occurs (useful for ScrollBarGadget() , ScrollAreaGadget() etc.). A gadget event can be unbound with UnbindGadgetEvent() .

Parameters

#Gadget The gadget to bind the event to.

@Callback() The callback procedure to call when the event occurs. It has to be declared like this:

```
1 Procedure EventHandler()  
2 ; Code  
3 EndProcedure
```

Regular functions like EventGadget() , EventWindow() , EventMenu() , EventType() and EventData() are available within the callback to get more information about the event.

Note: WindowEvent() and WaitWindowEvent() should never be called from inside the callback or the program can be locked or have wrong behavior.

EventType (optional) The event type to bind the event to. For a full list of supported types, see EventType() . **#PB_All** can be used to bind the event to any type.

Return value

None.

Example

```
1 Procedure ButtonHandler()  
2 Debug "Button click event on gadget #" + EventGadget()  
3 EndProcedure  
4  
5 OpenWindow(0, 100, 100, 200, 90, "Click test",  
6 #PB_Window_SystemMenu)  
7  
8 ButtonGadget(0, 10, 10, 180, 30, "Click me")  
9 BindGadgetEvent(0, @ButtonHandler())  
10  
11 ButtonGadget(1, 10, 50, 180, 30, "Click me")  
12 BindGadgetEvent(1, @ButtonHandler())
```

```

13 Repeat
14     Event = WaitWindowEvent()
15 Until Event = #PB_Event_CloseWindow

```

See Also

BindEvent() , BindMenuEvent() , UnbindGadgetEvent() , WindowEvent() , WaitWindowEvent()

121.84 UnbindGadgetEvent

Syntax

```
UnbindGadgetEvent(#Gadget, @Callback() [, EventType])
```

Description

Unbind a gadget event from a callback. If no matching event callback is found, this command has no effect.

Parameters

#Gadget The gadget to unbind the event.

@Callback() The callback procedure to unbind.

EventType (optional) The event type to unbind the event from. For a full list of supported types, see EventType() .

Return value

None.

Example

```

1  Procedure ButtonHandler()
2      Debug "Button click event on gadget #" + EventGadget()
3
4      UnbindGadgetEvent(0, @ButtonHandler()) ; Unbind it immediately
5      Debug "Event unbound"
6  EndProcedure
7
8  OpenWindow(0, 100, 100, 200, 50, "Click test",
9      #PB_Window_SystemMenu)
10
11  ButtonGadget(0, 10, 10, 180, 30, "Click me")
12
13  BindGadgetEvent(0, @ButtonHandler())
14
15  Repeat
16      Event = WaitWindowEvent()
17  Until Event = #PB_Event_CloseWindow

```

See Also

BindEvent() , BindGadgetEvent() , BindMenuEvent() , WindowEvent() , WaitWindowEvent()

Chapter 122

Gadget3D

Overview

The Gadget3D library allows to create complex Graphical User Interface (GUI) directly over the screen area, using the 3D engine. This is mainly intended for game or application which needs user inputs while running in fullscreen mode. This library is based on the regular PureBasic gadget library, and offer similar syntax and behavior. The GUI engine used is CEGUI 0.7.7, which offer some nice options like skinning, good speed and a lot of built-in gadgets. More information about CEGUI can be found here: <http://www.cebui.org.uk>.

It uses the 3D engine, so `InitEngine3D()` has to be called successfully before using these functions. Before using gadgets there will be normally opened a window first.

122.1 AddGadgetItem3D

Syntax

```
AddGadgetItem3D(#Gadget3D, Position, Text$)
```

Description

Add an item to the specified #Gadget3D.

Parameters

#Gadget3D The 3D gadget to use.

Position The item index where the new item should be inserted. To add the item at the start, use a value of 0. To add this item to the end of the current item list, use a value of -1.

Remember that when you add an item that all current items which come after the new one will have their positions increased by 1.

Text\$ The new item text.

Return value

None.

Remarks

The following gadgets are supported:

- `ComboBoxGadget3D()`
- `ListViewGadget3D()`
- `PanelGadget3D()`

See Also

ComboBoxGadget3D() , ListViewGadget3D() , PanelGadget3D()

122.2 ButtonGadget3D

Syntax

```
Result = ButtonGadget3D(#Gadget3D, x, y, Width, Height, Text$)
```

Description

Creates a button gadget in the current GadgetList.

Parameters

#Gadget3D A number to identify the new 3D gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Text\$ The text to display on the button.

Return value

Nonzero on success, zero otherwise. If #PB_Any was used as the #Gadget3D parameter then the return-value is the auto-generated gadget number on success.

Remarks

To add a 'mini help' to this gadget, use GadgetToolTip3D() .

The following functions can be used to control the gadget:

- SetGadgetText3D() : Changes the text of the ButtonGadget.
- GetGadgetText3D() : Returns the text of the ButtonGadget.

See Also

GadgetToolTip3D() , SetGadgetText3D() , GetGadgetText3D()

122.3 CheckBoxGadget3D

Syntax

```
Result = CheckBoxGadget3D(#Gadget3D, x, y, Width, Height, Text$)
```

Description

Creates a checkbox gadget in the current GadgetList.

Parameters

#Gadget3D A number to identify the new 3D gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Text\$ The text to display at the right of the checkbox.

Return value

Nonzero on success, zero otherwise. If `#PB_Any` was used as the `#Gadget3D` parameter then the return-value is the auto-generated gadget number on success.

Remarks

To add a 'mini help' to this gadget, use `GadgetToolTip3D()` .

The following functions can be used to control the gadget:

- `GetGadgetState3D()` can be used to get the current gadget state (checked or unchecked).
- `SetGadgetState3D()` can be used to change the gadget state (checked or unchecked).

See Also

`GadgetToolTip3D()` , `GetGadgetState3D()` , `SetGadgetState3D()`

122.4 ClearGadgetItems3D

Syntax

```
ClearGadgetItems3D(#Gadget3D)
```

Description

Clears all the items from the specified `#Gadget3D`.

Parameters

`#Gadget3D` The 3D gadget to use.

Return value

None.

Remarks

The Gadget must be one of the following types:

- `ComboBoxGadget3D()`
- `ListViewGadget3D()`
- `PanelGadget3D()`

See Also

`AddGadgetItem3D()`

122.5 CloseGadgetList3D

Syntax

```
CloseGadgetList3D()
```

Description

Closes the current gadget list creation and go back to the previous `GadgetList`.

Parameters

None.

Return value

None.

Remarks

This function is supported by the following 3D gadgets:

- ContainerGadget3D()
- PanelGadget3D()
- ScrollAreaGadget3D()

122.6 ComboBoxGadget3D

Syntax

```
Result = ComboBoxGadget3D(#Gadget3D, x, y, Width, Height [, Flags])
```

Description

Creates a ComboBox gadget in the current GadgetList. Once a ComboBox is created, its list of items is empty.

Parameters

#Gadget3D A number to identify the new 3D gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Flags (optional) It can be a combination (using the '||' operator) of the following constants:

```
#PB_ComboBox3D_Editable : Makes the ComboBox editable
```

Return value

Nonzero on success, zero otherwise. If **#PB_Any** was used as the **#Gadget3D** parameter then the return-value is the auto-generated gadget number on success.

Remarks

To add a 'mini help' to this gadget, use `GadgetToolTip3D()` .

The following functions can be used to act on the list contents:

- `AddGadgetItem3D()` : Add an item.
- `GetGadgetItemText3D()` : Returns the gadget item text.
- `CountGadgetItems3D()` : Count the items in the current combobox.
- `ClearGadgetItems3D()` : Remove all the items.
- `RemoveGadgetItem3D()` : Remove an item.
- `SetGadgetItemText3D()` : Changes the gadget item text.
- `GetGadgetState3D()` : Get the index (starting from 0) of the current element.
- `GetGadgetText3D()` : Get the (text) content of the current element.
- `SetGadgetState3D()` : Change the selected element.
- `SetGadgetText3D()` : Set the displayed text. If the `ComboBoxGadget` is not editable, the text must be in the dropdown list.

122.7 ContainerGadget3D

Syntax

```
Result = ContainerGadget3D(#Gadget3D, x, y, Width, Height)
```

Description

Creates a container gadget in the current GadgetList. It's a simple panel gadget which can contain other gadgets. Once the gadget is created, all future created gadgets will be created inside the container. When all the needed gadgets have been created, CloseGadgetList3D() must be called to return to the previous GadgetList. OpenGadgetList3D() can be used later to add others gadgets on the fly in the container area.

Parameters

#Gadget3D A number to identify the new 3D gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Return value

Nonzero on success, zero otherwise. If #PB_Any was used as the #Gadget3D parameter then the return-value is the auto-generated gadget number on success.

Remarks

To add a 'mini help' to this gadget, use GadgetToolTip3D() .

122.8 CountGadgetItems3D

Syntax

```
Result = CountGadgetItems3D(#Gadget3D)
```

Description

Returns the number of items in the specified #Gadget3D.

Parameters

#Gadget3D The 3D gadget to use.

Return value

The number of items in the specified #Gadget3D.

Remarks

This function is supported by the following 3D gadgets:

- ComboBoxGadget3D()
- ListViewGadget3D()
- PanelGadget3D()

122.9 DisableGadget3D

Syntax

```
DisableGadget3D(#Gadget3D, State)
```

Description

Disable or enable the gadget.

Parameters

#Gadget3D The 3D gadget to enable or disable.

State The new state of the gadget. If State = 1, the gadget will be disabled, if State = 0 it will be enabled.

Return value

None.

122.10 EditorGadget3D

Syntax

```
Result = EditorGadget3D(#Gadget3D, x, y, Width, Height [, Flags])
```

Description

Creates an editor gadget in the current GadgetList.

Parameters

#Gadget3D A number to identify the new 3D gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Flags (optional) It can be one of the following value:

#PB_Editor3D_ReadOnly: The user cannot edit the text in the gadget.

Return value

Nonzero on success, zero otherwise. If **#PB_Any** was used as the **#Gadget3D** parameter then the return-value is the auto-generated gadget number on success.

Remarks

To add a 'mini help' to this gadget, use `GadgetToolTip3D()`.

The following functions can be used to act on the editor content:

- `GetGadgetText3D()` : Get the text content of the editor gadget.
- `SetGadgetText3D()` : Change the text content of the editor gadget.
- `SetGadgetAttribute3D()` : With the following attribute:

`#PB_Editor3D_ReadOnly`: Set the `read-only` state. (0 means editable, nonzero means `read-only`)

- `GetGadgetAttribute3D()` : With the following attribute:

`#PB_Editor3D_ReadOnly`: Get the `read-only` state. (0 means editable, nonzero means `read-only`)

122.11 FrameGadget3D

Syntax

```
Result = FrameGadget3D(#Gadget3D, x, y, Width, Height, Text$)
```

Description

Creates a frame gadget in the current GadgetList. This kind of gadget is decorative only.

Parameters

#Gadget3D A number to identify the new 3D gadget. `#PB_Any` can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Text\$ (optional) The text to display in the frame.

Return value

Nonzero on success, zero otherwise. If `#PB_Any` was used as the `#Gadget3D` parameter then the return-value is the auto-generated gadget number on success.

Remarks

As this Gadget is decorative only, `GadgetToolTip3D()` cannot be used. This Gadget also receives no events.

122.12 FreeGadget3D

Syntax

```
FreeGadget3D(#Gadget3D)
```

Description

Free and remove the 3D gadget from the display (and free its gadgetlist if the gadget was a container).

Parameters

#Gadget3D The 3D gadget to free. If `#PB_All` is specified, all the remaining 3D gadgets are freed.

Return value

None.

Remarks

All remaining 3D gadgets are automatically freed when the program ends.

122.13 GadgetID3D

Syntax

```
GadgetID = GadgetID3D(#Gadget3D)
```

Description

Returns the unique system identifier of the 3D gadget.

Parameters

#Gadget3D The 3D gadget to use.

Return value

Returns the unique system identifier of the 3D gadget.

122.14 GadgetToolTip3D

Syntax

```
GadgetToolTip3D(#Gadget3D, Text$)
```

Description

Associates the specified Text\$ with the 3D gadget. A tooltip text is text which is displayed when the mouse cursor is over the gadget for a small amount of time (typically a yellow floating box).

Parameters

#Gadget3D The 3D gadget to use.

Text\$ The text to display in the tooltip.

Return value

None.

Remarks

This function is supported by the following 3D gadgets:

- ButtonGadget3D()
- CheckBoxGadget3D()
- ComboBoxGadget3D()
- ContainerGadget3D()
- EditorGadget3D()
- ImageGadget3D()
- ListViewGadget3D()
- OptionGadget3D()
- PanelGadget3D()

- ProgressBarGadget3D()
- ScrollBarGadget3D()
- SpinGadget3D()
- StringGadget3D()

122.15 GadgetX3D

Syntax

```
Result = GadgetX3D(#Gadget3D)
```

Description

Returns the X position of the specified 3D gadget.

Parameters

#Gadget3D The 3D gadget to use.

Return value

The X position, in pixels, of the specified 3D gadget starting from the inside of the left edge of the gadget list that the gadget is in.

See Also

GadgetY3D()

122.16 GadgetY3D

Syntax

```
Result = GadgetY3D(#Gadget3D)
```

Description

Returns the Y position of the specified 3D gadget.

Parameters

#Gadget3D The 3D gadget to use.

Return value

The Y position, in pixels, of the specified 3D gadget starting from the inside of the top edge of the gadget list that the gadget is in.

See Also

GadgetX3D()

122.17 GadgetHeight3D

Syntax

```
Result = GadgetHeight3D(#Gadget3D)
```

Description

Returns the height of the specified 3D gadget.

Parameters

#Gadget3D The 3D gadget to use.

Return value

The height, in pixels, of the specified 3D gadget starting from the inside of the top edge of the gadget list that the gadget is in.

See Also

GadgetWidth3D()

122.18 GadgetType3D

Syntax

```
Result = GadgetType3D(#Gadget3D)
```

Description

Returns the type of gadget that is represented by the specified 3D gadget. It can be useful to write generic functions that work with more than one type of gadget.

Parameters

#Gadget3D The 3D gadget to use.

Return value

It can be one of the following values:

<code>#PB_GadgetType3D_Button</code>	<code>: ButtonGadget3D()</code>
<code>#PB_GadgetType3D_CheckBox</code>	<code>: CheckBoxGadget3D()</code>
<code>#PB_GadgetType3D_ComboBox</code>	<code>: ComboBoxGadget3D()</code>
<code>#PB_GadgetType3D_Container</code>	<code>: ContainerGadget3D()</code>
<code>#PB_GadgetType3D_Editor</code>	<code>: EditorGadget3D()</code>
<code>#PB_GadgetType3D_Frame</code>	<code>: FrameGadget3D()</code>
<code>#PB_GadgetType3D_Image</code>	<code>: ImageGadget3D()</code>

<code>#PB_GadgetType3D_ListView</code>	: <code>ListViewGadget3D()</code>
<code>#PB_GadgetType3D_Option</code>	: <code>OptionGadget3D()</code>
<code>#PB_GadgetType3D_Panel</code>	: <code>PanelGadget3D()</code>
<code>#PB_GadgetType3D_ProgressBar</code>	: <code>ProgressBarGadget3D()</code>
<code>#PB_GadgetType3D_ScrollArea</code>	: <code>ScrollAreaGadget3D()</code>
<code>#PB_GadgetType3D_ScrollBar</code>	: <code>ScrollBarGadget3D()</code>
<code>#PB_GadgetType3D_Spin</code>	: <code>SpinGadget3D()</code>
<code>#PB_GadgetType3D_String</code>	: <code>StringGadget3D()</code>
<code>#PB_GadgetType3D_Text</code>	: <code>TextGadget3D()</code>
<code>#PB_GadgetType3D_Unknown</code>	: The type is unknown. Most likely it is not a PB gadget at all.

122.19 GadgetWidth3D

Syntax

```
Result = GadgetWidth3D(#Gadget3D)
```

Description

Returns the width of the specified 3D gadget.

Parameters

#Gadget3D The 3D gadget to use.

Return value

The width, in pixels, of the specified 3D gadget starting from the inside of the top edge of the gadget list that the gadget is in.

See Also

`GadgetHeight3D()`

122.20 GetActiveGadget3D

Syntax

```
Result = GetActiveGadget3D()
```

Description

Returns the `#Gadget3D` number of the Gadget that currently has the keyboard focus.

Parameters

None.

Return value

The 3D gadget number that currently has the keyboard focus. If no 3D gadget has the focus, -1 is returned.

See Also

SetActiveGadget3D()

122.21 GetGadgetAttribute3D

Syntax

```
Value = GetGadgetAttribute3D(#Gadget3D, Attribute)
```

Description

Gets the attribute value of the specified 3D gadget.

Parameters

#Gadget3D The 3D gadget to use.

Attribute The attribute to get.

Return value

The attribute value of the specified 3D gadget.

Remarks

This function is supported by the following 3D gadgets:

- EditorGadget3D()
- PanelGadget3D()
- ProgressBarGadget3D()
- ScrollAreaGadget3D()
- ScrollBarGadget3D()
- SpinGadget3D()

122.22 GetGadgetData3D

Syntax

```
Result = GetGadgetData3D(#Gadget3D)
```

Description

Returns the data value that has been stored for this gadget with the SetGadgetData3D() function. This allows to associate a custom value with any gadget.

Parameters

#Gadget3D The 3D gadget to use.

Return value

The current data value associated with the 3D gadget, or zero if no value has been set.

Remarks

This function is supported by all 3D gadgets. See `SetGadgetData3D()` for an example.

See Also

`SetGadgetData3D()`

122.23 GetGadgetItemData3D

Syntax

```
Result = GetGadgetItemData3D(#Gadget3D, Item)
```

Description

Returns the value that was previously stored with this 3D gadget item with the `SetGadgetItemData3D()` function. This allows to associate a custom value with the items of a gadget.

Parameters

#Gadget3D The 3D gadget to use.

Item The item to get the data value. First item index starts from zero.

Return value

The current data value associated with the 3D gadget item, or zero if no value has been set.

Remarks

This function is supported by the following 3D gadgets:

- `ComboBoxGadget3D()`
- `ListViewGadget3D()`

See `SetGadgetItemData3D()` for an example.

See Also

`SetGadgetItemData3D()`

122.24 GetGadgetState3D

Syntax

```
Result = GetGadgetState3D(#Gadget3D)
```

Description

Returns the current state of the 3D gadget.

Parameters

#Gadget3D The 3D gadget to use.

Return value

The current state of the 3D gadget.

Remarks

This function is supported by the following 3D gadgets:

- CheckBoxGadget3D() : returns 1 if checked, 0 otherwise.
- ComboBoxGadget3D() : returns the currently selected item index, -1 if none is selected.
- ImageGadget3D() : returns the TextureID of the currently displayed image.
- ListViewGadget3D() : returns the currently selected item index, -1 if none is selected.
- OptionGadget3D() : returns 1 if activated, 0 otherwise.
- PanelGadget3D() : returns the current panel index, -1 if no panel.
- ProgressBarGadget3D() : returns the current value of the ProgressBar.
- ScrollBarGadget3D() : returns the current slider position.
- SpinGadget3D() : returns the current value of the SpinGadget.

See Also

SetGadgetState3D()

122.25 GetGadgetItemText3D

Syntax

```
Result\$$ = GetGadgetItemText3D(#Gadget3D, Item [, Column])
```

Description

Returns the item text of the specified 3D gadget item.

Parameters

#Gadget3D The 3D gadget to use.

Item The item to get the text. First item index starts from zero.

Column (optional) Not used for now.

Return value

The item text of the specified 3D gadget item.

Remarks

This function is supported by the following 3D gadgets:

- `ComboBoxGadget3D()` - 'Item' is the index of the item in the ComboBox list. 'Column' will be ignored.
- `ListViewGadget3D()` - 'Item' is the index of the entry from which you want to receive the content. 'Column' will be ignored.
- `PanelGadget3D()` - 'Item' is the panel from which you want to receive the header text.

See Also

`SetGadgetItemText3D()`

122.26 GetGadgetItemState3D

Syntax

```
Result = GetGadgetItemState3D(#Gadget3D, Item)
```

Description

Returns the item state of the specified 3D gadget item.

Parameters

#Gadget3D The 3D gadget to use.

Item The item to get the state. First item index starts from zero.

Return value

The item state of the specified 3D gadget item.

Remarks

This function is supported by the following 3D gadgets:

- `ListViewGadget3D()` : returns 1 if the item is selected, 0 otherwise.

See Also

`SetGadgetItemState3D()`

122.27 GetGadgetText3D

Syntax

```
String\$ = GetGadgetText3D(#Gadget3D)
```

Description

Returns the text content of the specified 3D gadget.

Parameters

#Gadget3D The 3D gadget to use.

Return value

The text content of the specified 3D gadget.

Remarks

This function is supported by the following 3D gadgets:

- ButtonGadget3D() : return the text of the ButtonGadget.
- ComboBoxGadget3D() - return the content of the current item.
- EditorGadget3D() - return the text content of the editor gadget. Please note, that several lines of text are normally separated with "Chr(13)+Chr(10)".
- ListViewGadget3D() - return the content of the current item.
- StringGadget3D() - return contents of the StringGadget.
- TextGadget3D() - return contents of the TextGadget.

See Also

SetGadgetText3D()

122.28 HideGadget3D

Syntax

```
HideGadget3D(#Gadget3D, State)
```

Description

Hide or show a 3D gadget.

Parameters

#Gadget3D The 3D gadget to use.

State The new state of the 3D gadget. If State = 1, the gadget will be hidden, if State = 0 it will be shown.

Return value

None.

See Also

DisableGadget3D()

122.29 ImageGadget3D

Syntax

```
Result = ImageGadget3D(#Gadget3D, x, y, Width, Height, TextureID [,  
Flags])
```

Description

Creates an image gadget in the current GadgetList.

Parameters

#Gadget3D A number to identify the new 3D gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

TextureID The texture to display in the 3D gadget. **TextureID()** can be used to get a valid value. If this value is sets to zero, then the texture is removed from the gadget.

Flags (optional) It can be one of the following constants:

#PB_Image3D_Border : display a sunken border around the image.

Return value

Nonzero on success, zero otherwise. If **#PB_Any** was used as the **#Gadget3D** parameter then the return-value is the auto-generated gadget number on success.

Remarks

To add a 'mini help' to this gadget, use **GadgetToolTip3D()** .

The following function can be used to act on the gadget:

- **SetGadgetState3D()** : Change the current Image of the gadget. A valid **TextureID** can be easily obtained with the **TextureID()** function. If the **TextureID** is sets to zero, then the texture is removed from the gadget.

122.30 IsGadget3D

Syntax

```
Result = IsGadget3D(#Gadget3D)
```

Description

Tests if the given 3D gadget is valid and correctly initialized.

Parameters

#Gadget3D The 3D gadget to test.

Return value

Nonzero if the gadget is valid, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a gadget is ready to use.

122.31 ListViewGadget3D

Syntax

```
Result = ListViewGadget3D(#Gadget3D, x, y, Width, Height)
```

Description

Creates a ListView gadget in the current GadgetList. Once a ListView is created, its list of items is empty.

Parameters

#Gadget3D A number to identify the new 3D gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Return value

Nonzero on success, zero otherwise. If #PB_Any was used as the #Gadget3D parameter then the return-value is the auto-generated gadget number on success.

Remarks

To add a 'mini help' to this gadget, use GadgetToolTip3D() .

The following functions can be used to act on the list content:

- AddGadgetItem3D() : Add an item.
- RemoveGadgetItem3D() : Remove an item.
- ClearGadgetItems3D() : Remove all the items
- CountGadgetItems3D() : Returns the number of items currently in the #Gadget3D.
- GetGadgetItemData3D() : Get the value that was stored with the gadget item.
- GetGadgetItemState3D() : Returns nonzero if the item is selected, zero otherwise.
- GetGadgetItemText3D() : Get the content of the given item.
- GetGadgetState3D() : Get the index of the selected item or -1 if there is no selected item.
- GetGadgetText3D() : Get the content of the selected item.
- SetGadgetItemData3D() : store a value with the given item.
- SetGadgetItemState3D() : Selects or deselects the given item.
- SetGadgetItemText3D() : Set the text of the given item.
- SetGadgetState3D() : Change the selected item. If -1 is specified, the selection will be removed.
- SetGadgetText3D() : Selects the item with the given text (the text must match exactly).

122.32 OpenGadgetList3D

Syntax

```
OpenGadgetList3D(#Gadget3D [, Item])
```

Description

Use the specified 3D gadget as a GadgetList, to dynamically add new gadgets to it. Once the all the needed changes are done, CloseGadgetList3D() should be called.

Parameters

#Gadget3D The 3D gadget to use.

Item (optional) The 3D gadget item to use. This is only supported for PanelGadget3D() .

Remarks

This function is supported by the following 3D gadgets:

- ContainerGadget3D()
- PanelGadget3D() : The optional 'Item' parameter needs to be specified
- ScrollAreaGadget3D()

122.33 OptionGadget3D

Syntax

```
Result = OptionGadget3D(#Gadget3D, x, y, Width, Height, Text$)
```

Description

Creates an OptionGadget in the current GadgetList. The first time this function is called, a group is created and all following calls of OptionGadget3D() will add a gadget to this group. To finish the group, just create a gadget of another type. These kind of gadgets are very useful as only one gadget from the group can be selected at any time.

Parameters

#Gadget3D A number to identify the new 3D gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Text\$ The text to display, near the selection button.

Return value

Nonzero on success, zero otherwise. If #PB_Any was used as the #Gadget3D parameter then the return-value is the auto-generated gadget number on success.

Remarks

To add a 'mini help' to this gadget, use GadgetToolTip3D() .

122.34 PanelGadget3D

Syntax

```
Result = PanelGadget3D(#Gadget3D, x, y, Width, Height)
```

Description

Creates a panel gadget in the current GadgetList. Once a panel is created, its list of panels is empty.

Parameters

#Gadget3D A number to identify the new 3D gadget. #PB_Any can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Return value

Nonzero on success, zero otherwise. If `#PB_Any` was used as the `#Gadget3D` parameter then the return-value is the auto-generated gadget number on success.

Remarks

To add a 'mini help' to this gadget, use `GadgetToolTip3D()` .

The following functions can be used to act on the gadget:

- `AddGadgetItem3D()` : Add a panel.
- `RemoveGadgetItem3D()` : Remove a panel.
- `CountGadgetItems3D()` : Count the number of panels.
- `ClearGadgetItems3D()` : Remove all panels.
- `GetGadgetItemText3D()` : Retrieve the text of the specified item.
- `SetGadgetItemText3D()` : Change the text of the specified item.
- `SetGadgetState3D()` : Change the active panel.
- `GetGadgetState3D()` : Get the index of the current panel.
- `GetGadgetAttribute3D()` : With one of the following attributes: (there must be at least 1 item for this to work)

```
#PB_Panel3D_ItemWidth : Returns the width of the inner area where
                        the gadgets are displayed.
#PB_Panel3D_ItemHeight: Returns the height of the inner area
                        where the gadgets are displayed.
#PB_Panel3D_TabHeight : Returns height of the panel tabs on top
                        of the gadget.
```

You must call `AddGadgetItem3D()` to add a panel before you can create other gadgets inside the panel gadget. The next gadgets will then be created automatically in the new panel. This is very convenient. When the `PanelGadget` item has been filled with all the needed gadgets, `CloseGadgetList3D()` must be called to return to the previous `GadgetList`. This means that a `PanelGadget` can be created inside another `PanelGadget`.

122.35 ProgressBarGadget3D

Syntax

```
Result = ProgressBarGadget3D(#Gadget3D, x, y, Width, Height,
                             Minimum, Maximum)
```

Description

Creates a `ProgressBar` gadget in the current `GadgetList`.

Parameters

#Gadget3D A number to identify the new 3D gadget. `#PB_Any` can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Minimum, Maximum The minimum and maximum values that the progress bar can take.

Return value

Nonzero on success, zero otherwise. If `#PB_Any` was used as the `#Gadget3D` parameter then the return-value is the auto-generated gadget number on success.

Remarks

To add a 'mini help' to this gadget, use `GadgetToolTip3D()` .

The following functions can be used to act on the gadget:

- `SetGadgetState3D()` : Change progress bar value.
- `GetGadgetState3D()` : Get the current progress bar value.
- `SetGadgetAttribute3D()` : With the following attributes:

```
#PB_ProgressBar3D_Minimum : Changes the minimum value.  
#PB_ProgressBar3D_Maximum : Changes the maximum value.
```

- `GetGadgetAttribute3D()` : With the following attributes:

```
#PB_ProgressBar3D_Minimum : Returns the minimum value.  
#PB_ProgressBar3D_Maximum : Returns the maximum value.
```

122.36 RemoveGadgetItem3D

Syntax

```
RemoveGadgetItem3D(#Gadget3D, Position)
```

Description

Removes an item of the specified `#Gadget3D` at the given `Position`.

Parameters

#Gadget3D The 3D gadget to use.

Position The item position to remove. The first position index starts from zero.

Return value

None.

Remarks

Supported gadgets:

- `ComboBoxGadget3D()`
- `PanelGadget3D()`
- `ListViewGadget3D()`

122.37 ResizeGadget3D

Syntax

```
ResizeGadget3D(#Gadget3D, x, y, Width, Height)
```

Description

Resize the specified 3D gadget to the given position and dimensions.

Parameters

#Gadget The gadget to resize.

x, y, Width, Height The new position and dimensions of the gadget. To ease the building of real-time resizable Graphical User Interfaces (GUIs), **#PB_Ignore** can be passed as any parameter (x, y, Width or Height) and this parameter will not be changed.

Return value

None.

Example

```
1  [...]
2
3  ResizeGadget3D(0, #PB_Ignore, #PB_Ignore, 300, #PB_Ignore) ; Only
    change the gadget width.
```

122.38 ScrollBarGadget3D

Syntax

```
Result = ScrollBarGadget3D(#Gadget3D, x, y, Width, Height, Minimum,
    Maximum, PageLength [, Flags])
```

Description

Creates a scrollbar gadget in the current GadgetList. It's widely used when displaying only a part of an object.

Parameters

#Gadget3D A number to identify the new 3D gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Minimum, Maximum The range of values that the scrollbar can take. These values should be between 0 and 10,000 to avoid limitations on some operating systems.

PageLength The amount of values that are part of the current displayed "page".

For example you can have a picture which is 100 pixels width and you only see 25 pixels.

What you see is called a 'page', in this example, the page length will be 25, the Minimum will be 0 and the Maximum will be 100.

Flags (optional) It can be a combination of the following values:

#PB_ScrollBar3D_Vertical : The scrollbar is vertical (instead of horizontal, which is the default).

Return value

Nonzero on success, zero otherwise. If **#PB_Any** was used as the **#Gadget3D** parameter then the return-value is the auto-generated gadget number on success.

Remarks

To add a 'mini help' to this gadget, use `GadgetToolTip3D()` .

The following functions can be used to act on this gadget:

- `GetGadgetState3D()` : Returns the current slider position (value between the Minimum-Maximum range).

- `SetGadgetState3D()` : Changes the current slider position.

- `GetGadgetAttribute3D()` : With one of the following attributes:

- `#PB_ScrollBar3D_Minimum` : Returns the minimum scroll position.
 - `#PB_ScrollBar3D_Maximum` : Returns the maximum scroll position.
 - `#PB_ScrollBar3D_PageLength`: Returns the `PageLength` value.

- `SetGadgetAttribute3D()` : With one of the following attributes:

- `#PB_ScrollBar3D_Minimum` : Changes the minimum scroll position.
 - `#PB_ScrollBar3D_Maximum` : Changes the maximum scroll position.
 - `#PB_ScrollBar3D_PageLength`: Changes the `PageLength` value.

122.39 ScrollAreaGadget3D

Syntax

```
Result = ScrollAreaGadget3D(#Gadget3D, x, y, Width, Height,  
    ScrollAreaWidth, ScrollAreaHeight, ScrollStep)
```

Description

Creates a `ScrollArea` gadget in the current `GadgetList`. It's very useful when a gadget is too big to fit the window dimension. In that case, it can be put into a scrollarea. All the scrolling is handled automatically by the gadget. This is a container gadget, intended to have one or several gadget in its scroll area. Once the gadget is created, all future created gadgets will be created inside the scroll area. When all the needed gadgets have been created, `CloseGadgetList3D()` must be called to return to the previous `GadgetList`. `OpenGadgetList3D()` can be used later to add others gadgets on the fly in the scroll area.

Parameters

#Gadget3D A number to identify the new 3D gadget. `#PB_Any` can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

ScrollAreaWidth, ScrollAreaHeight The dimensions of the scrollable area inside the gadget. These can also be smaller than the outer dimensions, in this case scrolling will be disabled.

ScrollStep (optional) The amount of pixels to scroll when the user presses the scroll bar arrows.

Return value

Nonzero on success, zero otherwise. If `#PB_Any` was used as the `#Gadget3D` parameter then the return-value is the auto-generated gadget number on success.

Remarks

The following functions can be used to act on a ScrollAreaGadget:

GetGadgetAttribute3D() : With one of the following attribute:

```
#PB_ScrollArea3D_InnerWidth : Returns the width (in pixels) of
the contained scrollable area.
#PB_ScrollArea3D_InnerHeight : Returns the height (in pixels) of
the contained scrollable area.
#PB_ScrollArea3D_X : Returns the current horizontal
scrolling position (in pixels).
#PB_ScrollArea3D_Y : Returns the current vertical
scrolling position (in pixels).
```

SetGadgetAttribute3D() : With one of the following attribute:

```
#PB_ScrollArea3D_InnerWidth : Changes the width (in pixels) of
the contained scrollable area.
#PB_ScrollArea3D_InnerHeight : Changes the height (in pixels) of
the contained scrollable area.
#PB_ScrollArea3D_X : Changes the current horizontal
scrolling position (in pixels).
#PB_ScrollArea3D_Y : Changes the current vertical
scrolling position (in pixels).
```

122.40 SetActiveGadget3D

Syntax

```
SetActiveGadget3D(#Gadget3D)
```

Description

Activates (sets the focus on) the gadget specified by the given #Gadget3D number. This is mainly used with ComboBoxGadget3D() and StringGadget3D() . Activating a gadget allows it to become the current object to receive messages and handle keystrokes.

Parameters

#Gadget3D The gadget to activate. If sets to -1, then the focus is removed (if any).

Return value

None.

See Also

GetActiveGadget3D()

122.41 SetGadgetAttribute3D

Syntax

```
SetGadgetAttribute3D(#Gadget3D, Attribute, Value)
```

Description

Changes the attribute value of the specified 3D gadget.

Parameters

#Gadget3D The gadget to use.

Attribute The attribute to set.

Value The new attribute value.

Return value

None.

Remarks

This function is available for all gadgets which support attributes:

- EditorGadget3D()
- ProgressBarGadget3D()
- ScrollAreaGadget3D()
- ScrollBarGadget3D()
- SpinGadget3D()

122.42 SetGadgetData3D

Syntax

```
SetGadgetData3D(#Gadget3D , Value)
```

Description

Stores the given value with the specified #Gadget3D. This value can later be read with the GetGadgetData3D() function. This allows to associate a custom value with any gadget.

Parameters

#Gadget3D The gadget to use.

Value The new value to associate with the 3D gadget.

Return value

None.

Remarks

This function is supported by all PureBasic gadgets.

See Also

GetGadgetData3D()

122.43 SetGadgetItemData3D

Syntax

```
SetGadgetItemData3D(#Gadget3D, Item, Value)
```

Description

Stores the given value with the specified #Gadget3D item. This value can later be read with the GetGadgetItemData3D() function. This allows to associate a custom value with the items of a gadget. This value will remain with the item, even if the item changes its index (for example because other items were deleted).

Parameters

#Gadget3D The gadget to use.

Item The item to set the data value. First item index starts from zero.

Value The new value to associate with the 3D gadget item.

Return value

None.

Remarks

This function is supported by the following 3D gadgets:

- ComboBoxGadget3D()
- ListViewGadget3D()

See Also

GetGadgetItemData3D()

122.44 SetGadgetItemState3D

Syntax

```
SetGadgetItemState3D(#Gadget3D, Item, State)
```

Description

Changes the item state of the specified #Gadget3D.

Parameters

#Gadget3D The gadget to use.

Item The item to set the state. First item index starts from zero.

State The new state of the 3D gadget item. Its value depends of the gadget type:
- ListViewGadget3D() : 1 if the item should be selected, 0 otherwise.

Return value

None.

122.45 SetGadgetItemText3D

Syntax

```
SetGadgetItemText3D(#Gadget3D, Item, Text$ [, Column])
```

Description

Changes the item text of the specified #Gadget3D.

Parameters

#Gadget3D The gadget to use.

Item The item to set the text. First item index starts from zero.

Text\$ The new text for the item.

Column (optional) Not used for now.

Return value

None.

Remarks

This function is supported by the following 3D gadgets:

- ComboBoxGadget3D()
- ListViewGadget3D()
- PanelGadget3D()

122.46 SetGadgetState3D

Syntax

```
SetGadgetState3D(#Gadget3D, State)
```

Description

Changes the current state of the specified #Gadget3D.

Parameters

#Gadget3D The gadget to use.

State The new 3D gadget state.

Return value

None.

Remarks

This function is supported by the following 3D gadgets:

- `CheckBoxGadget3D()` : 1 to check it, 0 otherwise.
- `ComboBoxGadget3D()` : change the currently selected item.
- `ImageGadget3D()` : change the current image of the gadget.
- `ListViewGadget3D()` : change the currently selected item. . If -1 is specified, it will remove the selection.
- `OptionGadget3D()` : 1 to activate it, 0 otherwise.
- `PanelGadget3D()` : change the current panel.
- `ProgressBarGadget3D()` : change progress bar value.
- `ScrollBarGadget3D()` : change the current slider position.
- `SpinGadget3D()` : change the current value.

122.47 SetGadgetText3D

Syntax

```
SetGadgetText3D(#Gadget3D , Text$)
```

Description

Change the gadget text content of the specified 3D gadget.

Parameters

#Gadget3D The gadget to use.

Text\$ The new 3D gadget text.

Return value

None.

Remarks

This function is supported by the following 3D gadgets:

- `ButtonGadget3D()` : change the text of the ButtonGadget.
- `ComboBoxGadget3D()` : Set the displayed text. If the ComboBoxGadget is not editable, the text must be in the dropdown list.
- `EditorGadget3D()` : change the text content of the editor gadget. If you want to add several lines of text at once, separate them with "Chr(13)+Chr(10)".
- `FrameGadget3D()` : change the title of the FrameGadget.
- `ListViewGadget3D()` : selects the item that exactly matches the given text.
- `StringGadget3D()` : change the content of the StringGadget.
- `TextGadget3D()` : change the content of the TextGadget.

122.48 SpinGadget3D

Syntax

```
Result = SpinGadget3D(#Gadget3D , x , y , Width , Height , Minimum ,  
Maximum)
```

Description

Creates a spin gadget in the current GadgetList.

Parameters

#Gadget3D A number to identify the new 3D gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Minimum, Maximum The minimum and maximum values that the SpinGadget can take.

Return value

Nonzero on success, zero otherwise. If **#PB_Any** was used as the **#Gadget3D** parameter then the return-value is the auto-generated gadget number on success.

Remarks

To add a 'mini help' to this gadget, use `GadgetToolTip3D()` .

The following functions can be used to act on the gadget:

`GetGadgetState3D()` : Get the current gadget value.

`SetGadgetState3D()` : Change the gadget value. For displaying the new value you still must use `SetGadgetText3D()` !

`GetGadgetText3D()` : Get the text contained in the gadget.

`SetGadgetText3D()` : Change the text contained in the gadget.

`GetGadgetAttribute3D()` : With one of the following attributes:

#PB_Spin3D_Minimum : Returns the minimum value.

#PB_Spin3D_Maximum : Returns the maximum value.

`SetGadgetAttribute3D()` : With one of the following attributes:

#PB_Spin3D_Minimum : Changes the minimum value.

#PB_Spin3D_Maximum : Changes the maximum value.

122.49 StringGadget3D

Syntax

```
Result = StringGadget3D(#Gadget3D, x, y, Width, Height, Content$ [,
    Flags])
```

Description

Creates a String gadget in the current GadgetList. This gadget accepts only one line of text. To get multi-line input, use the `EditorGadget3D()` function.

Parameters

#Gadget3D A number to identify the new 3D gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Content\$ The initial text content.

Flags (optional) It can be a combination of the following values:

#PB_String3D_Numeric : Only (positive) integer numbers are accepted.

#PB_String3D_Password : Password mode, displaying only '*' instead of normal characters.

#PB_String3D_ReadOnly : Read-only mode. No text can be entered.

Return value

Nonzero on success, zero otherwise. If `#PB_Any` was used as the `#Gadget3D` parameter then the return-value is the auto-generated gadget number on success.

Remarks

Later the content can be changed with `SetGadgetText3D()` and received with `GetGadgetText3D()` . The following events are supported through `EventType3D()` :

```
#PB_EventType3D_Change      : The text has been modified by the
    user .
#PB_EventType3D_Focus       : The StringGadget got the focus.
#PB_EventType3D_LostFocus   : The StringGadget lost the focus.
```

To add a 'mini help' to this gadget, use `GadgetToolTip3D()` .

122.50 TextGadget3D

Syntax

```
Result = TextGadget3D(#Gadget3D, x, y, Width, Height, Text$)
```

Description

Creates a text gadget in the current `GadgetList`. This is a basic text area for displaying, not entering, text.

Parameters

#Gadget3D A number to identify the new 3D gadget. `#PB_Any` can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Text\$ The text to display.

Return value

Nonzero on success, zero otherwise. If `#PB_Any` was used as the `#Gadget3D` parameter then the return-value is the auto-generated gadget number on success.

Remarks

The content can be changed with the function `SetGadgetText3D()` and can be obtained with `GetGadgetText3D()` .

This Gadget doesn't receive any user events, and `GadgetToolTip3D()` can't be used with it.

Chapter 123

Help

Overview

The help feature is a very important component for any type of software. It allows the end-users the opportunity to discover the software's features quickly and easily. The PureBasic help functions allow the programmer the ability to display standard help files with both global and contextual support.

On Microsoft Windows, there are two different help-file formats which are supported: The ".hlp" format (which is the old) and the ".chm" format (which is the new). The ".chm" format is based on HTML, which stands for Hyper Text Markup Language.

123.1 CloseHelp

Syntax

```
CloseHelp()
```

Description

Close the help window previously opened with `OpenHelp()` .

Parameters

None.

Return value

None.

See Also

`OpenHelp()`

123.2 OpenHelp

Syntax

```
OpenHelp(Filename$, Topic$)
```

Description

Open and display a help window.

Parameters

Filename\$ The name of either the .chm or .hlp file to be opened.

Topic\$ The internal name of the page to display, if the Topic\$ parameter is not an empty string, the OpenHelp function will display the specified page.

Return value

None.

Example

```
1  OpenHelp("help.chm", "index.htm")
2  OpenHelp("help.chm", "print.txt")
```

See Also

CloseHelp()

Chapter 124

Http

Overview

Http is the name of the protocol used by web browsers to access remote information, such as a web page. Each item has a unique address in order to access it from anywhere, this is its: URL (Uniform Resource Locator). The functions within this library are designed to provide easy manipulation of URLs and the capability to retrieve remote files.

On Linux, 'libcurl' needs to be installed to have the HTTP commands working (most of Linux distributions come with it already installed).

124.1 AbortHTTP

Syntax

```
AbortHTTP(HttpConnection)
```

Description

Aborts the progress of the specified asynchronous download, started either with `ReceiveHTTPFile()` or `ReceiveHTTPMemory()`. It's also usable for `HTTPRequest()` and `HTTPRequestMemory()` (if `#PB_HTTP_Aynchronous` was used).

Parameters

HttpConnection The HTTP connection to abort.

Return value

None.

Remarks

The value `#PB_HTTP_Aborted` will be returned by `HTTPProgress()`. `FinishHTTP()` has to be called once the download has been aborted.

See Also

`HTTPProgress()`, `FinishHTTP()`

124.2 FinishHTTP

Syntax

```
Result = FinishHTTP(URLConnection)
```

Description

Free the resources associated to the specified asynchronous download, started either with ReceiveHTTPFile() or ReceiveHTTPMemory(). It must be always called after a successful call of ReceiveHTTPFile() or ReceiveHTTPMemory().

Parameters

URLConnection The HTTP connection to finish.

Return value

Depending on which Receive command was used to start the download, FinishHTTP() returns one of the following results:

ReceiveHTTPFile() : Amount of received bytes (size of the stored file).

ReceiveHTTPMemory() with the `#PB_HTTP_Aynchronous` flag set: Address of the memory area where the received data was stored. With MemorySize() the amount of received data can be determined then.

Remarks

The value `#PB_HTTP_Aborted` will be returned by HTTPProgress().

See Also

HTTPProgress(), AbortHTTP()

124.3 GetURLPart

Syntax

```
Result\% = GetURLPart(URL$, Parameter%)
```

Description

Get a specific part of the given URL\$. This can be a named parameter in the URL string or another part of the URL.

Parameters

URL\$ The URL\$ to get the part from. The URL\$ may contain parameters, which are useful when a scripting language such as: (PHP) is being used on that server. The syntax is as follows: (http://www.purebasic.com/index.php3?test=1) Here, the parameter is named "test" and its associated value is "1".

Parameter\$ The parameter to return from the URL\$. The parameters are not case-sensitive. This parameter can also specify one of the following constants to access other parts of the URL:

```

#PB_URL_Protocol: returns the protocol from the URL$
#PB_URL_Site: returns the site from the URL$
#PB_URL_Port: returns the port from the URL$ (if specified)
#PB_URL_Parameters: returns all the parameters from the URL$
#PB_URL_Path: returns the path from the URL$
#PB_URL_User: returns the username from the URL$ (if
specified)
#PB_URL_Password: returns the password from the URL$ (if
specified)

```

Return value

Returns the parameter value or other part of the URL.

Example

```

1  URL$ =
   "http://user:pass@www.purebasic.com:80/index.php3?test=1&ok=2"
2
3  Debug GetURLPart(URL$, #PB_URL_Protocol) ; Will print "http"
4  Debug GetURLPart(URL$, #PB_URL_Site)     ; Will print
   "www.purebasic.com"
5  Debug GetURLPart(URL$, #PB_URL_Port)     ; Will print "80"
6  Debug GetURLPart(URL$, #PB_URL_Parameters) ; Will print
   "test=1&ok=2"
7  Debug GetURLPart(URL$, #PB_URL_Path)     ; Will print "index.php3"
8  Debug GetURLPart(URL$, #PB_URL_User)     ; Will print "user"
9  Debug GetURLPart(URL$, #PB_URL_Password) ; Will print "pass"
10 Debug GetURLPart(URL$, "test")           ; Will print "1"
11 Debug GetURLPart(URL$, "ok")             ; Will print "2"

```

See Also

SetURLPart() , URLDecoder()

124.4 HTTPProgress

Syntax

```
Result = HTTPProgress(HttpConnection)
```

Description

Returns the progress of the specified asynchronous download, started either with ReceiveHTTPFile() or ReceiveHTTPMemory() .

Parameters

HttpConnection The HTTP connection to monitor.

Return value

The status of the download. It can be the current number of received bytes or one of the following values:

```
#PB_Http_Success      : the download has been successfully finished.
#PB_Http_Failed       : the download has failed.
#PB_Http_Aborted      : the download has been aborted with
    AbortHTTP()
```

Example

```
1  Download =
   ReceiveHTTPMemory("http://www.purebasic.com/download/OgreAssimpConverter.zip"
   #PB_HTTP_Aynchronous)
2  If Download
3      Repeat
4          Progress = HTTPProgress(Download)
5          Select Progress
6              Case #PB_Http_Success
7                  *Buffer = FinishHTTP(Download)
8                  Debug "Download finished (size: " + MemorySize(*Buffer) +
   ") "
9                  FreeMemory(*Buffer)
10                 End
11
12             Case #PB_Http_Failed
13                 Debug "Download failed"
14                 FinishHTTP(Download)
15                 End
16
17             Case #PB_Http_Aborted
18                 Debug "Download aborted"
19                 FinishHTTP(Download)
20                 End
21
22             Default
23                 Debug "Current download: " + Progress
24
25             EndSelect
26
27             Delay(500) ; Don't stole the whole CPU
28         Forever
29     Else
30         Debug "Download error"
31     EndIf
```

See Also

ReceiveHTTPFile() , ReceiveHTTPMemory()

124.5 HTTPInfo

Syntax

```
Result\$ = HTTPInfo(HTTPRequest, Type [, Flags])
```

Description

Returns information about an HTTP request created with HTTPRequest() or HTTPRequestMemory() .

Parameters

HTTPRequest The HTTP request to get the info.

Type The specific info to get. It can be one of the following values:

```
#PB_HTTP_Statuscode : the server status code (200: OK, 404:
page not found etc.).
#PB_HTTP_Response   : the server response, as text. To get
the raw server response, use HTTPMemory()
.
#PB_HTTP_Headers     : the request headers.
#PB_HTTP_ErrorMessage: the HTTP request error message, if
something goes wrong (mostly for debugging purpose).
```

Flags (optional) The format of the HTTP response or header encoding. It can be one of the following values:

```
#PB_UTF8   (default)
#PB_Ascii
```

Return value

A string depending of the 'Type' parameter.

Example

```
1  HttpRequest = HTTPRequest(#PB_HTTP_Get, "https://www.google.com")
2  If HttpRequest
3      Debug "Response: " + HTTPInfo(HTTPRequest, #PB_HTTP_Response)
4      Debug "StatusCode: " + HTTPInfo(HTTPRequest,
#PB_HTTP_Statuscode)
5
6      FinishHTTP(HTTPRequest)
7  Else
8      Debug "Request creation failed"
9  EndIf
```

See Also

HTTPRequest() , HTTPRequestMemory()

124.6 HTTPMemory

Syntax

```
*Buffer = HTTPMemory(HTTPRequest)
```

Description

Returns a memory buffer containing the whole response of an HTTP request created with `HttpRequest()` or `HttpRequestMemory()` . Once done with it, the buffer needs to be freed with `FreeMemory()` . If the flag `#PB_HTTP_Aynchronous` was used when calling `HttpRequest()` or `HttpRequestMemory()` , this command is only valid if the request has been successful.

Parameters

HttpRequest The HTTP request to get the raw response.

Return value

A memory buffer containing the whole response of an HTTP request. Once done with it, the buffer needs to be freed with `FreeMemory()` .

Example

```
1  HttpRequest = HttpRequest(#PB_HTTP_Get, "https://www.google.com")
2  If HttpRequest
3      Debug "Response: " + HTTPInfo(HttpRequest, #PB_HTTP_Response)
4
5      *Response = HTTPMemory(HttpRequest)
6
7      FinishHTTP(HttpRequest)
8
9      Debug "Response size: " + MemorySize(*Response)
10     FreeMemory(*Response)
11
12 Else
13     Debug "Request creation failed"
14 EndIf
```

See Also

`HttpRequest()` , `HttpRequestMemory()`

124.7 HTTPProxy

Syntax

`HTTPProxy(URL$ [, User$, Password$])`

Description

Specify a proxy to use for the following HTTP commands: `ReceiveHTTPFile()` , `ReceiveHTTPMemory()` , `HttpRequest()` and `HttpRequestMemory()` .

Parameters

URL\$ The URL to use for the proxy. Default is HTTP proxy if no prefix is specified. Available prefixes to specify proxy type:


```

http://      - HTTP proxy (default)
socks4://    - SOCKS4 proxy
socks4a://   - SOCKS4 proxy with domain name support rather
               than IP address
socks5://    - SOCKS5 proxy
socks5h://   - SOCKS5 proxy and ask the proxy to do the
               hostname resolving

```

User\$, Password\$ (optional) The user and password to use to connect to the proxy (if any).

Return value

None.

Remarks

On Linux, 'libcurl' needs to be installed to have this command working (most of Linux distributions comes with it already installed).

Example

```

1  HTTPProxy("socks4://127.0.0.1")
2
3  Filename$ = SaveFileRequester("Where to save index.php ?", "",
4                               "", 0)
5
6  If ReceiveHTTPFile("http://www.purebasic.com/index.php",
7                     Filename$)
8      Debug "Success"
9  Else
10     Debug "Failed"
11 EndIf

```

See Also

ReceiveHTTPFile() , ReceiveHTTPMemory() , HTTPRequest() , HTTPRequestMemory()

124.8 HTTPTimeout

Syntax

```
HTTPTimeout(ConnectTimeout [, GlobalTimeout])
```

Description

Specify timeout to use for the following HTTP commands: ReceiveHTTPFile() , ReceiveHTTPMemory() , HTTPRequest() and HTTPRequestMemory() .

Parameters

ConnectTimeout The timeout, in milliseconds, to abort the HTTP request if the remote host doesn't connect. The default timeout is set to 30 seconds. `#PB_Default` can be used to revert to the default timeout value (30 seconds).

GlobalTimeout (optional) The global timeout, in milliseconds, to abort the HTTP request if it is still not finished. There is no timeout by default. `#PB_Default` can be used to revert to the default behaviour (no timeout). Note: if a connection is slow, it could abort a valid unfinished request.

Return value

None.

Remarks

If "Create thread-safe executable" is enabled in the compiler options then every thread has its own timeouts, which means two threads can have different timeouts at the same time.

Example

```
1  HTTPTimeout(5000) ; Allows 5 secs to connect to the server
2
3  HttpRequest = HttpRequest(#PB_HTTP_Get, "https://www.google.com")
4  If HttpRequest
5      Debug "StatusCode: " + HTTPInfo(HttpRequest,
6      #PB_HTTP_StatusCode)
7      Debug "Response: " + HTTPInfo(HttpRequest, #PB_HTTP_Response)
8
9      FinishHTTP(HttpRequest)
10 Else
11     Debug "Request creation failed"
12 EndIf
```

See Also

ReceiveHTTPFile() , ReceiveHTTPMemory() , HttpRequest() , HttpRequestMemory()

124.9 ReceiveHTTPFile

Syntax

```
Result = ReceiveHTTPFile(URL$, Filename$ [, Flags [, UserAgent$]])
```

Description

Download a file to disk from the given URL\$.

Parameters

URL\$ The URL to download from.

Filename\$ The local filename to write the file to. If the filename does not include a full path, it is interpreted relative to the current directory . If the file exists, it will be overwritten.

Flags (optional) It can be a combination of the following values:

```
#PB_HTTP_Asyncronous: starts the download asynchronously.
#PB_HTTP_NoRedirect  : don't follow automatic redirections.
```

UserAgent\$ (optional) Change the user agent for the HTTP request. Default user agent is set to "Mozilla/5.0 Gecko/41.0 Firefox/41.0" for maximum compatibility.

Return value

Returns nonzero if the download was successful, zero otherwise. If `#PB_HTTP_Asyncronous` was specified, it returns the 'HttpConnection' value needed for `HTTPProgress()`, `AbortHTTP()` and `FinishHTTP()`. When using `#PB_HTTP_Asyncronous`, `FinishHTTP()` needs to be called when the download is finished (successfully or not).

Remarks

On Linux, 'libcurl' needs to be installed to have this command working (most of Linux distributions comes with it already installed).

If the remote file was not found, it will contains the webserver error.

Example

```
1  Filename$ = SaveFileRequester("Where to save index.php ?", "",
2      "", 0)
3  If ReceiveHTTPFile("https://www.purebasic.com/index.php",
4      Filename$)
5      Debug "File received and written to disk. If the remote file
6      was not found, it will contains the webserver error."
7  Else
8      Debug "No file received (network error)."
9  EndIf
```

See Also

`URLEncoder()`

124.10 ReceiveHTTPMemory

Syntax

```
*Buffer = ReceiveHTTPMemory(URL$ [, Flags [, UserAgent$]])
```

Description

Download a file to a new memory buffer from the given URL\$.

Parameters

URL\$ The URL to download from.

Flags (optional) It can be a combination of the following values:

`#PB_HTTP_Asyncronous`: starts the download asynchronously. To get the memory buffer address use `FinishHTTP()`

`#PB_HTTP_NoRedirect` : don't follow automatic redirections.

UserAgent\$ (optional) Change the user agent for the HTTP request. Default user agent is set to "Mozilla/5.0 Gecko/41.0 Firefox/41.0" for maximum compatibility.

Return value

Returns the new memory buffer address if the download was successful, zero otherwise. MemorySize() can be used to get the size of the downloaded item. The memory buffer needs to be freed with FreeMemory() once finished using it. If #PB_HTTP_Asyncronous was specified, it returns the 'HttpConnection' value needed for HTTPProgress() , AbortHTTP() and FinishHTTP() . When using #PB_HTTP_Asyncronous, FinishHTTP() needs to be called when the download is finished (successfully or not).

Remarks

On Linux, 'libcurl' needs to be installed to have this command working (most of Linux distributions comes with it already installed).

Example

```
1  *Buffer = ReceiveHTTPMemory("http://www.purebasic.com/index.php")
2  If *Buffer
3      Size = MemorySize(*Buffer)
4      Debug "Content: " + PeekS(*Buffer, Size,
5          #PB_UTF8|#PB_ByteLength)
6      FreeMemory(*Buffer)
7  Else
8      Debug "Failed"
9  EndIf
```

See Also

URLEncoder()

124.11 HTTPRequest

Syntax

```
Result = HTTPRequest(Type, URL$ [, Data$ [, Flags [, Headers()]]])
```

Description

Send an HTTP request with optional text data. If binary data needs to be sent, HTTPRequestMemory() can be used. This command is designed to handle REST like web APIs easily. FinishHTTP() needs to be always called once the request has been executed.

Parameters

Type The type of the request. Can be one of the following values:

```
#PB_HTTP_Get      : GET request ('Data$' parameter will be
                    ignored)
#PB_HTTP_Post     : POST request ('Data$' parameter will be
                    sent if specified)
#PB_HTTP_Put      : PUT request ('Data$' parameter will be sent
                    if specified)
#PB_HTTP_Patch    : PATCH request ('Data$' parameter will be
                    sent if specified)
#PB_HTTP_Delete   : DELETE request ('Data$' parameter will be
                    sent if specified)
```

URL\$ The URL to send the request to.

Data\$ (optional) The text data to send with the request (will be sent as UTF-8 format).

Flags (optional) It can be a combination of the following values:

```
#PB_HTTP_Asyncronous: starts the download asynchronously.
#PB_HTTP_NoRedirect   : don't follow automatic redirections.
#PB_HTTP_NoSSLCheck   : don't check if the SSL certificate is
                        valid (can be useful for testing purpose).
#PB_HTTP_HeadersOnly  : gets headers only.
#PB_HTTP_WeakSSL       : to support older servers.
#PB_HTTP_Debug        : to print in the console debug
                        information.
```

Headers() (optional) A map of string pair to specify additional headers for the request.

Example:

```
1  NewMap Header$()
2  Header$("Content-Type") = "text/plain"
3  Header$("User-Agent") = "Firefox 54.0"
4  Header$("NoParamHeader") = "" ; When putting no string
   value, it will an empty parameter
```

Return value

Returns the HTTP request identifier if the call has been successfully initialized, zero otherwise. HTTPInfo() can be used to get some info about the request. If #PB_HTTP_Asyncronous was specified, HTTPProgress() can be used and AbortHTTP() must be used. HTTPMemory() can be used to get the result as a raw buffer (the raw buffer must be freed with FreeMemory()). FinishHTTP() has to be always called to finish a successfully initialized HTTP request, even if the call was synchronous.

Remarks

On Linux, 'libcurl' needs to be installed to have this command working (most of Linux distributions comes with it already installed).

Example

```
1  HttpRequest = HTTPRequest(#PB_HTTP_Get , "https://www.google.com")
2  If HttpRequest
3      Debug "StatusCode: " + HTTPInfo(HttpRequest ,
   #PB_HTTP_StatusCode)
4      Debug "Response: " + HTTPInfo(HttpRequest , #PB_HTTP_Response)
5
6      FinishHTTP(HttpRequest)
7  Else
8      Debug "Request creation failed"
9  EndIf
```

Example: with headers

```
1  NewMap Header$()
2  Header$("Content-Type") = "plaintext"
3  Header$("User-Agent") = "Firefox 54.0"
4
```

```

5  HttpRequest = HTTPRequest(#PB_HTTP_Get, "https://www.google.com",
   "", 0, Header$())
6  If HttpRequest
7      Debug "StatusCode: " + HTTPInfo(HTTPRequest,
   #PB_HTTP_StatusCode)
8      Debug "Response: " + HTTPInfo(HTTPRequest, #PB_HTTP_Response)
9
10     FinishHTTP(HTTPRequest)
11 Else
12     Debug "Request creation failed"
13 EndIf

```

Example: Asynchronous

```

1  HttpRequest = HTTPRequest(#PB_HTTP_Get,
   "https://www.purebasic.com/download/PureBasic_Demo.zip", "",
   #PB_HTTP_Asyncronous)
2  If HttpRequest
3      Debug "StatusCode: " + HTTPInfo(HTTPRequest,
   #PB_HTTP_StatusCode)
4
5      Repeat
6          Progress = HTTPProgress(HttpRequest)
7          Select Progress
8              Case #PB_HTTP_Success
9                  Debug "Download finished"
10                 *Buffer = HTTPMemory(HTTPRequest)
11                 If *Buffer
12                     Debug "Memory buffer size: "+MemorySize(*buffer) ; Here
the buffer can be written to disk or used directory
13                     FreeMemory(*Buffer)
14                 EndIf
15                 FinishHTTP(HTTPRequest) ; Always call FinishHTTP() when
request is finished
16                 Break
17
18                 Case #PB_HTTP_Failed
19                     Debug "Download failed"
20                     FinishHTTP(HTTPRequest) ; Always call FinishHTTP() when
request failed
21                     Break
22
23                 Case #PB_HTTP_Aborted
24                     Debug "Download aborted"
25                     FinishHTTP(HTTPRequest) ; Always call FinishHTTP() when
request is aborted
26
27                 Default
28                     Debug "Current download: " + Progress ; The current
download progress, in bytes
29                     Delay(100)
30                 EndSelect
31             Forever
32 Else
33     Debug "Request creation failed"
34 EndIf

```

See Also

URLEncoder() , AbortHTTP()

124.12 HTTPRequestMemory

Syntax

```
Result = HTTPRequestMemory(Type, URL$ [, *Data, DataSize [, Flags  
[, Headers()]]])
```

Description

Send an HTTP request with optional binary data. If text only data needs to be sent HTTPRequest() can be used. This command is designed to handle REST like web API easily. FinishHTTP() needs to be always called once the request has been executed.

Parameters

Type The type of the request. Can be one of the following values:

```
#PB_HTTP_Get      : GET request ('*Data' parameter will be  
                    ignored)  
#PB_HTTP_Post     : POST request ('*Data' parameter will be sent  
                    if specified)  
#PB_HTTP_Put      : PUT request ('*Data' parameter will be sent  
                    if specified)  
#PB_HTTP_Patch    : PATCH request ('*Data' parameter will be  
                    sent if specified)  
#PB_HTTP_Delete   : DELETE request ('*Data' parameter will be  
                    sent if specified)
```

URL\$ The URL to send the request to.

***Data (optional)** The memory buffer to send with the request.

DataSize (optional) The size (in bytes) of the memory buffer to send with the request.

Flags (optional) It can be a combination of the following values:

```
#PB_HTTP_Asynchronous : starts the download asynchronously.  
#PB_HTTP_NoRedirect    : don't follow automatic redirections.  
#PB_HTTP_NoSSLCheck    : don't check if the SSL certificate is  
                        valid (can be useful for testing purpose).  
#PB_HTTP_HeadersOnly   : gets headers only.  
#PB_HTTP_WeakSSL       : to support older servers.  
#PB_HTTP_Debug         : to print in the console debug  
                        information.
```

Headers() (optional) A map of string pair to specify additional headers for the request.

Example:

```
1  NewMap Header$()  
2  Header$("ContentType") = "octectstream"  
3  Header$("User-Agent") = "Firefox 54.0"  
4  Header$("NoParamHeader") = "" ; When putting no string  
   value, it will be an empty parameter
```

Return value

Returns the HTTP request identifier if the call has been successfully initialized, zero otherwise. HTTPInfo() can be used to get some info about the request. If #PB_HTTP_Asynchrouous was specified, HTTPProgress() and AbortHTTP() can be used. HTTPMemory() can be used to get the result as a raw buffer (the raw buffer must be freed with FreeMemory()). FinishHTTP() has to be always called to finish a successfully initialized HTTP request, even if the call was synchronous.

Remarks

On Linux, 'libcurl' needs to be installed to have this command working (most of Linux distributions comes with it already installed).

Example

```
1  HttpRequest = HTTPRequestMemory(#PB_HTTP_Get ,
2      "https://www.google.com")
3  If HttpRequest
4      Debug "StatusCode: " + HTTPInfo(HttpRequest ,
5          #PB_HTTP_StatusCode)
6      Debug "Response: " + HTTPInfo(HttpRequest , #PB_HTTP_Response)
7      FinishHTTP(HttpRequest)
8  Else
9      Debug "Request creation failed"
10 EndIf
```

See Also

HttpRequest() , URLEncoder() , AbortHTTP()

124.13 URLEncoder

Syntax

Result\\$ = URLEncoder(URL\$ [, Format])

Description

Returns a decoded URL\$ which has been encoded with the HTTP format.

Parameters

URL\$ The URL to decode. To encode an URL, use URLEncoder() .

Format (optional) The format of the URL encoding. It can be one of the following values:

```
#PB_UTF8    (default)
#PB_Ascii
```

Return value

Returns the decoded URL.

Remarks

A URL\$ may not contain certain characters such as: tab, space, accent letter etc., therefore these characters must be encoded, which basically involves using "%" as an escape character. If the URL\$ is not in an encoded format, this function will have no effect on the given "URL\$" and the return-value of that "URL\$" will remain unchanged.

Example

```
1  Debug
   URLDecoder("http://www.purebasic.com/test%20with%20space.php3")
2  ; Will print "http://www.purebasic.com/test with space.php3"
```

See Also

URLEncoder()

124.14 URLEncoder

Syntax

Result\\$ = URLEncoder(URL\$ [, Format])

Description

Returns the URL\$ encoded to the HTTP format.

Parameters

URL\$ The URL to encode.

Format (optional) The format of the string before encoding it. It can be one of the following values:

```
#PB_UTF8    (default)
#PB_Ascii
```

Return value

Returns the encoded URL. To convert an encoded URL back to a unencoded format, use URLDecoder()

Remarks

A URL\$ may not contain certain characters such as: tab, space, accent letter etc., therefore these characters must be encoded, which basically involves using "%" as an escape character. Because this function use the RFC 3986 standard, some characters will not be encoded, such as (non-exhaustive list):

```
"_" | "-" | "." | "!" | "~" | "*" | "'" | "(" | ")" |
";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" |
"," | "" | "#" | "%" |
```

Anyway, if you need to encode them, you will need to use the following table:
https://www.w3schools.com/tags/ref_urlencode.asp
For example in UTF8:

```

"-" -> %2D | "_" -> %5F | "." -> %2E | "!" -> %21 | "~" -> %7E |
"*" -> %2A | "'" -> %27 | "(" -> %28 | ")" -> %29 | ";" -> %3B |
"/" -> %2F | "?" -> %3F | ":" -> %3A | "@" -> %40 | "&" -> %26 |
"=" -> %3D | "+" -> %2B | "$" -> %24 | "," -> %2C | "\"" -> %22 |
"#" -> %23 | "%" -> %25 |

```

Example

```

1  Debug URLEncoder("http://www.purebasic.com/test with space.php3")
2  ; Will print "http://www.purebasic.com/test%20with%20space.php3"
3
4  Debug URLEncoder("http://www.ok.com value=zzz ?yyy/")
5  ; Will print "http://www.ok.com%20value=zzz%20?yyy/"

```

See Also

URLDecoder()

124.15 SetURLPart

Syntax

Result\\$ = SetURLPart(URL\$, Parameter\$, Value\$)

Description

Set a specific part of the given URL\$.

Parameters

URL\$ The URL to modify. A URL\$ may contain parameters, which are useful when a scripting language such as: (PHP) is being used on that server. The syntax is as follows:
(http://www.purebasic.com/index.php3?test=1) Here, the parameter is named "test" and its associated value is "1".

In order to set a specific part of a URL, the information provided within "URL\$" must (at the minimum), be of the following format: ("http://www.purebasic.com")

Parameter\$ The parameter to modify. The parameters are not case-sensitive. Moreover, Parameter\$ may be one of the following constants:

```

#PB_URL_Protocol: change the protocol of the URL$
#PB_URL_Site: change the site of the URL$
#PB_URL_Port: change the port of the URL$ (or adds it if not
specified)
#PB_URL_Parameters: changes all the parameters of the URL$
(or adds it if not specified)
#PB_URL_Path: changes the path of the URL$ (or adds it if not
specified)
#PB_URL_User: changes the username of the URL$ (or adds it if
not specified)
#PB_URL_Password: changes the password of the URL$ (or adds
it if not specified - the "user" needs to exist)

```

Value\$ The value to assign to the given parameter or URL part.

Return value

Returns the modified URL.

Example

```
1  URL$ = "http://www.test.com/hello.php3"
2
3  URL$ = SetURLPart(URL$, #PB_URL_Protocol, "ftp")
4  URL$ = SetURLPart(URL$, #PB_URL_Site, "www.purebasic.com")
5  URL$ = SetURLPart(URL$, #PB_URL_Port, "80")
6  URL$ = SetURLPart(URL$, #PB_URL_Path, "english/index.php3")
7  URL$ = SetURLPart(URL$, #PB_URL_User, "user")
8  URL$ = SetURLPart(URL$, #PB_URL_Password, "pass")
9  URL$ = SetURLPart(URL$, "test", "1")
10 URL$ = SetURLPart(URL$, "ok", "2")
11
12 Debug URL$ ; Will print
    "ftp://user:pass@www.purebasic.com:80/english/index.php3?test=1&ok=2"
```

See Also

GetURLPart() , URLEncoder()

Chapter 125

Image

Overview

Images are graphics objects which can be displayed in a window or in several gadgets. PureBasic supports beside BMP and ICON (.ico, Windows only) image types all other image types, which are supported by the ImagePlugin Library .

Transparent PNG images can be used to enable transparency in the gadgets , menu and toolbars images. On Windows, transparent ICON images can be used as well but PNG should be preferred as it works on all platforms. Transparency of BMP images is not supported.

125.1 AddImageFrame

Syntax

```
Result = AddImageFrame(#Image [, Index])
```

Description

Adds a new frame to the specified image. The new frame will have the same dimension and depth like the image.

Parameters

#Image The image to add a frame.

Index (optional) The index (starting from 0) to insert the frame. If not specified the new frame will be added at the end of the frame list.

Return value

Returns nonzero if the new frame has been successfully created, zero otherwise.

See Also

CreateImage() , RemoveImageFrame() , ImageFrameCount() , SetImageFrame() ,
GetImageFrame()

125.2 RemoveImageFrame

Syntax

```
Result = RemoveImageFrame(#Image, Index)
```

Description

Removes the specified frame from the image.

Parameters

#Image The image to remove a frame.

Index The index (starting from 0) of the frame to remove.

Return value

Returns nonzero if the frame has been successfully removed, zero otherwise.

See Also

CreateImage() , AddImageFrame() , ImageFrameCount() , SetImageFrame() , GetImageFrame()

125.3 GetImageFrame

Syntax

```
Index = GetImageFrame(#Image)
```

Description

Gets the current frame index of the image.

Parameters

#Image The image to use.

Return value

The current frame index (starting from 0). If the image is not multi-frame, it will always return 0.

See Also

CreateImage() , AddImageFrame() , RemoveImageFrame() , ImageFrameCount() ,
GetImageFrame()

125.4 SetImageFrame

Syntax

```
SetImageFrame(#Image, Index)
```

Description

Changes the current frame of the image. If the image is not multi-frame, this function has no effect. ImageOutput() , ImageVectorOutput() , CopyImage() and GrabImage() works on the current frame.

Parameters

#Image The image to use.

Index The index (starting from 0) of the frame to set as current.

Return value

None.

See Also

CreateImage() , AddImageFrame() , RemoveImageFrame() , ImageFrameCount() , GetImageFrame()

125.5 ImageFrameCount

Syntax

```
Result = ImageFrameCount(#Image)
```

Description

Returns the number of frames of the image.

Parameters

#Image The image to get the frame count.

Return value

Returns the frame count of the image. If the image is not multi-frame, it will always return 1.

See Also

CreateImage() , AddImageFrame() , RemoveImageFrame() , SetImageFrame() , GetImageFrame()

125.6 GetImageFrameDelay

Syntax

```
Result = GetImageFrameDelay(#Image)
```

Description

Returns the display delay (in milliseconds) for the current image frame. Each frame can have its own delay.

Parameters

#Image The image to get the frame delay.

Return value

Returns the display delay (in milliseconds) for the current image frame.

See Also

CreateImage() , AddImageFrame() , RemoveImageFrame() , SetImageFrame() , GetImageFrame()
, SetImageFrameDelay()

125.7 SetImageFrameDelay

Syntax

```
Result = SetImageFrameDelay(#Image, Delay)
```

Description

Sets the display delay (in milliseconds) for the current image frame. Each frame can have its own delay.

Parameters

#Image The image to set the frame delay.

Return value

Sets the display delay (in milliseconds) for the current image frame.

See Also

CreateImage() , AddImageFrame() , RemoveImageFrame() , SetImageFrame() , GetImageFrame()
, GetImageFrameDelay()

125.8 CatchImage

Syntax

```
Result = CatchImage(#Image, *MemoryAddress [, Size])
```

Description

Load the specified image from the given memory area.

Parameters

#Image A number to identify the loaded image. #PB_Any can be specified to auto-generate this number.

***MemoryAddress** The memory address from which to load the image.

Size (optional) The size of the image in bytes. The size is optional as the loader can determine from the image when to stop reading. It is however advisable to provide a size when loading unknown images, as the loader can then handle corrupted images correctly (without specifying the image size, a corrupt image can crash the program).

Return value

Returns nonzero if the image was loaded successfully and zero if the image could not be loaded. If #PB_Any was specified as the #Image parameter then the auto-generated number is returned on success.

Remarks

The limit for the image size that can be handled depends on the operating system and the available amount of memory. If enough memory is available, then images up to at least 8192x8192 pixels can be handled by all operating systems supported by PureBasic.

When an image is loaded, it is converted either in 24-bit (if the image depth is less or equal to 24-bit) or in 32-bit (if the image has an alpha channel). A loaded image can be freed by using the FreeImage() function.

This function is useful when using the 'IncludeBinary' PureBasic keyword. Then images can be packed inside the executable. Nevertheless, use this option with care, as it will take more memory than storing the file in an external file (the file are both in executable memory and load in physical memory).

The image can be in BMP, icon (.ico, only on Windows) or any other format supported by the ImagePlugin library. Transparency of BMP images is not supported. The following functions can be used to enable additional image formats:

UseJPEGImageDecoder()

UseJPEG2000ImageDecoder()

UsePNGImageDecoder()

UseTIFFImageDecoder()

UseTGAImageDecoder()

UseGIFImageDecoder()

Example

```
1  If OpenWindow(0, 0, 0, 148, 148, "ImageGadget",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      If CatchImage(0, ?Geebee)
4          ImageGadget(0, 10, 10, 0, 0, ImageID(0))
5      EndIf
```



```

5      Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
6      EndIf
7
8      DataSection
9          GeeBee:
10         IncludeBinary #PB_Compiler_Home +
            "examples/sources/Data/Geebee2.bmp"
11     EndDataSection

```

Note: The "?" is a pointer to a label. More information about pointers and memory access can be found in the relating chapter [here](#) .

See Also

CreateImage() , LoadImage() , FreeImage() , ImagePlugin library

125.9 CopyImage

Syntax

```
Result = CopyImage(#Image1, #Image2)
```

Description

Creates an identical copy of an image. If the image is multi-frame, the current frame will be used for the copy.

Parameters

#Image1 The source image.

#Image2 A number to identify the new copy. #PB_Any can be specified to auto-generate this number.

Note: The number of an existing image created using #PB_Any can not be used as the target image. Instead, the existing image must be freed and a new number generated by passing #PB_Any here.

Return value

Returns nonzero if the image was copied successfully and zero if the copy could not be created. If #PB_Any was specified as the #Image2 parameter then the auto-generated number is returned on success.

See Also

GrabImage() , FreeImage()

125.10 CreateImage

Syntax

```
Result = CreateImage(#Image, Width, Height [, Depth [, BackColor]])
```

Description

Create an empty image (with black background) which can be used to do rendering on it.

Parameters

#Image A number to identify the new image. **#PB_Any** can be specified to auto-generate this number.

Width, Height The dimensions of the new image. Both the width and height must be greater than zero.

Depth (optional) The color depth for the new image. Valid values can be: 24 and 32. The default is 24-bit if no depth is specified.

BackColor (optional) The back RGB() color used when the image is created. This color can be set to **#PB_Image_Transparent** to create an image with the alpha channel set to full transparent. This only has an effect on 32-bit images. The default backcolor is black if not specified.

Return value

Returns nonzero if the image was created successfully and zero if the image could not be created. If **#PB_Any** was specified as the **#Image** parameter then the auto-generated number is returned on success.

Remarks

The limit for the image size that can be handled depends on the operating system and the available amount of memory. If enough memory is available, then images up to at least 8192x8192 pixels can be handled by all operating systems supported by PureBasic.

You can use the several other functions for acting with the created image:

StartDrawing() with ImageOutput() to draw on the created image

StartVectorDrawing() with ImageVectorOutput() to draw on the created image using vector drawing

CopyImage() to create another image from the actual one

GrabImage() to create another image from a given area of the actual one

DrawImage() with ImageID() to draw the image on actual output channel.

ImageGadget() for displaying image on an application window

ButtonImageGadget() for creating an image button on an application window

See Also

LoadImage() , CatchImage() , FreeImage()

125.11 EncodeImage

Syntax

```
*Buffer = EncodeImage(#Image [, ImagePlugin [, Flags [, Depth]])
```

Description

Encode the specified image into a memory buffer.

Parameters

#Image The image to encode.

ImagePlugin (optional) The format to encode the image in. This can be one of the following values:

```

#PB_ImagePlugin_BMP      : encode the image in BMP (default)
#PB_ImagePlugin_JPEG     : encode the image in JPEG
    (UseJPEGImageEncoder())
has to be used)
#PB_ImagePlugin_JPEG2000 : encode the image in JPEG2000
    (UseJPEG2000ImageEncoder())
has to be used)
#PB_ImagePlugin_PNG      : encode the image in PNG
    (UsePNGImageEncoder())
has to be used)

```

Flags (optional) Parameters for the image plug-in. For now, only the quality setting is supported: a number from 0 (worse quality) to 10 (maximum quality). Only the JPEG and JPEG 2000 plugins currently support it (default quality is set to '7' if no flags are specified). When an image is encoded using palletized depth (1, 4 or 8), the following flag is available for combination:

```
#PB_Image_FloydSteinberg: Apply a Floyd-Steinberg dithering.
```

Depth (optional) The depth in which to save the image. Valid values are 1, 4, 8, 24 and 32. The default value is the original image depth. For now, only PNG encoder support palletized image format (1, 4 or 8-bit).

Return value

Returns a pointer to a newly allocated memory buffer containing the encoded image, or zero if the encoding has failed. `MemorySize()` can be used to get the size of the encoded image. `FreeMemory()` has to be used to free the buffer after use.

See Also

`LoadImage()` , `ImagePlugin` library

125.12 FreeImage

Syntax

```
FreeImage(#Image)
```

Description

Free the specified image and release its associated memory.

Parameters

#Image The image to free. If **#PB_All** is specified, all the remaining images are freed.

Return value

None.

Remarks

All remaining images are automatically freed when the program ends.

See Also

CreateImage() , LoadImage() , CatchImage() , CopyImage() , GrabImage()

125.13 GrabImage

Syntax

```
Result = GrabImage(#Image1, #Image2, x, y, Width, Height)
```

Description

Create a new image with the selected area on the source image. If the image is multi-frame, the current frame will be used for the grab.

Parameters

#Image1 The source image.

#Image2 A number to identify the new image. **#PB_Any** can be specified to auto-generate this number.

Note: The number of an existing image created using **#PB_Any** can not be used as the target image. Instead, the existing image must be freed and a new number generated by passing **#PB_Any** here.

x, y, Width, Height The location and size of the area to copy into the new image.

Return value

Returns nonzero if the image was created successfully and zero if the image could not be created. If **#PB_Any** was specified as the **#Image2** parameter then the auto-generated number is returned on success.

See Also

CreateImage() , LoadImage() , CatchImage() , CopyImage()

125.14 ImageDepth

Syntax

```
Result = ImageDepth(#Image [, Flags])
```

Description

Returns the depth of the **#Image**, as it is stored internally by PureBasic.

Parameters

#Image The image to use.

Flags (optional) The kind of depth to return. It can be one of the following values:

```

#PB_Image_InternalDepth: Returns the depth of the decoded
image (default). Valid return values are:
- 24 (24-bit, 16 millions
colors)
- 32 (32-bit, 16 millions
colors + 8-bit alpha channel)
#PB_Image_OriginalDepth: Returns the original depth, as it
was before the decoding. Valid return values are:
- 1 (1-bit, 2 colors,
monochrome image)
- 4 (4-bit, 16 colors)
- 8 (8-bit, 256 colors)
- 16 (16-bit, 65536 colors)
- 24 (24-bit, 16 millions
colors)
- 32 (32-bit, 16 millions
colors + 8-bit alpha channel)

```

Return value

Returns one of the possible depth values described above.

Remarks

When loading an image, PureBasic converts it either in 24-bit or in 32-bit, depending if the source image has an alpha channel or not. Every image which has a depth less than 24-bit will be internally converted to 24-bit.

See Also

ImageWidth() , ImageHeight()

125.15 ImageFormat

Syntax

```
Result = ImageFormat(#Image)
```

Description

Return the original image format.

Parameters

#Image The image to use.

Return value

Returns the image original format. It can be one of the following value:

```

#PB_ImagePlugin_JPEG
#PB_ImagePlugin_JPEG2000
#PB_ImagePlugin_PNG
#PB_ImagePlugin_TGA
#PB_ImagePlugin_TIFF

```

```
#PB_ImagePlugin_BMP
#PB_ImagePlugin_ICON
#PB_ImagePlugin_GIF
```

If the image was not loaded from one of these formats, it will return zero. This is the case for images created with `CreateImage()` or `GrabImage()` .

See Also

`LoadImage()` , `CreateImage()` , `CatchImage()` , `GrabImage()` , `ImagePlugin` library

125.16 ImageHeight

Syntax

```
Result = ImageHeight(#Image)
```

Description

Returns the height of the given image.

Parameters

#Image The image to use.

Return value

Returns the height of the image in pixels.

See Also

`ImageWidth()` , `ImageDepth()`

125.17 ImageID

Syntax

```
Result = ImageID(#Image)
```

Description

Returns the ImageID of the image.

Parameters

#Image The image to use.

Return value

Returns the operating system handle of the image.

See Also

`DrawImage()` , `ImageGadget()` , `ButtonImageGadget()` , `CanvasGadget()`

125.18 ImageOutput

Syntax

```
OutputID = ImageOutput(#Image)
```

Description

Returns the OutputID of the image to perform 2D rendering operation on it. Alternatively, the ImageVectorOutput() command can be used to perform vector drawing on the image. If the image is multi-frame, the current frame will be used.

Parameters

#Image The image to draw on.

Return value

Returns the output ID or zero if drawing is not possible. This value should be passed directly to the StartDrawing() function to start the drawing operation. The return-value is valid only for one drawing operation and cannot be reused.

Example

```
1 StartDrawing(ImageOutput(#Image))
2 ; do some drawing stuff here...
3 StopDrawing()
```

Remarks

This command cannot be used with loaded icon files (*.ico).

See Also

StartDrawing() , ImageVectorOutput()

125.19 ImageVectorOutput

Syntax

```
VectorOutputID = ImageVectorOutput(#Image [, Unit])
```

Description

Returns the OutputID of the image to perform 2D vector drawing operations on it. If the image is multi-frame, the current frame will be used.

Parameters

#Image The image to draw on.

Unit (optional) Specifies the unit used for measuring distances on the drawing output. The default unit for images is `#PB_Unit_Pixel`.

```
#PB_Unit_Pixel      : Values are measured in pixels (or dots
                      in case of a printer)
#PB_Unit_Point      : Values are measured in points (1/72 inch)
#PB_Unit_Inch       : Values are measured in inches
#PB_Unit_Millimeter : Values are measured in millimeters
```

Return value

Returns the output ID or zero if drawing is not possible. This value should be passed directly to the `StartVectorDrawing()` function to start the drawing operation. The return-value is valid only for one drawing operation and cannot be reused.

Example

```
1  StartVectorDrawing(ImageVectorOutput(#Image, #PB_Unit_Millimeter))
2  ; do some drawing stuff here...
3  StopVectorDrawing()
```

Remarks

This command cannot be used with loaded icon files (*.ico).

See Also

`StartVectorDrawing()` , `ImageOutput()`

125.20 ImageWidth

Syntax

```
Result = ImageWidth(#Image)
```

Description

Returns the width of the given image.

Parameters

#Image The image to use.

Return value

Returns the width of the image in pixels.

See Also

`ImageHeight()` , `ImageDepth()`

125.21 IsImage

Syntax

```
Result = IsImage(#Image)
```

Description

Tests if the given image number is a valid and correctly initialized image.

Parameters

#Image The image to test.

Return value

Returns nonzero if #Image is a valid image and zero if not.

Remarks

This function is bulletproof and can be used with any value. This is the correct way to ensure an image is ready to use.

See Also

CreateImage() , LoadImage() , CatchImage() , CopyImage() , GrabImage()

125.22 LoadImage

Syntax

```
Result = LoadImage(#Image, Filename$ [, Flags])
```

Description

Load the specified image from a file.

Parameters

#Image A number to identify the loaded image. #PB_Any can be specified to auto-generate this number.

Filename\$ The name of the file to load. If the filename does not include a full path, it is interpreted relative to the current directory .

Flags (optional) This parameter currently has no meaning. If it is specified, 0 should be used for future compatibility.

Return value

Returns nonzero if the image was loaded successfully and zero if the image could not be loaded. If #PB_Any was specified as the #Image parameter then the auto-generated number is returned on success.

Remarks

The limit for the image size that can be handled depends on the operating system and the available amount of memory. If enough memory is available, then images up to at least 8192x8192 pixels can be handled by all operating systems supported by PureBasic.

When an image is loaded, it is converted either in 24-bit (if the image depth is less or equal to 24-bit) or in 32-bit (if the image has an alpha channel). A loaded image can be freed by using the `FreeImage()` function.

The image can be in BMP, icon (.ico, only on Windows) or any other format supported by the ImagePlugin library. Alpha channel is supported for 32-bit BMP images. The following functions can be used to enable automatically more image formats:

`UseJPEGImageDecoder()`

`UseJPEG2000ImageDecoder()`

`UsePNGImageDecoder()`

`UseTIFFImageDecoder()`

`UseTGAImageDecoder()`

`UseGIFImageDecoder()`

You can use the several other functions for acting with the loaded image:

`StartDrawing()` with `ImageOutput()` to draw on the loaded image

`StartVectorDrawing()` with `ImageVectorOutput()` to draw on the created image using vector drawing

`CopyImage()` to create another image from the actual one

`GrabImage()` to create another image from a given area of the actual one

`DrawImage()` with `ImageID()` to draw the image on actual output channel.

`DrawAlphaImage()` with `ImageID()` to draw the image (with alpha channel) on actual output channel.

`ImageGadget()` for displaying image on an application window

`ButtonImageGadget()` for creating an image button on an application window

See Also

`CreateImage()` , `CatchImage()` , `CopyImage()` , `GrabImage()` , ImagePlugin library

125.23 ResizeImage

Syntax

```
Result = ResizeImage(#Image, Width, Height [, Mode])
```

Description

Resize the #Image to the given dimension.

Parameters

#Image The image to resize.

Width, Height The new dimensions of the image. Both width and height must be greater than zero. **#PB_Ignore** can be specified for width or height, so this value won't be changed.

Mode (optional) The resize method. It can be one of the following values:

```
#PB_Image_Smooth: Resize the image with smoothing (default).  
#PB_Image_Raw   : Resize the image without any interpolation.
```

Return value

Returns nonzero if the operation succeeded and zero if it failed.

Remarks

This function changes the handle of the used image. Therefore it must be newly assigned e.g. to an ImageGadget() with SetGadgetState() .

This function does not work with icon images (.ico) and with a multi-frame image.

The limit for the image size that can be handled depends on the operating system and the available amount of memory. If enough memory is available, then images up to at least 8192x8192 pixels can be handled by all operating systems supported by PureBasic.

See Also

ImageWidth() , ImageHeight()

125.24 SaveImage

Syntax

```
Result = SaveImage(#Image, Filename$ [, ImagePlugin [, Flags [,
    Depth]])
```

Description

Saves the specified image to disk.

Parameters

#Image The image to save.

Filename\$ The file to save to. If the filename does not include a full path, it is interpreted relative to the current directory .

ImagePlugin (optional) The format to save the image in. This can be one of the following values:

```
#PB_ImagePlugin_BMP      : Save the image in BMP (default)
#PB_ImagePlugin_JPEG     : Save the image in JPEG
    (UseJPEGImageEncoder()
has to be used)
#PB_ImagePlugin_JPEG2000 : Save the image in JPEG2000
    (UseJPEG2000ImageEncoder()
has to be used)
#PB_ImagePlugin_PNG      : Save the image in PNG
    (UsePNGImageEncoder()
has to be used)
```

Flags (optional) Optional parameters for the image plug-in. For now, only the quality setting is supported: a number from 0 (worse quality) to 10 (maximum quality). Only the JPEG and JPEG 2000 plugins currently support it (default quality is set to '7' if no flags are specified). When an image is saved using palettized depth (1, 4 or 8), the following flag is available:

```
#PB_Image_FloydSteinberg: Apply a Floyd-Steinberg dithering.
```

When a 32-bit image is saved in a lower depth (24-bit or lower) the following flags are available to change the color used to fill the transparent area:

```
#PB_Image_WhiteAlphaBackground: the resulting image will use
    white color when transparent area is found (default)
#PB_Image_BlackAlphaBackground: the resulting image will use
    black color when transparent area is found
```

Depth (optional) The depth in which to save the image. Valid values are 1, 4, 8, 24 and 32. The default value is the original image depth. For now, only BMP and PNG encoders support palletized image format (1, 4 or 8-bit).

Return value

Returns nonzero if the operation succeeded and zero if it failed.

Example

```
1  UsePNGImageEncoder()
2
3  If CreateImage(0, 33, 33)
4      If StartDrawing(ImageOutput(0))
5          DrawingMode(#PB_2DDrawing_Transparent)
6          Box(0, 0, 33, 33, $FF0000) ; draw a blue box and save it as
PNG image file below
7          StopDrawing()
8      EndIf
9      file$ = SaveFileRequester("Select file to save",
GetCurrentDirectory(), "*.png|*.png", 0)
10     If file$
11         If GetExtensionPart(file$) = ""
12             file$ + ".png" ; add the file extension if needed
13         EndIf
14         If SaveImage(0, file$, #PB_ImagePlugin_PNG)
15             MessageRequester("File created", file$)
16         Else
17             MessageRequester("Error", "Failed to save file")
18         EndIf
19     EndIf
20     FreeImage(0)
21 EndIf
```

See Also

ImageDepth() , ImagePlugin library

Chapter 126

ImagePlugin

Overview

PureBasic supports external image formats through the use of a dynamic native plug-in system. By use of this system, only the required encoder or decoder will be added to the final executable, this in turn, dramatically reduces the size of the final program.

If for example, the application only requires the JPEG decoder, only the code which deals with the JPEG decoder is used.

Another nice feature is: if several decoders are being used, these decoders will be detected automatically, through the use of automatic image format detection.

The following functions support image plugins:

LoadImage() , CatchImage() , SaveImage() , LoadSprite() , CatchSprite() and SaveSprite() .

126.1 UseGIFImageDecoder

Syntax

```
UseGIFImageDecoder()
```

Description

Enables the GIF image support for the CatchImage() , LoadImage() , CatchSprite() and LoadSprite() functions. Only LoadImage() and CatchImage() support multi-frame GIF (it will result in a multi-frame image).

Parameters

None.

Return value

None.

Remarks

All formats are supported, including the progressive and interlaced format.

See Also

LoadImage() , CatchImage() , LoadSprite() , CatchSprite()

126.2 UseJPEGImageDecoder

Syntax

```
UseJPEGImageDecoder()
```

Description

Enables the JPEG (Joint Picture Expert Group) image support for the CatchImage() , LoadImage() , CatchSprite() and LoadSprite() functions.

Parameters

None.

Return value

None.

Remarks

All formats are supported, including the progressive format. The JPEG format makes use of destructive compression (lossy), what this means, is that the picture loses a certain amount of the original information when that picture is compressed. This compression algorithm is actually one of the best available, it will dramatically decrease the image size without the introduction of a lot of visible artifacts.

See Also

UseJPEGImageEncoder() , LoadImage() , CatchImage() , LoadSprite() , CatchSprite()

126.3 UseJPEGImageEncoder

Syntax

```
UseJPEGImageEncoder()
```

Description

Enables the JPEG (Joint Picture Expert Group) image support for the SaveImage() and SaveSprite() functions.

Parameters

None.

Return value

None.

Remarks

The JPEG format makes use of destructive compression (lossy), what this means, is that the picture loses a certain amount of the original information when that picture is compressed. This compression algorithm is actually one of the best available, it will dramatically decrease the image size without the introduction of a lot of visible artifacts.

See Also

UseJPEGImageDecoder() , SaveImage() , SaveSprite()

126.4 UseJPEG2000ImageDecoder

Syntax

```
UseJPEG2000ImageDecoder()
```

Description

Enables the JPEG 2000 image support for the CatchImage() , LoadImage() , CatchSprite() and LoadSprite() functions.

Parameters

None.

Return value

None.

Remarks

The JPEG 2000 format makes use of destructive compression (lossy), what this means, is that the picture loses a certain amount of the original information when that picture is compressed. This compression algorithm is actually one of the best available, it will dramatically decrease the image size without the introduction of a lot of visible artifacts.

See Also

UseJPEG2000ImageEncoder() , LoadImage() , CatchImage() , LoadSprite() , CatchSprite()

126.5 UseJPEG2000ImageEncoder

Syntax

```
UseJPEG2000ImageEncoder()
```

Description

Enables the JPEG 2000 image support for the SaveImage() and SaveSprite() functions.

Parameters

None.

Return value

None.

Remarks

The JPEG 2000 format makes use of destructive compression (lossy), what this means, is that the picture loses a certain amount of the original information when that picture is compressed. This compression algorithm is actually one of the best available, it will dramatically decrease the image size without the introduction of a lot of visible artifacts. This encoder has the capability to encode 32-bit images with alpha channel.

See Also

UseJPEG2000ImageDecoder() , SaveImage() , SaveSprite()

126.6 UsePNGImageDecoder

Syntax

```
UsePNGImageDecoder()
```

Description

Enables the PNG (Portable Network Graphic) image support for the CatchImage() , LoadImage() , CatchSprite() and LoadSprite() functions.

Parameters

None.

Return value

None.

Remarks

All formats are supported, including the progressive format. The PNG format is well known by web designers, as it is now one of the more popular formats. The PNG format makes use of non-destructive compression (lossless) which means that the picture does not lose any information when that picture is compressed. This is actually the best lossless compression algorithm available.

See Also

UsePNGImageEncoder() , LoadImage() , CatchImage() , LoadSprite() , CatchSprite()

126.7 UsePNGImageEncoder

Syntax

```
UsePNGImageEncoder()
```

Description

Enables the PNG (Portable Network Graphic) image support for SaveImage() and SaveSprite() .

Parameters

None.

Return value

None.

Remarks

The PNG format is well known by web designers, as it is now one of the more popular formats. The PNG format makes use of non-destructive compression (lossless) which means that the picture does not lose any information when that picture is compressed. This is actually the best lossless compression algorithm available. This encoder has the capability to encode 32-bit images with alpha channel.

See Also

UsePNGImageDecoder() , SaveImage() , SaveSprite()

126.8 UseTGAImageDecoder

Syntax

```
UseTGAImageDecoder()
```

Description

Enables the TGA (Targa) image support for the CatchImage() , LoadImage() , CatchSprite() and LoadSprite() functions.

Parameters

None.

Return value

None.

Remarks

The TGA format makes use of non-destructive compression (lossless) which means that the picture does not lose any information when it is compressed. The TGA image size is usually large, since the compression is very weak (Run Length Encoding - RLE). At this point, the TGA alpha channel information (if any) is ignored when using this format.

See Also

LoadImage() , CatchImage() , LoadSprite() , CatchSprite()

126.9 UseTIFFImageDecoder

Syntax

```
UseTIFFImageDecoder()
```

Description

Enables the TIFF image support for the CatchImage() , LoadImage() , CatchSprite() and LoadSprite() functions.

Parameters

None.

Return value

None.

Remarks

The TIFF format is very complex and internally supports several sub-compression formats such as: JPEG, LZW, etc. All forms of the TIFF format are supported, using this format will result in programs which are quite large.

See Also

`LoadImage()` , `CatchImage()` , `LoadSprite()` , `CatchSprite()`

Chapter 127

Joint

Overview

Joints are used to define a link between two entities , to have an automatic behavior when one of the entity is moved. It can be used to simulate real life interaction between two objects, like for example: a door, a train and so on. `InitEngine3D()` must be called successfully before using the joint functions.

127.1 EnableHingeJointAngularMotor

Syntax

```
EnableHingeJointAngularMotor(#Joint, Enable, TargetVelocity,  
                             MaxMotorImpulse)
```

Description

Enables the angular motor on the specified hinge joint.

Parameters

#Joint The joint to use. The joint has to be created with `HingeJoint()` .

Enable Enables or disables the hinge joint angular motor. If set to **#True**, the motor is enabled, if set to **#False**, the motor is disabled.

TargetVelocity The velocity the motor should produce.

MaxMotorImpulse The maximum impulse the motor can set.

Return value

None.

See Also

`HingeJoint()`

127.2 HingeJointMotorTarget

Syntax

```
HingeJointMotorTarget(#Joint, Angle, Velocity)
```

Description

Sets the specified hinge joint motor target.

Parameters

#Joint The joint to use. The joint has to be created with HingeJoint() .

Angle The angle (in degree) of the motor to reach.

Velocity The velocity the motor should produce.

Return value

None.

See Also

HingeJoint()

127.3 FreeJoint

Syntax

```
FreeJoint(#Joint)
```

Description

Frees the specified joint and releases all its associated memory. The joint must not be used (by using its number with the other functions in this library) after calling this function.

Parameters

#Joint The joint to free. If **#PB_All** is specified, all the remaining joints are freed.

Return value

None.

Remarks

All remaining joints are automatically freed when the program ends.

See Also

PointJoint() , HingeJoint() , ConeTwistJoint() , SliderJoint()

127.4 IsJoint

Syntax

```
Result = IsJoint(#Joint)
```

Description

Tests if the given joint number is a valid and correctly initialized joint.

Parameters

#Joint The joint to test.

Return value

Nonzero if the joint is valid, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a joint is ready to use.

127.5 GenericJoint

Syntax

```
Result = GenericJoint(#Joint, EntityID, TransformX, TransformY,  
    TransformZ, EntityID2, TransformX2, TransformY2, TransformZ2)
```

Description

Creates a new joint, based on one or two points.

Parameters

#Joint The number to identify the new joint. **#PB_Any** can be used to auto-generate this number.

EntityID The entity id associated to the joint. To get a valid entity id, use `EntityID()` .

TransformX, TransformY, TransformZ The transformation to apply for the first entity.

EntityID2 (optional) The second entity associated to the joint. If this parameter isn't specified, a single joint is created between the pivot point and the entity. If this parameter is specified, the first entity is anchored to the second entity. To get a valid entity id, use `EntityID()` .

TransformX2, TransformY2, TransformZ2 (optional) The transformation to apply for the second entity.

Return value

Returns zero if the joint can't be created. If **#PB_Any** is used as '**#Joint**' parameter, the new joint number will be returned as 'Result'.

Remarks

GetJointAttribute() and SetJointAttribute() can be used with the following attribute to change the joint behavior:

```
#PB_Joint_EnableSpring: sets to #True to enable spring, #False to
                        disable it (default)
#PB_Joint_Stiffness    : sets the stiffness. Values should be
                        between 1 and 10000
#PB_Joint_Damping      : sets the damping. Values should be
                        between 0 and 1 (0 means a very strong damping).
#PB_Joint_Position     : sets the position on the axis
#PB_Joint_NoLimit      : free axis position
#PB_Joint_LowerLimit   : lower limit
#PB_Joint_UpperLimit   : upper limit
```

See Also

FreeJoint() , GetJointAttribute() , SetJointAttribute()

127.6 PointJoint

Syntax

```
Result = PointJoint(#Joint, EntityID, PivotX, PivotY, PivotZ [,
                    EntityID2, PivotX2, PivotY2, PivotZ2])
```

Description

Creates a new joint, based on one or two points.

Parameters

#Joint The number to identify the new joint. #PB_Any can be used to auto-generate this number.

EntityID The entity id associated to the joint. To get a valid entity id, use EntityID() .

PivotX, PivotY, PivotZ The coordinates of the first pivot point for the joint. Relative to the center of the first entity.

EntityID2 (optional) The second entity associated to the joint. If this parameter isn't specified, a single joint is created between the pivot point and the entity. If this parameter is specified, the first entity is anchored to the second entity. To get a valid entity id, use EntityID() .

PivotX2, PivotY2, PivotZ2 The coordinates of the second pivot point for the joint. Relative to the center of the second entity.

Return value

Returns zero if the joint can't be created. If #PB_Any is used as '#Joint' parameter, the new joint number will be returned as 'Result'.

Remarks

GetJointAttribute() and SetJointAttribute() can be used with the following attribute to change the joint behavior:

```
#PB_PointJoint_Tau : Tau value of the joint
#PB_PointJoint_Damping: Damping value of the joint
```

See Also

FreeJoint() , GetJointAttribute() , SetJointAttribute()

127.7 HingeJoint

Syntax

```
Result = HingeJoint(#Joint, EntityID, PivotX, PivotY, PivotZ,  
    AxisX, AxisY, AxisZ, EntityID2, PivotX2, PivotY2, PivotZ2,  
    AxisX2, AxisY2, AxisZ2)
```

Description

Creates a new hinge joint between the two given entities. Hinge can be used to simulate door, moving bridge etc.

Parameters

#Joint The number to identify the new joint. #PB_Any can be used to auto-generate this number.

EntityID The first entity id associated to the joint. To get a valid entity id, use EntityID() .

PivotX, PivotY, PivotZ The coordinates of the first pivot point for the joint. Relative to the center of the first entity.

AxisX, AxisY, AxisZ The orientation of the axis for the joint.

EntityID2 The second entity id associated to the joint. To get a valid entity id, use EntityID() .

PivotX2, PivotY2, PivotZ2 The coordinates of the second pivot point for the joint. Relative to the center of the second entity.

AxisX2, AxisY2, AxisZ2 The orientation of the axis for the joint.

Return value

Returns zero if the joint can't be created. If #PB_Any is used as '#Joint' parameter, the new joint number will be returned as 'Result'.

Remarks

GetJointAttribute() and SetJointAttribute() can be used with the following attribute to change the joint behavior:

```
#PB_HingeJoint_LowerLimit : Lower limit of the joint  
#PB_HingeJoint_UpperLimit : Upper limit of the joint
```

See Also

FreeJoint() , GetJointAttribute() , SetJointAttribute() , EnableHingeJointAngularMotor()

127.8 ConeTwistJoint

Syntax

```
Result = ConeTwistJoint(#Joint, EntityID, TransformX, TransformY,  
                        TransformZ, EntityID2, TransformX2, TransformY2, TransformZ2)
```

Description

Creates a new cone twist joint between the two given entities. Cone twist can be used to attach arm or legs to body, etc.

Parameters

#Joint The number to identify the new joint. #PB_Any can be used to auto-generate this number.

EntityID The first entity id associated to the joint. To get a valid entity id, use EntityID() .

TransformX, TransformY, TransformZ The transformation to apply for the first entity.

EntityID2 The second entity id used to create the joint. To get a valid entity id, use EntityID() .

TransformX2, TransformY2, TransformZ2 The transformation to apply for the second entity.

Return value

Returns zero if the joint can't be created. If #PB_Any is used as '#Joint' parameter, the new joint number will be returned as 'Result'.

Remarks

GetJointAttribute() and SetJointAttribute() can be used with the following attribute to change the joint behavior:

```
#PB_ConeTwistJoint_SwingSpan : First swing span of the joint  
#PB_ConeTwistJoint_SwingSpan2 : Second swing span of the joint  
#PB_ConeTwistJoint_TwistSpan : Twist span of the joint
```

See Also

FreeJoint() , GetJointAttribute() , SetJointAttribute()

127.9 SliderJoint

Syntax

```
Result = SliderJoint(#Joint, EntityID, TransformX, TransformY,  
                    TransformZ, EntityID2, TransformX2, TransformY2, TransformZ2)
```

Description

Create a new slider joint between the two given entities. Slider can be used to move an entity with constraint on a plane surface, etc.

Parameters

#Joint The number to identify the new joint. #PB_Any can be used to auto-generate this number.

EntityID The first entity id associated to the joint. To get a valid entity id, use EntityID() .

TransformX The X value of the transformation for the first entity.

TransformY The Y value of the transformation for the first entity.

TransformZ The Z value of the transformation for the first entity.

EntityID2 The second entity used to create the joint. To get a valid entity id, use EntityID() .

TransformX2 The X value of the transformation for the second entity.

TransformY2 The Y value of the transformation for the second entity.

TransformZ2 The Z value of the transformation for the second entity.

Return value

Returns zero if the joint can't be created. If #PB_Any is used as '#Joint' parameter, the new joint number will be returned as 'Result'.

Remarks

GetJointAttribute() and SetJointAttribute() can be used with the following attribute to change the joint behavior:

```
#PB_SliderJoint_LowerLimit : Lower limit of the joint
#PB_SliderJoint_UpperLimit : Upper limit of the joint
```

See Also

FreeJoint() , GetJointAttribute() , SetJointAttribute()

127.10 GetJointAttribute

Syntax

```
Result = GetJointAttribute(#Joint, Attribute)
```

Description

Get the specified attribute value of the given joint and its associated entities.

Parameters

#Joint The joint to use.

Attribute The attribute to get.

Return value

Returns the value of the specified attribute or 0 if the joint does not support the attribute.

Remarks

This function is available for all joints which support attributes. See the individual joint commands for the supported attributes:

- PointJoint()
- ConeTwistJoint()
- HingeJoint()
- SliderJoint()

See Also

SetJointAttribute()

127.11 SetJointAttribute

Syntax

```
SetJointAttribute(#Joint, Attribute, Value [, Axis])
```

Description

Set the specified attribute value of the given joint and its associated entities.

Parameters

#Joint The joint to use.

Attribute The attribute to set.

Value Value of the attribute to set.

Axis (optional) The Axis to use to set the attribute. Only supported by GenericJoint() , the axis are indexed as following:

```
0, 1 and 2: translation axis (x, y, z)
3, 4 and 5: rotation axis (x, y, z)
```

Return value

None.

Remarks

This function is available for all joints which support attributes. See the individual joint commands for the supported attributes:

- GenericJoint()
- PointJoint()
- ConeTwistJoint()
- HingeJoint()
- SliderJoint()

See Also

GetJointAttribute()

Chapter 128

Joystick

Overview

PureBasic provides full access to the joysticks which are connected to the computer. This library supports joysticks and complex gamepads with several pads, triggers and many buttons.

Remarks

On Windows, this library uses the DirectX technology. On Linux it uses SDL.

128.1 InitJoystick

Syntax

```
Result = InitJoystick()
```

Description

Initialize the joystick environment for later use. This function must be called before any other functions within this library.

Return value

Returns the number of joysticks available for use.

Remarks

This function can be called on a regular basis to detect if new joysticks have been connected.

See Also

ExamineJoystick()

128.2 ExamineJoystick

Syntax

```
Result = ExamineJoystick(#Joystick)
```

Description

Updates the current joystick state. It needs to be called before using the following functions: JoystickButton() , JoystickAxisX() , JoystickAxisY() and JoystickAxisZ() .

Parameters

#Joystick The joystick to use. The first joystick index is 0. The number of available joysticks is returned by InitJoystick() .

Return value

Returns nonzero if the joystick state has been correctly updated, zero otherwise.

Remarks

In fullscreen mode the command FlipBuffers() needs to be called inside your event loop to handle all events properly.

See Also

JoystickButton() , JoystickAxisX() , JoystickAxisY() , JoystickAxisZ() .

128.3 JoystickAxisX

Syntax

```
Result = JoystickAxisX(#Joystick [, Pad [, Mode]])
```

Description

Returns the joystick X axis state.

Parameters

#Joystick The joystick to use. The first joystick index is 0. The number of available joysticks is returned by InitJoystick() .

Pad (optional) The pad to use, if the joystick has multiple pads. The first pad index is 0.

Mode (optional) The mode can be one of the following values:

```
#PB_Absolute: Returned value is either -1 (left), 0 (no  
movement) or 1 (right) (default)  
#PB_Relative: Returned value is between the range -1000  
(left) and 1000 (right). If the gamepad  
doesn't support relative movement, the result  
will be -1000, 0 or 1000.
```

Return value

Returns the joystick X axis value, depending of the specified mode.

Remarks

ExamineJoystick() has to be called before this function is used, to update the current joystick state.

See Also

ExamineJoystick() , JoystickAxisY() , JoystickAxisZ()

128.4 JoystickAxisY

Syntax

```
Result = JoystickAxisY(#Joystick [, Pad [, Mode]])
```

Description

Returns the joystick Y axis state.

Parameters

#Joystick The joystick to use. The first joystick index is 0. The number of available joysticks is returned by InitJoystick() .

Pad (optional) The pad to use, if the joystick has multiple pads. The first pad index is 0.

Mode (optional) The mode can be one of the following values:

```
#PB_Absolute: Returned value is either -1 (up), 0 (no
               movement) or 1 (down) (default)
#PB_Relative: Returned value is between the range -1000 (up)
               and 1000 (down). If the gamepad doesn't
               support relative movement, the result will be
               -1000, 0 or 1000.
```

Return value

Returns the joystick Y axis value, depending of the specified mode.

Remarks

ExamineJoystick() has to be called before this function is used, to update the current joystick state.

See Also

ExamineJoystick() , JoystickAxisX() , JoystickAxisZ()

128.5 JoystickAxisZ

Syntax

```
Result = JoystickAxisZ(#Joystick [, Pad [, Mode]])
```

Description

Returns the joystick Z axis state. This axis is often referred to as trigger on new gamepads.

Parameters

#Joystick The joystick to use. The first joystick index is 0. The number of available joysticks is returned by `InitJoystick()` .

Pad (optional) The pad to use, if the joystick has multiple pads. The first pad index is 0.

Mode (optional) The mode can be one of the following values:

```
#PB_Absolute: Returned value is either -1, 0 (no movement) or  
1 (default)  
#PB_Relative: Returned value is between the range -1000 and  
1000. If the gamepad doesn't  
support relative movement, the result will be  
-1000, 0 or 1000.
```

Return value

Returns the joystick Z axis value, depending of the specified mode.

Remarks

`ExamineJoystick()` has to be called before this function is used, to update the current joystick state.

See Also

`ExamineJoystick()` , `JoystickAxisX()` , `JoystickAxisY()`

128.6 JoystickName

Syntax

```
Result\$_ = JoystickName(#Joystick)
```

Description

Returns the joystick's name. It can be useful when having several joysticks connected, to identify the right one.

Parameters

#Joystick The joystick to use. The first joystick index is 0. The number of available joysticks is returned by `InitJoystick()` .

Return value

Returns the joystick name.

See Also

`InitJoystick()`

128.7 JoystickButton

Syntax

```
Result = JoystickButton(#Joystick, Button)
```

Description

Returns the joystick button state.

Parameters

#Joystick The joystick to use. The first joystick index is 0. The number of available joysticks is returned by `InitJoystick()` .

Button The joystick button to query. The first button index is 1.

Returns zero if the specified button is not pressed, else it returns nonzero. Any number of buttons may be pressed at the same time.

Remarks

`ExamineJoystick()` has to be called before this function is used, to update the current joystick state.

See Also

`ExamineJoystick()`

Chapter 129

Json

Overview

The JSON library provides functions to parse, create or modify data in JSON format. JSON (JavaScript Object Notation) is a lightweight data-interchange format supported by many programming languages. An introduction to the format can be found [here](#). This library understands and produces the JSON format as defined by [RFC-7159](#).

129.1 AddJSONElement

Syntax

```
Result = AddJSONElement(JSONValue [, Index])
```

Description

Add a new array element to a JSON value of type `#PB_JSON_Array`.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Array`.

Index (optional) The index at which the new value will be inserted into the array. If the index is outside of the range of the array, the new value will be inserted either at the start (for `Index < 0`) or the end of the array. If this parameter is not specified, the new value is added at the end of the array.

Return value

Returns the address of the added JSON value. The newly added value initially has type `#PB_JSON_Null`.

Example

```
1  If CreateJSON(0)
2      ArrayValue = SetJSONArray(JSONValue(0))
3
4      ; add element at the end
5      For i = 1 To 5
6          NumValue = AddJSONElement(ArrayValue)
7          SetJSONInteger(NumValue, i)
8      Next i
```



```

9
10     ; insert at a specific index
11     StrValue = AddJSONElement(ArrayValue, 1)
12     SetJSONString(StrValue, "Hello")
13
14     Debug ComposeJSON(0)
15 EndIf

```

See Also

SetJSONArray() , RemoveJSONElement() , ResizeJSONElements() , ClearJSONElements() ,
GetJSONElement() , JSONArraySize() , JSONType()

129.2 AddJSONMember

Syntax

```
Result = AddJSONMember(JSONValue, Key$)
```

Description

Add a new member to a JSON value of type `#PB_JSON_Object`. If a member with the specified key already exists, it will be replaced.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Object`.

Key\$ The key for the new member. If a member with the same key exists in the object, it will be replaced.

Return value

Returns the address of the added JSON member value. The newly added value initially has type `#PB_JSON_Null`.

Example

```

1  If CreateJSON(0)
2      ObjectValue = SetJSONObject(JSONValue(0))
3
4      FirstName = AddJSONMember(ObjectValue, "FirstName")
5      SetJSONString(FirstName, "John")
6
7      LastName = AddJSONMember(ObjectValue, "LastName")
8      SetJSONString(LastName, "Smith")
9
10     Debug ComposeJSON(0)
11 EndIf

```

See Also

SetJSONObject() , RemoveJSONMember() , ClearJSONMembers() , GetJSONMember() ,
ExamineJSONMembers() , JSONObjectSize() , JSONType()

129.3 CatchJSON

Syntax

```
Result = CatchJSON(#JSON, *Buffer, Size [, Flags])
```

Description

Parse JSON data from a memory buffer. The contents of the memory buffer are expected to be encoded in UTF-8 format. The JSONValue() function can be used to access the contained JSON value(s) after parsing.

Parameters

#JSON A number to identify the new JSON. #PB_Any can be used to auto-generate this number.

***Buffer** A readable memory location.

Length Length (in bytes) of the memory location.

Flags (optional) If set to #PB_JSON_NoCase, the JSON data will be parsed case insensitive. The default is to be case sensitive.

Return value

Nonzero if the JSON data was parsed correctly, zero otherwise. If #PB_Any was used for the #JSON parameter then the generated number is returned on success.

Remarks

In case of an error, the JSONErrorMessage(), JSONErrorLine() and JSONErrorPosition() functions can be used to get more information about the error.

To parse JSON data directly from a string, the ParseJSON() function can be used instead.

JSON is a case sensitive data format. However, in some situations, such as deserializing structures with ExtractJSONStructure() or similar commands it may be useful to treat JSON objects as case insensitive. The #PB_JSON_NoCase flag causes all member related functions to treat keys in this object as case insensitive.

See Also

CreateJSON(), ParseJSON(), LoadJSON(), JSONValue(), FreeJSON(), JSONErrorMessage(), JSONErrorLine(), JSONErrorPosition(), ExportJSON()

129.4 ClearJSONElements

Syntax

```
ClearJSONElements(JSONValue)
```

Description

Remove all array elements from a JSON value of type #PB_JSON_Array.

Parameters

JSONValue The JSON value. The value must be of type #PB_JSON_Array.

Return value

None.

Example

```
1 ParseJSON(0, "[1, 2, 3, 4]")
2
3 ; clear the values and add a new string
4 ClearJSONElements(JSONValue(0))
5 SetJSONString(AddJSONElement(JSONValue(0)), "Hello")
6
7 Debug ComposeJSON(0)
```

See Also

SetJSONArray(), AddJSONElement(), RemoveJSONElement(), ResizeJSONElements(),
GetJSONElement(), JSONArraySize(), JSONType()

129.5 ClearJSONMembers

Syntax

```
ClearJSONMembers(JSONValue)
```

Description

Remove all object members from a JSON value of type `#PB_JSON_Object`.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Object`.

Return value

None.

Example

```
1 Input$ = "{ " + Chr(34) + "x" + Chr(34) + ": 10, " + Chr(34) +
   "y" + Chr(34) + ": 20 }"
2 Debug Input$
3 ParseJSON(0, Input$)
4
5 ; clear the members and add a new one
6 ClearJSONMembers(JSONValue(0))
7 SetJSONString(AddJSONMember(JSONValue(0), "Hello"), "World")
8
9 Debug ComposeJSON(0)
```

See Also

SetJSONObject(), AddJSONMember(), RemoveJSONMember(), GetJSONMember(),
ExamineJSONMembers(), JSONObjectSize(), JSONType()

129.6 ComposeJSON

Syntax

```
Result\$$ = ComposeJSON(#JSON [, Flags])
```

Description

Compose the given JSON data into a string. A string can be parsed back into JSON data using the ParseJSON() function.

Parameters

#JSON The JSON to compose.

Flags (optional) If set to **#PB_JSON_PrettyPrint**, the composed string will contain additional newline and whitespace for better readability. The extra whitespace is not significant to the JSON format. The output will have the same meaning to a JSON reader with or without this flag.

Return value

The JSON data as a string.

Remarks

The output string has the string format of the executable (Ascii or Unicode). JSON is generally encoded in UTF-8, so when writing the result string to a file or sending it to another application, it is advised to convert the string to UTF-8 before doing so.

Example

```
1  If CreateJSON(0)
2      Person = SetJSONObject(JSONValue(0))
3      SetJSONString(AddJSONMember(Person, "FirstName"), "John")
4      SetJSONString(AddJSONMember(Person, "LastName"), "Smith")
5      SetJSONInteger(AddJSONMember(Person, "Age"), 42)
6
7      Debug ComposeJSON(0, #PB_JSON_PrettyPrint)
8  EndIf
```

See Also

SaveJSON() , ExportJSON() , ExportJSONSize() , ParseJSON()

129.7 CreateJSON

Syntax

```
Result = CreateJSON(#JSON [, Flags])
```

Description

Create new, empty JSON data. Initially, the data will contain a JSON value of type **#PB_JSON_Null**. The JSONValue() function can be used to access this value to change it.

Parameters

#JSON A number to identify the new JSON. **#PB_Any** can be used to auto-generate this number.

Flags (optional) If set to **#PB_JSON_NoCase**, the JSON data will be treated as case insensitive. The default is to be case sensitive.

Return value

Nonzero if the JSON data was created correctly, zero otherwise. If **#PB_Any** was used for the **#JSON** parameter then the generated number is returned on success.

Remarks

JSON is a case sensitive data format. However, in some situations, such as deserializing structures with `ExtractJSONStructure()` or similar commands it may be useful to treat JSON objects as case insensitive. The **#PB_JSON_NoCase** flag causes all member related functions to treat keys in this object as case insensitive.

Example

```
1  If CreateJSON(0)
2      Person = SetJSONObject(JSONValue(0))
3      SetJSONString(AddJSONMember(Person, "FirstName"), "John")
4      SetJSONString(AddJSONMember(Person, "LastName"), "Smith")
5      SetJSONInteger(AddJSONMember(Person, "Age"), 42)
6
7      Debug ComposeJSON(0, #PB_JSON_PrettyPrint)
8  EndIf
```

See Also

`CatchJSON()` , `LoadJSON()` , `ParseJSON()` , `JSONValue()` , `FreeJSON()`

129.8 ExamineJSONMembers

Syntax

```
Result = ExamineJSONMembers(JSONValue)
```

Description

Starts to examine the members of a JSON value of type **#PB_JSON_Object**. The individual members can be examined with the `NextJSONMember()` , `JSONMemberKey()` and `JSONMemberValue()` functions.

Parameters

JSONValue The JSON value to examine. The value must be of type **#PB_JSON_Object**.

Return value

Returns nonzero if the object can be enumerated or zero if there was an error.

Example

```
1   Input$ = "{ " + Chr(34) + "x" + Chr(34) + ": 10, " +  
2           Chr(34) + "y" + Chr(34) + ": 20, " +  
3           Chr(34) + "z" + Chr(34) + ": 30 }"  
4  
5   ParseJSON(0, Input$)  
6   ObjectValue = JSONValue(0)  
7  
8   If ExamineJSONMembers(ObjectValue)  
9       While NextJSONMember(ObjectValue)  
10          Debug JSONMemberKey(ObjectValue) + " = " +  
            GetJSONInteger(JSONMemberValue(ObjectValue))  
11      Wend  
12  EndIf
```

See Also

NextJSONMember(), JSONMemberKey(), JSONMemberValue(), GetJSONMember(),
SetJSONObject(), JSONType()

129.9 ExportJSON

Syntax

```
Result = ExportJSON(#JSON, *Buffer, Size [, Flags])
```

Description

Export the given JSON data to a memory location. The JSON data will be encoded in UTF-8 format.

Parameters

#JSON The JSON to export.

***Buffer** A writable memory location.

Size The size of the memory location. If the size is not large enough to hold the entire JSON data, the function will fill the memory location with data, but then return failure. The ExportJSONSize() function can be used to determine the needed size.

Flags (optional) If set to **#PB_JSON_PrettyPrint**, the composed string will contain additional newline and whitespace for better readability. The extra whitespace is not significant to the JSON format. The output will have the same meaning to a JSON reader with or without this flag.

Return value

Returns the number of bytes written to the memory location on success. If the function fails, the result is 0.

See Also

ExportJSONSize(), ComposeJSON(), SaveJSON(), CatchJSON()

129.10 ExportJSONSize

Syntax

```
Result = ExportJSONSize(#JSON [, Flags])
```

Description

Returns the size in bytes needed to successfully export the given JSON data to a memory buffer with the specified flags.

Parameters

#JSON The JSON to export.

Flags (optional) The flags to be used in the corresponding call to ExportJSON(). The only allowed value is **#PB_JSON_PrettyPrint**.

Return value

The number of bytes needed to export the JSON data.

See Also

ExportJSON(), ComposeJSON(), SaveJSON()

129.11 ExtractJSONArray

Syntax

```
ExtractJSONArray(JSONValue, Array())
```

Description

Extract elements from the given JSON value of type **#PB_JSON_Array** into the specified Array(). The array will be resized to the number of elements contained in the JSON value.

Parameters

JSONValue The JSON value. The value must be of type **#PB_JSON_Array**.

Array() The array to fill with the JSON elements. The array will be resized to have the same size as the JSON value. Any previous content of the array will be lost.

Return value

None.

Remarks

The extraction is performed recursively if the array has a structure type. If the JSON value contains any elements that do not have the proper type to match the Array(), they will be ignored and the corresponding array element will be left empty.

If the specified Array() has more than one dimension, the JSON data is expected to be a nested array of arrays to represent the multi-dimensional data. See the below example for more details.

Example

```
1 ParseJSON(0, "[1, 3, 5, 7, 9]")
2
3 Dim a(0)
4 ExtractJSONArray(JSONValue(0), a())
5
6 For i = 0 To ArraySize(a())
7     Debug a(i)
8 Next i
```

Example

```
1 ParseJSON(0, "[[0, 1, 2], [3, 4, 5], [6, 7, 8]]")
2
3 Dim a(0, 0)
4 ExtractJSONArray(JSONValue(0), a())
5
6 For x = 0 To 2
7     For y = 0 To 2
8         Debug a(x, y)
9     Next y
10 Next x
```

See Also

ExtractJSONList() , ExtractJSONMap() , ExtractJSONStructure() , InsertJSONArray() ,
InsertJSONList() , InsertJSONMap() , InsertJSONStructure() , SetJSONArray() , JSONType()

129.12 ExtractJSONList

Syntax

```
ExtractJSONList(JSONValue, List())
```

Description

Extract elements from the given JSON value of type `#PB_JSON_Array` into the specified List(). The list will be resized to the number of elements contained in the JSON value.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Array`.

List() The list to fill with the JSON elements. The list will be resized to have the same size as the JSON value. Any previous content of the list will be lost.

Return value

None.

Remarks

The extraction is performed recursively if the list has a structure type. If the JSON value contains any elements that do not have the proper type to match the List(), they will be ignored and the corresponding list element will be left empty.

Example

```
1  Input$ = "[ {\" + Chr(34) + \"x\" + Chr(34) + \": 10, \" + Chr(34) +  
   \"y\" + Chr(34) + \": 20\", \" +  
2    \"{\" + Chr(34) + \"x\" + Chr(34) + \": 30, \" + Chr(34) +  
   \"y\" + Chr(34) + \": 50\", \" +  
3    \"{\" + Chr(34) + \"x\" + Chr(34) + \": -5, \" + Chr(34) +  
   \"y\" + Chr(34) + \": 100\" } ] \"  
4  
5  Structure Location  
6    x.l  
7    y.l  
8  EndStructure  
9  
10 NewList Locations.Location()  
11  
12 ParseJSON(0, Input$)  
13 ExtractJSONList(JSONValue(0), Locations())  
14  
15 ForEach Locations()  
16   Debug Str(Locations()\x) + ", " + Str(Locations()\y)  
17 Next
```

See Also

ExtractJSONArray() , ExtractJSONMap() , ExtractJSONStructure() , InsertJSONArray() ,
InsertJSONList() , InsertJSONMap() , InsertJSONStructure() , SetJSONArray() , JSONType()

129.13 ExtractJSONMap

Syntax

```
ExtractJSONMap( JSONValue , Map() )
```

Description

Extract members from the given JSON value of type `#PB_JSON_Object` into the specified Map(). The map will be resized to the number of elements contained in the JSON value.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Object`.

Map() The map to fill with the JSON elements. The map will be resized to have the same size as the JSON value. Any previous content of the map will be lost.

Return value

None.

Remarks

The extraction is performed recursively if the map has a structure type. If the JSON value contains any members that do not have the proper type to match the Map(), they will be ignored and the corresponding map element will be left empty.

Example

```
1  Input$ = "{" + Chr(34) + "enabled" + Chr(34) + ": 1, " +  
2           Chr(34) + "displayed" + Chr(34) + ": 1, " +  
3           Chr(34) + "visible" + Chr(34) + ": 0 }"  
4  ParseJSON(0, Input$)  
5  
6  NewMap Options()  
7  ExtractJSONMap(JSONValue(0), Options())  
8  
9  Debug Options("enabled")  
10 Debug Options("visible")
```

See Also

ExtractJSONArray() , ExtractJSONList() , ExtractJSONStructure() , InsertJSONArray() ,
InsertJSONList() , InsertJSONMap() , InsertJSONStructure() , SetJSONObject() , JSONType()

129.14 ExtractJSONStructure

Syntax

```
ExtractJSONStructure(JSONValue, *Buffer, Structure [, Flags])
```

Description

Extract members from the given JSON value of type `#PB_JSON_Object` into the specified structure memory. The structure will be cleared of any previous content before extracting the JSON values, unless `#PB_JSON_NoClear` flag is set.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Object`.

***Buffer** The address of the structure memory to fill.

Structure The type of the structure to fill.

Flags (optional) If set to `#PB_JSON_NoClear`, the structure won't be cleared before extracting the JSON data: if the JSON data doesn't specify a structure field, the current field value will be kept. If not specified, the whole structure will be cleared before extracting data from JSON.

Return value

None.

Remarks

The extraction is performed recursively if the structure contains further structures, arrays, lists or maps. If the JSON value contains any members that do not have the proper type to match a structure member they will be ignored and the corresponding structure member is left empty. Any '*' or '\$' characters are stripped from the structure member names before comparing them to the JSON object members. So a member key must not include these characters to be properly matched to a structure member.

The comparison of member keys to structure member names is performed case sensitive. If the `#JSON` data was created or parsed with the `#PB_JSON_NoCase` flag, the comparison is performed case insensitive.

Example

```
1  Structure Person
2      Name$
3      Age.l
4      List Books.s()
5  EndStructure
6
7  Input$ = "{" + Chr(34) + "Name" + Chr(34) + ": " + Chr(34) +
8      "John Smith" + Chr(34) + ", " +
9      Chr(34) + "Age" + Chr(34) + ": 42, " +
10     Chr(34) + "Books" + Chr(34) + ": [" +
11     Chr(34) + "Investing For Dummies" +
12     Chr(34) + ", " +
13     Chr(34) + "A Little Bit of Everything
14     For Dummies" + Chr(34) + "]" }"
15
16 ParseJSON(0, Input$)
17 ExtractJSONStructure(JSONValue(0), @P.Person, Person)
18
19 Debug P\Name$
20 Debug P\Age
21 Debug ListSize(P\Books())
```

See Also

ExtractJSONArray() , ExtractJSONList() , ExtractJSONMap() , InsertJSONArray() ,
InsertJSONList() , InsertJSONMap() , InsertJSONStructure() , SetJSONObject() , JSONType()

129.15 FreeJSON

Syntax

FreeJSON(#JSON)

Description

Frees the JSON data and its contained values.

Parameters

#JSON The JSON data to free. If **#PB_All** is specified, all the remaining JSON objects are freed.

Return value

None.

Remarks

All remaining JSON objects are automatically freed when the program ends.

See Also

IsJSON() , CreateJSON() , ParseJSON() , LoadJSON() , CatchJSON()

129.16 GetJSONBoolean

Syntax

```
Result = GetJSONBoolean(JSONValue)
```

Description

Return the boolean value of a JSON value of type `#PB_JSON_Boolean`. A JSON value can be set to a boolean with `SetJSONBoolean()` .

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Boolean`.

Return value

The boolean value `#True` or `#False`.

Example

```
1 ParseJSON(0, "[true, true, false]")
2
3 Debug GetJSONBoolean(GetJSONElement(JSONValue(0), 0))
4 Debug GetJSONBoolean(GetJSONElement(JSONValue(0), 1))
5 Debug GetJSONBoolean(GetJSONElement(JSONValue(0), 2))
```

See Also

`SetJSONBoolean()` , `GetJSONDouble()` , `GetJSONElement()` , `GetJSONFloat()` ,
`GetJSONInteger()` , `GetJSONMember()` , `GetJSONString()` , `GetJSONQuad()` , `JSONType()`

129.17 GetJSONDouble

Syntax

```
Result.d = GetJSONDouble(JSONValue)
```

Description

Return the value of a JSON value of type `#PB_JSON_Number` as a double precision floating point value.

A JSON value can be set to a number with `SetJSONDouble()` , `SetJSONFloat()` , `SetJSONInteger()` or `SetJSONQuad()` .

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Number`.

Return value

The number as a double.

Example

```
1 ParseJSON(0, "[1, 1.23, 1.23e-3]")
2
3 Debug GetJSONDouble(GetJSONElement(JSONValue(0), 0))
4 Debug GetJSONDouble(GetJSONElement(JSONValue(0), 1))
5 Debug GetJSONDouble(GetJSONElement(JSONValue(0), 2))
```

See Also

SetJSONDouble(), SetJSONFloat(), SetJSONInteger(), SetJSONQuad(), GetJSONBoolean(),
GetJSONElement(), GetJSONFloat(), GetJSONInteger(), GetJSONMember(),
GetJSONString(), GetJSONQuad(), JSONType()

129.18 GetJSONElement

Syntax

```
Result = GetJSONElement(JSONValue, Index)
```

Description

Return the JSON array element at the given 'Index' of a JSON value of type `#PB_JSON_Array`.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Array`.

Index The index of the array element to return. The index must be between 0 and `JSONArraySize() - 1`.

Return value

The address of the JSON value at the specified array index. If the given 'Index' is out of range, the result is 0.

Example

```
1 ParseJSON(0, "[1, 2, 3, 4, 5]")
2
3 For i = 0 To JSONArraySize(JSONValue(0)) - 1
4     Debug GetJSONInteger(GetJSONElement(JSONValue(0), i))
5 Next i
```

See Also

SetJSONArray(), AddJSONElement(), RemoveJSONElement(), ResizeJSONElements(),
ClearJSONElements(), JSONArraySize(), JSONType()

129.19 GetJSONFloat

Syntax

```
Result.f = GetJSONFloat(JSONValue)
```

Description

Return the value of a JSON value of type `#PB_JSON_Number` as a single precision floating point value.

A JSON value can be set to a number with `SetJSONDouble()` , `SetJSONFloat()` , `SetJSONInteger()` or `SetJSONQuad()` .

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Number`.

Return value

The number as a float.

Example

```
1 ParseJSON(0, "[1, 1.23, 1.23e-3]")
2
3 Debug GetJSONFloat(GetJSONElement(JSONValue(0), 0))
4 Debug GetJSONFloat(GetJSONElement(JSONValue(0), 1))
5 Debug GetJSONFloat(GetJSONElement(JSONValue(0), 2))
```

See Also

`SetJSONDouble()` , `SetJSONFloat()` , `SetJSONInteger()` , `SetJSONQuad()` , `GetJSONBoolean()` , `GetJSONDouble()` , `GetJSONElement()` , `GetJSONInteger()` , `GetJSONMember()` , `GetJSONString()` , `GetJSONQuad()` , `JSONType()`

129.20 GetJSONInteger

Syntax

```
Result = GetJSONInteger(JSONValue)
```

Description

Return the value of a JSON value of type `#PB_JSON_Number` as an integer value.

A JSON value can be set to a number with `SetJSONDouble()` , `SetJSONFloat()` , `SetJSONInteger()` or `SetJSONQuad()` .

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Number`.

Return value

The number as an integer.

Example

```
1 ParseJSON(0, "[1, 2, 3, 4, 5]")
2
3 For i = 0 To JSONArraySize(JSONValue(0)) - 1
4     Debug GetJSONInteger(GetJSONElement(JSONValue(0), i))
5 Next i
```

See Also

SetJSONDouble(), SetJSONFloat(), SetJSONInteger(), SetJSONQuad(), GetJSONBoolean(),
GetJSONDouble(), GetJSONElement(), GetJSONFloat(), GetJSONMember(),
GetJSONString(), GetJSONQuad(), JSONType()

129.21 GetJSONMember

Syntax

Result = GetJSONMember(JSONValue, Key\$)

Description

Return the JSON object member with the given Key\$ of a JSON value of type
`#PB_JSON_Object`.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Object`.

Key\$ The key of the member to return.

The key is compared case sensitive unless the `#PB_JSON_NoCase` flag has been specified
when creating or parsing the JSON data.

Return value

The address of the JSON value with the specified key. If the given 'Key\$' does not exist in the
object, the result is 0.

Example

```
1 Input$ = "{ " + Chr(34) + "x" + Chr(34) + ": 10, " +
2           Chr(34) + "y" + Chr(34) + ": 20, " +
3           Chr(34) + "z" + Chr(34) + ": 30 }"
4
5 ParseJSON(0, Input$)
6
7 Debug GetJSONInteger(GetJSONMember(JSONValue(0), "x"))
8 Debug GetJSONInteger(GetJSONMember(JSONValue(0), "y"))
9 Debug GetJSONInteger(GetJSONMember(JSONValue(0), "z"))
```

See Also

SetJSONObject(), AddJSONMember(), RemoveJSONMember(), ClearJSONMembers(),
ExamineJSONMembers(), JSONObjectSize(), JSONType()

129.22 GetJSONString

Syntax

`Result\$$ = GetJSONString(JSONValue)`

Description

Return the value of a JSON value of type `#PB_JSON_String` as a string.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_String`.

Return value

The string contained in the JSON value.

Example

```
1 ParseJSON(0, Chr(34) + "The quick brown fox jumped over the lazy  
   dog" + Chr(34))  
2  
3 Debug GetJSONString(JSONValue(0))
```

See Also

`SetJSONString()` , `GetJSONBoolean()` , `GetJSONDouble()` , `GetJSONElement()` ,
`GetJSONFloat()` , `GetJSONInteger()` , `GetJSONMember()` , `GetJSONQuad()` , `JSONType()`

129.23 GetJSONQuad

Syntax

`Result.q = GetJSONQuad(JSONValue)`

Description

Return the value of a JSON value of type `#PB_JSON_Number` as an quad value.
A JSON value can be set to a number with `SetJSONDouble()` , `SetJSONFloat()` ,
`SetJSONInteger()` or `SetJSONQuad()` .

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Number`.

Return value

The number as an quad.

Example

```
1 ParseJSON(0, "[1, 2, 3, 4, 5]")
2
3 For i = 0 To JSONArraySize(JSONValue(0)) - 1
4     Debug GetJSONQuad(GetJSONElement(JSONValue(0), i))
5 Next i
```

See Also

SetJSONDouble(), SetJSONFloat(), SetJSONInteger(), SetJSONQuad(), GetJSONBoolean(),
GetJSONDouble(), GetJSONElement(), GetJSONFloat(), GetJSONInteger(),
GetJSONMember(), GetJSONString(), JSONType()

129.24 InsertJSONArray

Syntax

```
InsertJSONArray(JSONValue, Array())
```

Description

Insert the specified Array() into the given JSON value. The JSON value will be changed to type `#PB_JSON_Array`.

Parameters

JSONValue The JSON value. The previous content of the value will be changed to the content of the Array().

Array() The array to insert into the JSON value.

Return value

None.

Remarks

If the specified Array() has more than one dimension, the JSON value will be filled with a nested array of arrays to represent the multi-dimensional data. See the below example for more details.

Example

```
1 Dim Colors.s(3)
2 Colors(0) = "red"
3 Colors(1) = "yellow"
4 Colors(2) = "green"
5 Colors(3) = "blue"
6
7 If CreateJSON(0)
8     InsertJSONArray(JSONValue(0), Colors())
9     Debug ComposeJSON(0)
10 EndIf
```

Example

```
1 Dim matrix(2, 2)
2 matrix(0, 0) = 1
3 matrix(1, 1) = 1
4 matrix(2, 2) = 1
5
6 If CreateJSON(0)
7     InsertJSONArray(JSONValue(0), matrix())
8     Debug ComposeJSON(0)
9 EndIf
```

See Also

InsertJSONList() , InsertJSONMap() , InsertJSONStructure() , ExtractJSONArray() ,
ExtractJSONList() , ExtractJSONMap() , ExtractJSONStructure() ,

129.25 InsertJSONList

Syntax

```
InsertJSONList(JSONValue, List())
```

Description

Insert the specified List() into the given JSON value. The JSON value will be changed to type `#PB_JSON_Array`.

Parameters

JSONValue The JSON value. The previous content of the value will be changed to the content of the List().

List() The list to insert into the JSON value.

Return value

None.

Example

```
1 NewList Names.s()
2 AddElement(Names()): Names() = "John"
3 AddElement(Names()): Names() = "Jane"
4 AddElement(Names()): Names() = "Jim"
5
6 If CreateJSON(0)
7     InsertJSONList(JSONValue(0), Names())
8     Debug ComposeJSON(0, #PB_JSON_PrettyPrint)
9 EndIf
```

See Also

InsertJSONArray() , InsertJSONMap() , InsertJSONStructure() , ExtractJSONArray() ,
ExtractJSONList() , ExtractJSONMap() , ExtractJSONStructure() ,

129.26 InsertJSONMap

Syntax

```
InsertJSONMap(JSONValue, Map())
```

Description

Insert the specified Map() into the given JSON value. The JSON value will be changed to type `#PB_JSON_Object`.

Parameters

JSONValue The JSON value. The previous content of the value will be changed to the content of the Map().

Map() The map to insert into the JSON value.

Return value

None.

Example

```
1  NewMap Colors()  
2  Colors("red") = $0000FF  
3  Colors("green") = $00FF00  
4  Colors("blue") = $FF0000  
5  
6  If CreateJSON(0)  
7      InsertJSONMap(JSONValue(0), Colors())  
8      Debug ComposeJSON(0, #PB_JSON_PrettyPrint)  
9  EndIf
```

See Also

InsertJSONArray(), InsertJSONList(), InsertJSONStructure(), ExtractJSONArray(),
ExtractJSONList(), ExtractJSONMap(), ExtractJSONStructure(),

129.27 InsertJSONStructure

Syntax

```
InsertJSONStructure(JSONValue, *Buffer, Structure)
```

Description

Insert the contents of the specified structure memory into the given JSON value. The JSON value will be changed to type `#PB_JSON_Object` and contain one member for each member in the structure.

Parameters

JSONValue The JSON value. The previous content of the value will be changed to the content of the structure.

***Buffer** The address of the structure to insert into the JSON value.

Structure The type of the structure to insert.

Return value

None.

Example

```
1  Structure Person
2      FirstName$
3      LastName$
4      Age.l
5      List Books.s()
6  EndStructure
7
8  Define P.Person
9      P\FirstName$ = "John"
10     P\LastName$  = "Smith"
11     P\Age        = 42
12     AddElement(P\Books()): P\Books() = "Investing For Dummies"
13     AddElement(P\Books()): P\Books() = "English Grammar For Dummies"
14     AddElement(P\Books()): P\Books() = "A Little Bit of Everything
15                                     For Dummies"
16
17 If CreateJSON(0)
18     InsertJSONStructure(JSONValue(0), @P, Person)
19     Debug ComposeJSON(0, #PB_JSON_PrettyPrint)
20 EndIf
```

See Also

InsertJSONArray(), InsertJSONList(), InsertJSONMap(), ExtractJSONArray(),
ExtractJSONList(), ExtractJSONMap(), ExtractJSONStructure(),

129.28 IsJSON

Syntax

```
Result = IsJSON(#JSON)
```

Description

Tests if the given #JSON number represents valid and correctly initialized JSON data.

Parameters

#JSON The JSON to use.

Return value

Nonzero if #JSON is valid JSON data, zero otherwise.

Remarks

This function is bulletproof and can be used with any value. If the 'Result' is not zero then the object is valid and initialized, otherwise it will equal zero.

See Also

CreateJSON() , CatchJSON() , LoadJSON() , ParseJSON() , FreeJSON()

129.29 JSONArraySize

Syntax

```
Result = JSONArraySize(JSONValue)
```

Description

Returns the number of elements in a JSON value of type #PB_JSON_Array.

Parameters

JSONValue The JSON value. The value must be of type #PB_JSON_Array.

Return value

The number of elements in the JSON array.

Example

```
1 ParseJSON(0, "[1, 2, null, true]")
2 Debug JSONArraySize(JSONValue(0))
```

See Also

SetJSONArray() , AddJSONElement() , RemoveJSONElement() , ResizeJSONElements() ,
ClearJSONElements() , GetJSONElement() , JSONType()

129.30 JSONErrorLine

Syntax

```
Result = JSONErrorLine()
```

Description

Returns the line number within the JSON input of the last failed JSON parsing operation with ParseJSON() , CatchJSON() or LoadJSON() .

Parameters

None.

Return value

The line number (1-based) of the last JSON parser error.

See Also

JSONErrorPosition() , JSONErrorMessage() , ParseJSON() , CatchJSON() , LoadJSON()

129.31 JSONErrorMessage

Syntax

```
Result\$$ = JSONErrorMessage()
```

Description

Returns a message describing the cause for the failure at the last JSON parsing operation with ParseJSON() , CatchJSON() or LoadJSON() .

Parameters

None.

Return value

The error message in english.

Example

```
1  If ParseJSON(0, "[1, 2, 3 4]")
2      ; work with the data
3  Else
4      Debug JSONErrorMessage()
5  EndIf
```

See Also

JSONErrorLine() , JSONErrorPosition() , ParseJSON() , CatchJSON() , LoadJSON()

129.32 JSONErrorPosition

Syntax

```
Result = JSONErrorPosition()
```

Description

Returns the character position within the line of the last failed JSON parsing operation with ParseJSON() , CatchJSON() or LoadJSON() .

Parameters

None.

Return value

The character position (1-based) of the last JSON parser error within the line reported by `JSONErrorLine()` .

See Also

`JSONErrorLine()` , `JSONErrorMessage()` , `ParseJSON()` , `CatchJSON()` , `LoadJSON()`

129.33 JSONMemberKey

Syntax

```
Result\$$ = JSONMemberKey(JSONValue)
```

Description

After a call to `NextJSONMember()` , returns the key of the currently examined JSON object member of the specified JSON value of type `#PB_JSON_Object`.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Object` and currently being examined with `ExamineJSONMembers()` .

Return value

The key of the current JSON object member.

Example

See `ExamineJSONMembers()` for an example.

See Also

`ExamineJSONMembers()` , `NextJSONMember()` , `JSONMemberValue()`

129.34 JSONMemberValue

Syntax

```
Result = JSONMemberValue(JSONValue)
```

Description

After a call to `NextJSONMember()` , returns the address of the currently examined JSON object member of the specified JSON value of type `#PB_JSON_Object`.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Object` and currently being examined with `ExamineJSONMembers()` .

Return value

The address of the current JSON object member.

Example

See `ExamineJSONMembers()` for an example.

See Also

`ExamineJSONMembers()` , `NextJSONMember()` , `JSONMemberKey()`

129.35 JSONObjectSize

Syntax

```
Result = JSONObjectSize(JSONValue)
```

Description

Returns the number of members in a JSON value of type `#PB_JSON_Object`.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Object`.

Return value

The number of members in the JSON object.

Example

```
1 Input$ = "{ " + Chr(34) + "x" + Chr(34) + ": 10, " +  
2           Chr(34) + "y" + Chr(34) + ": 20, " +  
3           Chr(34) + "z" + Chr(34) + ": 30 }"  
4  
5 ParseJSON(0, Input$)  
6 Debug JSONObjectSize(JSONValue(0))
```

See Also

`SetJSONObject()` , `AddJSONMember()` , `RemoveJSONMember()` , `ClearJSONMembers()` ,
`GetJSONMember()` , `ExamineJSONMembers()` , `JSONType()`

129.36 JSONType

Syntax

`Result = JSONType(JSONValue)`

Description

Returns the type of the given JSON value.

Parameters

JSONValue The JSON value.

Return value

It can be one of the following:

`#PB_JSON_Null`

The value represents the JSON literal null.

`#PB_JSON_String`

The value contains a string. `GetJSONString()` can be used to read the string.

`#PB_JSON_Number`

The value contains a number. `GetJSONDouble()` , `GetJSONFloat()` , `GetJSONInteger()` or `GetJSONQuad()` can be used to read the number.

`#PB_JSON_Boolean`

The value contains a boolean. `GetJSONBoolean()` can be used to read the value.

`#PB_JSON_Array`

The value contains an array of JSON elements. `JSONArraySize()` returns the size of the array. `GetJSONElement()` can be used to get a specific array element. `AddJSONElement()` , `RemoveJSONElement()` , `ResizeJSONElements()` or `ClearJSONElements()` can be used to modify the array.

`#PB_JSON_Object`

The value contains an object (a set of key/value pairs). `JSONObjectSize()` returns the number of members in the object. `GetJSONMember()` returns a specific member value. `ExamineJSONMembers()` can be used to examine the member values. `AddJSONMember()` , `RemoveJSONMember()` or `ClearJSONMembers()` can be used to modify the object.

Example

```
1 ; A procedure that accepts any JSON value and returns a string
2 ;
3 Procedure.s GetAnyValue(Value)
4     Select JSONType(Value)
5         Case #PB_JSON_Null: ProcedureReturn "null"
6         Case #PB_JSON_String: ProcedureReturn GetJSONString(Value)
```

```

7      Case #PB_JSON_Number: ProcedureReturn
      StrD(GetJSONDouble(Value))
8      Case #PB_JSON_Boolean: ProcedureReturn
      Str(GetJSONBoolean(Value))
9      Case #PB_JSON_Array:   ProcedureReturn "array"
10     Case #PB_JSON_Object:  ProcedureReturn "object"
11     EndSelect
12 EndProcedure
13
14 ParseJSON(0, "[1, 2, true, null, " + Chr(34) + "hello" + Chr(34)
+ "]" )
15 For i = 0 To JSONArraySize(JSONValue(0)) - 1
16     Debug GetAnyValue(GetJSONElement(JSONValue(0), i))
17 Next i

```

See Also

JSONValue() , SetJSONArray() , SetJSONBoolean() , SetJSONDouble() , SetJSONFloat() , SetJSONInteger() , SetJSONNull() , SetJSONObject() , SetJSONString() , SetJSONQuad()

129.37 JSONValue

Syntax

Result = JSONValue(#JSON)

Description

Returns the value of the specified #JSON data. The type of the value can be checked with JSONType() .

Parameters

#JSON The JSON data to return the value of.

Return value

The JSON value. The result is never 0 for a valid #JSON data.

Remarks

Every #JSON data contains exactly one JSON value (containing possibly nested values). Newly created #JSON data from CreateJSON() contains a value of type #PB_JSON_Null.

The type of the JSON value or its content can be modified with one of the following functions:

- SetJSONArray() : Change the value to an (empty) array
- SetJSONBoolean() : Change the value to a boolean
- SetJSONDouble() : Change the value to a number
- SetJSONFloat() : Change the value to a number
- SetJSONInteger() : Change the value to a number
- SetJSONNull() : Change the value to a 'null'
- SetJSONObject() : Change the value to an (empty) object
- SetJSONString() : Change the value to a string
- SetJSONQuad() : Change the value to a number

Example

```
1 ParseJSON(0, Chr(34) + "The quick brown fox jumped over the lazy  
dog" + Chr(34))  
2  
3 Debug GetJSONString(JSONValue(0))
```

See Also

JSONType()

129.38 LoadJSON

Syntax

```
Result = LoadJSON(#JSON, FileName$ [, Flags])
```

Description

Parse JSON data from a file. The contents of the file are expected to be encoded in UTF-8 format. Files with another character encoding cannot be read by this command. The JSONValue() function can be used to access the contained JSON value(s) after parsing.

Parameters

#JSON A number to identify the new JSON. #PB_Any can be used to auto-generate this number.

FileName\$ The name of the file containing the JSON data.

Flags (optional) If set to #PB_JSON_NoCase, the JSON data will be parsed case insensitive. The default is to be case sensitive.

Return value

Nonzero if the JSON data was parsed correctly, zero otherwise. If #PB_Any was used for the #JSON parameter then the generated number is returned on success.

Remarks

In case of an error, the JSONErrorMessage() , JSONErrorLine() and JSONErrorPosition() functions can be used to get more information about the error.

JSON is a case sensitive data format. However, in some situations, such as deserializing structures with ExtractJSONStructure() or similar commands it may be useful to treat JSON objects as case insensitive. The #PB_JSON_NoCase flag causes all member related functions to treat keys in this object as case insensitive.

See Also

CreateJSON() , CatchJSON() , ParseJSON() , JSONValue() , FreeJSON() , JSONErrorMessage() , JSONErrorLine() , JSONErrorPosition() , SaveJSON()

129.39 NextJSONMember

Syntax

```
Result = NextJSONMember(JSONValue)
```

Description

After a call to `ExamineJSONMembers()`, this function is used to iterate over all members of the specified JSON value of type `#PB_JSON_Object`.
`JSONMemberKey()` and `JSONMemberValue()` can be used to get information about the current member.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Object` and `ExamineJSONMembers()` must have been called on this value.

Return value

Returns non-zero if another JSON member was found. If the result is zero then there are no more JSON members to be examined.

Example

See `ExamineJSONMembers()` for an example.

See Also

`ExamineJSONMembers()`, `JSONMemberKey()`, `JSONMemberValue()`

129.40 ParseJSON

Syntax

```
Result = ParseJSON(#JSON, Input$ [, Flags])
```

Description

Parse JSON data from a string. The `JSONValue()` function can be used to access the contained JSON value(s) after parsing.

Parameters

#JSON A number to identify the new JSON. `#PB_Any` can be used to auto-generate this number.

Input\$ The string containing the JSON data to parse.

Flags (optional) If set to `#PB_JSON_NoCase`, the JSON data will be parsed case insensitive. The default is to be case sensitive.

Return value

Nonzero if the JSON data was parsed correctly, zero otherwise. If `#PB_Any` was used for the `#JSON` parameter then the generated number is returned on success.

Remarks

In case of an error, the `JSONErrorMessage()` , `JSONErrorLine()` and `JSONErrorPosition()` functions can be used to get more information about the error.

To parse JSON data directly from a memory buffer, the `CatchJSON()` function can be used instead. JSON is a case sensitive data format. However, in some situations, such as deserializing structures with `ExtractJSONStructure()` or similar commands it may be useful to treat JSON objects as case insensitive. The `#PB_JSON_NoCase` flag causes all member related functions to treat keys in this object as case insensitive.

Example

```
1  If ParseJSON(0, "[1, 2, 3, 4, 5]")
2      For i = 0 To JSONArraySize(JSONValue(0)) - 1
3          Debug GetJSONInteger(GetJSONElement(JSONValue(0), i))
4      Next i
5  Else
6      Debug JSONErrorMessage()
7  EndIf
```

See Also

`CreateJSON()` , `CatchJSON()` , `LoadJSON()` , `JSONValue()` , `FreeJSON()` , `JSONErrorMessage()` , `JSONErrorLine()` , `JSONErrorPosition()` , `ExportJSON()` , `ComposeJSON()`

129.41 RemoveJSONElement

Syntax

```
RemoveJSONElement(JSONValue, Index)
```

Description

Remove the element at the specified index from a JSON value of type `#PB_JSON_Array`.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Array`.

Index The index of the element to remove. The value must be between 0 and `JSONArraySize()` - 1.

Return value

None.

Example

```
1  ParseJSON(0, "[1, 2, 3, 4, 5]")
2  RemoveJSONElement(JSONValue(0), 2)
3  Debug ComposeJSON(0)
```

See Also

SetJSONArray() , AddJSONElement() , ResizeJSONElements() , ClearJSONElements() ,
GetJSONElement() , JSONArraySize() , JSONType()

129.42 RemoveJSONMember

Syntax

```
RemoveJSONMember( JSONValue , Key$ )
```

Description

Remove the member with the specified key from a JSON value of type `#PB_JSON_Object`.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Object`.

Key\$ The key of the member to remove.

Return value

None.

Example

```
1   Input$ = "{ " + Chr(34) + "x" + Chr(34) + ": 10, " +  
2               Chr(34) + "y" + Chr(34) + ": 20, " +  
3               Chr(34) + "z" + Chr(34) + ": 30 }"  
4  
5   ParseJSON(0, Input$)  
6   RemoveJSONMember( JSONValue(0), "x")  
7   Debug ComposeJSON(0, #PB_JSON_PrettyPrint)
```

See Also

SetJSONObject() , AddJSONMember() , ClearJSONMembers() , GetJSONMember() ,
ExamineJSONMembers() , JSONObjectSize() , JSONType()

129.43 ResizeJSONElements

Syntax

```
ResizeJSONElements( JSONValue , Size )
```

Description

Resize a JSON value of type `#PB_JSON_Array` so that it has the given number of elements.

Parameters

JSONValue The JSON value. The value must be of type `#PB_JSON_Array`.

Size The new size of the array. This specifies the total number of elements (not the index of the highest array element like `Dim`).

Return value

None.

Remarks

If new elements are added to the array, they will have the type `#PB_JSON_Null`.

Example

```
1 ParseJSON(0, "[1, 2, 3, 4, 5]")
2
3 ResizeJSONElements(JSONValue(0), 3)
4 Debug ComposeJSON(0)
5
6 ResizeJSONElements(JSONValue(0), 10)
7 Debug ComposeJSON(0)
```

See Also

SetJSONArray() , AddJSONElement() , RemoveJSONElement() , ClearJSONElements() ,
GetJSONElement() , JSONArraySize() , JSONType()

129.44 SaveJSON

Syntax

```
Result = SaveJSON(#JSON, FileName$ [, Flags])
```

Description

Save the given JSON data to a file. The file will be encoded in UTF-8 (without a leading byte-order mark).

Parameters

#JSON The JSON to save.

FileName\$ The name of the file containing the JSON data.

Flags (optional) If set to `#PB_JSON_PrettyPrint`, the saved data will contain additional newline and whitespace for better readability. The extra whitespace is not significant to the JSON format. The output will have the same meaning to a JSON reader with or without this flag.

Return value

Returns non-zero if the file was saved successfully. If there is an error while saving the file, the result is zero.

See Also

ComposeJSON() , ExportJSON() , ExportJSONSize() , LoadJSON()

129.45 SetJSONArray

Syntax

```
Result = SetJSONArray(JSONValue)
```

Description

Change the type of the JSON value to `#PB_JSON_Array`. The array will have no elements (even if the value previously contained array elements).

Parameters

JSONValue The JSON value.

Return value

Returns the Array number, zero otherwise.

Example

```
1  If CreateJSON(0)
2      ArrayValue = SetJSONArray(JSONValue(0))
3      SetJSONString(AddJSONElement(ArrayValue), "hello")
4      SetJSONString(AddJSONElement(ArrayValue), "world")
5
6      Debug ComposeJSON(0)
7  EndIf
```

See Also

AddJSONElement() , RemoveJSONElement() , ResizeJSONElements() , ClearJSONElements() ,
GetJSONElement() , JSONArraySize() , JSONType()

129.46 SetJSONBoolean

Syntax

```
SetJSONBoolean(JSONValue, Value)
```

Description

Change the type of the JSON value to `#PB_JSON_Boolean` and store the given boolean value.

Parameters

JSONValue The JSON value.

Value The boolean value to store. A non-zero value is stored as `#True`, a value of 0 is stored as `#False`.

Return value

None.

Example

```
1  If CreateJSON(0)
2      ArrayValue = SetJSONArray(JSONValue(0))
3      SetJSONBoolean(AddJSONElement(ArrayValue), #True)
4      SetJSONBoolean(AddJSONElement(ArrayValue), #False)
5
6      Debug ComposeJSON(0)
7  EndIf
```

See Also

GetJSONBoolean(), SetJSONArray(), SetJSONDouble(), SetJSONFloat(), SetJSONInteger(), SetJSONNull(), SetJSONObject(), SetJSONString(), SetJSONQuad()

129.47 SetJSONDouble

Syntax

```
SetJSONDouble(JSONValue, Value.d)
```

Description

Change the type of the JSON value to `#PB_JSON_Number` and store the given double value.

Parameters

JSONValue The JSON value.

Value.d The value to store.

Return value

None.

Remarks

Note that JSON does not permit the special floating point values `+Infinity`, `-Infinity` or `NaN` in JSON data. If such a value is set with this function, it will be replaced by a JSON 'null' literal when the data is being saved or encoded. The functions `IsInfinity()` or `IsNaN()` can be used to detect this case.

Example

```
1  If CreateJSON(0)
2      ArrayValue = SetJSONArray(JSONValue(0))
3      SetJSONDouble(AddJSONElement(ArrayValue), 1.23)
4      SetJSONDouble(AddJSONElement(ArrayValue), 4.56)
5
6      Debug ComposeJSON(0)
7  EndIf
```

See Also

GetJSONDouble() , SetJSONArray() , SetJSONBoolean() , SetJSONFloat() , SetJSONInteger() , SetJSONNull() , SetJSONObject() , SetJSONString() , SetJSONQuad()

129.48 SetJSONFloat

Syntax

```
SetJSONFloat(JSONValue, Value.f)
```

Description

Change the type of the JSON value to `#PB_JSON_Number` and store the given float value.

Parameters

JSONValue The JSON value.

Value.f The value to store.

Return value

None.

Remarks

Note that JSON does not permit the special floating point values +Infinity, -Infinity or NaN in JSON data. If such a value is set with this function, it will be replaced by a JSON 'null' literal when the data is being saved or encoded. The functions `IsInfinity()` or `IsNaN()` can be used to detect this case.

Example

```
1  If CreateJSON(0)
2      ArrayValue = SetJSONArray(JSONValue(0))
3      SetJSONFloat(AddJSONElement(ArrayValue), 1.23)
4      SetJSONFloat(AddJSONElement(ArrayValue), 4.56)
5
6      Debug ComposeJSON(0)
7  EndIf
```

See Also

GetJSONFloat() , SetJSONArray() , SetJSONBoolean() , SetJSONDouble() , SetJSONInteger() , SetJSONNull() , SetJSONObject() , SetJSONString() , SetJSONQuad()

129.49 SetJSONInteger

Syntax

```
SetJSONInteger(JSONValue, Value)
```

Description

Change the type of the JSON value to `#PB_JSON_Number` and store the given integer value.

Parameters

JSONValue The JSON value.

Value The value to store.

Return value

None.

Example

```
1  If CreateJSON(0)
2      ArrayValue = SetJSONArray(JSONValue(0))
3      SetJSONInteger(AddJSONElement(ArrayValue), 1)
4      SetJSONInteger(AddJSONElement(ArrayValue), 2)
5      SetJSONInteger(AddJSONElement(ArrayValue), 3)
6
7      Debug ComposeJSON(0)
8  EndIf
```

See Also

GetJSONInteger(), SetJSONArray(), SetJSONBoolean(), SetJSONDouble(), SetJSONFloat(), SetJSONNull(), SetJSONObject(), SetJSONString(), SetJSONQuad()

129.50 SetJSONNull

Syntax

```
SetJSONNull(JSONValue)
```

Description

Clear the JSON value and set the type to #PB_JSON_Null.

Parameters

JSONValue The JSON value.

Return value

None.

Example

```
1  ParseJSON(0, "[1, 2, 3, 4, 5]")
2  SetJSONNull(GetJSONElement(JSONValue(0), 2))
3  SetJSONNull(GetJSONElement(JSONValue(0), 3))
4  Debug ComposeJSON(0)
```

See Also

SetJSONArray() , SetJSONBoolean() , SetJSONDouble() , SetJSONFloat() , SetJSONInteger() , SetJSONObject() , SetJSONString() , SetJSONQuad()

129.51 SetJSONObject

Syntax

```
Result = SetJSONObject(JSONValue)
```

Description

Change the type of the JSON value to `#PB_JSON_Object`. The object will have no members (even if the value previously contained object members).

Parameters

JSONValue The JSON value.

Return value

Returns the Object number, zero otherwise.

Example

```
1  If CreateJSON(0)
2      ObjectValue = SetJSONObject(JSONValue(0))
3      SetJSONInteger(AddJSONMember(ObjectValue, "x"), 10)
4      SetJSONInteger(AddJSONMember(ObjectValue, "y"), 20)
5      SetJSONInteger(AddJSONMember(ObjectValue, "z"), 30)
6
7      Debug ComposeJSON(0, #PB_JSON_PrettyPrint)
8  EndIf
```

See Also

AddJSONMember() , RemoveJSONMember() , ClearJSONMembers() , GetJSONMember() , ExamineJSONMembers() , JSONObjectSize() , JSONType()

129.52 SetJSONQuad

Syntax

```
SetJSONQuad(JSONValue, Value.q)
```

Description

Change the type of the JSON value to `#PB_JSON_Number` and store the given quad value.

Parameters

JSONValue The JSON value.

Value.q The value to store.

Return value

None.

Example

```
1  If CreateJSON(0)
2      ArrayValue = SetJSONArray(JSONValue(0))
3      SetJSONQuad(AddJSONElement(ArrayValue), 1)
4      SetJSONQuad(AddJSONElement(ArrayValue), 2)
5      SetJSONQuad(AddJSONElement(ArrayValue), 3)
6
7      Debug ComposeJSON(0)
8  EndIf
```

See Also

GetJSONQuad() , SetJSONArray() , SetJSONBoolean() , SetJSONDouble() , SetJSONFloat() , SetJSONInteger() , SetJSONNull() , SetJSONObject() , SetJSONString()

129.53 SetJSONString

Syntax

```
SetJSONString(JSONValue, String$)
```

Description

Change the type of the JSON value to `#PB_JSON_String` and store the given string.

Parameters

JSONValue The JSON value.

String\$ The string to store.

Return value

None.

Example

```
1  If CreateJSON(0)
2      ArrayValue = SetJSONArray(JSONValue(0))
3      SetJSONString(AddJSONElement(ArrayValue), "with escaped new" +
4      Chr(13) + Chr(10) + "line")
5      SetJSONString(AddJSONElement(ArrayValue), "with escaped \
6      backslash")
7
8      Debug ComposeJSON(0)
9  EndIf
```

See Also

GetJSONString() , SetJSONArray() , SetJSONBoolean() , SetJSONDouble() , SetJSONFloat() , SetJSONInteger() , SetJSONNull() , SetJSONObject() , SetJSONQuad()

Chapter 130

Keyboard

Overview

PureBasic provides fast and easy access to the keyboard. This capability should only be used in applications where raw and extremely fast access is required, such as in games for instance. It uses the DirectX technology.

This library was created for games and multimedia applications, which need fast keyboard access based on DirectX. For regular Windows applications, it is better to use the `AddKeyboardShortcut()` function.

130.1 InitKeyboard

Syntax

```
Result = InitKeyboard()
```

Description

Initializes the keyboard environment for later use. This function has to be called before any other function in this library.

Parameters

None.

Return value

Nonzero if the keyboard access can be initialized, zero otherwise.

130.2 ExamineKeyboard

Syntax

```
ExamineKeyboard()
```

Description

Updates the keyboard state. This function has to be called before using `KeyboardInkey()`, `KeyboardPushed()` or `KeyboardReleased()`.

Parameters

None.

Return value

None.

Remarks

In fullscreen mode the command `FlipBuffers()` need to be called inside your event loop to handle all events properly.

See Also

`KeyboardInkey()` , `KeyboardPushed()` `KeyboardReleased()` .

130.3 KeyboardInkey

Syntax

```
String\$_ = KeyboardInkey()
```

Description

Returns the last typed character, very useful when keyboard input is required within a gaming application, such as the name in highscore, in game console, etc.).

Parameters

None.

Return value

The last typed character.

Example

```
1  If InitSprite() And InitKeyboard() And OpenScreen(800, 600, 32,
   " ")
2      Repeat
3          FlipBuffers()
4          ClearScreen(RGB(0, 0, 0))
5
6          ExamineKeyboard()
7
8          ; If we press the 'Back' key, we will delete the last
   character
9          ;
10         If KeyboardReleased(#PB_Key_Back)
11             FullText$ = Left(FullText$, Len(FullText$)-1)
12         Else
13             result$=KeyboardInkey()
```

```

14         If FindString("1234567890
    'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz", result$)
    ; Or your chosen valid characters
15         FullText$ + result$
16         EndIf ; Add the new text to the current one (if any)
17     EndIf
18
19     ; Display the result
20     ;
21     If StartDrawing(ScreenOutput())
22         DrawingMode(1)
23         FrontColor(RGB(128, 255, 0))
24         DrawText(20, 20, "Just type some text...:")
25         DrawText(20, 40, FullText$)
26     StopDrawing()
27     EndIf
28     Until KeyboardPushed(#PB_Key_Escape)
29 EndIf

```

See Also

ExamineKeyboard()

130.4 KeyboardMode

Syntax

`KeyboardMode(Flags)`

Description

Changes the current behavior of the keyboard. This function affects KeyboardPushed() and KeyboardReleased() .

Parameters

Flags It can be a combination (using the '|' operator) of the following values:

```

#PB_Keyboard_Qwerty           : The keyboard ignores the
    default language layout and always uses the QWERTY one
    (default behavior).
#PB_Keyboard_International    : The keyboard uses the default
    language layout to map the keys (can be useful for non
    QWERTY keyboards).
#PB_Keyboard_AllowSystemKeys: The 'OS' system keys are
    allowed (like Win+R etc.). This can be annoying in
    fullscreen mode
if the user
    presses on them accidentally.

```

Return value

None.

See Also

KeyboardPushed() , KeyboardReleased() .

Supported OS

Windows

130.5 KeyboardPushed

Syntax

```
Result = KeyboardPushed(KeyID)
```

Description

Checks if the specified key is pressed. Any number of keys may be pressed at the same time. The function ExamineKeyboard() must be called before this function in order to update the keyboard state. The keyboard behavior can be changed with KeyboardMode() .

To check if a specified key has been pushed and released, see KeyboardReleased() .

Parameters

KeyID The identifier of the key to be checked. List of available keys:

`#PB_Key_All` ; *All keys are tested. Very useful for any key checks.*

`#PB_Key_1`
`#PB_Key_2`
`#PB_Key_3`
`#PB_Key_4`
`#PB_Key_5`
`#PB_Key_6`
`#PB_Key_7`
`#PB_Key_8`
`#PB_Key_9`
`#PB_Key_0`

`#PB_Key_A`
`#PB_Key_B`
`#PB_Key_C`
`#PB_Key_D`
`#PB_Key_E`
`#PB_Key_F`
`#PB_Key_G`
`#PB_Key_H`
`#PB_Key_I`
`#PB_Key_J`
`#PB_Key_K`
`#PB_Key_L`
`#PB_Key_M`
`#PB_Key_N`
`#PB_Key_O`
`#PB_Key_P`
`#PB_Key_Q`
`#PB_Key_R`
`#PB_Key_S`

#PB_Key_T
#PB_Key_U
#PB_Key_V
#PB_Key_W
#PB_Key_X
#PB_Key_Y
#PB_Key_Z

#PB_Key_Escape
#PB_Key_Minus
#PB_Key_Equals
#PB_Key_Back
#PB_Key_Tab
#PB_Key_LeftBracket
#PB_Key_RightBracket
#PB_Key_Return
#PB_Key_LeftControl
#PB_Key_SemiColon
#PB_Key_Apostrophe
#PB_Key_Grave
#PB_Key_LeftShift
#PB_Key_BackSlash
#PB_Key_Comma
#PB_Key_Period
#PB_Key_Slash
#PB_Key_RightShift
#PB_Key_Multiply
#PB_Key_LeftAlt
#PB_Key_Space
#PB_Key_Capital
#PB_Key_F1
#PB_Key_F2
#PB_Key_F3
#PB_Key_F4
#PB_Key_F5
#PB_Key_F6
#PB_Key_F7
#PB_Key_F8
#PB_Key_F9
#PB_Key_F10
#PB_Key_F11
#PB_Key_F12
#PB_Key_NumLock
#PB_Key_Scroll
#PB_Key_Pad0
#PB_Key_Pad1
#PB_Key_Pad2
#PB_Key_Pad3
#PB_Key_Pad4
#PB_Key_Pad5
#PB_Key_Pad6
#PB_Key_Pad7
#PB_Key_Pad8
#PB_Key_Pad9
#PB_Key_Add
#PB_Key_Subtract
#PB_Key_Decimal
#PB_Key_PadEnter
#PB_Key_RightControl

```

#PB_Key_PadComma
#PB_Key_Divide
#PB_Key_RightAlt
#PB_Key_Pause
#PB_Key_Home
#PB_Key_Up
#PB_Key_Down
#PB_Key_Left
#PB_Key_Right
#PB_Key_End
#PB_Key_PageUp
#PB_Key_PageDown
#PB_Key_Insert
#PB_Key_Delete

```

Return value

Nonzero if the specified key is pushed, zero otherwise.

Example

```

1  If InitSprite() And InitKeyboard() And OpenScreen(800,600,16,"")
2      Repeat
3          FlipBuffers()
4
5          If StartDrawing(ScreenOutput())
6              DrawText(0, 0, "Press ESC to quit")
7              StopDrawing()
8          EndIf
9
10         ExamineKeyboard()
11         If KeyboardPushed(#PB_Key_Escape)      ; press Esc to quit
12             End
13         EndIf
14     Forever
15 EndIf

```

See Also

ExamineKeyboard() , KeyboardReleased()

130.6 KeyboardReleased

Syntax

```
Result = KeyboardReleased(KeyID)
```

Description

Checks if the specified key has been pushed and released. This function is useful for switch key checks, like a 'Pause' key in a game (one time the game is paused, next time it will continue). The function ExamineKeyboard() must be called before this function to update the keyboard state. The keyboard behavior can be changed with KeyboardMode() .

Parameters

KeyID The identifier of the key to be checked. For a full list of available keys see `KeyboardPushed()` .

Return value

Nonzero if the specified key has been pushed and released, zero otherwise.

Example

```
1  If InitSprite() And InitKeyboard() And OpenScreen(800,600,16,"")
2      Paused = #False
3      Repeat
4          FlipBuffers()
5
6          If StartDrawing(ScreenOutput())
7
8              ExamineKeyboard()
9              If KeyboardReleased(#PB_Key_P)
10                 If Paused = #False
11                     Paused = #True
12                 Else
13                     Paused = #False
14                 EndIf
15             EndIf
16
17             DrawingMode(0)
18
19             If Paused = #False
20                 DrawText(20, 20, "Program is running...")
21             Else
22                 DrawText(20, 20, "Program paused...  ")
23             EndIf
24
25             StopDrawing()
26         EndIf
27     Until KeyboardPushed(#PB_Key_Escape)
28 EndIf
```

See Also

`ExamineKeyboard()` , `KeyboardPushed()`

Chapter 131

Library

Overview

Libraries are shared OS components which contain specific functions available to the programmer. For example, a library may contain functions designed to handle and manipulate pictures with ease. Each OS provides a number of shared libraries which help to ease the programmers life. With PureBasic, it is possible to access these external libraries not only easily, but also dynamically! The reason that libraries are so useful, is due to the fact that they are separate from the applications which use them, but at the same time shared between these applications. A library file requires only that it be loaded into memory once, this therefore saves memory, especially when it is a commonly used library. The programmer also benefits, since there is no need to keep re-inventing the wheel every time a clever feature is called for within an application. In addition, libraries are an excellent place to store procedures which are used by several programs. These libraries are easy to update, and when changes are made, there is no need to change the main executable which makes use of this library.

On Windows, shared libraries are known as: 'DLL's, or Dynamic Link Libraries.
On Linux shared libraries are known as: 'SO's, or System Objects.
On MacOS shared libraries are known as: 'DYLIB's, or Dynamic Libraries.
For more information on library calling standards (cdecl, stdcall, fastcall), see [here](#).

131.1 CloseLibrary

Syntax

```
CloseLibrary(#Library)
```

Description

Closes a library previously opened using OpenLibrary() , and frees any memory associated with this library.

Parameters

#Library The library to close. If **#PB_All** is specified, all the remaining libraries are closed.

Return value

None.

See Also

OpenLibrary()

131.2 CallCFunction

Syntax

```
Result = CallCFunction(#Library, FunctionName$ [,Parameter1 [,  
    Parameter2...]])
```

Description

Calls a function, in the specified library, in such a manner that the parameters are handled in the same fashion as a normal 'C' language function.

Parameters

#Library The library from which the function will be called.

FunctionName\$ The name of the function to call. The function name is case sensitive.

Parameter1, Parameter2, ... The parameters for the function. The number of parameters must match the parameters of the called function. The maximum number of supported parameters is 20.

Return value

Returns the return-value of the called function or zero if the library does not contain a function with the given name.

Remarks

If the function is described as 'cdecl' then this function must be used. However, most DLLs used within the Microsoft Windows OS do not use this form, so CallFunction() should be sufficient in most cases. To call a function that uses the 'stdcall' calling convention, use the CallFunction() function.

Note: This function is not very flexible and does not handle string/float/double/quad parameters or string/float/double/quad returns. The use of prototypes is now strongly recommended.

See Also

CallFunction() , GetFunction() , prototypes

131.3 CallCFunctionFast

Syntax

```
Result = CallCFunctionFast(*FunctionPointer [,Parameter1 [,  
    Parameter2...]])
```

Description

Calls a function directly, using its address. The function is expected to use the cdecl calling convention (the convention used by the C language).

Parameters

***FunctionPointer** The address of the function to call.

This pointer may be found using the `GetFunction()` , `GetFunctionEntry()` or `LibraryFunctionAddress()` functions. The use of this function is the fastest method with which to call library functions, especially when the results of a call to: `GetFunction()` or `LibraryFunctionAddress()` have been stored. This is due to the fact that this function is not required to search for the name of the library function.

Parameter1, Parameter2, ... The parameters for the function. The number of parameters must match the parameters of the called function. The maximum number of supported parameters is 20.

Return value

Returns the return-value of the called function.

Remarks

If the function is described as 'cdecl' then the use of this function is required. However, most DLLs used within the Microsoft Windows OS do not use this form, so `CallFunctionFast()` should be sufficient in most cases. To call a function that uses the 'stdcall' calling convention, use the `CallFunctionFast()` function.

Note: This function is not very flexible and does not handle string/float/double/quad parameters or string/float/double/quad returns. The use of prototypes is now strongly recommended.

Example

```
1  ProcedureC Function1()  
2      Debug "I call Function1 by its name"  
3  EndProcedure  
4  
5  NewMap *FuncPtr()  
6  *FuncPtr("Function1") = @Function1()  
7  
8  
9  CallCFunctionFast(*FuncPtr("Function1"))
```

See Also

`GetFunction()` , `CallFunctionFast()` , prototypes

131.4 CallFunction

Syntax

```
Result = CallFunction(#Library, FunctionName$ [,Parameter1 [,  
    Parameter2...]])
```

Description

Calls a function in the specified library, by using its name. The specified library must have previously been opened with the `OpenLibrary()` function. The function is expected to use the `stdcall` calling convention (the standard in most DLLs on Windows).

Parameters

#Library The library from which the function will be called.

FunctionName\$ The name of the function to call. The function name is case sensitive.

Parameter1, Parameter2, ... The parameters for the function. The number of parameters must match the parameters of the called function. The maximum number of supported parameters is 20.

Return value

Returns the return-value of the called function or zero if the library does not contain a function with the given name.

Remarks

To call a function that uses the 'cdecl' calling convention, use the `CallCFunction()` function.

Note: This function is not very flexible and does not handle string/float/double/quad parameters or string/float/double/quad returns. The use of prototypes is now strongly recommended.

See Also

`CallCFunction()` , `GetFunction()` , prototypes

131.5 CallFunctionFast

Syntax

```
Result = CallFunctionFast(*FunctionPointer [,Parameter1 [,  
    Parameter2...]])
```

Description

Calls a function directly, using its address. The function is expected to use the stdcall calling convention (the standard in most DLLs on Windows).

Parameters

***FunctionPointer** The address of the function to call.

This pointer may be found using the `GetFunction()` , `GetFunctionEntry()` or `LibraryFunctionAddress()` functions. The use of this function is the fastest method with which to call library functions, especially when the results of a call to: `GetFunction()` or `LibraryFunctionAddress()` have been stored. This is due to the fact that this function is not required to search for the name of the library function.

Parameter1, Parameter2, ... The parameters for the function. The number of parameters must match the parameters of the called function. The maximum number of supported parameters is 20.

Return value

Returns the return-value of the called function.

Remarks

To call a function that uses the 'cdecl' calling convention, use the `CallCFunctionFast()` function.
Note: This function is not very flexible and does not handle string/float/double/quad parameters or string/float/double/quad returns. The use of prototypes is now strongly recommended.

See Also

`CallCFunctionFast()` , `GetFunction()` , prototypes

131.6 CountLibraryFunctions

Syntax

```
Result = CountLibraryFunctions(#Library)
```

Description

Counts the number of functions available in a library. The library must be open when this function is called.

Parameters

#Library The number of the library which contains the functions to count. This number must be identical to that used previously with `OpenLibrary()` .

Return value

Returns the number of functions available in the library.

Supported OS

Windows, Linux

131.7 ExamineLibraryFunctions

Syntax

```
Result = ExamineLibraryFunctions(#Library)
```

Description

Initiates the process of examining the functions contained within a library.

Parameters

#Library The number of the library which contains the functions to examine. This number must be identical to that used previously with: `OpenLibrary()` .

Return value

Returns nonzero if the functions can be examined and zero if not.

Remarks

If this function succeeds, the programmer may then step through the functions in the library by using `NextLibraryFunction()` . At that point, information such as the name and address of each function may be obtained by using the `LibraryFunctionName()` and `LibraryFunctionAddress()` functions respectively.

See Also

`NextLibraryFunction()` , `LibraryFunctionAddress()` , `LibraryFunctionName()`

Supported OS

Windows, Linux

131.8 GetFunction

Syntax

```
Result = GetFunction(#Library, FunctionName$)
```

Description

Checks if the library, previously opened using the `OpenLibrary()` function, contains the given function and returns the function pointer.

Parameters

#Library The number of the library which contains the functions to find. This number must be identical to that used previously with `OpenLibrary()` .

FunctionName\$ The name of the function from which to get the pointer. The function name is case sensitive.

Return value

Returns the address to the function in the library on success and zero if the library does not contain a function with that name.

Remarks

The function can be called by its address using prototypes . The functions `CallFunctionFast()` and `CallCFunctionFast()` can also be used for that, but prototypes are the recommended method as they are more flexible.

See the prototype section for an example.

See Also

`GetFunctionEntry()` , `CallFunctionFast()` , `CallCFunctionFast()` , prototypes

131.9 GetFunctionEntry

Syntax

```
Result = GetFunctionEntry(#Library, FunctionEntry)
```

Description

Checks if the library, contains the given function entry. This searches for a library function by its position within the libraries function table, rather than by its name.

Parameters

#Library The number of the library which contains the functions to find. This number must be identical to that used previously with `OpenLibrary()` .

FunctionEntry A number representing the function index in the library. The first function is at index 1.

Return value

Returns the address to the function in the library on success and zero if the library does not contain a function with that index.

Remarks

The function can be called by its address using prototypes . The functions `CallFunctionFast()` and `CallCFunctionFast()` can also be used for that, but prototypes are the recommended method as they are more flexible.

See Also

`GetFunction()` , `CallFunctionFast()` , `CallCFunctionFast()` , prototypes

Supported OS

Windows

131.10 IsLibrary

Syntax

```
Result = IsLibrary(#Library)
```

Description

Tests if the given library number is valid and if the library has been correctly initialized.

Parameters

#Library The library value to test.

Return value

Returns nonzero if the input is a valid library and zero if not.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a library is ready for use.

See Also

OpenLibrary() , CloseLibrary()

131.11 LibraryFunctionAddress

Syntax

```
Result = LibraryFunctionAddress()
```

Description

Returns the address of the function in the library currently being examined with the ExamineLibraryFunctions() and NextLibraryFunction() functions.

Parameters

None.

Return value

Returns the address of the current function in memory. This is useful as parameter for the functions CallFunctionFast() or CallCFunctionFast() .

Remarks

The function can be called by its address using prototypes . The functions CallFunctionFast() and CallCFunctionFast() can also be used for that, but prototypes are the recommended method as they are more flexible.

See Also

ExamineLibraryFunctions() , NextLibraryFunction() , LibraryFunctionName() , CallFunctionFast() , CallCFunctionFast()

Supported OS

Windows, Linux

131.12 LibraryFunctionName

Syntax

```
Result\$$ = LibraryFunctionName()
```

Description

Returns the name of the function in the library currently being examined with the ExamineLibraryFunctions() and NextLibraryFunction() functions.

Parameters

None.

Return value

Returns the name of the current function.

See Also

ExamineLibraryFunctions() , NextLibraryFunction() , LibraryFunctionAddress()

Supported OS

Windows, Linux

131.13 LibraryID

Syntax

```
Result = LibraryID(#Library)
```

Description

Returns the unique ID which identifies the specified library in the operating system.

Parameters

#Library The library to use.

Return value

Returns the unique ID.

131.14 NextLibraryFunction

Syntax

```
Result = NextLibraryFunction()
```

Description

Moves to the next library function in an enumeration started with ExamineLibraryFunctions() .

Parameters

None.

Return value

Returns nonzero if the next library function was found and zero if there are no more functions to be examined.

See Also

ExamineLibraryFunctions() , LibraryFunctionName() , LibraryFunctionAddress()

Supported OS

Windows, Linux

131.15 OpenLibrary

Syntax

```
Result = OpenLibrary(#Library, Filename$)
```

Description

Opens a shared library in order that the functions within it may be accessed.

Parameters

#Library A number to identify this library. #PB_Any can be used to auto-generate this number.

Filename\$ The filename of the library to load. If the filename does not include a path, then the operating system will search for the library in its system folders, the applications directory and the current directory. On Linux and MacOS, to load the library from the current directory, the './' path prefix needs to be specified.

Return value

Returns nonzero if the library was opened successfully and zero if not. If #PB_Any was used as the #Library parameter then the generated number for the library is returned on success.

Remarks

It's mandatory, that the created PureBasic executable and the shared library are using the same architecture (for example a x86 DLL can't be opened by x64 executables).

See Also

CloseLibrary() , GetFunction() , prototypes

Chapter 132

Light

Overview

Lights are essential components in order to create worlds which look realistic. As with real life light, it is possible to specify many attributes of this light such as: the color, specular color, direction and more. `InitEngine3D()` must be called successfully before using the light functions.

132.1 CopyLight

Syntax

```
Result = CopyLight(#Light, #NewLight)
```

Description

Creates a new light which is the exact copy of the specified light. All light attributes such as: color, specular color, position etc. are duplicated.

Parameters

#Light The light to copy.

#NewLight A number to identify the new light. `#PB_Any` can be used to auto-generate this number.

Return value

Nonzero if the light was successfully duplicated, zero otherwise. If `#PB_Any` was used for the `#NewLight` parameter then the generated number is returned on success.

132.2 CreateLight

Syntax

```
Result = CreateLight(#Light, Color [, x, y, z [, Flags]])
```

Description

Creates a new `#Light` of the given color in the current world. If `WorldShadows()` is required, it must be called before creating lights.

Parameters

#Light A number to identify the light. `#PB_Any` can be used to auto-generate this number.

Color The color of the new light. Valid colors may be easily created using the `RGB()` function.

x, y, z (optional) The initial light absolute position. If omitted the light is created at position "0,0,0".

Flags (optional) It can be one of the following value:

```
#PB_Light_Point      : Creates a point light (the light is
                        emitted in all directions) (default).
#PB_Light_Directional : Creates a directional light.
#PB_Light_Spot        : Creates a spot type light.
                        SpotLightRange()
                        can be used to change the light behavior.
```

Return value

Nonzero if the light was successfully created, zero otherwise. If `#PB_Any` was used for the `#Light` parameter then the generated number is returned on success.

Example

```
1  CreateLight(0, RGB(255,0,0)) ; Creates a red light
2
3  CreateLight(1, RGB(0,255,0), 0, 100.7, 50) ; Creates a green
    light, at the position (0, 100.7, 50)
```

See Also

`FreeLight()`

132.3 FreeLight

Syntax

```
FreeLight(#Light)
```

Description

Frees the specified `#Light`. All its associated memory is released and at this point, the object no longer may be used.

Parameters

#Light The light to free. If `#PB_All` is specified, all the remaining lights are freed.

Return value

None.

Remarks

All remaining lights are automatically freed when the program ends.

See Also

CreateLight()

132.4 HideLight

Syntax

```
HideLight(#Light, State)
```

Description

Hides or shows the specified #Light.

Parameters

#Light The light to use.

State It can be one of following values:

```
#True : the #Light is hidden  
#False: the #Light is shown
```

Return value

None.

132.5 IsLight

Syntax

```
Result = IsLight(#Light)
```

Description

Tests if the given light is valid and correctly initialized.

Parameters

#Light The light to test.

Return value

Nonzero if the light is valid, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a light is ready to use.

See Also

CreateLight()

132.6 GetLightColor

Syntax

```
Result = GetLightColor(#Light, Type)
```

Description

Gets the specified light color value.

Parameters

#Light The light to use.

Type The color type to get. It can be one of the following value:

```
#PB_Light_DiffuseColor : get the diffuse color value.  
#PB_Light_SpecularColor: get the specular color value.
```

Return value

The specified light RGB color value.

See Also

SetLightColor()

132.7 SetLightColor

Syntax

```
SetLightColor(#Light, Type, Color)
```

Description

Changes the light color value.

Parameters

#Light The light to use.

Type The color type to change. It can be one of the following value:

```
#PB_Light_DiffuseColor : change the diffuse color.  
#PB_Light_SpecularColor: change the specular color.
```

Color The new RGB color value. It can be easily created with RGB() .

Return value

None.

See Also

GetLightColor()

132.8 SpotLightRange

Syntax

```
SpotLightRange(#Light, InnerAngle, OuterAngle [, Falloff])
```

Description

Changes the spot **#Light** behavior. The light has to be created with the **#PB_Light_Spot** flag.

Parameters

#Light The light to use.

InnerAngle Inner angle of the light.

OuterAngle Outer angle of the light.

Falloff Falloff of the light amongst the distance. A value of 1 means a linear falloff, a value less than 1 means a slower falloff while a value greater than 1 means a faster falloff.

Return value

None.

132.9 LightLookAt

Syntax

```
LightLookAt(#Light, x, y, z)
```

Description

Changes the **#Light** orientation in the world, to point towards the specified x,y,z point.

Parameters

#Light The light use.

x, y, z The point to look at.

Return value

None.

132.10 DisableLightShadows

Syntax

```
DisableLightShadows(#Light, State)
```

Description

Disables or enables the light shadow casting.

Parameters

#Light The light use.

State The new state of the light shadow casting. If sets to **#True**, the light shadow casting will be disabled. If sets to **#False** it will be enabled.

Return value

None.

132.11 MoveLight

Syntax

```
MoveLight(#Light, x, y, z [, Mode])
```

Description

Moves the specified light.

Parameters

#Light The light to use.

x, y, z The new position of the light.

Mode (optional) The move mode. It can be one of the following values:

```
#PB_Relative: relative move, from the current light position
               (default).
#PB_Absolute: absolute move to the specified position.
```

combined with one of the following values:

```
#PB_Local  : local move.
#PB_Parent : move relative to the parent position.
#PB_World  : move relative to the world.
```

Return value

None.

See Also

RotateLight()

132.12 LightDirection

Syntax

```
LightDirection(#Light, x, y, z)
```

Description

Changes the direction of a light. The position of the light is not changed.

Parameters

#Light The light to use.

x, y, z The direction vector (usually values between -1.0 and 1.0, if not it will be automatically normalized).

Return value

None.

See Also

LightDirectionX() , LightDirectionY() , LightDirectionZ()

132.13 LightDirectionX

Syntax

```
Result = LightDirectionX(#Light [, Mode])
```

Description

Returns the 'x' direction vector of the light.

Parameters

#Light The light to use.

Mode (optional) The mode to get the 'x' direction vector. It can be one of the following value:

```
#PB_Absolute: get the absolute 'x' direction vector of the  
light in the world (default).  
#PB_Relative: get the 'x' direction vector of the light  
relative to its parent.
```

Return value

The 'x' direction vector of the light. This value is always between -1.0 and 1.0.

See Also

LightDirectionY() , LightDirectionZ() , LightDirection()

132.14 LightDirectionY

Syntax

```
Result = LightDirectionY(#Light [, Mode])
```

Description

Returns the 'y' direction vector of the light.

Parameters

#Light The light to use.

Mode (optional) The mode to get the 'y' direction vector. It can be one of the following value:

```
#PB_Absolute: get the absolute 'y' direction vector of the
               light in the world (default).
#PB_Relative: get the 'y' direction vector of the light
               relative to its parent.
```

Return value

The 'y' direction vector of the light. This value is always between -1.0 and 1.0.

See Also

LightDirectionX() , LightDirectionZ() , LightDirection()

132.15 LightDirectionZ

Syntax

```
Result = LightDirectionZ(#Light [, Mode])
```

Description

Returns the 'z' direction vector of the light.

Parameters

#Light The light to use.

Mode (optional) The mode to get the 'z' direction vector. It can be one of the following value:

```
#PB_Absolute: get the absolute 'z' direction vector of the
               light in the world (default).
#PB_Relative: get the 'z' direction vector of the light
               relative to its parent.
```

Return value

The 'z' direction vector of the light. This value is always between -1.0 and 1.0.

See Also

LightDirectionX() , LightDirectionY() , LightDirection()

132.16 LightX

Syntax

```
Result = LightX(#Light [, Mode])
```

Description

Returns the current position of the light in the world.

Parameters

#Light The light to use.

Mode (optional) The mode to get the 'x' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'x' position of the light in  
the world (default).  
#PB_Relative: get the 'x' position of the light relative to  
its parent.
```

Return value

The 'x' position of the light.

See Also

LightY() , LightZ() , MoveLight()

132.17 LightY

Syntax

```
Result = LightY(#Light [, Mode])
```

Description

Returns the current position of the light in the world.

Parameters

#Light The light to use.

Mode (optional) The mode to get the 'y' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'y' position of the light in  
the world (default).  
#PB_Relative: get the 'y' position of the light relative to  
its parent.
```

Return value

The 'y' position of the light.

See Also

LightX() , LightZ() , MoveLight()

132.18 LightZ

Syntax

```
Result = LightZ(#Light [, Mode])
```

Description

Returns the current position of the light in the world.

Parameters

#Light The light to use.

Mode (optional) The mode to get the 'z' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'z' position of the light in  
the world (default).  
#PB_Relative: get the 'z' position of the light relative to  
its parent.
```

Return value

The 'z' position of the light.

See Also

LightX() , LightY() , MoveLight()

132.19 LightAttenuation

Syntax

```
LightAttenuation(#Light, Range, Attenuation)
```

Description

Changes the light attenuation.

Parameters

#Light The light to use.

Range The light range (in world unit) above which the light won't affect the world anymore.

Attenuation The global attenuation of the light. A value of 0.0 means no global attenuation (it will still attenuate depending of the range). It can be used to adjust the light brightness.

Return value

None.

See Also

SetLightColor()

132.20 RotateLight

Syntax

```
RotateLight(#Light, x, y, z [, Mode])
```

Description

Rotates the #Light according to the specified x,y,z angle values.

Parameters

#Light The light to use.

x, y, z The new rotation to apply to the light. Values are in degree ranging from 0 to 360.

Mode (optional) It can be one of the following value:

```
#PB_Absolute: absolute rotation (default).
#PB_Relative: relative rotation based on the previous light
rotation.
```

Return value

None.

132.21 LightRoll

Syntax

```
Result = LightRoll(#Light [, Mode])
```

Description

Gets the roll of the #Light.

Parameters

#Light The light to use.

Mode (optional) The mode to get the roll. It can be one of the following value:

```
#True : the roll is the raw value, but it can't be used in
RotateLight()
to get back the same orientation (default).
#False: the roll is adjusted, so it can put back in
RotateLight()
to get back the same orientation.
```

Return value

The current roll value of the light.

See Also

LightYaw() , LightPitch()

132.22 LightPitch

Syntax

```
Result = LightPitch(#Light [, Mode])
```

Description

Gets the pitch of the #Light.

Parameters

#Light The light to use.

Mode (optional) The mode to get the pitch. It can be one of the following value:

```
#True : the pitch is the raw value, but it can't be used in
        RotateLight()
to get back the same orientation (default).
#False: the pitch is adjusted, so it can be put back in
        RotateLight()
to get back the same orientation.
```

Return value

The current pitch value of the light.

See Also

LightYaw() , LightRoll()

132.23 LightYaw

Syntax

```
Result = LightYaw(#Light [, Mode])
```

Description

Gets the yaw of the #Light.

Parameters

#Light The light to use.

Mode (optional) The mode to get the yaw. It can be one of the following value:

```
#True : the yaw is the raw value, but it can't be used in
        RotateLight()
to get back the same orientation (default).
#False: the yaw is adjusted, so it can be put back in
        RotateLight()
to get back the same orientation.
```

Return value

The current yaw value of the light.

See Also

LightPitch() , LightRoll()

132.24 LightID

Syntax

```
LightID = LightID(#Light)
```

Description

Returns the unique system identifier of the light.

Parameters

#Light The light to use.

Return value

The unique system identifier of the light.

Chapter 133

List

Overview

Lists (also known as linked-lists) are structures for storing data which are dynamically allocated depending of your need. It is a list of elements (the data you want to store) and each element is fully independent of the others. You can add as many elements you want (or as many as will fit into the memory of your computer), insert elements at the position you need, delete some other and more. This kind of data management is very useful as it's one of the best ways to handle data when you do not know how many elements you will need to store, or if you are often changing how many elements there are.

Before you can work with Lists, you must declare them first. This could be done with the keyword `NewList` . For saving the contents are also often used structures .

Lists can be sorted using `SortList()` or `SortStructuredList()` , and can also be randomized using `RandomizeList()` .

To specifically search the contents of a List, using of loops is recommended: `For : Next` , `ForEach : Next` , `Repeat : Until` or `While : Wend` .

The first element is at position 0, the next at 1 and so on.

Other possibilities for storing data are the use of Arrays and Maps .

When using threads , the same list can not be accessed or modified at the same time in different threads. Mutexes can to be used to ensure the list is only used by one thread.

133.1 AddElement

Syntax

```
*Result = AddElement(List())
```

Description

Adds a new empty element after the current element or as the first item in the list if there are no elements in it. This new element becomes the current element of the list.

Parameters

List() The name of your list variable, created with the `NewList` function. You must include the parentheses after the list name.

Return value

Returns non-zero if the new element was created and zero otherwise. The value returned is a pointer to the new element data.

Example

```
1   ; The simplest way to use AddElement
2   NewList simple.w()
3   AddElement(simple())      ; Creates the first new element in the
   list
4   simple() = 23
5
6   AddElement(simple())      ; Current position is the first element,
   so we add one to the second position
7   simple() = 45
8
9
10  ; This shows how to use the return-value of AddElement
11  NewList advanced.l()
12  If AddElement(advanced()) <> 0
13      advanced() = 12345
14  Else
15      MessageRequester("Error!", "Unable to allocate memory for new
   element", #PB_MessageRequester_OK)
16  EndIf
17
18
19  ; A small structure to demonstrate the use of the pointer to the
   new element
20  Structure Programmer
21      Name.s
22      Strength.b
23  EndStructure
24
25  NewList Programmers.Programmer() ; The list for storing the
   elements
26
27  *Element.Programmer = AddElement(Programmers())
28  If *Element<>0
29      *Element\Name = "Dave"
30      *Element\Strength = 3      ; Wow, super-strong geek! ;)
31  Else
32      MessageRequester("Error!", "Unable to allocate memory for new
   element", #PB_MessageRequester_OK)
33  EndIf
```

See Also

InsertElement() , DeleteElement() , ClearList()

133.2 ChangeCurrentElement

Syntax

```
ChangeCurrentElement(List(), *NewElement)
```

Description

Changes the current element of the specified list to the given new element. This function is very useful if you want to "remember" an element, and restore it after performing other processing.

Parameters

List() The name of the list, created with the NewList function. You must include the parentheses after the list name.

***NewElement** The new element to set as the current element for the list. The element must be a pointer to another element which exists in this list. You should get this address by using the @ operator on the list name and not through any other method.

Return value

None.

Example: Simplified

```
1  *Old_Element = @mylist()    ; Get the address of the current
   element
2
3  ResetList(mylist())         ; Perform a search for all elements
   named
4  While NextElement(mylist()) ; "John" and change them to "J"
5      If mylist()\name = "John"
6          mylist()\name = "J"
7      EndIf
8  Wend
9
10 ChangeCurrentElement(mylist(), *Old_Element) ; Restore previous
   current element (from before the search)
```

Example: Complete

```
1  NewList myList()
2
3  AddElement(myList())
4  myList() = 100
5
6  AddElement(myList())
7  myList() = 200
8  *element = @myList()
9
10 AddElement(myList())
11 myList() = 300
12
13 Debug myList()                ; Displays 300 (last
   element)
14 ChangeCurrentElement(myList(), *element) ; Restore list position
15 Debug myList()                ; Displays 200
16
17 ForEach myList()
18     If @myList() = *element
19         Debug "element: " + myList() ; Displays "element:
   200"
20     EndIf
21 Next
```

See Also

SelectElement() , PushListPosition() , PopListPosition()

133.3 ClearList

Syntax

```
ClearList(List())
```

Description

Clears all the elements in this list and releases their memory. After this call the list is still usable, but the list is empty (i.e. there are no elements in it).

Parameters

List() The name of your list variable, created with the NewList function. You must include the parentheses after the list name.

Return value

None.

Remarks

PureBasic will only free the memory for the elements. If you have been using the list for something such as storing handles of objects that you create directly with the OS, there is no way PureBasic (or any other language) can know what they are. Therefore, in cases such as that, you should go through the elements in the list and free the objects yourself.

Example

```
1  NewList Numbers.w()
2
3  ; A small loop to add many items to the list
4  For i=1 To 100
5      AddElement(Numbers())
6      Numbers() = i
7  Next
8
9  ; Proof that items have been added to the list
10 MessageRequester("Information", "There are
    "+Str(ListSize(Numbers()))+" elements in the list",
    #PB_MessageRequester_OK)
11
12 ; Clear the list and show that the list really is empty
13 ClearList(Numbers())
14 MessageRequester("Information", "There are
    "+Str(ListSize(Numbers()))+" elements in the list",
    #PB_MessageRequester_OK)
```

See Also

DeleteElement() , FreeList()

133.4 CompareList

Syntax

```
Result = CompareList(List1(), List2() [, Flags])
```

Description

Compare each elements of the two lists for equality. Recursively compares also contents of structured lists with dynamic elements (such as embedded arrays, lists or maps). The two lists are considered the equal if they have the same type and size and if each pair of elements is equal.

Parameters

List1(), **List2()** The lists to compare.

Flags (optional) Can be a combination of the following values:

```
#PB_String_CaseSensitive : String comparison is case
                           sensitive (a=a). (default)
#PB_String_NoCase       : String comparison is case
                           insensitive(a=A).
#PB_Memory_FollowPointers: If a structure element is a
                           pointer that is not 0, recursively compare the pointer
                           target.

                           The default is to compare only
                           the pointer value itself. See remarks below for details.
```

Return value

Returns nonzero if both lists are the same or zero if they differ.

Remarks

The `#PB_Memory_FollowPointers` option is for advanced users and requires special care to avoid crashes. If this option is used then all pointer values must point to valid and initialized memory or have the value 0. It is also not allowed to have loops in the pointed elements (a chain of pointers that refers back to itself).

Example

```
1  NewList A$()
2  AddElement(A$()) : A$() = "Jim"
3  AddElement(A$()) : A$() = "John"
4  AddElement(A$()) : A$() = "Jack"
5
6  NewList B$()
7  AddElement(B$()) : B$() = "JIM" ; Case differs
8  AddElement(B$()) : B$() = "John"
9  AddElement(B$()) : B$() = "Jack"
10
11 Debug CompareList(A$(), B$()) ; Not equal
12 Debug CompareList(A$(), B$(), #PB_String_NoCase) ; Equal
```

See Also

CompareArray() , CompareMap()

133.5 CopyList

Syntax

```
Result = CopyList(SourceList(), DestinationList())
```

Description

Copy the contents of one list to another list. After a successful copy, the two lists are identical.

Parameters

SourceList() The list from which the elements will be copied.

DestinationList() The list to which the elements will be copied. The elements in this list before the copy will be deleted. If this list does not have the same type (native or structured) as the SourceList() then the copy will fail.

Return value

Returns non-zero if the copy succeeded and zero otherwise.

Example

```
1  NewList Friends$()
2  NewList FriendsCopy$()
3
4  AddElement(Friends$())
5  Friends$() = "John"
6
7  AddElement(Friends$())
8  Friends$() = "Elise"
9
10 CopyList(Friends$(), FriendsCopy$())
11
12 ForEach FriendsCopy$()
13     Debug FriendsCopy$()
14 Next
```

See Also

CopyArray() , CopyMap()

133.6 FreeList

Syntax

```
FreeList(List())
```

Description

Free the specified list and release all its associated memory. To access this list again later, NewList has to be called for it.

Parameters

List() The name of the list to free.

Return value

None.

See Also

ClearList()

133.7 ListSize

Syntax

```
Result = ListSize(List())
```

Description

Returns the number of elements in the list. It does not change the current element. This function is very fast (it doesn't iterate all the list but uses a cached result) and can be safely used to determine if a list is empty or not.

Parameters

List() The name of your list variable, created with the NewList function. You must include the parentheses after the list name.

Return value

The total number of elements in the list. If the list is not initialized, it returns -1 (for example after FreeList()).

Example

```
1  NewList countme.w()
2
3  ; Small loop to add some elements to the list.
4  For i=0 To 10
5      AddElement(countme())
6      countme() = i * 23
7  Next
8
9  ; Show how many elements there are in the list. I hope you thought
10 ; of the same value as this example ;)
11 MessageRequester("Information", "There are
    "+Str(ListSize(countme()))+" elements in the list",
    #PB_MessageRequester_OK)
```

See Also

ListIndex()

133.8 DeleteElement

Syntax

```
*Result = DeleteElement(List() [, Flags])
```

Description

Remove the current element from the list. After this call, the new current element is the previous element (the one before the deleted element). If that element does not exist (in other words, you deleted the first element in the list) then there is no more current element, as it will be before the first element, like after a ResetList() .

Parameters

List() List() - The name of your list variable, created with the NewList function. You must include the parentheses after the list name.

Flags (optional) If this parameter is set to 1 and the first element is deleted, the new current element will be the second one. This flag ensures there will be always a valid current element after a delete as long as there are still elements in the list.

Return value

Returns the memory address of the new current element of the list. If the list has no current element after the deletion, the result is 0.

Example

```
1  NewList people.s()
2
3  AddElement(people()) : people() = "Tom"
4  AddElement(people()) : people() = "Dick"
5  AddElement(people()) : people() = "Harry"
6  AddElement(people()) : people() = "Bob"
7
8  FirstElement(people())
9  DeleteElement(people(), 1)
10 MessageRequester("Information", "First person in list is
    "+people(), #PB_MessageRequester_Ok)
11
12 LastElement(people())      ; Moves to "Bob"
13 PreviousElement(people())  ; Moves to "Harry"
14 DeleteElement(people())    ; And deletes him. there is an element
    before Harry, so it becomes the current
15 MessageRequester("Information", "Current person in list is
    "+people(), #PB_MessageRequester_Ok)
```

See Also

AddElement() , InsertElement() , ClearList()

133.9 FirstElement

Syntax

```
*Result = FirstElement(List())
```

Description

Changes the current list element to the first list element.

Parameters

List() The name of your list variable, created with the NewList function. You must include the parentheses after the list name.

Return value

Returns the address of the data in the first list element if successful and zero if there are no elements in the list.

Example

```
1      ; An example of simple usage
2      NewList Numbers.w()
3
4      AddElement(Numbers())
5      Numbers() = 5
6      AddElement(Numbers())
7      Numbers() = 8
8
9      FirstElement(Numbers())
10     MessageRequester("Information", "First element value is
      "+Str(Numbers()), #PB_MessageRequester_OK)
11
12
13     ; An example which uses the return-value
14     NewList Numbers.w()
15
16     If FirstElement(Numbers()) <> 0
17         MessageRequester("Information", "First element value is
      "+Str(Numbers()), #PB_MessageRequester_OK)
18     Else
19         MessageRequester("Information", "List is empty",
      #PB_MessageRequester_OK)
20     EndIf
21
22     AddElement(Numbers())
23     Numbers() = 5
24     AddElement(Numbers())
25     Numbers() = 8
26
27     If FirstElement(Numbers()) <> 0
28         MessageRequester("Information", "First element value is
      "+Str(Numbers()), #PB_MessageRequester_OK)
29     Else
30         MessageRequester("Information", "List is empty",
      #PB_MessageRequester_OK)
```

```

31 EndIf
32
33
34 ; An example which is only for advanced users
35 NewList Numbers.w()
36
37 AddElement(Numbers())
38 Numbers() = 5
39 AddElement(Numbers())
40 Numbers() = 8
41
42 *Element.Word = FirstElement(Numbers())
43 If *Element
44     MessageRequester("Information", "First element value is
45     "+Str(*Element.w), #PB_MessageRequester_OK)
46 Else
47     MessageRequester("Information", "List is empty",
48     #PB_MessageRequester_OK)
49 EndIf

```

See Also

LastElement() , PreviousElement() , NextElement() , SelectElement() , ListIndex()

133.10 InsertElement

Syntax

```
*Result = InsertElement(List())
```

Description

Inserts a new empty element before the current element, or at the start of the list if the list is empty (i.e. has no elements in it). This new element becomes the current element of the list.

Parameters

List() The name of your list variable, created with the NewList function. You must include the parentheses after the list name.

Return value

Returns non-zero if the new element was created and zero otherwise. The value returned is a pointer to the new element data.

Example

```

1 ; The simplest way to use InsertElement
2 NewList simple.w()
3 InsertElement(simple()) ; Creates the first new element in the
4 list
5 simple() = 23
6 InsertElement(simple()) ; Current position is the first
7 element, so we add this element to the start of the list

```

```

7   simple() = 45 ; The old first element is now the
   second element in the list
8
9
10  ; This shows how to use the return-value of InsertElement
11  NewList advanced.l()
12  If InsertElement(advanced()) <> 0
13      advanced() = 12345
14  Else
15      MessageRequester("Error!", "Unable to allocate memory for new
   element", #PB_MessageRequester_OK)
16  EndIf
17
18
19  ; A small structure to demonstrate the use of the pointer to the
   new element
20  Structure Programmer
21      Name.s
22      Strength.b
23  EndStructure
24
25  NewList Programmers.Programmer() ; The list for storing the
   elements
26
27  *Element.Programmer = InsertElement(Programmers())
28  If *Element<>0
29      *Element\Name = "Dave"
30      *Element\Strength = 3 ; Wow, super-strong geek! ;)
31  Else
32      MessageRequester("Error!", "Unable to allocate memory for new
   element", #PB_MessageRequester_OK)
33  EndIf

```

See Also

AddElement() , DeleteElement() , ClearList()

133.11 LastElement

Syntax

```
*Result = LastElement(List())
```

Description

Change the current list element to the last list element.

Parameters

List() The name of your list variable, created with the NewList function. You must include the parentheses after the list name.

Return value

Returns the address of the data in the last list element if successful and zero if there if there are no elements in the list.

Example

```
1      ; An example of simple usage
2      NewList Numbers.w()
3
4      AddElement(Numbers())
5      Numbers() = 5
6      AddElement(Numbers())
7      Numbers() = 8
8
9      LastElement(Numbers())
10     MessageRequester("Information", "Last element value is
        "+Str(Numbers()), #PB_MessageRequester_OK)
11
12
13     ; An example which uses the return-value
14     NewList Numbers.w()
15
16     If LastElement(Numbers()) <> 0
17         MessageRequester("Information", "Last element value is
            "+Str(Numbers()), #PB_MessageRequester_OK)
18     Else
19         MessageRequester("Information", "List is empty",
            #PB_MessageRequester_OK)
20     EndIf
21
22     AddElement(Numbers())
23     Numbers() = 5
24     AddElement(Numbers())
25     Numbers() = 8
26
27     If LastElement(Numbers()) <> 0
28         MessageRequester("Information", "Last element value is
            "+Str(Numbers()), #PB_MessageRequester_OK)
29     Else
30         MessageRequester("Information", "List is empty",
            #PB_MessageRequester_OK)
31     EndIf
32
33
34     ; An example which is only for advanced users
35     NewList Numbers.w()
36
37     AddElement(Numbers())
38     Numbers() = 5
39     AddElement(Numbers())
40     Numbers() = 8
41
42     *Element.Word = LastElement(Numbers())
43     If *Element
44         MessageRequester("Information", "Last element value is
            "+Str(*Element\w), #PB_MessageRequester_OK)
45     Else
46         MessageRequester("Information", "List is empty",
            #PB_MessageRequester_OK)
47     EndIf
```

See Also

FirstElement() , PreviousElement() , NextElement() , SelectElement() , ListIndex()

133.12 ListIndex

Syntax

```
Index = ListIndex(List())
```

Description

Find out the position of the current element in the list, considering that the first element is at the position 0. This function is very fast, and can be used a lot without performance issue (it doesn't iterate the list but uses a cached value).

Parameters

List() The name of your list variable, created with the NewList function. You must include the parentheses after the list name.

Return value

A number containing the position of the current element within the list. The first element is at position 0, the next at 1 and so on. A value of -1 means there is no current element (either the list is empty or ResetList() has been used).

Example

```
1  NewList fruit.s()
2
3  AddElement(fruit()) : fruit() = "oranges"
4  AddElement(fruit()) : fruit() = "bananas"
5  AddElement(fruit()) : fruit() = "apples"
6  AddElement(fruit()) : fruit() = "pears"
7
8  FirstElement(fruit())
9  MessageRequester("Fruit: "+fruit(), "Now at position
   "+Str(ListIndex(fruit()))), #PB_MessageRequester_OK)
10
11 NextElement(fruit())
12 MessageRequester("Fruit: "+fruit(), "Now at position
   "+Str(ListIndex(fruit()))), #PB_MessageRequester_OK)
13
14 NextElement(fruit())
15 MessageRequester("Fruit: "+fruit(), "Now at position
   "+Str(ListIndex(fruit()))), #PB_MessageRequester_OK)
16
17 NextElement(fruit())
18 MessageRequester("Fruit: "+fruit(), "Now at position
   "+Str(ListIndex(fruit()))), #PB_MessageRequester_OK)
```

See Also

SelectElement() , ListSize()

133.13 NextElement

Syntax

```
*Result = NextElement(List())
```

Description

Moves from the current element to the next element in the list, or onto the first element if you have previously called ResetList()

Parameters

List() The name of your list variable, created with the NewList function. You must include the parentheses after the list name.

Return value

Returns the address of the data in the next list element if successful and zero if there is no next element.

Example

```
1  NewList Scores.w()
2
3  For i=1 To 10
4      AddElement(Scores())
5      Scores() = 100 - i
6  Next
7
8  ResetList(Scores())
9  While NextElement(Scores())
10     ; This is OK since the first call to NextElement() will move
    the current element to the first item in the list
11     MessageRequester("Score", Str(Scores()),
    #PB_MessageRequester_OK)
12 Wend
```

See Also

ResetList() , PreviousElement() , FirstElement() , LastElement() , SelectElement() , ListIndex()

133.14 PreviousElement

Syntax

```
*Result = PreviousElement(List())
```

Description

Moves from the current element to the previous element in the list.

Parameters

List() The name of your list variable, created with the NewList function. You must include the parentheses after the list name.

Return value

Returns the address of the data in the previous list element if successful and zero if there is no previous element.

Example

```
1  NewList Numbers.w()  
2  
3  For i=1 To 10  
4      AddElement(Numbers())  
5      Numbers() = i  
6  Next  
7  
8  Repeat  
9      MessageRequester("Number", Str(Numbers()),  
10     #PB_MessageRequester_OK)  
10  Until PreviousElement(Numbers()) = 0
```

See Also

NextElement() , FirstElement() , LastElement() , SelectElement() , ListIndex()

133.15 ResetList

Syntax

```
ResetList(List())
```

Description

Resets the current list element to be before the first element. This means no element is actually valid. However, this is very useful to allow you to process all the elements by using NextElement() .

Parameters

List() The name of your list variable, created with the NewList function. You must include the parentheses after the list name.

Return value

None.

Example

```
1  NewList Friends.s()
2
3  AddElement(Friends())
4  Friends() = "Arnaud"
5
6  AddElement(Friends())
7  Friends() = "Seb"
8
9  ResetList(Friends())
10 While NextElement(Friends())
11     Debug Friends() ; Display all the list elements
12 Wend
```

See Also

NextElement() , ListIndex()

133.16 SelectElement

Syntax

```
*Result = SelectElement(List(), Position)
```

Description

Change the current list element to the element at the specified position. This is very useful if you want to jump to a specific position in the list without manually iterating through the list using a loop.

Parameters

List() The name of your list variable, created with the NewList function. You must include the parentheses after the list name.

Position The position to move to in the list, considering that the first item in the list is at position 0, the next is at 1 and so on. You must make sure that you do not specify a position that is outside of the number of elements in the list!

Return value

Returns the data address of the selected element if successful or zero if the position is out of range.

Remarks

As lists don't use an index internally, this function will jump compulsory to every element in the list until the target position is reached which will take time if the list is large. If a faster method is needed, ChangeCurrentElement() should be used.

Example

```
1  NewList myList.l()
2
3  AddElement(mylist()) : myList() = 23
4  AddElement(mylist()) : myList() = 56
5  AddElement(mylist()) : myList() = 12
6  AddElement(mylist()) : myList() = 73
7
8  SelectElement(mylist(), 0)
9  MessageRequester("Position", "At position 0, the value is
   "+Str(mylist()),0)
10
11 SelectElement(mylist(), 2)
12 MessageRequester("Position", "At position 2, the value is
   "+Str(mylist()),0)
13
14 SelectElement(mylist(), 1)
15 MessageRequester("Position", "At position 1, the value is
   "+Str(mylist()),0)
16
17 SelectElement(mylist(), 3)
18 MessageRequester("Position", "At position 3, the value is
   "+Str(mylist()),0)
```

See Also

ChangeCurrentElement()

133.17 SwapElements

Syntax

```
SwapElements(List(), *FirstElement, *SecondElement)
```

Description

Swaps the position of two elements in the specified list. This command is a fast way to reorganize a list, because it does not actually move the element data itself.

Parameters

List() The name of your list variable, created with the NewList function. You must include the parentheses after the list name.

***FirstElement** Address of the first element to swap. You can get this address by using the @ operator on the list name.

***SecondElement** Address of the second element to swap. You can get this address by using the @ operator on the list name.

Return value

None.

Example

```
1  NewList Numbers()
2
3  For k=0 To 10
4      AddElement(Numbers())
5      Numbers() = k
6  Next
7
8  SelectElement(Numbers(), 3) ; Get the 4th element (first element
   is 0)
9  *FirstElement = @Numbers()
10
11 SelectElement(Numbers(), 9) ; Get the 10th element
12 *SecondElement = @Numbers()
13
14 ; Swap the 3 with the 9
15 ;
16 SwapElements(Numbers(), *FirstElement, *SecondElement)
17
18 ; Prove it
19 ;
20 ForEach Numbers()
21     Debug Numbers()
22 Next
```

See Also

MoveElement()

133.18 MoveElement

Syntax

```
MoveElement(List(), Location [, *RelativeElement])
```

Description

Moves the current element of the specified list to a different position in the list. The moved element remains the current element of the list. This is a fast operation because the element data itself is not moved to change the location in the list.

Parameters

List() The name of your list variable, created with the NewList function. You must include the parentheses after the list name.

Location Location where to move the current element. This can be one of the following values:

```
#PB_List_First : Move the element to the beginning of the list
#PB_List_Last  : Move the element to the end of the list
#PB_List_Before : Move the element before the *RelativeElement
#PB_List_After  : Move the element after the *RelativeElement
```

***RelativeElement (optional)** The address of another element in relation to which the current element should be moved. This parameter is required when the 'Location' parameter is #PB_List_Before or #PB_List_After. You can get this address from an element by using the @ operator on the list name.

Return value

None.

Example

```
1  NewList Numbers()
2
3  For k=0 To 10
4      AddElement(Numbers())
5      Numbers() = k
6  Next
7
8  SelectElement(Numbers(), 5)
9  *Relative = @Numbers()           ; get address
   of element 5
10
11 SelectElement(Numbers(), 0)
12 MoveElement(Numbers(), #PB_List_After, *Relative) ; move after
   element 5
13
14 SelectElement(Numbers(), 10)
15 MoveElement(Numbers(), #PB_List_First)           ; move to the
   beginning
16
17 ; Result
18 ;
19 ForEach Numbers()
20     Debug Numbers()
21 Next
```

See Also

SwapElements()

133.19 PushListPosition

Syntax

```
PushListPosition(List())
```

Description

Remembers the current element (if any) of the list so it can later be restored using PopListPosition(). The position is remembered on a stack (a lifo stack: last in, first out), so multiple calls to this function are possible.

Parameters

List() The name of your list variable, created with the NewList function. You must include the parentheses after the list name.

Return value

None.

Remarks

This function can be used to remember the current element, so an iteration can be made over the list using `NextElement()` or `ForEach` and the current element can be restored after the iteration using `PopListPosition()` . Multiple calls can be made to this function, as long as each is balanced with a corresponding `PopListPosition()` call later.

Note: It is not allowed to delete an element that is a remembered current element using the `DeleteElement()` or `ClearList()` function. This may result in a crash when `PopListPosition()` is called because the elements memory is no longer valid.

The stack used here is a memory area, not a list.

Example

```
1  NewList Numbers()
2  AddElement(Numbers()): Numbers() = 1
3  AddElement(Numbers()): Numbers() = 2
4  AddElement(Numbers()): Numbers() = 5
5  AddElement(Numbers()): Numbers() = 3
6  AddElement(Numbers()): Numbers() = 5
7  AddElement(Numbers()): Numbers() = 2
8
9  ; A simple duplicate elimination using a nested iteration
10 ;
11 ForEach Numbers()
12     Value = Numbers()
13     PushListPosition(Numbers())
14     While NextElement(Numbers())
15         If Numbers() = Value
16             DeleteElement(Numbers())
17         EndIf
18     Wend
19     PopListPosition(Numbers())
20 Next
21
22 ForEach Numbers()
23     Debug Numbers()
24 Next
```

See Also

`PopListPosition()` , `SelectElement()` , `ChangeCurrentElement()` , `NextElement()` ,
`PreviousElement()` , `ForEach`

133.20 PopListPosition

Syntax

```
PopListPosition(List())
```

Description

Restores the current element of the list previously remembered using `PushListPosition()` .

Parameters

List() The name of your list variable, created with the `NewList` function. You must include the parentheses after the list name.

Return value

None.

Remarks

The state of the list will be the same as it was on the corresponding call to `PushListPosition()` . If there was no current element when `PushListPosition()` was called then there is no current element after this call as well.

See the `PushListPosition()` function for an example.

See Also

`PushListPosition()` , `SelectElement()` , `ChangeCurrentElement()` , `NextElement()` , `PreviousElement()` , `ForEach`

133.21 MergeLists

Syntax

```
MergeLists(SourceList(), DestinationList() [, Location])
```

Description

Moves all elements from the `SourceList()` to the `DestinationList()`. This is a fast operation because the element data itself is not moved to merge the two lists.

Parameters

SourceList() The list from which the elements will be taken. This list will be empty after the function returns.

DestinationList() The list to move the elements to. This list will contain the items of both lists after the function returns.

Location (optional) Location where to insert the elements in the `DestinationList()`. This can be one of the following values:

```
#PB_List_Last    : Append the elements at the end of
DestinationList() (default)
#PB_List_First   : Insert the elements at the beginning of
DestinationList()
#PB_List_Before  : Insert the elements before the current
element of DestinationList()
#PB_List_After   : Insert the elements after the current
element of DestinationList()
```

Return value

None.

Example

```
1  NewList A.s()
2  AddElement(A()): A() = "a0"
3  AddElement(A()): A() = "a1"
4  AddElement(A()): A() = "a2"
5  AddElement(A()): A() = "a3"
6
7  NewList B.s()
8  AddElement(B()): B() = "b0"
9  AddElement(B()): B() = "b1"
10 AddElement(B()): B() = "b2"
11 AddElement(B()): B() = "b3"
12
13 ; Insert the elements of A() before the "b1" element in B()
14 SelectElement(B(), 1)
15 MergeLists(A(), B(), #PB_List_Before)
16
17 ForEach B()
18     Debug B()
19 Next
```

See Also

SplitList()

133.22 SplitList

Syntax

```
SplitList(SourceList(), DestinationList() [, KeepCurrent])
```

Description

Moves the elements in SourceList() from the current element onwards to the DestinationList(). This is a fast operation because the element data itself is not moved to split the list.

Parameters

SourceList() The list from which the elements will be split. The current element of this list specifies the point at which to split the list. If there is no current element, then all elements remain in SourceList().

DestinationList() The list to move the elements to. Any existing elements in this list are deleted before the new elements are added.

KeepCurrent (optional) Whether the current item in SourceList() remains in SourceList() or is moved to DestinationList(). If this parameter is **#True**, then the current element remains in SourceList(). If it is **#False** (default), then the current element is moved to DestinationList().

Return value

None.

Remarks

If 'KeepCurrent' is set to `#True` then the new current element in `SourceList()` will be the previous element of the list. If there is no previous element then the list will no longer have a current element after this function returns. The `DestinationList()` will have no current element.

Example

```
1  NewList A()
2  NewList B()
3
4  For i = 0 To 10
5      AddElement(A())
6      A() = i
7  Next i
8
9  ; split A() at element 5 and move the remaining elements to B()
10 SelectElement(A(), 5)
11 SplitList(A(), B())
12
13
14 Debug " -- A() -- "
15 ForEach A()
16     Debug A()
17 Next
18
19 Debug " -- B() -- "
20 ForEach B()
21     Debug B()
22 Next
```

See Also

`MergeLists()`

Chapter 134

Mail

Overview

Mail is now a common way for information to be exchanged between two distant computers. The functions within this library allow for not only the creation of mail, but also the ability send this mail to either a single recipient or to multiple recipients. In addition, optional attachments may be made when desired. This may then be sent through a mail server.

Note: on Linux, 'libcurl' needs to be installed to have the mail commands working (most of Linux distributions comes with it already installed).

134.1 AddMailAttachment

Syntax

```
Result = AddMailAttachment(#Mail, Description$, Filename$ [,
    MimeType$])
```

Description

Add a file attachment to the mail.

Parameters

#Mail The mail to use.

Description\$ The information string displayed for the attachment in the mail.

Filename\$ The file to be added as an attachment. If the filename does not include a full path, it is interpreted relative to the current directory . Once the attachment is added, the local file may be changed or deleted, as the entire content of the file is copied and attached to the mail.

MimeType\$ (optional) The type of the attached file. If this parameter is omitted, then the file extension will be used to determine that file's mime type. Below there is a list of the available mime types. If the extension of the file does not match any of the available mime types, then the "application/octet-stream" mime type will be used.

Return value

Returns nonzero if the attachment was added successfully and zero if not.

Remarks

Any number of attachments may be added to the mail, but the size limit available for a single attachment is currently set at 100MB. Most servers and clients do not have the capacity to handle attachments of that size, therefore it is advised that each attachment be kept to a reasonable size. The available mime types are:

application/acad	AutoCAD	dwg
application/clariscad	ClarisCAD	ccad
application/drafting	MATRA Prelude	drafting drw
application/dxf	AutoCAD	dxf
application/i-deas	SDRC I-deas	unv
application/iges	Exchange format	CAO IGES igs,iges
application/oda	ODA	oda
application/pdf	Adobe Acrobat	pdf
application/postscript	PostScript	ai,eps,ps
application/pro_eng	ProEngineer	prt
application/rtf	Rich text	rtf
application/set	CAO SET	set
application/sla	stereolithography	stl
application/solids	MATRA Solids	dwg
application/step	data STEP	step
application/vda	surface	vda
application/x-mif	Framemaker	mif
application/x-csh	Script C-Shell (UNIX)	dwg
application/x-dvi	text dvi	dvi
application/hdf	data	hdf
application/x-latex	LaTeX	latex
application/x-netcdf	netCDF	nc,cdf
application/x-sh	Script Bourne Shell	dwg
application/x-tcl	Script Tcl	tcl
application/x-tex	file Tex	tex
application/x-texinfo	eMacs texinfo,	texi
application/x-troff	Troff	t,tr,troff
application/x-troff-man	Troff/macro	man man
application/x-troff-me	Troff/macro	ME me
application/x-troff-ms	Troff/macro	MS ms
application/x-wais-source	Source Wais	src
application/x-bcpio	CPIO binary	bcpio
application/x-cpio	CPIO Posix	cpio
application/x-gtar	Tar GNU	gtar
application/x-shar	Archives Shell	shar
application/x-sv4cpio	CPIO SVR4n	sv4cpio
application/x-sv4crc	CPIO SVR4 avec CRC	sc4crc
application/x-tar	archive tar	tar
application/x-ustar	archive tar Posix	man
application/zip	archive ZIP	man
audio/basic	audio au,snd	
audio/x-aiff	audio AIFF	aif,aiff,aifc
audio/x-wav	audio Wave	wav
image/gif	Images gif	man
image/ief	Images exchange format	ief
image/jpeg	Images Jpeg	jpg,jpeg,jpe
image/png	Images Png	png
image/tiff	Images Tiff	tiff,tif
image/x-cmu-raster	Raster cmu	cmu
image/x-portable-anymap	Anymap PBM	pnm
image/x-portable-bitmap	Bitmap PBM	pbm
image/x-portable-graymap	Graymap PBM	pgm
image/x-portable-pixmap	Pixmap PBM	ppm

image/x-rgb	Image RGB rgb
image/x-xbitmap	Images Bitmap X xbm
image/x-xpixmap	Images Pixmap X xpm
image/x-xwindowdump	Images dump X Window man
multipart/x-zip	archive zip zip
multipart/x-gzip	archive GNU zip gz,gzip
text/html	HTML htm,html
text/plain	raw text txt,g,h,c,cc,hh,m,f90
text/richtext	rich text rtx
text/tab-separated-value	value split text tsv
text/x-setext	text Struct etx
video/mpeg	Video MPEG mpeg,mpg,mpe
video/quicktime	Video QuickTime qt,mov
video/msvideo	Video Microsoft Windows avi
video/x-sgi-movie	Video MoviePlayer movie

Example

```

1  If CreateMail(0, "test@purebasic.com", "Hello")
2      If AddMailAttachment(0, "Attachment 1",
3          OpenFileRequester("Choose an attachment", "", "", 0))
4          Debug "Attachment correctly added"
5      Else
6          Debug "Attachment failed"
7      EndIf
    EndIf

```

See Also

AddMailAttachmentData() , CreateMail()

134.2 AddMailAttachmentData

Syntax

```
Result = AddMailAttachmentData(#Mail, Description$, *Buffer,
    BufferLength [, MimeType$])
```

Description

Add memory data as an attachment to the mail.

Parameters

#Mail The mail to use.

Description\$ The information string displayed for the attachment in the mail.

***Buffer** The memory area of the data that should be added as the attachment. Once the attachment has been added, the memory area may be changed or freed, since its entire contents are copied into the mail.

BufferLength The size of the attachment in bytes.

MimeType\$ (optional) The type of the attached file. If this parameter is omitted, then the file extension will be used to determine that files mime type. See the AddMailAttachment() command for a list of available mime types. If the extension of the file does not match any of the available mime types, then the "application/octet-stream" mime type will be used.

Return value

Returns nonzero if the attachment was added successfully and zero if not.

Remarks

Any number of attachments may be added to the mail, but the size limit available for a single attachment is currently set at 100MB. Most servers and clients do not have the capacity to handle attachments of that size, therefore it is advised that each attachment be kept to a reasonable size

Example

```
1  If CreateMail(0, "test@purebasic.com", "Hello")
2
3      If AddMailAttachmentData(0, "Attachment 1", ?Hello, 5)
4          Debug "Attachment correctly added"
5      Else
6          Debug "Attachment failed"
7      EndIf
8  EndIf
9
10
11  DataSection
12      Hello:
13          Data.b 'H', 'e', 'l', 'l', 'o'
```

See Also

AddMailAttachment() , CreateMail()

134.3 AddMailRecipient

Syntax

```
AddMailRecipient(#Mail, Address$, Flags)
```

Description

Add a recipient to the specified mail.

Parameters

#Mail The mail to use.

Address\$ The address of the recipient. The address have to be one of the following format:

```
"joe.doe@domain.com"
"<joe.doe@domain.com>"
"Joe Doe <joe.doe@domain.com>"
```

Flags The categories to which to add the recipient. It can be a combination of the following values:

```
#PB_Mail_To : Main(s) recipient(s) of the mail
#PB_Mail_Cc : Recipient(s) which are in copy of the mail (and
everybody see it)
#PB_Mail_Bcc: Recipient(s) which are in copy of the mail (but
nobody see it)
```

Return value

None.

Example

```
1  If CreateMail(0, "test@purebasic.com", "Hello")
2      AddMailRecipient(0, "andre@purebasic.com", #PB_Mail_To) ; Andre
    is the main recipient
3      AddMailRecipient(0, "fred@purebasic.com", #PB_Mail_Cc) ; Fred
    is in copy
4      AddMailRecipient(0, "timo@purebasic.com", #PB_Mail_Bcc) ; Timo
    is in copy as well, but Andre and Fred doesn't know it
5  EndIf
```

See Also

RemoveMailRecipient() , CreateMail()

134.4 CreateMail

Syntax

Result = CreateMail(#Mail, From\$, Subject\$ [, Encoding])

Description

Create a new, empty mail.

Parameters

#Mail A number to identify the new mail. #PB_Any can be used to auto-generate this number.

From\$ The sender address for the mail. The address have to be one of the following format:

```
"joe.doe@domain.com"
"<joe.doe@domain.com>"
"Joe Doe <joe.doe@domain.com>"
```

Subject\$ The subject line for the mail.

Encoding (optional) The encoding for the mail. It can be one of the following values:

```
#PB_Ascii : The mail body will be in ascii
#PB_UTF8  : The mail body will be in UTF-8
(default)
```

Return value

Returns nonzero if the mail was created successfully and zero if not. If #PB_Any was used as the #Mail parameter, then the auto-generated number is returned in case of success.

Remarks

SetMailBody() , SetMailAttribute() , AddMailAttachment() and AddMailAttachmentData() can be used to change the content of the #Mail.

Note: According to the [RFC 2822](#) standard a line break in an e-mail need to be done always using the CRLF chars.

On Linux, 'libcurl' needs to be installed to have the mail commands working (most of Linux distributions comes with it already installed).

Example

```
1  If CreateMail(0, "test@purebasic.com", "Hello")
2      SetMailBody(0, "This is a body !" + #CRLF$ + "Second line")
3      Debug "Mail created"
4  Else
5      Debug "Can't create the mail"
6  EndIf
```

See Also

SetMailBody() , SetMailAttribute() , AddMailAttachment() , AddMailAttachmentData() , SendMail() , FreeMail()

134.5 FreeMail

Syntax

```
FreeMail(#Mail)
```

Description

Free the specified mail and release its associated memory.

Parameters

#Mail The mail to free. If **#PB_All** is specified, all the remaining mails are freed.

Return value

None.

Remarks

If there is any mail remaining, they are all automatically freed when the program ends.

See Also

CreateMail()

134.6 GetMailAttribute

Syntax

Result\\$\$ = GetMailAttribute(#Mail, Attribute)

Description

Return the specified mail attribute.

Parameters

#Mail The mail to use.

Attribute The attribute to get. It may be one of the following values:

```
#PB_Mail_From      : Get the 'From' attribute, set with  
                    CreateMail()  
.  
#PB_Mail_Subject: Get the 'Subject' attribute, set with  
                    CreateMail()  
.  
#PB_Mail_XMailer: Get the 'X-Mailer' attribute (if any)  
#PB_Mail_Date     : Get the 'Date' attribute (if any)  
#PB_Mail_Custom  : Get customs fields (if any)
```

Return value

Returns the attribute as a string. An empty string is returned if the attribute does not exist.

Remarks

SetMailAttribute() may be used to change the #Mail attributes.

Example

```
1  If CreateMail(0, "test@purebasic.com", "Hello")  
2      Debug GetMailAttribute(0, #PB_Mail_From) ; Will print  
    "test@purebasic.com"  
3      Debug GetMailAttribute(0, #PB_Mail_Subject) ; Will print "Hello"  
4  EndIf
```

See Also

SetMailAttribute() , CreateMail()

134.7 GetMailBody

Syntax

Result\\$\$ = GetMailBody(#Mail)

Description

Return the specified mail body, previously set with SetMailBody() .

Parameters

#Mail The mail to use.

Return value

Returns the body as a string.

Example

```
1  If CreateMail(0, "test@purebasic.com", "Hello")
2      SetMailBody(0, "This is the body")
3      Debug GetMailBody(0) ; Will print "This is the body"
4  EndIf
```

See Also

SetMailBody() , CreateMail()

134.8 IsMail

Syntax

```
Result = IsMail(#Mail)
```

Description

Tests if the given mail number is valid and if the mail has been correctly initialized.

Parameters

#Mail The number to test.

Return value

Returns nonzero if the specified number was valid.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure that the mail is ready for use.

See Also

CreateMail() , FreeMail()

134.9 MailProgress

Syntax

```
Result = MailProgress(#Mail)
```

Description

Return the progress of the specified mail transfer, started with SendMail() .

Parameters

#Mail The mail to use.

Return value

Returns the number of transferred bytes, or one of the following values:

```
#PB_Mail_Connected: The mail transfer is in its initialization
phase.
#PB_Mail_Finished : The mail transfer is finished correctly.
#PB_Mail_Error    : The mail transfer is finished but an error
occurred.
```

See Also

SendMail()

134.10 RemoveMailRecipient

Syntax

```
RemoveMailRecipient(#Mail [, Address$ [, Flags]])
```

Description

Remove a recipient from the specified mail.

Parameters

#Mail The mail to use.

Address\$ (optional) The address to remove. It must match an address from a call to AddMailRecipient() . If not specified, all the recipients are removed from the mail.

Flags (optional) The categories from which to remove the recipient. It can be a combination of the following values:

```
#PB_Mail_To : Main(s) recipient(s) of the mail
#PB_Mail_Cc : Recipient(s) which are in copy of the mail (and
everybody see it)
#PB_Mail_Bcc: Recipient(s) which are in copy of the mail (but
nobody see it)
```

If not specified, all the categories are removed for the specified address.

Return value

None.

Example

```
1  If CreateMail(0, "test@purebasic.com", "Hello")
2      AddMailRecipient(0, "andre@purebasic.com", #PB_Mail_To) ; Andre
    is the main recipient
3      AddMailRecipient(0, "fred@purebasic.com", #PB_Mail_Cc) ; Fred
    is in copy
4      AddMailRecipient(0, "timo@purebasic.com", #PB_Mail_Bcc) ; Timo
    is in copy as well, but Andre and Fred are not aware of it
5
6      ; Ensure Fred is removed from every destination :-)
7      RemoveMailRecipient(0, "fred@purebasic.com")
8  EndIf
```

See Also

AddMailRecipient()

134.11 SendMail

Syntax

```
Result = SendMail(#Mail, SmtP$ [, Port [, Flags [, User$,
    Password$]]])
```

Description

Send the specified mail.

Parameters

#Mail The mail to send.

SmtP\$ The address of the mail server to use for sending the mail.

Port (optional) The port of the mail server. The default is port 25.

Flags (optional) It can be a combination of the following values:

```
#PB_Mail_Asynchronous: sends the mail in the background.
MailProgress()
can be used to follow the progress.
#PB_Mail_UseSSL       : uses TLS/SSL to send the mail (the
server needs to support this protocol).
#PB_Mail_UseSMTPS     : uses SMTPS to send the mail (the
server needs to support this protocol).
#PB_Mail_Debug        : to print in the console debug
information.
```

User\$, Password\$ (optional) The user and password used for SMTP authentication, if the server requires it.

Return value

Returns nonzero if the mail was sent correctly and zero otherwise.

Example: Simple SMTP

```
1 ; Note: change the address and smtp to have a working example
2 ;
3 If CreateMail(0, "test@youraddress.com", "Hello")
4     AddMailRecipient(0, "youraddress@youraddress.com", #PB_Mail_To)
5
6     Debug SendMail(0, "smtp.yourfavoritesmtp.com")
7 EndIf
```

Example: Using Hotmail (TLS)

```
1 Login$ = "you@hotmail.com"
2 Password$ = "password"
3 Receiver$ = "test@gmail.com"
4
5 If CreateMail(0, Login$, "Hello")
6     AddMailRecipient(0, Receiver$, #PB_Mail_To)
7     Debug SendMail(0, "smtp-mail.outlook.com", 587,
8         #PB_Mail_UseSSL, Login$, Password$)
9 EndIf
```

Example: Using GMail (TLS+SMTPS)

```
1 ; Be sure to use the right login and right password
2 ;
3 Login$ = "yourlogin"
4 Password$ = "password"
5 Receiver$ = "test@gmail.com"
6
7 If CreateMail(0, Login$ + "@gmail.com", "Hello")
8     AddMailRecipient(0, Receiver$, #PB_Mail_To)
9     Debug SendMail(0, "smtp.gmail.com", 465, #PB_Mail_UseSSL |
10         #PB_Mail_UseSMTPS, Login$, Password$)
11 EndIf
```

See Also

CreateMail() , MailProgress()

134.12 SetMailAttribute

Syntax

```
SetMailAttribute(#Mail, Attribute, Value$)
```

Description

Change the specified mail attribute with the new value.

Parameters

#Mail The mail to use.

Attribute The attribute to change. It may be one of the following values:

```
#PB_Mail_From      : Change the 'From' attribute, set with  
CreateMail()  
.  
#PB_Mail_Subject: Change the 'Subject' attribute, set with  
CreateMail()  
.  
#PB_Mail_XMailer: Change the 'X-Mailer' attribute (not sent  
by default).  
#PB_Mail_Date      : Change the 'Date' attribute (default is the  
computer date).  
#PB_Mail_Custom : Add customs fields (can be multi-line).
```

Value\$ The new value for the attribute.

Return value

None.

Example

```
1  If CreateMail(0, "test@purebasic.com", "Hello")  
2      SetMailAttribute(0, #PB_Mail_XMailer, "PureMailer")  
3      Debug GetMailAttribute(0, #PB_Mail_XMailer) ; Will print  
        "PureMailer"  
4  EndIf
```

See Also

GetMailAttribute() , CreateMail()

134.13 SetMailBody

Syntax

```
SetMailBody(#Mail, Body$)
```

Description

Change the mail body. GetMailBody() can be used to read the body content.

Parameters

#Mail The mail to use.

Body\$ The string for the new mail body.

Return value

None.

Remarks

According to the [RFC 2822](#) standard a line break in an e-mail need to be done always using the `#CRLF$` chars.

Example

```
1  If CreateMail(0, "test@purebasic.com", "Hello")
2      SetMailBody(0, "This is the body")
3      Debug GetMailBody(0) ; Will print "This is the body"
4  EndIf
```

See Also

GetMailAttribute() , CreateMail()

Chapter 135

Map

Overview

Maps (also known as hashtable or dictionary) are structures for storing data which are dynamically allocated depending of your need. It is a collection of elements (the data you want to store) and each element is fully independent of the others. You can add as many elements as you want (or as many as will fit into the memory of your computer), and accessing it back using a key. This kind of data management is very useful when you need fast access to a random element. The inserting order of the elements is not kept when using a map (unlike a List) and therefore they can't be sorted.

Before you can work with Maps, you must declare them first. This could be done with the keyword `NewMap` . structures are also often used to store multiple data in a single element.

To specifically search the contents of a Map, the use of loops is recommended: `For : Next` , `ForEach : Next` , `Repeat : Until` or `While : Wend` .

Other possibilities for storing data are the use of Arrays and Lists .

When using threads , the same map can not be accessed or modified at the same time in different threads. Mutexes can to be used to ensure the map is only used by one thread.

135.1 AddMapElement

Syntax

```
Result = AddMapElement(Map(), Key$ [, Flags])
```

Description

Adds a new empty element in the `Map()` using the specified key. This new element becomes the current element of the map.

Parameters

Map() The map to which to add the element.

Key\$ The key for the new element.

Flags (optional) Flags can be one of the following values:

```
#PB_Map_ElementCheck : Checks if an element with a same key  
already exists, and replaces it (default).  
#PB_Map_NoElementCheck: No element check, so if a previous  
element with the same key was already present, it  
will be not replaced but kept in the map,  
unreachable with direct access. It will remain unreachable
```


`until` the newly added element has been deleted. Such unreachable elements will still be listed when enumerating all the map elements with `ForEach` or `NextMapElement()`. This mode is faster but also more error prone, so use it `with` caution.

Return value

Returns nonzero on success and zero on failure. The value returned is a pointer to the new element data.

Remarks

This function isn't mandatory when dealing with maps, as elements are automatically added when affecting a value to them.

Example

```
1  NewMap Country.s()
2
3  ; Regular way to add an element
4  Country("US") = "United States"
5
6  ; The same using AddMapElement()
7  AddMapElement(Country(), "FR")
8  Country() = "France"
9
10 ForEach Country()
11     Debug Country()
12 Next
```

See Also

`DeleteMapElement()` , `ClearMap()` , `MapSize()`

135.2 ClearMap

Syntax

```
ClearMap(Map())
```

Description

Clears all the elements in the specified map and releases their memory. After this call the map is still usable, but is empty (i.e. there are no more elements in it).

Parameters

Map() The map to clear.

Return value

None.

Remarks

PureBasic will only free the memory for the elements. If you have been using the map for something such as storing handles of objects that you create directly with the OS, there is no way PureBasic (or any other language) can know what they are. Therefore, in cases such as that, you should go through the elements in the map and free the objects yourself.

Example

```
1  NewMap Country.s()
2
3  Country("FR") = "France"
4  Country("US") = "United States"
5
6  ; Proof that items have been added to the map
7  MessageRequester("Information", "There are " +
8    Str(MapSize(Country())) + " elements in the map")
9
10 ; Clear the map and show that the map really is empty
11 ClearMap(Country())
12 MessageRequester("Information", "There are " +
13   Str(MapSize(Country())) + " element in the map")
```

See Also

AddMapElement() , DeleteMapElement()

135.3 CompareMap

Syntax

```
Result = CompareMap(Map1(), Map2() [, Flags])
```

Description

Compare each elements of the two maps for equality. Recursively compares also contents of structured maps with dynamic elements (such as embedded arrays, lists or maps). The two maps are considered the equal if they have the same type and size and if each pair of elements is equal.

Parameters

Map1(), **Map2()** The maps to compare.

Flags (optional) Can be a combination of the following values:

```
#PB_String_CaseSensitive : String comparison is case
                           sensitive (a=a). (default)
#PB_String_NoCase       : String comparison is case
                           insensitive(a=A).
#PB_Memory_FollowPointers: If a structure element is a
                           pointer that is not 0, recursively compare the pointer
                           target.
```

The `default` is to compare only the pointer value itself. See remarks below for details.

Return value

Returns nonzero if both maps are the same or zero if they differ.

Remarks

The `#PB_Memory_FollowPointers` option is for advanced users and requires special care to avoid crashes. If this option is used then all pointer values must point to valid and initialized memory or have the value 0. It is also not allowed to have loops in the pointed elements (a chain of pointers that refers back to itself).

Example

```
1  NewMap A()  
2  A("de") = 49  
3  A("fr") = 33  
4  A("uk") = 44  
5  
6  NewMap B() ; one element missing  
7  B("de") = 49  
8  B("uk") = 44  
9  
10 Debug CompareMap(A(), B()) ; Not equal
```

See Also

`CompareArray()` , `CompareList()`

135.4 CopyMap

Syntax

```
Result = CopyMap(SourceMap(), DestinationMap())
```

Description

Copy every element of the source to the destination map.

Parameters

SourceMap() The map to copy from.

DestinationMap() The map to copy to. The existing elements in this map will be freed. After a successful copy, the two maps are identical.

Return value

Returns nonzero on success and zero on failure. If the two maps do not have the same type then the copy will fail.

Example

```
1  NewMap Age()
2  NewMap AgeCopy()
3
4  Age("John") = 15
5  Age("Elise") = 30
6
7  CopyMap(Age(), AgeCopy())
8
9  Debug AgeCopy("John")
10 Debug AgeCopy("Elise")
```

See Also

CopyArray() , CopyList()

135.5 FreeMap

Syntax

```
FreeMap(Map())
```

Description

Free the specified map and release all its associated memory. To access it again NewMap has to be called.

Parameters

Map() The map to free.

Return value

None.

See Also

ClearMap()

135.6 MapSize

Syntax

```
Result = MapSize(Map())
```

Description

Returns the number of elements in the specified map. It does not change the current element.

Parameters

Map() The map to use.

Return value

Returns the number of elements in the map.

Remarks

This function is very fast (it doesn't iterate all the map but uses a cached result) and can be safely used to determine if a map is empty or not. If the map is not initialized, it returns -1 (for example after `FreeMap()`).

Example

```
1  NewMap Country.s()
2
3  Country("FR") = "France"
4  Country("US") = "United States"
5
6  ; Will print '2'
7  Debug "Size of the map: " + Str(MapSize(Country()))
```

135.7 DeleteMapElement

Syntax

`Result = DeleteMapElement(Map() [, Key$])`

Description

Removes the current element or the element with the given key from the specified map.

Parameters

Map() The map to use.

Key\$ (optional) The key for the item to remove. If this is not specified, then the current element of the map is removed.

Return value

Returns the memory address of the new current element of the map. If the map has no current element after the deletion, the result is 0.

Remarks

After this call, the new current element is the previous element (the one before the deleted element), which is an arbitrary element, as a map isn't sorted. If that element does not exist (in other words, you deleted the first element in the map) then there is no more current element, as it will be before the first element, like after a `ResetMap()` . If there was only one element in the map when you deleted it, then you are left with no current element!

If the optional 'Key\$' parameter is specified then there will be no more current element after this call. So don't use this parameter if the command is used inside a `ForEach : Next` loop!

Example

```
1  NewMap Country.s()
2
3  Country("US") = "United States"
4  Country("FR") = "France"
5  Country("GE") = "Germany"
6
7  ; Delete a country
8  DeleteMapElement(Country(), "FR")
9
10 ForEach Country()
11     Debug Country()
12 Next
```

See Also

AddMapElement() , ClearMap() , MapSize()

135.8 FindMapElement

Syntax

```
Result = FindMapElement(Map(), Key$)
```

Description

Change the current map element to the element associated at the specified key.

Parameters

Map() The map to use.

Key\$ The key to find.

Return value

Returns nonzero if the key was found and zero otherwise. The value returned is a pointer to the element data.

Example

```
1  NewMap Country.s()
2
3  Country("US") = "United States"
4  Country("FR") = "France"
5  Country("GE") = "Germany"
6
7  If FindMapElement(Country(), "US")
8      Debug "'US' is in the country list."
9  Else
10     Debug "'US' is NOT in the country list !"
11 EndIf
12
13 If FindMapElement(Country(), "UK")
```

```

14     Debug "'UK' is in the country list."
15 Else
16     Debug "'UK' is NOT in the country list !"
17 EndIf

```

See Also

AddMapElement() , DeleteMapElement() , MapKey()

135.9 MapKey

Syntax

Key\\$ = MapKey(Map())

Description

Returns the key of the current map element.

Parameters

Map() The map to use.

Return value

Returns the key of the current element. If there is no current element, an empty string is returned.

Example

```

1  NewMap Country.s()
2
3  Country("US") = "United States"
4  Country("FR") = "France"
5  Country("GE") = "Germany"
6
7  ForEach Country()
8      Debug MapKey(Country())
9  Next

```

See Also

ResetMap() , NextMapElement()

135.10 NextMapElement

Syntax

Result = NextMapElement(Map())

Description

Moves from the current element to the next element in the specified map, or onto the first element if ResetMap() was previously called.

Parameters

Map() The map to use.

Return value

Returns nonzero if the next element was set and zero if there is no next element. The value returned is a pointer to the new element data.

Example

```
1  NewMap Country.s()  
2  
3  Country("US") = "United States"  
4  Country("FR") = "France"  
5  Country("GE") = "Germany"  
6  
7  ResetMap(Country())  
8  While NextMapElement(Country())  
9      Debug Country()  
10 Wend
```

See Also

ResetMap() , MapKey()

135.11 ResetMap

Syntax

```
ResetMap(Map())
```

Description

Resets the current element of the specified map to be before the first element. This means no more current element. However, this is very useful to process all the elements by using NextMapElement() .

Parameters

Map() The map to use.

Return value

None.

Example

```
1  NewMap Country.s()  
2  
3  Country("US") = "United States"  
4  Country("FR") = "France"  
5  Country("GE") = "Germany"  
6
```



```

7   ResetMap(Country())
8   While NextMapElement(Country())
9       Debug Country()
10  Wend

```

See Also

NextMapElement()

135.12 PushMapPosition

Syntax

```
PushMapPosition(Map())
```

Description

Remembers the current element (if any) of the map so it can later be restored using PopMapPosition(). The position is remembered on a stack structure, so multiple calls to this function are possible.

Parameters

Map() The map to use.

Return value

None.

Remarks

This function can be used to remember the current element, so an iteration can be made over the map using NextMapElement() or ForEach and the current element can be restored after the iteration using PopMapPosition(). Multiple calls can be made to this function, as long as each is balanced with a corresponding PopMapPosition() call later.

Note: It is not allowed to delete an element that is a remembered current element using the DeleteMapElement() or ClearMap() function. This may result in a crash when PopMapPosition() is called because the elements memory is no longer valid.

Example

```

1   NewMap Numbers()
2   Numbers("A") = 1
3   Numbers("B") = 2
4   Numbers("C") = 5
5   Numbers("D") = 3
6   Numbers("E") = 2
7   Numbers("F") = 5
8
9   ; A simple duplicate elimination using a nested iteration
10  ;
11  ForEach Numbers()
12      Value = Numbers()
13      PushMapPosition(Numbers())

```

```

14     While NextMapElement(Numbers())
15         If Numbers() = Value
16             DeleteMapElement(Numbers())
17         EndIf
18     Wend
19     PopMapPosition(Numbers())
20 Next
21
22 ForEach Numbers()
23     Debug Numbers()
24 Next

```

See Also

PopMapPosition() , FindMapElement() , NextMapElement() , ResetMap() , ForEach

135.13 PopMapPosition

Syntax

```
PopMapPosition (Map ())
```

Description

Restores the current element of the map previously remembered using PushMapPosition() .

Parameters

Map() The map to use.

Return value

None.

Remarks

The state of the map will be the same as it was on the corresponding call to PushMapPosition() . If there was no current element when PushMapPosition() was called then there is no current element after this call as well.

See the PushMapPosition() function for an example.

See Also

PushMapPosition() , FindMapElement() , NextMapElement() , ResetMap() , ForEach

Chapter 136

Material

Overview

Materials are composed of one or several textures and sometimes of some colors. They are widely used by the other objects of the 3D world like the entities , billboards and particles to give them a skin.

Each material has a lot a properties like the shading, ambient and specular color, etc. to allow realistic looking materials like wood, water, glass and more.

InitEngine3D() must be called successfully before using the Material functions.

136.1 AddMaterialLayer

Syntax

```
AddMaterialLayer(#Material , TextureID [, Mode [,
TextureCoordinateIndex]])
```

Description

Adds a new layer to the material and put the specified texture in it.

Parameters

#Material The material to use.

TextureID The texture to add. A valid 'TextureID' can be easily obtained with TextureID() .

Mode (optional) It can be one of the following values:

```
#PB_Material_Add          : Performs a pixel 'Add' operation
                           over previous layer (black color is like transparent)
#PB_Material_AddSigned    : Performs a pixel signed 'Add'
                           operation over previous layer (black color is like
                           transparent)
#PB_Material_Subtract     : Performs a pixel 'Subtract'
                           operation over previous layer (black color is like
                           transparent)
#PB_Material_Replace      : Performs a pixel 'Replace'
                           operation over previous layer
#PB_Material_AlphaBlend   : Use the AlphaChannel layer of the
                           texture (should be a TGA or PNG one) and blend it with the
                           previous layer
#PB_Material_Modulate     : Performs a pixel 'Multiply'
                           operation over previous layer
```

```
#PB_Material_ModulateX2 : Performs a pixel 'Multiply'
operation over previous layer and brighten afterwards (x2)
#PB_Material_ModulateX4 : Performs a pixel 'Multiply'
operation over previous layer and brighten afterwards (x4)
#PB_Material_BlendDiffuseAlpha: Use the AlphaChannel layer of
the texture and blend it with the previous layer with diffuse
#PB_Material_BlendCurrentAlpha: Use the AlphaChannel layer of
the current stage and blend it with the previous layer
```

TextureCoordinateIndex (optional) Texture coordinate index (default value is 0).

Return value

None.

See Also

RemoveMaterialLayer()

136.2 CopyMaterial

Syntax

```
Result = CopyMaterial(#Material, #NewMaterial)
```

Description

Creates a new material which is the exact copy of the specified material.

Parameters

#Material The material to copy.

#NewMaterial A number to identify the new material. #PB_Any can be used to auto-generate this number.

Return value

Nonzero if the material was successfully duplicated, zero otherwise. If #PB_Any was used for the #NewMaterial parameter, then the generated number is returned on success.

136.3 CountMaterialLayers

Syntax

```
Result = CountMaterialLayers(#Material)
```

Description

Returns the number of layers the material contains.

Parameters

#Material The material to use.

Return value

The number of layers the material contains.

136.4 CreateMaterial

Syntax

```
Result = CreateMaterial(#Material, TextureID [, Color])
```

Description

Creates a new material using the specified texture.

Parameters

#Material A number to identify the new material. **#PB_Any** can be used to auto-generate this number.

TextureID The texture to use or **#Null** if no texture is needed. A valid 'TextureID' can be easily obtained with TextureID() .

Color (optional) The color to use for ambient and diffuse. A valid RGB color can be created with RGB() .

Return value

Nonzero if the material was successfully created, zero otherwise. If **#PB_Any** was used for the **#Material** parameter, then the generated number is returned on success.

136.5 CreateAnimatedMaterial

Syntax

```
Result = CreateAnimatedMaterial(#Material, TextureArray(),  
    Duration.f)
```

Description

Creates a new animated material using the specified textures.

Parameters

#Material A number to identify the new animated material. **#PB_Any** can be used to auto-generate this number.

TextureArray() An array of TextureID, with all the textures to use for the animation. TextureID() can be used to get a valid texture identifier.

Duration The animation duration in seconds. It can be a fractional number, for example: 1.5 which will be one second and half. Once the animation is finished, it will automatically start again.

Return value

Nonzero if the animated material was successfully created, zero otherwise. If **#PB_Any** was used for the **#Material** parameter, then the generated number is returned on success.

Example

```
1 CreateTexture(0, 256, 256)
2 CreateTexture(1, 256, 256)
3 CreateTexture(2, 256, 256)
4
5 Dim Textures(2)
6 Textures(0) = TextureID(0)
7 Textures(1) = TextureID(1)
8 Textures(2) = TextureID(2)
9
10 ; Creates an animated material with 3 textures, which will play
    for 500ms
11 ;
12 CreateAnimatedMaterial(0, Textures(), 0.5)
```

136.6 CreateShader

Syntax

Result = CreateShader(ShaderID, VertexProgram\$, FragmentProgram\$)

Description

Creates a new shader using the specified vertex and fragment programs.

Parameters

ShaderID A number between 0 and 65536 to identify the new shader. To select one of the predefined shader, use one of the following values as 'ShaderID' (VertexProgram\$ and FragmentProgram\$ values will be ignored):

```
#PB_Material_ColorShader
#PB_Material_PerpixelShader
#PB_Material_BumpShader
#PB_Material_SkyShader
#PB_Material_WaterShader
#PB_Material_WaterShaderRTT
#PB_Material_OceanShader
#PB_Material_PointSpriteSphereShader
```

VertexProgram\$ The GLSL program to use for vertices.

FragmentProgram\$ The GLSL program to use for fragments.

Return value

Nonzero if the shader was successfully created, zero otherwise.

Remarks

Shaders usually need to receive parameters to work and two functions allow to pass these parameters: MaterialShaderAutoParameter() (automatic parameter) and MaterialShaderParameter() (shader parameter).

PureBasic also allows to set these parameters directly in the shader programs by adding at the end of the 'uniform' declaration line: "//" followed by either:

- the auto parameter number preceded by a "+" (cf. 'ParameterType' constants of the MaterialShaderAutoParameter() function).
- one or more values (separated by a space) corresponding to the shader parameter (MaterialShaderParameter()).

Example:

```
uniform vec4 camera_pos;//+76 <- automatic parameter #PB_Shader_CameraPosition
uniform vec4 fog_params;//+31 <- automatic parameter #PB_Shader_FogParams
uniform vec3 speed;//0.5 1 0 <- shader parameter 'speed'
uniform sampler2D diffuseMap;//1 <- shader parameter texture number
```

See Also

CreateShaderMaterial() , MaterialShaderAutoParameter() , MaterialShaderParameter()

136.7 CreateShaderMaterial

Syntax

```
Result = CreateShaderMaterial(#Material , ShaderID)
```

Description

Creates a new shader based material.

Parameters

#Material A number to identify the new material. #PB_Any can be used to auto-generate this number.

ShaderID The shader identifier. CreateShader() is used to create a new shader. The ShaderID field must contain either

- the ID of a shader previously created with CreateShader()
- the ID of a predefined shader:

```
#PB_Material_ColorShader
#PB_Material_PerpixelShader
#PB_Material_BumpShader
#PB_Material_SkyShader
#PB_Material_WaterShader
#PB_Material_WaterShaderRTT
#PB_Material_OceanShader
#PB_Material_PointSpriteSphereShader
#PB_MaterialShader_CubicEnvShader
#PB_MaterialShader_CubicEnvBumpShader
```

Description of the predefined shaders:

```
-----
#PB_Material_ColorShader

Effect: uniform color in 'perpixel'.

texture(s) : none

MaterialShaderParameter : none

Taking into account of :
- SetMaterialColor(#Material ,
  #PB_Material_AmbientColor|#PB_Material_DiffuseColor , color)
```

```

- MaterialShininess(#Material, Shininess , SpecularColor)
- AmbientColor
- Fog
- ScaleMaterial / ScrollMaterial / RotateMaterial

-----
#PB_Material_PerpixelShader

Effect: material in 'perpixel'

texture(s) :
- 0: sampler2D color

MaterialShaderParameter : none

Taking into account of :
- SetMaterialColor(#Material,
  #PB_Material_AmbientColor|#PB_Material_DiffuseColor, color)
- MaterialShininess(#Material, Shininess , SpecularColor)
- LightAttenuation
- AmbientColor
- Fog
- ScaleMaterial / ScrollMaterial / RotateMaterial

-----
#PB_Material_BumpShader

Effect: material with relief

texture(s) :
- 0: sampler2D color
- 1: sampler2D normal

MaterialShaderParameter :
#PB_Shader_Fragment
- float bumpy - default value: 1 (relief amplitude)

Taking into account of :
- SetMaterialColor(#Material,
  #PB_Material_AmbientColor|#PB_Material_DiffuseColor, color)
- MaterialShininess(#Material, Shininess , SpecularColor)
- LightAttenuation
- AmbientColor
- Fog
- ScaleMaterial / ScrollMaterial / RotateMaterial

remark(s):
- the tangents must be defined (use 'BuildMeshTangent')

-----
#PB_Material_SkyShader

Effect: displays a sky with clouds and sun

texture(s) :
- 0: uniform sampler2D color (clouds)

MaterialShaderParameter :
#PB_Shader_Fragment
- vec3 speed - default value: 10 0 0 (speed of clouds (X Z))

```


- float height - default value: 200 (height of the 1st cloud layer)
- int layer - default value: 2 (number of additional cloud layers)
- float scale - default value: 4000 (texture scale)

Taking into account of :

- SetMaterialColor(#Material, #PB_Material_AmbientColor, color): color of the sun sky at the zenith
- SetMaterialColor(#Material, #PB_Material_DiffuseColor, color): color of the sky sun on the horizon
- Fog

#PB_Material_WaterShader

Effect : allows you to display a water surface

texture(s) :

- 0: sampler2D normal

MaterialShaderParameter :

#PB_Shader_Fragment

- float scale - default value: 0.1 (texture scale)
- float height - default value: 1 (wave height)

Taking into account of :

- SetMaterialColor(#Material, #PB_Material_DiffuseColor, color): water color (Alpha: opacity)
- SetMaterialColor(#Material, #PB_Material_SpecularColor, color): sky reflection color
- Fog

#PB_Material_WaterShaderRTT

Effect : allows to display a water surface with reflection (via RTT)

texture(s) :

- 0 : sampler2D normal
- 1 : sampler2D RTT (render to texture)

MaterialShaderParameter : none

Taking into account of :

- SetMaterialColor(#Material, #PB_Material_DiffuseColor, color): water color (Alpha: opacity)
- Fog
- ScaleMaterial / ScrollMaterial / RotateMaterial

#PB_Material_OceanShader

Effect : allows to display a water surface with waves

texture(s) :

- 0: sampler2D normal
- 1: sampler2D color (foam)

```

MaterialShaderParameter :
#PB_Shader_Vertex
- float wavebig - default value: 0.5 (wave amplitude)
- float swell - default value: 0.5 (swell amplitude)
#PB_Shader_Fragment
- float wavelittle - default value: 0.5 (amplitude of "small"
  waves)
- float foam - default value: 0.5 (foam)

```

```

Taking into account of :
- SetMaterialColor(#Material, #PB_Material_AmbientColor,
  color): water color
- SetMaterialColor(#Material, #PB_Material_DiffuseColor,
  color): color of the sky reflection
- Fog

```

```

remark(s):
- the alpha value of the texture 'normal' contains the height
  of the wave

```

```

#PB_Material_PointSpriteSphereShader

```

```

Effect: displays spheres at the vertices of the mesh

```

```

texture(s) : none

```

```

MaterialShaderParameter : none

```

```

Taking into account :
- SetMaterialColor(#Material,
  #PB_Material_AmbientColor|#PB_Material_DiffuseColor, color)
- MaterialShininess(#Material, Shininess , SpecularColor)
- AmbientColor
- Fog

```

```

remark(s):
- the material must be of type PointSprite :
  SetMaterialAttribute(0,#PB_Material_PointSprite,1)
- the mesh must be of type PointList :
  CreateMesh(#Mesh,#PB_Mesh_PointList)
- the radius of the sphere is defined by the texture
  coordinate U
- the color of the sphere is defined by the vertex color
example: MeshVertex(posX, posY, posZ, 10, 0,RGB(255,0,0))

```

```

#PB_MaterialShader_CubicEnv

```

```

Effect: reflection via cubic texture

```

```

texture(s) :
- 0: samplerCube cubemap
- 1: uniform sampler2D color

```

```

MaterialShaderParameter :
#PB_Shader_Fragment
- float glossy - default value : 0.5 (reflect intensity)

```

```

Taking into account of :
- SetMaterialColor(#Material,
  #PB_Material_AmbientColor|#PB_Material_DiffuseColor, color)
- AmbientColor
- Fog
- ScaleMaterial / ScrollMaterial / RotateMaterial

remark(s) :
- cubic textures must be 'clamped':
  SetMaterialAttribute(#texture, #PB_Material_TAM,
    #PB_Material_ClampTAM)

-----
#PB_MaterialShader_CubicEnvBump

Effect: reflection via cubic texture with relief

texture(s) :
- 0: samplerCube cubemap
- 1: sampler2D color
- 2: sampler2D normal

MaterialShaderParameter :
#PB_Shader_Fragment
- float glossy - default value : 0.5 (reflect intensity)
- float bumpy - default value : 1 (relief amplitude)

Taking into account of :
- SetMaterialColor(#Material,
  #PB_Material_AmbientColor|#PB_Material_DiffuseColor, color)
- MaterialShininess(#Material, Shininess, SpecularColor)
- AmbientColor
- Fog
- ScaleMaterial / ScrollMaterial / RotateMaterial

remark(s) :
- cubic textures must be 'clamped' :
  SetMaterialAttribute(#texture, #PB_Material_TAM,
    #PB_Material_ClampTAM)
- the tangents must be defined (use 'BuildMeshTangent')

```

Return value

Nonzero if the shader based material was successfully created, zero otherwise. If `#PB_Any` was used for the `#Material` parameter, then the generated number is returned on success.

136.8 MaterialShaderAutoParameter

Syntax

```
MaterialShaderAutoParameter(#Material, ProgramType, ParameterName$,
  ParameterType, ExtraValue.f)
```

Description

Set a parameter value for the shader based material previous created with `CreateShaderMaterial()` .

Parameters

#Material The material to use.

ProgramType Program type. It can be one of the following values:

```
#PB_Shader_Vertex    : use the vertex program.  
#PB_Shader_Fragment : use the fragment program.
```

ParameterName\$ Name of the parameter with the GLSL program to use.

ParameterType Parameter type. It can be one of the following values:

```
#PB_Shader_AmbientLightColour  
#PB_Shader_AnimationParametric  
#PB_Shader_CameraPosition  
#PB_Shader_CameraPositionObjectSpace  
#PB_Shader_Costime02pi  
#PB_Shader_Costime0X  
#PB_Shader_Custom  
#PB_Shader_DerivedAmbientLightColour  
#PB_Shader_DerivedLightDiffuseColour  
#PB_Shader_DerivedLightDiffuseColourArray  
#PB_Shader_DerivedLightSpecularColour  
#PB_Shader_DerivedLightSpecularColourArray  
#PB_Shader_DerivedSceneColour  
#PB_Shader_FarClipDistance  
#PB_Shader_FogColour  
#PB_Shader_Fov  
#PB_Shader_Fps  
#PB_Shader_Fragment  
#PB_Shader_Integer  
#PB_Shader_InverseProjectionMatrix  
#PB_Shader_InverseTextureSize  
#PB_Shader_InverseTransposeProjectionMatrix  
#PB_Shader_InverseTransposeViewMatrix  
#PB_Shader_InverseTransposeViewprojMatrix  
#PB_Shader_InverseTransposeWorldMatrix  
#PB_Shader_InverseTransposeWorldviewMatrix  
#PB_Shader_InverseTransposeWorldviewprojMatrix  
#PB_Shader_InverseViewMatrix  
#PB_Shader_InverseViewportHeight  
#PB_Shader_InverseViewportWidth  
#PB_Shader_InverseViewprojMatrix  
#PB_Shader_InverseWorldMatrix  
#PB_Shader_InverseWorldviewMatrix  
#PB_Shader_InverseWorldviewprojMatrix  
#PB_Shader_LightAttenuation  
#PB_Shader_LightAttenuationArray  
#PB_Shader_LightCastsShadows  
#PB_Shader_LightCount  
#PB_Shader_LightCustom  
#PB_Shader_LightDiffuseColour  
#PB_Shader_LightDiffuseColourArray  
#PB_Shader_LightDiffuseColourPowerScaled  
#PB_Shader_LightDiffuseColourPowerScaledArray  
#PB_Shader_LightDirection  
#PB_Shader_LightDirectionArray  
#PB_Shader_LightDirectionObjectSpace  
#PB_Shader_LightDirectionObjectSpaceArray  
#PB_Shader_LightDirectionViewSpace
```

```

#PB_Shader_LightDirectionViewSpaceArray
#PB_Shader_LightDistanceObjectSpace
#PB_Shader_LightDistanceObjectSpaceArray
#PB_Shader_LightNumber
#PB_Shader_LightPosition
#PB_Shader_LightPositionArray
#PB_Shader_LightPositionObjectSpace
#PB_Shader_LightPositionObjectSpaceArray
#PB_Shader_LightPositionViewSpace
#PB_Shader_LightPositionViewSpaceArray
#PB_Shader_LightPowerScale
#PB_Shader_LightPowerScaleArray
#PB_Shader_LightSpecularColour
#PB_Shader_LightSpecularColourArray
#PB_Shader_LightSpecularColourPowerScaled
#PB_Shader_LightSpecularColourPowerScaledArray
#PB_Shader_LodCameraPosition
#PB_Shader_LodCameraPositionObjectSpace
#PB_Shader_NearClipDistance
#PB_Shader_PackedTextureSize
#PB_Shader_PassIterationNumber
#PB_Shader_PassNumber
#PB_Shader_ProjectionMatrix
#PB_Shader_SceneDepthRange
#PB_Shader_ShadowColour
#PB_Shader_ShadowExtrusionDistance
#PB_Shader_ShadowSceneDepthRange
#PB_Shader_Sintime01
#PB_Shader_Sintime02pi
#PB_Shader_Sintime0X
#PB_Shader_SpotlightParams
#PB_Shader_SpotlightParamsArray
#PB_Shader_SpotlightViewprojMatrix
#PB_Shader_SpotlightViewprojMatrixArray
#PB_Shader_SpotlightWorldviewprojMatrix
#PB_Shader_SurfaceAmbientColour
#PB_Shader_SurfaceDiffuseColour
#PB_Shader_SurfaceEmissiveColour
#PB_Shader_SurfaceShininess
#PB_Shader_SurfaceSpecularColour
#PB_Shader_Tantime01
#PB_Shader_Tantime02pi
#PB_Shader_Tantime0X
#PB_Shader_TexelOffsets
#PB_Shader_TextureMatrix
#PB_Shader_TextureSize
#PB_Shader_TextureViewprojMatrix
#PB_Shader_TextureViewprojMatrixArray
#PB_Shader_TextureWorldviewprojMatrixArray
#PB_Shader_Time
#PB_Shader_Time01
#PB_Shader_Time01Packed
#PB_Shader_Time02pi
#PB_Shader_Time02piPacked
#PB_Shader_Time0X
#PB_Shader_Time0XPacked
#PB_Shader_TransposeProjectionMatrix
#PB_Shader_TransposeViewMatrix
#PB_Shader_TransposeViewprojMatrix

```

```

#PB_Shader_TransposeWorldMatrix
#PB_Shader_TransposeWorldviewMatrix
#PB_Shader_TransposeWorldviewprojMatrix
#PB_Shader_Vector3
#PB_Shader_Vector4
#PB_Shader_Vertex
#PB_Shader_VertexWinding
#PB_Shader_ViewDirection
#PB_Shader_ViewMatrix
#PB_Shader_ViewportSize
#PB_Shader_ViewportWidth
#PB_Shader_ViewprojMatrix
#PB_Shader_ViewSideVector
#PB_Shader_ViewUpVector
#PB_Shader_WorldDualquaternionArray2x4
#PB_Shader_WorldMatrix
#PB_Shader_WorldMatrixArray
#PB_Shader_WorldMatrixArray3x4
#PB_Shader_WorldviewMatrix
#PB_Shader_WorldviewprojMatrix

```

ExtraValue Extra value passed to the program parameter.

Return value

None.

Remarks

The automatic parameters correspond to the parameters of the 3D engine (positions of the camera, lights, color of the material, fog...) this function allows to make the link between the 'uniform' variable of the shader and these parameters.

Example

```

1  MaterialShaderAutoParameter(#Material, #PB_Shader_Fragment,
    #PB_Shader_CameraPosition, "camerapos", 0)

```

See Also

CreateShaderMaterial()

136.9 MaterialShaderParameter

Syntax

```

MaterialShaderParameter(#Material, ProgramType, ParameterName$,
    ParameterType, v1.f, v2.f, v3.f, v4.f)

```

Description

Set specific parameters to the shader.

Parameters

#Material The material to use.

ProgramType Program type. It can be one of the following values:

```
#PB_Shader_Vertex : use the vertex program.  
#PB_Shader_Fragment: use the fragment program.
```

ParameterName\$ Name of the parameter with the GLSL program to set the new value.

ParameterType Parameter type. It can be one of the following values:

```
#PB_Shader_Integer: the parameter use an integer type (v2, v3  
and v4 parameters are ignored)  
#PB_Shader_Float : the parameter use a float type (v2, v3  
and v4 parameters are ignored)  
#PB_Shader_Vector3: the parameter use a Vector3 type (v4  
parameter is ignored)  
#PB_Shader_Vector4: the parameter use a Vector4 type
```

v1 First parameter value. Mandatory for all parameter types.

v2 Second parameter value. Mandatory for #PB_Shader_Vector3 and #PB_Shader_Vector4 types.

v3 Third parameter value. Mandatory for #PB_Shader_Vector3 and #PB_Shader_Vector4 types.

v4 Fourth parameter value. Mandatory for #PB_Shader_Vector4 type.

Return value

None.

Remarks

GLSL shaders uses vec4 type to pass a color. Example:

```
1 MaterialShaderParameter(0, #PB_Shader_Fragment, 4,  
   Red(Color)/255, Green(Color)/255, Blue(Color)/255,  
   Alpha(Color)/255)  
2 MaterialShaderParameter(#Material, #PB_Shader_Fragment, "bumpy",  
   #PB_Shader_Float, 0.5,0,0,0)
```

136.10 MaterialShaderTexture

Syntax

```
MaterialShaderTexture(#Material, TextureID1, TextureID2,  
   TextureID3, TextureID4)
```

Description

Sets the textures to use for the shader based material previous created with CreateShaderMaterial() .

Parameters

#Material The shader material to use.

TextureID1 The first texture to use. Use `TextureID()` to get a valid `TextureID`, or `#Null` if not needed.

TextureID2 The second texture to use. Use `TextureID()` to get a valid `TextureID`, or `#Null` if not needed.

TextureID3 The third texture to use. Use `TextureID()` to get a valid `TextureID`, or `#Null` if not needed.

TextureID4 The fourth texture to use. Use `TextureID()` to get a valid `TextureID`, or `#Null` if not needed.

Return value

None.

136.11 DisableMaterialLighting

Syntax

```
DisableMaterialLighting(#Material, State)
```

Description

Enables or disables the dynamic `#Material` lighting. The object which will use this material will be not affected by a dynamic light, created with the `CreateLight()` function. Dynamic lighting is enabled by default when a material is created.

Parameters

#Material The material to use.

State It can be one of the following values:

```
#True : dynamic lighting is disabled.  
#False: dynamic lighting is enabled.
```

Return value

None.

Remarks

To get the current material lighting state, use `GetMaterialAttribute()` .

136.12 FreeMaterial

Syntax

```
FreeMaterial(#Material)
```

Description

Frees the specified `#Material`. All its associated memory is released and this object can't be used anymore.

Parameters

#Material The material to free. If **#PB_All** is specified, all the remaining materials are freed.

Return value

None.

Remarks

All remaining materials are automatically freed when the program ends.

136.13 IsMaterial

Syntax

```
Result = IsMaterial(#Material)
```

Description

Tests if the given material is valid and correctly initialized.

Parameters

#Material The material to test.

Return value

Nonzero if the material is valid, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a material is ready to use.

136.14 GetMaterialAttribute

Syntax

```
Result = GetMaterialAttribute(#Material, Attribute)
```

Description

Get the specified material attribute.

Parameters

#Material The material to use.

Attribute The attribute to get. It can be one of the following values:

```

#PB_Material_Shininess      : get the material shininess, as
    set with MaterialShininess()
.
#PB_Material_TextureRotate  : get the material rotate value,
    in degree.
#PB_Material_TextureUScale  : get the material uscale value,
    see ScaleMaterial()
.
#PB_Material_TextureVScale  : get the material vscale value,
    see ScaleMaterial()
.
#PB_Material_TextureUScroll : get the material uscroll value,
    see ScrollMaterial()
.
#PB_Material_TextureVScroll : get the material vscroll value,
    see ScrollMaterial()
.
#PB_Material_DepthCheck     : get the material depth check
    state (enabled or disabled).
#PB_Material_DepthWrite     : get the material depth write
    state (enabled or disabled).
#PB_Material_Lighting       : get the material lightning
    value, as set with DisableMaterialLighting()
.
#PB_Material_ShadingMode    : get the material shading mode
    value, as set with MaterialShadingMode()
.
#PB_Material_CullingMode    : get the material culling mode
    value, as set with MaterialCullingMode()
.

```

Return value

The value of the specified attribute.

136.15 GetMaterialColor

Syntax

```
Result = GetMaterialColor(#Material, Type)
```

Description

Get the specified material color.

Parameters

#Material The material to use.

Type The color type to get. It can be one of the following values:

```

#PB_Material_AmbientColor: the color used by default,
    without other lighting.
#PB_Material_DiffuseColor: the color the material will
    reflect when it will be dynamically lighted.
                                For example, using a full white
                                color will result at normal lighting (all colors

```

are reflected). If you use a red, then only the red colors of the material will be reflected, resulting as a full red material (or black one, if the material doesn't contain any red color).

#PB_Material_SpecularColor: the color the material will reflect when it will be dynamically lighted by a light which has a specular color value. For example, using a full white color will result at normal lighting (all colors are reflected). If you use a red, then only the red colors of the material will be reflected, resulting as a full red material (or black one, if the material doesn't contain any red color).

#PB_Material_SelfIlluminationColor: the color the material will emit even if no light reaches it.

Return value

Returns the RGB color value. Separate color channel value can easily be retrieved with Red() , Green() and Blue() .

See Also

SetMaterialColor()

136.16 SetMaterialColor

Syntax

```
SetMaterialColor(#Material , Type , Color)
```

Description

Set the specified material color.

Parameters

#Material The material to use.

Type The color type to set. It can be a combination of the following values:

#PB_Material_AmbientColor: the color used by default, without other lighting.

#PB_Material_DiffuseColor: the color the material will reflect when it will be dynamically lighted.
For example, using a full white color will result at normal lighting (all colors are reflected). If you use a red, then only the red colors of the material will be reflected, resulting as a full red material (or black one, if the material doesn't contain any red color).

#PB_Material_SpecularColor: the color the material will reflect when it will be dynamically lighted by a light

which has a specular color value. For example, using a full white color will result at normal lighting (all colors are reflected). If you use a red, then only the red colors of the material will be reflected, resulting as a full red material (or black one, if the material doesn't contain any red color).

`#PB_Material_SelfIlluminationColor`: the color the material will emit even if no light reaches it.

Color The RGB color value to set. A valid RGB value can be created with `RGB()`.

Return value

None.

See Also

`GetMaterialColor()`

136.17 MaterialBlendingMode

Syntax

`MaterialBlendingMode(#Material, Mode)`

Description

Changes the way the material will be blended with the scene (screen background).

Parameters

#Material The material to use.

Mode It can be one of the following values:

`#PB_Material_Add` : Performs a pixel 'Add' operation over the scene (black color is like transparent).

`#PB_Material_AlphaBlend`: Uses the AlphaChannel layer of the texture (should be a TGA or PNG one) to blend it with the scene.

`#PB_Material_Color` : Uses the texture transparent color value when blending the material with the scene.

Return value

None.

136.18 MaterialFilteringMode

Syntax

`MaterialFilteringMode(#Material, Mode [, MaxAnisotropicValue])`

Description

Changes the material filtering mode.

Parameters

#Material The material to use. If this parameter value is set to **#PB_Default**, the filtering mode value used for future created material is changed.

Mode It can be one of the following values:

```
#PB_Material_None      : Don't filter the material which
                        becomes very pixelated when the camera gets close.
#PB_Material_Bilinear  : Performs a bilinear filtering when
                        the camera gets close, resulting to a smooth, a bit blurred
                        picture.
#PB_Material_Trilinear : Performs a trilinear filtering when
                        the camera gets close, resulting in the best picture quality
                        possible.
#PB_Material_Anisotropic: Set the maximum anisotropic value.
```

When a material is created, the bilinear filtering is used by default. Using a filter doesn't have a big performance impact on the rendering, as many graphics cards do it using the hardware.

MaxAnisotropicValue (optional) The maximum anisotropic value, if the mode **#PB_Material_Anisotropic** is set. This value is usually between 1 and 8.

Return value

None.

136.19 MaterialID

Syntax

```
MaterialID = MaterialID(#Material)
```

Description

Returns the unique system identifier of the material.

Parameters

#Material The material to use.

Return value

The unique system identifier of the material.

136.20 MaterialShadingMode

Syntax

```
MaterialShadingMode(#Material, Mode)
```

Description

Changes the **#Material** shading mode.

Parameters

#Material The material to use.

Mode It can be one of the following values:

```
#PB_Material_Flat      : The material uses the flat mode, the
    lighting is done face by face.
#PB_Material_Gouraud   : Performs a shading using the Gouraud
    algorithm (default).
#PB_Material_Phong     : Performs a shading using the Phong
    algorithm.
```

combined with one of the following values:

```
#PB_Material_Solid     : The material will be rendered in
    solid, textured mode (default).
#PB_Material_Wireframe : The material will be rendered in
    wireframe mode.
#PB_Material_Point     : The material will be rendered using
    only edge points.
```

Return value

None.

Remarks

To get the current material shading mode, use `GetMaterialAttribute()` .

See Also

`GetMaterialAttribute()`

136.21 MaterialCullingMode

Syntax

```
MaterialCullingMode(#Material , Mode)
```

Description

Set the culling mode for the material.

Parameters

#Material The material to use.

Mode It can be one of the following values:

```
#PB_Material_NoCulling      : no culling.
#PB_Material_ClockWiseCull  : clockwise culling.
#PB_Material_AntiClockWiseCull : anticlockwise culling.
```

Return value

None.

Remarks

To get the current material culling mode, use `GetMaterialAttribute()` .

See Also

`GetMaterialAttribute()`

136.22 MaterialShininess

Syntax

```
MaterialShininess(#Material, Shininess [, SpecularColor])
```

Description

Changes the shininess of the `#Material` (the size of the specular highlights).

Parameters

#Material The material to use.

Shininess The new shininess value.

SpecularColor (optional) The specular color to use. A valid RGB color can be created with `RGB()` .

Return value

None.

Remarks

To get the current shininess value, use `GetMaterialAttribute()` .

See Also

`GetMaterialAttribute()`

136.23 MaterialTextureAliases

Syntax

```
MaterialTextureAliases(#Material, TextureID1, TextureID2,  
TextureID3, TextureID4)
```

Description

Set textures for use in a material script. It allows to use the same material script with dynamic textures. In the material script, the texture reference needs to be changed from `'texture mytexture.jpg'` to `'texture_alias texture1'` (or `'texture_alias texture2'`, `'texture_alias texture3'`, `'texture_alias texture4'`).

Parameters

#Material The material to use.

TextureID1 The TextureID() to use for the first texture alias (identified as 'texture_alias texture1' in the material script), or zero if no texture is needed.

TextureID2 The TextureID() to use for the second texture alias (identified as 'texture_alias texture2' in the material script), or zero if no texture is needed.

TextureID3 The TextureID() to use for the third texture alias (identified as 'texture_alias texture3' in the material script), or zero if no texture is needed.

TextureID4 The TextureID() to use for the fourth texture alias (identified as 'texture_alias texture4' in the material script), or zero if no texture is needed.

Return value

None.

See Also

GetScriptMaterial() , TextureID()

136.24 GetScriptMaterial

Syntax

```
Result = GetScriptMaterial(#Material , Name$)
```

Description

Get a material defined in an OGRE script file. Scripts are loaded and parsed when calling Parse3DScripts() .

Parameters

#Material A number to identify the new material. #PB_Any can be used to auto-generate this number.

Name\$ The name of the material in the script files.

Return value

Nonzero if the material was successfully created, zero otherwise. If #PB_Any was used for the #Material parameter then the generated number is returned on success.

136.25 MaterialFog

Syntax

```
MaterialFog(#Material , Color , Intensity , StartDistance , EndDistance)
```

Description

Adds a fog effect on the specified material.

Parameters

#Material The material to use.

Color The color of the fog effect. `RGB()` can be used to get a valid color value.

Intensity The fog intensity. If set to zero, the fog effect is removed.

StartDistance The distance from (in world units) where the fog should start.

EndDistance The distance from (in world units) where the fog is fully opaque.

Return value

None.

136.26 ReloadMaterial

Syntax

```
ReloadMaterial(MaterialName$, ScriptFilename$, ParseScript)
```

Description

Reloads a material from an OGRE script based on its name. This is useful when using customized materials stored in script files.

Parameters

MaterialName\$ The material name in scripts.

ScriptFilename\$ The script filename.

ParseScript If set to `#True` the 'ScriptFilename\$' is parsed again to get updated material information. If set to `#False`, the material from when the script was parsed first will be used. A material can be retrieved from OGRE scripts with `GetScriptMaterial()` .

Return value

None.

136.27 ResetMaterial

Syntax

```
ResetMaterial(ObjectType)
```

Description

Resets all materials for the specified object types.

Parameters

ObjectType It can be one of the following values:

```
#PB_Entity: resets materials for all entities
.
#PB_ParticleEmitter: resets materials for all particle
emitters
.
#PB_BillboardGroup: resets materials for all billboards
.
```

Return value

None.

136.28 SetMaterialAttribute

Syntax

```
SetMaterialAttribute(#Material, Attribute, Value [, Layer])
```

Description

Sets the specified attribute value to the given material.

Parameters

#Material The material to use.

Attribute The attribute to set. It can be one of the following values:

```
#PB_Material_DepthCheck : Enables or disables the depth check
for the material.
                                Value can be #True (depth check
                                enabled) or #False (depth check disabled).
#PB_Material_DepthWrite : Enables or disables the depth write
for the material.
                                Value can be #True (depth write
                                enabled) or #False (depth write disabled).
#PB_Material_AlphaReject: Enables alpha reject on the texture
(useful for semi transparent textures like
trees, window, etc).
#PB_Material_TAM        : Changes the 'Texture Addressing
Mode' (TAM). It can be one of the following values:
                                #PB_Material_WrapTAM : wrap mode
                                #PB_Material_MirrorTAM: mirror mode
                                #PB_Material_ClampTAM : clamp mode
                                #PB_Material_BorderTAM: border mode
#PB_Material_EnvironmentMap: Enables environment map. It can
be one of the following values:
                                #PB_Material_NoMap    : disable
environment map
                                #PB_Material_PlanarMap : planar
environment map
                                #PB_Material_CurvedMap : curved
environment map
```

```

                                #PB_Material_ReflectionMap:
reflection environment map
                                #PB_Material_NormalMap      : normal
environment map
#PB_Material_ProjectiveTexturing: Enable projective texturing
                                for this material. The value is a #Camera number to use.
#PB_Material_PointSprite : Enables or disables point sprite
                                for the material.
                                If greater than zero, the value is
                                the point size / 1000. If zero, point size is disabled.
#PB_Material_DepthBias      : Enables or disables the depth bias
                                for the material.
                                If greater than zero, the value is
                                the depth bias / 1000. If zero, depth bias is disabled.

```

Value Value of the attribute to set.

Layer (optional) The layer to use. The first layer is zero (if this parameter is omitted, the layer zero is used).

Return value

None.

See Also

GetMaterialAttribute()

136.29 ScrollMaterial

Syntax

```
ScrollMaterial(#Material, x, y, Mode [, Layer])
```

Description

Scrolls the material layer according to x,y values.

Parameters

#Material The material to use.

x, y The texture scroll offset, in pixels.

Mode It can be one of the following values:

```

#PB_Material_Fixed      : The material is scrolled of the
                          specified x,y offset, without regards of the previous scroll
                          function.
#PB_Material_Animated: Each frame is the material is scrolled
                          of the specified x,y offset automatically.

```

Layer (optional) The layer to scroll. The first layer is zero (if this parameter is omitted, the layer zero is scrolled).

Return value

None.

Remarks

To get the current scroll values, use `GetMaterialAttribute()` .

See Also

`GetMaterialAttribute()`

136.30 RemoveMaterialLayer

Syntax

```
RemoveMaterialLayer(#Material)
```

Description

Removes the top most (last added) material layer.

Parameters

#Material The material to use.

Return value

None.

See Also

`AddMaterialLayer()`

136.31 ScaleMaterial

Syntax

```
ScaleMaterial(#Material, x, y [, Layer])
```

Description

Scales the material. The parameters 'x' and 'y' are scale factors.

Parameters

#Material The material to use.

x, y The scale factors (the current material size is multiplied by these values):

- a value of 1.0 means that the size isn't changed
- a value between 0.0 and 1.0 means that the material is scaled down (ie: a scale of 0.5 will be half the size)
- a value above 1.0 means that the material is scaled up (ie: a scale of 2.0 will double the size)

Layer (optional) The layer to scale. The first layer is zero (if this parameter is omitted, the layer zero is scaled).

Return value

None.

Remarks

To get the current scale values, use `GetMaterialAttribute()` .

See Also

`GetMaterialAttribute()`

136.32 RotateMaterial

Syntax

```
RotateMaterial(#Material, Angle, Mode [, Layer])
```

Description

Rotates the material layer according to the angle value.

Parameters

#Material The material to use.

Angle The rotation angle, in degree.

Mode It can be one of the following values:

```
#PB_Material_Fixed    : The material is rotated of the  
    specified 'Angle', without regards of the previous rotate  
    function.  
#PB_Material_Animated: Each frame is the material is rotated  
    of the specified 'Angle' automatically.
```

Layer (optional) The layer to rotate. The first layer is zero (if this parameter is omitted, the layer zero is rotated).

Return value

None.

Remarks

To get the current rotate value, use `GetMaterialAttribute()` .

See Also

`GetMaterialAttribute()`

136.33 MaterialAnimation

Syntax

```
MaterialAnimation(#Material, Texture$, NbFrames, Time.f)
```

Description

Add an animated texture to the material. An animated texture is composed of any number of textures, all the same size, with the frame number appended before the extension in their filename. For example, if "test.jpg" is specified as 'Texture\$' and 'NbFrames' is set to 3, the textures "test_0.jpg", "test_1.jpg" and "test_2.jpg" will be loaded and used for the material.

Parameters

#Material The material to use.

Texture\$ The texture filename to use.

NbFrames Number of frames of the animated texture.

Time Time (in milliseconds) to play the whole animation. Once fully played, it will loop again from the start.

Return value

None.

Remarks

Usually, all frames are contained in a zip file with the same name as texture (ex Texture.zip) and accessible by Add3DArchive() and Parse3DScripts() .

Chapter 137

Math

Overview

The math library provides basic mathematical functions such as: `Cos()` , `Sin()` , `Pow()` , `Log()` etc... Almost all these functions work with floats or doubles, i.e. numbers with the `.f` or `.d` extensions. **If a function is used with a double value as input or output, it will automatically switch to double precision instead of single precision for its calculations.**

137.1 Abs

Syntax

```
Result.f(.d) = Abs(Number.f(.d))
```

Description

Returns the absolute value of the given float value.

Parameters

Number.f or Number.d The number from which to get the absolute value. This function will only work correctly with float numbers. With integers, it will fail if the integer is too large (due to a loss of precision).

Return value

Returns the absolute value. The return-value is always positive.

Remarks

This function handles and returns float or double values.

Example

```
1  Debug Abs(3.14159) ; Will display '3.14159'
2  Debug Abs(-3.14159) ; will display '3.14159'
3
4  pif.f = -3.14159
5  pid.d = -3.14159
6  Debug Abs(pif) ; Returns a float number
7  Debug Abs(pid) ; Returns a double number
```

See Also

Sign()

137.2 ACos

Syntax

```
Result.f(.d) = ACos(Value.f(.d))
```

Description

Returns the arc-cosine of the specified value.

Parameters

Value.f or **Value.d** The input value. It must be between -1.0 and 1.0 (-1.0 and 1.0 included).

Return value

Returns the resulting angle in radian. It can be transformed into degrees using the Degree() function.

Remarks

This is the inverse function of Cos() . This function handles and returns float or double values.

Example

```
1  Debug ACos(1) ; will display '0.0'
2  Debug Acos(-1) ; will display 'pi' (approximately 3.14159...)
```

See Also

Cos() , ACosH() , Degree()

137.3 ACosH

Syntax

```
Result.f(.d) = ACosH(Value.f(.d))
```

Description

Returns the area hyperbolic cosine of the specified value.

Parameters

Value.f or **Value.d** The input value. It must be greater than or equal to 1.

Return value

Returns the hyperbolic angle.

Remarks

This is the inverse function of `CosH()` . This function handles and returns float or double values.

Example

```
1  Debug ACosH(1) ; will display '0.0'
2  Debug Exp(ACosH(0.5 * Sqr(5))) ; will display '1.618033...' (the
    golden ratio)
```

See Also

`CosH()` , `@acos`

137.4 ASin

Syntax

```
Result.f(.d) = ASin(Value.f(.d))
```

Description

Returns the arc-sine of the specified value.

Parameters

Value.f or **Value.d** The input value. It must be between -1.0 and 1.0 (-1.0 and 1.0 included).

Return value

Returns the resulting angle in radian. It can be transformed into degrees using the `Degree()` function.

Remarks

This is the inverse function of `Sin()` . This function handles and returns float or double values.

Example

```
1  Debug ASin(1) ; will display '1.570796...' (pi/2)
2  Debug ASin(0) ; will display '0.0'
```

See Also

`Sin()` , `ASinH()` , `Degree()`

137.5 ASinH

Syntax

`Result.f(.d) = ASinH(Value.f(.d))`

Description

Returns the area hyperbolic sine of the specified value.

Parameters

Value.f or Value.d The input value. The range of Value.f(.d) is not limited.

Return value

Returns the hyperbolic angle.

Remarks

This is the inverse function of SinH() . This function handles and returns float or double values.

Example

```
1  Debug ASinH(0)           ; will display '0.0'
2  Debug Exp(ASinH(0.5))    ; will display '1.618033...' (the golden
    ratio)
```

See Also

SinH() , ASin()

137.6 ATan

Syntax

`Result.f(.d) = ATan(Value.f(.d))`

Description

Returns the arc-tangent of the specified value.

Parameters

Value.f or Value.d The input value. Its range is not limited.

Return value

Returns the resulting angle in radian. It can be transformed into degrees using the Degree() function.

Remarks

This is the inverse function of Tan() . This function handles and returns float or double values.

Example

```
1  Debug ATan(1) ; will display '0.785398...' (pi/4)
```

See Also

Tan() , ATanH() , Degree()

137.7 ATan2

Syntax

```
Result.f(.d) = ATan2(x.f(.d), y.f(.d))
```

Description

Calculates the angle in radian between the x axis and a line drawn in the direction specified by 'x' and 'y'. It can be used to calculate angles between lines in 2D or to transform rectangular coordinates into polar coordinates.

Parameters

x.f, y.f or x.d, y.d The direction of the line to calculate the angle from. Zero values are allowed.

Return value

Returns the resulting angle in radian. The result can be transformed into degrees using the Degree() function.

Remarks

This function calculates the value $\text{ATan}(y/x)$ and examines the sign of x and y to place the angle in the correct quadrant. It also handles the cases where y is zero to avoid division by zero errors. The result is always between $-\pi$ and $+\pi$. Negative angles indicate that the line is below the x axis, positive values indicate that line is above the x axis. If 'x' and 'y' are zero then the function returns 0.0.

This function handles and returns float or double values.

Example

```
1  Debug ATan2(10, 10) ; will display #PI/4 (45 degrees in radian)
```

See Also

ATan() , Degree()

137.8 ATanH

Syntax

```
Result.f(.d) = ATanH(Value.f(.d))
```

Description

Returns the area hyperbolic tangent of the specified value.

Parameters

Value.f or Value.d The input value. It must be between -1.0 and 1.0 (not including -1.0 and 1.0).

Return value

Returns the hyperbolic angle.

Remarks

This is the inverse function of TanH() . This function handles and returns float or double values.

Example

```
1  Debug Exp(ATanH(0.2 * Sqr(5))) ; will display '1.618033...' (the  
    golden ratio)
```

See Also

TanH() , ATan()

137.9 Cos

Syntax

Result.f(.d) = Cos(Angle.f(.d))

Description

Returns the cosine of the specified angle.

Parameters

Angle.f or Angle.d The input angle in radians. Use Radian() to convert an angle from degrees to radian.

Return value

Returns the cosine of the angle. The result is always between -1.0 and 1.0.

Remarks

The inverse function of Cos() is ACos() . This function handles and returns float or double values.

Example

```
1  Debug Cos(3.141593)
```

See Also

ACos() , CosH() , Radian()

137.10 CosH

Syntax

`Result.f(.d) = CosH(Angle.f(.d))`

Description

Returns the hyperbolic cosine of the specified hyperbolic angle.

Parameters

Angle.f or Angle.d The input hyperbolic angle.

Return value

Returns the hyperbolic cosine of the angle. The result will be greater than or equal to 1.

Remarks

The inverse function of `CosH()` is `ACosH()` . This function handles and returns float or double values.

Example

```
1  Debug CosH(0) ; will display 1.0
```

See Also

`ACosH()` , `Cos()`

137.11 Degree

Syntax

`Result.f(.d) = Degree(Angle.f(.d))`

Description

Converts the given angle from radian to degree.

Parameters

Angle.f or Angle.d The input angle in radian.

Return value

Returns the angle in degrees.

Remarks

There is no normalization to ensure that the resulting angle is between 0 and 360. If the input was larger than $\#PI*2$ then the result will be larger than 360. Likewise, a negative input will result in a negative output.

The reverse transformation can be made with the `Radian()` function. This function handles and returns float or double values.

Example

```
1  Debug Degree(#PI/4) ; will display 45.0
```

See Also

`Radian()`

137.12 Exp

Syntax

```
Result.f(.d) = Exp(Number.f(.d))
```

Description

Returns the result of the exponential function. This is the value e raised to the power 'Number'.

Parameters

Number.f or **Number.d** The input for the exponential function.

Return value

Returns the value e raised to the power 'Number'.

Remarks

This is the inverse function of `Log()` . This function handles and returns float or double values.

See Also

`Log()`

137.13 Infinity

Syntax

```
Result.f(.d) = Infinity()
```

Description

Returns the special floating-point value representing positive infinity. Negative infinity can be calculated using `"-Infinity"`.

Parameters

None.

Return value

Returns the value representing infinity. The result is a float or double value depending on whether it is assigned to a float or double variable.

Remarks

Infinity and negative infinity are special values. They behave in calculations in the way you would generally expect. For example dividing infinity by any positive number (except 0 or infinity) will result in infinity again. The `IsInfinity()` function can be used to check if a value represents positive or negative infinity.

Example

```
1  Debug IsInfinity(Infinity() / 1000) ; will display 1.0
```

See Also

`IsInfinity()` , `NaN()`

137.14 Int

Syntax

```
Result = Int(Number.f(.d))
```

Description

Returns the integer part of a float number.

Parameters

Number.f or **Number.d** The input value. This can be a float or double value.

Return value

Returns the integer part by cutting off the fractional part.

Remarks

This function returns an integer value. To get a quad value, use the `IntQ()` function. The integer part is created by cutting off the fractional part of the value. There is no rounding. To perform rounding, use the `Round()` function.

Example

```
1  Debug Int(10.565) ; will display '10'
```

See Also

`IntQ()` , `Round()`

137.15 IntQ

Syntax

```
Result = IntQ(Number.f(.d))
```

Description

Returns the integer part of a float number as a quad.

Parameters

Number.f or **Number.d** The input value. This can be a float or double value.

Return value

Returns the integer part by cutting off the fractional part.

Remarks

This function returns a quad value. The `Int()` function has only the range of an integer, but it may be faster to execute on 32-bit systems.

The integer part is created by cutting off the fractional part of the value. There is no rounding. To perform rounding, use the `Round()` function.

Example

```
1  Debug IntQ(12345678901.565) ; will display '12345678901'
```

See Also

`Int()` , `Round()`

137.16 IsInfinity

Syntax

```
Result.f(.d) = IsInfinity(Value.f(.d))
```

Description

Returns nonzero if the given value represents positive or negative infinity.

Parameters

Value.f or **Value.d** The value to check for infinity.

Return value

Returns nonzero if the input value represents positive or negative infinity and zero otherwise.

Remarks

Checking for the infinity values should not be done using normal comparison, because it depends on the hardware implementation whether infinity is considered equal to itself or not. The value of positive infinity can be generated with the `Infinity()` function.

This function handles and returns float or double values.

Example

```
1 Result = IsInfinity(Infinity()) ; infinity
2 Result = IsInfinity(Log(0))    ; -infinity
3 Result = IsInfinity(1234.5)    ; a finite number
4 Result = IsInfinity(NaN())     ; NaN is not the same as infinity
```

See Also

`Infinity()` , `IsNaN()`

137.17 IsNaN

Syntax

```
Result.f(.d) = IsNaN(Value.f(.d))
```

Description

Returns nonzero if the given value 'Not a Number'. This value is the result of some invalid calculations. It can also be generated using the `NaN()` function.

Parameters

Value.f or Value.d The value to check for NaN.

Return value

Returns nonzero if the input value is not a number and zero otherwise.

Remarks

NaN is a special value. Testing for it should not be done using normal comparisons because there are actually many different values for NaN and whether or not NaN is considered equal with itself in comparisons depends on the hardware.

This function handles and returns float or double values.

Example

```
1 Result = IsNaN(NaN()) ; NaN
2 Result = IsNaN(Sqr(-1)) ; NaN
3 Result = IsNaN(1234.5) ; a normal number
4 Result = IsNaN(Infinity()) ; infinity is not NaN
```

See Also

`NaN()` , `IsInfinity()`

137.18 Pow

Syntax

```
Result.f(.d) = Pow(Number.f(.d), Power.f(.d))
```

Description

Returns the given number, raised to the given power.

Parameters

Number.f or **Number.d** The mantissa.

Power.f(d) The exponent. If 'Number.f(d)' is negative, the exponent must be a whole number.

Return value

Returns the value of 'Number' raised to the power 'Power'.

Remarks

This function handles and returns float or double values.

The symbol ' ^ ' itself is not a supported operator. This is what the `Pow()` function is available for.

Example

```
1  Debug Pow(2.0, 3.0)      ; will display '8.0'
2  Debug Pow(27.0, 1/3.0)   ; will display '3.0', the cubic-root
```

See Also

`Sqr()`

137.19 Log

Syntax

```
Result.f(.d) = Log(Number.f(.d))
```

Description

Returns the natural Log (ie log to the base e) of the given number.

Parameters

Number.f or **Number.d** The number to calculate the Log from. This must be a positive value.

Return value

Returns the natural Log of the number.

Remarks

This is the inverse function of `Exp()` . This function handles and returns float or double values.

See Also

`Exp()` , `Log10()`

137.20 Log10

Syntax

```
Result.f(.d) = Log10(Number.f(.d))
```

Description

Returns the log in base 10 of the given number.

Parameters

Number.f or **Number.d** The number to calculate the log from. This must be a positive value.

Return value

Returns the log in base 10.

Remarks

This function handles and returns float or double values.

See Also

`Log()`

137.21 Mod

Syntax

```
Result.f(.d) = Mod(Number.f(.d), Divisor.f(.d))
```

Description

Returns the remainder of the division of `Number.f(.d)` by `Divisor.f(.d)`.

Parameters

Number.f or **Number.d** The number to divide. The input does not need to be a whole number.

Divisor.f or **Divisor.d** The number by which to divide. The input does not need to be a whole number.

Return value

Returns the remainder of the division. The result has the same sign as the `Number.f(d)` parameter.

Remarks

This is the floating-point version of the `'%'` operator for integers. This function handles and returns float or double values.

137.22 NaN

Syntax

```
Result.f(.d) = NaN()
```

Description

Returns the special floating-point value representing 'Not a Number'. This value is returned from invalid calculations such as calculating the square root of a negative number.

Parameters

None.

Return value

Returns the value representing NaN. The result is a float or double value depending on whether it is assigned to a float or double variable.

Remarks

NaN is a special value. Using NaN in any calculation with other values will again return the value NaN. The `IsNaN()` function can be used to check whether a variable represents the value NaN.

Example

```
1  Debug IsNaN(NaN() * 5 + 2) ; will display 1.0
```

See Also

`IsNaN()` , `Infinity()`

137.23 Radian

Syntax

```
Result.f(.d) = Radian(Angle.f(.d))
```

Description

Converts the given angle from degrees into radian.

Parameters

Angle.f or Angle.d The input angle in degree.

Return value

Returns the angle in radian.

Remarks

There is no normalization to ensure that the resulting angle is between 0 and $\#PI*2$. If the input was larger than 360 then the result will be larger than $\#PI*2$. Likewise, a negative input will result in a negative output.

The reverse transformation can be made with the `Degree()` function. This function handles and returns float or double values.

Example

```
1  Debug Radian(90) ; will display #PI/2
```

See Also

`Degree()`

137.24 Random

Syntax

```
Result = Random(Maximum [, Minimum])
```

Description

Returns a random number from zero to the given maximum value (both values included).

Parameters

Maximum The maximum value. This value needs to be positive (zero included) and may not exceed the maximum positive integer value.

Minimum (optional) The minimum value. It may not exceed the maximum value. If specified, the random number will be from the minimum value to the maximum value (both values included). This value needs to be positive (zero included) and may not exceed the maximum positive integer value.

Return value

Returns a value from zero to the maximum value (both values included), unless a minimum value is specified.

Remarks

Additionally RandomSeed() may be used to change the random number seed. RandomData() can be used to fill a memory buffer with random data. RandomizeArray() or RandomizeList() can be used, to randomize the elements of an array or a list.

Note: This command uses a pseudorandom number generator which is very fast and produces randomly looking output but it is not strong enough for cryptographic purposes. The slower CryptRandom() command can be used if more secure random number generation is required.

Example

```
1 Repeat
2     Dice = Random(6, 1) ; get a value between 1 and 6, including 1
      and 6
3     Choice = MessageRequester("Roll the dice", "You got a " + Dice
      + ", roll again?", #PB_MessageRequester_YesNo)
4 Until Choice = #PB_MessageRequester_No
```

See Also

RandomSeed() , RandomData() , CryptRandom() , RandomizeArray() , RandomizeList()

137.25 RandomData

Syntax

```
RandomData(*Buffer, Length)
```

Description

Fills the specified memory buffer with random data.

Parameters

***Buffer** The memory address of the buffer to fill.

Length The size of the buffer to fill.

Return value

None.

Remarks

This command uses the same random seed as the Random() command. RandomSeed() may be used to change the random number seed.

Note: This command uses a pseudorandom number generator which is very fast and produces randomly looking output but it is not strong enough for cryptographic purposes. The slower CryptRandomData() command can be used if more secure random number generation is required.

Example

```
1  ; Create an image with random content
2  ;
3  CreateImage(0, 200, 200)
4  If StartDrawing(ImageOutput(0))
5      *Buffer = DrawingBuffer()
6      Pitch   = DrawingBufferPitch()
7
8      RandomData(*Buffer, Pitch*200)
9
10     StopDrawing()
11 EndIf
12
13 OpenWindow(0, 0, 0, 200, 200, "Random Image",
14     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
15 ImageGadget(0, 0, 0, 200, 200, ImageID(0))
16
17 Repeat
18     Until WaitWindowEvent() = #PB_Event_CloseWindow
```

See Also

RandomSeed() , Random() , CryptRandomData()

137.26 RandomSeed

Syntax

```
RandomSeed(Value)
```

Description

Changes the random number seed for the values returned with Random() and RandomData() .

Parameters

Value The new seed for the random number generator.

Return value

None.

Remarks

Each time a program is started, a new seed is generated, therefore RandomSeed() is useful only when the goal is to generate the same random numbers in the same order every time the program is executed.

See Also

Random() , RandomData() , RandomizeArray() , RandomizeList()

137.27 Round

Syntax

`Result.f(.d) = Round(Number.f(.d), Mode)`

Description

Round the specified float number according to the given mode.

Parameters

Number.f or Number.d The number to round.

Mode The rounding mode. This can be one of the following values:

```
#PB_Round_Down    : rounds down the number
#PB_Round_Up      : rounds up the number
#PB_Round_Nearest : rounds to the nearest integer number (0.5
                    and up will round up)
```

Remarks

To turn a floating-point number into an integer without rounding, use `Int()` or `IntQ()`. This function handles and returns float or double values.

Example

```
1  Debug Round(11.6, #PB_Round_Down)    ; Will print '11.0'
2  Debug Round(-3.6, #PB_Round_Down)    ; Will print '-4.0'
3
4  Debug Round(11.6, #PB_Round_Up)      ; Will print '12.0'
5  Debug Round(-1.6, #PB_Round_Up)      ; Will print '-1.0'
6
7  Debug Round(11.6, #PB_Round_Nearest) ; Will print '12.0'
8  Debug Round(11.4, #PB_Round_Nearest) ; Will print '11.0'
9  Debug Round(11.5, #PB_Round_Nearest) ; Will print '12.0'
10 Debug Round(-7.5, #PB_Round_Nearest) ; Will print '-8.0'
```

See Also

`Int()` , `IntQ()`

137.28 Sign

Syntax

`Result = Sign(Number.f(.d))`

Description

Returns an integer value representing the sign of the given number.

Parameters

Number.f or Number.d The number to get the sign from.

Return value

- Returns 0 if Number is zero.
- Returns 1 if Number is positive.
- Returns -1 if Number is negative.

Remarks

This function handles float or double values but returns an integer.

Example

```
1  Debug Sign(-7)    ; will display -1
2  Debug Sign(0)     ; will display  0
3  Debug Sign(7)     ; will display  1
```

See Also

Abs()

137.29 Sin

Syntax

`Result.f(.d) = Sin(Angle.f(.d))`

Description

Returns the sine of the specified angle.

Parameters

Angle.f or Angle.d The input angle in radian. The Radian() function can be used to transform an angle from degrees to radian.

Return value

Returns the sine of the angle. The result will be between -1.0 and 1.0.

Remarks

The inverse function of Sin() is ASin() . This function handles and returns float or double values.

Example

```
1  Debug Sin(1.5708)
```

See Also

ASin() , SinH() , Radian()

137.30 SinH

Syntax

```
Result.f(.d) = SinH(Angle.f(.d))
```

Description

Returns the hyperbolic sine of the specified hyperbolic angle.

Parameters

Angle.f or **Angle.d** The input angle.

Return value

Returns the hyperbolic sine of the angle.

Remarks

The inverse function of SinH() is ASinH() . This function handles and returns float or double values.

Example

```
1  Debug SinH(Log(1.618033)) ; Will be approximately 0.5
```

See Also

ASinH() , Sin()

137.31 Sqr

Syntax

```
Result.f(.d) = Sqr(Number.f(.d))
```

Description

Returns the square root of the specified number.

Parameters

Number.f or **Number.d** The input number. This must be a positive value.

Return value

Returns the square root of the number.

Remarks

This function handles and returns float or double values.

See Also

Pow()

137.32 Tan

Syntax

```
Result.f(.d) = Tan(Angle.f(.d))
```

Description

Returns the tangent of the specified angle.

Parameters

Angle.f or Angle.d The input angle in radian. The Radian() function can be used to transform an angle from degrees to radian.

Return value

Returns the tangent of the specified angle.

Remarks

The inverse function of Tan() is ATan() . This function handles and returns float or double values.

Example

```
1  Debug Tan(0.785398) ; will be approximately '1.0'
```

See Also

ATan() , ATan2() , TanH() , Radian()

137.33 TanH

Syntax

```
Result.f(.d) = TanH(Angle.f(.d))
```

Description

Returns the hyperbolic tangent of the specified hyperbolic angle.

Parameters

Angle.f or Angle.d The input angle.

Return value

Returns the hyperbolic tangent of the angle.

Remarks

The inverse function of TanH() is ATanH() . This function handles and returns float or double values.

Example

```
1  Debug TanH(Log(1.618033)) ; will display '0.447213...' (1/5 *  
    Sqr(5))
```

See Also

ATanH() , Tan()

Chapter 138

Memory

Overview

Sometimes it's very useful have raw access to the system memory (RAM) to process some time consuming operations and speed them up. This library allows to allocate any number of memory buffers and to use them directly within PureBasic.

Note: Direct memory manipulation must be handled with care. Accessing memory areas outside of allocated buffers will cause the program to crash.

138.1 AllocateMemory

Syntax

```
*MemoryID = AllocateMemory(Size [, Flags])
```

Description

Allocates a contiguous memory area with the specified size in bytes. The new memory area will be cleared and filled with zeros.

Parameters

Size The size in bytes for the new memory area.

Flags (optional) It can be one of the following values:

```
#PB_Memory_NoClear: don't fill the new memory area with  
                    zeros. It can help to have faster allocation if the  
                    allocated memory is used immediately.
```

Return value

Returns the address of the allocated memory, or zero if the memory cannot be allocated.

Remarks

FreeMemory() can be used to return the allocated memory back to the system.

ReAllocateMemory() can be used to change the size of the allocated area. All the allocated memory areas are automatically freed when the programs ends.

If the program crashes at this command, it is usually a result of a memory corruption at an earlier time in the program by writing at an area outside of the allocated memory area. Such an error can be narrowed down to the real cause using the purifier debugger tool.

Example

```
1  *MemoryID = AllocateMemory(5000)
2  If *MemoryID
3      Debug "Starting address of the 5000 Byte memory area:"
4      Debug *MemoryID
5      PokeS(*MemoryID, "Store this string in the memory area")
6      FreeMemory(*MemoryID) ; will also be done automatically at the
    end of program
7  Else
8      Debug "Couldn't allocate the requested memory!"
9  EndIf
```

See Also

ReAllocateMemory() , FreeMemory() , MemorySize()

138.2 AllocateStructure

Syntax

```
*Item.StructureName = AllocateStructure(StructureName)
```

Description

Allocates a new dynamic structure item. This dynamic structure item is properly initialized and ready to use, without the need to call InitializeStructure() . To access the structure data, a pointer associated with the specified 'StructureName' has to be used.

Parameters

StructureName The name of the structure used to create the new dynamic item. The structure has to be already created.

Return value

The address of the new dynamic structure item, zero otherwise.

Remarks

This command is for advanced users and shouldn't be needed for most programs. It's often a better choice to use a structured array , list or map to store dynamic structured items.

FreeStructure() can be used to free the dynamic structure item. All dynamic structures are automatically freed when the programs ends.

If the program crashes at this command, it is usually a result of a memory corruption at an earlier time in the program by writing at an area outside of the allocated memory area. Such an error can be narrowed down to the real cause using the purifier debugger tool.

Example

```
1  Structure People
2      Name$
3      List Friends$()
4  EndStructure
```

```

5
6  *DynamicPeople.People = AllocateStructure(People)
7  *DynamicPeople\Name$ = "Fred"
8  AddElement(*DynamicPeople\Friends$())
9  *DynamicPeople\Friends$() = "Stef"
10
11  Debug *DynamicPeople\Name$
12  Debug *DynamicPeople\Friends$()
13
14  FreeStructure(*DynamicPeople)

```

See Also

FreeStructure()

138.3 CompareMemory

Syntax

```
Result = CompareMemory(*MemoryID1, *MemoryID2, Size)
```

Description

Compares the content of two memory areas.

Parameters

***MemoryID1, *MemoryID2** The addresses of the two memory areas to compare.

Size The amount of bytes to compare.

Return value

Returns nonzero if the two areas contain the same bytes or zero if the content does not match.

See Also

AllocateMemory() , CompareMemoryString() , MemorySize()

138.4 CompareMemoryString

Syntax

```
Result = CompareMemoryString(*String1, *String2 [, Mode [, Length
    [, Flags]])
```

Description

Compare two strings at the specified memory addresses.

Parameters

***String1, *String2** The addresses of the strings to compare.

Mode (optional) Character comparison mode. This can be one of the following values:

```
#PB_String_CaseSensitive : Case sensitive search (a=a).  
                          (default)  
#PB_String_NoCase       : Case insensitive search using all  
                          unicode character set (Ã = Ã ).  
#PB_String_NoCaseAscii  : Case insensitive search using only  
                          ascii character set (a=A).  
                          Much faster than #PB_String_NoCase  
                          when full unicode support is not required.
```

Length (optional) The number of characters to compare. If this parameter is not specified or has the value -1, the strings are compared until a null-character is reached. If the strings are not null-terminated, this parameter must be specified.

Flags (optional) The string format to use when comparing the strings. This can be one of the following values:

```
#PB_Ascii   : Compares the strings as ascii  
#PB_UTF8    : Compares the strings as UTF8  
#PB_Unicode : Compares the strings as unicode (default)
```

Return value

Returns one of the following values:

```
#PB_String_Equal : if String1 equals String2  
#PB_String_Lower : if String1 is lower than String2  
#PB_String_Greater : if String1 greater than String2
```

See Also

PokeS() , PeekS() , MemoryStringLength() , CopyMemoryString() , CompareMemory() , MemorySize()

138.5 CopyMemory

Syntax

```
CopyMemory(*SourceMemoryID , *DestinationMemoryID , Size)
```

Description

Copy a memory area starting from the *SourceMemoryID to the *DestinationMemoryID.

Parameters

***SourceMemoryID** The address where the bytes are copied from.

***DestinationMemoryID** The address where the bytes are copied to.

Size The amount of bytes to copy.

Return value

None.

Remarks

The source and destination buffers may not overlap. To copy memory to a destination address that overlaps with the source buffer, use `MoveMemory()` .

See Also

`MoveMemory()` , `CopyMemoryString()` , `AllocateMemory()` , `MemorySize()`

138.6 CopyMemoryString

Syntax

```
Result = CopyMemoryString(*String [, @*DestinationMemoryID])
```

Description

Copy the string from the specified address to the destination memory address if specified, or at the end of the previous buffer if omitted.

Parameters

***String** The address of the string to copy. The string must be terminated with a null-character. The string is expected to be in the PB string format.

@*DestinationMemoryID (optional) The pointer to a variable holding the address with the destination buffer. After the string has been copied, the variable `*DestinationMemoryID` will point to the null-character at the end of the copied string, so a further call to this function will append the new string to the previous one.
If this parameter is omitted, the address from the previous call is used.

Return value

Returns the value of `*DestinationMemoryID` after the string was copied.

Example

```
1  *Buffer = AllocateMemory(1000)
2  *Pointer = *Buffer
3  CopyMemoryString("Hello", @*Pointer)
4  CopyMemoryString(" World") ; This one will be put just after
   "Hello"
5  *Pointer-4                ; Come back from 2 characters unicode
   (on the 'l' of 'World')
6  CopyMemoryString("LD")    ; Finally the last two letters will
   be uppercase
7  Debug PeekS(*Buffer)
```

See Also

`CopyMemory()` , `PeekS()` , `PokeS()`

138.7 FillMemory

Syntax

```
FillMemory(*Memory, Size [, Value [, Type]])
```

Description

Fills the memory area with the specified value by repeatedly writing that value.

Parameters

***Memory** The address of the memory area to fill.

Size The size in bytes of the memory area to fill.

Value (optional) The value to write into the memory area. The default is the value 0.

Type (optional) The type for the value. This can be one of the following constants:

```
#PB_Byte      : Fills the memory using a byte (1 byte) value  
                (default).  
#PB_Ascii     : Fills the memory using a byte (1 byte) value.  
#PB_Word      : Fills the memory using a word (2 bytes) value.  
#PB_Unicode   : Fills the memory using a word (2 bytes) value.  
#PB_Character : Fills the memory using a character (2 bytes in  
                unicode  
            ).  
#PB_Long      : Fills the memory using a long (4 bytes) value.  
#PB_Integer   : Fills the memory using an integer value (4  
                bytes in 32-bit executable, 8 bytes in 64-bit executable).
```

Return value

None.

Example

```
1  *Buffer = AllocateMemory(500)  
2  
3  FillMemory(*Buffer, 500) ; Fill 500 bytes of value 0 (clear the  
    memory area)  
4  FillMemory(*Buffer, 500, $FF) ; Fill 500 bytes of value $FF  
5  FillMemory(*Buffer, 500, $BADFOOD, #PB_Long) ; Fill 500 bytes of  
    value $BADFOOD
```

See Also

AllocateMemory() , MemorySize()

138.8 FreeMemory

Syntax

```
FreeMemory(*MemoryID)
```

Description

Free the memory previously allocated with `AllocateMemory()` , `ReAllocateMemory()` , `Ascii()` or `UTF8()` .

Parameters

***MemoryID** The address of the memory area to free. This must be a value returned from either `AllocateMemory()` or `ReAllocateMemory()` or `Ascii()` or `UTF8()` .

Return value

None.

Remarks

If the program crashes at this command even though the input seems correct, it is usually a result of a memory corruption at an earlier time in the program by writing at an area outside of the allocated memory area. Such an error can be narrowed down to the real cause using the purifier debugger tool.

All remaining allocated memory blocks are automatically freed when the program ends.

See Also

`AllocateMemory()` , `ReAllocateMemory()` , `Ascii()` , `UTF8()`

138.9 FreeStructure

Syntax

```
FreeStructure(*Item)
```

Description

Free the dynamic structure item previously allocated with `AllocateStructure()` . There is no need to call `ClearStructure()` before freeing the structure.

Parameters

***Item** The address of the dynamic structure item to free. This must be a value returned from `AllocateStructure()` .

Return value

None.

Remarks

If the program crashes at this command even though the input seems correct, it is usually a result of a memory corruption at an earlier time in the program by writing at an area outside of the allocated memory area. Such an error can be narrowed down to the real cause using the purifier debugger tool.

All remaining allocated dynamic structure items are automatically freed when the program ends.

See Also

AllocateStructure()

138.10 MemorySize

Syntax

```
Result = MemorySize(*MemoryID)
```

Description

Returns the length of the given memory area.

Parameters

***MemoryID** The address of the memory area to get the size from. This must be a value returned from either AllocateMemory() or ReAllocateMemory() or Ascii() or UTF8() .

Return value

Returns the size of the given memory area in bytes.

See Also

AllocateMemory() , ReAllocateMemory() , FreeMemory() , Ascii() , UTF8()

138.11 MemoryStringLength

Syntax

```
Result = MemoryStringLength(*String [, Flags])
```

Description

Returns the length (in characters) of the given zero terminated string.

Parameters

***String** The address of the string to get the length from.

Flags (optional) The string format to use. This can be one of the following values:

```
#PB_Ascii    : Reads the strings as ascii.
#PB_UTF8     : Reads the strings as UTF8
#PB_Unicode  : Reads the strings as unicode (by default, see
               unicode
               mode).
```

Combined with one of the following value:

```
#PB_ByteLength: only valid when using the #PB_UTF8 flag, the
               result will represent bytes (not characters).
               It can be useful as UTF8 has variable
               character lengths.
```

Return value

Returns the length of the string in characters.

See Also

PokeS() , PeekS() , AllocateMemory()

138.12 MoveMemory

Syntax

```
MoveMemory(*SourceMemoryID, *DestinationMemoryID, Size)
```

Description

Copy a memory area starting from the *SourceMemoryID to the *DestinationMemoryID. Overlapping of the two memory areas is allowed.

Parameters

***SourceMemoryID** The address where the bytes are copied from.

***DestinationMemoryID** The address where the bytes are copied to.

Size The amount of bytes to copy.

Return value

None.

Remarks

This command can be slower than CopyMemory() , but it ensures that the bytes are copied correctly, even if the two memory areas overlap.

See Also

CopyMemory() , AllocateMemory() , MemorySize()

138.13 ReAllocateMemory

Syntax

```
*NewMemoryID = ReAllocateMemory(*MemoryID, Size [, Flags])
```

Description

Resizes the given memory buffer to a new size. The memory may be copied to a new location in the process if there is not enough memory available at its current location.

Parameters

***MemoryID** The address of the memory area to resize. This value must be the result of a call to `AllocateMemory()` or `ReAllocateMemory()` .
If this parameter is `#Null`, the command acts like `AllocateMemory()` and allocates a new memory area of the given size.

Size The size in bytes for the resized or allocated buffer.

Flags (optional) It can be one of the following values:

`#PB_Memory_NoClear`: don't fill the extended memory area with zeros. It can help to have faster allocation if the extended memory is used immediately. If the memory size is reduced, this flag has no effect.

Return value

Returns the new address of the memory area if it could be resized. In this case, the old '*MemoryID' address can no longer be used. If resizing the memory area failed (because there is not enough memory available), the result is zero, and the '*MemoryID' address is still valid with the existing memory area and the old size.

Remarks

If the size of the memory area is increased, any new bytes are initially filled with zeros unless the `#PB_Memory_NoClear` flag is specified.

If the program crashes at this command even though the input seems correct, it is usually a result of a memory corruption at an earlier time in the program by writing at an area outside of the allocated memory area. Such an error can be narrowed down to the real cause using the purifier debugger tool.

All remaining allocated memory blocks are automatically freed when the program ends.

Example

```
1  *MemoryID = AllocateMemory(1000)
2  PokeS(*MemoryID, "Store this string")
3  ; do something more with it here...
4  ;
5  *NewMemoryID = ReAllocateMemory(*MemoryID, 2000) ; need more
    memory
6  If *NewMemoryID
7      ; work with *NewMemoryID now with size 2000
8      Debug "The old contents are still here:"
9      Debug PeekS(*NewMemoryID)
10     ;
11     FreeMemory(*NewMemoryID)
12 Else
13     ; resizing failed, keep working with *MemoryID (size 1000)
14     ;
15     FreeMemory(*MemoryID)
16 EndIf
```

See Also

`AllocateMemory()` , `FreeMemory()` , `MemorySize()`

138.14 PeekA

Syntax

```
Value.a = PeekA(*MemoryBuffer)
```

Description

Reads an ascii character (1 byte) from the specified memory address.

Parameters

***MemoryBuffer** The address to read from.

Return value

Returns the value of the ascii character.

See Also

PokeA()

138.15 PeekB

Syntax

```
Value.b = PeekB(*MemoryBuffer)
```

Description

Reads a byte (1 byte) number from the specified memory address.

Parameters

***MemoryBuffer** The address to read from.

Return value

Returns the value of the byte.

See Also

PokeB()

138.16 PeekC

Syntax

```
Value.c = PeekC(*MemoryBuffer)
```

Description

Reads a character (2 bytes in unicode) number from the specified memory address.

Parameters

***MemoryBuffer** The address to read from.

Return value

Returns the value of the character.

See Also

PokeC()

138.17 PeekD

Syntax

```
Value.d = PeekD(*MemoryBuffer)
```

Description

Reads a double (8 bytes) number from the specified memory address.

Parameters

***MemoryBuffer** The address to read from.

Return value

Returns the value of the double.

See Also

PokeD()

138.18 PeekI

Syntax

```
Value.i = PeekI(*MemoryBuffer)
```

Description

Reads an integer (4 bytes in 32-bit executable, 8 bytes in 64-bit executable) number from the specified memory address.

Parameters

***MemoryBuffer** The address to read from.

Return value

Returns the value of the integer.

See Also

PokeI()

138.19 PeekL

Syntax

```
Value.l = PeekL(*MemoryBuffer)
```

Description

Reads a long (4 bytes) number from the specified memory address.

Parameters

***MemoryBuffer** The address to read from.

Return value

Returns the value of the long.

See Also

PokeL()

138.20 PeekW

Syntax

```
Value.w = PeekW(*MemoryBuffer)
```

Description

Reads a word (2 bytes) number from the specified memory address.

Parameters

***MemoryBuffer** The address to read from.

Return value

Returns the value of the word.

See Also

PokeW()

138.21 PeekF

Syntax

```
Value.f = PeekF(*MemoryBuffer)
```

Description

Reads a float (4 bytes) from the specified memory address.

Parameters

***MemoryBuffer** The address to read from.

Return value

Returns the value of the float.

See Also

PokeF()

138.22 PeekQ

Syntax

```
Value.q = PeekQ(*MemoryBuffer)
```

Description

Reads a quad (8 bytes) number from the specified memory address.

Parameters

***MemoryBuffer** The address to read from.

Return value

Returns the value of the quad.

See Also

PokeQ()

138.23 PeekS

Syntax

```
Text\[extract_itex] = PeekS(*MemoryBuffer [, Length [, Format]])
```

Description

Reads a string from the specified memory address.

Parameters

***MemoryBuffer** The address to read from.

Length (optional) The maximum number of characters to read. If this parameter is not specified or -1 is used then there is no maximum. The string is read until a terminating null-character is encountered or the maximum length is reached.

Format (optional) The string format to use when reading the string. This can be one of the following values:

```
#PB_Ascii    : Reads the strings as ascii.
#PB_UTF8     : Reads the strings as UTF8
#PB_Unicode  : Reads the strings as unicode (default).
```

Combined with one of the following value:

```
#PB_ByteLength: only valid when using the #PB_UTF8 flag, the
'Length' will represent bytes (not characters).
                It can be useful as UTF8 has variable
                character lengths.
```

Return value

Returns the read string.

See Also

PokeS() , MemoryStringLength() , CompareMemoryString() , CopyMemoryString()

138.24 PeekU

Syntax

```
Value.u = PeekU(*MemoryBuffer)
```

Description

Reads an unicode character (2 bytes) from the specified memory address.

Parameters

***MemoryBuffer** The address to read from.

Return value

Returns the value of the unicode character.

See Also

PokeU()

138.25 PokeA

Syntax

```
PokeA(*MemoryBuffer, Number)
```

Description

Writes an ascii character (1 byte) to the specified memory address.

Parameters

***MemoryBuffer** The address to write to.

Number The value to write.

Return value

None.

See Also

PeekA()

138.26 PokeB

Syntax

```
PokeB(*MemoryBuffer, Number)
```

Description

Writes a byte (1 byte) number to the specified memory address.

Parameters

***MemoryBuffer** The address to write to.

Number The value to write.

Return value

None.

See Also

PeekB()

138.27 PokeC

Syntax

```
PokeC(*MemoryBuffer, Number)
```

Description

Writes a character (2 bytes in unicode) number to the specified memory address.

Parameters

***MemoryBuffer** The address to write to.

Number The value to write.

Return value

None.

See Also

PeekC()

138.28 PokeD

Syntax

```
PokeD(*MemoryBuffer, Number)
```

Description

Writes a double (8 bytes) number to the specified memory address.

Parameters

***MemoryBuffer** The address to write to.

Number The value to write.

Return value

None.

See Also

PeekD()

138.29 PokeI

Syntax

```
PokeI(*MemoryBuffer, Number)
```

Description

Writes an integer (4 bytes in 32-bit executable, 8 bytes in 64-bit executable) number to the specified memory address.

Parameters

***MemoryBuffer** The address to write to.

Number The value to write.

Return value

None.

See Also

PeekI()

138.30 PokeL

Syntax

```
PokeL(*MemoryBuffer , Number)
```

Description

Writes a long (4 bytes) number to the specified memory address.

Parameters

***MemoryBuffer** The address to write to.

Number The value to write.

Return value

None.

See Also

PeekL()

138.31 PokeQ

Syntax

```
PokeQ(*MemoryBuffer , Number)
```

Description

Writes a quad (8 bytes) number to the specified memory address.

Parameters

***MemoryBuffer** The address to write to.

Number The value to write.

Return value

None.

See Also

PeekQ()

138.32 PokeW

Syntax

```
PokeW(*MemoryBuffer, Number)
```

Description

Writes a word (2 bytes) number to the specified memory address.

Parameters

***MemoryBuffer** The address to write to.

Number The value to write.

Return value

None.

See Also

PeekW()

138.33 PokeF

Syntax

```
PokeF(*MemoryBuffer, Number.f)
```

Description

Writes a float (4 bytes) to the specified memory address.

Parameters

***MemoryBuffer** The address to write to.

Number The value to write.

Return value

None.

See Also

PeekF()

138.34 PokeS

Syntax

```
Result = PokeS(*MemoryBuffer, Text$[, Length[, Flags]])
```

Description

Writes a string to the specified memory address, followed by a null-character for termination.

Parameters

***MemoryBuffer** The address to write to.

Text\$ The string to write.

Length (optional) The maximum number of characters to write. If this parameter is not specified or -1 is used then the full length is written. The terminating null-character that is always written (unless the `#PB_String_NoZero` flag is set) is not included in this count.

Flags (optional) The string format to use when writing the string. This can be one of the following values:

```
#PB_Ascii    : Writes the strings in ascii
#PB_UTF8     : Writes the strings in UTF8
#PB_Unicode  : Writes the strings in unicode (default)
```

It can be combined with the following constants:

```
#PB_String_NoZero: Doesn't write the terminating
null-character.
```

Return value

The amount of bytes written to memory, not including the terminating null-character. The amount of written bytes differs from the string length in characters if the format is `#PB_UTF8` or `#PB_Unicode`.

See Also

`PeekS()` , `CopyMemoryString()`

138.35 PokeU

Syntax

```
PokeU(*MemoryBuffer, Number)
```

Description

Writes an unicode character (2 bytes) to the specified memory address.

Parameters

***MemoryBuffer** The address to write to.

Number The value to write.

Return value

None.

See Also

`PeekU()`

Chapter 139

Menu

Overview

Creating and managing menus in PureBasic is simple. You can, of course, tailor the menus to your specific needs.

To use a menu you must first start by creating one with either `CreateMenu()` for normal menus, or `CreatePopupMenu()` for pop-up menus.

macOS:

On OSX an application menu is never attached to a window but always to the desktop. The menu at the top of the desktop shows the items from the application that has focus.

There are predefined menu events `#PB_Menu_Quit`, `#PB_Menu_About` and `#PB_Menu_Preferences` to represent the entries in the application menu which is present on every macOS program. Their values are negative to not conflict with any menu entries defined in the program. They are reported from `EventMenu()` as regular menu events.

139.1 CloseSubMenu

Syntax

```
CloseSubMenu()
```

Description

Close the current sub menu and return to the enclosing menu after a previous call to `OpenSubMenu()` .

Parameters

None.

Return value

None.

Remarks

For an example and a preview image see `OpenSubMenu()` .

See Also

`OpenSubMenu()`

139.2 CreateMenu

Syntax

```
Result = CreateMenu(#Menu, WindowID)
```

Description

Creates a new empty menu on the given window.

Parameters

#Menu A number to identify the new menu. **#PB_Any** can be used to auto-generate this number.

WindowID The window for the new menu. It can be obtained using the WindowID() function.

Return value

Nonzero if the menu was created successfully, zero otherwise. If **#PB_Any** was used for the **#Menu** parameter then the generated number is returned on success.

Remarks

To create a menu with support for images, use CreateImageMenu() .

Once created, this menu becomes the current menu for further item additions. It's now possible to use functions such as MenuItem() , MenuBar() , OpenSubMenu() to populate the menu.

To handle menu events properly, see the description of following functions:

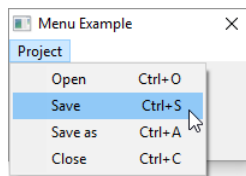
WaitWindowEvent() (alternatively WindowEvent())

EventWindow()

EventMenu()

Example

```
1  If OpenWindow(0, 200, 200, 200, 100, "Menu Example")
2      If CreateMenu(0, WindowID(0))          ; menu creation starts....
3          MenuTitle("Project")
4              MenuItem(1, "Open"      +Chr(9)+"Ctrl+O")
5              MenuItem(2, "Save"      +Chr(9)+"Ctrl+S")
6              MenuItem(3, "Save as" +Chr(9)+"Ctrl+A")
7              MenuItem(4, "Close"    +Chr(9)+"Ctrl+C")
8      EndIf
9      Repeat : Until WaitWindowEvent()=#PB_Event_CloseWindow
10  EndIf
```



See Also

CreateImageMenu() , CreatePopupMenu() , CreatePopupMenu() , FreeMenu() , MenuItem() , MenuTitle() , MenuItem() , MenuBar() , OpenSubMenu()

139.3 CreateImageMenu

Syntax

```
Result = CreateImageMenu(#Menu, WindowID [, Flags])
```

Description

Creates a new empty menu on the given window with support for images in the menu items.

Parameters

#Menu A number to identify the new menu. **#PB_Any** can be used to auto-generate this number.

WindowID The window for the new menu. It can be obtained using the WindowID() function.

Flags (optional) None.

Return value

Nonzero if the menu was successfully created, zero otherwise. If **#PB_Any** was used for the **#Menu** parameter then the generated number is returned on success.

Remarks

Once created, this menu becomes the current menu for further item additions. It's now possible to use functions such as MenuTitle() , MenuItem() , MenuBar() , OpenSubMenu() to populate the menu.

To handle menu events properly, see the description of following functions:

WaitWindowEvent() (alternatively WindowEvent())

EventWindow()

EventMenu()

Example

```
1  UsePNGImageDecoder()
2
3  If OpenWindow(0, 200, 200, 200, 100, "Image menu Example")
4      If CreateImageMenu(0, WindowID(0)) ; menu creation starts....
5          MenuTitle("Project")
6          MenuItem(1, "Open" +Chr(9)+"Ctrl+O", LoadImage(0,
7              #PB_Compiler_Home + "examples/sources/Data/ToolBar/Open.png"))
8          MenuItem(2, "Save" +Chr(9)+"Ctrl+S", LoadImage(1,
9              #PB_Compiler_Home + "examples/sources/Data/ToolBar/Save.png"))
10         MenuBar()
11         MenuItem(3, "Quit" +Chr(9)+"Ctrl+Q")
12     EndIf
13     Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
14     EndIf
```

See Also

CreateMenu() , CreatePopupMenu() , CreatePopupMenuImage() , FreeMenu() , MenuTitle() , MenuItem() , MenuBar() , OpenSubMenu()

139.4 CreatePopupMenu

Syntax

```
Result = CreatePopupMenu(#Menu)
```

Description

Creates a new empty popup menu.

Parameters

#Menu A number to identify the new menu. **#PB_Any** can be used to auto-generate this number.

Return value

Nonzero if the menu was successfully created, zero otherwise. If **#PB_Any** was used for the **#Menu** parameter then the generated number is returned on success.

Remarks

To create a popup menu with support for images, use `CreatePopupMenuImage()`. Once created, this menu becomes the current menu for further item additions. It's now possible to use functions such as `MenuItem()`, `MenuBar()`, `OpenSubMenu()` to populate the menu.

`DisplayPopupMenu()` can be used to display this popup menu at any position on the screen.

To handle menu events properly, see the description of following functions:

`WaitWindowEvent()` (alternatively `WindowEvent()`)

`EventWindow()`

`EventMenu()`

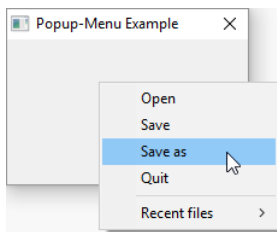
Example

```
1  If OpenWindow(0, 200, 200, 200, 120, "Popup-Menu Example")
2
3      If CreatePopupMenu(0)                ; creation of the pop-up menu
4          MenuItem(1, "Open")              ; You can use all commands for
          begins...                          creating a menu
5          MenuItem(2, "Save")              ; just like in a normal menu...
6          MenuItem(3, "Save as")
7          MenuItem(4, "Quit")
8          MenuBar()
9          OpenSubMenu("Recent files")
10         MenuItem(5, "PureBasic.exe")
11         MenuItem(6, "Test.txt")
12         CloseSubMenu()
13     EndIf
14
15     Repeat
16         Event = WaitWindowEvent()          ; check for window events
17
18         Select Event
19             Case #PB_Event_RightClick      ; right mouse button was
                clicked =>
```

```

20         DisplayPopupMenu(0, WindowID(0)) ; now display the
        popup-menu
21
22         Case #PB_Event_Menu ; an item of the popup-menu was
        clicked
23         Select EventMenu() ; get the clicked menu item...
24             Case 1 : Debug "Menu: Open"
25             Case 2 : Debug "Menu: Save"
26             Case 3 : Debug "Menu: Save as"
27             Case 4 : End
28             Case 5 : Debug "Menu: PureBasic.exe"
29             Case 6 : Debug "Menu: Text.txt"
30         EndSelect
31
32     EndSelect
33
34     Until Event = #PB_Event_CloseWindow
35 EndIf

```



See Also

CreatePopupMenu(), DisplayPopupMenu(), CreateMenu(), CreateImageMenu(), FreeMenu(), MenuTitle(), MenuItem(), MenuBar(), OpenSubMenu()

139.5 CreatePopupMenu

Syntax

```
Result = CreatePopupMenu(#Menu [, Flags])
```

Description

Creates a new empty popup menu with image support for its items.

Parameters

#Menu A number to identify the new menu. #PB_Any can be used to auto-generate this number.

Flags (optional) None.

Return value

Returns nonzero if the menu was created successfully and zero if not. If #PB_Any was used for the #Menu parameter, then the generated number is returned on success.

Remarks

Once created, this menu becomes the current menu for further item additions. It's now possible to use functions such as `MenuTitle()` , `MenuItem()` , `MenuBar()` , `OpenSubMenu()` to populate the menu.

`DisplayPopupMenu()` can be used to display this popup menu at any position on the screen.

To handle menu events properly, see the description of following functions:

`WaitWindowEvent()` (alternatively `WindowEvent()`)

`EventWindow()`

`EventMenu()`

Example

```
1  UsePNGImageDecoder()
2
3  If CreateImage(0,16,16,32)
4      StartDrawing(ImageOutput(0))
5      Box(0,0,15,15,RGB(255,255,128))
6      DrawRotatedText(-5,3, "+", 45, RGB(255,0,128))
7      StopDrawing()
8  EndIf
9
10 OpenWindow(0, 200, 200, 200, 120, "Image Popup-Menu Example")
11
12 If LoadImage(1,#PB_Compiler_Home +
13     "examples/sources/Data/ToolBar/Open.png") ; load an icon
14     If CreatePopupMenu(0) ; creation of the pop-up menu
15         begins...
16         MenuItem(1, "Open", ImageID(1)) ; Display the icon
17         MenuItem(2, "Save" ; just like in a normal menu...
18         MenuItem(3, "Save as")
19         MenuItem(4, "Quit", ImageID(0)) ; Display our own icon
20         MenuBar()
21         OpenSubMenu("Recent files")
22             MenuItem(5, "PureBasic.exe")
23             MenuItem(6, "Test.txt")
24         CloseSubMenu()
25     EndIf
26
27 Repeat
28     Event = WaitWindowEvent() ; check for window events
29
30     Select Event
31         Case #PB_Event_RightClick ; right mouse button was clicked
32             =>
33             DisplayPopupMenu(0, WindowID(0)) ; now display the
34             popup-menu
35
36         Case #PB_Event_Menu ; an item of the popup-menu was
37         clicked
38             Select EventMenu() ; get the clicked menu item...
39                 Case 1 : Debug "Menu: Open"
40                 Case 2 : Debug "Menu: Save"
41                 Case 3 : Debug "Menu: Save as"
42                 Case 4 : End
43                 Case 5 : Debug "Menu: PureBasic.exe"
44                 Case 6 : Debug "Menu: Text.txt"
45             EndSelect
46
47     Until Event = #PB_Event_CloseWindow
```

```

42         EndSelect
43     Until Event = #PB_Event_CloseWindow
44 EndIf

```

See Also

CreatePopupMenu() , DisplayPopupMenu() , CreateMenu() , CreateImageMenu() , FreeMenu() , MenuTitle() , MenuItem() , MenuBar() , OpenSubMenu() , SysTrayIconMenu()

139.6 DisplayPopupMenu

Syntax

```
DisplayPopupMenu(#Menu, WindowID [, x, y])
```

Description

Displays a PopupMenu under the current mouse position or at the given screen location.

Parameters

#Menu The menu to display. It must have been created with CreatePopupMenu() or CreatePopupImageMenu() .

WindowID The window with which to associate the popup menu. This value can be obtained with the WindowID() function.

x, y (optional) The location at which the menu should be displayed in screen coordinates. These are coordinates in pixels relative to the upper-left corner of the primary monitor. If this parameter is not specified, the menu is displayed at the current mouse position.

Return value

None.

Remarks

The popup menu will be hidden again when the user selects an item or clicks somewhere outside of the area of the popup menu.

For an example and a preview image see the CreatePopupMenu() help.

See Also

CreatePopupMenu() , CreatePopupImageMenu()

139.7 DisableMenuItem

Syntax

```
DisableMenuItem(#Menu, MenuItem, State)
```

Description

Disable (or enable) a menu item in the given menu.

Parameters

#Menu The menu to use.

MenuItem The number of the menu item to disable or enable.

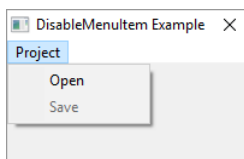
State The new state for the menu item. A value of 1 disables the menu item and a value of 0 enables it.

Return value

None.

Example

```
1  If OpenWindow(0, 200, 200, 200, 100, "DisableMenuItem Example")
2      If CreateMenu(0, WindowID(0))
3          MenuTitle("Project")
4          MenuItem(1, "Open")
5          MenuItem(2, "Save")
6          DisableMenuItem(0, 2, 1)      ; disable the second menu item
      (Save)
7      EndIf
8
9      Repeat
10         Until WaitWindowEvent() = #PB_Event_CloseWindow
11     EndIf
```



See Also

MenuItem() , SetMenuItemState() , SetMenuItemText()

139.8 FreeMenu

Syntax

```
FreeMenu(#Menu)
```

Description

Frees the specified menu and all its resources.

Parameters

#Menu The menu to free. If **#PB_All** is specified, all the remaining menus are freed.

Return value

None.

Remarks

All remaining menus are automatically freed when the program ends.

See Also

CreateMenu() , CreateImageMenu() , CreatePopupMenu() , CreatePopupMenuImageMenu()

139.9 GetMenuItemState

Syntax

```
Result = GetMenuItemState(#Menu, MenuItem)
```

Description

Returns the checkbox state of a menu item.

Parameters

#Menu The menu to use.

MenuItem The menu item number to get the state of.

Return value

Returns nonzero if the menu item is checked and zero otherwise.

Remarks

Use SetMenuItemState() to change the state of a menu item.

Example

```
1  If OpenWindow(0, 200, 200, 200, 100, "GetMenuItemState Example")
2      If CreateMenu(0, WindowID(0))
3          MenuTitle("Project")
4          MenuItem(1, "Changed")
5          SetMenuItemState(0, 1, 1)          ; set check mark for the
previously created menu item
6      EndIf
7      Repeat
8          Event = WaitWindowEvent()          ; wait for an event
9          If Event = #PB_Event_Menu          ; a menu event appeared
10             If EventMenu() = 1              ; first menu item was
clicked
11                 If GetMenuItemState(0, 1) = 1 ; actual item state =
check marked
12                     SetMenuItemState(0, 1, 0) ; now remove the check mark
13                 Else                          ; actual item state = no
check mark
14                     SetMenuItemState(0, 1, 1) ; now set the check mark
15                 EndIf
16             EndIf
17         EndIf
18     Until Event = #PB_Event_CloseWindow
19 EndIf
```

See Also

SetMenuItemState() , GetMenuItemText() , MenuItem()

139.10 GetMenuItemText

Syntax

```
Text\$\$ = GetMenuItemText(#Menu, Item)
```

Description

Returns the text from the specified menu item.

Parameters

#Menu The menu to use.

Item The item to get the text from.

Return value

Returns the menu item text.

See Also

SetMenuItemText() , MenuItem()

139.11 GetMenuTitleText

Syntax

```
Text\$\$ = GetMenuTitleText(#Menu, Title)
```

Description

Returns the title text of the specified menu title item.

Parameters

#Menu The menu to use.

Title The index of the menu title item to read the title from.

Return value

Returns the text of the menu title item.

See Also

MenuTitle() , SetMenuTitleText()

139.12 HideMenu

Syntax

```
HideMenu(#Menu, State)
```

Description

Hide or show the specified menu.

Parameters

#Menu The menu to hide or show.

State The new state for the menu. A value of 1 hides the menu and a value of 0 shows it.

Return value

None.

See Also

CreateMenu() , CreateImageMenu()

Supported OS

Windows, Linux

139.13 IsMenu

Syntax

```
Result = IsMenu(#Menu)
```

Description

Tests if the given menu is valid and correctly initialized.

Parameters

#Menu The menu to test.

Return value

Nonzero if the menu is valid, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a menu is ready to use.

See Also

CreateMenu() , CreatePopupMenu() , CreateImageMenu() , CreatePopupMenu()

139.14 MenuBar

Syntax

`MenuBar()`

Description

Creates a separator bar in the current menu.

Parameters

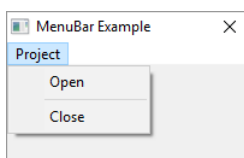
None.

Return value

None.

Example

```
1  If OpenWindow(0, 200, 200, 200, 100, "MenuBar Example")
2      If CreateMenu(0, WindowID(0)) ; here the menu creating
        starts....
3          MenuItem("Project")
4          MenuItem(1, "Open")
5          MenuBar() ; here the separator bar will be
        inserted
6          MenuItem(4, "Close")
7      EndIf
8      Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
9  EndIf
```



See Also

`MenuItem()` , `OpenSubMenu()` , `CreateMenu()` , `CreatePopupMenu()`

139.15 MenuHeight

Syntax

`Result = MenuHeight()`

Description

Returns the height of the menu title bar. This allows the correct height of a window to be calculated when using a menu.

Parameters

None.

Return value

Returns the height in pixels of the menu bar.

Remarks

Linux & macOS: This command will always returns 0, as the menu bar isn't part of the window (it is always located on the main bar at the very top of the screen). Therefore, MenuHeight() can be used seamlessly on every OS to adjust the window size depending on the actual menu height.

139.16 MenuItem

Syntax

```
MenuItem(MenuItemID, Text$ [, ImageID])
```

Description

Creates a new item on the current menu.

Parameters

MenuItemID A number to identify this menu item in events and commands like SetMenuItemState(). This value should be between 0 and 65535.

Text\$ The text for the menu item. On Windows you can use the special '&' character to underline a specific letter:
"&File" will actually display: File

ImageID (optional) The image to be displayed next to the menu item. The menu must be created with CreateImageMenu() or CreatePopupImageMenu() for the image to be displayed. This value can be obtained using the ImageID() function.

Return value

None.

Remarks

The dimensions of the images are 16x16 pixels.

To have a keyboard shortcut (will be activated with the AddKeyboardShortcut() function, except on macOS) aligned to the right side of the menu (for example, "Save Ctrl+S") use the tab character to give the correct space. The tab character has an ASCII code of 9 so use the function Chr() with the number 9 to insert a tab character. The code may look something like this:

```
1 MenuItem(1, "&Open" + Chr(9) + "Ctrl+O")
```

The supported modifiers are:

```
1 - "Ctrl" : Control key
2 - "Shift" : Shift key
3 - "Alt" : Alt key
4 - "Cmd" : Command/Apple key (OS X only)
```

They can be mixed together with the "+" character: "Save As" + Chr(9) + "Ctrl+Shift+S". On OS X, when a shortcut is created in the menu, there is no need to call the AddKeyboardShortcut() function with this shortcut.

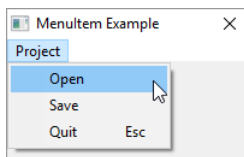
macOS: the 'Quit', 'Preferences' and 'About' items are considered as specials and need to be placed in the 'Application' menu to have the look'n'feel of macOS applications. PureBasic supports the #PB_Menu_Quit, #PB_Menu_Preferences and #PB_Menu_About constants (to be specified as the 'MenuItemID') for these kind of menu items. When one of these constants is detected, the item isn't inserted in the current menu, but in the 'Application' menu. If a shortcut was specified, it is simply ignored and replaced by the standard one. These 3 constants aren't defined on others OS, to allow flexible numbering on these OS.

Example

```

1  If OpenWindow(0, 200, 200, 200, 100, "MenuItem Example")
2      If CreateMenu(0, WindowID(0))
3          MenuTitle("Project")
4              MenuItem(1, "Open")          ; normal item
5              MenuItem(2, "&Save")          ; item with underlined character,
              the underline will only
6              ; be displayed, if menu is called
              with F10 + arrow keys
7              MenuItem(3, "Quit"+Chr(9)+"Esc") ; item with separate
              shortcut text
8          EndIf
9          Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
10             EndIf

```



See Also

MenuTitle(), MenuBar(), OpenSubMenu()

139.17 MenuID

Syntax

```
MenuID = MenuID(#Menu)
```

Description

Returns the unique system identifier of the given menu.

Parameters

#Menu The menu to use.

Return value

Returns the ID of the menu. This sometimes also known as 'Handle'. Look at the extra chapter "Handles and Numbers" for more information.

See Also

CreateMenu() , CreatePopupMenu() , CreateImageMenu() , CreatePopupMenuImageMenu()

139.18 MenuItem

Syntax

```
MenuItem(Title$)
```

Description

Creates a new title item on the menu.

Parameters

Title\$ The text to display in the title item. On Windows you can use the special '&' character to underline a specific letter, if the graphic theme allows it:
"&File" will actually display: File

Return value

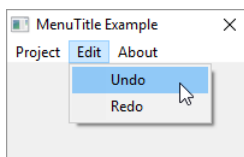
None.

Remarks

MenuItem() can't be used in popup menus , as there are no menu titles.

Example

```
1  If OpenWindow(0, 200, 200, 200, 100, "MenuItem Example")
2      If CreateMenu(0, WindowID(0))
3          MenuItem("Project")          ; normal menu title with following
            item
4              MenuItem(1, "Open")
5              MenuItem(2, "Close")
6              MenuItem("&Edit")          ; menu title with underlined
            character, the underline
7                                  ; will only be displayed, when
            called with F10 key
8              MenuItem(3, "Undo")
9              MenuItem(4, "Redo")
10             MenuItem("About")          ; only menu title
11         EndIf
12         Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
13     EndIf
```



See Also

MenuItem() , MenuBar() , OpenSubMenu()

139.19 OpenSubMenu

Syntax

```
OpenSubMenu(Text$ [, ImageID])
```

Description

Creates an empty submenu in the current menu.

Parameters

Text\$ The text for the submenu.

Windows:

In the Text\$ argument, you can use the special '&' character to underline a specific letter: "&File" will actually display: File

ImageID (optional) An optional image to display next to the submenu. This parameter only has an effect if the current menu was created using the CreateImageMenu() or CreatePopupImageMenu() command. This value can be obtained using the ImageID() function.

Return value

None.

Remarks

It is not possible to rename an OpenSubMenu easily except with Windows which returns a menu number.

Ex:

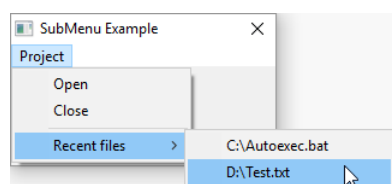
```
SubMenu = OpenSubMenu ("New")
```

```
SetMenuItemText (0, SubMenu, "Open")
```

With Linux and macOS you will have to juggle with several menus or destroy and recreate it.

Example

```
1  If OpenWindow(0, 200, 200, 220, 100, "SubMenu Example")
2    If CreateMenu(0, WindowID(0))
3      MenuTitle("Project")
4      MenuItem(1, "Open")
5      MenuItem(2, "Close")
6      MenuBar()
7      OpenSubMenu("Recent files")      ; begin submenu
8      MenuItem( 3, "C:\Autoexec.bat")
9      MenuItem( 4, "D:\Test.txt")
10     CloseSubMenu()                  ; end submenu
11   EndIf
12   Repeat : Until WaitWindowEvent()=#PB_Event_CloseWindow
13   EndIf
```



See Also

CloseSubMenu() , MenuItem() , MenuBar()

139.20 SetMenuItemState

Syntax

```
SetMenuItemState(#Menu, MenuItem, State)
```

Description

Changes the specified MenuItem state. This functions allows you to display a 'check mark' next to the menu item text.

Parameters

#Menu The menu to use.

MenuItem The menu item to set the state for.

State The state to set. The check is not displayed when State equals 0, if State equals something else then the 'check mark' will be displayed.

Return value

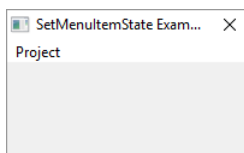
None.

Remarks

GetMenuItemState() can be used to retrieve the current menu item state.

Example

```
1  If OpenWindow(0, 200, 200, 200, 100, "SetMenuItemState Example")
2      If CreateMenu(0, WindowID(0))
3          MenuItem("Project")
4          MenuItem(1, "Changed")
5          SetMenuItemState(0, 1, 1)      ; set check mark for the
           previously created menu item
6      EndIf
7      Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
8  EndIf
```



See Also

GetMenuItemState()

139.21 SetMenuItemText

Syntax

```
SetMenuItemText(#Menu, Item, Text$)
```

Description

Changes the text of the specified menu item.

Parameters

#Menu The menu to use.

Item The item number of the item to change.

Text\$ The new text for the item.

Return value

None.

See Also

GetMenuItemText() , MenuItem()

139.22 SetMenuTitleText

Syntax

```
SetMenuTitleText(#Menu, Title, Text$)
```

Description

Changes the specified menu title item text.

Parameters

#Menu The menu to use.

Title The title item index to change.

Text\$ The new text for the title item.

Return value

None.

See Also

GetMenuTitleText() , MenuTitle()

139.23 BindMenuEvent

Syntax

```
BindMenuEvent(#Menu, MenuItem, @Callback())
```

Description

Bind a menu event to a callback. It's an additional way to handle events in PureBasic, which works without problem with the regulars WindowEvent() / WaitWindowEvent() commands. A menu event can be unbound with UnbindMenuEvent() .

Parameters

#Menu The menu to bind the event to.

MenuItem The menu item within the menu to bind the event to.

@Callback() The callback procedure to call when the event occurs. It has to be declared like this:

```
1  Procedure EventHandler()  
2      ; Code  
3  EndProcedure
```

Regular functions like EventGadget() , EventWindow() , EventMenu() , EventType() and EventData() are available within the callback to get more information about the event.
Note: WindowEvent() and WaitWindowEvent() should never be called from inside the callback or the program can be locked or have wrong behavior.

Return value

None.

Example

```
1  Procedure TestHandler()  
2      Debug "Test menu event"  
3  EndProcedure  
4  
5  Procedure QuitHandler()  
6      Debug "Quit menu event"  
7      End  
8  EndProcedure  
9  
10 OpenWindow(0, 100, 100, 200, 50, "Click test",  
11     #PB_Window_SystemMenu)  
12 CreateMenu(0, WindowID(0))  
13     MenuTitle("File")  
14     MenuItem(0, "Test")  
15     MenuItem(1, "Quit")  
16  
17 BindMenuEvent(0, 0, @TestHandler())  
18 BindMenuEvent(0, 1, @QuitHandler())  
19  
20 Repeat  
21     Event = WaitWindowEvent()  
22 Until Event = #PB_Event_CloseWindow
```

See Also

BindGadgetEvent() , BindMenuEvent() , UnbindEvent() , WindowEvent() , WaitWindowEvent()

139.24 UnbindMenuEvent

Syntax

```
UnbindMenuEvent(#Menu, MenuItem, @Callback())
```

Description

Unbind a menu event from a callback. If no matching event callback is found, this command has no effect.

Parameters

#Menu The menu to unbind the event.

MenuItem The menu item within the menu to unbind the event.

@Callback() The callback procedure to unbind.

Return value

None.

Example

```
1  Procedure TestHandler()  
2      Debug "Test menu event"  
3  EndProcedure  
4  
5  Procedure QuitHandler()  
6      Debug "Quit menu event"  
7      End  
8  EndProcedure  
9  
10 OpenWindow(0, 100, 100, 200, 50, "Click test",  
11     #PB_Window_SystemMenu)  
12 CreateMenu(0, WindowID(0))  
13     MenuTitle("File")  
14     MenuItem(0, "Test")  
15     MenuItem(1, "Quit")  
16  
17 BindMenuEvent(0, 0, @TestHandler())  
18 BindMenuEvent(0, 1, @QuitHandler())  
19  
20 UnbindMenuEvent(0, 1, @QuitHandler()) ; Unbind the quit event  
21  
22 Repeat  
23     Event = WaitWindowEvent()  
24 Until Event = #PB_Event_CloseWindow
```

See Also

`BindEvent()` , `BindGadgetEvent()` , `BindMenuEvent()` , `WindowEvent()` , `WaitWindowEvent()`

Chapter 140

Mesh

Overview

Meshes are 3D objects composed of many vertices (and triangles) which are linked together to make a shape. A mesh may have an optional skeleton , with bones to allow real-time animation. The skeleton can be either directly found in the mesh file or dynamically created. A mesh may not be moved or displayed directly, but requires to be manipulated through an entity . `InitEngine3D()` must be called successfully before using the Mesh functions.

140.1 CreateMesh

Syntax

```
Result = CreateMesh(#Mesh [, Type [, Mode])
```

Description

Creates a new empty #Mesh. After creation, the further commands of this library like `MeshVertexPosition()` or `MeshFace()` can be used to build it.

Parameters

#Mesh The number to identify the new mesh. `#PB_Any` can be used to auto-generate this number.

Type (optional) The type of the new mesh. It can be one of the following value:

```
#PB_Mesh_TriangleList : the mesh will be composed of a list
                        of triangles (default).
#PB_Mesh_TriangleStrip: the mesh will be composed of a list
                        of connected triangles (vertices are shared).
#PB_Mesh_TriangleFan  : the mesh will be composed of a list
                        of triangles sharing the same central vertex point.
#PB_Mesh_PointList    : the mesh will be composed of a list
                        of points.
#PB_Mesh_LineList      : the mesh will be composed of a list
                        of lines.
#PB_Mesh_LineStrip     : the mesh will be composed of a list
                        of connected lines (vertices are shared).
```

Mode (optional) The mode of the new mesh. It can be one of the following value:

```

#PB_Mesh_Static : once created, the mesh can't be modified
any more with the mesh update functions (default).
#PB_Mesh_Dynamic: once created, the mesh can be modified with
the mesh update functions.

```

Return value

Returns nonzero if the mesh was created successfully and zero if there was an error. If `#PB_Any` was used as the `#Mesh` parameter, then the new generated number is returned on success.

Remarks

If the `#Mesh` was already created, then it is freed and replaced by a new one.

140.2 CreateDataMesh

Syntax

```
Result = CreateDataMesh(#Mesh, Array.MeshVertex() [, Mode])
```

Description

Creates a new `#Mesh` from the specified 2 dimensional array of `MeshVertex` type. This command allows faster mesh creation than using `CreateMesh()` by preparing an array and submitting in one batch to the command.

Parameters

#Mesh The number to identify the new mesh. `#PB_Any` can be used to auto-generate this number.

Array A 2 dimensional array of type `MeshVertex`, which will be used to create the new mesh. The `MeshVertex` structure is defined as the following:

```

1  Structure MeshVertex
2      x.f
3      y.f
4      z.f
5      NormalX.f
6      NormalY.f
7      NormalZ.f
8      TangentX.f
9      TangentY.f
10     TangentZ.f
11     u.f
12     v.f
13     Color.l
14 EndStructure

```

Mode (optional) The way to link vertices. It can be one of the following value:

```

#PB_Mesh_DiagonalRegular1      : diagonals are aligned in the
same way
#PB_Mesh_DiagonalRegular2      : diagonals are aligned in the
other way
#PB_Mesh_DiagonalAlternate      : diagonals are once in a way
and once in the other way (alternate)

```

```
#PB_Mesh_DiagonalShortestLength: diagonals are between the
two closest vertices
#PB_Mesh_DiagonalClosestNormal : diagonals are between the
two closest normals (scalar product). Best mode but requires
normals.
```

Remarks

If the #Mesh was already created, then it is freed and replaced by a new one.

140.3 CopyMesh

Syntax

```
Result = CopyMesh(#Mesh, #NewMesh)
```

Description

Creates a #NewMesh which is the exact copy of the specified #Mesh. If #PB_Any is used as the '#NewMesh' parameter, then the new mesh number will be returned as 'Result'. Dynamic meshes are not supported for copy (meshes created with the #PB_Mesh_Dynamic flag).

If the Result equals 0, then the mesh copy has failed. If #NewMesh already exists, then it is freed and replaced by a new one.

140.4 FreeMesh

Syntax

```
FreeMesh(#Mesh)
```

Description

Free the specified #Mesh. All its associated memory is released and the object may not be used anymore.

Parameters

#Mesh The mesh to free. If #PB_All is specified, all the remaining meshes are freed.

Return value

None.

Remarks

All remaining meshes are automatically freed when the program ends.

140.5 IsMesh

Syntax

```
Result = IsMesh(#Mesh)
```

Description

Tests if the given #Mesh is valid and the mesh has been correctly initialized.

This function is bulletproof and may be used with any value. If Result equals zero then the given mesh has not been properly created or initialized. This is the correct way to ensure a mesh is ready to use.

140.6 LoadMesh

Syntax

```
Result = LoadMesh(#Mesh, Filename$)
```

Description

Loads a new mesh. Before loading a mesh, an archive must be specified with Add3DArchive() .

Parameters

#Mesh A number to identify the new mesh. #PB_Any can be used to auto-generate this number.

Filename\$ The filename of the mesh.

Return value

Nonzero if the mesh was successfully loaded, zero otherwise. If #PB_Any was used for the #Mesh parameter then the generated number is returned on success.

Remarks

Mesh is required to be in the OGRE .mesh format. A command-line tool based on [assimp](#) is available to convert many 3d formats into the OGRE mesh, including materials and animations. It can be downloaded here: [OgreAssimpConverter.zip](#) (Windows only). Some problems have been reported with shadow and converted meshes, if that happen use OgreMeshUpdater.exe on the newly converted mesh and it should fix it.

It is also possible to use existing exporter for Milkshape, Lightwave, Blender or 3DS Max. More information may be found on the [OGRE website](#).

See Also

FreeMesh()

140.7 MeshID

Syntax

```
Result = MeshID(#Mesh)
```

Description

Returns the unique MeshID of the #Mesh. The use of this function is required especially by the CreateEntity() function.

140.8 GetMeshData

Syntax

```
Result = GetMeshData(#Mesh, SubMesh, DataArray(), Flags,  
    FirstIndex, LastIndex)
```

Description

Get internal mesh data, like vertices, face etc. Dynamic meshes are not supported (meshes created with the #PB_Mesh_Dynamic flag).

Parameters

#Mesh The mesh to use.

SubMesh The submesh to get the data from. The first submesh index is 0 (main mesh).

DataArray() The array to receive the data. It has to be an array of type "MeshVertex" or "MeshFace" depending of the specified flags.

Flags Specifies which kind of data needs to be retrieved. It can be one of the following values:

```
#PB_Mesh_Vertex: DataArray() is an array of type "MeshVertex".  
#PB_Mesh_Face  : DataArray() is an array of type "MeshFace".
```

combined with:

```
#PB_Mesh_UVCoordinate : Get the UV coordinate information  
    (only for #PB_Mesh_Vertex flag)  
#PB_Mesh_Normal       : Get the normal information (only for  
    #PB_Mesh_Vertex flag)  
#PB_Mesh_Color        : Get the color information (only for  
    #PB_Mesh_Vertex flag)  
#PB_Mesh_Tangent      : Get the tangent information (only for  
    #PB_Mesh_Vertex flag)
```

The "MeshVertex" and "MeshFace" structures are defined like this:

```
Structure MeshVertex  
    x.f  
    y.f  
    z.f  
    NormalX.f ; only used if #PB_Mesh_Normal flag is set  
    NormalY.f ;  
    NormalZ.f ;  
    TangentX.f  
    TangentY.f  
    TangentZ.f
```

```

    u.f      ; only used if #PB_Mesh_UVCoordinate flag is set
    v.f      ;
    Color.l   ; only used if #PB_Mesh_Color flag is set
EndStructure

Structure MeshFace
    Index.l
EndStructure

```

FirstIndex, LastIndex First and last index to get the data from.

Return value

Returns nonzero on success and zero on failure. If success, `dataArray()` has been resized and contains the mesh information.

See Also

`SetMeshData()`

140.9 SetMeshData

Syntax

```

Result = SetMeshData(#Mesh, SubMesh, dataArray(), Flags,
    FirstIndex, LastIndex)

```

Description

Set internal mesh data, like vertices, face etc. Dynamic meshes are not supported (meshes created with the `#PB_Mesh_Dynamic` flag).

Parameters

#Mesh The mesh to use.

SubMesh The submesh to set the data to. The first submesh index is 0 (main mesh).

dataArray() The array containing the data to set. It has to be an array of type "MeshVertex" or "MeshFace" depending on the specified flags.

Flags Specifies which kind of data needs to be set. It can be one of the following values:

```

#PB_Mesh_Vertex: dataArray() is an array of type "MeshVertex".
#PB_Mesh_Face   : dataArray() is an array of type "MeshFace".

```

combined with:

```

#PB_Mesh_UVCoordinate : Set the UV coordinate information
    (only for #PB_Mesh_Vertex flag)
#PB_Mesh_Normal        : Set the normal information (only for
    #PB_Mesh_Vertex flag)
#PB_Mesh_Color         : Set the color information (only for
    #PB_Mesh_Vertex flag)

```

The "MeshVertex" and "MeshFace" structures are defined like this:

```

Structure MeshVertex
    x.f
    y.f
    z.f
    NormalX.f ; only used if #PB_Mesh_Normal flag is set
    NormalY.f ;
    NormalZ.f ;
    TangentX.f
    TangentY.f
    TangentZ.f
    u.f      ; only used if #PB_Mesh_UVCoordinate flag is set
    v.f      ;
    Color.l   ; only used if #PB_Mesh_Color flag is set
EndStructure

Structure MeshFace
    Index.l
EndStructure

```

FirstIndex, LastIndex First and last index to set the data to.

Return value

Returns nonzero on success and zero on failure.

See Also

GetMeshData()

140.10 BuildMeshShadowVolume

Syntax

```
BuildMeshShadowVolume(#Mesh)
```

Description

Create the shadow volume for the specified #Mesh. It is required if the mesh needs to cast a shadow. It should be done once the mesh creation is completely done, or the shadow will not match the mesh.

Parameters

#Mesh The mesh to act on.

Return value

None.

See Also

CreateMesh()

140.11 CreateLine3D

Syntax

```
Result = CreateLine3D(#Mesh, x, y, z, Color, x2, y2, z2, Color2)
```

Description

Create a new 3D line mesh. The line is a wireframe object which can be used to ease debugging. To change the line position, just create it again.

Parameters

#Mesh The number to identify the new mesh. **#PB_Any** can be used to auto-generate this number.

x, y, z The first point coordinate of the line, in world units.

Color The color to be used by the first point. **RGB()** can be used to get a valid color.

x2, y2, z2 The second point coordinate of the line, in world units.

Color2 The color to be used by the second point. If this color is different than the color set for the first point, then a gradient between this two colors will be created. **RGB()** can be used to get a valid color.

Return value

Returns nonzero if the mesh was created successfully and zero if there was an error. If **#PB_Any** was used as the **#Mesh** parameter then the new generated number is returned on success.

Example

```
1  InitEngine3D()
2  InitSprite()
3
4  OpenWindow(0, 0, 0, 640, 480, "Line3D example",
5      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
6  OpenWindowedScreen(WindowID(0), 0, 0, 640, 480, 0, 0, 0)
7
8  ; Light
9  CreateLight(#PB_Any, RGB(25, 25, 180), -5, 10, 5, #PB_Light_Point)
10
11 ; Camera
12 CreateCamera(0, 0, 0, 100, 100)
13 MoveCamera(0, 2, 1, 3, #PB_Absolute | #PB_Local)
14 CameraLookAt(0, 0, 0, 0)
15
16 ; Create the line and attach it to the scene
17 CreateLine3D(0, 0, 0, 0, RGB(255, 0, 0), 1, 1, 1, RGB(0, 0, 255))
18 CreateEntity(0, MeshID(0), #PB_Material_None)
19
20 Repeat
21     RenderWorld()
22     FlipBuffers()
23 Until WaitWindowEvent(1) = #PB_Event_CloseWindow
```

See Also

FreeMesh() , CreateSphere() , CreateMesh() , CreateCube() , CreatePlane() , CreateCylinder()

140.12 CreateCube

Syntax

```
Result = CreateCube(#Mesh, Size)
```

Description

Create a new cube mesh.

Parameters

#Mesh The number to identify the new mesh. **#PB_Any** can be used to auto-generate this number.

Size Size, in world unit, of the cube.

Return value

Returns nonzero if the mesh was created successfully and zero if there was an error. If **#PB_Any** was used as the **#Mesh** parameter, then the new generated number is returned on success.

Example

```
1  InitEngine3D()
2  InitSprite()
3
4  OpenWindow(0, 0, 0, 640, 480, "Cube example",
5      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
6  OpenWindowedScreen(WindowID(0), 0, 0, 640, 480, 0, 0, 0)
7
8  ; Light
9  CreateLight(#PB_Any, RGB(25, 25, 180), -5, 10, 5, #PB_Light_Point)
10
11 ; Camera
12 CreateCamera(0, 0, 0, 100, 100)
13 MoveCamera(0, 2, 1, 3, #PB_Absolute | #PB_Local)
14 CameraLookAt(0, 0, 0, 0)
15
16 ; Create the cube and attach it to the scene
17 CreateCube(0, 1)
18 CreateEntity(0, MeshID(0), #PB_Material_None)
19
20 Repeat
21     RenderWorld()
22     FlipBuffers()
23 Until WaitWindowEvent(1) = #PB_Event_CloseWindow
```

See Also

FreeMesh() , CreateSphere() , CreateMesh() , CreateCylinder() , CreatePlane() , CreateLine3D()

140.13 CreateSphere

Syntax

```
Result = CreateSphere(#Mesh, Radius.f [, NbSegments, NbRings])
```

Description

Create a new sphere mesh.

Parameters

#Mesh The number to identify the new mesh. **#PB_Any** can be used to auto-generate this number.

Radius Radius, in world unit, of the sphere.

NbSegments (optional) Number of segments to create the sphere (default: 16). The segments are the vertical lines of the sphere. The more segments, the more realistic the sphere will be, but it will also affect the rendering speed if too many of them are specified.

NbRings (optional) Number of rings to create the sphere (default: 16). The rings are the horizontal lines of the sphere. The more rings, the more realistic the sphere will be, but it will also affect the rendering speed if too many of them are specified.

Return value

Returns nonzero if the mesh was created successfully and zero if there was an error. If **#PB_Any** was used as the **#Mesh** parameter then the new generated number is returned on success.

Example

```
1  InitEngine3D()
2  InitSprite()
3
4  OpenWindow(0, 0, 0, 640, 480, "Sphere example",
5      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
6  OpenWindowedScreen(WindowID(0), 0, 0, 640, 480, 0, 0, 0)
7
8  ; Light
9  CreateLight(#PB_Any, RGB(25, 25, 180), -5, 10, 5, #PB_Light_Point)
10
11 ; Camera
12 CreateCamera(0, 0, 0, 100, 100)
13 MoveCamera(0, 2, 1, 3, #PB_Absolute | #PB_Local)
14 CameraLookAt(0, 0, 0, 0)
15
16 ; Create the sphere and attach it to the scene
17 CreateSphere(0, 1)
18 CreateEntity(0, MeshID(0), #PB_Material_None)
19
20 Repeat
21     RenderWorld()
22     FlipBuffers()
23 Until WaitWindowEvent(1) = #PB_Event_CloseWindow
```

See Also

FreeMesh() , CreateCylinder() , CreateMesh() , CreateCube() , CreatePlane() , CreateLine3D()

140.14 CreateTube

Syntax

```
Result = CreateTube(#Mesh, OuterRadius.f, InnerRadius.f, Height.f  
[, NbBaseSegments, NbHeightSegments])
```

Description

Create a new tube mesh.

Parameters

#Mesh The number to identify the new mesh. **#PB_Any** can be used to auto-generate this number.

OuterRadius Outer radius, in world unit, of the tube.

InnerRadius Inner radius, in world unit, of the tube.

Height Height, in world unit, of the tube.

NbBaseSegments (optional) Number of segments used for the base of the tube (default: 16).

NbHeightSegments (optional) Number of segments used for the height of the tube (default: 1).

Return value

Returns nonzero if the mesh was created successfully and zero if there was an error. If **#PB_Any** was used as the **#Mesh** parameter then the new generated number is returned on success.

Example

```
1  InitEngine3D()  
2  InitSprite()  
3  
4  OpenWindow(0, 0, 0, 640, 480, "Tube example",  
5      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
6  OpenWindowedScreen(WindowID(0), 0, 0, 640, 480, 0, 0, 0)  
7  
8  ; Light  
9  CreateLight(#PB_Any, RGB(25, 25, 180), -5, 10, 5, #PB_Light_Point)  
10  
11 ; Camera  
12 CreateCamera(0, 0, 0, 100, 100)  
13 MoveCamera(0, 2, 4, 3, #PB_Absolute | #PB_Local)  
14 CameraLookAt(0, 0, 0, 0)  
15  
16 ; Create the tube and attach it to the scene  
17 CreateTube(0, 0.5, 0.4, 2)  
18 CreateEntity(0, MeshID(0), #PB_Material_None)  
19  
20 Repeat  
21     RenderWorld()
```



```

21     FlipBuffers()
22     Until WaitWindowEvent(1) = #PB_Event_CloseWindow

```

See Also

FreeMesh() , CreateCylinder() , CreateMesh() , CreateCube() , CreatePlane() , CreateLine3D()

140.15 CreateTorus

Syntax

```
Result = CreateTorus(#Mesh, Radius.f, SectionRadius.f, Height.f [,
    NbSectionSegments, NbCircleSegments)
```

Description

Create a new torus mesh.

Parameters

#Mesh The number to identify the new mesh. **#PB_Any** can be used to auto-generate this number.

Radius Radius, in world unit, of the torus.

InnerRadius Radius of the section, in world unit, of the torus.

Height Height, in world unit, of the torus.

NbSectionSegments (optional) Number of segments used for the section of the torus (default: 16).

NbCircleSegments (optional) Number of segments used for the circle of the torus (default: 16).

Return value

Returns nonzero if the mesh was created successfully and zero if there was an error. If **#PB_Any** was used as the **#Mesh** parameter then the new generated number is returned on success.

Example

```

1  InitEngine3D()
2  InitSprite()
3
4  OpenWindow(0, 0, 0, 640, 480, "Torus example",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
5  OpenWindowedScreen(WindowID(0), 0, 0, 640, 480, 0, 0, 0)
6
7  ; Light
8  CreateLight(#PB_Any, RGB(25, 25, 180), -5, 10, 5, #PB_Light_Point)
9
10 ; Camera
11 CreateCamera(0, 0, 0, 100, 100)
12 MoveCamera(0, 2, 4, 3, #PB_Absolute | #PB_Local)
13 CameraLookAt(0, 0, 0, 0)
14
15 ; Create the torus and attach it to the scene

```

```

16 CreateTorus(0, 1, 0.3)
17 CreateEntity(0, MeshID(0), #PB_Material_None)
18
19 Repeat
20     RenderWorld()
21     FlipBuffers()
22 Until WaitWindowEvent(1) = #PB_Event_CloseWindow

```

See Also

FreeMesh(), CreateCylinder(), CreateMesh(), CreateCube(), CreatePlane(), CreateLine3D()

140.16 CreateCapsule

Syntax

```
Result = CreateCapsule(#Mesh, Radius.f, Height.f [, NbRings,
    NbSegments, NbHeightSegments])
```

Description

Create a new capsule mesh.

Parameters

#Mesh The number to identify the new mesh. **#PB_Any** can be used to auto-generate this number.

Radius Radius, in world unit, of the capsule.

Height Height, in world unit, of the capsule.

NbRings (optional) Number of rings used to create the capsule (default: 8).

NbSegments (optional) Number of segments used for the capsule (default: 16).

NbHeightSegments (optional) Number of height segments used for the capsule (default: 1).

Return value

Returns nonzero if the mesh was created successfully and zero if there was an error. If **#PB_Any** was used as the **#Mesh** parameter, then the new generated number is returned on success.

Example

```

1  InitEngine3D()
2  InitSprite()
3
4  OpenWindow(0, 0, 0, 640, 480, "Capsule example",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
5  OpenWindowedScreen(WindowID(0), 0, 0, 640, 480, 0, 0, 0)
6
7  ; Light
8  CreateLight(#PB_Any, RGB(25, 25, 180), -5, 10, 5, #PB_Light_Point)
9
10 ; Camera
11 CreateCamera(0, 0, 0, 100, 100)

```

```

12 MoveCamera(0, 2, 0, 5, #PB_Absolute | #PB_Local)
13 CameraLookAt(0, 0, 0, 0)
14
15 ; Create the capsule and attach it to the scene
16 CreateCapsule(0, 1, 1)
17 CreateEntity(0, MeshID(0), #PB_Material_None)
18
19 Repeat
20     RenderWorld()
21     FlipBuffers()
22 Until WaitWindowEvent(1) = #PB_Event_CloseWindow

```

See Also

FreeMesh() , CreateCylinder() , CreateMesh() , CreateCube() , CreatePlane() , CreateLine3D()

140.17 CreateIcoSphere

Syntax

```
Result = CreateIcoSphere(#Mesh, Radius.f [, Iterations])
```

Description

Create a new ico-sphere mesh.

Parameters

#Mesh The number to identify the new mesh. **#PB_Any** can be used to auto-generate this number.

Radius Radius, in world unit, of the ico-sphere.

Iterations (optional) Number of iterations used to create the ico-sphere (default: 2).

Return value

Returns nonzero if the mesh was created successfully and zero if there was an error. If **#PB_Any** was used as the **#Mesh** parameter, then the new generated number is returned on success.

Example

```

1  InitEngine3D()
2  InitSprite()
3
4  OpenWindow(0, 0, 0, 640, 480, "IcoSphere example",
5             #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
6  OpenWindowedScreen(WindowID(0), 0, 0, 640, 480, 0, 0, 0)
7
8  ; Light
9  CreateLight(#PB_Any, RGB(25, 25, 180), -5, 10, 5, #PB_Light_Point)
10
11 ; Camera
12 CreateCamera(0, 0, 0, 100, 100)
13 MoveCamera(0, 2, 0, 5, #PB_Absolute | #PB_Local)
14 CameraLookAt(0, 0, 0, 0)

```

```

14
15 ; Create the icosphere and attach it to the scene
16 CreateIcoSphere(0, 1)
17 CreateEntity(0, MeshID(0), #PB_Material_None)
18
19 Repeat
20     RenderWorld()
21     FlipBuffers()
22 Until WaitWindowEvent(1) = #PB_Event_CloseWindow

```

See Also

FreeMesh() , CreateCylinder() , CreateMesh() , CreateCube() , CreatePlane() , CreateLine3D()

140.18 CreateCone

Syntax

```
Result = CreateCone(#Mesh, Radius.f, Height.f [, NbBaseSegments,
NbHeightSegments])
```

Description

Create a new cone.

Parameters

#Mesh The number to identify the new mesh. **#PB_Any** can be used to auto-generate this number.

Radius Radius, in world unit, of the cone.

Height Height, in world unit, of the cone.

NbBaseSegments (optional) Number of segments used for the base of the cone (default: 16).

NbHeightSegments (optional) Number of segments used for the height of the cone (default: 1).

Return value

Returns nonzero if the mesh was created successfully and zero if there was an error. If **#PB_Any** was used as the **#Mesh** parameter, then the new generated number is returned on success.

Example

```

1  InitEngine3D()
2  InitSprite()
3
4  OpenWindow(0, 0, 0, 640, 480, "Cone example",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
5  OpenWindowedScreen(WindowID(0), 0, 0, 640, 480, 0, 0, 0)
6
7  ; Light
8  CreateLight(#PB_Any, RGB(25, 25, 180), -5, 10, 5, #PB_Light_Point)
9
10 ; Camera

```

```

11 CreateCamera(0, 0, 0, 100, 100)
12 MoveCamera(0, 2, 1, 3, #PB_Absolute | #PB_Local)
13 CameraLookAt(0, 0, 0, 0)
14
15 ; Create the cone and attach it to the scene
16 CreateCone(0, 0.5, 1)
17 CreateEntity(0, MeshID(0), #PB_Material_None)
18
19 Repeat
20     RenderWorld()
21     FlipBuffers()
22 Until WaitWindowEvent(1) = #PB_Event_CloseWindow

```

See Also

FreeMesh() , CreateSphere() , CreateCylinder() , CreateMesh() , CreateCube() , CreatePlane() , CreateLine3D()

140.19 CreateCylinder

Syntax

```
Result = CreateCylinder(#Mesh, Radius.f, Height.f [,
    NbBaseSegments, NbHeightSegments, CloseTop])
```

Description

Create a new cylinder mesh.

Parameters

#Mesh The number to identify the new mesh. #PB_Any can be used to auto-generate this number.

Radius Radius, in world unit, of the cylinder.

Height Height, in world unit, of the cylinder.

NbBaseSegments (optional) Number of segments used for the base of the cylinder (default: 16).

NbHeightSegments (optional) Number of segments used for the height of the cylinder (default: 1).

CloseTop (optional) Specify if the cylinder should be closed on top and bottom, or if it should be left open (like a pipe). Set it to #True (default) to create a closed cylinder or to #False else.

Return value

Returns nonzero if the mesh was created successfully and zero if there was an error. If #PB_Any was used as the #Mesh parameter then the new generated number is returned on success.

Example

```
1  InitEngine3D()
2  InitSprite()
3
4  OpenWindow(0, 0, 0, 640, 480, "Cylinder example",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
5  OpenWindowedScreen(WindowID(0), 0, 0, 640, 480, 0, 0, 0)
6
7  ; Light
8  CreateLight(#PB_Any, RGB(25, 25, 180), -5, 10, 5, #PB_Light_Point)
9
10 ; Camera
11 CreateCamera(0, 0, 0, 100, 100)
12 MoveCamera(0, 2, 1, 3, #PB_Absolute | #PB_Local)
13 CameraLookAt(0, 0, 0, 0)
14
15 ; Create the cylinder and attach it to the scene
16 CreateCylinder(0, 0.5, 1)
17 CreateEntity(0, MeshID(0), #PB_Material_None)
18
19 Repeat
20     RenderWorld()
21     FlipBuffers()
22 Until WaitWindowEvent(1) = #PB_Event_CloseWindow
```

See Also

FreeMesh() , CreateSphere() , CreateCone() , CreateMesh() , CreateCube() , CreatePlane() , CreateLine3D()

140.20 CreatePlane

Syntax

```
Result = CreatePlane(#Mesh, TileSizeX, TileSizeZ, TileCountX,
    TileCountZ, TextureRepeatCountX, TextureRepeatCountZ)
```

Description

Create a new plane mesh.

Parameters

#Mesh The number to identify the new mesh. #PB_Any can be used to auto-generate this number.

TileSizeX X size of the a single tile, in world unit, of the plane. A tile is the base component of a plane. A plane can be composed of lot of tiles to make it bigger and allow deformation.

TileSizeZ Z size of the a single tile, in world unit, of the plane. A tile is the base component of a plane. A plane can be composed of lot of tiles to make it bigger and allow deformation.

TileCountX Number of tiles used to create the X axis of the plane.

TileCountZ Number of tiles used to create the Z axis of the plane.

TextureRepeatCountX Number of time the texture applied to the plane will be repeated on the X axis. To apply the whole texture all over the X axis, just use 1.

TextureRepeatCountZ Number of time the texture applied to the plane will be repeated on the Z axis. To apply the whole texture all over the Z axis, just use 1.

Return value

Returns nonzero if the mesh was created successfully and zero if there was an error. If `#PB_Any` was used as the `#Mesh` parameter then the new generated number is returned on success.

Remarks

The total number of vertices is limited by the product "TileCountX" x "TileCountZ" which must not exceed 65100.

Example

```
1  InitEngine3D()
2  InitSprite()
3
4  OpenWindow(0, 0, 0, 640, 480, "Plane example",
5      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
6  OpenWindowedScreen(WindowID(0), 0, 0, 640, 480, 0, 0, 0)
7
8  ; Light
9  CreateLight(#PB_Any, RGB(25, 25, 180), -5, 10, 5, #PB_Light_Point)
10
11 ; Camera
12 CreateCamera(0, 0, 0, 100, 100)
13 MoveCamera(0, 2, 1, 3, #PB_Absolute | #PB_Local)
14 CameraLookAt(0, 0, 0, 0)
15
16 ; Create the plane and attach it to the scene
17 CreatePlane(0, 2, 2, 1, 1, 0, 0)
18 CreateEntity(0, MeshID(0), #PB_Material_None)
19
20 Repeat
21     RenderWorld()
22     FlipBuffers()
23 Until WaitWindowEvent(1) = #PB_Event_CloseWindow
```

See Also

`FreeMesh()` , `CreateSphere()` , `CreateMesh()` , `CreateCube()` , `CreatePlane()` , `CreateLine3D()`

140.21 MeshDirectAdd

Syntax

```
MeshDirectAdd(#Mesh, MeshVertexArray(), MeshFaceArray(), Type,
    MaterialID)
```

Description

Add a new submesh to the specified `#Mesh`. A mesh can have any number of submeshes. A submesh position is relative to the mesh position.

Parameters

MeshVertexArray() The array containing the submesh vertex data. It has to be an array of type "MeshVertex".

MeshFaceArray() The array containing the submesh face data. It has to be an array of type "MeshFace".

Type The type of the new submesh. It can be one of the following value:

```
#PB_Mesh_TriangleList : the submesh will be composed of a
list of triangles (default).
#PB_Mesh_TriangleStrip: the submesh will be composed of a
list of connected triangles (vertices are shared).
#PB_Mesh_TriangleFan  : the submesh will be composed of a
list of triangles sharing the same central vertex point.
#PB_Mesh_PointList    : the submesh will be composed of a
list of points.
#PB_Mesh_LineList     : the submesh will be composed of a
list of lines.
#PB_Mesh_LineStrip    : the submesh will be composed of a
list of connected lines (vertices are shared).
```

MaterialID The default material to use for the submesh. To get a valid 'MaterialID', use `MaterialID()` .

Return value

None.

See Also

`FreeMesh()` , `CreateMesh()` , `LoadMesh()`

140.22 AddSubMesh

Syntax

```
AddSubMesh([Type])
```

Description

Add a new submesh to the current mesh previously created with `CreateMesh()` . A mesh can have any number of submeshes. A submesh position is relative to the mesh position. Once a submesh is created, use the following commands to build it: `MeshVertexPosition()` , `MeshFace()` and `MeshIndex()` .

Parameters

Type (optional) The type of the new submesh. It can be one of the following value:

```
#PB_Mesh_TriangleList : the submesh will be composed of a
list of triangles (default).
#PB_Mesh_TriangleStrip: the submesh will be composed of a
list of connected triangles (vertices are shared).
#PB_Mesh_TriangleFan  : the submesh will be composed of a
list of triangles sharing the same central vertex point.
```


`#PB_Mesh_PointList` : the submesh will be composed of a list of points.
`#PB_Mesh_LineList` : the submesh will be composed of a list of lines.
`#PB_Mesh_LineStrip` : the submesh will be composed of a list of connected lines (vertices are `shared`).

Return value

None.

See Also

`FreeMesh()` , `CreateMesh()` , `MeshVertexPosition()` , `MeshFace()`

140.23 MeshIndexCount

Syntax

```
Result = MeshIndexCount(#Mesh [, SubMesh])
```

Description

Return the number of indexes in the mesh.

Parameters

Mesh The mesh to use.

SubMesh (optional) If specified, it will return the number of indexes in the specified submesh.
The first submesh index is 0 (main mesh).

Return value

Returns the number of indexes in the mesh, or zero if the mesh or the submesh doesn't exists.

See Also

`CreateMesh()` , `LoadMesh()` , `MeshVertexCount()`

140.24 MeshVertexCount

Syntax

```
Result = MeshVertexCount(#Mesh [, SubMesh])
```

Description

Return the number of vertices of the mesh.

Parameters

Mesh The mesh to use.

SubMesh (optional) If specified, it will return the number of vertices of the specified submesh.
The first submesh index is 0 (main mesh).

Return value

Returns the vertices number of the mesh, or zero if the mesh or the submesh doesn't exist.

See Also

CreateMesh() , LoadMesh() , MeshIndexCount()

140.25 UpdateMeshBoundingBox

Syntax

```
UpdateMeshBoundingBox(#Mesh)
```

Description

Update the bounding box of the mesh. If a mesh has been manually modified, its bounding box has to be recalculated, especially if the mesh is used for collisions. The bounding box is the smallest box which can contain the whole mesh.

Parameters

Mesh The mesh to use.

Return value

None.

See Also

CreateMesh()

140.26 UpdateMesh

Syntax

```
UpdateMesh(#Mesh, SubMesh)
```

Description

Start the mesh update, to modify in real time its vertices and other values. The mesh has to be created with the `#PB_Mesh_Dynamic` flag. Once the mesh modifications are finished, `FinishMesh()` needs to be called. The mesh can use the following commands to change their properties: `MeshIndex()` , `MeshFace()` , `MeshVertexPosition()` , `MeshVertexNormal()` , `MeshVertexTangent()` , `MeshVertexColor()` and `MeshVertexTextureCoordinate()` .

Parameters

Mesh The mesh to use.

SubMesh The submesh index to modify. The first submesh index is 0 (main mesh).

Return value

None.

See Also

CreateMesh() , MeshIndex() , MeshFace() , MeshVertexPosition() , MeshVertexNormal() , MeshVertexTangent() , MeshVertexColor() MeshVertexTextureCoordinate()

140.27 MeshIndex

Syntax

```
MeshIndex(Index)
```

Description

Add or update a single vertex in the mesh being created with CreateMesh() or updated with UpdateMesh() . It behaves like the command MeshFace() , but with an arbitrary number of vertices. When using the mode #PB_Mesh_LineList or #PB_Mesh_LineStrip, there are only two vertices per line, so MeshIndex() needs to be used in this case.

Parameters

Index The vertex index.

Return value

None.

See Also

UpdateMesh() , MeshIndex() , MeshFace() , MeshVertexPosition() , MeshVertexNormal() , MeshVertexTangent() , MeshVertexColor() MeshVertexTextureCoordinate()

140.28 MeshRadius

Syntax

```
Result = MeshRadius(Mesh)
```

Description

Returns the radius of the smallest sphere which can contain the mesh.

Parameters

Mesh The mesh to use.

Return value

Returns the radius the mesh.

See Also

CreateMesh()

140.29 MeshVertex

Syntax

```
MeshVertex(x, y, z, u.f, v.f, Color [, NormalX, NormalY, NormalZ])
```

Description

Add a vertex to the current mesh previously created with CreateMesh() . Specific attributes to the newly created vertex can be added with MeshVertexTangent() . To create a new face use MeshFace() .

Parameters

x, y, z The position of the new vertex.

u The u value. This value is the X position in the texture where the vertex should map. This value is usually between 0 and 1, where 0 is the texture X origin and 1 is the texture X end (see MeshVertexTextureCoordinate() for more information).

v The v value. This value is the Y position in the texture where the vertex should map. This value is usually between 0 and 1, where 0 is the texture Y origin and 1 is the texture Y end (see MeshVertexTextureCoordinate() for more information).

Color Color of the vertex. This color can be in RGB or RGBA format (see MeshVertexColor() for more information).

NormalX, NormalY, NormalZ The normal vector (see MeshVertexNormal() for more information).

Return value

None.

See Also

CreateMesh() , MeshFace() , MeshVertexNormal() , MeshVertexTangent() , MeshVertexColor() , MeshVertexTextureCoordinate()

140.30 MeshVertexPosition

Syntax

```
MeshVertexPosition(x, y, z)
```

Description

Add a new vertex to the current mesh previously created with `CreateMesh()` . To set specific attributes to the newly created vertex, use the following commands: `MeshVertexNormal()` , `MeshVertexTangent()` , `MeshVertexColor()` and `MeshVertexTextureCoordinate()` . If several attributes needs to be specified, `MeshVertex()` can be used instead. To create a new face use `MeshFace()` .

Parameters

x, y, z The position of the new vertex.

Return value

None.

See Also

`CreateMesh()` , `MeshFace()` , `MeshVertexNormal()` , `MeshVertexTangent()` , `MeshVertexColor()` , `MeshVertexTextureCoordinate()` , `MeshVertex()`

140.31 MeshVertexNormal

Syntax

```
MeshVertexNormal(x, y, z)
```

Description

Set normal information to the current vertex previously added with `MeshVertexPosition()` or `MeshVertex()` . The normal vector is used to calculate lightning on an object. To automatically computes the vector normal once the mesh is created, use `NormalizeMesh()` .

Parameters

x, y, z The normal vector.

Return value

None.

See Also

`CreateMesh()` , `MeshVertexPosition()` , `MeshVertexColor()` , `MeshVertexTextureCoordinate()` , `NormalizeMesh()` , `MeshVertex()`

140.32 MeshVertexTangent

Syntax

```
MeshVertexTangent(x, y, z)
```

Description

Set tangent information to the current vertex previously added with `MeshVertexPosition()` or `MeshVertex()` . The tangent vector is mainly used in shader scripts. To automatically compute the tangent vector once the mesh is created, use `BuildMeshTangents()` .

Parameters

x, y, z The tangent vector.

Return value

None.

See Also

`CreateMesh()` , `MeshVertexPosition()` , `MeshVertexNormal()` , `MeshVertexColor()` , `MeshVertexTextureCoordinate()` , `BuildMeshTangents()` , `MeshVertex()`

140.33 MeshVertexColor

Syntax

```
MeshVertexColor(Color)
```

Description

Set color information to the current vertex previously added with `MeshVertexPosition()` or `MeshVertex()` . To have any effect, the material associated to the mesh has to be defined with `SetMaterialColor(#Material, #PB_Material_AmbientColor, -1)` and `AmbientColor()` () set to a positive value. This function needs a vertex created before to apply color to it.

Parameters

Color Color of the vertex. This color can be in RGB or RGBA format.

Return value

None.

See Also

`CreateMesh()` , `MeshVertexPosition()` , `MeshVertexNormal()` , `MeshVertexTangent()` , `MeshVertexTextureCoordinate()` , `MeshVertex()`

140.34 MeshVertexTextureCoordinate

Syntax

```
MeshVertexTextureCoordinate(u.f [, v.f [, w.f]])
```

Description

Set UVW information to the current vertex previously added with `MeshVertexPosition()` or `MeshVertex()` . The UVW information is used to apply the texture on the mesh.

Parameters

- u** The u value. This value is the X position in the texture where the vertex should map. This value is usually between 0 and 1, where 0 is the texture X origin and 1 is the texture X end.
- v (optional)** The v value. This value is the Y position in the texture where the vertex should map. This value is usually between 0 and 1, where 0 is the texture Y origin and 1 is the texture Y end.
- w (optional)** The w value (for cubic textures). This value is the Z position in the texture where the vertex should map. This value is usually between 0 and 1, where 0 is the texture Z origin and 1 is the texture Z end.

Return value

None.

See Also

CreateMesh() , MeshVertexPosition() , MeshVertexNormal() , MeshVertexTangent() , MeshVertexColor() , MeshVertex()

140.35 MeshFace

Syntax

```
MeshFace(Vertex1, Vertex2, Vertex3 [, Vertex4])
```

Description

Add or update a face to the current mesh previously created with CreateMesh() . The specified vertices must exist. The first vertex index starts from 0. The created face is a triangle or a quad. MeshIndex() can be used if the number of vertices is more than four.

Parameters

- Vertex1** The first vertex index used to create the face.
- Vertex2** The second vertex index used to create the face.
- Vertex3** The third vertex index used to create the face.
- Vertex4 (optional)** The fourth vertex index used to create the face, which will result in a quad.

Return value

None.

See Also

CreateMesh() , MeshVertexPosition() , MeshVertex()

140.36 FinishMesh

Syntax

```
FinishMesh(StaticMesh)
```

Description

Finish the creation of the current mesh started with CreateMesh() .

Parameters

StaticMesh If set to `#True`, the mesh will be converted to a static mesh and will be not modifiable anymore. If set to `#False`, the mesh will still be modifiable with UpdateMesh() . Static meshes are faster to render than dynamic meshes.

Return value

None.

See Also

CreateMesh()

140.37 NormalizeMesh

Syntax

```
NormalizeMesh(#Mesh [, SubMesh])
```

Description

Normalize the mesh or the submesh. It will automatically compute the normal vector for all vertices of the specified mesh or submesh. Dynamic meshes are not supported (meshes created with the `#PB_Mesh_Dynamic` flag).

Parameters

Mesh The mesh to use.

SubMesh (optional) If specified, it will only normalize the submesh. The first submesh index is 0 (main mesh).

Return value

None.

See Also

CreateMesh() , MeshVertexNormal()

140.38 BuildMeshTangents

Syntax

```
BuildMeshTangents (#Mesh)
```

Description

Automatically computes the tangent vectors for all vertices of the specified mesh. Dynamic meshes are not supported (meshes created with the `#PB_Mesh_Dynamic` flag).

Parameters

Mesh The mesh to use.

Return value

None.

See Also

CreateMesh() , MeshVertexTangent()

140.39 AddMeshManualLOD

Syntax

```
AddMeshManualLOD (#Mesh , #MeshLOD , Distance.f)
```

Description

Add a new level of detail (LOD) to the mesh. The `#Mesh` will be replaced automatically with `#MeshLOD` (which is often a simplified version of `#Mesh`, with less details) when displayed above the specified distance from the camera. Several LOD mesh can be used for the same `#Mesh` depending of the distance.

Parameters

Mesh The mesh to use.

MeshLOD The mesh to use when the distance from the camera has been reached.

Distance The minimum distance from the camera where the `#MeshLOD` should be used instead of `#Mesh`.

Return value

None.

See Also

CreateMesh() , BuildMeshLOD()

140.40 BuildMeshLOD

Syntax

```
BuildMeshLOD(#Mesh [, NbLOD, Distance.f, ReductionValue.f])
```

Description

Build automatically one or several level of detail (LOD) for the mesh. The **#Mesh** will be replaced automatically with less complex mesh when displayed above the specified distance from the camera. If more precise LOD meshes are required, AddMeshManualLOD() can be used.

Parameters

Mesh The mesh to use.

NbLOD (optional) Number of needed LOD for this mesh. If this parameter isn't used, then the LODs will be automatically calculated.

Distance (optional) The minimum distance from the camera where the first LOD mesh will be used instead of **#Mesh**. For the next LOD meshes, the distance will be calculated using this formula: 'Distance / SqrF(1-ReductionValue)'.

ReductionValue (optional) The reduction to apply, between 0 (no reduction) and 1 (100% reduction).

Example: CreateMeshLodLevels(**#Mesh**, 3, 100, 0.75)

- The first reduction of the original mesh starts from 100 units of the camera, vertices number halved by 4 (75% reduction). - The second reduction of the original mesh starts from 200 units of the camera, vertices number halved by 16. - The third reduction of the original mesh starts from 400 units of the camera, vertices number halved by 64.

Return value

None.

See Also

CreateMesh() , AddMeshManualLOD()

140.41 SaveMesh

Syntax

```
SaveMesh(#Mesh , Filename$)
```

Description

Save the mesh. The saved mesh can be loaded back with the LoadMesh() command.

Parameters

Mesh The mesh to save.

Filename\$ The filename and path to the new mesh file. If the filename does not include a full path, it is interpreted relative to the current directory .

Return value

None.

See Also

CreateMesh() , LoadMesh()

140.42 SetMeshMaterial

Syntax

```
SetMeshMaterial(#Mesh, MaterialID [, SubMesh])
```

Description

Set the mesh default material.

Parameters

Mesh The mesh to use.

MaterialID The default material to use for the mesh. To get a valid 'MaterialID', use MaterialID() .

SubMesh (optional) If specified, the material will be applied only to the submesh. The first submesh index is 0 (main mesh).

Return value

None.

See Also

CreateMesh() , LoadMesh()

140.43 SubMeshCount

Syntax

```
Result = SubMeshCount(#Mesh)
```

Description

Returns the number of submeshes of the mesh.

Parameters

Mesh The mesh to use.

Return value

Returns the number of submeshes of the mesh.

See Also

CreateMesh() , LoadMesh() , AddSubMesh()

140.44 TransformMesh

Syntax

```
TransformMesh(#Mesh, x, y, z, ScaleX, ScaleY, ScaleZ, RotateX,  
             RotateY, RotateZ [, SubMesh])
```

Description

Transform the Mesh according to the given parameters. Dynamic meshes are not supported for transform (meshes created with the `#PB_Mesh_Dynamic` flag).

Parameters

Mesh The mesh to transform.

x, y, z New position of the mesh, relative to its node. If the submesh parameter is specified, it specifies the new position of the submesh relative to its parent.

ScaleX Applies a scale factor on the X axis to the mesh. If the submesh parameter is specified, the scale is applied the submesh.

ScaleY Applies a scale factor on the Y axis to the mesh. If the submesh parameter is specified, the scale is applied the submesh.

ScaleZ Applies a scale factor on the Z axis to the mesh. If the submesh parameter is specified, the scale is applied the submesh.

RotateX Applies a rotation, in degree, on the X axis to the mesh. If the submesh parameter is specified, the rotation is applied to the submesh.

RotateY Applies a rotation, in degree, on the Y axis to the mesh. If the submesh parameter is specified, the rotation is applied to the submesh.

RotateZ Applies a rotation, in degree, on the Z axis to the mesh. If the submesh parameter is specified, the rotation is applied to the submesh.

SubMesh (optional) If specified, the transformation will be applied only to the submesh. The first submesh index is 0 (main mesh).

Return value

None.

See Also

CreateMesh() , LoadMesh()

Chapter 141

Mouse

Overview

PureBasic provides full access to mice plugged into the computer. It supports standard mice with up to 3 buttons. This library is optimized and uses very low level functions especially for games. Do not use the functions of this library in a regular application, in this case carry out the mouse queries with WindowMouseX() , WindowMouseY() and EventType() . On Windows, DirectX is used.

141.1 InitMouse

Syntax

```
Result = InitMouse()
```

Description

Initializes the mouse environment for later use. You should call this function before any other functions in this library. If the result is zero, no mouse is available.

Parameters

None.

Return value

Nonzero if the mouse functions are available, zero otherwise. This function tries to open DirectX (v3.0 with NT4.0 compatibility or v7.0 otherwise); if it fails then DirectX is either not available or is too old to be used.

141.2 ExamineMouse

Syntax

```
Result = ExamineMouse()
```

Description

Updates the mouse state. This function should be used before MouseDeltaX() , MouseDeltaY() , MouseX() , MouseY() or MouseButton() .

Parameters

None.

Return value

Nonzero if the mouse state has been updated, zero otherwise.

Remarks

The first call to `ExamineMouse()` captures the mouse to the active Screen or 'WindowedScreen'. The mouse is available again on your whole system after calling `ReleaseMouse(#True)` or after the end of that program.

In fullscreen mode the command `FlipBuffers()` need to be called inside your event loop to handle all events properly.

See Also

`ExamineMouse()` , `MouseDeltaX()` , `MouseDeltaY()` , `MouseX()` , `MouseY()` , `MouseButton()`

141.3 MouseButton

Syntax

```
Result = MouseButton(Button)
```

Description

Returns zero if the specified button number is not pressed, otherwise the button is pressed. Any number of buttons can be pressed at the same time. `ExamineMouse()` must be called before this function to update the actual button's state.

Parameters

Button It can be one of the following constants:

```
#PB_MouseButton_Left   : Tests if the left mouse button is  
pressed  
#PB_MouseButton_Right  : Tests if the right mouse button is  
pressed  
#PB_MouseButton_Middle: Tests if the middle mouse button is  
pressed
```

Return value

Nonzero if the specified mouse button is pressed, zero otherwise.

See Also

`ExamineMouse()`

141.4 MouseDeltaX

Syntax

```
Result = MouseDeltaX()
```

Description

Returns the mouse 'x' movement (in pixels) since the last call of this function.

Parameters

None.

Return value

The mouse 'x' movement (in pixels) since the last call of this function. It can be either a negative or positive value, depending on whether or not the mouse has been moved to the left or the right since the last call. `ExamineMouse()` should be called before this function to update the actual mouse position.

See Also

`ExamineMouse()` , `MouseDeltaY()`

141.5 MouseDeltaY

Syntax

```
Result = MouseDeltaY()
```

Description

Returns the mouse 'y' movement (in pixels) since the last call of this function.

Parameters

None.

Return value

The mouse 'y' movement (in pixels) since the last call of this function. It can be either a negative or positive value, depending on whether or not the mouse has been moved up or down since the last call. `ExamineMouse()` should be called before this function to update the actual mouse position.

See Also

`ExamineMouse()` , `MouseDeltaX()`

141.6 MouseLocate

Syntax

```
MouseLocate(x, y)
```

Description

Changes the absolute position (in pixels) of the mouse in the current screen. This is useful when using MouseX() or MouseY() .

Parameters

x, y The new absolute position (in pixels) of the mouse in the current screen.

Return value

None.

See Also

ExamineMouse() , MouseX() , MouseY()

141.7 MouseWheel

Syntax

```
Result = MouseWheel()
```

Description

Returns the number of ticks the mouse wheel has moved since the last function call. ExamineMouse() should be called before this function to update the mouse status.

Parameters

None.

Return value

The number of ticks the mouse wheel has moved since the last function call. The value is positive if the wheel has moved forward and negative if the wheel has moved backwards.

See Also

ExamineMouse()

Supported OS

Windows

141.8 MouseX

Syntax

`Result = MouseX()`

Description

Returns the actual mouse 'x' position (in pixels) on the current screen. The position can be changed easily with the MouseLocate() function. ExamineMouse() should be called before this function to update the actual mouse position.

Parameters

None.

Return value

The actual mouse 'x' position (in pixels) on the current screen.

See Also

ExamineMouse() , MouseLocate() , MouseY()

141.9 MouseY

Syntax

`Result = MouseY()`

Description

Returns the actual mouse 'y' position (in pixels) on the current screen. The position can be changed easily with the MouseLocate() function. ExamineMouse() should be called before this function to update the actual mouse position.

Parameters

None.

Return value

The actual mouse 'y' position (in pixels) on the current screen.

See Also

ExamineMouse() , MouseLocate() , MouseX()

141.10 ReleaseMouse

Syntax

```
ReleaseMouse(State)
```

Description

Locks or releases the mouse to allow its use under standard OS. This is typically called after checking the result of `IsScreenActive()` function.

Parameters

```
State    #True   : the mouse is released.  
         #False  : the mouse is locked.
```

Return value

None.

See Also

`IsScreenActive()`

Chapter 142

Movie

Overview

PureBasic provides simple, yet powerful functions to integrate movie playback inside an application or game.

Windows: as it uses the DirectX technology (DirectShow), any kind of media can be played with this library: AVI, MPG, DivX, etc.

Linux and Raspberry: it uses libvlc, therefore the 'libvlc-dev' package needs to be installed to be able to use this library. All formats supported by vlc will be available.

MacOS X: it uses QuickTime technology, therefore any kind of media (depending of the installed plugins) can be played with this library: AVI, MPG, DivX, etc.

Note: on some OS, music files can also played by this library but it is not officially supported and somewhat broken. Better use the sound library for this.

142.1 FreeMovie

Syntax

```
FreeMovie(#Movie)
```

Description

Frees the specified movie and all its resources.

Parameters

#Movie The movie to free. If **#PB_All** is specified, all the remaining movies are freed.

Return value

None.

Remarks

All remaining movies are automatically freed when the program ends.

See Also

IsMovie() , LoadMovie()

142.2 InitMovie

Syntax

```
Result = InitMovie()
```

Description

Initialize the movie environment for later use. You must call this function before any other functions in this library.

Parameters

None.

Return value

Returns nonzero if initialization was successful. If zero is returned then the movie library cannot be used.

Remarks

This function tries to open DirectX (v3.0 with NT4.0 compatibility or v7.0 else); if it fails then DirectX is either not available or is too old be used.

See Also

LoadMovie()

142.3 IsMovie

Syntax

```
Result = IsMovie(#Movie)
```

Description

Tests if the given #Movie number is a valid and correctly initialized movie.

Parameters

#Movie The movie to use.

Return value

Returns nonzero if #Movie is a valid movie and zero otherwise.

Remarks

This function is bulletproof and can be used with any value. If the 'Result' is not zero then the object is valid and initialized, otherwise it will equal zero. This is the correct way to ensure a movie is ready to use.

See Also

LoadMovie()

142.4 LoadMovie

Syntax

```
Result = LoadMovie(#Movie, Filename$)
```

Description

Loads the specified movie file and prepares it for playback.

Parameters

#Movie A number to identify the loaded movie. **#PB_Any** can be used to auto-generate this number.

Filename\$ The file name of the movie.

Return value

Returns nonzero if the movie was loaded correctly and zero if loading the movie failed (format not supported or file not found). If **#PB_Any** was used for the **#Movie** parameter then the generated number is returned on success.

Remarks

InitMovie() must be called once before loading movies. Further information about the loaded movie can be read with the MovieInfo() , MovieLength() , MovieWidth() and MovieHeight() commands.

See Also

PlayMovie() , MovieInfo() , MovieLength() , MovieWidth() , MovieHeight() , FreeMovie() , InitMovie()

142.5 MovieAudio

Syntax

```
MovieAudio(#Movie, Volume, Balance)
```

Description

Control the audio stream of a movie. Volume and balance can be modified during playback. Changes occur immediately.

Parameters

#Movie The movie to use.

Volume The volume for the movie. This value can be between 0 and 100 (100 being the loudest).

Balance The balance for the movie. This value can be between -100 and 100 (-100 is full left, 0 is middle (normal mode) and 100 is full right).

Return value

None.

See Also

PlayMovie()

Supported OS

Windows, MacOS X

142.6 MovieHeight

Syntax

```
Height = MovieHeight(#Movie)
```

Description

Returns the height of the movie.

Parameters

#Movie The movie to use.

Return value

Returns the height of the movie in pixels. If the result equals -1, then no video stream is found (or isn't compatible), however the audio stream can still be played.

See Also

MovieWidth() , MovieLength() , MovieInfo()

142.7 MovieInfo

Syntax

```
Result = MovieInfo(#Movie , Flags)
```

Description

Returns additional information about the movie.

Parameters

#Movie The movie to use.

Flags The information to return. Supported values are:
0: return the number of frame per seconds (*1000).

Return value

Returns the value specified in the 'Flags' parameter.

See Also

MovieLength() , MovieWidth() , MovieHeight()

Supported OS

Windows, MacOS X

142.8 MovieLength

Syntax

```
Length = MovieLength(#Movie)
```

Description

Returns the length of the movie.

Parameters

#Movie The movie to use.

Return value

Returns the length of the movie in frames.

See Also

MovieInfo() , MovieWidth() , MovieHeight()

Supported OS

Windows, MacOS X

142.9 MovieSeek

Syntax

```
MovieSeek(#Movie , Frame.q)
```

Description

Change the movie's playback position to the given frame.

Parameters

#Movie The movie to use.

Frame The frame to be the new playback position.

Return value

None.

See Also

MovieStatus()

Supported OS

Windows, MacOS X

142.10 MovieStatus

Syntax

```
Result.q = MovieStatus(#Movie)
```

Description

Get the playback status of the movie.

Parameters

#Movie The movie to use.

Return value

Returns one of the following values:

-1: Movie is paused.

0: Movie is stopped

> 0: Movie is playing. The returned value is the current frame number displayed.

See Also

MovieSeek()

142.11 MovieWidth

Syntax

```
Width = MovieWidth(#Movie)
```

Description

Returns the width of the movie.

Parameters

#Movie The movie to use.

Return value

Returns the width of the movie in pixels. If the result equals -1, then no video stream is found (or isn't compatible), however the audio stream can still be played.

See Also

MovieWidth() , MovieLength() , MovieInfo()

142.12 PauseMovie

Syntax

```
PauseMovie(#Movie)
```

Description

Pauses the movie playback. Playback can be resumed using the ResumeMovie() function.

Parameters

#Movie The movie to pause.

Return value

None.

See Also

PlayMovie() , ResumeMovie() , StopMovie()

142.13 PlayMovie

Syntax

```
Result = PlayMovie(#Movie , WindowID)
```

Description

Start to play a movie previously loaded with LoadMovie() on the specified window.

Parameters

#Movie The movie to play.

WindowID The window to play the movie on. This value can be easily obtained by using the WindowID() function.

It's also possible to play a movie on a full screen, simply use the result of ScreenID() as 'WindowID'.

Return value

Returns nonzero if the movie started playing correctly and zero if there was an error.

Remarks

The function `ResizeMovie()` can be used to resize and move the movie on this window (to not use the full window area, for example).

See Also

`LoadMovie()` , `StopMovie()` , `MovieWidth()` , `MovieHeight()` , `MovieLength()` , `MovieInfo()`

142.14 `ResizeMovie`

Syntax

```
ResizeMovie(#Movie, x, y, Width, Height)
```

Description

Resize and move the movie display area on the movie window.

Parameters

#Movie The movie to resize.

x, y, Width, Height The new location and size of the movie in pixels.

Return value

None.

See Also

`PlayMovie()` , `MovieWidth()` , `MovieHeight()`

142.15 `ResumeMovie`

Syntax

```
ResumeMovie(#Movie)
```

Description

Continue to play the movie, after a `PauseMovie()` call.

Parameters

#Movie The movie to resume.

Return value

None.

See Also

`PauseMovie()` , `PlayMovie()` , `StopMovie()`

142.16 StopMovie

Syntax

```
StopMovie(#Movie)
```

Description

Stop playing the movie. If the movie is played again, it will restart from the beginning.

Parameters

#Movie The movie to stop.

Return value

None.

See Also

PlayMovie() , PauseMovie() , ResumeMovie()

Chapter 143

Music

Overview

PureBasic can replay standard music modules to have nice music background during a game or an application. The music modules are well known from Demo makers, as it's an efficient way to create music on a computer. The tools used to create the music modules are named 'Trackers' (ProTracker, FastTracker, Impulse Tracker...). The advantages of a music module against a .wav/mp3 file are its very small size, a virtually endless length, the very fast replay, possible jump on a certain music part to fit to the screen action etc... It's of course possible to mix standard sound and music to have both played simultaneously.

The **ModPlug** library is used to have very high quality music playback and multiple tracker support.

The command `InitSound()` has to be called successfully before using any of the music related functions.

143.1 CatchMusic

Syntax

```
Result = CatchMusic(#Music, *Buffer, Size)
```

Description

Loads the specified music from the specified memory buffer. `PlayMusic()` can be used to start playing the music. **ModPlug** supports a lot of music formats, which includes: Protracker (4 channels), FastTracker (up to 32 channels, 16-bit quality), Impulse Tracker, etc.

Parameters

#Music A number to identify the new music module. `#PB_Any` can be used to auto-generate this number.

***Buffer** The memory buffer to load the music module from.

Size The buffer size.

Return value

Nonzero if the music module has been successfully loaded, zero otherwise.

Remarks

This function is useful when using the 'IncludeBinary ' PureBasic keyword. Then music modules can be packed inside the executable. Nevertheless, use this option with care, as it will take more memory than storing the music in an external file (the music is both in executable memory and load in physical memory).

Example

```
1  CatchMusic(0, ?Music, ?MusicEnd-?Music)
2  End
3
4  DataSection
5      Music:
6          IncludeBinary "Music.xml"
7      MusicEnd:
8  EndDataSection
```

See Also

LoadMusic() , PlayMusic()

143.2 FreeMusic

Syntax

```
FreeMusic(#Music)
```

Description

Stop and remove the specified music previously loaded with the LoadMusic() or CatchMusic() functions from memory. Once a music has been freed, it can't be played anymore.

Parameters

#Music The music to free. If **#PB_All** is specified, all the remaining musics are freed.

Return value

None.

Remarks

All remaining musics are automatically freed when the program ends.

143.3 GetMusicPosition

Syntax

```
Position = GetMusicPosition(#Music)
```

Description

Returns the current pattern position of the playing music module.

Parameters

#Music The music module to use.

Return value

The current pattern position of the playing music module. The first pattern position is zero.

See Also

SetMusicPosition()

143.4 GetMusicRow

Syntax

```
Row = GetMusicRow(#Music)
```

Description

Returns the current row position in the pattern of the playing music module.

Parameters

#Music The music module to use.

Return value

The current row position of the playing music module. The first row position is zero.

See Also

SetMusicPosition()

143.5 IsMusic

Syntax

```
Result = IsMusic(#Music)
```

Description

Tests if the given music module is valid and correctly initialized.

Parameters

#Music The music module to test.

Return value

Nonzero if the music module is valid, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a music module is ready to use.

See Also

LoadMusic() , CatchMusic()

143.6 LoadMusic

Syntax

```
Result = LoadMusic(#Music, Filename$)
```

Description

Loads the specified music module. PlayMusic() can be used to start playing the music. **ModPlug** supports a lot of music formats, which includes: Protracker (4 channels), FastTracker (up to 32 channels, 16-bit quality), Impulse Tracker, etc.

Parameters

#Music A number to identify the new music module. #PB_Any can be used to auto-generate this number.

Filename\$ The filename of the music module to load.

Return value

Nonzero if the music module has been successfully loaded, zero otherwise.

See Also

CatchMusic() , PlayMusic()

143.7 MusicVolume

Syntax

```
MusicVolume(#Music, Volume.f)
```

Description

Change the master volume of the specified music, in real-time.

Parameters

#Music The music module to use.

Volume.f The new volume to set for the music. Volume values can be from 0 to 100. Can be useful for fade-in/fade-out.

Return value

None.

143.8 PlayMusic

Syntax

```
PlayMusic(#Music)
```

Description

Starts to play the specified music previously loaded with the LoadMusic() or CatchMusic() functions.

Parameters

#Music The music module to play.

Return value

None.

See Also

LoadMusic() , CatchMusic() , StopMusic()

143.9 SetMusicPosition

Syntax

```
SetMusicPosition(#Music , Position)
```

Description

Changes the current pattern position of the playing #Music to the new one.

Parameters

#Music The music module to use.

Position The new music module pattern position. The first pattern position starts from 0.

Return value

None.

See Also

GetMusicPosition()

143.10 StopMusic

Syntax

```
StopMusic(#Music)
```

Description

Stop the #Music (if it was playing).

Parameters

#Music The music module to stop. If set to **#PB_All**, then all the currently playing musics are stopped.

Return value

None.

See Also

PlayMusic()

Chapter 144

Network

Overview

PureBasic supports the official protocol to exchange data via the internet: **TCP/IP** in both IPv4 and IPv6 form. This means that games and applications can be written using the well-established client/server model. With these functions, its possible to create any kind of internet-related application (browsers, web servers, ftp clients, etc) or even multiplayer games. It also fully supports TLS encryption through the **LibreSSL** library.

144.1 CloseNetworkConnection

Syntax

```
CloseNetworkConnection(Connection)
```

Description

Close the specified connection.

Parameters

Connection The connection to close. This is the result of either a call to `OpenNetworkConnection()` or `EventClient()` .

Return value

None.

Remarks

If a client calls this function the server will receive a `#PB_NetworkEvent_Disconnect` event. If this function is called from a server the connection will be closed without further notice to the client. When a server receives a `#PB_NetworkEvent_Disconnect` event its associated client connection is automatically closed and `CloseNetworkConnection()` must not be called in this case. All remaining opened connections are automatically closed when the program ends.

See Also

`OpenNetworkConnection()` , `EventClient()` , `CloseNetworkServer()`

144.2 ConnectionID

Syntax

```
Result = ConnectionID(Connection)
```

Description

Returns the unique system identifier of the connection.

Parameters

Connection The connection to use. This is the result of either a call to `OpenNetworkConnection()` or `EventClient()` .

Return value

Returns the system identifier. This result is sometimes also known as 'Handle'. Look at the extra chapter `Handles and Numbers` for more information.

See Also

`ServerID()`

144.3 ServerID

Syntax

```
Result = ServerID(#Server)
```

Description

Returns the unique system identifier of the server.

Parameters

#Server The server to use.

Return value

Returns the system identifier. This result is sometimes also known as 'Handle'. Look at the extra chapter `Handles and Numbers` for more information.

See Also

`ConnectionID()`

144.4 CloseNetworkServer

Syntax

```
CloseNetworkServer(#Server)
```

Description

Shutdown the specified running server. All clients connected to this server are automatically removed. The port is freed and can be reused.

Parameters

#Server The server to close.

Return value

None.

See Also

CreateNetworkServer() , CloseNetworkConnection()

144.5 CreateNetworkServer

Syntax

```
Result = CreateNetworkServer(#Server, Port [, Flags [, BoundIP$]])
```

Description

Create a new network server on the local computer using the specified port. To support TLS encryption, UseNetworkTLS() needs to be called before this command and a TLS flag needs to be specified.

Parameters

#Server A number to identify the new server. #PB_Any can be used to auto-generate this number.

Port The port to use for the server. Port values can range from 6000 to 7000 (this is a recommended area space, but it can go from 0 to 65000 in reality).

Flags (optional) Can be one of the following values:

#PB_Network_TCP: the server will use the TCP network protocol (default)

#PB_Network_UDP: the server will use the UDP network protocol

combined with one of the following values (using the '||' operand):

#PB_Network_IPv4: create a server using IPv4 (default)

#PB_Network_IPv6: create a server using IPv6

combined with one of the following values (using the '||' operand):

```
#PB_Network_NoTLS : no encryption will be used for this
server (default)
#PB_Network_TLSv1 : all non-deprecated TLS protocols will be
supported (currently v1.2 and v1.3)
#PB_Network_TLSv1_0: TLS v1.0 protocol will be supported
(warning, this protocol is deprecated for security reasons)
#PB_Network_TLSv1_1: TLS v1.1 protocol will be supported
(warning, this protocol is deprecated for security reasons)
#PB_Network_TLSv1_2: TLS v1.2 protocol will be supported
#PB_Network_TLSv1_3: TLS v1.3 protocol will be supported
```

BoundIP\$ (optional) The local IP address to bind the server. By default, the server is created on all available local interfaces, and accept connections from them. It can be useful to restrict the server to only one interface (for example "127.0.0.1") to avoid connection attempt from other interfaces. On Windows, binding only to the localhost avoid to trigger the built-in firewall.

Return value

Returns nonzero if the server was created successfully and zero if creation failed, for example, due to the port being already in use. If #PB_Any was used as the #Server parameter then the generated number is returned on success.

Remarks

Any number of servers can run simultaneously on the same computer, the only restriction being that two servers can not run using the same port and the same protocol (#PB_Network_TCP and #PB_Network_UDP). It's possible to create two servers using the same port, one using IPv4 and the other using IPv6. NetworkServerEvent() can be used to monitor server for events.

See Also

OpenNetworkConnection() , CloseNetworkServer() , NetworkServerEvent() , UseNetworkTLS()

144.6 ExamineIPAddresses

Syntax

```
Result = ExamineIPAddresses([Format])
```

Description

Start examining the available IP addresses on the local computer. NextIPAddress() is used to retrieve each IP.

Parameters

Format (optional) The format of the IP to examine. It can be one of the following value:

```
#PB_Network_IPv4: examine IPv4 addresses (default).
#PB_Network_IPv6: examine IPv6 addresses. Returned addresses
needs to be freed with FreeIP()
after use.
```

Return value

Returns nonzero if examining the addresses works and zero if it failed.

Example: IPv4

```
1  If ExamineIPAddresses()  
2      Repeat  
3          IP = NextIPAddress()  
4          If IP  
5              Debug "IPv4: " + IPString(IP)  
6          EndIf  
7      Until IP = 0  
8  EndIf
```

Example: IPv6

```
1  If ExamineIPAddresses(#PB_Network_IPv6)  
2      Repeat  
3          IP = NextIPAddress()  
4          If IP  
5              Debug "IPv6: " + IPString(IP, #PB_Network_IPv6)  
6              FreeIP(IP)  
7          EndIf  
8      Until IP = 0  
9  EndIf
```

See Also

NextIPAddress()

144.7 FreeIP

Syntax

```
FreeIP(IPAddress)
```

Description

Free an IPv6 address. This function only works with IPv6 addresses returned by MakeIPAddress(), NextIPAddress() and GetClientIP(), and must not be used with IPv4 addresses.

Parameters

IPAddress The IPv6 address to free.

Return value

None.

See Also

NextIPAddress(), MakeIPAddress(), GetClientIP()

144.8 HostName

Syntax

```
String\$ = HostName()
```

Description

Returns the computer's hostname.

Parameters

None.

Return value

Returns the host name.

144.9 IPString

Syntax

```
String\$ = IPString(IPAddress [, Format])
```

Description

Returns the string representation in dotted form (ie: "127.0.0.1" for IPv4 or "::1" for IPv6) of the specified numerical IPAddress.

Parameters

IPAddress The IP address. For IPv6, this address needs to be the result of MakeIPAddress() , NextIPAddress() or GetClientIP() .

Format (optional) The format of the IP to convert. It can be one of the following value:

```
#PB_Network_IPv4: convert an IPv4 address (default).  
#PB_Network_IPv6: convert an IPv6 address.
```

Return value

Returns the IP address as a string.

See Also

MakeIPAddress() , IPAddressField()

144.10 IPAddressField

Syntax

```
Result = IPAddressField(IPAddress, Field [, Format])
```

Description

Returns the given field value of the specified IP address.

Parameters

IPAddress The IP address. For IPv6, this address needs to be created with `MakeIPAddress()` .

Field The field to return. This value can be a value between 0 and 3 (0 being the leftmost value, 3 being the rightmost) for IPv4 and 0 to 7 for IPv6.

Format (optional) The format of the IP. It can be one of the following value:

```
#PB_Network_IPv4: an IPv4 address (default).  
#PB_Network_IPv6: an IPv6 address.
```

Return value

Returns the value of the specified field, in the range 0 to 255.

Remarks

This function is especially useful with:

- `IPAddressGadget()`
- `MakeIPAddress()`

See Also

`MakeIPAddress()` , `IPString()` , `IPAddressGadget()`

144.11 MakeIPAddress

Syntax

```
Result = MakeIPAddress(Field0, Field1, Field2, Field3 [, Field4,  
Field5, Field6, Field7])
```

Description

Returns the equivalent numeric value of the specified IP address.

Parameters

Field0, Field1, Field2, Field3 The individual fields of the address. Each field can only have a value between 0 and 255.

Field4, Field5, Field6, Field7 (optional) The remaining individual fields for IPv6 address. Each field can only have a value between 0 and 255. When these fields are specified, an IPv6 address is created. When the address is no more needed, it has to be manually freed with `FreeIP()` . IPv4 address must not be freed with `FreeIP()` .

Return value

Returns the IP address.

Remarks

This function is especially useful with:

- `IPAddressGadget()`

See Also

IPString() , IPAddressField()

144.12 EventServer

Syntax

```
Server = EventServer()
```

Description

This function returns the number of the server which has received data, allowing multiple servers to be managed on one computer. It is only needed on the server side.

Parameters

None.

Return value

Returns the #Server number of the server causing the event.

See Also

NetworkServerEvent() , EventClient()

144.13 EventClient

Syntax

```
Connection = EventClient()
```

Description

This function returns the connection of the client that sent data and is only needed on the server side.

Parameters

None.

Return value

Returns the connection of the client that caused the event.

Remarks

The functions GetClientIP() and GetClientPort() can be used to get more information on the client that sent data.

See Also

NetworkServerEvent() , GetClientIP() , GetClientPort()

144.14 GetClientIP

Syntax

```
IP = GetClientIP(Client)
```

Description

This function returns the IP address of the client and should be called after EventClient(). If the connection is an IPv6 connection the returned address must be freed with FreeIP().

Parameters

Client The client for which to get the IP.

Return value

Returns the IP address of the client.

See Also

GetClientPort() , IPString() , EventClient()

144.15 GetClientPort

Syntax

```
Port = GetClientPort(Client)
```

Description

Returns the client port and should be called after EventClient() .

Parameters

Client The client for which to get the port.

Return value

Returns the port of the client.

See Also

GetClientIP() , EventClient()

144.16 NetworkClientEvent

Syntax

```
Result = NetworkClientEvent(Connection)
```

Description

Checks if an event happened on a network connection created with OpenNetworkConnection() .

Parameters

Connection The connection to check.

Return value

Returns one of the following values:

```
#PB_NetworkEvent_None      : Nothing has happened
#PB_NetworkEvent_Data      : Raw data has been received (to be
                             read with ReceiveNetworkData()
)
#PB_NetworkEvent_Disconnect: The client has been disconnected
                             (connection closed).
```

See Also

ReceiveNetworkData() , NetworkServerEvent()

144.17 NetworkServerEvent

Syntax

```
Result = NetworkServerEvent([#Server])
```

Description

Checks if an event happened on one of the open network servers.

Parameters

#Server (optional) The server to use to check the events. When used, only the events which belongs to this server are handled, all other events are left untouched.

Return value

Returns one of the following values:

```
#PB_NetworkEvent_None      : Nothing has happened
#PB_NetworkEvent_Connect   : A new client has been connected to
                             the server (not available with #PB_Network_UDP connections)
#PB_NetworkEvent_Data      : Raw data has been received (to be
                             read with ReceiveNetworkData()
)
#PB_NetworkEvent_Disconnect: A client has quit the server
                             (disconnection). Its associated connection is
                             automatically closed,
                             CloseNetworkConnection()
                             must not be called for this client.
                             (Not available with #PB_Network_UDP
                             connections.)
```

Remarks

The server that received the event can be determined with EventServer() . The client that caused the event can be determined with EventClient() .

See Also

ReceiveNetworkData() , EventServer() , EventClient() , CreateNetworkServer()

144.18 NextIPAddress

Syntax

```
Result = NextIPAddress()
```

Description

Returns the next IP address of the local machine. ExamineIPAddresses() must be called before this command.

Parameters

None.

Return value

Returns the next IP address in numerical form. If the result is 0 then there are no more IP addresses to be examined. If ExamineIPAddresses() is called with the `#PB_Network_IPv6` format, the returned IP needs to be freed with FreeIP() after use.

Example: IPv4

```
1  If ExamineIPAddresses()
2      Repeat
3          IP = NextIPAddress()
4          If IP
5              Debug "IPv4: " + IPString(IP)
6          EndIf
7      Until IP = 0
8  EndIf
```

Example: IPv6

```
1  If ExamineIPAddresses(#PB_Network_IPv6)
2      Repeat
3          IP = NextIPAddress()
4          If IP
5              Debug "IPv6: " + IPString(IP, #PB_Network_IPv6)
6              FreeIP(IP)
7          EndIf
8      Until IP = 0
9  EndIf
```

See Also

ExamineIPAddresses() , IPString()

144.19 OpenNetworkConnection

Syntax

```
Connection = OpenNetworkConnection(ServerName$, Port [, Flags [,
    Timeout [, LocalIP$ [, LocalPort]]]])
```

Description

Opens a network connection to the specified server. To support TLS encryption, UseNetworkTLS() needs to be called before this command and a TLS flag needs to be specified.

Parameters

ServerName\$ The server to connect to. This can be an IP address or a full name (ie: "127.0.0.1" or "ftp.home.net").

Port The port on the server to connect to.

Flags (optional) This can be one of the following values:

```
#PB_Network_TCP: The connection will use the TCP network
                  protocol (default)
#PB_Network_UDP: The connection will use the UDP network
                  protocol. The connection will not be
                  explicitly created, as UDP is a
                  connectionless protocol, but it will add an
                  entry in the PureBasic management stack and
                  allow to send data via UDP using
                  the regular SendNetworkData()
                  (and related) functions.
```

combined with one of the following values (using the '||' operand):

```
#PB_Network_IPv4: open the connection using IPv4 (default)
#PB_Network_IPv6: open the connection using IPv6
```

combined with one of the following values (using the '||' operand):

```
#PB_Network_NoTLS : no encryption will be supported for this
                    connection (default)
#PB_Network_TLSv1 : all non-deprecated TLS protocols will be
                    supported (currently v1.2 and v1.3)
#PB_Network_TLSv1_0: TLS v1.0 protocol will be supported
                    (warning, this protocol is deprecated for security reasons)
#PB_Network_TLSv1_1: TLS v1.1 protocol will be supported
                    (warning, this protocol is deprecated for security reasons)
#PB_Network_TLSv1_2: TLS v1.2 protocol will be supported
#PB_Network_TLSv1_3: TLS v1.3 protocol will be supported
```

Timeout (optional) The maximum time (in milliseconds) before aborting the connection attempt. Usually, it shouldn't be set to a too low value (less than 5000 milliseconds), as initializing a connection over the net can take some time.

LocalIP\$ (optional) The local IP address to bind the connection to.

LocalPort (optional) The local port to bind the connection to. By default a random local port is automatically chosen for the new connection, but it can be overridden with this parameter.

Return value

Returns the connection identifier for the opened connection if the connection was established correctly. If there was an error the result is zero.

See Also

NetworkClientEvent() , SendNetworkData() , ReceiveNetworkData() , CloseNetworkConnection() , UseNetworkTLS()

144.20 ReceiveNetworkData

Syntax

```
Result = ReceiveNetworkData(Connection , *DataBuffer ,  
    DataBufferLength)
```

Description

Receives raw data from the specified client. This function can be used by both client and server applications and should be called only after having received a `#PB_NetworkEvent_Data` event.

Parameters

Connection The connection to receive data from. On the server side 'Connection' is the client which has send the data (can be easily obtained with EventClient()). On client side, 'Connection' is returned by OpenNetworkConnection() .

***DataBuffer** The memory address to receive the data to.

DataBufferLength The length of the buffer to receive data to.

Return value

Returns the number of bytes received. If 'Result' is equal to DataBufferLength then more data is available to be read. If an error occurred on the connection (link broken, connection close by the server etc...) 'Result' will be -1.

Remarks

On UDP connections, the maximum 'DataBufferLength' is 2048. On TCP connections, the maximum 'DataBufferLength' is 65536.

See Also

NetworkClientEvent() , NetworkServerEvent() , SendNetworkData()

144.21 SendNetworkData

Syntax

```
Result = SendNetworkData(Connection , *MemoryBuffer , Length)
```

Description

Sends raw data to the specified client. This function can be used by both client and server applications.

Parameters

Connection The connection to send data to. On the server side, 'Connection' is the client which should receive this data. On the client side, 'Connection' is returned by `OpenNetworkConnection()` .

***MemoryBuffer** The address of the data to send.

Length The length of the data to send.

Return value

Returns the number of bytes actually sent. If the number of bytes returned is not equal to the 'Length' parameter the receiving buffer of the user is probably full. If nothing could be sent then 'Result' will equal -1.

Remarks

On UDP connections, the maximum 'Length' is 2048. On TCP connections, the maximum 'Length' is 65536.

See Also

`SendNetworkString()`

144.22 SendNetworkString

Syntax

```
Result = SendNetworkString(Connection, String$ [, Format])
```

Description

Send a string to the specified client. This function can be used by both client and server applications.

Parameters

Connection The connection to send the string to. On the server side, 'Connection' is the client which should receive this data. On the client side, 'Connection' is returned by `OpenNetworkConnection()` .

String\$ The string to send.

Format (optional) The string format to use when sending the string. This can be one of the following values:

```
#PB_Ascii   : sends the strings as ASCII
#PB_UTF8    : sends the strings as UTF8 (default)
#PB_Unicode : send the strings as unicode
```

Return value

Returns the number of bytes sent.

Remarks

SendNetworkString() provides a solution to quickly send strings. The string will be sent without the terminating null-character, and can be received using ReceiveNetworkData() , after NetworkServerEvent() / NetworkClientEvent() returned #PB_NetworkEvent_Data). In unicode mode the string is sent as UTF-8, which is processor independent (unlike UTF-16 which is tied to processor endianness). There is no ReceiveNetworkString() command.

See Also

SendNetworkData() , ReceiveNetworkData()

144.23 UseNetworkTLS

Syntax

```
UseNetworkTLS([PrivateKey$, Certificate$ [, CaCertificate$]])
```

Description

Enable and configure TLS support for network library. This command must be called before using CreateNetworkServer() or OpenNetworkConnection() with the TLS flags.

Parameters

PrivateKey\$ (optional) The private key to use for TLS when using CreateNetworkServer() . When using TLS only with OpenNetworkConnection() , this parameter is not needed. It should be in following format:

```
1 "-----BEGIN PRIVATE KEY-----"+#LF$+
2 "MIICdQIBADANBgkqhkiG9w0BAQEFAASCAl8wgJbAgEAAoGBAPEbSQq/uwESZduPtDd83qXX
"-----END PRIVATE KEY-----"
```

Certificate\$ (optional) The certificat to use for TLS when using CreateNetworkServer() . When using TLS only with OpenNetworkConnection() , this parameter is not needed. It should be in following format:

```
1 "-----BEGIN CERTIFICATE-----"+#LF$+
2 "MIICnTCCAgYCCQDOAWy2vzfcpzANBgkqhkiG9w0BAQUFADCBkjELMAkGA1UEBhMCVVMxDjAM
"-----END CERTIFICATE-----"
```

CaCertificate\$ (optional) The intermediate authority certificate who issued the 'Certificate\$' to use for TLS when using CreateNetworkServer() . When using TLS only with OpenNetworkConnection() , this parameter is not needed. It should be in the same format than 'Certificate\$'.

Return value

None.

Example: TLS Server

```
1 Timeout = 15000 ; 15 secs overall timeout
2
3 PrivateKey$ = "-----BEGIN PRIVATE KEY-----"+#LF$+
4             "MIICdQIBADANBgkqhkiG9w0BAQEFAASCAl8wggJbAgEAAoGBAPEbSQq/uwE$ZduPtDd83q
5             "-----END PRIVATE KEY-----"
6
7 Cert$ = "-----BEGIN CERTIFICATE-----"+#LF$+
8         "MIICnTCCAgYCCQDOAWy2vzfcpzANBgkqhkiG9w0BAQUFADCBkjELMAkGA1UEBhMCVVMxD
9         "-----END CERTIFICATE-----"
10
11 UseNetworkTLS(PrivateKey$, Cert$)
12 If CreateNetworkServer(0, 20252, #PB_Network_TCP |
13   #PB_Network_IPv4 | #PB_Network_TLSv1, "127.0.0.1")
14   Debug "TLS Server launched on 127.0.0.1:20252 - Waiting for
15   connections..."
16
17   Repeat
18     Select NetworkServerEvent()
19     Case #PB_NetworkEvent_Data
20
21       Debug "Data recieved !"
22       ClientID = EventClient()
23       *Buffer = AllocateMemory(1000)
24       If *Buffer
25         Debug ReceiveNetworkData(ClientID, *Buffer, 1000)
26
27         Debug "String: "+PeekS(*Buffer, -1, #PB_UTF8)
28         SendNetworkString(ClientID, "Well received !!!")
29
30         FreeMemory(*Buffer)
31       EndIf
32
33     Case #PB_NetworkEvent_None
34       Delay(100) ; Don't steal the whole CPU time while waiting
35       for connection
36       Timeout - 100
37     EndSelect
38
39   Until Timeout <= 0
40
41   Debug "Closing server after " + Str(Timeout/1000) + " secs
42   timeout"
43
44   CloseNetworkServer(0)
45 Else
46   Debug "Can't create the TLS server"
47 EndIf
```

Example: TLS Client

```
1 Timeout = 150000 ; 15 secs overall timeout
2
3 UseNetworkTLS()
4
5 Connection = OpenNetworkConnection("127.0.0.1", 20252,
6   #PB_Network_TCP | #PB_Network_IPv4 | #PB_Network_TLSv1)
```

```

6   If Connection
7
8       ; Send some string to the server to test
9       SendNetworkString(Connection, "GET / HTTP/1.1" + #CRLF$ +
    "Host: www.purebasic.fr" + #CRLF$ + #CRLF$)
10
11     Repeat
12         Select NetworkClientEvent(Connection)
13             Case #PB_NetworkEvent_Data
14                 *Buffer = AllocateMemory($FFFF)
15                 If *Buffer
16                     Length = ReceiveNetworkData(Connection, *Buffer,
    MemorySize(*Buffer))
17                     Received$ + PeekS(*Buffer, Length, #PB_UTF8 |
    #PB_ByteLength)
18                     Debug "Recieved string: "+ Received$
19                     FreeMemory(*Buffer)
20                 EndIf
21
22             Case #PB_NetworkEvent_Disconnect
23                 Debug "Disconnected"
24                 Break
25
26             Case #PB_NetworkEvent_None
27                 Delay(100)
28                 Timeout - 100
29
30         EndSelect
31     Until Timeout <= 0
32
33     CloseNetworkConnection(Connection)
34 Else
35     Debug "Can't open the connection on 127.0.0.1:20252"
36 EndIf

```

See Also

CreateNetworkServer() , OpenNetworkConnection()

Chapter 145

Node

Overview

Nodes are containers which can be used to group objects like entities , sound , camera , billboard and even nodes together. Once an object is attached to a node, its position and movement is relative to the node position. It allows easy hierarchical object management. `InitEngine3D()` must be called successfully before using the node functions.

145.1 AttachNodeObject

Syntax

```
AttachNodeObject(#Node, ObjectID)
```

Description

Attaches an existing object to a node.

Parameters

#Node The node to use.

ObjectID The object to attach. The supported objects are the following:

```
- Entity : use EntityID()
as 'ObjectID'.
- Sound3D: use SoundID3D()
as 'ObjectID'.
- Text3D : use Text3DID()
as 'ObjectID'.
- Camera : use CameraID()
as 'ObjectID'.
- Light  : use LightID()
as 'ObjectID'.
- Node   : use NodeID()
as 'ObjectID'.
- Mesh   : use MeshID()
as 'ObjectID' (supported only if #PB_Mesh_Dynamic flag is set).
- BillboardGroup : use BillboardGroupID()
as 'ObjectID'.
- ParticleEmitter: use ParticleEmitterID()
as 'ObjectID'.
```

Return value

None.

Remarks

An object can be detached from a node with `DetachNodeObject()` .

See Also

`DetachNodeObject()`

145.2 DetachNodeObject

Syntax

```
DetachNodeObject(#Node, ObjectID)
```

Description

Detaches a previously attached object from a node.

Parameters

#Node The node to use.

ObjectID The object to detach. The supported objects are the following:

```
    - Entity : use EntityID()
as 'ObjectID'.
    - Sound3D: use SoundID3D()
as 'ObjectID'.
    - Camera : use CameraID()
as 'ObjectID'.
    - Light  : use LightID()
as 'ObjectID'.
    - Node   : use NodeID()
as 'ObjectID'.
    - Mesh   : use MeshID()
as 'ObjectID' (supported only if #PB_Mesh_Dynamic flag is set).
    - BillboardGroup : use BillboardGroupID()
as 'ObjectID'.
    - ParticleEmitter: use ParticleEmitterID()
as 'ObjectID'.
```

Return value

None.

Remarks

An object can be attached to a node with `AttachNodeObject()` .

See Also

`AttachNodeObject()`

145.3 CreateNode

Syntax

```
Result = CreateNode(#Node [, x, y, z])
```

Description

Creates a new node at the specified position.

Parameters

#Node A number to identify the new node. **#PB_Any** can be used to auto-generate this number.

x, y, z (optional) The absolute position in the world of the new node. By default the new node is created at position "0, 0, 0".

Return value

Nonzero if the node was successfully created, zero otherwise. If **#PB_Any** was used for the **#Node** parameter then the generated number is returned on success.

See Also

FreeNode()

145.4 NodeID

Syntax

```
NodeID = NodeID(#Node)
```

Description

Returns the unique system identifier of the node.

Parameters

#Node The node to use.

Return value

Returns the unique system identifier of the node.

145.5 NodeLookAt

Syntax

```
NodeLookAt(#Node, x, y, z [, DirectionX, DirectionY, DirectionZ])
```

Description

The point (in world unit) that a node is facing. The position of the node is not changed.

Parameters

#Node The node to use.

x, y, z The position (in world unit) to point the node at.

DirectionX, DirectionY, DirectionZ (optional) The vector direction of the node (values between -1.0 and 1.0).

Return value

None.

145.6 NodeX

Syntax

```
Result = NodeX(#Node [, Mode])
```

Description

Returns the 'x' position of the node in the world.

Parameters

#Node The node to use.

Mode (optional) The mode to get the 'x' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'x' position of the node in  
the world (default).  
#PB_Relative: get the 'x' position of the node relative to  
its parent.
```

Return value

The 'x' position of the node in the world.

See Also

NodeY() , NodeZ()

145.7 NodeY

Syntax

```
Result = NodeY(#Node [, Mode])
```

Description

Returns the 'y' position of the node in the world.

Parameters

#Node The node to use.

Mode (optional) The mode to get the 'y' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'y' position of the node in  
the world (default).  
#PB_Relative: get the 'y' position of the node relative to  
its parent.
```

Return value

The 'y' position of the node in the world.

See Also

NodeX() , NodeZ()

145.8 NodeZ

Syntax

```
Result = NodeZ(#Node [, Mode])
```

Description

Returns the 'z' position of the node in the world.

Parameters

#Node The node to use.

Mode (optional) The mode to get the 'z' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'z' position of the node in  
the world (default).  
#PB_Relative: get the 'z' position of the node relative to  
its parent.
```

Return value

The 'z' position of the node in the world.

See Also

NodeX() , NodeY()

145.9 FreeNode

Syntax

```
FreeNode(#Node)
```

Description

Frees the specified node created with CreateNode() before. All its associated memory is released and this object can't be used anymore. The attached objects are not freed automatically, and can be re-used.

Parameters

#Node The node to free. If **#PB_All** is specified, all the remaining nodes are freed.

Return value

None.

Remarks

All remaining nodes are automatically freed when the program ends.

145.10 IsNode

Syntax

```
Result = IsNode(#Node)
```

Description

Tests if the given node is valid and correctly initialized.

Parameters

#Node The node to test.

Return value

Nonzero if the node is valid, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a node is ready to use.

See Also

CreateNode()

145.11 MoveNode

Syntax

```
MoveNode(#Node, x, y, z [, Mode])
```

Description

Move the specified node.

Parameters

#Node The node to move.

x, y, z The new position of the node.

Mode (optional) The move mode. It can be one of the following values:

```
#PB_Relative: relative move, from the current node position  
              (default).  
#PB_Absolute: absolute move to the specified position.
```

combined with one of the following values:

```
#PB_Local  : local move.  
#PB_Parent: move relative to the parent position.  
#PB_World  : move relative to the world.
```

Return value

None.

See Also

RotateNode()

145.12 RotateNode

Syntax

```
RotateNode(#Node, x, y, z [, Mode])
```

Description

Rotates the node according to the specified x,y,z angle values.

Parameters

#Node The node to rotate.

x, y, z The new rotation to apply to the node. Values are in degree, ranging from 0 to 360.

Mode (optional) The rotate mode. It can be one of the following values:

```
#PB_Absolute: absolute rotation (default).  
#PB_Relative: relative rotation based on the previous node  
              rotation.
```

Return value

None.

See Also

MoveNode()

145.13 ScaleNode

Syntax

```
ScaleNode(#Node, x, y, z [, Mode])
```

Description

Scales the node according to the specified x,y,z values. When using `#PB_Relative` mode, this is a factor based scale which means the node size will be multiplied with the given value to obtain the new size.

Parameters

#Node The node to use.

x, y, z The scale values.

Mode (optional) The scale mode. It can be one of the following value:

```
#PB_Relative: relative scale, based on the previous size  
(default). Using 1.0 for scale value will let this value  
unchanged.  
#PB_Absolute: absolute scale, in world unit.
```

Return value

None.

Remarks

The attached entities doesn't scale automatically with the node, unless the attribute `#PB_Entity_InheritScale` has been set.

145.14 NodeFixedYawAxis

Syntax

```
NodeFixedYawAxis(#Node, Enable [, VectorX, VectorY, VectorZ])
```

Description

Change the fixed yaw axis of the node. The default behaviour of a node is to yaw around its own Y axis.

Parameters

#Node The node to use.

Enable Enable or disable the use of a custom yaw axis. If set to **#True**, a new axis vector has to be specified. If set to **#False**, the node will yaw around its own Y axis.

VectorX, VectorY, VectorZ (optional) Vector direction of the new yaw axis (values between -1.0 and 1.0). 'Enable' parameter has to be set to **#True** to have any effect.

Return value

None.

145.15 NodeRoll

Syntax

```
Result = NodeRoll(#Node [, Mode])
```

Description

Get the roll of the node.

Parameters

#Node The node to use.

Mode (optional) The mode to get the roll. It can be one of the following value:

```
#True : the roll is the raw value, but it can't be used in
RotateNode()
to get back the same orientation (default).
#False: the roll is adjusted, so it can be put back in
RotateNode()
to get back the same orientation.
```

Return value

The current roll value of the node.

See Also

NodeYaw() , NodePitch()

145.16 NodePitch

Syntax

```
Result = NodePitch(#Node [, Mode])
```

Description

Get the pitch of the node.

Parameters

#Node The node to use.

Mode (optional) The mode to get the pitch. It can be one of the following value:

```
#True : the pitch is adjusted, so it can put back in
        RotateNode()
to get back the same orientation (default).
#False: the pitch is the raw value, but it can't be used in
        RotateNode()
to get back the same orientation.
```

Return value

The current pitch value of the node.

See Also

NodeYaw() , NodeRoll()

145.17 NodeYaw

Syntax

```
Result = NodeYaw(#Node [, Mode])
```

Description

Get the yaw of the node.

Parameters

#Node The node to use.

Mode (optional) The mode to get the yaw. It can be one of the following value:

```
#True : the yaw is adjusted, so it can put back in
        RotateNode()
to get back the same orientation (default).
#False: the yaw is the raw value, but it can't be used in
        RotateNode()
to get back the same orientation.
```

Return value

The current yaw value of the node.

See Also

NodePitch() , NodeRoll()

Chapter 146

NodeAnimation

Overview

Node animation allows to define track to follow for a node, with predefined key frame and smooth interpolation. Then an object like a camera can be attached to this node and be moved easily on the track.

InitEngine3D() should be called successfully before using the node animation functions.

146.1 CreateNodeAnimation

Syntax

```
Result = CreateNodeAnimation(#NodeAnimation, NodeID, Length,  
                             Interpolation, RotationInterpolation)
```

Description

Creates a new node animation of the specified length. A node animation doesn't exist physically in the 3D world, it is an virtual track to move a node (and its attached object) easily around the world.

Parameters

#NodeAnimation The number to identify the new node animation. #PB_Any can be used to auto-generate this number.

NodeID The node to move automatically. NodeID() can be use to get a valid node identifier.

Length The length of the new animation (in milliseconds).

Interpolation The type of interpolation to apply between points. It can be one of the following value:

#PB_NodeAnimation_Linear: each points will be linked together using a straight line, which can result to abrupt direction change

#PB_NodeAnimation_Spline: each points will be linked together using a spline, which results in smooth direction change, but is also slower.

RotationInterpolation The type of interpolation for the rotation to apply between points. It can be one of the following value:

```
#PB_NodeAnimation_LinearRotation:    each points will be
    linked together using a straight line, which can result to
    abrupt rotation
#PB_NodeAnimation_SphericalRotation: each points will be
    linked together using a curve, which results in smoother
    rotation, but is also slower.
```

Return value

Returns zero if the #NodeAnimation can't be created. If #PB_Any is used as '#NodeAnimation' parameter, the new node animation number is returned.

See Also

FreeNodeAnimation()

146.2 FreeNodeAnimation

Syntax

```
FreeNodeAnimation(#NodeAnimation)
```

Description

Frees a node animation and releases all its associated memory. This node animation must not be used (by using its number with the other functions in this library) after calling this function, unless you create it again.

Parameters

#NodeAnimation The node animation to free. If #PB_All is specified, all the remaining node animations are freed.

Return value

None.

Remarks

All remaining node animations are automatically freed when the program ends.

See Also

CreateNodeAnimation()

146.3 CreateNodeAnimationKeyFrame

Syntax

```
CreateNodeAnimationKeyFrame(#NodeAnimation, Time, x, y, z)
```

Description

Create a new keyframe for the `#NodeAnimation`. A keyframe is a point in the world at a specified time. When the node animation will be played, it will follow every keyframe and thus moving from points to points. The move will be interpolated to respect the time constraint. For example, if the first keyframe is set at time 0, the second at time 1000 and the third at time 3000, going from the first keyframe to the second will takes 1000 milliseconds, and going from the second keyframe to the third will takes 2000 milliseconds. The overall animation will take 3000 milliseconds.

Parameters

#NodeAnimation The node animation to use.

Time The time in the animation to set the keyframe (in milliseconds). This value has to be between zero and the 'Length' defined in `CreateNodeAnimation()` .

x, y, z The new keyframe position in the world.

Return value

None.

See Also

`CreateNodeAnimation()`

146.4 GetNodeAnimationKeyFrameTime

Syntax

```
Result = GetNodeAnimationKeyFrameTime(#NodeAnimation, KeyFrame)
```

Description

Returns the `#NodeAnimation` keyframe time.

Parameters

#NodeAnimation The node animation to use.

#Keyframe The node animation keyframe index. The first keyframe index is zero.

Return value

Returns the node animation keyframe time (in milliseconds), or zero if the keyframe doesn't exists.

See Also

`CreateNodeAnimationKeyFrame()`

146.5 SetNodeAnimationKeyFramePosition

Syntax

```
SetNodeAnimationKeyFramePosition(#NodeAnimation, KeyFrame, x, y, z)
```

Description

Changes the keyframe position for the #NodeAnimation.

Parameters

#NodeAnimation The node animation to use.

#Keyframe The node animation keyframe index. The first keyframe index is zero. If the keyframe doesn't exist, this function has no effect.

x, y, z The new keyframe position in the world.

Return value

None.

See Also

CreateNodeAnimationKeyFrame()

146.6 GetNodeAnimationKeyFrameX

Syntax

```
Result = GetNodeAnimationKeyFrameX(#NodeAnimation, KeyFrame)
```

Description

Returns the #NodeAnimation keyframe 'x' position.

Parameters

#NodeAnimation The node animation to use.

#Keyframe The node animation keyframe index. The first keyframe index is zero.

Return value

Returns the node animation keyframe 'x' position in the world, or zero if the keyframe doesn't exist.

See Also

CreateNodeAnimationKeyFrame(), GetNodeAnimationKeyFrameY(),
GetNodeAnimationKeyFrameZ()

146.7 GetNodeAnimationKeyFrameY

Syntax

```
Result = GetNodeAnimationKeyFrameY(#NodeAnimation, KeyFrame)
```

Description

Returns the #NodeAnimation keyframe 'y' position.

Parameters

#NodeAnimation The node animation to use.

#Keyframe The node animation keyframe index. The first keyframe index is zero.

Return value

Returns the node animation keyframe 'y' position in the world, or zero if the keyframe doesn't exist.

See Also

CreateNodeAnimationKeyFrame(), GetNodeAnimationKeyFrameX(),
GetNodeAnimationKeyFrameZ()

146.8 GetNodeAnimationKeyFrameZ

Syntax

```
Result = GetNodeAnimationKeyFrameZ(#NodeAnimation, KeyFrame)
```

Description

Returns the #NodeAnimation keyframe 'z' position.

Parameters

#NodeAnimation The node animation to use.

#Keyframe The node animation keyframe index. The first keyframe index is zero.

Return value

Returns the node animation keyframe 'z' position in the world, or zero if the keyframe doesn't exist.

See Also

CreateNodeAnimationKeyFrame(), GetNodeAnimationKeyFrameX(),
GetNodeAnimationKeyFrameY()

146.9 SetNodeAnimationKeyFrameRotation

Syntax

```
SetNodeAnimationKeyFrameRotation(#NodeAnimation, KeyFrame, x, y, z  
[, w, Mode])
```

Description

Changes the keyframe rotation for the #NodeAnimation.

Parameters

#NodeAnimation The node animation to use.

#Keyframe The node animation keyframe index. The first keyframe index is zero. If the keyframe doesn't exist, this function has no effect.

x, y, z The keyframe 'x, y, z' rotation in the world. Values depend on the selected mode.

w (optional) The keyframe 'w' rotation in the world (only used for #PB_Orientation_Quaternion and #PB_Orientation_Direction).

Mode (optional) The rotation mode. It can be one of the following values:

```
#PB_Orientation_PitchYawRoll: 'x' (pitch), 'y' (yaw), 'z'  
    (roll), applied in this order (default).  
#PB_Orientation_Quaternion   : 'x', 'y', 'z', 'w' for  
    quaternion values  
#PB_Orientation_Direction    : 'x', 'y', 'z' for direction  
    vector, and 'w' for rotation (roll).
```

Return value

None.

See Also

CreateNodeAnimationKeyFrame(), GetNodeAnimationKeyFramePitch(),
GetNodeAnimationKeyFrameYaw(), GetNodeAnimationKeyFrameRoll()

146.10 GetNodeAnimationKeyFramePitch

Syntax

```
Result = GetNodeAnimationKeyFramePitch(#NodeAnimation, KeyFrame)
```

Description

Returns the #NodeAnimation keyframe pitch.

Parameters

#NodeAnimation The node animation to use.

#Keyframe The node animation keyframe index. The first keyframe index is zero.

Return value

Returns the current node animation keyframe pitch, or zero if the keyframe doesn't exists.

See Also

CreateNodeAnimationKeyFrame() , GetNodeAnimationKeyFrameYaw() ,
GetNodeAnimationKeyFrameRoll()

146.11 GetNodeAnimationKeyFrameYaw

Syntax

```
Result = GetNodeAnimationKeyFrameYaw( #NodeAnimation , KeyFrame )
```

Description

Returns the #NodeAnimation keyframe yaw.

Parameters

#NodeAnimation The node animation to use.

#Keyframe The node animation keyframe index. The first keyframe index is zero.

Return value

Returns the current node animation keyframe yaw, or zero if the keyframe doesn't exists.

See Also

CreateNodeAnimationKeyFrame() , GetNodeAnimationKeyFramePitch() ,
GetNodeAnimationKeyFrameRoll()

146.12 GetNodeAnimationKeyFrameRoll

Syntax

```
Result = GetNodeAnimationKeyFrameRoll( #NodeAnimation , KeyFrame )
```

Description

Returns the #NodeAnimation keyframe roll.

Parameters

#NodeAnimation The node animation to use.

#Keyframe The node animation keyframe index. The first keyframe index is zero.

Return value

Returns the current node animation keyframe roll, or zero if the keyframe doesn't exists.

See Also

CreateNodeAnimationKeyFrame() , GetNodeAnimationKeyFramePitch() ,
GetNodeAnimationKeyFrameYaw()

146.13 SetNodeAnimationKeyFrameScale

Syntax

```
SetNodeAnimationKeyFrameScale(#NodeAnimation , KeyFrame , x , y , z)
```

Description

Changes the keyframe scale factor for the #NodeAnimation. The scale factor will be applied to the node associated to the animation.

Parameters

#NodeAnimation The node animation to use.

#Keyframe The node animation keyframe index. The first keyframe index is zero. If the keyframe doesn't exist, this function has no effect.

x The new keyframe 'x' scale factor. If this value is 1.0, no scaling will be applied on the 'x' axis.

y The new keyframe 'y' scale factor. If this value is 1.0, no scaling will be applied on the 'y' axis.

z The new keyframe 'z' scale factor. If this value is 1.0, no scaling will be applied on the 'z' axis.

Return value

None.

Remarks

The scale factor will resize the node by multiplying its size with the given factor:

- a scale factor of 1.0 will not affect the size of the node
- a scale factor of 2.0 will double the size of the node
- a scale factor of 0.5 will half the size of the node

See Also

CreateNodeAnimationKeyFrame() , GetNodeAnimationKeyFramePitch() ,
GetNodeAnimationKeyFrameYaw() , GetNodeAnimationKeyFrameRoll()

146.14 AddNodeAnimationTime

Syntax

```
AddNodeAnimationTime(#NodeAnimation , Time)
```

Description

Add time to the specified #NodeAnimation.

Parameters

#NodeAnimation The node animation to use.

Time The time to add (in milliseconds) to the specified animation, relative to the current animation time.

Return value

None.

See Also

StartNodeAnimation()

146.15 StartNodeAnimation

Syntax

```
StartNodeAnimation(#NodeAnimation [, Flags])
```

Description

Start the specified #NodeAnimation. The animation is always started from the beginning.

Parameters

#NodeAnimation The node animation to use.

Flag Flags can be a combination of the following values:

```
#PB_NodeAnimation_Once:    Play the animation only once. By
    default the animation loops automatically when its end is
    reached.
```

```
                                NodeAnimationStatus()
can be used to detect the animation end.
```

Return value

None.

See Also

StopNodeAnimation() , NodeAnimationStatus() , AddNodeAnimationTime()

146.16 StopNodeAnimation

Syntax

```
StopNodeAnimation(#NodeAnimation)
```

Description

Stop the specified #NodeAnimation.

Parameters

#NodeAnimation The node animation to stop.

Return value

None.

See Also

StartNodeAnimation()

146.17 NodeAnimationStatus

Syntax

```
Result = NodeAnimationStatus(#NodeAnimation)
```

Description

Return the specified #NodeAnimation status.

Parameters

#NodeAnimation The node animation to use.

Return value

The return value can be one of the following constants:

```
#PB_NodeAnimation_Stopped: The animation is stopped (or has ended).  
#PB_NodeAnimation_Started: The animation is running.
```

See Also

StartNodeAnimation() , StopNodeAnimation()

146.18 GetNodeAnimationTime

Syntax

```
Result = GetNodeAnimationTime(#NodeAnimation)
```

Description

Returns the current #NodeAnimation time.

Parameters

#NodeAnimation The node animation to use.

Return value

The current node animation time (in milliseconds) or 0 if the animation isn't running.

See Also

StartNodeAnimation() , AddNodeAnimationTime() , SetNodeAnimationTime()

146.19 SetNodeAnimationTime

Syntax

```
SetNodeAnimationTime(#NodeAnimation, Time)
```

Description

Changes the current #NodeAnimation time. This is an absolute time position. To change the time relative to the current time, use AddNodeAnimationTime() .

Parameters

#NodeAnimation The node animation to use.

Time The absolute time to set (in milliseconds).

Return value

None.

See Also

StartNodeAnimation() , AddNodeAnimationTime() , GetNodeAnimationTime()

146.20 GetNodeAnimationLength

Syntax

```
Result = GetNodeAnimationLength(#NodeAnimation)
```

Description

Returns the #NodeAnimation length.

Parameters

#NodeAnimation The node animation to use.

Return value

The node animation length (in milliseconds).

See Also

StartNodeAnimation() , SetNodeAnimationLength()

146.21 SetNodeAnimationLength

Syntax

```
SetNodeAnimationLength(#NodeAnimation, Length)
```

Description

Change the #NodeAnimation length.

Parameters

#NodeAnimation The node animation to use.

Length The new node animation length (in milliseconds).

See Also

StartNodeAnimation(), GetNodeAnimationLength()

146.22 GetNodeAnimationWeight

Syntax

```
Result = GetNodeAnimationWeight(#NodeAnimation)
```

Description

Returns the #NodeAnimation weight. The weight is useful when playing several animations at once. For example to do a smooth transition from one animation to another, it is possible to reduce progressively the weight of the first animation and increase the weight of the second animation.

Parameters

#NodeAnimation The node animation to use.

Return value

The current node animation weight (value between 0.0 and 1.0). If the weight is 0, then the animation has no effect. If the weight is 1, then animation is fully playing.

See Also

StartNodeAnimation()

146.23 SetNodeAnimationWeight

Syntax

```
SetNodeAnimationWeight(#NodeAnimation, Weight)
```

Description

Changes the #NodeAnimation weight. The weight is useful when playing several animations at once. For example to do a smooth transition from one animation to another, it is possible to reduce progressively the weight of the first animation and increase the weight of the second animation.

Parameters

#NodeAnimation The node animation to use.

Weight The new node animation weight (value between 0.0 and 1.0). If the weight is 0, then the animation has no effect. If the weight is 1, then animation is fully playing.

Return value

None.

See Also

StartNodeAnimation()

Chapter 147

OnError

Overview

This library provides a way to track program errors (program crashes) similar to the PureBasic debugger but without the drawbacks of a bigger filesize and lower execution speed when using the debugger. This way the final version of a program which is shipped to the end user can still intercept program errors and provide some information about the error to the user so he can report it back to the developer.

The PureBasic debugger is still the better tool for finding bugs during the development phase, as it provides much more detailed information about the state of the program (like actual variable values) and also has a number of interactive features to quickly locate bugs.

Note: If both this library and the PureBasic debugger are used, not all errors will be caught by the OnError library as some checks are made and reported by the debugger even before the actual code with the error is executed. To test OnError in the IDE, be sure to turn off the debugger ([DisableDebugger](#) is not enough).

This library can provide information about the source code file and line at which the error occurred through the `ErrorFile()` and `ErrorLine()` commands, but only if this feature is enabled when compiling the program (it causes a very small slowdown in execution speed to track the line numbers). To enable this feature, enable the "Enable OnError lines support" checkbox in the compiler options or specify the `/LINENUMBERING` (Windows) or `-linenumbering` (Linux, Mac OSX) command-line switch when compiling from the command-line .

On Linux, in a fatal error handler, UI shouldn't be used, and the program should exits as fast as possible as it is in an unstable state.

147.1 OnErrorExit

Syntax

```
OnErrorExit ()
```

Description

Changes the action taken if an error occurs to directly exit the program, even if the default action on the system is not to exit the program on this kind of error. The system may display an error dialog or print an error-message on the console on exit.

Parameters

None.

Return value

None.

Remarks

To exit the program silently and suppress any system message, use `OnErrorCall()` and end the program from the error handler.

Example

```
1  MessageRequester("OnError test", "Test start")
2
3  OnErrorExit()
4  Pokes(10, "Hello World") ; Cause a #PB_OnError_InvalidMemory error
5
6  MessageRequester("OnError test", "This should never be displayed")
```

147.2 OnErrorCall

Syntax

```
OnErrorCall(@ErrorHandler())
```

Description

Changes the action taken if an error occurs to call the specified handler procedure. The handler can display information about the error to the user using the commands of this library and do any needed cleanup to shutdown the application. The program will end as soon as the handler returns.

Parameters

@ErrorHandler() The address of a procedure of the following form:

```
1  Procedure ErrorHandler()
2      ; Your code here
3  EndProcedure
```

Return value

None.

Example

```
1  Procedure ErrorHandler()
2      MessageRequester("OnError test", "The following error happened:
3      " + ErrorMessage())
4  EndProcedure
5
6  MessageRequester("OnError test", "Test start")
7
8  OnErrorCall(@ErrorHandler())
9  Pokes(10, "Hello World") ; Cause a #PB_OnError_InvalidMemory error
10
11 MessageRequester("OnError test", "This should never be displayed")
```

147.3 OnErrorGoto

Syntax

```
OnErrorGoto(?LabelAddress)
```

Description

Changes the action taken when an error occurs to jump to the specified label address and continue program execution there. After the jump to the label, the functions of this library can be used to get further information about the error.

Parameters

?LabelAddress The address of a label in the program to jump to.

Return value

None.

Remarks

The program stack will not be adjusted before jumping to the label, so local variables should not be accessed as they may not be reachable anymore. It is also not safe to continue normal program execution after an error as things like the return address of a procedure may be wrong if the stack is no longer correct. The best practice is just to gather and display information about the error and then exit the program.

Example

```
1  MessageRequester("OnError test", "Test start")
2
3  OnErrorGoto(?ErrorHandler)
4  Pokes(10, "Hello World") ; Cause a #PB_OnError_InvalidMemory error
5
6  MessageRequester("OnError test", "This should never be displayed")
7  End
8
9  ErrorHandler:
10 MessageRequester("OnError test", "The following error happened: "
11    + ErrorMessage())
12 End
```

147.4 OnErrorDefault

Syntax

```
OnErrorDefault()
```

Description

Changes the action taken when an error occurs back to the system default. This usually means displaying an error dialog and exiting the program, but it may also mean to just ignore certain errors. To exit the program on every error, use `OnErrorExit()`.

Parameters

None.

Return value

None.

Remarks

When using the OnError library inside a DLL the best practice is to set the error handler at the beginning of every public DLL function and resetting it back to the default with this command before returning to the caller to make sure there is no interference between the OnError library and any exception handling done by the calling program.

Example

```
1  Procedure ErrorHandler()  
2      MessageRequester("OnError test", "The following error happened:  
   " + ErrorMessage())  
3  EndProcedure  
4  
5  MessageRequester("OnError test", "Test start")  
6  
7  OnErrorCall(@ErrorHandler())  
8  OnErrorDefault()           ; Comment this to get the handler call  
   instead of the system error handling  
9  Pokes(10, "Hello World") ; Cause a #PB_OnError_InvalidMemory error  
10  
11 MessageRequester("OnError test", "This should never be displayed")
```

147.5 ErrorCode

Syntax

```
Result = ErrorCode()
```

Description

Returns the error code of the currently handled error. This command only returns a meaningful value if there was an error handled by OnErrorCall() or OnErrorGoto() .

Parameters

None.

Return value

One of the following values:

#PB_OnError_InvalidMemory invalid location	: Read or write operation on an invalid location
#PB_OnError_Floatingpoint	: Floating-point error
#PB_OnError_Breakpoint (non-PureBasic breakpoints)	: Debugger breakpoint reached

```
#PB_OnError_IllegalInstruction      : Attempt to execute an illegal
instruction
#PB_OnError_PrivilegedInstruction: Attempt to execute a
privileged (system-) instruction
#PB_OnError_DivideByZero           : Division by zero (Windows
only)
```

Linux and Mac OSX report `#PB_OnError_Floatingpoint` for division by zero errors. In addition, every OS may have more possible error values. On Windows, custom errors can be raised with the `RaiseError()` command.

147.6 ErrorMessage

Syntax

```
Result\$_ = ErrorMessage([ErrorCode])
```

Description

Returns an error-message for the given error code in english.

Parameters

ErrorCode (optional) A specific error code. See `ErrorCode()` for the available error codes.

Return value

The message for the currently handled error. If the 'ErrorCode' parameter is set, an english error-message for this specific error will be returned.

147.7 ErrorLine

Syntax

```
Result = ErrorLine()
```

Description

Returns the line number in the source code where the current error occurred. This command only returns a meaningful value if there was an error handled by `OnErrorCall()` or `OnErrorGoto()`. The tracking of line numbers needs to be enabled on compilation for this command to return the actual line number. To enable this feature, enable the "Enable OnError lines support" checkbox in the compiler options or specify the `/LINENUMBERING` (Windows) or `-linenumbering` (Linux, Mac OSX) command-line switch when compiling from the command-line.

Parameters

None.

Return value

The line number of the error, or -1 if the OnError lines support is disabled.

147.8 ErrorFile

Syntax

```
Result\$$ = ErrorFile()
```

Description

Returns the filename of the source code or includefile where the current error occurred. This command only returns a meaningful value if there was an error handled by `OnErrorCall()` or `OnErrorGoto()`.

The tracking of line numbers needs to be enabled on compilation for this command to return the actual file name. To enable this feature, enable the "Enable OnError lines support" checkbox in the compiler options or specify the `/LINENUMBERING` (Windows) or `-linenumbering` (Linux, Mac OSX) command-line switch when compiling from the command-line.

Parameters

None.

Return value

The filename of the error, or "OnError line support disabled" if the OnError lines support is disabled. (A simple way to check if the OnError lines support is enabled is to check if the result of `ErrorLine()` is not -1.)

147.9 ErrorAddress

Syntax

```
Result = ErrorAddress()
```

Description

Returns the memory address of the assembly instruction that caused the current error. This command only returns a meaningful value if there was an error handled by `OnErrorCall()` or `OnErrorGoto()`.

Parameters

None.

Return value

The memory address of the assembly instruction that caused the current error.

147.10 ErrorTargetAddress

Syntax

```
Result = ErrorTargetAddress()
```

Description

After an error with the code `#PB_OnError_InvalidMemory`, this command returns the memory address which was read/written when the error occurred. This command has no meaning for other error codes.

Parameters

None.

Return value

The memory address which was read/written when the error occurred, after a `#PB_OnError_InvalidMemory` error.

147.11 ErrorRegister

Syntax

```
Result = ErrorRegister(Register)
```

Description

Returns the content of the specified CPU register at the time of the error. This command only returns a meaningful value if there was an error handled by `OnErrorCall()` or `OnErrorGoto()`.

Parameters

Register The available register constants depend on the CPU type for which the program is compiled. The following values are available:

x86:

```
#PB_OnError_EAX
#PB_OnError_EBX
#PB_OnError_ECX
#PB_OnError_EDX
#PB_OnError_EBP
#PB_OnError_ESI
#PB_OnError_EDI
#PB_OnError_ESP
#PB_OnError_Flags
```

x64:

```
#PB_OnError_RAX
#PB_OnError_RCX
#PB_OnError_RDX
#PB_OnError_RBX
#PB_OnError_RSP
#PB_OnError_RBP
#PB_OnError_RSI
#PB_OnError_RDI
#PB_OnError_R8
#PB_OnError_R9
...
#PB_OnError_R15
#PB_OnError_Flags
```

Return value

The content of the specified CPU register at the time of the error.

147.12 RaiseError

Syntax

```
RaiseError(ErrorNumber)
```

Description

Artificially create the given error. The appropriate error action will be taken (call of the error handler or program termination by the system if no handler is set). The ErrorNumber will be available inside the error handler with the ErrorCode() command.

Parameters

ErrorNumber On Windows, the error number can include application defined errors. The value can be between 0 and 268435455 (a 27bit number). On Linux or Mac OSX, only the following errors can be raised:

```
#PB_OnError_InvalidMemory      : Read or write operation
    on an invalid location
#PB_OnError_Floatingpoint      : Floating-point error
#PB_OnError_Breakpoint         : Debugger breakpoint
    reached (non-PureBasic breakpoints)
#PB_OnError_IllegalInstruction  : Attempt to execute an
    illegal instruction
#PB_OnError_PrivilegedInstruction: Attempt to execute a
    privileged (system-) instruction
```

Return value

None.

Example

```
1  Procedure ErrorHandler()
2      MessageRequester("OnError test", "The following error happened:
3      " + ErrorMessage())
4  EndProcedure
5
6  MessageRequester("OnError test", "Test start")
7
8  OnErrorCall(@ErrorHandler())
9  RaiseError(#PB_OnError_InvalidMemory)
10
11 MessageRequester("OnError test", "This should never be displayed")
```

See Also

ErrorCode()

147.13 ExamineAssembly

Syntax

```
Result = ExamineAssembly(*Address [, *EndAddress])
```

Description

Initializes the disassembling at the given address or address range.

Important: The disassembly commands use the [Udis86 disassembler library](#) to decode the instructions. This library is released under the BSD license which can be viewed here . If `ExamineAssembly()` and the related commands are used in a program that is to be made public, the above linked licence text must be included with the software.

Parameters

***Address** The address of the first instruction to disassemble.

***EndAddress (optional)** If specified, the disassembling will end (`NextInstruction()` will return zero) as soon as `*EndAddress` is reached. If not specified, the disassembling will run until `NextInstruction()` is no longer called.

Return value

Nonzero if disassembling is possible, zero otherwise.

Example

```
1  DisableDebugger ; do not disassemble any debugger related
   instructions
2
3  Code_Start:
4      ; Place code to be disassembled here
5      a = (Random(100) * 5) + 2000
6  Code_End:
7
8  Text$ = "Disassembled code: " + Chr(13)
9  If ExamineAssembly(?Code_Start, ?Code_End)
10     While NextInstruction()
11         Text$ + RSet(Hex(InstructionAddress()), SizeOf(Integer)*2,
12             "0")
13         Text$ + " " + InstructionString() + Chr(13)
14     Wend
15 EndIf
16 MessageRequester("Result", Text$)
```

See Also

`NextInstruction()` , `InstructionAddress()` , `InstructionString()`

147.14 NextInstruction

Syntax

```
Result = NextInstruction()
```

Description

Disassembles the next instruction after a call to `ExamineAssembly()` . Information about the disassembled instruction can be read with `InstructionString()` and `InstructionAddress()` .

Parameters

None.

Return value

Nonzero if the instruction was successfully disassembled, zero otherwise ('*EndAddress' that was specified in `ExamineAssembly()` was reached).

See Also

`ExamineAssembly()` , `InstructionAddress()` , `InstructionString()`

147.15 InstructionAddress

Syntax

```
Result = InstructionAddress()
```

Description

Returns the address of the instruction that was disassembled by a call to `NextInstruction()` .

Parameters

None.

Return value

The address of the instruction that was disassembled by a call to `NextInstruction()` .

See Also

`NextInstruction()` , `InstructionString()`

147.16 InstructionString

Syntax

```
Result\$_ = InstructionString()
```

Description

Returns a string representation of the instruction that was disassembled by a call to `NextInstruction()` .

Parameters

None.

Return value

A string representation of the instruction that was disassembled by a call to `NextInstruction()` .

See Also

NextInstruction() , InstructionAddress()

Chapter 148

Packer

Overview

The packer library provides an efficient set of functions to compress and uncompress data and manage archives. Several packer formats are supported from Zip to Lzma.

148.1 AddPackFile

Syntax

```
Result = AddPackFile(#Pack, Filename$, PackedFilename$)
```

Description

Add and compress the file to the specified pack previously created with CreatePack(). Adding a large file can take a long time.

Parameters

#Pack The pack to use.

Filename\$ The filename\$ to add to the pack.

PackedFilename\$ The filename to use in the archive to store the file.

Return value

Nonzero if the file has been successfully added to the packfile. If the file can't be compressed, it will be stored as is in the packfile.

Example: Using a tree file structure

```
1 UseZipPacker()
2
3 Path$ = #PB_Compiler_Home + "examples/sources/" ; path to the
  PureBasic examples data/media files
4 PackFile$ = GetTemporaryDirectory() + "PureBasicTest.zip" ; path
  to the pack file which should be created and opened later
5 If CreatePack(0, PackFile$)
6     AddPackFile(0, Path$ + "Data/world.png", "Data/world.png")
7     AddPackFile(0, Path$ + "Data/test.pref", "Data/test.pref")
8     AddPackFile(0, Path$ + "Data/CdPlayer.ico", "Data/CdPlayer.ico")
```

```

9      AddPackFile(0, Path$ + "2DDrawing.pb", "2DDrawing.pb")
10     ClosePack(0)
11     Debug "PackFile successfully created: " + PackFile$
12 Else
13     Debug "Error creating the pack file!"
14 EndIf
15
16 Path$ = GetTemporaryDirectory() + "PureBasicTestTree/" ; path to
    a directory into which the pack files should be extracted
17 If OpenPack(0, PackFile$)
18     If (CreateDirectory(Path$) And CreateDirectory(Path$ +
    "Data/")) Or FileSize(Path$ + "Data/") = -2
19         If ExaminePack(0)
20             Debug "Extracting archive into: " + Path$
21             While NextPackEntry(0)
22                 Debug " - name: " + PackEntryName(0) + ", size: " +
    PackEntrySize(0)
23                 If UncompressPackFile(0, Path$ + PackEntryName(0),
    PackEntryName(0)) = -1
24                     Debug "Error: unsuccessful unpacking of file: " +
    PackEntryName(0)
25                 EndIf
26             Wend
27         Else
28             Debug "Error examining the pack file!"
29         EndIf
30     ClosePack(0)
31     RunProgram(Path$) ; open the directory with the unpacked
    files
32 Else
33     Debug "Error while creating the folder to unpack the archive
    contents!"
34 EndIf
35 Else
36     Debug "Error opening the pack file!"
37 EndIf

```

For another example see `UncompressPackFile()` .

See Also

`CreatePack()`

148.2 AddPackDirectory

Syntax

```
Result = AddPackDirectory(#Pack, DirectoryName$)
```

Description

Add a new empty directory to the specified pack previously created with `CreatePack()` .

Parameters

#Pack The pack to use.

DirectoryName\$ The empty directory name to be added to the pack.

Return value

Nonzero if the empty directory has been successfully added to the packfile.

See Also

CreatePack()

148.3 AddPackMemory

Syntax

```
Result = AddPackMemory(#Pack, *Buffer, Size, PackedFilename$)
```

Description

Add and compress the memory buffer to the specified pack previously created with CreatePack() .

Parameters

#Pack The pack to use.

***Buffer** The memory buffer to add to the pack.

Size The size of the memory to add to the pack.

PackedFilename\$ The filename to use in the archive to store the memory buffer.

Return value

Nonzero if the buffer has been successfully added to the packfile. If the memory buffer can't be compressed, it will be stored as is in the packfile.

See Also

CreatePack()

148.4 ClosePack

Syntax

```
ClosePack(#Pack)
```

Description

Close the specified pack file.

Parameters

#Pack The pack to close. If **#PB_All** is specified, all the remaining packs are closed.

Return value

None.

Remarks

All remaining opened packs are automatically closed when the program ends.

148.5 CompressMemory

Syntax

```
Result = CompressMemory(*Buffer, Size, *Output, OutputSize [,  
    Plugin [, Level]])
```

Description

Compress the buffer content into the output buffer. The output buffer length needs to be at least as long as the buffer to compress.

Parameters

***Buffer** The memory buffer to compress.

Size The size of the memory to compress.

***Output** The memory buffer to store the compressed data.

OutputSize The memory buffer size to store the compressed data.

Plugin (optional) The plugin to use, if more than one packer plugin has been registered. It can be one of the following value:

```
#PB_PackerPlugin_BriefLZ: use the BriefLZ packer to compress  
    the memory. UseBriefLZPacker()  
has to be called to register the plugin.  
#PB_PackerPlugin_Zip      : use the Zip packer to compress the  
    memory. UseZipPacker()  
has to be called to register the plugin.  
#PB_PackerPlugin_Lzma     : use the Lzma packer to compress the  
    memory. UseLzmaPacker()  
has to be called to register the plugin.
```

Level (optional) The compression level to use. It is an integer value ranging from 0 (lower compression ratio, faster compression) to 9 (higher compression ratio, slower compression).

Return value

Return the compressed size if the buffer has been successfully compressed, zero otherwise. If the buffer data can't be compressed (already compressed data generally don't compress anymore), it will return zero.

See Also

UncompressMemory()

148.6 ExaminePack

Syntax

```
Result = ExaminePack(#Pack)
```

Description

Start to examine the pack content. NextPackEntry() has to be called to list the entries found in the pack file. The pack has to be previously opened with OpenPack() or CatchPack() . Packs being created with CreatePack() can not be examined.

Parameters

#Pack The pack to use.

Return value

Nonzero if the pack can be examined, zero otherwise.

Example

```
1  UseZipPacker()
2
3  ; Open the packed file
4  If OpenPack(0, "mycompressedfiles.zip")
5
6      ; List all the entries
7      If ExaminePack(0)
8          While NextPackEntry(0)
9              Debug "Name: " + PackEntryName(0) + ", Size: " +
PackEntrySize(0)
10             Wend
11         EndIf
12
13     ClosePack(0)
14 EndIf
```

See Also

OpenPack() , CatchPack() , NextPackEntry()

148.7 NextPackEntry

Syntax

```
Result = NextPackEntry(#Pack)
```

Description

Go to the next entry in the pack file. ExaminePack() has to be called before calling this command. To get more information about the current entry, use PackEntrySize() , PackEntryType() and PackEntryName() . To uncompress the current entry, use UncompressPackMemory() or UncompressPackFile() .

Parameters

#Pack The pack to use.

Return value

Nonzero if the pack contain another entry, zero otherwise.

See Also

OpenPack() , CatchPack() , PackEntrySize() , PackEntryDate() , PackEntryType() , PackEntryName() , UncompressPackMemory() , UncompressPackFile()

148.8 PackEntryDate

Syntax

```
Result.q = PackEntryDate(#Pack [, DateType])
```

Description

Returns the file date of the current pack entry, set with NextPackEntry() .

Parameters

#Pack The pack to use.

DateType The type of the date to return. It can be one of the following values:

```
#PB_Date_Created : returns the file creation date.
#PB_Date_Accessed: returns the last file access date.
#PB_Date_Modified: returns the last file modification date.
```

Return value

Returns the file date or -1 if no date is available.

See Also

OpenPack() , CatchPack() , NextPackEntry() , PackEntrySize() , PackEntryName() , PackEntryType()

148.9 PackEntryType

Syntax

```
Result = PackEntryType(#Pack)
```

Description

Returns the type of the current entry, set with NextPackEntry() .

Parameters

#Pack The pack to use.

Return value

Returns one of the following values:

```
#PB_Packer_File      : the current entry is a file
#PB_Packer_Directory: the current entry is a directory
```

See Also

OpenPack() , CatchPack() , NextPackEntry() , PackEntrySize() , PackEntryName() , PackEntryDate()

148.10 PackEntrySize

Syntax

```
Result = PackEntrySize(#Pack [, Mode])
```

Description

Returns the size of the current entry, set with NextPackEntry() .

Parameters

#Pack The pack to use.

Mode (optional) The size type to get. It can be one of the following values:

```
#PB_Packer_UncompressedSize: returns the uncompressed size of
the entry (default).
#PB_Packer_CompressedSize  : returns the compressed size of
the entry.
                                This is only supported for
BriefLZ archives.
```

Return value

Returns the size of the current entry.

See Also

OpenPack() , CatchPack() , NextPackEntry() , PackEntryType() , PackEntryName() , PackEntryDate()

148.11 PackEntryName

Syntax

```
Result\$$ = PackEntryName(#Pack)
```

Description

Returns the name of the current entry, set with NextPackEntry() .

Parameters

#Pack The pack to use.

Return value

Returns the name of the current entry.

See Also

OpenPack() , CatchPack() , NextPackEntry() , PackEntryType() , PackEntrySize() , PackEntryDate()

148.12 CreatePack

Syntax

```
Result = CreatePack(#Pack, Filename$ [, Plugin [, Level]])
```

Description

Create a new empty pack file. If the file already exists, it will be replaced with a new empty file. Before creating a pack file, at least one packer has to be registered with one of the following command: UseZipPacker() , UseBriefLZPacker() .

Parameters

#Pack A number to identify the new pack file. #PB_Any can be used as a parameter to auto-generate this number.

Filename\$ The filename for the new pack file.

Plugin (optional) The plugin to use, if more than one packer plugin has been registered. It can be one of the following value:

```
#PB_PackerPlugin_BriefLZ: use the BriefLZ packer to create
the pack file. UseBriefLZPacker()
has to be called to register the plugin.
#PB_PackerPlugin_Zip      : use the Zip packer to create the
pack file. UseZipPacker()
has to be called to register the plugin.
#PB_PackerPlugin_Lzma     : use the Lzma packer to create the
pack file (also known as 7z archive). UseLzmaPacker()
has to be called to register the plugin.
#PB_PackerPlugin_Tar      : use the Tar packer to create the
pack file. UseTarPacker()
has to be called to register the plugin.
                           It can be combined with
#PB_Packer_Gzip or #PB_Packer_Bzip2 to create compressed tar
archive.
```

Level (optional) The compression level to use. It is an integer value ranging from 0 (lower compression ratio, faster compression) to 9 (higher compression ratio, slower compression).

Return value

Returns nonzero if the pack file has been successfully created, zero otherwise. If `#PB_Any` was used as the `#Pack` parameter then the generated pack number is returned.

```
1  UseZipPacker()
2
3  ; Create the packed file
4  If CreatePack(0, "mycompressedfiles.zip")
5
6      ; add your files
7      AddPackFile(0, "Image1.bmp", "Image1.bmp")
8      AddPackFile(0, "Image2.bmp", "Image2.bmp")
9      AddPackFile(0, "mywave1.wav", "mywave1.wav")
10     AddPackFile(0, "mywave2.wav", "mywave2.wav")
11     ClosePack(0)
12 EndIf
```

See Also

ClosePack()

148.13 OpenPack

Syntax

```
Result = OpenPack(#Pack, Filename$ [, Plugin])
```

Description

Open a previously existing pack file. Before opening a pack file, at least one packer has to be registered with one of the following command: UseZipPacker(), UseLzmaPacker(), UseTarPacker(), UseBriefLZPacker(). Once opened, the pack content can be listed with ExaminePack().

Parameters

#Pack A number to identify pack file to open. `#PB_Any` can be used as a parameter to auto-generate this number.

Filename\$ The filename for the pack file to open.

Plugin (optional) The plugin to use, if more than one packer plugin has been registered. It can be one of the following value:

```
#PB_PackerPlugin_BriefLZ: use the BriefLZ packer to open the
                           pack file. UseBriefLZPacker()
has to be called to have any effect.
#PB_PackerPlugin_Zip     : use the Zip packer to open the pack
                           file. UseZipPacker()
has to be called to have any effect.
#PB_PackerPlugin_Lzma    : use the Lzma packer to open the
                           pack file. UseLzmaPacker()
has to be called to have any effect.
#PB_PackerPlugin_Tar     : use the Tar packer to open the pack
                           file. UseTarPacker()
has to be called to have any effect.
```

Return value

Returns nonzero if the pack file has been successfully opened, zero otherwise. If `#PB_Any` was used as the `#Pack` parameter then the generated pack number is returned.

```
1  UseZipPacker()
2
3  ; Open the packed file
4  If OpenPack(0, "mycompressedfiles.zip")
5
6      ; List all the entries
7      If ExaminePack(0)
8          While NextPackEntry(0)
9              Debug "Name: " + PackEntryName(0) + ", Size: " +
PackEntrySize(0)
10             Wend
11         EndIf
12
13     ClosePack(0)
14 EndIf
```

See Also

ClosePack() , ExaminePack() , CatchPack()

148.14 CatchPack

Syntax

```
Result = CatchPack(#Pack, *MemoryAddress, Size [, Plugin])
```

Description

Open a previously existing pack file from memory. Before opening a pack file, at least one packer has to be registered with one of the following command: UseZipPacker() , UseLzmaPacker() , UseTarPacker() , UseBriefLZPacker() . Once opened, the pack content can be listed with ExaminePack() .

Parameters

#Pack A number to identify pack file to open. `#PB_Any` can be used as a parameter to auto-generate this number.

***MemoryAddress** The memory address where the pack file is located.

Size The size of the pack file in bytes.

Plugin (optional) The plugin to use, if more than one packer plugin has been registered. It can be one of the following value:

```
#PB_PackerPlugin_BriefLZ: use the BriefLZ packer to open the
pack file. UseBriefLZPacker()
has to be called to have any effect.
#PB_PackerPlugin_Zip      : use the Zip packer to open the pack
file. UseZipPacker()
has to be called to have any effect.
#PB_PackerPlugin_Lzma     : use the Lzma packer to open the
pack file. UseLzmaPacker()
```

```

has to be called to have any effect.
#PB_PackerPlugin_Tar      : use the Tar packer to open the pack
    file. UseTarPacker()
has to be called to have any effect.

```

Return value

Returns nonzero if the pack file has been successfully opened, zero otherwise. If `#PB_Any` was used as the `#Pack` parameter then the generated pack number is returned.

See Also

ClosePack() , ExaminePack() , OpenPack()

148.15 UncompressMemory

Syntax

```

Result = UncompressMemory(*Buffer, Size, *Output, OutputSize [,
    Plugin])

```

Description

Uncompress the buffer content into the output buffer. The output buffer length needs to be at least as long as the buffer to uncompress.

Parameters

***Buffer** The memory buffer to uncompress.

Size The size of the memory to uncompress.

***Output** The memory buffer to store the uncompressed data.

OutputSize The memory buffer size to store the uncompressed data. It must be at least of the size of the uncompressed data.

Plugin (optional) The plugin to use, if more than one packer plugin has been registered. It can be one of the following value:

```

#PB_PackerPlugin_BriefLZ: use the BriefLZ packer to
    uncompress the memory. UseBriefLZPacker()
has to be called to have any effect.
#PB_PackerPlugin_Zip      : use the Zip packer to uncompress
    the memory. UseZipPacker()
has to be called to have any effect.
#PB_PackerPlugin_Lzma     : use the Lzma packer to uncompress
    the memory. UseLzmaPacker()
has to be called to have any effect.
#PB_PackerPlugin_Jcalg1  : use the Jcalg1 packer to uncompress
    the memory. UseJcalg1Packer()
has to be called to have any effect.

```

Return value

Return the uncompressed size if the buffer has been successfully uncompressed, -1 otherwise.

See Also

CompressMemory()

148.16 UncompressPackMemory

Syntax

```
Result = UncompressPackMemory(#Pack, *Buffer, Size [,  
    PackedFilename$])
```

Description

Uncompress into the memory buffer from the current pack entry being examined with ExaminePack() and NextPackEntry() .

Parameters

#Pack The pack to use.

***Buffer** The memory buffer to uncompress the pack entry.

Size The size of the memory to uncompress the pack entry.

PackedFilename\$ (optional) The packed filename entry to uncompress. If this parameter is not specified, the current entry being examined with ExaminePack() and NextPackEntry() is uncompressed.

Return value

Return the uncompressed size if the pack entry has been successfully uncompressed into the memory buffer, -1 otherwise.

See Also

OpenPack() , CatchPack() , ExaminePack() , NextPackEntry()

148.17 UncompressPackFile

Syntax

```
Result = UncompressPackFile(#Pack, Filename$ [, PackedFilename$])
```

Description

Uncompress into the specified filename from the current pack entry being examined with ExaminePack() and NextPackEntry() . If the filename already exists, it will be erased and replaced with the new uncompressed data.

Parameters

#Pack The pack to use.

Filename\$ The filename to uncompress the current pack entry.

PackedFilename\$ (optional) The packed filename entry to uncompress. If this parameter is not specified, the current entry being examined with ExaminePack() and NextPackEntry() is uncompressed.

Return value

Return the uncompressed size if the pack entry has been successfully uncompressed into the filename, -1 otherwise.

Example

```
1  UseZipPacker()
2
3  Path$ = #PB_Compiler_Home + "examples/sources/Data/" ; path to
   the PureBasic examples data/media files
4  PackFile$ = GetTemporaryDirectory() + "PureBasicTest.zip" ; path
   to the pack file which should be created and opened later
5  If CreatePack(0, PackFile$)
6      AddPackFile(0, Path$ + "world.png", "world.png")
7      AddPackFile(0, Path$ + "test.pref", "test.pref")
8      AddPackFile(0, Path$ + "CdPlayer.ico", "CdPlayer.ico")
9      AddPackFile(0, Path$ + "Background.bmp", "Background.bmp")
10     ClosePack(0)
11     Debug "PackFile successfully created: " + PackFile$
12 Else
13     Debug "Error creating the pack file!"
14 EndIf
15
16 Path$ = GetTemporaryDirectory() + "PureBasicTest/" ; path to a
   directory into which the pack files should be extracted
17 If OpenPack(0, PackFile$)
18     If CreateDirectory(Path$) Or FileSize(Path$) = -2
19         If ExaminePack(0)
20             Debug "Extracting archive into: " + Path$
21             While NextPackEntry(0)
22                 Debug " - name: " + PackEntryName(0) + ", size: " +
PackEntrySize(0)
23                 If UncompressPackFile(0, Path$ + PackEntryName(0),
PackEntryName(0)) = -1
24                     Debug "Error: unsuccessful unpacking of file: " +
PackEntryName(0)
25                 EndIf
26             Wend
27         EndIf
28         ClosePack(0)
29         RunProgram(Path$) ; open the directory with the unpacked
files
30     Else
31         Debug "Error while creating the folder '" + Path$ + "' to
unpack the archive contents!"
32     EndIf
33 Else
34     Debug "Error opening the pack file!"
35 EndIf
```

For another example see AddPackFile() .

See Also

OpenPack() , CatchPack() , ExaminePack() , NextPackEntry()

148.18 UseZipPacker

Syntax

```
UseZipPacker()
```

Description

Enable Zip compress, uncompress and archive support to the packer library. The created pack will be compatible with other Zip archives in 2.0 format. The created archive size can be up to 2GB. For more information: [Wikipedia - Zip](#).

Parameters

None.

Return value

None.

See Also

OpenPack() , CreatePack() , CompressMemory() , UncompressMemory()

148.19 UseLzmaPacker

Syntax

```
UseLzmaPacker()
```

Description

Enable Lzma compress, uncompress and 7z archive support to the packer library. Lzma compression is considered as one of the best available multipurpose compression algorithm. It provides very good compress ratio and fast uncompress. Compressing can be slow. For more information: [Wikipedia - Lzma](#).

Parameters

None.

Return value

None.

See Also

OpenPack() , CompressMemory() , UncompressMemory()

148.20 UseTarPacker

Syntax

`UseTarPacker()`

Description

Enable Tar compress, uncompress and Tar archive support to the packer library. Bzip2 and Gzip compression are both supported. Compressing and uncompressing Tar archives are usually fast.

Parameters

None.

Return value

None.

See Also

`OpenPack()` , `CompressMemory()` , `UncompressMemory()`

148.21 UseBriefLZPacker

Syntax

`UseBriefLZPacker()`

Description

Enable BriefLZ compress, uncompress and archive support to the packer library. The created archives are a custom format created for PureBasic. BriefLZ compression is very fast and the packer is very small, it could be the right choice for program which needs a small executable size. Uncompress is also very fast, but compress ratio is not as good as Zip or Lzma.

Parameters

None.

Return value

None.

See Also

`OpenPack()` , `CreatePack()` , `CompressMemory()` , `UncompressMemory()`

148.22 UseJcalg1Packer

Syntax

```
UseJcalg1Packer()
```

Description

Enable Jcalg1 uncompress support to the packer library. This is an old algorithm which was used in previous version of PureBasic, so it is still available to allow to support old compressed files. Compression and archive support is no more available. This packer is only available on Windows x86 (32-bit). This packer is deprecated and no more supported.

Parameters

None.

Return value

None.

See Also

UncompressMemory()

Supported OS

Windows

148.23 IsPack

Syntax

```
Result = IsPack(#Pack)
```

Description

Tests if the given pack number is a valid and correctly initialized pack.

Parameters

#Pack The pack to test.

Return value

Returns nonzero if #Pack is a valid pack and zero if not.

Remarks

This function is bulletproof and can be used with any value. This is the correct way to ensure an pack is ready to use.

See Also

CreatePack() , OpenPack()

Chapter 149

Particle

Overview

Particles systems are widely used in the 3D scenes to simulate hard to predict events, like the rain, fire, explosions etc. PureBasic allows to create any number of particle emitters as needed which are running in an autonomous manner. Each emitter has its own shape, and its own properties like the velocity, speed, emission rate and more.

InitEngine3D() must be called successfully before using any of the Particle functions.

149.1 CreateParticleEmitter

Syntax

```
Result = CreateParticleEmitter(#ParticleEmitter, Width, Height,  
    Depth, Type, [, x.f, y.f, z.f])
```

Description

Creates a new empty particle emitter of the given size.

Parameters

#ParticleEmitter A number to identify the new particle emitter. #PB_Any can be used to auto-generate this number.

Width, Height, Depth The size of the new particle emitter (respectively x,y and z dimension). The default size of the future Particles can be specified with the ParticleSize() function.

Type It can be one of the following value:

```
#PB_Particle_Point: The emitter is a single point  
#PB_Particle_Box  : The emitter is a box, with an width,  
    height and depth
```

x, y, z (optional) The initial position of the particle emitter in the world.

Return value

Nonzero if the particle emitter has been successfully created, zero otherwise. If #PB_Any was used as #ParticleEmitter parameter then the auto-generated number of the new particle emitter will be returned.

See Also

FreeParticleEmitter()

149.2 IsParticleEmitter

Syntax

```
Result = IsParticleEmitter(#ParticleEmitter)
```

Description

Tests if the given particle emitter is valid and correctly initialized.

Parameters

#ParticleEmitter The particle emitter to test.

Return value

Nonzero if the particle emitter is valid, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a particle emitter is ready to use.

See Also

CreateParticleEmitter()

149.3 DisableParticleEmitter

Syntax

```
DisableParticleEmitter(#ParticleEmitter, State)
```

Description

Enables or disables the particle emitter. When disabled, no more particle are emitted.

Parameters

#ParticleEmitter The particle emitter to use.

State It can take the following values:

```
#True : The particle emitter is disabled.  
#False: The particle emitter is enabled.
```

Return value

None.

149.4 ParticleEmitterID

Syntax

```
ParticleEmitterID = ParticleEmitterID(#ParticleEmitter)
```

Description

Returns the unique system identifier of the particle emitter.

Parameters

#ParticleEmitter The particle emitter to use.

Return value

The unique system identifier of the particle emitter.

149.5 ParticleEmitterX

Syntax

```
Result = ParticleEmitterX(#ParticleEmitter [, Mode])
```

Description

Returns the particle emitter 'x' position in the world.

Parameters

#ParticleEmitter The particle emitter to use.

Mode (optional) The mode to get the 'x' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'x' position of the particle  
emitter in the world (default).  
#PB_Relative: get the 'x' position of the particle emitter  
relative to its parent.
```

Return value

The particle emitter 'x' position in the world (according to the given 'Mode').

See Also

ParticleEmitterY() , ParticleEmitterZ()

149.6 ParticleEmitterY

Syntax

```
Result = ParticleEmitterY(#ParticleEmitter [, Mode])
```

Description

Returns the particle emitter 'y' position in the world.

Parameters

#ParticleEmitter The particle emitter to use.

Mode (optional) The mode to get the 'y' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'y' position of the particle
               emitter in the world (default).
#PB_Relative: get the 'y' position of the particle emitter
               relative to its parent.
```

Return value

The particle emitter 'y' position in the world (according to the given 'Mode').

See Also

ParticleEmitterX() , ParticleEmitterZ()

149.7 ParticleEmitterZ

Syntax

```
Result = ParticleEmitterZ(#ParticleEmitter [, Mode])
```

Description

Returns the particle emitter 'z' position in the world.

Parameters

#ParticleEmitter The particle emitter to use.

Mode (optional) The mode to get the 'z' position. It can be one of the following value:

```
#PB_Absolute: get the absolute 'z' position of the particle
               emitter in the world (default).
#PB_Relative: get the 'z' position of the particle emitter
               relative to its parent.
```

Return value

The particle emitter 'z' position in the world (according to the given 'Mode').

See Also

ParticleEmitterX() , ParticleEmitterY()

149.8 ParticleEmitterAngle

Syntax

```
ParticleEmitterAngle(#ParticleEmitter, Angle.f)
```

Description

Changes the emitted particle angles. All particles will have the same angle.

Parameters

#ParticleEmitter The particle emitter to use.

Angle.f The new particle angle, in degree.

Return value

None.

149.9 ParticleEmissionRate

Syntax

```
ParticleEmissionRate(#ParticleEmitter, Rate)
```

Description

Changes the #ParticleEmitter emission rate.

Parameters

#ParticleEmitter The particle emitter to use.

Rate The new particle emission rate, in particles per second.

Return value

None.

149.10 ParticleMaterial

Syntax

```
ParticleMaterial(#ParticleEmitter, MaterialID)
```

Description

Assigns a material to the specified particle emitter. This material will be used by all the particles of this emitter. An emitter can only have one material assigned at once.

Parameters

#ParticleEmitter The particle emitter to use.

MaterialID The new material to use for this particle emitter. A valid 'MaterialID' can be easily obtained with the MaterialID() function.

Return value

None.

See Also

MaterialID()

149.11 ParticleTimeToLive

Syntax

```
ParticleTimeToLive(#ParticleEmitter, MinimumTime, MaximumTime)
```

Description

Change the particles time to live for the particle emitter. Every particle will live at least 'MinimumTime' and at most the 'MaximumTime' (in world time unit). A random value, within this range, will be used by every particle emitted.

Parameters

#ParticleEmitter The particle emitter to use.

MinimumTime The minimum time for a particle before being freed.

MaximumTime The maximum time for a particle before being freed.

Return value

None.

149.12 ParticleVelocity

Syntax

```
ParticleVelocity(#ParticleEmitter, Minimum, Maximum)
```

Description

Changes the particles velocity. A random velocity value, picked in the 'Minimum' and 'Maximum' range, will be used by every particle emitted.

Parameters

#ParticleEmitter The particle emitter to use.

Minimum The minimum velocity for a particle.

Maximum The maximum velocity for a particle.

Return value

None.

149.13 ParticleAcceleration

Syntax

```
ParticleAcceleration(#ParticleEmitter, x.f, y.f, z.f)
```

Description

Changes the particles acceleration vector. It can be useful to simulate gravity, wind, etc.

Parameters

#ParticleEmitter The particle emitter to use.

x, y, z The acceleration (force) vector to apply to all particles.

Return value

None.

149.14 ParticleSize

Syntax

```
ParticleSystem(#ParticleEmitter, Width, Height)
```

Description

Change the particles dimensions. Particles are 2D planes (billboards) which always face the camera. All particles in an emitter always have the same size.

Parameters

#ParticleEmitter The particle emitter to use.

Width, Size The new size of the particles, in world unit.

Return value

None.

149.15 ParticleColorRange

Syntax

```
ParticleSystem(#ParticleEmitter, StartColor, EndColor)
```

Description

Changes the particles color range for the particle emitter. Every emitted particle will get a random value, within the 'StartColor' and 'EndColor' range (gradient between these two color).

Parameters

#ParticleEmitter The particle emitter to use.

StartColor The first color for the particle. RGB() can be used to get a valid color value.

EndColor The last color for the particle. RGB() can be used to get a valid color value.

Return value

None.

149.16 ParticleColorFader

Syntax

```
ParticleColorFader(#ParticleEmitter, RedRate.f, GreenRate.f,  
                  BlueRate.f, AlphaRate.f)
```

Description

Changes the particles color fader rate.

Parameters

#ParticleEmitter The particle emitter to use.

RedRate, GreenRate, BlueRate, AlphaRate The color channels fader rate to apply, per second. Examples values and their effect:

```
0: no change will be applied  
-1: will subtract 256 to the color component every second.  
1: will add 256 to the color component every second.  
-0.25: will subtract 64 to the color component every second.
```

Return value

None.

149.17 FreeParticleEmitter

Syntax

```
FreeParticleEmitter(#ParticleEmitter)
```

Description

Frees the particle emitter. All its associated memory is released and this object can't be used anymore.

Parameters

#ParticleEmitter The particle emitter to free. If **#PB_All** is specified, all the remaining particle emitters are freed.

Return value

None.

Remarks

All remaining particle emitters are automatically freed when the program ends.

149.18 HideParticleEmitter

Syntax

```
HideParticleEmitter(#ParticleEmitter, State)
```

Description

Hides or shows the specified particle emitter.

Parameters

#ParticleEmitter The particle emitter to use.

State It can take the following values:

```
#True : the particle emitter is hidden
#False: the particle emitter is shown
```

Return value

None.

149.19 MoveParticleEmitter

Syntax

```
MoveParticleEmitter(#ParticleEmitter, x.f, y.f, z.f [, Mode])
```

Description

Move the specified particle emitter.

Parameters

#ParticleEmitter The particle emitter to move.

x, y, z The new position of the particle emitter.

Mode It can be one of the following values:

```
#PB_Relative: relative move, from the current particle
               emitter position (default).
#PB_Absolute: absolute move to the specified position.
```

combined with one of the following values:

```
#PB_Local : local move.
#PB_Parent: move relative to the parent position.
#PB_World : move relative to the world.
```

Return value

None.

See Also

ParticleEmitterX() , ParticleEmitterY() , ParticleEmitterZ()

149.20 ParticleEmitterDirection

Syntax

```
ParticleEmitterDirection(#ParticleEmitter, x.f, y.f, z.f)
```

Description

Changes the direction of the particle emitter according to the specified x,y,z values.

Parameters

#ParticleEmitter The particle emitter to use.

x, y, z The new direction vector of the particle emitter.

Return value

None.

149.21 ResizeParticleEmitter

Syntax

```
ResizeParticleEmitter(#ParticleEmitter, Width, Height, Depth)
```

Description

Resizes the particle emitter to the new specified 'Width', 'Height' and 'Depth' dimension.

Parameters

#ParticleEmitter The particle emitter to resize.

Width, Height, Depth The new size of the particle emitter (respectively x,y and z dimension).

Return value

None.

149.22 GetScriptParticleEmitter

Syntax

```
Result = GetScriptParticleEmitter(#ParticleEmitter, Name$)
```

Description

Gets a particle emitter defined in an OGRE script file.

Parameters

#ParticleEmitter A number to identify the new particle emitter. **#PB_Any** can be used to auto-generate this number.

Name\$ The name of the particle emitter in the OGRE script file. This name is case-sensitive.

Return value

Zero if the particle emitter isn't found or can't be loaded from script files. If `#PB_Any` is used as `'#ParticleEmitter'` parameter, the new particle emitter number is returned.

149.23 ParticleSpeedFactor

Syntax

```
ParticleSpeedFactor(#ParticleEmitter, Factor.f)
```

Description

Changes the current particle emitter speed factor.

Parameters

#ParticleEmitter The particle emitter to use.

Factor The new particle emitter speed factor. The default speed factor is 1. When using a value greater than 1, then the emitting speed will be increased. When using a value lower than 1, then the emitting speed will be decreased. For example, using a speed factor of 4.5, will have the particle emitting speed multiplied by 4.5. Using a speed factor of 0.5, will have the particle emitting speed divided by 2.

Return value

None.

149.24 ParticleScaleRate

Syntax

```
ParticleScaleRate(#ParticleEmitter, Rate.f)
```

Description

Changes the current particle emitter scale rate.

Parameters

#ParticleEmitter The particle emitter to use.

Rate The new particle emitter scale rate. The default rate is 0, which means the particle will be the same size. When using a value greater than 0, the particle size will be increase over time (each second, the rate will be added to the current size). When using a value lower than 0, the particle size will be decreased over time (each second, the rate will be subtracted from the current size).

For example:

A rate of 10 will increase the particle size of 10 units per second.

A rate of -2 will decrease the particle size of 2 units per second.

Return value

None.

149.25 ParticleAngle

Syntax

```
ParticleAngle(#ParticleEmitter, RangeStart.f, RangeEnd.f [,  
             SpeedRangeStart.f, SpeedRangeEnd.f])
```

Description

Changes the particle angle when emitted.

Parameters

#ParticleEmitter The particle emitter to use.

RangeStart, RangeEnd The particle range (in degree) between 0 and 360. For example a range of 45 to 90 will only emit particle with an random angle between 45 and 90 degrees.

SpeedRangeStart, SpeedRangeEnd Specify the rotation speed range of the emitted particle. A random rotation speed will be picked from this range when a new particle is emitted.

Remarks

To avoid display glitches, the particle material should be configured to be non-repeatable using `SetMaterialAttribute(Material, #PB_Material_TAM, #PB_Material_ClampTAM)`.

Return value

None.

Chapter 150

Preference

Overview

Preference file contains user defined program parameters stored on the disk and read back when the program start again (like '.INI' files under Windows for example). PureBasic offers the possibility to create hierarchical preference file easily usable on any computer platform. The file format is in unicode UTF-8 with a BOM one preference per line, using the 'Keyword = Value' syntax. Groups can be created for better reading.

150.1 ClosePreferences

Syntax

```
ClosePreferences()
```

Description

Closes a preference file previously opened with OpenPreferences() or created with CreatePreferences() .

Parameters

None.

Return value

None.

Example

```
1 ; Open a preference file
2 OpenPreferences(#PB_Compiler_Home
3   +"examples/sources/Data/test.pref")
4 ; Choose the group "Window"
5 PreferenceGroup("Window")
6
7 ; Display the "w" key of this group
8 MessageRequester("Info", "w = " + ReadPreferenceLong ("w", 0))
9
10 ; Close the preference file
11 ClosePreferences()
```

See Also

CreatePreferences() , OpenPreferences()

150.2 CreatePreferences

Syntax

```
Result = CreatePreferences(Filename$ [, Flags])
```

Description

Creates a new empty preference file. If the file already exists, the file is erased.

Parameters

Filename\$ The filename of the new preference file.

Flags (optional) It can be a combination of the following values:

```
#PB_Preference_NoSpace      : no spaces will be put around  
the equal sign joining key and values.  
It can be useful when dealing  
with external preferences files which doesn't  
accept spaces around equal  
sign.  
#PB_Preference_GroupSeparator: add an empty line between the  
groups to ease readability of the file.  
#PB_Preference_NoBOM        : Don't add the UTF-8 BOM when  
writing the file. The file will still be written using UTF-8  
encoding.
```

Return value

Nonzero if the file has been successfully created, zero otherwise.

Remarks

Once created, the functions like WritePreferenceString() can be used to write data. When values are written in preferences, it's possible to read them back with the functions like ReadPreferenceString() .

To remove a key or a group, use RemovePreferenceKey() and RemovePreferenceGroup() . To create or change the current group, use PreferenceGroup() .

Once all write operations are done, ClosePreferences() needs to be called to really write the preferences back to disk.

Example

```
1 ; Create a preference file named Setup.ini  
2 CreatePreferences(GetTemporaryDirectory()+"Setup.ini")  
3  
4 ; Create a group named "Window"  
5 PreferenceGroup("Window")  
6 WritePreferenceLong("X", 10) ; X = 10  
7 WritePreferenceLong("Y", 10) ; Y = 10  
8 WritePreferenceLong("W", 800) ; W = 800
```

```

9      WritePreferenceLong("H", 600)          ; H = 600
10     WritePreferenceFloat("%", 20)          ; % = 20.000000
11     WritePreferenceString("Title", "PureNote") ; Title = "PureNote"
12
13     ; Close the preference file
14     ClosePreferences()

```

See Also

ClosePreferences() , ProgramParameter()

150.3 ExaminePreferenceGroups

Syntax

```
Result = ExaminePreferenceGroups()
```

Description

Starts the enumeration of all the groups found in the current preference file. NextPreferenceGroup() can be used to list all the group found.

Parameters

None.

Return value

Nonzero if the enumeration has been successfully started, zero otherwise.

Example

```

1      ; Open a preference file
2      OpenPreferences(#PB_Compiler_Home
3                      +"examples/sources/Data/test.pref")
4
5      ; Examine groups
6      ExaminePreferenceGroups()
7      While NextPreferenceGroup() ; While group exists
8          MessageRequester("Groups", PreferenceGroupName()) ; Display
9          this group
10     Wend
11
12     ; Close the preference file
13     ClosePreferences()

```

See Also

NextPreferenceGroup()

150.4 ExaminePreferenceKeys

Syntax

```
Result = ExaminePreferenceKeys()
```

Description

Starts the enumeration of all the keys found in the current group of the preference file. The current group can be selected with `PreferenceGroup()` or by examining all groups with `ExaminePreferenceGroups()`. `NextPreferenceKey()` can be used to list all the keys found.

Parameters

None.

Return value

Nonzero if the enumeration has been successfully started, zero otherwise.

Example

```
1  ; Open a preference file
2  OpenPreferences(#PB_Compiler_Home
   + "examples/sources/Data/test.pref")
3
4  ; Select a group
5  PreferenceGroup("Window")
6
7  ; Examine keys in the group 'Window'
8  ExaminePreferenceKeys()
9  While NextPreferenceKey() ; While a key exists
10     Debug "Key group 'Window': " + PreferenceKeyName() + " = " +
        PreferenceKeyValue() ; Display the key with its data
11 Wend
12
13 ; Close the preference file
14 ClosePreferences()
```

See Also

`ExaminePreferenceGroups()` , `NextPreferenceGroup()` , `NextPreferenceKey()`

150.5 FlushPreferenceBuffers

Syntax

```
Result = FlushPreferenceBuffers()
```

Description

Ensures that all preferences changes are written to disk.

Parameters

None.

Return value

Nonzero if the preferences has been successfully written the disk. If an error occurred (ie: disk full, disk error), it will return zero.

See Also

CreatePreferences() , OpenPreferences()

150.6 NextPreferenceGroup

Syntax

```
Result = NextPreferenceGroup()
```

Description

Retrieves information about the next group found in the enumeration started with ExaminePreferenceGroups() . To get the name of the group, use PreferenceGroupName() . The current examined preference group will also be used when values are read from the preferences or when the keys are examined with ExaminePreferenceKeys() .

Parameters

None.

Return value

Nonzero if more groups are available in the current enumeration, zero otherwise.

Example

```
1  ; Open a preference file
2  OpenPreferences(#PB_Compiler_Home
   + "examples/sources/Data/test.pref")
3
4  ; Examine groups
5  ExaminePreferenceGroups()
6  While NextPreferenceGroup() ; While group exists
7      MessageRequester("Groups", PreferenceGroupName()) ; Display
   this group
8  Wend
9
10 ; Close the preference file
11 ClosePreferences()
```

See Also

ExaminePreferenceGroups() , PreferenceGroupName() , ExaminePreferenceKeys()

150.7 NextPreferenceKey

Syntax

```
Result = NextPreferenceKey()
```

Description

Retrieves information about the next key found in the enumeration started with `ExaminePreferenceKeys()`. To get the name and the value of the key, use `PreferenceKeyName()` and `PreferenceKeyValue()`.

Parameters

None.

Return value

Nonzero if more keys are available in the current enumeration, zero otherwise.

Example

```
1  ; Open a preference file
2  OpenPreferences(#PB_Compiler_Home
   + "examples/sources/Data/test.pref")
3
4  ; Select a group
5  PreferenceGroup("Window")
6
7  ; Examine keys in the group 'Window'
8  ExaminePreferenceKeys()
9  While NextPreferenceKey() ; While a key exists
10     MessageRequester("Key group 'Window'", PreferenceKeyName() + "
   = " + PreferenceKeyValue()) ; Display the key with its data
11 Wend
12
13 ; Close the preference file
14 ClosePreferences()
```

See Also

`ExaminePreferenceKeys()`, `PreferenceKeyName()`, `PreferenceKeyValue()`

150.8 PreferenceGroupName

Syntax

```
Group\$ = PreferenceGroupName()
```

Description

Returns the name of the current group being enumerated with `ExaminePreferenceGroups()` or previously selected with `PreferenceGroup()`.

Parameters

None.

Return value

The name of the current group being enumerated with `ExaminePreferenceGroups()` or previously selected with `PreferenceGroup()` .

Example

```
1  ; Open a preference file
2  OpenPreferences(#PB_Compiler_Home
   + "examples/sources/Data/test.pref")
3
4  ; Examine groups
5  ExaminePreferenceGroups()
6  While NextPreferenceGroup() ; While group exists
7      MessageRequester("Groups", PreferenceGroupName()) ; Display
   this group
8  Wend
9
10 ; Close the preference file
11 ClosePreferences()
```

See Also

`ExaminePreferenceGroups()`

150.9 PreferenceKeyName

Syntax

```
Key\$ = PreferenceKeyName()
```

Description

Returns the name of the current key being enumerated with `ExaminePreferenceKeys()` . To get the value of the key, use `PreferenceKeyValue()` .

Parameters

None.

Return value

The name of the current key being enumerated with `ExaminePreferenceKeys()` .

Example

```
1 ; Open a preference file
2 OpenPreferences(#PB_Compiler_Home
   +"examples/sources/Data/test.pref")
3
4 ; Select a group
5 PreferenceGroup("Window")
6
7 ; Examine keys in the group 'Window'
8 ExaminePreferenceKeys()
9 While NextPreferenceKey() ; While a key exists
10     MessageRequester("Key group 'Window'", PreferenceKeyName() + "
    = " + PreferenceKeyValue()) ; Display the key with its data
11 Wend
12
13 ; Close the preference file
14 ClosePreferences()
```

See Also

ExaminePreferenceKeys() , PreferenceKeyValue()

150.10 PreferenceKeyValue

Syntax

Value\\$_ = PreferenceKeyValue()

Description

Returns the value, in string form, of the current key being enumerated with ExaminePreferenceKeys(). To get the name of the key, use PreferenceKeyName() .

Parameters

None.

Return value

The value, in string form, of the current key being enumerated with ExaminePreferenceKeys() .

Example

```
1 ; Open a preference file
2 OpenPreferences(#PB_Compiler_Home
   +"examples/sources/Data/test.pref")
3
4 ; Select a group
5 PreferenceGroup("Window")
6
7 ; Examine keys in the group 'Window'
8 ExaminePreferenceKeys()
9 While NextPreferenceKey() ; While a key exists
```



```

10 |     MessageRequester("Key group 'Window'", PreferenceKeyName() + "
    |     = " + PreferenceKeyValue()) ; Display the key with its data
11 | Wend
12 |
13 | ; Close the preference file
14 | ClosePreferences()

```

For another example (the keys are the data themselves), see WritePreferenceString() .

See Also

ExaminePreferenceKeys() , PreferenceKeyName()

150.11 OpenPreferences

Syntax

```
Result = OpenPreferences(Filename$ [, Flags [, Encoding]])
```

Description

Opens a previously existing preference file.

Parameters

Filename\$ The filename of the preference file.

Flags (optional) It can be a combination of the following values:

```

#PB_Preference_NoSpace: no spaces will be put around the
                        equal sign joining key and values.
                        It can be useful when dealing with
                        external preferences files which doesn't
                        accept spaces around equal sign.
#PB_Preference_GroupSeparator: add an empty line between the
                        groups to ease readability of the file.

```

Encoding (optional) To force a string encoding when reading the preference file. If an UTF-8 BOM is found in at the start of the file, this parameter is ignored and UTF-8 will be used. It can be one of the following value:

```

#PB_UTF8 : The file will be read as UTF-8, even if it doesn't
           have a BOM.
#PB_Ascii: The file will be read as Ascii, unless there an
           UTF-8 BOM. If the preferences are written back to file, it
           will be using UTF-8.

```

Return value

Nonzero if the file has been successfully opened, zero otherwise.

Remarks

If the file wasn't found or can't be opened, it is still possible to use the read functions and it will return the specified default value. It is very useful to initialize in one step the program variables. The functions like `ReadPreferenceString()` can be used to read the preference values stored in the file.

To remove a key or a group, use `RemovePreferenceKey()` and `RemovePreferenceGroup()`. To create or change the current group, use `PreferenceGroup()`.

It is possible to change existing values with `WritePreferenceString()` and similar functions.

Once all write operations are done, `ClosePreferences()` needs to be called to really write the preferences back to disk (if it has been modified inbetween).

Example

```
1  ; Open a preference file
2  OpenPreferences(#PB_Compiler_Home
3      +"examples/sources/Data/test.pref")
4
5  ; Examine Groups
6  ExaminePreferenceGroups()
7  ; For each group
8  While NextPreferenceGroup()
9      text$ = text$ + PreferenceGroupName() + #LF$ ; its name
10     ; Examine keys for the current group
11     ExaminePreferenceKeys()
12     ; For each key
13     While NextPreferenceKey()
14         text$ = text$ + PreferenceKeyName() + " = " +
15         PreferenceKeyValue() + #LF$ ; its name and its data
16     Wend
17     text$ = text$ + #LF$
18 Wend
19
20 ; Display all groups and all keys with data
21 MessageRequester("test.pref", text$)
22
23 ; Close the preference file
24 ClosePreferences()
```

See Also

`ClosePreferences()`

150.12 PreferenceGroup

Syntax

```
Result = PreferenceGroup(Name$)
```

Description

Creates a new group (in the form: [Name\$]) or changes the current group in the preference file. All following read or write operations will be restricted to this group. To move outside of any groups, an empty 'Name\$' can be used.

Parameters

Name\$ The new group name.

Return value

Nonzero if the group was already created, zero otherwise.

Remarks

If the group does not exist yet, it will not be created immediately. It will instead be created once the first key is written to it. This allows to use the `PreferenceGroup()` function to test if a group exists without creating many empty groups. The group search is case-insensitive.

Example

```
1  ; Open a preference file
2  OpenPreferences(#PB_Compiler_Home
3      +"examples/sources/Data/test.pref")
4
5  ; Select a group
6  PreferenceGroup("Window")
7
8  ; Examine keys in the group 'Window'
9  ExaminePreferenceKeys()
10 While NextPreferenceKey() ; While a key exists
11     MessageRequester("Keys from the 'Window' group",
12         PreferenceKeyName() + " = " + PreferenceKeyValue()) ; Display
13     the key with its data
14 Wend
15
16 ; Close the preference file
17 ClosePreferences()
```

150.13 PreferenceComment

Syntax

```
PreferenceComment(Text$)
```

Description

Writes a new comment line in the current preference file.

Parameters

Text\$ The new comment to write.

Return value

None.

Example

```
1 ; Creating the file prefs.txt
2 If CreatePreferences(GetTemporaryDirectory()+"Prefs.txt")
3
4     PreferenceComment("The coordinates of the window.") ; Write a
comment
5     PreferenceComment("") ; Write a comment (empty line)
6
7     PreferenceGroup("Window")
8     WritePreferenceLong ("X", 100)
9     WritePreferenceLong ("Y", 125)
10
11     ClosePreferences()
12
13     RunProgram(GetTemporaryDirectory()+"Prefs.txt")
14 EndIf
```

150.14 ReadPreferenceDouble

Syntax

```
Result.d = ReadPreferenceDouble(Key$, DefaultValue)
```

Description

Try to read the specified associated 'Key\$' value.

Parameters

Key\$ The key to read the value from. If the key is inside a group, PreferenceGroup() has to be used to change the current group.

DefaultValue The default value to return if the key isn't found or the preference file haven't been opened correctly (file missing for example).

Return value

The value associated to the specified key, as a double number. if the key isn't found the default value is returned.

Example

```
1 ; Open a preference file
2 OpenPreferences(#PB_Compiler_Home
+ "examples/sources/Data/test.pref")
3
4 ; Open the group 'Window'
5 PreferenceGroup("Window")
6
7 ; Examine keys until w = 800
8 NextPreferenceKey() ; x = 0
9 NextPreferenceKey() ; y = 0
10 NextPreferenceKey() ; w = 800
11
12 ; Read the key
```

```

13  key$ = PreferenceKeyName()
14
15  ; Display the key and its data with different format
16  MessageRequester("Integer key", Str(ReadPreferenceInteger(key$,
17    0)))
17  MessageRequester("Float key", StrF(ReadPreferenceFloat(key$, 0),
18    6))
18  MessageRequester("Double key", StrD(ReadPreferenceDouble(key$,
19    0), 15))
19  MessageRequester("Long key", Str(ReadPreferenceLong(key$, 0)))
20  MessageRequester("Quad key", Str(ReadPreferenceQuad(key$, 0)))
21  MessageRequester("String key", ReadPreferenceString(key$, "0"))
22
23  ; Close the preference file
24  ClosePreferences()

```

See Also

ReadPreferenceFloat() , ReadPreferenceInteger() , ReadPreferenceLong() , ReadPreferenceQuad()
 , ReadPreferenceString()

150.15 ReadPreferenceFloat

Syntax

```
Result.f = ReadPreferenceFloat(Key$, DefaultValue)
```

Description

Try to read the specified associated 'Key\$' value.

Parameters

Key\$ The key to read the value from. If the key is inside a group, PreferenceGroup() has to be used to change the current group.

DefaultValue The default value to return if the key isn't found or the preference file haven't been opened correctly (file missing for example).

Return value

The value associated to the specified key, as a float number. if the key isn't found the default value is returned.

Example

See the example at ReadPreferenceDouble()

See Also

ReadPreferenceDouble() , ReadPreferenceInteger() , ReadPreferenceLong() ,
 ReadPreferenceQuad() , ReadPreferenceString()

150.16 ReadPreferenceInteger

Syntax

```
Result = ReadPreferenceInteger(Key$, DefaultValue)
```

Description

Try to read the specified associated 'Key\$' value.

Parameters

Key\$ The key to read the value from. If the key is inside a group, PreferenceGroup() has to be used to change the current group.

DefaultValue The default value to return if the key isn't found or the preference file haven't been opened correctly (file missing for example).

Return value

The value associated to the specified key, as an integer number. if the key isn't found the default value is returned.

Example

See the examples at WritePreferenceInteger() and ReadPreferenceDouble() .

See Also

ReadPreferenceDouble() , ReadPreferenceFloat() , ReadPreferenceLong() , ReadPreferenceQuad()
, ReadPreferenceString()

150.17 ReadPreferenceLong

Syntax

```
Result = ReadPreferenceLong(Key$, DefaultValue)
```

Description

Try to read the specified associated 'Key\$' value.

Parameters

Key\$ The key to read the value from. If the key is inside a group, PreferenceGroup() has to be used to change the current group.

DefaultValue The default value to return if the key isn't found or the preference file haven't been opened correctly (file missing for example).

Return value

The value associated to the specified key, as a long number. if the key isn't found the default value is returned.

Example

See the example at `ReadPreferenceDouble()`

See Also

`ReadPreferenceDouble()` , `ReadPreferenceFloat()` , `ReadPreferenceInteger()` ,
`ReadPreferenceQuad()` , `ReadPreferenceString()`

150.18 `ReadPreferenceQuad`

Syntax

```
Result.q = ReadPreferenceQuad(Key$, DefaultValue)
```

Description

Try to read the specified associated 'Key\$' value.

Parameters

Key\$ The key to read the value from. If the key is inside a group, `PreferenceGroup()` has to be used to change the current group.

DefaultValue The default value to return if the key isn't found or the preference file haven't been opened correctly (file missing for example).

Return value

The value associated to the specified key, as a quad number. if the key isn't found the default value is returned.

Example

See the example at `ReadPreferenceDouble()`

See Also

`ReadPreferenceDouble()` , `ReadPreferenceFloat()` , `ReadPreferenceInteger()` ,
`ReadPreferenceLong()` , `ReadPreferenceString()`

150.19 `ReadPreferenceString`

Syntax

```
Result\[extract_itex] = ReadPreferenceString(Key[/extract_itex], DefaultValue[/extract_itex])
```

Description

Try to read the specified associated 'Key\$' value.

Parameters

Key\$ The key to read the value from. If the key is inside a group, `PreferenceGroup()` has to be used to change the current group.

DefaultValue The default value to return if the key isn't found or the preference file haven't been opened correctly (file missing for example).

Return value

The value associated to the specified key, as a string. if the key isn't found the default value is returned.

Example

See the example at `ReadPreferenceDouble()`

See Also

`ReadPreferenceDouble()` , `ReadPreferenceFloat()` , `ReadPreferenceInteger()` ,
`ReadPreferenceLong()` , `ReadPreferenceQuad()`

150.20 RemovePreferenceGroup

Syntax

```
RemovePreferenceGroup(Group$)
```

Description

Removes the specified 'Group\$' and all its keys.

Parameters

Group\$ The group name to remove.

Return value

None.

Example

```
1 ; Create the preference file prefs.txt
2 If CreatePreferences(GetTemporaryDirectory()+"Prefs.txt",
   #PB_Preference_GroupSeparator)
3
4 PreferenceGroup("Window")
5 WritePreferenceLong ("X", 100)
6 WritePreferenceLong ("Y", 125)
7 WritePreferenceString("Title", "PureNote")
8
9 PreferenceGroup("event")
10 WritePreferenceString("percentage", "percentage")
11 WritePreferenceFloat("%", 100)
12
```



```

13     ClosePreferences()
14
15     RunProgram(GetTemporaryDirectory()+"Prefs.txt")
16 EndIf
17
18 MessageRequester("Info", "Delete the group 'event' ")
19
20 ; Open a preference file
21 OpenPreferences(GetTemporaryDirectory()+"Prefs.txt")
22
23 ; Delete the group 'event'
24 RemovePreferenceGroup("event")
25
26 ClosePreferences()
27
28 RunProgram(GetTemporaryDirectory()+"Prefs.txt")

```

See Also

PreferenceGroup() , ExaminePreferenceGroups()

150.21 RemovePreferenceKey

Syntax

```
RemovePreferenceKey(Key$)
```

Description

Removes the specified key and its value.

Parameters

Key\$ The key name to remove.

Return value

None.

Example

```

1  If CreatePreferences(GetTemporaryDirectory()+"Prefs.txt",
2     #PB_Preference_GroupSeparator)
3     PreferenceGroup("Window")
4     WritePreferenceLong ("X", 100)
5     WritePreferenceLong ("Y", 125)
6     WritePreferenceString("Title", "PureNote")
7     ClosePreferences()
8     RunProgram(GetTemporaryDirectory()+"Prefs.txt")
9 EndIf
10
11 MessageRequester("Info", "Delete the key 'Title' ")
12
13 ; Open the preference file
14 OpenPreferences(GetTemporaryDirectory()+"Prefs.txt")

```

```

14 PreferenceGroup("Window")
15
16 ; Delete the key 'Title'
17 RemovePreferenceKey("Title")
18
19 ClosePreferences()
20 RunProgram(GetTemporaryDirectory()+"Prefs.txt")

```

See Also

ExaminePreferenceKeys()

150.22 WritePreferenceFloat

Syntax

```
WritePreferenceFloat(Key$, Value.f)
```

Description

Creates or changes the specified key and its associated float value under the form: 'Key\$ = Value' in the preference file, previously created with CreatePreferences() or opened with OpenPreferences() .

Parameters

Key\$ The key name to write. If PreferenceGroup() has been used, then the write is restricted to the current group. If the key was already created, its associated value is replaced with the new specified value.

Value The float value to associate with the key.

Return value

None.

Example

```

1 ; Create the preference file prefs.txt
2 If CreatePreferences(GetTemporaryDirectory()+"Prefs.txt",
  #PB_Preference_GroupSeparator)
3   PreferenceGroup("Window")
4     WritePreferenceString("Title", "PureNote")
5     WritePreferenceLong ("X", 100)
6     WritePreferenceLong ("Y", 125)
7     WritePreferenceInteger("I", 1024)
8     WritePreferenceQuad("Q", 9223372036854775807)
9     WritePreferenceFloat("%", 20.10)
10    WritePreferenceDouble("D", 0.0123456789 )
11    ClosePreferences()
12    RunProgram(GetTemporaryDirectory()+"Prefs.txt")
13 EndIf

```

See Also

WritePreferenceDouble() , WritePreferenceInteger() , WritePreferenceLong() ,
WritePreferenceQuad() , WritePreferenceString()

150.23 WritePreferenceDouble

Syntax

```
WritePreferenceDouble(Key$, Value.d)
```

Description

Creates or changes the specified key and its associated double value under the form: 'Key\$ = Value' in the preference file, previously created with CreatePreferences() or opened with OpenPreferences() .

Parameters

Key\$ The key name to write. If PreferenceGroup() has been used, then the write is restricted to the current group. If the key was already created, its associated value is replaced with the new specified value.

Value The double value to associate with the key.

Return value

None.

Example

See the example at WritePreferenceFloat()

See Also

WritePreferenceFloat() , WritePreferenceInteger() , WritePreferenceLong() ,
WritePreferenceQuad() , WritePreferenceString()

150.24 WritePreferenceInteger

Syntax

```
WritePreferenceInteger(Key$, Value)
```

Description

Creates or changes the specified key and its associated integer value under the form: 'Key\$ = Value' in the preference file, previously created with CreatePreferences() or opened with OpenPreferences() .

Parameters

Key\$ The key name to write. If PreferenceGroup() has been used, then the write is restricted to the current group. If the key was already created, its associated value is replaced with the new specified value.

Value The integer value to associate with the key.

Return value

None.

Example

```
1  ; Create a 'test.ini' preference file in the users temp folder.
2  ; Since we don't store many values, we don't create
   PreferenceGroups...
3  If CreatePreferences(GetTemporaryDirectory()+"test.ini")
4      WritePreferenceInteger("x", 35)
5      WritePreferenceInteger("y", 46)
6      WritePreferenceInteger("w", 200)
7      WritePreferenceInteger("h", 100)
8      ClosePreferences()
9  Else
10     Debug "Error: Can't create the 'test.ini' file!"
11 EndIf
12
13 ; Read that file and output the values
14 If OpenPreferences(GetTemporaryDirectory()+"test.ini")
15     Debug ReadPreferenceInteger("x", 0)
16     Debug ReadPreferenceInteger("y", 0)
17     Debug ReadPreferenceInteger("w", 400)
18     Debug ReadPreferenceInteger("h", 600)
19     ClosePreferences()
20 Else
21     Debug "Error: Can't open the 'test.ini' file!"
22 EndIf
```

For another example see at `WritePreferenceFloat()` .

See Also

`WritePreferenceFloat()` , `WritePreferenceDouble()` , `WritePreferenceLong()` ,
`WritePreferenceQuad()` , `WritePreferenceString()`

150.25 WritePreferenceLong

Syntax

```
WritePreferenceLong(Key$, Value)
```

Description

Creates or changes the specified key and its associated long value under the form: 'Key\$ = Value' in the preference file, previously created with `CreatePreferences()` or opened with `OpenPreferences()` .

Parameters

Key\$ The key name to write. If `PreferenceGroup()` has been used, then the write is restricted to the current group. If the key was already created, its associated value is replaced with the new specified value.

Value The long value to associate with the key.

Return value

None.

Example

See the example at `WritePreferenceFloat()`

See Also

`WritePreferenceFloat()` , `WritePreferenceDouble()` , `WritePreferenceInteger()` ,
`WritePreferenceQuad()` , `WritePreferenceString()`

150.26 WritePreferenceQuad

Syntax

```
WritePreferenceQuad(Key$, Value.q)
```

Description

Creates or changes the specified key and its associated float value under the form: 'Key\$ = Value' in the preference file, previously created with `CreatePreferences()` or opened with `OpenPreferences()` .

Parameters

Key\$ The key name to write. If `PreferenceGroup()` has been used, then the write is restricted to the current group. If the key was already created, its associated value is replaced with the new specified value.

Value The quad value to associate with the key.

Return value

None.

Example

See the example at `WritePreferenceFloat()`

See Also

`WritePreferenceFloat()` , `WritePreferenceDouble()` , `WritePreferenceInteger()` ,
`WritePreferenceLong()` , `WritePreferenceString()`

150.27 WritePreferenceString

Syntax

```
WritePreferenceString(Key$, Value$)
```

Description

Creates or changes the specified key and its associated string value under the form: 'Key\$ = Value' in the preference file, previously created with `CreatePreferences()` or opened with `OpenPreferences()` .

Parameters

Key\$ The key name to write. If `PreferenceGroup()` has been used, then the write is restricted to the current group. If the key was already created, its associated value is replaced with the new specified value.

Value\$ The string value to associate with the key.

Return value

None.

Example: "Standard"

See the example at `WritePreferenceFloat()` .

Example: "Use the preference keys for storing the data"

```
1      ; Create a 'ColorList.ini' preference file
2      ; Note: In this case the data values must be unique (no duplicate
      keys).
3      If CreatePreferences(GetTemporaryDirectory()+"ColorList.ini",
      #PB_Preference_NoSpace)
4          WritePreferenceString("FF0000", "")
5          WritePreferenceString("00FF00", "")
6          WritePreferenceString("0000FF", "")
7          WritePreferenceString("FF00FF", "")
8          ClosePreferences()
9      Else
10         Debug "Error: Can't create the 'ColorList.ini' file!"
11     EndIf
12
13     ; Read the 'ColorList.ini' file and output the keys (which are
      the data here)
14     If OpenPreferences(GetTemporaryDirectory()+"ColorList.ini")
15         ExaminePreferenceKeys()
16         While NextPreferenceKey()
17             Debug PreferenceKeyName()
18         Wend
19         ClosePreferences()
20     Else
21         Debug "Error: Can't read the 'ColorList.ini' file!"
22     EndIf
```

See Also

`WritePreferenceFloat()` , `WritePreferenceDouble()` , `WritePreferenceInteger()` ,
`WritePreferenceLong()` , `WritePreferenceQuad()`

Chapter 151

Printer

Overview

Printers are essential devices to transform virtual, numeric data to printed paper. Many types of software require a way to print data back to paper to be really useful. PureBasic allows to print any kind of data from raw text to pictures, in any resolution.

151.1 DefaultPrinter

Syntax

```
Result = DefaultPrinter()
```

Description

Selects the default printer as the current printer for print operation. This function has to be called before all other printer functions. Once DefaultPrinter() has been successfully called, StartPrinting() is used to actually start printing.

Parameters

None.

Return value

Nonzero if a default printer is available, zero otherwise.

See Also

StartPrinting()

151.2 NewPrinterPage

Syntax

```
NewPrinterPage()
```

Description

Creates a new empty page. The previous page is sent to the printer and can't be modified anymore. It has to be called inside a StartDrawing() /StopDrawing() block.

Parameters

None.

Return value

None.

Example

```
1  If PrintRequester()  
2      If StartPrinting("Two sheets")  
3          If StartDrawing(PrinterOutput())  
4              DrawingMode(#PB_2DDrawing_Transparent)  
5              DrawText(10, 10, "First page !", RGB(0, 0, 0))  
6  
7              ; Tell the printer to start a new page  
8              NewPrinterPage()  
9  
10             DrawText(10, 10, "Second page !", RGB(0, 0, 0))  
11  
12             StopDrawing()  
13         EndIf  
14  
15         StopPrinting()  
16     EndIf  
17 EndIf
```

151.3 PrinterOutput

Syntax

```
OutputID = PrinterOutput()
```

Description

Returns the OutputID of the current printer to be used with the StartDrawing() function. Drawing on the printer will be performed using pixel based drawing operations.

Parameters

None.

Return value

The OutputID of the current printer to be used with the StartDrawing() function.

Remarks

Drawing on a printer using pixel based drawing may reduce printing quality and adds extra complexity because the printers resolution must be taken into account. Vector based drawing using PrinterVectorOutput() should be preferred, as it provides resolution independent functions for high-quality printing.

Example

```
1 StartDrawing(PrinterOutput())
2 ; do some drawing stuff here...
3 StopDrawing()
```

See Also

StartDrawing() , PrinterVectorOutput()

151.4 PrinterVectorOutput

Syntax

VectorOutputID = `PrinterVectorOutput`([Unit])

Description

Returns the OutputID of the current printer to be used with the StartVectorDrawing() function.

Parameters

Unit (optional) Specifies the unit used for measuring distances on the drawing output. The default unit for printer output is `#PB_Unit_Point`.

```
#PB_Unit_Pixel      : Values are measured in pixels (or dots
in case of a printer)
#PB_Unit_Point      : Values are measured in points (1/72 inch)
#PB_Unit_Inch       : Values are measured in inches
#PB_Unit_Millimeter : Values are measured in millimeters
```

Return value

The OutputID of the current printer to be used with the StartVectorDrawing() function.

Example

```
1 StartVectorDrawing(PrinterVectorOutput(#PB_Unit_Point))
2 ; do some drawing stuff here...
3 StopVectorDrawing()
```

See Also

StartVectorDrawing() , PrinterOutput()

151.5 PrintRequester

Syntax

Result = `PrintRequester`()

Description

Open a regular print requester, to choose the printer and do some adjustments. This function must be called before all other printer functions. Once PrintRequester() has been successfully called, StartPrinting() is used to actually start printing.

Parameters

None.

Return value

Nonzero if the user successfully selected a printer, zero otherwise.

151.6 StartPrinting

Syntax

```
Result = StartPrinting(JobName$)
```

Description

Initializes the printer and starts the print operation.

Parameters

JobName\$ The name to be shown in the printer spooler and which identifies the new print operation.

Return value

Nonzero if the print operation has been started successfully, zero otherwise.

See Also

StopPrinting()

151.7 StopPrinting

Syntax

```
StopPrinting()
```

Description

Stop all the print operations and send the data to the printer.

Parameters

None.

Return value

None.

See Also

StartPrinting()

151.8 PrinterPageWidth

Syntax

```
Result = PrinterPageWidth()
```

Description

Return the width of the drawing area.

Parameters

None.

Return value

The width of the drawing area, in pixels. The number changes with the DPI of the printer. This means that a document printed at 75 DPI will have a drawing area 4 times smaller than a document printed at 150 DPI. This value refers to the client area, the one you can write on. It doesn't count the hardware printer margins, but doesn't add any 'soft' margins.

See Also

PrinterPageHeight()

151.9 PrinterPageHeight

Syntax

```
Result = PrinterPageHeight()
```

Description

Returns the height of the drawing area.

Parameters

None.

Return value

The height of the drawing area, in pixels. The number changes with the DPI of the printer. This means that a document printed at 75 DPI will have a drawing area 4 times smaller than a document printed at 150 DPI. This value refers to the client area, the one you can write on. It doesn't count the hardware printer margins, but doesn't add any 'soft' margins.

See Also

PrinterPageWidth()

Chapter 152

Process

Overview

The process library allows the programmer the ability to get information about the current program, as well as to execute other programs and communicate with them. This provides a platform for independent access to a programs environment block, the programs parameters, as well as standard input and output.

152.1 AvailableProgramOutput

Syntax

```
Result = AvailableProgramOutput(Program)
```

Description

Returns the number of bytes available to be read from the programs output.

Parameters

Program The program to use. It must have been started before with RunProgram() using the `#PB_Program_Read` flag.

Return value

The number of bytes available to be read from the programs output.

Remarks

The output may be read with either ReadProgramString() or ReadProgramData() . However, ReadProgramString() or ReadProgramData() remains blocked indefinitely if the program requests the intervention of the user. For example, return a yes/no choice to the program to continue.

See Also

ReadProgramString() , ReadProgramData()

152.2 CloseProgram

Syntax

```
CloseProgram(Program)
```

Description

Closes the connection with the given program (which was started with RunProgram()) and frees all related data.

Parameters

Program The program to close. It must have been started before with RunProgram() .

Return value

None.

Remarks

This does not terminate the program, it only closes the connection with it. To terminate the program, call KillProgram() first. Also if the program terminated normally, this function must still be called to properly release all data.

If the program was started with the #PB_Program_Write flag, CloseProgram() will cause the program to receive an EOF (end of file) on its standard input. This condition can also be produced without directly closing the connection to the program by calling WriteProgramData() with the special #PB_Program_Eof value.

See Also

KillProgram() , RunProgram()

152.3 CountProgramParameters

Syntax

```
Result = CountProgramParameters()
```

Description

Returns the number of parameters specified on the command line (or via RunProgram()).

Parameters

None.

Return value

The number of parameters specified on the command line (or via RunProgram()).

Remarks

ProgramParameter() may be used to read the individual parameters.

See Also

ProgramParameter()

152.4 EnvironmentVariableName

Syntax

```
Result\$ = EnvironmentVariableName()
```

Description

Returns the name of the current environment variable currently examined with `ExamineEnvironmentVariables()` and `NextEnvironmentVariable()`. `EnvironmentVariableValue()` may be used to return its value.

Parameters

None.

Return value

The name of the current environment variable currently examined with `ExamineEnvironmentVariables()` and `NextEnvironmentVariable()`.

See Also

`ExamineEnvironmentVariables()`, `NextEnvironmentVariable()`, `EnvironmentVariableValue()`

152.5 EnvironmentVariableValue

Syntax

```
Result\$ = EnvironmentVariableValue()
```

Description

Returns the value of the current environment variable currently examined with `ExamineEnvironmentVariables()` and `NextEnvironmentVariable()`. `EnvironmentVariableName()` may be used to return its name.

Parameters

None.

Return value

The value of the current environment variable currently examined with `ExamineEnvironmentVariables()` and `NextEnvironmentVariable()`.

See Also

`ExamineEnvironmentVariables()`, `NextEnvironmentVariable()`, `EnvironmentVariableName()`

152.6 ExamineEnvironmentVariables

Syntax

`Result = ExamineEnvironmentVariables()`

Description

Starts to examine the environment block of the program.

`NextEnvironmentVariable()` , `EnvironmentVariableName()` and `EnvironmentVariableValue()` may be used to read the individual environment variables.

Parameters

None.

Return value

Nonzero if the environment block can be read, zero otherwise.

Example

```
1 ; Will output all environment variables of the program
2 ;
3 OpenConsole()
4 If ExamineEnvironmentVariables()
5     While NextEnvironmentVariable()
6         PrintN(EnvironmentVariableName() + " = " +
7             EnvironmentVariableValue())
8     Wend
9 EndIf
10 PrintN("")
11 PrintN("Press Enter to quit.")
12 Input()
```

See Also

`NextEnvironmentVariable()` , `EnvironmentVariableName()` , `EnvironmentVariableValue()`

152.7 GetEnvironmentVariable

Syntax

`Result\$ = GetEnvironmentVariable(Name$)`

Description

Returns the content of the specified environment variable from the programs environment block.

Parameters

Name\$ The name of the environment variable to get.

Return value

The environment variable value, or an empty string if the environment variable does not exist.

Example

```
1 ; Display the content of the "PATH" environment variable
2 ;
3 Debug GetEnvironmentVariable("PATH")
```

See Also

SetEnvironmentVariable()

152.8 IsProgram

Syntax

```
Result = IsProgram(Program)
```

Description

Tests if the given program is a program that is executed by the RunProgram() function.

Parameters

Program The program to test.

Return value

Nonzero if the program is executed by the RunProgram() function, zero otherwise.

Remarks

This function is bulletproof and may be used with any value.

See Also

RunProgram()

152.9 KillProgram

Syntax

```
KillProgram(Program)
```

Description

Immediately terminates the given program (which was previously started with RunProgram()).

Parameters

Program The program to kill. It must have been started before with RunProgram() .

Return value

None.

Remarks

This terminates the program, but it does not close the connection with the program object of the terminated program. `CloseProgram()` must still be called to properly free all data associated with the program.

See Also

`CloseProgram()`

152.10 NextEnvironmentVariable

Syntax

```
Result = NextEnvironmentVariable()
```

Description

This function must be called after `ExamineEnvironmentVariables()` . It will go step-by-step through the environment variables of the program.

Parameters

None.

Return value

Nonzero if there is still more variables to read, zero otherwise.

Remarks

Use `EnvironmentVariableName()` and `EnvironmentVariableValue()` to get the name and content of the current variable.

See Also

`ExamineEnvironmentVariables()` , `EnvironmentVariableName()` , `EnvironmentVariableValue()`

152.11 ProgramExitCode

Syntax

```
Result = ProgramExitCode(Program)
```

Description

Returns the exitcode that was returned when the given program ended.

Parameters

Program The program to use. It must have been started before with RunProgram() .

Return value

The exitcode that was returned when the specified program ended.

Remarks

This function should only be used after the given program has ended. Use ProgramRunning() or WaitProgram() to make sure of that.

The exitcode allows for a single value to be returned from the program, back to the program that executed it. This is usually used to indicate either the error or the success of the program. Please note, that due to OS limitation on OS X and Linux the supported exitcode are in the range of 0 - 255.

To return an exitcode from a PB program, use the optional value with the [End](#) statement:

```
1  End 1      ; returns the exitcode 1
```

152.12 ProgramFilename

Syntax

```
Result\$_ = ProgramFilename()
```

Description

Returns the full path and filename of this program and may be used to find where the program is installed or the executable name. GetPathPart() or GetFilePart() may be used to get the path or filename of the program from the return string.

Parameters

None.

Return value

The full path and filename of this program.

If used inside a DLL, this function returns the path and filename of the DLL, not the main program that loaded it.

See Also

GetPathPart() , GetFilePart()

152.13 ProgramID

Syntax

```
Result = ProgramID(Program)
```

Description

Returns the unique system identifier for the given program. This is the so called "Process ID" or "PID".

Parameters

Program The program to use. It must have been started before with RunProgram() .

Return value

The unique system identifier for the given program.

If the identifier cannot be returned, -1 will be returned.

This will happen if RunProgram() is used to open a file in another program, like

RunProgram("Source.pb").

Note: the value returned here is not a handle (unlike with most other ...ID() functions). Rather, it is the Process ID which may be viewed within the Taskmanager. To get a process handle, use the OpenProcess_() API.

Note: The returned value may be false if the process is launched by a wrapper as is the case with Windows 10 and UWP (Universal Windows Platform) applications launched by ApplicationFrameHost.exe (AFH).

152.14 ProgramParameter

Syntax

```
Result\$_ = ProgramParameter([Index])
```

Description

Gets the next parameter string that was passed to the executable when it was launched.

Parameters

Index (optional) If specified, the parameter for that index will be returned. The first parameter index starts from 0.

Return value

The next parameter value, or an empty string if no more parameters are found. If index is specified it will return the specified parameter value.

Remarks

This function is especially useful for console programs , where the user passes one or more parameter at the program start.

Note: Relying on the return of an empty string to detect the last parameter is not a good practice, since the function also returns an empty string if an empty string was passed in "" on the command-line. The preferred method which should be used to get all parameters, is to get the count with CountProgramParameters() and then to call ProgramParameter() as often as needed. To check the behaviour of this function, or do several tests, please read the 'Executable command-line' paragraph in the page IDE Compiler options .

Example

1

```
MyProgram.exe MyText.txt /FAST "Special Mode"
```

The first time that ProgramParameter() is called, it will return "MyText.txt", the second time "/FAST" and the third time "Special Mode".

Example: Call the compiled program from command-line with the file to display as parameter

```
1  If OpenWindow(0, 0, 0, 520, 510, "Example...",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      EditorGadget(0, 5, 5, 510, 500, #PB_Editor_WordWrap)
4      path$ = ProgramParameter()
5      If FileSize(path$) >= 0
6          If ReadFile(0, path$)
7              Text$ = ReadString(0, #PB_File_IgnoreEOL)
8              SetGadgetText(0, Text$)
9          EndIf
10     Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
11 EndIf
```

See Also

CreatePreferences()

152.15 ProgramRunning

Syntax

```
Result = ProgramRunning(Program)
```

Description

Tests if the specified program is still running.

Parameters

Program The program to use. It must have been started before with RunProgram() .

Return value

Nonzero as long as the program has not yet ended, zero otherwise. If the program has been run with #PB_Program_Read flag, it will return nonzero as long there is something to read, even if the program is already ended.

See Also

RunProgram()

152.16 ReadProgramData

Syntax

```
Result = ReadProgramData(Program, *Buffer, Size)
```

Description

Reads data from the given programs output (stdout) and puts it in the specified buffer. This function waits until there is data available to read from the program. To prevent this wait, AvailableProgramOutput() may be used first to check if there is something to read.

Parameters

Program The program to use. It must have been started before with `RunProgram()` and the `#PB_Program_Read` flag.

***Buffer** The memory buffer to read the data into. A memory buffer can be created for example with `AllocateMemory()` .

Size The size to read, in bytes. The buffer should be large enough to handle this size.

Return value

The number of bytes actually read.

Remarks

The function reads up to 'size' bytes, but also less if not that much is available (but it will not return zero bytes because it always waits for some data to read).

However, the function remains blocked indefinitely if the program requests the intervention of the user. For example, return a yes / no choice to the program to continue.

See Also

`ReadProgramString()`

152.17 ReadProgramError

Syntax

```
Result\$ = ReadProgramError(Program [, Flags])
```

Description

Reads a line from the specified programs error output (stderr). Unlike `ReadProgramData()` , this function doesn't halt the program flow if no error output is available.

Parameters

Program The program to use. It must have been started before with `RunProgram()` and the `#PB_Program_Error` flag.

Flags (optional) The string format to use when reading the error output. The default format can be affected with the `#PB_Program_Ascii`, `#PB_Program_Unicode` and `#PB_Program_UTF8` flags for `RunProgram()` . This can be one of the following values:

```
#PB_Ascii    : Reads the error output as ascii
#PB_UTF8     : Reads the error output as UTF8 (default)
#PB_Unicode  : Reads the error output as unicode
```

Return value

The error string, or an empty string if there is no error output.

152.18 ReadProgramString

Syntax

```
Result\$$ = ReadProgramString(Program [, Flags])
```

Description

Reads a line from the output (stdout) of the given program. This function waits until there is data available to read from the program. To prevent this wait, AvailableProgramOutput() may be used first to check if there is something to read. This function also waits until a full line of output is available. If not line-based or raw output is to be read, ReadProgramData() may be used.

Parameters

Program The program to use. It must have been started before with RunProgram() and the #PB_Program_Read flag.

Flags (optional) The string format to use when reading the output. The default format can be affected with the #PB_Program_Ascii, #PB_Program_Unicode and #PB_Program_UTF8 flags for RunProgram(). This can be one of the following values:

```
#PB_Ascii    : Reads the output as ascii
#PB_UTF8     : Reads the output as UTF8 (default)
#PB_Unicode  : Reads the output as unicode
```

Return value

A string created from program output.

Remarks

However, the function remains blocked indefinitely if the program requests the intervention of the user. For example, return a yes / no choice to the program to continue.

See Also

ReadProgramData()

152.19 RemoveEnvironmentVariable

Syntax

```
RemoveEnvironmentVariable(Name$)
```

Description

Removes the given environment variable from the programs environment block.

Parameters

Name\$ The environment variable to remove.

Return value

None.

152.20 RunProgram

Syntax

```
Result = RunProgram(Filename$ [, Parameter$, WorkingDirectory$ [,  
    Flags [, SenderProgram]])
```

Description

Launches an external program.

Parameters

Filename\$ The executable name including its path.

Parameters\$ (optional) Specifies the command line parameters that will be passed to the program.

WorkingDirectory\$ (optional) Specifies the directory that will then be the current directory for the launched program.

Flags (optional) It can be a combination (using '|' OR operator) of the following values:

```
#PB_Program_Wait      : Wait until the launched program quits  
#PB_Program_Hide      : Launch the program in invisible mode  
#PB_Program_Open      : Open the program to communicate with it  
                        or get information about it.  
#PB_Program_Read      : Read the programs console output.  
                        (stdout)  
#PB_Program_Write     : Write to the input of the program.  
                        (stdin)  
#PB_Program_Error     : Read the error output of the program.  
                        (stderr)  
#PB_Program_Connect   : Connect another programs output to this  
                        programs input.  
#PB_Program_Ascii     : Default read/write operation are in  
                        ASCII.  
#PB_Program_Unicode   : Default read/write operation are in  
                        Unicode.  
#PB_Program_UTF8      : Default read/write operation are in UTF8  
                        (default).
```

A program executed with `#PB_Program_Open` must be closed with `CloseProgram()`. The 'Read', 'Write', 'Error' and 'Connect' flags require the `#PB_Program_Open` flag to be set as well.

When using the `#PB_Program_Connect` flag, another program must have been started before with `#PB_Program_Open` and `#PB_Program_Read`. The number returned when running this program must be passed as the 'SenderProgram' parameter to `RunProgram()`. The following functions may be used with a program that has been executed with the `#PB_Program_Open` flag:

- `IsProgram()` : check if the program number represents a valid program executed by `RunProgram()`.
- `ProgramID()` : returns the OS ProcessID for the program.
- `ProgramRunning()` : check if the program is still running.
- `WaitProgram()` : wait for the program to end.
- `KillProgram()` : terminate the program.
- `ProgramExitCode()` : get the exitcode of the program.
- `CloseProgram()` : close the connection to the program.

The following functions may be used for programs executed with the 'Read', 'Write' and 'Error' flags:

- AvailableProgramOutput() : check if the programs output is available.
- ReadProgramString() : read a string from the programs output.
- ReadProgramData() : read data from the programs output.
- ReadProgramError() : read a string from the programs error output.
- WriteProgramString() : write a string to the programs input.
- WriteProgramData() : write data to the programs input.

SenderProgram (optional) Needed in conjunction with the `#PB_Program_Connect` flag, must contain the returned number of another program started before with `RunProgram()` and the `#PB_Program_Open` and `#PB_Program_Read` flags. The output of the sender program will be sent directly to the input of the now executed program. Several programs may be connected in this way, to 'pipe' data through that group of connected programs.

Return value

Nonzero if the program has been successfully launched, zero otherwise.

If `#PB_Program_Open` was included in the Flags, the return-value is a number that identifies the program. It may be used in calls to get information about the program such as `ReadProgramString()` or `ProgramExitCode()` or other functions mentioned above.

Example

```

1  ; Executes the PB compiler with the /? option and displays the
   output
2  ;
3  Compiler = RunProgram(#PB_Compiler_Home+"compilers/pbcompiler",
   "-h", "", #PB_Program_Open | #PB_Program_Read)
4  Output$ = ""
5  If Compiler
6      While ProgramRunning(Compiler)
7          If AvailableProgramOutput(Compiler)
8              Output$ + ReadProgramString(Compiler) + Chr(13)
9          EndIf
10     Wend
11     Output$ + Chr(13) + Chr(13)
12     Output$ + "Exitcode: " + Str(ProgramExitCode(Compiler))
13
14     CloseProgram(Compiler) ; Close the connection to the program
15 EndIf
16
17 MessageRequester("Output", Output$)

```

On Windows `RunProgram()` uses the default application associated with the file type of the given file. An example: `RunProgram("Test.html")` will open the web browser, which is generally used for displaying web sites on your system.

152.21 SetEnvironmentVariable

Syntax

```
SetEnvironmentVariable(Name$, Value$)
```

Description

Creates an environment variable in the environment block of this program with given name and value. If a variable with this name already existed, its content will be changed to the new value.

The environment block of the program is passed on to other programs executed with `RunProgram()` , so this method may be used to pass on information to a program executed by this program. (The executed program may use `GetEnvironmentVariable()` to read the variables.)

Parameters

Name\$ The environment variable name.

Value\$ The new value for the environment variable.

Return value

None.

See Also

`GetEnvironmentVariable()`

152.22 WaitProgram

Syntax

```
Result = WaitProgram(Program [, Timeout])
```

Description

Halts the execution of the code until the specified program has ended or the optional timeout is reached.

Parameters

Program The program to use. It must have been started before with `RunProgram()` .

Timeout (optional) The timeout to use, in milliseconds.

Return value

Nonzero if the program has ended, zero if the timeout was reached.

152.23 WriteProgramData

Syntax

```
Result = WriteProgramData(Program, *Buffer, Size)
```

Description

Writes the data from the buffer to the specified programs input (stdin).

Parameters

Program The program to use. It must have been started before with RunProgram() and the #PB_Program_Write flag.

***Buffer** The memory buffer to write the data from. The special value #PB_Program_Eof can be used instead of a real memory buffer to make the program receive an EOF (end of file) in its input (which tells the program that there will be no more input). The 'Size' parameter is ignored in this case. After calling WriteProgramData() with this special value, no more input may be written to the program.

Size The size to write.

Return value

The number of bytes actually written.

See Also

WriteProgramString() , WriteProgramStringN()

152.24 WriteProgramString

Syntax

```
WriteProgramString(Program, String$ [, Flags])
```

Description

Writes the given string to the specified programs input (stdin).

Parameters

Program The program to use. It must have been started before with RunProgram() and the #PB_Program_Write flag.

String\$ The string to write. There is no newline written to the program after the string. To include the newline, WriteProgramStringN() may be used. To write raw data to the program, WriteProgramData() may be used. This function may also be used to send an EOF (end of file) to the program which tells the program that there will be no more input.

Flags (optional) The string format to use when writing the string to the program input. The default format can be affected with the #PB_Program_Ascii, #PB_Program_Unicode and #PB_Program_UTF8 flags for RunProgram() . This can be one of the following values:

```
#PB_Ascii    : Writes the string as ascii
#PB_UTF8     : Writes the string as UTF8 (default)
#PB_Unicode  : Writes the string as unicode
```

Return value

None.

See Also

WriteProgramStringN() , ReadProgramString()

152.25 WriteProgramStringN

Syntax

```
WriteProgramStringN(Program, String$ [, Flags])
```

Description

Writes the given string to the specified programs input (stdin) with an extra newline character.

Parameters

Program The program to use. It must have been started before with RunProgram() and the `#PB_Program_Write` flag.

String\$ The string to write. An extra newline character will be appended to the string. To write a string without a newline, use WriteProgramString() . To write raw data to the program, WriteProgramData() may be used. This function may also be used to send an EOF (end of file) to the program which tells the program that there will be no more input.

Flags (optional) The string format to use when writing the string to the program input. The default format can be affected with the `#PB_Program_Ascii`, `#PB_Program_Unicode` and `#PB_Program_UTF8` flags for RunProgram() . This can be one of the following values:

```
#PB_Ascii    : Writes the string as ascii
#PB_UTF8     : Writes the string as UTF8  (default)
#PB_Unicode  : Writes the string as unicode
```

Return value

None.

See Also

WriteProgramString() , ReadProgramString()

Chapter 153

RegularExpression

Overview

Regular expressions allow to do advanced pattern matching to quickly match, extract or replace an arbitrary information in a string. These kind of expressions are often difficult to read and write, but once you master them it makes a lot of things easier. Therefore this library is not for beginners, and you need to have some solid basis with PureBasic and programming in general to be able to use this library efficiently.

This library uses PCRE, which is an open source implementation of the Perl regular expression. All the regular expressions supported in PCRE will be supported in PureBasic. To have a complete list of supported pattern and arguments, please visit the PCRE page: <http://www.pcre.org/pcre.txt>. The PCRE license can be viewed [here](#).

Important: The PCRE license requires that a copyright notice and the license text itself be included in any software that uses the library. So if the RegularExpression library is used in software that is to be made public, the above linked license must be included with the software.

153.1 CountRegularExpressionGroups

Syntax

```
Result = CountRegularExpressionGroups(#RegularExpression)
```

Description

Returns the number of groups defined in the #RegularExpression. The matches of regular expression groups can be accessed with functions like RegularExpressionGroup().

Parameters

#RegularExpression The regular expression to use.

Return value

The number of groups defined in the regular expression.

Remarks

Groups in a regular expression are defined by surrounding a sub-expression with braces "(" and ")". The groups are numbered as they appear in the regular expression from left to right. The first group has index 1.

See Also

RegularExpressionGroup()

153.2 CreateRegularExpression

Syntax

```
Result = CreateRegularExpression(#RegularExpression, Pattern$ [,
                                Flags])
```

Description

Create a new regular expression using the specified pattern.

Parameters

#RegularExpression A number to identify the new regular expression. #PB_Any can be used to auto-generate this number.

Pattern\$ The regular expression which will be applied to the string to match, extract or replace.

Flags (optional) It can be a combination of the following values:

```
#PB_RegularExpression_DotAll      : '.' matches anything
    including newlines.
#PB_RegularExpression_Extended  : whitespace and '#' comments
    will be ignored.
#PB_RegularExpression_MultiLine : '^' and '$' match newlines
    within data.
#PB_RegularExpression_AnyNewLine: recognize 'CR', 'LF', and
    'CRLF' as newline sequences.
#PB_RegularExpression_NoCase     : comparison and matching
    will be case-insensitive
```

Return value

Returns nonzero if the regular expression was created successfully and zero if not. If #PB_Any was used for the #RegularExpression parameter then the generated number is returned on success. If an error has been detected in the pattern, the result will be zero. To get more information about the error, see RegularExpressionError() .

Remarks

If a regular expression isn't used anymore, use FreeRegularExpression() to free up some resources.

Example

```
1 ; This expression will match every word of 3 letter which begin
  ; by a lower case letter,
2 ; followed with the character 'b' and which ends with an
  ; uppercase letter. ex: abC
3 ;
4 If CreateRegularExpression(0, "[a-z]b[A-Z]")
5     Debug MatchRegularExpression(0, "abC") ; Will print 1
6     Debug MatchRegularExpression(0, "abc") ; Will print 0
7 Else
```

```

8     Debug RegularExpressionError()
9 EndIf

```

See Also

RegularExpressionError() , FreeRegularExpression()

153.3 ExamineRegularExpression

Syntax

```
Result = ExamineRegularExpression(#RegularExpression, String$)
```

Description

Starts matching the #RegularExpression against the given String\$. Individual matches can be iterated using the NextRegularExpressionMatch() function. From each match, the matching string, its position/length and any groups within the match can be extracted with the appropriate function.

Parameters

#RegularExpression The regular expression to use.

String\$ The string to apply the regular expression on.

Return value

Returns non-zero if the matching was started successfully. Whether an actual match was found can be determined by calling NextRegularExpressionMatch() .

Example

```

1   ; This expression will match every word of 3 letter which begin
    ; by a lower case letter,
2   ; followed with the character 'b' and which ends with an
    ; uppercase letter. ex: abC
3   ; Each match is printed with its position in the original string.
4   ;
5   If CreateRegularExpression(0, "[a-z]b[A-Z]")
6       If ExamineRegularExpression(0, "abC ABc zbA abc")
7           While NextRegularExpressionMatch(0)
8               Debug "Match: " + RegularExpressionMatchString(0)
9               Debug "    Position: " +
                Str(RegularExpressionMatchPosition(0))
10              Debug "    Length: " + Str(RegularExpressionMatchLength(0))
11          Wend
12      EndIf
13  Else
14      Debug RegularExpressionError()
15  EndIf

```

See Also

NextRegularExpressionMatch() , RegularExpressionMatchString() ,
RegularExpressionMatchPosition() , RegularExpressionMatchLength()

153.4 ExtractRegularExpression

Syntax

```
Result = ExtractRegularExpression(#RegularExpression, String$,  
    Array$())
```

Description

Extracts strings according to the #RegularExpression into an array.

Parameters

#RegularExpression The regular expression to use.

String\$ The string to apply the regular expression on.

Array\$() The extracted strings will be stored in this array. The array is automatically resized to the number of element matching the expression found in the specified string.

Return value

Returns the number of elements matching the regular expression in the string.

Example

```
1      ; This expression will match every word of 3 letter which begin  
      by a lower case letter,  
2      ; followed with the character 'b' and which ends with an  
      uppercase letter. ex: abC  
3      ;  
4      If CreateRegularExpression(0, "[a-z]b[A-Z]")  
5          Dim Result$(0)  
6          NbFound = ExtractRegularExpression(0, "abC ABc zBA abc",  
          Result$())  
7          For k = 0 To NbFound-1  
8              Debug Result$(k)  
9          Next  
10     Else  
11         Debug RegularExpressionError()  
12     EndIf
```

See Also

CreateRegularExpression()

153.5 FreeRegularExpression

Syntax

```
FreeRegularExpression(#RegularExpression)
```

Description

Free the specified #RegularExpression and release its associated memory.

Parameters

#RegularExpression Free the regular expression. If #PB_All is specified, all the remaining regular expressions are freed.

Return value

None.

Remarks

All remaining regular expressions are automatically freed when the program ends.

See Also

CreateRegularExpression()

153.6 IsRegularExpression

Syntax

```
Result = IsRegularExpression(#RegularExpression)
```

Description

Tests if the given #RegularExpression number is a valid and correctly initialized, regular expression.

Parameters

#RegularExpression The regular expression to use.

Return value

Returns nonzero if #RegularExpression is a valid regular expression and zero otherwise.

Remarks

This function is bulletproof and can be used with any value. If the 'Result' is not zero then the object is valid and initialized, otherwise it will equal zero. This is the correct way to ensure a regular expression is ready to use.

See Also

CreateRegularExpression()

153.7 MatchRegularExpression

Syntax

```
Result = MatchRegularExpression(#RegularExpression, String$)
```

Description

Tests the string against the #RegularExpression.

Parameters

#RegularExpression The regular expression to use.

String\$ The string to apply the regular expression on.

Return value

Returns nonzero if the string matches the regular expression, returns zero otherwise.

Example

```
1      ; This expression will match every word of 3 letter which begin
      by a lower case letter ,
2      ; followed with the character 'b' and which ends with an
      uppercase letter. ex: abC
3      ;
4      If CreateRegularExpression(0, "[a-z]b[A-Z]")
5          If MatchRegularExpression(0, "abC ABc zbA abc")
6              Debug "The string match !"
7          Else
8              Debug "No pattern found in the string"
9          EndIf
10     Else
11         Debug RegularExpressionError()
12     EndIf
```

See Also

CreateRegularExpression()

153.8 NextRegularExpressionMatch

Syntax

```
Result = NextRegularExpressionMatch(#RegularExpression)
```

Description

Iterates over all regular expression matches in the target string after a call to ExamineRegularExpression() .

Parameters

#RegularExpression The regular expression to use. `ExamineRegularExpression()` must have been called on this regular expression.

Return value

Returns non-zero if another match was found. If there are no more matches in the string, the result is zero.

Remarks

The following functions can be used to get information about the current match:

- `RegularExpressionMatchString()` : Get the current matching string
- `RegularExpressionMatchPosition()` : Get the position of the current match
- `RegularExpressionMatchLength()` : Get the length of the current match
- `RegularExpressionGroup()` : Extract the matching string of a group
- `RegularExpressionGroupPosition()` : Get the position (within the match) of a group
- `RegularExpressionGroupLength()` : Get the length of a group
- `RegularExpressionNamedGroup()` : Extract the matching string of a named group
- `RegularExpressionNamedGroupPosition()` : Get the position (within the match) of a named group
- `RegularExpressionNamedGroupLength()` : Get the length of a named group

Example

See `ExamineRegularExpression()` for an example.

See Also

`ExamineRegularExpression()`

153.9 RegularExpressionMatchString

Syntax

```
Result\$\$ = RegularExpressionMatchString(#RegularExpression)
```

Description

Returns the string that matched the `#RegularExpression` in the last call to `NextRegularExpressionMatch()`.

Parameters

#RegularExpression The regular expression to use. `ExamineRegularExpression()` and `NextRegularExpressionMatch()` must have been called on this regular expression.

Return value

The sub-string that matched the regular expression.

Example

See `ExamineRegularExpression()` for an example.

See Also

ExamineRegularExpression() , NextRegularExpressionMatch() ,
RegularExpressionMatchPosition() , RegularExpressionMatchLength()

153.10 RegularExpressionMatchPosition

Syntax

```
Result = RegularExpressionMatchPosition(#RegularExpression)
```

Description

Returns the position within the input string (passed to ExamineRegularExpression()) of the current match after a call to NextRegularExpressionMatch() .

Parameters

#RegularExpression The regular expression to use. ExamineRegularExpression() and NextRegularExpressionMatch() must have been called on this regular expression.

Return value

The position of the match within the original string. The first character in the string is at position 1.

Example

See ExamineRegularExpression() for an example.

See Also

ExamineRegularExpression() , NextRegularExpressionMatch() , RegularExpressionMatchString() ,
RegularExpressionMatchLength()

153.11 RegularExpressionMatchLength

Syntax

```
Result = RegularExpressionMatchLength(#RegularExpression)
```

Description

Returns the length in characters of the current matching string after a call to NextRegularExpressionMatch() .

Parameters

#RegularExpression The regular expression to use. ExamineRegularExpression() and NextRegularExpressionMatch() must have been called on this regular expression.

Return value

The length of the current match in characters.

Example

See `ExamineRegularExpression()` for an example.

See Also

`ExamineRegularExpression()` , `NextRegularExpressionMatch()` , `RegularExpressionMatchString()` , `RegularExpressionMatchPosition()`

153.12 RegularExpressionGroup

Syntax

```
Result\$$ = RegularExpressionGroup(#RegularExpression, Group)
```

Description

Extract the string matched by a group within the regular expression after a call to `NextRegularExpressionMatch()` .

Parameters

#RegularExpression The regular expression to use. `ExamineRegularExpression()` and `NextRegularExpressionMatch()` must have been called on this regular expression.

Group The index of the group to extract. The first group has index 1.

Return value

Returns the string matched by the regular expression group.

Remarks

Groups in a regular expression are defined by surrounding a sub-expression with braces "(" and ")". The groups are numbered as they appear in the regular expression from left to right. The first group has index 1. The `CountRegularExpressionGroups()` function can be used to find out the number of groups in a regular expression.

As an alternative, the `RegularExpressionNamedGroup()` function can be used.

Example

```
1 ; This expression matches a color setting string (with value red,
   green or blue)
2 ; The colors are grouped with () and the color value is extracted
   in case of a match
3 ;
4 If CreateRegularExpression(0, "color=(red|green|blue)")
5   If ExamineRegularExpression(0, "stype=bold, color=blue,
      margin=50")
6     While NextRegularExpressionMatch(0)
7         Debug "The color is " + RegularExpressionGroup(0, 1)
8     Wend
9   EndIf
10 Else
11     Debug RegularExpressionError()
12 EndIf
```

See Also

ExamineRegularExpression() , NextRegularExpressionMatch() ,
RegularExpressionGroupPosition() , RegularExpressionGroupLength() ,
RegularExpressionNamedGroup()

153.13 RegularExpressionGroupPosition

Syntax

```
Result = RegularExpressionGroupPosition(#RegularExpression, Group)
```

Description

Returns the position (within the current matching string) of the specified group after a call to NextRegularExpressionMatch() .

Parameters

#RegularExpression The regular expression to use. ExamineRegularExpression() and NextRegularExpressionMatch() must have been called on this regular expression.

Group The index of the group. The first group has index 1.

Return value

Returns the character position of the group within the matching string (not within the original input string!). The first character of the match has position 1.

Remarks

Groups in a regular expression are defined by surrounding a sub-expression with braces "(" and ")". The groups are numbered as they appear in the regular expression from left to right. The first group has index 1. The CountRegularExpressionGroups() function can be used to find out the number of groups in a regular expression.

As an alternative, the RegularExpressionNamedGroupPosition() function can be used.

See Also

ExamineRegularExpression() , NextRegularExpressionMatch() , RegularExpressionGroup() ,
RegularExpressionGroupLength() , RegularExpressionNamedGroupPosition()

153.14 RegularExpressionGroupLength

Syntax

```
Result = RegularExpressionGroupLength(#RegularExpression, Group)
```

Description

Returns the length of the specified regular expression group after a call to NextRegularExpressionMatch() .

Parameters

#RegularExpression The regular expression to use. `ExamineRegularExpression()` and `NextRegularExpressionMatch()` must have been called on this regular expression.

Group The index of the group. The first group has index 1.

Return value

Returns the character position of the group.

Remarks

Groups in a regular expression are defined by surrounding a sub-expression with braces "(" and ")". The groups are numbered as they appear in the regular expression from left to right. The first group has index 1. The `CountRegularExpressionGroups()` function can be used to find out the number of groups in a regular expression.

As an alternative, the `RegularExpressionNamedGroupLength()` function can be used.

See Also

`ExamineRegularExpression()` , `NextRegularExpressionMatch()` , `RegularExpressionGroup()` , `RegularExpressionGroupPosition()` , `RegularExpressionNamedGroupLength()`

153.15 RegularExpressionNamedGroup

Syntax

```
Result\$$ = RegularExpressionNamedGroup(#RegularExpression,  
    GroupName$)
```

Description

Extract the string matched by a named group within the regular expression after a call to `NextRegularExpressionMatch()` .

Parameters

#RegularExpression The regular expression to use. `ExamineRegularExpression()` and `NextRegularExpressionMatch()` must have been called on this regular expression.

GroupName\$ The name of the group to extract. The group name is case sensitive.

Return value

Returns the string matched by the regular expression group.

Remarks

Groups in a regular expression are defined by surrounding a sub-expression with braces "(" and ")". Groups can either accessed by index using functions like `RegularExpressionGroup()` or they can be assigned a name using the "(?<name>)" syntax as shown in the example below.

Example

```
1 ; This expression matches a color setting string (with value red,
   green or blue)
2 ; The colors are grouped with () which has the name "col"
   assigned to it.
3 ; This color name is then extracted
4 ;
5 If CreateRegularExpression(0, "color=(?<col>red|green|blue)")
6   If ExamineRegularExpression(0, "stype=bold, color=blue,
   margin=50")
7     While NextRegularExpressionMatch(0)
8       Debug "The color is " + RegularExpressionNamedGroup(0,
   "col")
9     Wend
10  EndIf
11 Else
12   Debug RegularExpressionError()
13 EndIf
```

See Also

ExamineRegularExpression() , NextRegularExpressionMatch() ,
RegularExpressionNamedGroupPosition() , RegularExpressionNamedGroupLength() ,
RegularExpressionGroup()

153.16 RegularExpressionNamedGroupPosition

Syntax

```
Result = RegularExpressionNamedGroupPosition(#RegularExpression,  
      GroupName$)
```

Description

Returns the position (within the current matching string) of the specified named group after a call to NextRegularExpressionMatch() .

Parameters

#RegularExpression The regular expression to use. ExamineRegularExpression() and NextRegularExpressionMatch() must have been called on this regular expression.

GroupName\$ The name of the group to extract. The group name is case sensitive.

Return value

Returns the character position of the group within the matching string (not within the original input string!). The first character of the match has position 1.

Remarks

Groups in a regular expression are defined by surrounding a sub-expression with braces "(" and ")". Groups can either accessed by index using functions like RegularExpressionGroupPosition() or they can be assigned a name using the "(?<name>)" syntax as shown in the example for the RegularExpressionNamedGroup() function.

See Also

ExamineRegularExpression() , NextRegularExpressionMatch() , RegularExpressionNamedGroup()
, RegularExpressionNamedGroupLength() , RegularExpressionGroupPosition()

153.17 RegularExpressionNamedGroupLength

Syntax

```
Result = RegularExpressionNamedGroupLength(#RegularExpression,  
      GroupName$)
```

Description

Returns the length of the specified named regular expression group after a call to NextRegularExpressionMatch() .

Parameters

#RegularExpression The regular expression to use. ExamineRegularExpression() and NextRegularExpressionMatch() must have been called on this regular expression.

GroupName\$ The name of the group to extract. The group name is case sensitive.

Return value

Returns the character position of the group.

Remarks

Groups in a regular expression are defined by surrounding a sub-expression with braces "(" and ")". Groups can either accessed by index using functions like RegularExpressionGroupLength() or they can be assigned a name using the "(?<name>)" syntax as shown in the example for the RegularExpressionNamedGroup() function.

See Also

ExamineRegularExpression() , NextRegularExpressionMatch() , RegularExpressionNamedGroup()
, RegularExpressionNamedGroupPosition() , RegularExpressionGroupLength()

153.18 ReplaceRegularExpression

Syntax

```
Result\<String> = ReplaceRegularExpression(#RegularExpression, String$,  
      ReplaceString$)
```

Description

Replaces all strings matching the #RegularExpression with 'ReplaceString\$'.

Parameters

#RegularExpression The regular expression to use.

String\$ The string to apply the regular expression on.

ReplaceString\$ The string to use to replace the matched expression.

Return value

Returns a new string with all matched expressions replaced with 'ReplaceString\$'.

Remarks

Back references (usually described as \1, \2, etc.) are not supported. ExtractRegularExpression() combined with ReplaceString() should achieve the requested behaviour.

Example

```
1      ; This expression will match every word of 3 letter which begin
      by a lower case letter,
2      ; followed with the character 'b' and which ends with an
      uppercase letter. ex: abC
3      ;
4      If CreateRegularExpression(0, "[a-z]b[A-Z]")
5          Result$ = ReplaceRegularExpression(0, "abC ABc zbA abc", "---")
6          Debug Result$ ; Will print "--- ABc --- abc"
7      Else
8          Debug RegularExpressionError()
9      EndIf
```

See Also

CreateRegularExpression()

153.19 RegularExpressionError

Syntax

```
Result\$$ = RegularExpressionError()
```

Description

Returns an human readable error (in english) about the latest failure of CreateRegularExpression().

Parameters

None.

Return value

Returns an human readable error (in english) about the latest failure of CreateRegularExpression().

Example

```
1      ; Here we put an extra bracket '[', so there is a syntax error in  
      the regular expression  
2      ;  
3      If CreateRegularExpression(0, "[a-z]b[[A-Z][]")  
4          Debug "Success"  
5      Else  
6          Debug RegularExpressionError()  
7      EndIf
```

See Also

CreateRegularExpression()

Chapter 154

Requester

Overview

Computer users are well used to requesters as almost any graphical application use at least one of them. They are very handy as some basic tasks (like opening a file, choose a color, choose a font etc...) are all performed through standard windows called 'requesters'.

154.1 ColorRequester

Syntax

```
Color = ColorRequester([CurrentColor [, ParentID])
```

Description

Opens the standard requester to choose a color. The chosen color is returned under a 24-bit number containing the red, green and blue value, as usual.

Parameters

CurrentColor (optional) Sets the default color when the requester is opened. RGB() can be used to get a valid color.

ParentID (optional) The parent window identifier. A valid window identifier can be retrieved with WindowID() .

Return value

The selected color, or -1 if the user canceled the requester. To easily get each RGB component value, Red() , Green() and Blue() can be used.

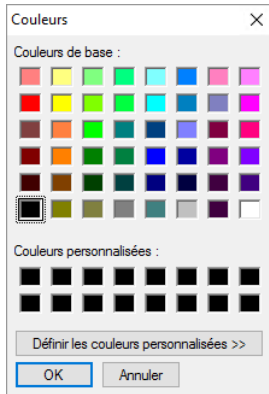
Example

```
1  Color = ColorRequester()
2  If Color > -1
3      a$ = "You have selected following color value:" + Chr(10) ;
        Chr(10) only needed
4      a$ + "24 Bit value: " + Str(Color) + Chr(10) ;
        for line-feed
5      a$ + "Red value:      " + Str(Red(Color)) + Chr(10)
6      a$ + "Green value:   " + Str(Green(Color)) + Chr(10)
7      a$ + "Blue value:    " + Str(Blue(Color))
```

```

8 Else
9     a$ = "The requester was canceled."
10 EndIf
11 MessageRequester("Information", a$, 0)
12 End

```



154.2 FontRequester

Syntax

```
Result = FontRequester(FontName$, FontSize, Flags [, Color [, Style
[, ParentID]])
```

Description

Opens the standard requester to choose a font. The functions SelectedFontColor() , SelectedFontName() , SelectedFontSize() and SelectedFontStyle() can be used after a successful call to get the needed information about the selected font.

Parameters

FontName\$ The default font name to use when the requester is opened.

FontSize The default font size to use when the requester is opened.

Flags It can be one of the following values:

```
#PB_FontRequester_Effects : Enable the requesters effects,
including color chooser (Windows only).
```

Color (optional) Sets the default color when the requester is opened. A valid color value you can get with the function RGB() .

Style (optional) Sets the default style when the requester is opened. See SelectedFontStyle() for available styles.

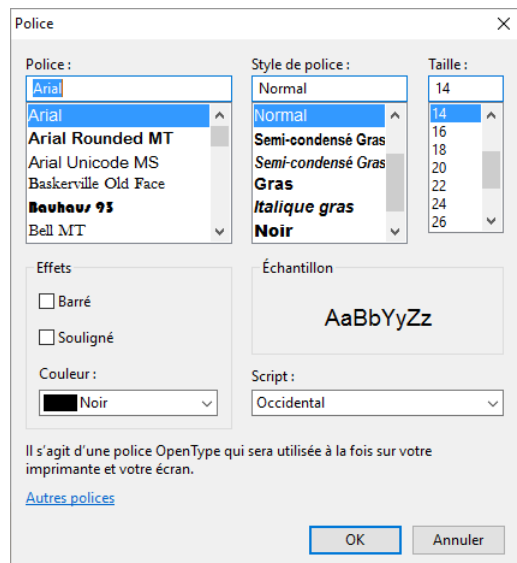
ParentID (optional) The parent window identifier. A valid window identifier can be retrieved with WindowID() .

Return value

Nonzero if a font has been chosen, zero if the user canceled the requester.

Example

```
1  FontName$ = "Arial"      ; set initial font (could also be blank)
2  FontSize  = 14           ; set initial size (could also be null)
3  Result = FontRequester(FontName$, FontSize,
4                        #PB_FontRequester_Effects)
5  If Result
6      Message$ = "You have selected following font:" + #LF$
7      Message$ + "Name:  " + SelectedFontName()      + #LF$
8      Message$ + "Size:  " + Str(SelectedFontSize()) + #LF$
9      Message$ + "Color: " + Str(SelectedFontColor()) + #LF$
10     If SelectedFontStyle() & #PB_Font_Bold
11         Message$ + "Bold" + #LF$
12     EndIf
13     If SelectedFontStyle() & #PB_Font_StrikeOut
14         Message$ + "StrikeOut" + #LF$
15     EndIf
16     If SelectedFontStyle() & #PB_Font_Underline
17         Message$ + "Underline" + #LF$
18     EndIf
19 Else
20     Message$ = "The requester was canceled."
21 EndIf
22 MessageRequester("FontRequester", Message$,
23                 #PB_MessageRequester_Ok)
```



See Also

SelectedFontColor() , SelectedFontName() , SelectedFontSize() , SelectedFontStyle()

154.3 InputRequester

Syntax

```
Text\$_ = InputRequester(Title$, Message$, DefaultText$ [, Flags [,
    ParentID]])
```

Description

Opens a blocking input requester to enter some text.

Parameters

Title\$ Title of the requester.

Message\$ Message displayed before the input field.

DefaultText\$ Default text in the input field.

Flags (optional) Can be the following value:

```
#PB_InputRequester_Password: set the field type as  
'password'. The displayed text will be hidden.
```

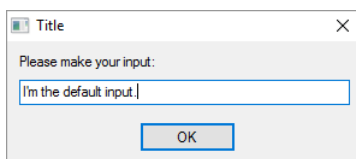
ParentID (optional) The parent window identifier. A valid window identifier can be retrieved with WindowID() .

Return value

Returns the text in the input field, or an empty string if the requester has been closed without hitting the "OK" button.

Example

```
1  Input$ = InputRequester("Title", "Please make your input:", "I'm  
   the default input.")  
2  
3  If Input$ > ""  
4      a$ = "You entered in the requester:" + Chr(10) ; Chr(10) only  
        needed  
5      a$ + Input$ ;  
        for line-feed  
6  Else  
7      a$ = "The requester was canceled or there was nothing entered."  
8  EndIf  
9  MessageRequester("Information", a$, 0)
```



154.4 MessageRequester

Syntax

```
Result = MessageRequester(Title$, Text$ [, Flags [, ParentID]])
```

Description

Opens a blocking requester to display some information. The program execution is totally stopped until the user close the requester.

Parameters

Title\$ The title of the requester window.

Text\$ The text displayed in the requester window.

Flags (optional) It can be one of the following value:

```
#PB_MessageRequester_Ok           : to have the 'ok' only
button (default)
#PB_MessageRequester_YesNo        : to have 'yes' or 'no'
buttons
#PB_MessageRequester_YesNoCancel : to have 'yes', 'no' and
'cancel' buttons
```

Combined with one of the following value:

```
#PB_MessageRequester_Info    : displays an info icon
#PB_MessageRequester_Warning : displays a warning icon
#PB_MessageRequester_Error   : displays an error icon
```

ParentID (optional) The parent window identifier. A valid window identifier can be retrieved with WindowID() .

Return value

It can be one of the following constants:

```
#PB_MessageRequester_Yes    : the 'yes' button was pressed
#PB_MessageRequester_No     : the 'no' button was pressed
#PB_MessageRequester_Cancel : the 'Cancel' button was pressed
```

Example: Simple MessageRequester (usually for information purposes only)

```
1 MessageRequester("Information", "Just a short information text.",
  #PB_MessageRequester_Ok | #PB_MessageRequester_Info)
```

Example: MessageRequester with Yes / No buttons (usually to question)

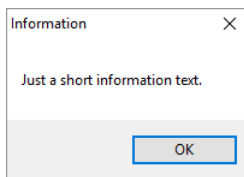
```
1 Result = MessageRequester("Title", "Please make your input:",
  #PB_MessageRequester_YesNo)
2 a$ = "Result of the requester was: "
3 If Result = #PB_MessageRequester_Yes ; pressed Yes button
4   a$ + "Yes "
5 Else ; pressed No button
6   a$ + "No "
7 EndIf
8 MessageRequester("Information", a$, #PB_MessageRequester_Ok)
```

Example: MessageRequester with Yes / No / Cancel buttons (usually to question)

```

1  Result = MessageRequester("Title", "Please make your input:",
2    #PB_MessageRequester_YesNoCancel)
3  a$ = "Result of the previously requester was: "
4  If Result = #PB_MessageRequester_Yes      ; pressed Yes button
5    a$ + "Yes"
6  ElseIf Result = #PB_MessageRequester_No   ; pressed No button
7    a$ + "No"
8  Else                                      ; pressed Cancel
9    button or Esc
10   a$ + "Cancel"
11 EndIf
12 MessageRequester("Information", a$, #PB_MessageRequester_Ok)

```



154.5 NextSelectedFilename

Syntax

```
Filename\$ = NextSelectedFilename()
```

Description

After `OpenFileRequester()` with `#PB_Requester_MultiSelection`, it returns the next selected file (if any).

Parameters

None.

Return value

A filename, or an empty string if there are no more selected filenames.

Example

```

1  Filename$ = OpenFileRequester("Choose some files","", "",0,
2    #PB_Requester_MultiSelection)
3
4  While Filename$
5    Debug Filename$
6    Filename$ = NextSelectedFilename()
7  Wend

```

154.6 OpenFileRequester

Syntax


```
Filename\$ = OpenFileDialog(Title$, DefaultFile$, Pattern$,
    PatternPosition [, Flags [, ParentID]])
```

Description

Opens the standard requester for the user to choose a file. The title can be specified to replace the default one. The DefaultFile\$ is useful to initialize the requester in the right directory and with the right filename.

Parameters

Title\$ The title of the requester window.

DefaultFile\$ The default file displayed when the requester is opened.

Pattern\$ A standard filter which allow to display only the files which end with such or such extension. It has to be in the following form : "Text file || *.txt || Music file || *.mus;*.mod" The pattern always works in pairs: name (which really appears in the filter) and extension (ie: *.txt). Several extensions can be specified for a single type by using the ; (semi-colon) separator (not supported on OSX, the requester always displays all files).

PatternPosition Specifies which pattern must be the default. It begins from 0 up to the number of pattern minus 1 (because the index for the first pattern begins at 0). Once the requester has been closed, SelectedFilePattern() can be used to get back the selected pattern (not supported on OSX).

Flags (optional) It can be a one of the following values:

```
#PB_Requester_MultiSelection: Enable the multiselection. (see
    NextSelectedFilename()
)
```

ParentID (optional) The parent window identifier. A valid window identifier can be retrieved with WindowID() .

Return value

The selected filename, or an empty string if the requester has been canceled by the user.

Remarks

The returned 'Filename\$' can be easily split into file, path and extension string with the following functions: GetFilePart() , GetPathPart() and GetExtensionPart() .

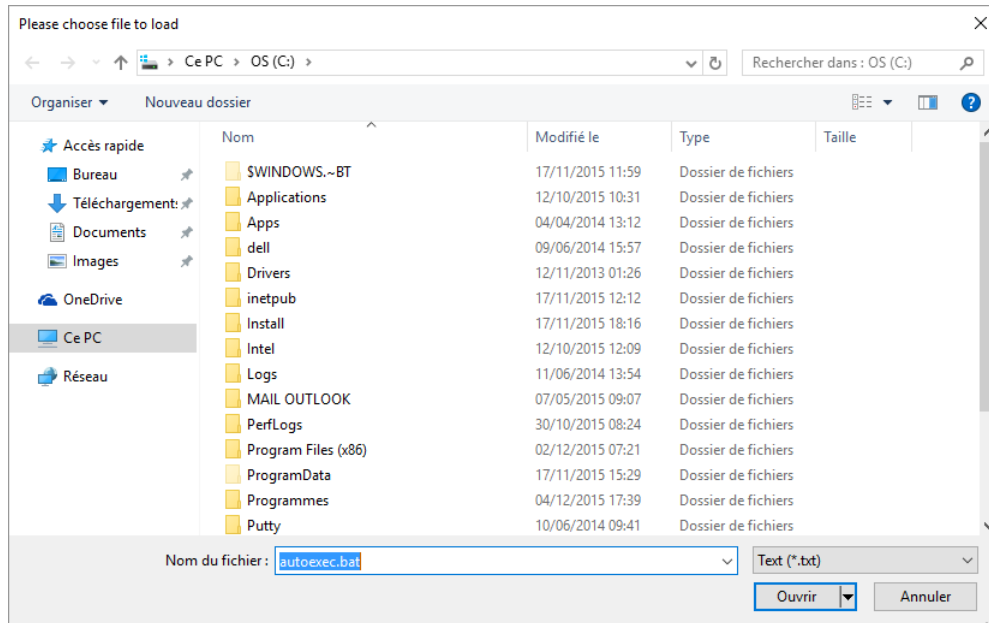
Example

```
1  StandardFile$ = "C:\autoexec.bat"    ; set initial file+path to
   display
2  ; With next string we will set the search patterns ("|" as
   separator) for file displaying:
3  ; 1st: "Text (*.txt)" as name, ".txt" and ".bat" as allowed
   extension
4  ; 2nd: "PureBasic (*.pb)" as name, ".pb" as allowed extension
5  ; 3rd: "All files (*.*)" as name, "*.*)" as allowed extension,
   valid for all files
6  Pattern$ = "Text (*.txt;*.bat)|*.txt;*.bat|PureBasic
   (*.pb)|*.pb|All files (*.*)|*.*"
7  Pattern = 0    ; use the first of the three possible patterns as
   standard
```

```

8   File$ = OpenFileDialog("Please choose file to load",
   StandardFile$, Pattern$, Pattern)
9   If File$
10      MessageRequester("Information", "You have selected following
   file:" + Chr(10) + File$, 0)
11  Else
12      MessageRequester("Information", "The requester was canceled.",
   0)
13  EndIf

```



See Also

NextSelectedFilename()

154.7 PathRequester

Syntax

```
Path\$ = PathRequester(Title$, InitialPath$ [, ParentID])
```

Description

Opens the standard path requester for the user to select a path.

Parameters

Title\$ The title of the requester window.

InitialPath\$ The initial path to use when the requester is opened.

ParentID (optional) The parent window identifier. A valid window identifier can be retrieved with WindowID() .

Return value

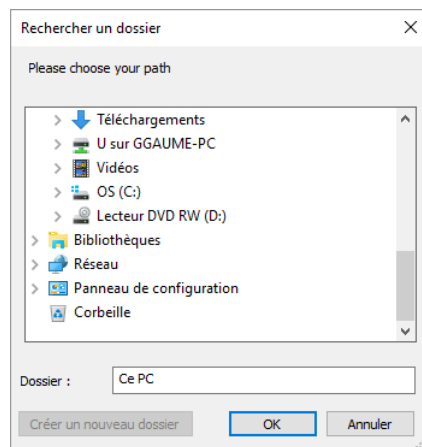
The selected path, or an empty string if the requester has been canceled by the user.

Remarks

The path is returned with the terminating '\' on Windows, or the terminating '/' on Linux and OS X.

Example

```
1 InitialPath$ = "C:\\" ; set initial path to display (could also
  be blank)
2 Path$ = PathRequester("Please choose your path", InitialPath$)
3 If Path$
4     MessageRequester("Information", "You have selected following
  path:" + Chr(10) + Path$, 0)
5 Else
6     MessageRequester("Information", "The requester was canceled.",
  0)
7 EndIf
```



154.8 SaveFileRequester

Syntax

```
Filename\_$ = SaveFileRequester(Title$, DefaultFile$, Pattern$,
  PatternPosition [, ParentID])
```

Description

Opens the standard requester for the user to save a file.

Parameters

Title\$ The title of the requester window.

DefaultFile\$ The filename to use when the requester is opened.

Pattern\$ A standard filter which allow to display only the files which end with such or such extension. It must be in the following form : "Text file || *.txt || Music file || *.mus;*.mod"
The pattern always work by pair: name (which really appears in the filter) and extension (ie: *.txt). Several extensions can be specified for a single type by using the ; (semi-colon) separator (not supported on OSX, the requester always displays all files).

PatternPosition Specifies which pattern must be the default. It begins from 0 up to the number of pattern minus 1 (because the index for the first pattern begins at 0). Once the requester has been closed, SelectedFilePattern() can be used to get back the selected pattern (not supported on OSX).

ParentID (optional) The parent window identifier. A valid window identifier can be retrieved with WindowID() .

Return value

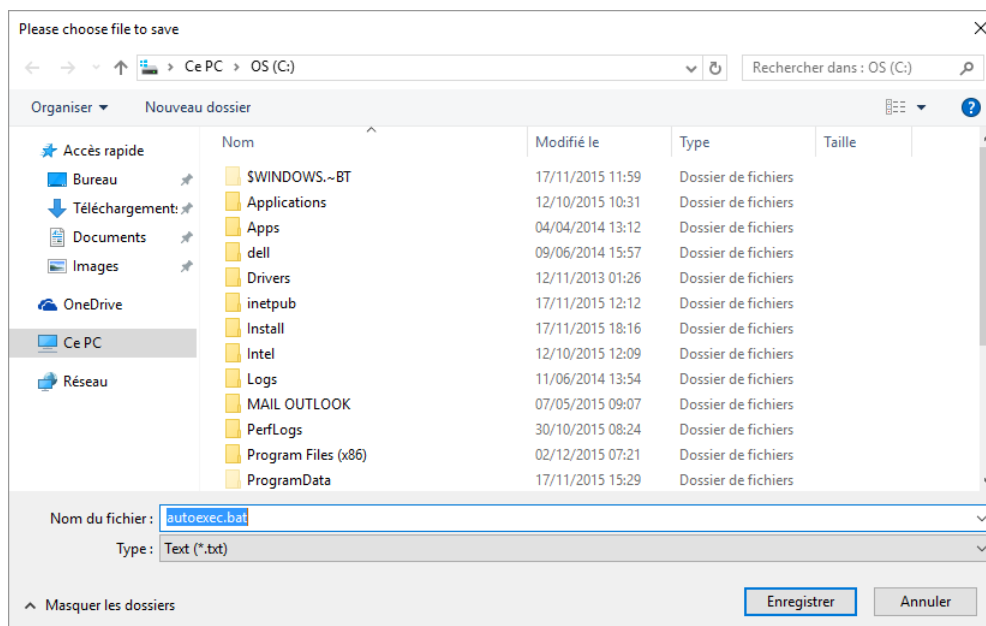
The selected filename, or an empty string if the requester has been canceled by the user.

Remarks

The returned 'Filename\$' can be easily split into file, path and extension string with the following functions: GetFilePart() , GetPathPart() and GetExtensionPart() .

Example

```
1  StandardFile$ = "C:\autoexec.bat"    ; set initial file+path to
   display
2  ; With next string we will set the search patterns ("|" as
   separator) for file displaying:
3  ; 1st: "Text (*.txt)" as name, ".txt" and ".bat" as allowed
   extension
4  ; 2nd: "PureBasic (*.pb)" as name, ".pb" as allowed extension
5  ; 3rd: "All files (*.*)" as name, "*.*)" as allowed extension,
   valid for all files
6  Pattern$ = "Text (*.txt)|*.txt;*.bat|PureBasic (*.pb)|*.pb|All
   files (*.*)|*.*"
7  Pattern = 0      ; use the first of the three possible patterns as
   standard
8  File$ = SaveFileRequester("Please choose file to save",
   StandardFile$, Pattern$, Pattern)
9  If File$
10     MessageRequester("Information", "You have selected following
        file:"+Chr(10)+File$, 0)
11 Else
12     MessageRequester("Information", "The requester was canceled.",
        0)
13 EndIf
```



154.9 SelectedFilePattern

Syntax

`Result = SelectedFilePattern()`

Description

Returns the selected pattern index chosen with `OpenFileRequester()` or `SaveFileRequester()` .

Parameters

None.

Return value

The selected pattern index chosen with `OpenFileRequester()` or `SaveFileRequester()` . The first pattern is at position 0.

Example

```

1 StandardFile$ = "C:\autoexec.bat"           ; initial path + file
2 Pattern$ = "Text (*.txt)|*.txt;*.bat|"      ; set first pattern
   (index = 0)
3 Pattern$ + "PureBasic (*.pb)|*.pb|"         ; set second pattern
   (index = 1)
4 Pattern$ + "Bmp (*.bmp)|*.bmp|"             ; set third pattern
   (index = 2)
5 Pattern$ + "Jpeg (*.jpg)|*.jpg|"            ; set fourth pattern
   (index = 3)
6 Pattern$ + "All files (*.*)|*.*"            ; set fifth pattern
   (index = 4)
7 Pattern = 1                                ; use the second of the five possible patterns as
   standard
8

```

```

9      ; Now we open a filerequester, you can change the pattern and
      will get the index after closing
10     File$ = OpenFileDialog("Please choose file to load",
      StandardFile$, Pattern$, Pattern)
11     Index = SelectedFilePattern()
12     If Index > -1
13         MessageRequester("Information", "Following pattern index was
      selected: "+Str(Index), 0)
14     Else
15         MessageRequester("Information", "The requester was canceled.",
      0)
16     EndIf

```

154.10 SelectedFontColor

Syntax

```
Color = SelectedFontColor()
```

Description

Returns the color of the font chosen by the user with the FontRequester() .

Parameters

None.

Return value

The color of the font chosen by the user with the FontRequester() . To get individual color RGB values Red() , Green() and Blue() can be used.

See Also

FontRequester()

Supported OS

Windows

154.11 SelectedFontName

Syntax

```
Name\$_ = SelectedFontName()
```

Description

Returns the name of the font chosen by the user with the FontRequester() .

Parameters

None.

Return value

The name of the font chosen by the user with the `FontRequester()` . This name can be used directly by the `LoadFont()` function.

See Also

`FontRequester()`

154.12 SelectedFontSize

Syntax

```
Size = SelectedFontSize()
```

Description

Returns the size of the font chosen by the user with the `FontRequester()` .

Parameters

None.

Return value

The size of the font chosen by the user with the `FontRequester()` .

See Also

`FontRequester()`

154.13 SelectedFontStyle

Syntax

```
Style = SelectedFontStyle()
```

Description

Returns the style of the font chosen by the user with the `FontRequester()` .

Parameters

None.

Return value

The style of the font chosen by the user with the `FontRequester()` . It can contain one or several of the following values:

```
#PB_Font_Bold    : Font is bold.
#PB_Font_Italic  : Font is italic.
```

To test if a value is present, use the `'&'` (bitwise AND) operator:

```
1   If SelectedFontStyle() & #PB_Font_Bold
2       ; The font is bold
3   EndIf
```

See Also

FontRequester()

Chapter 155

Runtime

Overview

Runtime objects are accessible even when the program is compiled, using their string references. For more information about the runtime concept, see [Runtime](#) .

155.1 GetRuntimeInteger

Syntax

```
Result = GetRuntimeInteger(Object$)
```

Description

Returns the integer value of the runtime object. If the runtime object is a procedure, it returns the procedure address.

Parameters

Object\$ Name of the object to get the value from. The following objects are supported:

- Variable: the object name is 'VariableName' (case insensitive).
- Constant: the object name is '#ConstantName' (case insensitive).
- Procedure: the object name is 'ProcedureName()' (case insensitive).

When accessing public module items, the module prefix name is mandatory, even if [UseModule](#) is used.

Return value

Returns the integer value of the specified object, or zero if not found. As zero is a valid integer value, [IsRuntime\(\)](#) can be used to determine if the runtime object really exists. If the variable is of float or double type, it is automatically converted to integer. If the runtime object is a procedure, it returns the procedure address.

See Also

[SetRuntimeInteger\(\)](#) , [IsRuntime\(\)](#)

155.2 GetRuntimeDouble

Syntax

```
Result = GetRuntimeDouble(Object$)
```

Description

Returns the double value of the runtime object.

Parameters

Object\$ Name of the object to get the value from. The following objects are supported:

- Variable: the object name is 'VariableName' (case insensitive).
- Constant: the object name is '#ConstantName' (case insensitive).

When accessing public module items, the module prefix name is mandatory, even if [UseModule](#) is used.

Return value

Returns the double value of the specified object, or zero if not found. As zero is a valid double value, `IsRuntime()` can be used to determine if the runtime object really exists. If the variable is of integer or float type, it is automatically converted to double.

See Also

`SetRuntimeDouble()` , `IsRuntime()`

155.3 GetRuntimeString

Syntax

```
Result\_$ = GetRuntimeString(Object$)
```

Description

Returns the string value of the runtime object.

Parameters

Object\$ Name of the object to get the value from. The following objects are supported:

- Variable: the object name is 'VariableName' (case insensitive).
- Constant: the object name is '#ConstantName' (case insensitive).

When accessing public module items, the module prefix name is mandatory, even if [UseModule](#) is used.

Return value

Returns the string value of the specified object, or an empty string if not found. As an empty string is a valid string value, `IsRuntime()` can be used to determine if the runtime object really exists.

See Also

`SetRuntimeString()` , `IsRuntime()`

155.4 IsRuntime

Syntax

```
Result = IsRuntime(Object$)
```

Description

Checks if the specified object is declared as runtime .

Parameters

Object\$ Name of the object to check. The following objects are supported:

- Variable: the object name is 'VariableName' (case insensitive).
- Constant: the object name is '#ConstantName' (case insensitive).
- Procedure: the object name is 'ProcedureName()' (case insensitive).

When accessing public module items, the module prefix name is mandatory, even if [UseModule](#) is used.

Return value

Returns nonzero if the specified object has been declared as runtime , zero otherwise.

155.5 SetRuntimeDouble

Syntax

```
SetRuntimeDouble(Object$, Value)
```

Description

Changes the double value of the runtime object.

Parameters

Object\$ Name of the object to set the value of. The following objects are supported:

- Variable: the object name is 'VariableName' (case insensitive).

When setting public module items, the module prefix name is mandatory, even if [UseModule](#) is used.

Value The new double value to set.

Return value

None.

See Also

[GetRuntimeDouble\(\)](#) , [IsRuntime\(\)](#)

155.6 SetRuntimeInteger

Syntax

```
SetRuntimeInteger(Object$, Value)
```

Description

Changes the integer value of the runtime object.

Parameters

Object\$ Name of the object to set the value of. The following objects are supported:

- Variable: the object name is 'VariableName' (case insensitive).

When setting public module items, the module prefix name is mandatory, even if [UseModule](#) is used.

Value The new integer value to set.

Return value

None.

See Also

[GetRuntimeInteger\(\)](#) , [IsRuntime\(\)](#)

155.7 SetRuntimeString

Syntax

```
SetRuntimeString(Object$, Value$)
```

Description

Changes the string value of the runtime object.

Parameters

Object\$ Name of the object to set the value of. The following objects are supported:

- Variable: the object name is 'VariableName' (case insensitive).

When setting public module items, the module prefix name is mandatory, even if [UseModule](#) is used.

Value\$ The new string value to set.

Return value

None.

See Also

[GetRuntimeString\(\)](#) , [IsRuntime\(\)](#)

Chapter 156

Scintilla

Overview

Scintilla is a free source code editing component. It comes with complete source code and a license that permits use in any project or product personal or commercial. The license may be viewed here . The source code, as well as the library documentation may be found on the [Scintilla Homepage](#).

From the Scintilla Homepage: As well as features found in standard text editing components, Scintilla includes features especially useful when editing and debugging source code. These include support for syntax styling, error indicators, code completion and call tips. The selection margin can contain markers like those used in debuggers to indicate breakpoints and the current line. Styling choices are more open than with many editors, allowing the use of proportional fonts, bold and italics, multiple foreground and background colors and multiple fonts.

Important: The scintilla license requires that a copyright notice be included in all software that uses it and the license text itself be included in the documentation for the software. So if this library is used in software that is to be made public, the above linked license **MUST** be included with the software.

PureBasic integrates the Scintilla library with the gadget library , so standard commands like `ResizeGadget()` or `HideGadget()` can be used to control it. Furthermore it provides the `ScintillaSendMessage()` function to communicate with the control to use its full potential. All structures and constants needed are already defined in PureBasic.

The Scintilla library is linked statically, so there is no extra shared object to include.

156.1 InitScintilla

Syntax

```
Result = InitScintilla([LibraryName$])
```

Description

Warning

This function is deprecated, it may be removed in a future version of PureBasic. It should not be used in newly written code.

This command is deprecated and no more needed.

Supported OS

Windows

156.2 ScintillaGadget

Syntax

```
Result = ScintillaGadget(#Gadget, x, y, Width, Height, @Callback())
```

Description

Creates a new scintilla editing control in the current GadgetList.

Parameters

#Gadget A number to identify the new gadget. **#PB_Any** can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

@Callback() The address of a procedure to receive events from the control. It must be defined as follows, where 'Gadget' is the gadget number of the control and *scinotify points to a structure with information on the event:

```
1  Procedure ScintillaCallBack(Gadget, *scinotify.SCNotification)
2      ; You code here
3  EndProcedure
```

Return value

Nonzero on success, zero otherwise. If **#PB_Any** was used as the **#Gadget** parameter then the return-value is the auto-generated gadget number on success.

Remarks

The following events are supported through EventType() :

#PB_EventType_RightClick

After creation, Scintilla specific commands may be sent to the control with the ScintillaSendMessage() command. In addition common gadget commands like ResizeGadget() or HideGadget() may be used with the control as well.

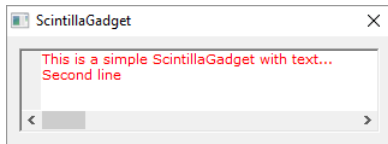
Example

```
1  If OpenWindow(0, 0, 0, 330, 90, "ScintillaGadget",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      ScintillaGadget(0, 10, 10, 320, 70, 0)
4
5      ; Output set to red color
6      ScintillaSendMessage(0, #SCI_STYLESETFORE, 0, RGB(255, 0, 0))
7
8      ; Set the initial text to the ScintillaGadget
9      *Text=UTF8("This is a simple ScintillaGadget with text...")
10     ScintillaSendMessage(0, #SCI_SETTEXT, 0, *Text)
11     FreeMemory(*Text) ; The buffer made by UTF8() has to be freed,
12     to avoid memory leak
13
14     ; Adding a second line of text with linebreak before
15     Text$ = Chr(10) + "Second line"
```

```

15     *Text=UTF8(Text$)
16     ScintillaSendMessage(0, #SCI_APPENDTEXT,
17     StringByteLength(Text$, #PB_UTF8), *Text)
18     FreeMemory(*Text)
19
20     Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
21     EndIf

```



156.3 ScintillaSendMessage

Syntax

```
Result = ScintillaSendMessage(#Gadget, Message [, Param [, LParam]])
```

Description

Sends a message to the scintilla control to perform a specific task.

Parameters

#Gadget The scintilla gadget to use.

Message The message to send. More information about the possible messages may be found on the [Scintilla Homepage](#). The #SCI_[...] constants representing the possible values are already defined in PureBasic.

Param (optional) The first custom parameter for the specified message. Default value if not specified is zero.

LParam (optional) The second custom parameter for the specified message. Default value if not specified is zero.

Return value

The result of the message sent.

Example: Text with red highlighting

```

1  #num_indicator = 0
2
3  If OpenWindow(0, 0, 0, 330, 90, "ScintillaGadget",
4      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
5
6      ScintillaGadget(0, 5, 5, 320, 80, 0)
7      *Text = UTF8("This is a simple ScintillaGadget with text..." +
8      #LF$ + "More text" + #LF$ + "Even more text!")
9      ScintillaSendMessage(0, #SCI_SETTEXT, 0, *Text)
10     FreeMemory(*Text) ; The buffer made by UTF8() has to be freed,
11     to avoid memory leak

```

```

10     ScintillaSendMessage(0, #SCI_INDICSETSTYLE, #num_indicator,
#INDIC_STRAIGHTBOX) ; first indicator with style 8 (0-19)
11     ScintillaSendMessage(0, #SCI_INDICSETFORE, #num_indicator,
#Red) ; first indicator with red color
12     ScintillaSendMessage(0, #SCI_SETINDICATORCURRENT,
#num_indicator, #INDIC_STRAIGHTBOX) ; set the indicator as the
current one
13     ScintillaSendMessage(0, #SCI_INDICSETUNDER, #num_indicator, 1)
; set indicator under the text, i.e. doesn't obscure it
14     ScintillaSendMessage(0, #SCI_INDICSETALPHA, #num_indicator,
127) ; set transparency
15     nLine = 1 ; line number starts from 0
16     start = ScintillaSendMessage(0, #SCI_POSITIONFROMLINE, nLine)
17     length = ScintillaSendMessage(0, #SCI_GETLINEENDPOSITION,
nLine) - start
18     ScintillaSendMessage(0, #SCI_INDICATORFILLRANGE, start, length)
; start and length
19
20     Repeat : Until WaitWindowEvent() = #PB_Event_CloseWindow
21 EndIf

```


Chapter 157

Screen

Overview

A screen is a surface used to display game accelerated content like sprites , or 3D worlds . A screen can be created either in a regular window , or using the whole display (fullscreen mode).

Windows: DirectX 9 is used for screen creation which your programs to use hardware acceleration if available. Two additional Subsystems are also available, depending of the needs: "OpenGL" and "DirectX11" which respectively leverage OpenGL and DirectX11 to handle screen creation.

Linux: OpenGL is used to manage the screen which allows to use hardware acceleration.

MacOS X: OpenGL is used to manage the screen which allows to use hardware acceleration.

157.1 ChangeGamma

Syntax

```
ChangeGamma( RedIntensity , GreenIntensity , BlueIntensity )
```

Description

Changes the Gamma for the current screen. This only works in full screen mode (not in windowed mode). Red, Green and Blue channels intensity can be changed individually. This function can be used to do full screen fade-in/fade-out, color splashing etc. If it does not do anything, then the hardware does not support it (no emulation is provided, due to the high number of operations required to perform it in software).

Parameters

RedIntensity, GreenIntensity, BlueIntensity The new intensity to apply. Valid values for each channel are from 0 to 255.

Return value

None.

Example

```
1  ChangeGamma(255, 255, 0) ; Will remove completely the blue  
   channel from the screen
```

Supported OS

Windows (DirectX)

157.2 ClearScreen

Syntax

```
ClearScreen(Color)
```

Description

Clear the whole screen with the specified color.

Parameters

Color The color to use to clear the screen. RGB() can be used to get a valid color value. A table with common colors is available [here](#) .

Return value

None.

Remarks

ClearScreen() has to be always called outside a StartDrawing() : StopDrawing() block.

157.3 CloseScreen

Syntax

```
CloseScreen()
```

Description

Close the current screen (either windowed or full screen mode). After closing a screen, all the sprites must be reloaded as the screen format has been lost and the video memory released. An application or game can switch from full screen to windowed mode on the fly without any problem.

Parameters

None.

Return value

None.

See Also

OpenScreen() , OpenWindowedScreen()

157.4 FlipBuffers

Syntax

```
FlipBuffers()
```

Description

Flip the back and front buffers of the current screen. The invisible area is now visible and vice versa, which allowss to do a 'double-buffering' effect (flicker free graphical displays). A screen must have been opened with `OpenScreen()` or `OpenWindowedScreen()` . The way the buffer are flipped (with or without synchronization) is set by `OpenScreen()` or `OpenWindowedScreen()` .

Parameters

None.

Return value

None.

Remarks

`FlipBuffers()` has to be called outside a `StartDrawing() : ... : StopDrawing()` program block. `FlipBuffers()` must be called in each event loop on a fullscreen to handle events for joystick / keyboard " / mouse properly.

See Also

`OpenScreen()` , `OpenWindowedScreen()`

157.5 IsScreenActive

Syntax

```
Result = IsScreenActive()
```

Description

Games and full screen applications using PureBasic functions run under a multitasking environment. This means that the user can switch back from full screen to the normal desktop. This change can be detected with this function and appropriate actions should be taken, such as `ReleaseMouse()` , pause the game, stop the sounds etc. This function must be called after `FlipBuffers()` , as the events are handled inside `FlipBuffers()` .

Parameters

None.

Return value

Nonzero if the screen is still active, zero otherwise.

See Also

OpenScreen() , OpenWindowedScreen() , ReleaseMouse() , FlipBuffers()

157.6 ScreenID

Syntax

```
Result = ScreenID()
```

Description

Returns the OS ScreenID.

Parameters

None.

Return value

The OS ScreenID. On Windows this is a normal WindowID so any function which needs such an ID can use it, like PlayMovie() for example.

Supported OS

Windows (DirectX), Linux (SDL)

157.7 ScreenWidth

Syntax

```
Result = ScreenWidth()
```

Description

Returns the current screen width, previously opened with OpenScreen() or OpenWindowedScreen() .

Parameters

None.

Return value

The current screen width, or zero if no screen is opened.

See Also

OpenScreen() , OpenWindowedScreen() , ScreenHeight() , ScreenDepth()

157.8 ScreenHeight

Syntax

`Result = ScreenHeight()`

Description

Returns the current screen height, previously opened with `OpenScreen()` or `OpenWindowedScreen()` .

Parameters

None.

Return value

The current screen height, or zero if no screen is opened.

See Also

`OpenScreen()` , `OpenWindowedScreen()` , `ScreenWidth()` , `ScreenDepth()`

157.9 ScreenDepth

Syntax

`Result = ScreenDepth()`

Description

Returns the current screen depth, previously opened with `OpenScreen()` or `OpenWindowedScreen()` .

Parameters

None.

Return value

The current screen depth, or zero if no screen is opened. Depth is a value between 8 and 32.

See Also

`OpenScreen()` , `OpenWindowedScreen()` , `ScreenWidth()` , `ScreenHeight()`

157.10 SetFrameRate

Syntax

```
SetFrameRate(FrameRate)
```

Description

Set the frame rate (in frames per second) for the current screen. This is especially useful for windowed screen mode where there is no refresh rate for the screen. This function sets the maximum number of times per second that the FlipBuffers() function is called.

Parameters

FrameRate The new framerate to set.

Return value

None.

See Also

OpenScreen() , OpenWindowedScreen() , FlipBuffers()

157.11 OpenScreen

Syntax

```
Result = OpenScreen(Width, Height, Depth, Title$ [, FlipMode [,  
RefreshRate]])
```

Description

Opens a new screen according to the specified 'Width', 'Height' and 'Depth'. InitSprite() has to be called successfully before using this command. The opened screen is created with 2 video buffers to allow double buffering, especially useful for games. The buffers can be manipulated with the FlipBuffers() function.

Parameters

Width, Height The screen resolution, in pixels. The specified resolution has to be supported or the screen won't be created. ExamineScreenModes() can be used to get a full list of supported resolution.

Depth It can be one the following values:

```
16: 65000 colors, fixed palette  
24: 16 Mo colors, fixed palette  
32: 16 Mo colors, faster than 24-bit mode, allows alpha  
blending
```

Title\$ The title for the application which will be displayed when switching back to the desktop. (Windows only). It will be displayed in the taskbar, so it's recommended to use a title related to the application name.

FlipMode (optional) Sets the screen synchronization methods used when flipping buffers (also known as 'Vertical blank synchronization'). It can have one of the following values:

```
#PB_Screen_NoSynchronization    : disable synchronization
#PB_Screen_WaitSynchronization : enable synchronization
                                (default value)
#PB_Screen_SmartSynchronization: enable synchronization,
                                with a CPU saver routine when the program doesn't consume
                                all the CPU time (full
screen mode only)
```

Waiting for the screen synchronization allows the flip to be perfect (no 'tearing' or other visible artifacts) because the flip is performed when the screen has been fully drawn (and when the screen spot is outside of visible screen area). This also link the flip frequency to the actual screen refresh, ie: for 60Hz screen it could have at most 60 flip per seconds, etc.

RefreshRate (optional) Set the refresh rate (in Hz) for the new screen. If it can't be handled, then `OpenScreen()` will fail. `ExamineScreenModes()` can be used to get a full list of supported refresh rates.

Note: on Windows, the refresh rate could be locked or forced by the video card drivers, so it could be inaccurate.

Return value

Nonzero if the screen has been successfully opened, zero otherwise.

Remarks

The Requester functions cannot be used on screens created with `OpenScreen`. To open a screen area on a regular window, see `OpenWindowedScreen()` .

See Also

`OpenWindowedScreen()` , `FlipBuffers()`

157.12 OpenWindowedScreen

Syntax

```
Result = OpenWindowedScreen(WindowID, x, y, Width, Height [,
    AutoStretch, RightOffset, BottomOffset [, FlipMode]])
```

Description

Open a new screen area according to given parameters on the given Window, which must be opened before using `OpenWindow()` . `InitSprite()` has to be called successfully before using this command. The "windowed screen" is able to use the hardware acceleration the same way than full-size `OpenScreen()` function. As a window is opened, the window events have to be processed with `WindowEvent()` to have the correct behaviour. All the events needs to be processed before flipping the buffers (see the examples below).

Parameters

WindowID The window to use to create the screen. `WindowID()` can be used to get a valid window identifier.

x, y The screen position in the specified window, in pixels.

Width, Height The screen resolution, in pixels.

AutoStretch (optional) If set to `#True`, then the screen area will automatically resize (and its contents will be zoomed to new screen size) when the window size changes.

RightOffset, BottomOffset (optional) Define the margin on the right and bottom of the window, in pixels (for statusbar for example).

FlipMode (optional) Sets the screen synchronization methods used when flipping buffers (also known as 'Vertical blank synchronization'). It can have one of the following values:

```
#PB_Screen_NoSynchronization    : disable synchronization
#PB_Screen_WaitSynchronization : enable synchronization
                                (default value)
#PB_Screen_SmartSynchronization: enable synchronization,
                                with a CPU saver routine when the program doesn't consume
                                all the CPU time (full screen mode only)
```

Waiting for the screen synchronization allows the flip to be perfect (no 'tearing' or other visible artifacts) because the flip is performed when the screen has been fully drawn (and when the screen spot is outside of visible screen area). This also link the flip frequency to the actual screen refresh, ie: for 60Hz screen it could have at most 60 flip per seconds, etc. The opened screen is created with 2 video buffers to allow you to do double buffering, especially useful for games. The buffers can be manipulated with the `FlipBuffers()` function.

Return value

Nonzero if the screen has been successfully opened, zero otherwise.

Remarks

Only one windowed screen can be opened at one time. The screen dimension can't be greater than the window size or artefacts can occurs.

Example: Fixed screen size with gadgets

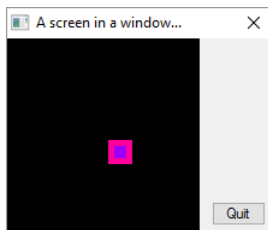
```
1  If InitSprite() = 0
2      MessageRequester("Error", "Can't open screen & sprite
   environment!", 0)
3      End
4  EndIf
5
6  If OpenWindow(0, 0, 0, 220, 160, "A screen in a window...",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
7      ButtonGadget(0, 170, 135, 45, 20, "Quit")
8
9      If OpenWindowedScreen(WindowID(0), 0, 0, 160, 160)
10         CreateSprite(0, 20, 20)
11         If StartDrawing(SpriteOutput(0))
12             Box(0, 0, 20, 20, RGB(255, 0, 155))
13             Box(5, 5, 10, 10, RGB(155, 0, 255))
14             StopDrawing()
15         EndIf
16     Else
17         MessageRequester("Error", "Can't open windowed screen!", 0)
18     End
19 EndIf
20 EndIf
21
22 direction = 2
23 Repeat
```



```

24      ; It's very important to process all the events remaining in
the queue at each frame
25      ;
26      Repeat
27          Event = WindowEvent()
28
29          Select Event
30              Case #PB_Event_Gadget
31                  If EventGadget() = 0
32                      End
33                  EndIf
34
35              Case #PB_Event_CloseWindow
36                  End
37          EndSelect
38      Until Event = 0
39
40      FlipBuffers()
41      ClearScreen(RGB(0, 0, 0))
42      DisplaySprite(0, x, x)
43      x + direction
44      If x > 140 : direction = -2 : EndIf
45      If x < 0   : direction = 2 : EndIf
46      Delay(1)
47  ForEver

```



Example: Screen with enabled auto-stretch and bottom-offset feature

```

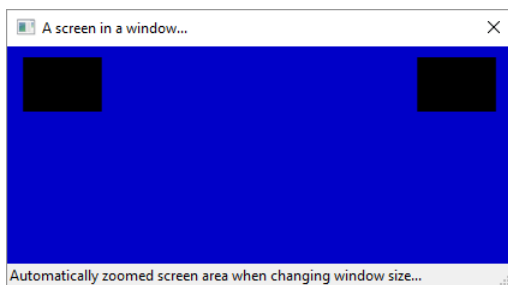
1  If InitSprite() = 0
2      MessageRequester("Error", "Can't open screen & sprite
environment!", 0)
3      End
4  EndIf
5
6  If OpenWindow(0, 0, 0, 420, 200, "A screen in a window...",
#PB_Window_SystemMenu | #PB_Window_SizeGadget |
#PB_Window_ScreenCentered)
7      CreateStatusBar(0, WindowID(0))
8      AddStatusBarField(420)
9
10     StatusBarText(0, 0, "Automatically zoomed screen area when
changing window size...")
11
12     If OpenWindowedScreen(WindowID(0), 0, 0, 420, 200, #True, 0, 20)
13
14         CreateSprite(0, 50, 50) ; Create an empty sprite, will be
whole black
15
16     Repeat

```

```

17      ; It's very important to process all the events remaining
    in the queue at each frame
18      ;
19      Repeat
20          Event = WaitWindowEvent(10)
21
22          If Event = #PB_Event_CloseWindow
23              End
24          EndIf
25          Until Event = 0
26
27          FlipBuffers()
28          ClearScreen(RGB(0, 0, 200)) ; A blue background
29
30          DisplaySprite(0, 10, 10) ; Display our black box at the
    left-top corner
31          DisplaySprite(0, 260, 10) ; Display our black box at the
    right-top corner
32      ForEver
33
34      Else
35          MessageRequester("Error", "Can't open windowed screen!", 0)
36      EndIf
37  EndIf

```



For a more detailed example look at

See Also

[OpenScreen\(\)](#)

157.13 ScreenOutput

Syntax

```
OutputID = ScreenOutput()
```

Description

Returns the OutputID of the currently used screen to perform 2D rendering operations on it. It will use the PureBasic 2DDrawing library and can only be used within a StartDrawing() / StopDrawing() block. The memory allocated in ScreenOutput() is released on StopDrawing().

Parameters

None.

Return value

The OutputID of the currently used screen to perform 2D rendering operations on it.

Remarks

On Linux and OS X, ScreenOutput() copies the whole screen buffer back to main memory to do 2D drawing operations (OpenGL doesn't allow direct buffer access). Therefore drawing on a screen is very slow and should be avoided. ScreenOutput() has to be called in the same thread where OpenScreen() was called.

The use of InitEngine3D() is forbidden, ScreenOutput() returns 0.

Example

```
1  StartDrawing(ScreenOutput())
2      ; do some drawing stuff here...
3  StopDrawing()
```

157.14 ExamineScreenModes

Syntax

```
Result = ExamineScreenModes()
```

Description

Starts to examine the available screen modes on the local computer. The screen modes list can be retrieved with the help of the NextScreenMode() function.

Parameters

None.

Return value

Nonzero if the screen modes has been successfully listed, zero otherwise.

See Also

NextScreenMode()

157.15 NextScreenMode

Syntax

```
Result = NextScreenMode()
```

Description

This function should be called after ExamineScreenModes() . It will go step-by-step into the screen modes list. The current screen mode information can be retrieved with the following functions: ScreenModeWidth() , ScreenModeHeight() , ScreenModeDepth() and ScreenModeRefreshRate() .

Parameters

None.

Return value

Nonzero if there is another screenmode to be listed, zero otherwise.

Example

```
1  InitSprite()
2
3  If ExamineScreenModes()
4      While NextScreenMode()
5          Debug
6              Str(ScreenModeWidth())+"x"+Str(ScreenModeHeight())+"x"+Str(ScreenModeDepth())
7          Wend
8      EndIf
```

See Also

ExamineScreenModes() , ScreenModeWidth() , ScreenModeHeight() , ScreenModeDepth() ,
ScreenModeRefreshRate()

157.16 ScreenModeDepth

Syntax

Depth = ScreenModeDepth()

Description

Returns the depth of the current screenmode listed with ExamineScreenModes() and NextScreenMode() functions.

Parameters

None.

Return value

The depth of the current listed screenmode. Depth can be either 8, 15, 16, 24 or 32-bit depending on the graphic card.

See Also

ExamineScreenModes() , NextScreenMode() , ScreenModeWidth() , ScreenModeHeight() ,
ScreenModeRefreshRate()

157.17 ScreenModeHeight

Syntax

```
Height = ScreenModeHeight()
```

Description

Returns the height of the current screenmode listed with `ExamineScreenModes()` and `NextScreenMode()` functions.

Parameters

None.

Return value

The height (in pixels) of the current listed screenmode.

See Also

`ExamineScreenModes()` , `NextScreenMode()` , `ScreenModeWidth()` , `ScreenModeDepth()` , `ScreenModeRefreshRate()`

157.18 ScreenModeRefreshRate

Syntax

```
RefreshRate = ScreenModeRefreshRate()
```

Description

Returns the refresh-rate of the current screenmode listed with `ExamineScreenModes()` and `NextScreenMode()` functions.

Parameters

None.

Return value

The refresh-rate (in hertz) of the current listed screenmode.

Remarks

On OS X, many notebook and monitors returns 0.

See Also

`ExamineScreenModes()` , `NextScreenMode()` , `ScreenModeWidth()` , `ScreenModeHeight()` , `ScreenModeDepth()`

157.19 ScreenModeWidth

Syntax

```
Width = ScreenModeWidth()
```

Description

Returns the width of the current screenmode listed with `ExamineScreenModes()` and `NextScreenMode()` functions.

Parameters

None.

Return value

The width (in pixels) of the current listed screenmode.

See Also

`ExamineScreenModes()` , `NextScreenMode()` , `ScreenModeHeight()` , `ScreenModeDepth()` , `ScreenModeRefreshRate()`

Chapter 158

SerialPort

Overview

The Serial port (also know as the RS-232 port) was first created in 1969, and despite its age, it is still widely used in the industry. The process of computer driven hardware prototype creation is simplified when the serial port is used. This library offers full access to the serial ports available on the computer. Some basic knowledge about the terms used in this documentation will be needed, check [Wikipedia - RS232](#) for more information.

158.1 AvailableSerialPortInput

Syntax

```
Result = AvailableSerialPortInput (#SerialPort)
```

Description

Returns the number of remaining bytes in the serial port input buffer.

Parameters

#SerialPort The serial port to use.

Return value

The number of remaining bytes in the serial port input buffer.

See Also

AvailableSerialPortOutput()

158.2 AvailableSerialPortOutput

Syntax

```
Result = AvailableSerialPortOutput (#SerialPort)
```

Description

Returns the number of remaining bytes in the serial port output buffer.

Parameters

#SerialPort The serial port to use.

Return value

The number of remaining bytes in the serial port output buffer.

See Also

AvailableSerialPortInput()

158.3 CloseSerialPort

Syntax

```
CloseSerialPort (#SerialPort)
```

Description

Closes the serial port previously opened with OpenSerialPort() .

Parameters

#SerialPort The serial port to close. If **#PB_All** is specified, all the remaining serial ports are freed.

Return value

None.

Remarks

All remaining opened serial ports are automatically closed when the program ends.

158.4 GetSerialPortStatus

Syntax

```
Result = GetSerialPortStatus (#SerialPort, Attribute)
```

Description

Returns the specified serial port status.

Parameters

#SerialPort The serial port to use.

Attribute It can be one of the following values:


```
#PB_SerialPort_RI : Get RI signal status (0 or 1)
#PB_SerialPort_DCD: Get DCD signal status (0 or 1)
#PB_SerialPort_DSR: Get DSR signal status (0 or 1)
#PB_SerialPort_CTS: Get CTS signal status (0 or 1)
#PB_SerialPort_XonCharacter : Character used for Xon/Xoff
handshaking sequence (between 1 and 255)
#PB_SerialPort_XoffCharacter: Character used for Xon/Xoff
handshaking sequence (between 1 and 255)
```

Return value

Returns the specified serial port status, according to the specified attribute.

158.5 IsSerialPort

Syntax

```
Result = IsSerialPort(#SerialPort)
```

Description

Tests if the given serial port is valid and correctly initialized.

Parameters

#SerialPort The serial port to test.

Return value

Nonzero if the serial port is valid, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a serial port is ready to use.

See Also

OpenSerialPort()

158.6 SerialPortError

Syntax

```
Result = SerialPortError(#SerialPort)
```

Description

Returns the error on the serial port when ReadSerialPortData() , WriteSerialPortData() or WriteSerialPortString() failed.

Parameters

#SerialPort The serial port to use.

Return value

Can be a combination of the following values:

```
#PB_SerialPort_RxOver: An input buffer overflow has occurred.  
                        There is either no room in the input buffer  
                        or a character was received after the  
                        end-of-file (EOF)  
#PB_SerialPort_OverRun: A character-buffer overrun has occurred.  
                        The next character is lost.  
#PB_SerialPort_RxParity : The hardware detected a parity error.  
#PB_SerialPort_Frame   : The hardware detected a framing error.  
#PB_SerialPort_Break    : The hardware detected a break condition.  
#PB_SerialPort_TxFull   : The application tried to transmit a  
                        character but the output buffer was full.  
#PB_SerialPort_IOE      : An I/O error occurred during  
                        communications with the device.  
#PB_SerialPort_WaitingCTS : Specifies whether transmission is  
                        waiting for the CTS (clear-to-send) signal to be sent.  
#PB_SerialPort_WaitingDSR : Specifies whether transmission is  
                        waiting for the DSR (data-set-ready) signal to be sent.  
#PB_SerialPort_WaitingRLSD : Specifies whether transmission is  
                        waiting for the RLSD (receive-line-signal-detect) signal to be  
                        sent.  
#PB_SerialPort_XoffReceived: Specifies whether transmission is  
                        waiting because the XOFF character was received.  
#PB_SerialPort_XoffSent    : Specifies whether transmission is  
                        waiting because the XOFF character was transmitted.  
                        Transmission halts when the XOFF  
                        character is transmitted to a system that takes the next  
                        character as XON, regardless of the  
                        actual character.  
#PB_SerialPort_EOFSent: Specifies whether the end-of-file (EOF)  
                        character has been received.
```

See Also

ReadSerialPortData() , WriteSerialPortData() , WriteSerialPortString()

158.7 SerialPortID

Syntax

```
SerialPortID = SerialPortID(#SerialPort)
```

Description

Returns the unique system identifier of the serial port.

Parameters

#SerialPort The serial port to use.

Return value

Returns the ID of the serial port. This function is useful when another library or API commands requires a serial port reference.

See Also

OpenSerialPort()

158.8 OpenSerialPort

Syntax

```
Result = OpenSerialPort(#SerialPort, SerialPortName$, Bauds,  
    Parity, Data, Stop.f, HandshakeMode, InputBufferSize,  
    OutputBufferSize)
```

Description

Opens a serial port for use.

Parameters

#SerialPort A number to identify the new serial port. #PB_Any can be used to auto-generate this number.

SerialPortName\$ Text identifier for the serial port, for example "COM1" on Windows or "/dev/ttyS0" on Linux.

Bauds Defines the speed of the serial connection and can be one of the following values:

50, 75, 110, 150, 300, 600, 1200, 1800, 2400
4800, 9600, 19200, 38400, 57600 or 115200

Parity Defines how the parity will be handled on the connection. It can be one of the following values:

```
#PB_SerialPort_NoParity      : No parity  
#PB_SerialPort_EvenParity   : Even parity  
#PB_SerialPort_MarkParity   : Mark parity  
#PB_SerialPort_OddParity    : Odd parity  
#PB_SerialPort_SpaceParity  : Space parity
```

Data Defines the data length, in byte (usually 7 or 8).

Stop Sets the number of stop bits (1, 1.5 or 2).

HandshakeMode The handshake mode. It can be one of the following values:

```
#PB_SerialPort_NoHandshake      : No handshaking  
#PB_SerialPort_RtsHandshake     : No handshaking but RTS is  
    set to 1  
#PB_SerialPort_RtsCtsHandshake  : RTS/CTS  
#PB_SerialPort_XonXoffHandshake : Xon/Xoff
```

InputBufferSize Defines the size of the input buffer, in bytes.

OutputBufferSize Defines the size of the output buffer, in bytes.

Return value

Nonzero if the serial port was successfully opened, zero otherwise (it may be already in use, or the parameters are wrong). If #PB_Any was used for the #SerialPort parameter then the generated number is returned on success.

Example

```
1  If OpenSerialPort(0, "COM1", 300, #PB_SerialPort_NoParity, 8, 1,  
    #PB_SerialPort_NoHandshake, 1024, 1024)  
2      Debug "Success"  
3  Else  
4      Debug "Failed"  
5  EndIf
```

See Also

CloseSerialPort()

158.9 ReadSerialPortData

Syntax

```
Result = ReadSerialPortData(#SerialPort, *Buffer, Length)
```

Description

Reads an arbitrary amount of data from the #SerialPort. If the input buffer was empty, this function will block until data is available. To check if data is available, use AvailableSerialPortInput().

Parameters

#SerialPort The serial port to use.

***Buffer** The memory address to use to put the read data.

Length The length to read from the serial port, in bytes. The specified buffer should be large enough to handle it.

Return value

The number of bytes which are actually read. It can be less than the requested length. If a read error has occurred, it will return zero.

See Also

AvailableSerialPortInput()

158.10 SerialPortTimeouts

Syntax

```
SerialPortTimeouts(#SerialPort, RIT, RTTC, RTTM, WTTC, WTTM)
```

Description

Changes the default serial port timeouts.

Parameters

#SerialPort The serial port to use.

RIT RIT stands for 'ReadIntervalTimeout'. Specifies the maximum time, in milliseconds, which are allowed to elapse between the arrival of two characters on the communications line. Default value is 100 ms.

RTTC RTTC stands for 'ReadTotalTimeoutConstant'. Specifies the constant, in milliseconds, used to calculate the total time-out period for read operations. For each read operation, this value is added to the product of the ReadTotalTimeoutMultiplier member and the requested number of bytes. Default value is 100 ms.

RTTM RTTM stands for 'ReadTotalTimeoutMultiplier'. Specifies the multiplier, in milliseconds, used to calculate the total time-out period for read operations. For each read operation, this value is multiplied by the requested number of bytes to be read. Default value is 10 ms.

W TTC W TTC stands for 'WriteTotalTimeoutConstant'. Specifies the constant, in milliseconds, used to calculate the total time-out period for write operations. For each write operation, this value is added to the product of the WriteTotalTimeoutMultiplier member and the number of bytes to be written. Default value is 10 ms.

W TTM W TTM stands for 'WriteTotalTimeoutMultiplier'. Specifies the multiplier, in milliseconds, used to calculate the total time-out period for write operations. For each write operation, this value is multiplied by the number of bytes to be written. Default value is 100 ms.

Return value

None.

See Also

OpenSerialPort()

158.11 SetSerialPortStatus

Syntax

```
SetSerialPortStatus(#SerialPort, Attribute, Value)
```

Description

Changes the specified serial port status.

Parameters

#SerialPort The serial port to use.

Attribute The attribute to set. It can be one of the following value:

```
#PB_SerialPort_DTR: set DTR signal status (value can be 0 or 1)
#PB_SerialPort_RTS: set RTS signal status (value can be 0 or 1)
#PB_SerialPort_TXD: Set TXD signal status (value can be 0 or 1)
#PB_SerialPort_XonCharacter : Character used for Xon/Xoff handshaking sequence (value can be between 1 and 255).
Default value is $11.
```

#PB_SerialPort_XoffCharacter: Character used for Xon/Xoff handshaking sequence (value can be between 1 and 255).
Default value is \$13.

Value The attribute value to set. See 'attribute' description to see which value is applicable.

Return value

None.

Example

```
1  If OpenSerialPort(0, "COM1", 300, #PB_SerialPort_NoParity, 8, 1,  
    #PB_SerialPort_XonXoffHandshake, 1024, 1024)  
2      SetSerialPortStatus(0, #PB_SerialPort_XonCharacter, 8)  
3      SetSerialPortStatus(0, #PB_SerialPort_XoffCharacter, 9)  
4      Debug "Success"  
5  Else  
6      Debug "Failed"  
7  EndIf
```

See Also

OpenSerialPort()

158.12 WriteSerialPortData

Syntax

```
Result = WriteSerialPortData(#SerialPort, *Buffer, Length)
```

Description

Writes an arbitrary amount of data to the specified serial port.

Parameters

#SerialPort The serial port to use.

***Buffer** The buffer to write.

Length The buffer length to write, in bytes.

Return value

The number of bytes written to the serial port, or zero if the operation has failed.

Remarks

To check how much data is available in the output buffer, use AvailableSerialPortOutput() .

See Also

OpenSerialPort() , WriteSerialPortString()

158.13 WriteSerialPortString

Syntax

```
Result = WriteSerialPortString(#SerialPort, String$ [, Format])
```

Description

Writes a string to the specified serial port.

Parameters

#SerialPort The serial port to use.

String\$ The string to write.

Format (optional) The string format to use. It can be one of the following value:

```
#PB_Ascii      : the string will be written in ascii format.
#PB_UTF8       : the string will be written in UTF8 format
                 (default).
#PB_Unicode    : the string will be written in unicode (UTF16)
                 format.
```

Return value

The number of bytes written to the serial port, or zero if the operation has failed.

Remarks

To check how much data is available in the output buffer, use AvailableSerialPortOutput() .

See Also

OpenSerialPort() , WriteSerialPortData()

Chapter 159

Skeleton

Overview

Skeleton are composed of bones and can associated to a mesh to animate it in a realistic manner. `InitEngine3D()` must be called successfully before using the Skeleton functions.

159.1 CreateSkeleton

Syntax

```
CreateSkeleton(#Mesh)
```

Description

Creates or replaces the skeleton of the `#Mesh`.

Parameters

`#Mesh` The number to identify the new mesh. `#PB_Any` can be used to auto-generate this number.

Return value

None.

159.2 CreateBone

Syntax

```
CreateBone(#Mesh, Bone$, ParentBone$, x, y, z, RotationX,  
           RotationY, RotationZ, RotationW, Mode)
```

Description

Creates a new bone for the specified `#Mesh`. If the mesh doesn't have a skeleton, it has to be created with `CreateSkeleton()` before using this command.

Parameters

#Mesh The mesh to use.

Bone\$ The new bone name.

ParentBone\$ The parent bone name. If this string value is empty, the bone is created at the root of the skeleton.

x, y, z The new bone coordinate, in world unit.

RotationX, RotationY, RotationZ, RotationW The new bone rotation.

Mode The mode to use for the bone rotation. It can be one of the following value:

```
#PB_Orientation_PitchYawRoll : 'RotationX' (pitch),  
    'RotationY' (yaw), 'RotationZ' (roll), applied in this order  
    (default).  
#PB_Orientation_Quaternion    : 'RotationX', 'RotationY',  
    'RotationZ', 'RotationW' for quaternion values  
#PB_Orientation_AngleAxis     : 'RotationX', 'RotationY',  
    'RotationZ' for rotation axis, and 'RotationW' for the angle  
    (in degree).  
#PB_Orientation_Direction     : alias for  
    #PB_Orientation_DirectionLDVZN  
#PB_Orientation_DirectionLDVX : 'RotationX', 'RotationY',  
    'RotationZ' for direction vector (local direction vector X),  
    and 'RotationW' for angle (in degree).  
#PB_Orientation_DirectionLDVY : 'RotationX', 'RotationY',  
    'RotationZ' for direction vector (local direction vector Y),  
    and 'RotationW' for angle (in degree).  
#PB_Orientation_DirectionLDVZ : 'RotationX', 'RotationY',  
    'RotationZ' for direction vector (local direction vector Z),  
    and 'RotationW' for angle (in degree).  
#PB_Orientation_DirectionLDVXN: 'RotationX', 'RotationY',  
    'RotationZ' for direction vector (local direction vector  
    -X), and 'RotationW' for angle (in degree).  
#PB_Orientation_DirectionLDVYN: 'RotationX', 'RotationY',  
    'RotationZ' for direction vector (local direction vector  
    -Y), and 'RotationW' for angle (in degree).  
#PB_Orientation_DirectionLDVZN: 'RotationX', 'RotationY',  
    'RotationZ' for direction vector (local direction vector  
    -Z), and 'RotationW' for angle (in degree).
```

Return value

None.

159.3 VertexBoneAssignment

Syntax

```
VertexBoneAssignment(#Mesh, SubMesh, VertexIndex, BoneIndex,  
    Weight.f)
```

Description

Assign a vertex to a bone (the same vertex can be assigned to several bones: the sum of bone weight must be equals to 1). Once the vertex assignment is finished, FinishBoneAssignment() has to be called.

Parameters

#Mesh The mesh to use.

SubMesh The submesh to get the data from. The first submesh index is 0 (main mesh).

VertexIndex The vertex index. The first index is 0.

BoneIndex The bone index, in creation order. The first index is 0.

Weight The bone weight, must be between 0 and 1.

Return value

None.

See Also

`FinishBoneAssignment()`

159.4 FinishBoneAssignment

Syntax

```
FinishBoneAssignment(#Mesh, SubMesh)
```

Description

Finish bone assignments for a mesh, previously started with `VertexBoneAssignment()` .

Parameters

#Mesh The mesh to use.

SubMesh The submesh to use. The first submesh index is 0 (main mesh).

Return value

None.

See Also

`VertexBoneAssignment()`

159.5 CreateSkeletonAnimation

Syntax

```
CreateSkeletonAnimation(#Mesh, AnimationName$, Length)
```

Description

Creates a new skeleton animation. A skeleton has to be created for this mesh with `CreateSkeleton()` . The new animation is empty and steps have to be created with `AddSkeletonAnimationKeyFrame()` .

Parameters

#Mesh The mesh to use.

AnimationName\$ The name of the animation.

Length The length of the animation (in milliseconds).

Return value

None.

See Also

AddSkeletonAnimationKeyFrame() , CreateSkeleton()

159.6 AddSkeletonAnimationKeyFrame

Syntax

```
AddSkeletonAnimationKeyFrame(#Mesh, AnimationName$, Bone$, Time,
    RotationX, RotationY, RotationZ, RotationW, Mode [, TranslateX,
    TranslateY, TranslateZ [, ScaleX, ScaleY, ScaleZ]])
```

Description

Creates a new step for the specified animation. A skeleton animation can be created with CreateSkeletonAnimation() .

Parameters

#Mesh The mesh to use.

AnimationName\$ The name of the animation to add the step.

Bone\$ The bone name to use.

Time The time in the animation to set the keyframe (in milliseconds). This value has to be between zero and the 'Length' defined in CreateSkeletonAnimation() .

RotationX, RotationY, RotationZ, RotationW The bone rotation for this step.

Mode The mode to use for the bone rotation. It can be one of the following value:

```
#PB_Orientation_PitchYawRoll : 'RotationX' (pitch),
    'RotationY' (yaw), 'RotationZ' (roll), applied in this order
    (default).
#PB_Orientation_Quaternion    : 'RotationX', 'RotationY',
    'RotationZ', 'RotationW' for quaternion values
#PB_Orientation_AngleAxis     : 'RotationX', 'RotationY',
    'RotationZ' for rotation axis, and 'RotationW' for the angle
    (in degree).
#PB_Orientation_Direction     : alias for
    #PB_Orientation_DirectionLDVZN
#PB_Orientation_DirectionLDVX : 'RotationX', 'RotationY',
    'RotationZ' for direction vector (local direction vector X),
    and 'RotationW' for angle (in degree).
#PB_Orientation_DirectionLDVY : 'RotationX', 'RotationY',
    'RotationZ' for direction vector (local direction vector Y),
    and 'RotationW' for angle (in degree).
```

```

#PB_Orientation_DirectionLDVZ : 'RotationX', 'RotationY',
    'RotationZ' for direction vector (local direction vector Z),
    and 'RotationW' for angle (in degree).
#PB_Orientation_DirectionLDVXN: 'RotationX', 'RotationY',
    'RotationZ' for direction vector (local direction vector
    -X), and 'RotationW' for angle (in degree).
#PB_Orientation_DirectionLDVYN: 'RotationX', 'RotationY',
    'RotationZ' for direction vector (local direction vector
    -Y), and 'RotationW' for angle (in degree).
#PB_Orientation_DirectionLDVZN: 'RotationX', 'RotationY',
    'RotationZ' for direction vector (local direction vector
    -Z), and 'RotationW' for angle (in degree).

```

TranslateX, TranslateY, TranslateZ (optional) The bone translation for this step.

ScaleX, ScaleY, ScaleZ (optional) The bone scale for this step.

Return value

None.

See Also

CreateSkeletonAnimation()

Chapter 160

Sort

Overview

Sometimes, elements have to be sorted to be usable or more convenient. PureBasic offers highly optimized functions in order to sort arrays and lists, either in ascending or descending order. `SortStructuredList` and `SortList` use Mergesort which is a stable sort, so if you first sort all the list by titles and then again by album you will get a list which is sorted by album and each album is sorted by title.

Note that this does not work with Arrays, as `SortArray` uses Quicksort which is unstable. (ie. the sorting of the secondary key would be lost)

Furthermore there are functions to reorder the elements of an array or a list in a random order.

160.1 CustomSortArray

Syntax

```
CustomSortArray(ArrayName(), @CompareProcedure() [, Options [,
    Start, End]])
```

Description

Sorts the specified array, according to the given options using a custom procedure to compare array elements. The array may have a structure or one of basic type: byte, word, long, integer, string or float. Multi-dimensioned arrays are not supported.

Parameters

ArrayName() The array to sort.

@CompareProcedure() The address of a procedure to compare array elements. The procedure must have two parameters that will receive the memory address of the two elements to compare and return one of the following values:

```
#PB_Sort_Greater: The first element is greater than the
second one.
#PB_Sort_Equal  : Both elements are equal.
#PB_Sort_Lesser : The first element is lesser than the second
one.
```

The compare procedure is called very often during the sorting process so it should do as little work as possible and be fast to execute.

Options (optional) It can be a combination of the following values:

`#PB_Sort_Ascending` : Sort the array in ascending order (lower values first). This is the **default**.
`#PB_Sort_Descending`: Sort the array in descending order (higher values first)

Start, End (optional) The index of the first and last element in the array that should be sorted.
 If these parameters are not specified, then the whole array is sorted.

Return value

None.

Example: Sort structured array

```

1  Structure CustomStruct
2      Value$
3  EndStructure
4
5  ; Sort by numeric value in the string element in the structure
6  Procedure CustomCompare(*a.CustomStruct, *b.CustomStruct)
7      If Val(*a\Value$) < Val(*b\Value$)
8          ProcedureReturn #PB_Sort_Lesser
9      ElseIf Val(*a\Value$) > Val(*b\Value$)
10         ProcedureReturn #PB_Sort_Greater
11     Else
12         ProcedureReturn #PB_Sort_Equal
13     EndIf
14 EndProcedure
15
16 Dim Test.CustomStruct(6)
17
18 Test(0)\Value$ = "0005"
19 Test(1)\Value$ = "2"
20 Test(2)\Value$ = "42"
21 Test(3)\Value$ = "7"
22 Test(4)\Value$ = "23"
23 Test(5)\Value$ = "100"
24 Test(6)\Value$ = "101"
25
26 CustomSortArray(Test(), @CustomCompare(), #PB_Sort_Ascending)
27
28 For i = 0 To 6
29     Debug Test(i)\Value$
30 Next
  
```

Example: Sort basic type array

```

1  ; Sort float elements by absolute value (ignoring the sign)
2  Procedure AbsCompare(*a.Float, *b.Float)
3      If Abs(*a\f) < Abs(*b\f)
4          ProcedureReturn #PB_Sort_Lesser
5      ElseIf Abs(*a\f) > Abs(*b\f)
6          ProcedureReturn #PB_Sort_Greater
7      Else
8          ProcedureReturn #PB_Sort_Equal
9      EndIf
  
```

```

10 EndProcedure
11
12 Dim Test.f(4)
13 Test(0) = 2.5
14 Test(1) = -3.0
15 Test(2) = 0
16 Test(3) = 12
17 Test(4) = -100
18
19 CustomSortArray(Test(), @AbsCompare())
20
21 For i = 0 To 4
22     Debug Test(i)
23 Next

```

See Also

SortArray() , SortStructuredArray() , RandomizeArray()

160.2 CustomSortList

Syntax

```
CustomSortList(ListName(), @CompareProcedure() [, Options [, Start,
End]])
```

Description

Sorts the specified list , according to the given options using a custom procedure to compare list elements. The list may have a structure or one of basic type : byte, word, long, integer, string or float.

Parameters

ListName() The list to sort.

@CompareProcedure() The address of a procedure to compare list elements. The procedure must have two parameters that will receive the memory address of the two elements to compare and return one of the following values:

```

#PB_Sort_Greater: The first element is greater than the
second one.
#PB_Sort_Equal  : Both elements are equal.
#PB_Sort_Lesser : The first element is lesser than the second
one.

```

The compare procedure is called very often during the sorting process so it should do as little work as possible and be fast to execute.

Options (optional) It can be a combination of the following values:

```

#PB_Sort_Ascending : Sort the list in ascending order (lower
values first). This is the default.
#PB_Sort_Descending: Sort the list in descending order
(higher values first)

```

Start, End (optional) The index of the first and last element in the list that should be sorted. If these parameters are not specified, then the whole list is sorted. The first list element is at position 0, the next at 1 and so on.

Return value

None.

Example: Sort structured list

```
1  Structure CustomStruct
2      Value$
3  EndStructure
4
5  ; Sort by numeric value in the string element in the structure
6  Procedure CustomCompare(*a.CustomStruct, *b.CustomStruct)
7      If Val(*a\Value$) < Val(*b\Value$)
8          ProcedureReturn #PB_Sort_Lesser
9      ElseIf Val(*a\Value$) > Val(*b\Value$)
10         ProcedureReturn #PB_Sort_Greater
11     Else
12         ProcedureReturn #PB_Sort_Equal
13     EndIf
14 EndProcedure
15
16 NewList Test.CustomStruct()
17
18 AddElement(Test()) : Test()\Value$ = "0005"
19 AddElement(Test()) : Test()\Value$ = "2"
20 AddElement(Test()) : Test()\Value$ = "42"
21 AddElement(Test()) : Test()\Value$ = "7"
22 AddElement(Test()) : Test()\Value$ = "23"
23 AddElement(Test()) : Test()\Value$ = "100"
24 AddElement(Test()) : Test()\Value$ = "101"
25
26 CustomSortList(Test(), @CustomCompare(), #PB_Sort_Ascending)
27
28 ForEach Test()
29     Debug Test()\Value$
30 Next
```

Example: Sort basic type list

```
1  ; Sort float elements by absolute value (ignoring the sign)
2  Procedure AbsCompare(*a.Float, *b.Float)
3      If Abs(*a\f) < Abs(*b\f)
4          ProcedureReturn #PB_Sort_Lesser
5      ElseIf Abs(*a\f) > Abs(*b\f)
6          ProcedureReturn #PB_Sort_Greater
7      Else
8          ProcedureReturn #PB_Sort_Equal
9      EndIf
10 EndProcedure
11
12 NewList Test.f()
13 AddElement(Test()) : Test() = 2.5
14 AddElement(Test()) : Test() = -3.0
15 AddElement(Test()) : Test() = 0
16 AddElement(Test()) : Test() = 12
17 AddElement(Test()) : Test() = -100
18
```



```

19 CustomSortList(Test(), @AbsCompare())
20
21 ForEach Test()
22     Debug Test()
23 Next

```

See Also

SortList() , SortStructuredList() , RandomizeList()

160.3 SortArray

Syntax

```
SortArray(ArrayName(), Options [, Start, End])
```

Description

Sorts the specified array , according to the given options. The array may be of one of basic type : byte, word, long, integer, string or float. For structured arrays, use SortStructuredArray() . Multi-dimensioned arrays are not supported.

Parameters

ArrayName() The array to sort.

Options It can be a combination of the following values:

```

#PB_Sort_Ascending : Sort the array in ascending order (lower
values first)
#PB_Sort_Descending: Sort the array in descending order
(higher values first)
#PB_Sort_NoCase     : Sort the string array without case
sensitivity (a=A, b=B etc..)

```

Start, End (optional) The index of the first and last element in the array that should be sorted. If these parameters are not specified, then the whole array is sorted.

Return value

None.

Remarks

Fixed strings are not supported by the sort routine. If an array is not fully filled, then null elements will be sorted first in ascending order and last in descending order. NaN numbers are not accepted when sorting as it produces random results.

See Also

SortStructuredArray() , CustomSortArray() , RandomizeArray()

160.4 SortList

Syntax

```
SortList(ListName(), Options [, Start, End])
```

Description

Sorts the specified list , according to the given options. The list may be of one of basic type : byte, word, long, integer, string or float. For structured list, use SortStructuredList() .

Parameters

ListName() The list to sort.

Options It can be a combination of the following values:

```
#PB_Sort_Ascending : Sort the list in ascending order (lower
values first)
#PB_Sort_Descending: Sort the list in descending order
(higher values first)
#PB_Sort_NoCase     : Sort the string list without case
sensitivity (a=A, b=B etc..)
```

Start, End (optional) The index of the first and last element in the list that should be sorted.

If these parameters are not specified, then the whole list is sorted.

The first list element is at position 0, the next at 1 and so on.

Return value

None.

Remarks

Fixed strings are not supported by the sort routine.

See Also

SortStructuredList() , CustomSortList() , RandomizeList()

160.5 SortStructuredArray

Syntax

```
SortStructuredArray(ArrayName(), Options ,
    OffsetOf(Structure\Field), TypeOf(Structure\Field) [, Start,
    End])
```

Description

Sorts the specified structured array , according to the given options. The array must have an associated structure .

Parameters

ArrayName() The array to sort.

Options It can be a combination of the following values:

```
#PB_Sort_Ascending : Sort the array in ascending order (lower
values first)
#PB_Sort_Descending: Sort the array in descending order
(higher values first)
#PB_Sort_NoCase     : Sort the string array without case
sensitivity (a=A, b=B etc...)
```

OffsetOf(Structure\Field) Offset of the field in the structure. OffsetOf() may be used to retrieve the field offset in the structure associated to the array.

TypeOf(Structure\Field) The field type of the field in the structure. It has to match the real structure field type. TypeOf() may be used to automatically retrieve the field type. Available types are:

```
#PB_Byte      : The structure field to sort is a byte (.b)
#PB_Word      : The structure field to sort is a word (.w)
#PB_Long      : The structure field to sort is a long (.l)
#PB_String    : The structure field to sort is a string (.s or
$, fixed strings are not supported)
#PB_Float     : The structure field to sort is a float (.f)
#PB_Double    : The structure field to sort is a double (.d)
#PB_Quad      : The structure field to sort is a quad (.q)
#PB_Character : The structure field to sort is a character (.c)
#PB_Integer   : The structure field to sort is an integer (.i)
#PB_Ascii     : The structure field to sort is an ascii
character (.a)
#PB_Unicode   : The structure field to sort is a unicode
character (.u)
```

Start, End (optional) The index of the first and last element in the array that should be sorted. If these parameters are not specified, then the whole array is sorted.

Remarks

Fixed strings are not supported by the sort routine. If an array is not fully filled, then null elements will be sorted first in ascending order and last in descending order.

NaN numbers are not accepted when sorting as it produces random results.

Example

```
1
2  Structure Animal
3      Name$
4      Speed.l
5  EndStructure
6
7  Dim Animals.Animal(2)
8
9  Animals(0)\Name$ = "Tiger"
10 Animals(0)\Speed = 10
11
12 Animals(1)\Name$ = "Jaguar"
13 Animals(1)\Speed = 40
```

```

14
15     Animals(2)\Name$ = "Zebra"
16     Animals(2)\Speed = 30
17
18     SortStructuredArray(Animals(), 0, OffsetOf(Animal\Name$),
19         TypeOf(Animal\Name$))
20
21     For k=0 To 2
22         Debug Animals(k)\Name$+" - Speed: "+Str(Animals(k)\Speed)
23     Next
24
25     SortStructuredArray(Animals(), 0, OffsetOf(Animal\Speed),
26         TypeOf(Animal\Speed))
27
28     For k=0 To 2
29         Debug Animals(k)\Name$+" - Speed: "+Str(Animals(k)\Speed)
30     Next

```

See Also

SortArray() , CustomSortArray() , RandomizeArray()

160.6 SortStructuredList

Syntax

```
SortStructuredList(ListName(), Options, OffsetOf(Structure\Field),
    TypeOf(Structure\Field) [, Start, End])
```

Description

Sorts the specified structured list , according to the given options. The list must have an associated structure .

Parameters

ListName() The list to sort.

Options It can be a combination of the following values:

```

#PB_Sort_Ascending : Sort the list in ascending order (lower
values first)
#PB_Sort_Descending: Sort the list in descending order
(higher values first)
#PB_Sort_NoCase     : Sort the string list without case
sensitivity (a=A, b=B etc..)

```

OffsetOf(Structure\Field) Offset of the field in the structure. OffsetOf() may be used to retrieve the field offset in the structure associated to the list.

.TypeOf(Structure\Field) The field type of the field in the structure. It has to match the real structure field type. TypeOf() may be used to automatically retrieve the field type. Available types are:

```

#PB_Byte      : The structure field to sort is a byte (.b)
#PB_Word      : The structure field to sort is a word (.w)
#PB_Long      : The structure field to sort is a long (.l)

```

```

#PB_String      : The structure field to sort is a string (.s or
                  $, fixed strings are not supported)
#PB_Float       : The structure field to sort is a float (.f)
#PB_Double      : The structure field to sort is a double (.d)
#PB_Quad        : The structure field to sort is a quad (.q)
#PB_Character    : The structure field to sort is a character (.c)
#PB_Integer     : The structure field to sort is an integer (.i)
#PB_Ascii       : The structure field to sort is an ascii
                  character (.a)
#PB_Unicode     : The structure field to sort is a unicode
                  character (.u)

```

Start, End (optional) The index of the first and last element in the list that should be sorted.
 If these parameters are not specified, then the whole list is sorted.
 The first list element is at position 0, the next at 1 and so on.

Remarks

Fixed strings are not supported by the sort routine.

Example

```

1
2  Structure Animal
3     Name$
4     Speed.l
5  EndStructure
6
7  NewList Animals.Animal()
8
9  AddElement(Animals())
10 Animals()\Name$ = "Tiger"
11 Animals()\Speed = 10
12
13 AddElement(Animals())
14 Animals()\Name$ = "Jaguar"
15 Animals()\Speed = 40
16
17 AddElement(Animals())
18 Animals()\Name$ = "Zebra"
19 Animals()\Speed = 30
20
21 SortStructuredList(Animals(), 0, OffsetOf(Animal\Name$),
22   TypeOf(Animal\Name$))
23
24 ForEach Animals()
25   Debug Animals()\Name$+" - Speed: "+Str(Animals()\Speed)
26 Next
27
28 SortStructuredList(Animals(), 0, OffsetOf(Animal\Speed),
29   TypeOf(Animal\Speed))
30
31 ForEach Animals()
32   Debug Animals()\Name$+" - Speed: "+Str(Animals()\Speed)
33 Next

```

See Also

SortList() , CustomSortList() , RandomizeList()

160.7 RandomizeArray

Syntax

```
RandomizeArray(ArrayName() [, Start, End])
```

Description

Reorders the elements of the given array in a random order.

Parameters

ArrayName() The array to randomize.

Start, End (optional) The index of the first and last element in the array that should be randomized. If these parameters are not specified, then the whole array is randomized.

Return value

None.

Remarks

This function uses the pseudorandom number generator of the Random() function to determine the new order of the array elements. It is therefore dependent on the current RandomSeed() .

See Also

SortArray() , SortStructuredArray() , RandomizeList() , Random() , RandomSeed()

160.8 RandomizeList

Syntax

```
RandomizeList(List() [, Start, End])
```

Description

Reorders the elements of the given list in a random order.

Parameters

List() The list to randomize.

Start, End (optional) The index of the first and last element in the list that should be randomized. If these parameters are not specified, then the whole list is randomized. The first list element is at position 0, the next at 1 and so on.

Return value

None.

Remarks

This function uses the pseudorandom number generator of the `Random()` function to determine the new order of the list elements. It is therefore dependent on the current `RandomSeed()` .

See Also

`SortList()` , `SortStructuredList()` , `RandomizeArray()` , `Random()` , `RandomSeed()`

Chapter 161

Sound

Overview

The PureBasic sound system provides an easy way to have sound inside application or game. It uses special functions to get the maximum speed of available hardware.

161.1 CatchSound

Syntax

```
Result = CatchSound(#Sound, *Buffer [, Size [, Flags]])
```

Description

Load a WAV (in PCM format, ADPCM is not supported) or any other format supported by the SoundPlugin library found at the specified address. The following functions can be used to enable automatically more sound formats:

```
UseFLACSoundDecoder()
```

```
UseOGGSoundDecoder()
```

Parameters

#Sound A number to identify the new sound. #PB_Any can be used to auto-generate this number.

***Buffer** The buffer to use to create the sound. This is a regular memory address.

Size (optional) The buffer size (in bytes). With WAV files, this parameter doesn't need to be specified. With other sound decoders, it has to be specified.

Flags (optional) It can be the following value:

```
#PB_Sound_Streaming: Enable streaming playback (only
supported for FLAC
and OGG
))
```

Return value

Nonzero if the sound has been successfully created, zero otherwise.

Example

```
1  CatchSound(0, ?Music)
2  End
3
4  DataSection
5      Music:
6          IncludeBinary "Sound.wav"
```

Remarks

The "?" is a pointer to a label. More information about pointers and memory access can be found in the relating chapter [here](#) .

See Also

LoadSound() , FreeSound() , PlaySound()

161.2 GetSoundPosition

Syntax

```
Result = GetSoundPosition(#Sound [, Mode [, Channel]])
```

Description

Get the current sound position.

Parameters

#Sound The sound to use.

Mode (optional) The mode used to get the position. It can be one of the following value:

```
#PB_Sound_Frame      : the position is returned in frame
                      (default).
#PB_Sound_Millisecond: the position is returned in
                      milliseconds.
```

Channel (optional) The channel to get the position. It's the value returned by PlaySound() when using the `#PB_Sound_MultiChannel` flag.

Return value

The current sound position or -1 if an error occurred.

Remarks

Sounds loaded with the `#PB_Sound_Streaming` flag are not supported.

Example

```
1  If InitSound()                ; Initialize Sound system
2      UseOGGSoundDecoder()      ; Use ogg files
3
4      ; Loads 2 sounds
5      If LoadSound(0, #PB_Compiler_Home +
6          "Examples/3D/Data/Siren.ogg")
7          If LoadSound(1, #PB_Compiler_Home +
8              "Examples/3D/Data/Roar.ogg")
9              ; The siren is playing
10             PlaySound(0)
11
12             ; Display the position
13             Repeat
14                 Pos = GetSoundPosition(0, #PB_Sound_Millisecond)
15                 Delay(100)          ; Wait 100 ms
16                 Debug Pos          ; Display the position
17                 If Pos > 1000      ; Stop after 1 second
18                     Break
19                 EndIf
20             ForEver
21
22             ; Then 2 sounds are playing together
23             PlaySound(1)
24
25             MessageRequester("Info", "Ok to stop.")
26             End
27         EndIf
28     Else
29         Debug "Warning! The sound environment couldn't be initialized.
30             So no sound commands can be used..."
31     EndIf
```

See Also

SetSoundPosition()

161.3 SetSoundPosition

Syntax

```
SetSoundPosition(#Sound, Position, [, Mode [, Channel]])
```

Description

Set the current sound position.

Parameters

#Sound The sound to use.

Position The new position to set.

Mode (optional) The mode used to set the position. It can be one of the following value:

`#PB_Sound_Frame` : the position is specified in frame (default).
`#PB_Sound_Millisecond`: the position is specified in milliseconds.

Channel (optional) The channel to set the position. It's the value returned by `PlaySound()` when using the `#PB_Sound_MultiChannel` flag.

Return value

None.

Remarks

Sounds loaded with the `#PB_Sound_Streaming` flag are not supported.

Example

```
1  If InitSound()           ; Initialize Sound system
2      UseOGGSoundDecoder() ; Use ogg files
3
4      ; Loads a sound from a file
5      If LoadSound(0, #PB_Compiler_Home +
6          "Examples/3D/Data/Siren.ogg")
7          ; The siren is playing
8          PlaySound(0)
9
10         ; Change the position at 2 seconds
11         SetSoundPosition(0, 2000, #PB_Sound_Millisecond)
12
13         MessageRequester("Info", "Ok to stop.")
14     End
15 EndIf
16 Else
17     Debug "Warning! The sound environment couldn't be initialized.
18     So no sound commands can be used..."
19 EndIf
```

See Also

`GetSoundPosition()`

161.4 FreeSound

Syntax

`FreeSound(#Sound)`

Description

Stops and removes a sound previously loaded with `LoadSound()` or `CatchSound()` from memory. Once a sound has been freed, it can't be played anymore.

Parameters

#Sound The sound to free. If **#PB_All** is specified, all the remaining sounds are freed.

Return value

None.

Remarks

All remaining sounds are automatically freed when the program ends.

Example

```
1  If InitSound()           ; Initialize Sound system
2      UseOGGSoundDecoder() ; Use ogg files
3
4      ; Loads a sound from a file
5      If LoadSound(0, #PB_Compiler_Home +
6          "Examples/3D/Data/Siren.ogg")
7          ; The sound is playing
8          PlaySound(0, #PB_Sound_Loop)
9
10         MessageRequester("Info", "Ok to stop.")
11
12         FreeSound(0) ; The sound is freed
13     End
14 EndIf
15 Else
16     Debug "Warning! The sound environment couldn't be initialized.
17         So no sound commands can be used..."
18 EndIf
```

161.5 InitSound

Syntax

```
Result = InitSound([NbMaxChannels])
```

Description

Initializes the sound environment. This function must be always called before any other sound function and should always check its result. If the sound environment fails, it's absolutely necessary to disable all the sound functions calls.

Parameters

NbMaxChannels (optional) Sets the maximum number of channels to use simultaneously to play sounds. It can range from 1 to 254 (the default value if this parameter is not specified is 64).

Return value

Nonzero if the sound environment has been setup correctly, zero otherwise (for example if no sound card is available).

Example

```
1  If InitSound()           ; Initialize Sound system
2  UseOGGSoundDecoder()     ; Use ogg files
3
4  ; Loads a sound from a file
5  If LoadSound(0, #PB_Compiler_Home + "Examples/3D/Data/Siren.ogg")
6  PlaySound(0, #PB_Sound_Loop) ; Start playing the sound in a
   loop
7
8  MessageRequester("Info", "Ok to stop.")
9
10 FreeSound(0) ; The sound is freed
11 EndIf
12 End
13 Else
14 Debug "Warning! The sound environment couldn't be initialized.
   So no sound commands can be used..."
15 EndIf
```

161.6 IsSound

Syntax

Result = IsSound(#Sound)

Description

Tests if the specified number is a valid and correctly initialized sound.

Parameters

#Sound The sound to use.

Return value

Nonzero if the specified number is a valid sound, zero otherwise.

Remarks

This function is bulletproof and can be used with any value. This is the correct way to ensure a sound is ready to use.

Example

```
1  If IsSound(0) = 0
2  MessageRequester("Info", "The sound is not valid.")
3  EndIf
```

See Also

FreeSound()

161.7 LoadSound

Syntax

```
Result = LoadSound(#Sound, Filename$ [, Flags])
```

Description

Load a WAV (in PCM format, ADPCM is not supported) or any other format supported by the SoundPlugin library into memory. The following functions can be used to enable automatically more sound format:

```
UseFLACSoundDecoder()
```

```
UseOGGSoundDecoder()
```

Parameters

#Sound A number to identify the new sound. **#PB_Any** can be used to auto-generate this number.

Filename\$ The filename to use to load the sound.

Flags (optional) It can be the following value:

```
    #PB_Sound_Streaming: Enable sound streaming playback (only
                        supported for FLAC
    and OGG
))
```

Return value

Nonzero if the sound has been successfully loaded, zero otherwise.

Example

```
1  If InitSound()           ; Initialize Sound system
2      UseOGGSoundDecoder() ; Use ogg files
3
4      ; Loads a sound from a file
5      If LoadSound(0, #PB_Compiler_Home + "Examples/3D/Data/Siren.ogg")
6          PlaySound(0, #PB_Sound_Loop) ; Start playing the sound in a
          loop
7
8          MessageRequester("Info", "Ok to stop.")
9
10         FreeSound(0) ; The sound is freed
11     EndIf
12     End
13 Else
14     Debug "Warning! The sound environment couldn't be initialized.
        So no sound commands can be used..."
15 EndIf
```

See Also

CatchSound() , FreeSound() , PlaySound()

161.8 PauseSound

Syntax

```
PauseSound(#Sound [, Channel])
```

Description

Pause the sound.

Parameters

#Sound The sound to use. If **#PB_All** is specified, all the sounds (and all channels) are paused.

Channel (optional) The channel to use. It's the value returned by `PlaySound()` when using the **#PB_Sound_MultiChannel** flag. If **#PB_All** is specified, all the channels of the sound are paused.

Return value

None.

Remarks

Sounds loaded with the **#PB_Sound_Streaming** flag are not supported.

Example

```
1  If InitSound()           ; Initialize Sound system
2      UseOGGSoundDecoder() ; Use ogg files
3
4      ; Loads a sound from a file
5      If LoadSound(0, #PB_Compiler_Home + "Examples/3D/Data/Siren.ogg")
6          ; The sound is playing
7          PlaySound(0, #PB_Sound_Loop)
8
9          MessageRequester("Info", "Ok to pause.")
10         PauseSound(0) ; Pause
11
12         MessageRequester("Info", "Ok to resume.")
13         ResumeSound(0) ; Resume
14
15         MessageRequester("Info", "Ok to stop.")
16
17         FreeSound(0) ; The sound is freed
18     End
19 EndIf
20 Else
21     Debug "Warning! The sound environment couldn't be initialized.
22         So no sound commands can be used..."
23 EndIf
```

See Also

`LoadSound()` , `ResumeSound()`

161.9 ResumeSound

Syntax

```
ResumeSound(#Sound [, Channel])
```

Description

Resume the sound playing.

Parameters

#Sound The sound to use. If **#PB_All** is specified, all the sounds (and all channels) are resumed.

Channel (optional) The channel to use. It's the value returned by `PlaySound()` when using the **#PB_Sound_MultiChannel** flag. If **#PB_All** is specified, all the channels of the sound are resumed.

Return value

None.

Remarks

Sounds loaded with the **#PB_Sound_Streaming** flag are not supported.

Example

```
1  If InitSound()           ; Initialize Sound system
2      UseOGGSoundDecoder() ; Use ogg files
3
4      ; Loads a sound from a file
5      If LoadSound(0, #PB_Compiler_Home + "Examples/3D/Data/Siren.ogg")
6          ; The sound is playing
7          PlaySound(0, #PB_Sound_Loop)
8
9          MessageRequester("Info", "Ok to pause.")
10         PauseSound(0) ; Pause
11
12         MessageRequester("Info", "Ok to resume.")
13         ResumeSound(0) ; Resume
14
15         MessageRequester("Info", "Ok to stop.")
16
17         FreeSound(0) ; The sound is freed
18     End
19 EndIf
20 Else
21     Debug "Warning! The sound environment couldn't be initialized.
22     So no sound commands can be used..."
23 EndIf
```

See Also

`LoadSound()` , `PauseSound()`

161.10 PlaySound

Syntax

`Result = PlaySound(#Sound [, Flags [, Volume]])`

Description

Start to play the specified sound.

Parameters

#Sound The sound to play.

Flags (optional) It can be a combination of the following values:

```
0                : no flag
#PB_Sound_Loop   : play the sound continuously (starts
                  again when end is reached)
#PB_Sound_MultiChannel: play the sound in a new channel
                  instead of stopping the
                  previously played sound. This allows to use the same
                  sound and to play it on different
                  channels at once. 'Result' will be new allocated
                  channel, and can be used by the
                  other sound commands like SoundVolume()
, SoundPan()
etc.
```

Volume (optional) Sets the initial volume of the #Sound. The valid values are from 0 (no volume) to 100 (full volume). The default value is 100.

Return value

The channel number, if the #PB_Sound_MultiChannel flag is used.

Example

```
1  If InitSound()           ; Initialize Sound system
2      UseOGGSoundDecoder() ; Use ogg files
3
4      ; Loads a sound from a file
5      If LoadSound(0, #PB_Compiler_Home + "Examples/3D/Data/Siren.ogg")
6          ; The sound is playing
7          PlaySound(0, #PB_Sound_Loop, 20)
8
9          MessageRequester("Info", "Ok to stop.")
10
11         FreeSound(0) ; The sound is freed
12     EndIf
13     End
14 Else
15     Debug "Warning! The sound environment couldn't be initialized.
16         So no sound commands can be used..."
17 EndIf
```

See Also

StopSound() , FreeSound() , PauseSound() , ResumeSound()

161.11 GetSoundFrequency

Syntax

```
Result = GetSoundFrequency(#Sound [, Channel])
```

Description

Get the current frequency of the sound.

Parameters

#Sound The sound to use.

Channel (optional) The channel to use. It's the value returned by PlaySound() when using the **#PB_Sound_MultiChannel** flag.

Return value

Returns the current frequency (in Hz) of the sound.

Example

```
1  If InitSound()           ; Initialize Sound system
2      UseOGGSoundDecoder() ; Use ogg files
3
4      ; Loads a sound from a file
5      If LoadSound(0, #PB_Compiler_Home + "Examples/3D/Data/Siren.ogg")
6          ; The sound is playing
7          PlaySound(0, #PB_Sound_Loop, 20)
8
9          MessageRequester("Info", "The average frequency is " +
10             Str(GetSoundFrequency(0)) + " Hz")
11
12             MessageRequester("Info", "Ok to stop.")
13
14             FreeSound(0) ; The sound is freed
15         End
16     EndIf
17 Else
18     Debug "Warning! The sound environment couldn't be initialized.
19         So no sound commands can be used..."
20 EndIf
```

See Also

SetSoundFrequency()

Supported OS

Windows

161.12 SetSoundFrequency

Syntax

```
SetSoundFrequency(#Sound, Frequency [, Channel])
```

Description

Set the new frequency, in real-time, for the sound. The new frequency value is saved for the sound, so it's not needed to call it every time.

Parameters

#Sound The sound to use.

Frequency The sound new frequency (in Hz) to set. Valid values are from 1000 Hz to 100000 Hz.

Channel (optional) The channel to use. It's the value returned by PlaySound() when using the **#PB_Sound_MultiChannel** flag. If **#PB_All** is specified, all the channels of the sound are affected.

Return value

None.

Example

```
1  If InitSound()           ; Initialize Sound system
2      UseOGGSoundDecoder() ; Use ogg files
3
4      ; Loads a sound from a file
5      If LoadSound(0, #PB_Compiler_Home + "Examples/3D/Data/Siren.ogg")
6          ; The sound is playing
7          PlaySound(0, #PB_Sound_Loop, 20)
8
9          MessageRequester("Info", "The average frequency is " +
10             Str(GetSoundFrequency(0)) + " Hz")
11             PauseSound(0)
12
13             SetSoundFrequency(0, 16000)
14
15             ResumeSound(0)
16
17             MessageRequester("Info", "The average frequency is " +
18                 Str(GetSoundFrequency(0)) + " Hz")
19
20             FreeSound(0) ; The sound is freed
21         End
22     EndIf
23 Else
24     Debug "Warning! The sound environment couldn't be initialized.
25         So no sound commands can be used..."
26 EndIf
```

See Also

GetSoundFrequency()

Supported OS

Windows

161.13 SoundStatus

Syntax

```
Result = SoundStatus(#Sound [, Channel])
```

Description

Get the current sound status.

Parameters

#Sound The sound to use.

Channel (optional) The channel to use. It's the value returned by PlaySound() when using the #PB_Sound_MultiChannel flag.

Return value

The current sound status. It can be one of the following value:

```
#PB_Sound_Stopped: the sound is stopped.  
#PB_Sound_Playing: the sound is playing.  
#PB_Sound_Paused : the sound is paused.  
#PB_Sound_Unknown: the sound is in an unknown state (an error  
occurred when getting the state).
```

Example

```
1  Procedure SelectStatus(Status)  
2      Select Status  
3          Case #PB_Sound_Stopped  
4              MessageRequester("Info", "The sound is stopped.")  
5  
6          Case #PB_Sound_Playing  
7              MessageRequester("Info", "The sound is playing.")  
8  
9          Case #PB_Sound_Paused  
10             MessageRequester("Info", "The sound is paused.")  
11  
12             Case #PB_Sound_Unknown  
13                 MessageRequester("Info", "Status unknown.")  
14  
15             Default  
16                 MessageRequester("Info", "Status unknown.")  
17  
18         EndSelect  
19     EndProcedure  
20  
21     If InitSound() ; Initialize Sound system  
22         UseOGGSoundDecoder() ; Use ogg files  
23  
24         ; Loads a sound from a file
```

```

25     If LoadSound(0, #PB_Compiler_Home + "Examples/3D/Data/Siren.ogg")
26         ; The sound is playing
27         PlaySound(0, #PB_Sound_Loop, 20)
28         SelectStatus(SoundStatus(0))
29
30         PauseSound(0)
31         SelectStatus(SoundStatus(0))
32
33         ResumeSound(0)
34         SelectStatus(SoundStatus(0))
35
36         StopSound(0)
37         SelectStatus(SoundStatus(0))
38
39         FreeSound(0) ; The sound is freed
40     End
41 EndIf
42 Else
43     Debug "Warning! The sound environment couldn't be initialized.
44         So no sound commands can be used..."
45 EndIf

```

161.14 SoundPan

Syntax

```
SoundPan(#Sound, Pan [, Channel])
```

Description

Sets the new pan value, in real-time, for the #Sound. The pan value is saved for the #Sound, so it's not needed to call it every time. The panning is a way to play a sound on a stereo equipment.

Parameters

#Sound The sound to use.

Pan The new pan value. Valid values are from -100 (full left) to 100 (full right). If the pan value is zero, then the sound is played on right and left speaker, equally.

Channel (optional) The channel to use. It's the value returned by PlaySound() when using the #PB_Sound_MultiChannel flag.

Return value

None.

Example

```

1  If InitSound()           ; Initialize Sound system
2      UseOGGSoundDecoder() ; Use ogg files
3
4      ; Loads a sound from a file
5      If LoadSound(0, #PB_Compiler_Home + "Examples/3D/Data/Siren.ogg")
6          ; The sound is playing
7          PlaySound(0, #PB_Sound_Loop, 20)

```

```

8
9     MessageRequester("Info", "The sound is playing in stereo.")
10
11     SoundPan(0, -100)
12     MessageRequester("Info", "The sound is playing only on the
left channel.")
13
14     SoundPan(0, 100)
15     MessageRequester("Info", "The sound is playing only on the
right channel.")
16
17     SoundPan(0, 0)
18     MessageRequester("Info", "The sound is playing in stereo.")
19
20     FreeSound(0) ; The sound is freed
21     End
22 EndIf
23 Else
24     Debug "Warning! The sound environment couldn't be initialized.
So no sound commands can be used..."
25 EndIf

```

Supported OS

Windows, MacOS X

161.15 SoundLength

Syntax

```
SoundLength(#Sound [, Mode])
```

Description

Get the length of the sound.

Parameters

#Sound The sound to use.

Mode (optional) The mode used to get the length. It can be one of the following value:

```

#PB_Sound_Frame      : the length is returned in frame
                      (default).
#PB_Sound_Millisecond: the length is returned in milliseconds.

```

Return value

The length of the sound, or -1 if an error occurred.

Remarks

Sounds loaded with the `#PB_Sound_Streaming` flag are not supported.

Example

```
1  If InitSound()           ; Initialize Sound system
2      UseOGGSoundDecoder() ; Use ogg files
3
4      ; Loads a sound from a file
5      If LoadSound(0, #PB_Compiler_Home + "Examples/3D/Data/Siren.ogg")
6          ; The sound is playing
7          PlaySound(0, #PB_Sound_Loop, 20)
8
9          MessageRequester("Info", "The length of the sound is "+
10             Str(SoundLength(0)) + " frames.")
11
12          MessageRequester("Info", "The length of the sound is "+
13             Str(SoundLength(0, #PB_Sound_Millisecond)) + " ms.")
14
15          FreeSound(0) ; The sound is freed
16      End
17  EndIf
18 Else
19      Debug "Warning! The sound environment couldn't be initialized.
20      So no sound commands can be used..."
21  EndIf
```

161.16 SoundVolume

Syntax

`SoundVolume(#Sound, Volume.f [, Channel])`

Description

Change the sound volume, in real-time.

Parameters

#Sound The sound to use. If **#PB_All** is specified, all the sounds (and all channels) are affected.

Volume.f The new volume for the sound. Valid values are from 0 (no volume) to 100 (full volume).

Channel (optional) The channel to use. It's the value returned by `PlaySound()` when using the **#PB_Sound_MultiChannel** flag. If **#PB_All** is specified, all the channels of the sound are affected.

Return value

None.

Example

```
1  If InitSound()           ; Initialize Sound system
2      UseOGGSoundDecoder() ; Use ogg files
3
4      ; Loads a sound from a file
5      If LoadSound(0, #PB_Compiler_Home + "Examples/3D/Data/Siren.ogg")
```

```

6      ; The sound is playing
7      PlaySound(0, #PB_Sound_Loop, 20)
8
9      MessageRequester("Info", "The sound volume is at 20%")
10
11     SoundVolume(0, 80)
12     MessageRequester("Info", "The sound volume is at 80%")
13
14     FreeSound(0) ; The sound is freed
15     End
16 EndIf
17 Else
18     Debug "Warning! The sound environment couldn't be initialized.
19     So no sound commands can be used..."
EndIf

```

See Also

LoadSound()

161.17 StopSound

Syntax

```
StopSound(#Sound [, Channel])
```

Description

Stops the specified sound (if it was playing).

Parameters

#Sound The sound to stop. If this value is set to `#PB_All`, then all sounds currently playing are stopped.

Channel (optional) The channel to use. It's the value returned by `PlaySound()` when using the `#PB_Sound_MultiChannel` flag.

Return value

None.

Example

```

1  If InitSound()           ; Initialize Sound system
2      UseOGGSoundDecoder() ; Use ogg files
3
4      ; Loads a sound from a file
5      If LoadSound(0, #PB_Compiler_Home + "Examples/3D/Data/Siren.ogg")
6          ; The sound is playing
7          PlaySound(0, #PB_Sound_Loop, 20)
8
9          MessageRequester("Info", "Ok to stop.")
10
11         StopSound(0)

```



```
12     MessageRequester("Info", "Sound stopped")
13
14     FreeSound(0) ; The sound is freed
15     End
16 EndIf
17 Else
18     Debug "Warning! The sound environment couldn't be initialized.
19     So no sound commands can be used..."
EndIf
```

See Also

PlaySound()

Chapter 162

Sound3D

Overview

The Sound3D library allows to handle sounds in 3D space. For example, it can handle automatic fading depending on the distance. It is based on the regular sound library syntax and behavior. This library uses the 3D engine, so `InitEngine3D()` has to be called successfully before using these functions.

162.1 FreeSound3D

Syntax

```
FreeSound3D (#Sound3D)
```

Description

Stop and remove a 3D sound previously loaded with `LoadSound3D()` from memory. Once a sound has been freed, it can't be played anymore.

Parameters

#Sound3D The 3D sound to free. If **#PB_All** is specified, all the remaining 3D sounds are freed.

Return value

None.

Remarks

All remaining sounds are automatically freed when the program ends.

162.2 IsSound3D

Syntax

```
Result = IsSound3D (#Sound3D)
```

Description

Tests if the given sound is valid and correctly initialized.

Parameters

#Sound3D The sound to test.

Return value

Nonzero if the sound is valid, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a sound is ready to use.

162.3 LoadSound3D

Syntax

```
Result = LoadSound3D(#Sound3D, Filename$ [, Flags])
```

Description

Loads a mono WAV or OGG sound file. The sound has to be mono, as stereo sounds don't allow spacial positioning.

Parameters

#Sound3D A number to identify the new sound. **#PB_Any** can be used to auto-generate this number.

Filename\$ The sound filename to load.

Flags (optional) It can be the following value:

```
#PB_Sound3D_Streaming: Enable sound streaming playback
```

Return value

Nonzero if the sound has been successfully loaded, zero otherwise.

Remarks

A sound doesn't hold its position. It can be attached to a node object to have its own position. A sound is played relative to the listener location. To change the listener location, use `SoundListenerLocate()`.

162.4 PlaySound3D

Syntax

```
PlaySound3D(#Sound3D [, Flags])
```

Description

Starts to play the specified sound.

Parameters

#Sound3D The sound to play.

Flags (optional) It can be the following value:

```
#PB_Sound3D_Loop: play the sound continuously (starts again
when end is reached)
```

Return value

None.

See Also

StopSound()

162.5 SoundVolume3D

Syntax

```
SoundVolume3D(#Sound3D, Volume)
```

Description

Set the new volume, in real-time, for the #Sound3D. The volume value is saved for the #Sound3D, so it's not needed to call it every time.

Parameters

#Sound3D The sound to use.

Volume The new volume for the sound. The valid values are from 0 (no volume) to 100 (full volume).

Return value

None.

162.6 StopSound3D

Syntax

```
StopSound3D(#Sound3D)
```

Description

Stops the specified sound (if it was playing).

Parameters

#Sound3D The sound to stop. If this value is set to #PB_All, then all sounds currently playing are stopped.

Return value

None.

See Also

PlaySound()

162.7 SoundID3D

Syntax

```
SoundID3D = SoundID3D(#Sound3D)
```

Description

Returns the unique system identifier of the sound.

Parameters

#Sound3D The sound to use.

Return value

Returns the unique system identifier of the sound.

162.8 SoundRange3D

Syntax

```
SoundRange3D(#Sound3D, Minimum, Maximum)
```

Description

Set the range, in world units, for the sound emission.

Parameters

#Sound3D The sound to use.

Minimum The distance from the sound location when the sound volume starts fading.

Maximum The distance from the sound location when the sound can still be heard. Above this distance, the sound will stop playing. Between the minimum and maximum value, the sound will fade according to the current listener position. To change the listener location, use `SoundListenerLocate()` .

Return value

None.

See Also

SoundListenerLocate()

162.9 SoundCone3D

Syntax

```
SoundCone3D(#Sound3D, InnerCone.f, OuterCone.f, OuterConeVolume)
```

Description

Set the cone angle, to create a directional sound.

Parameters

#Sound3D The sound to use.

InnerCone, OuterCone Specifies the cone angle, in degrees (values are from 0 to 360).

OuterConeVolume The sound volume outside of the cone (values are from 0 to 100).

Return value

None.

162.10 SoundListenerLocate

Syntax

```
SoundListenerLocate(x, y, z)
```

Description

Changes the absolute sound listener (the ear) location in the world.

Parameters

x, y, z The new sound listener absolute position, in world units.

Return value

None.

See Also

SoundRange3D()

Chapter 163

SoundPlugin

Overview

PureBasic supports external sound formats through a dynamic native plug-in system. This means that only the needed encoder or decoder is added to the final executable, which decreases the final program size substantially. For example, if only the OGG decoder is needed, only the code dealing with the OGG decoder will be used. Another nice feature is the automatic sound format detection, whenever several different decoders are used. The following functions support the Sound plugins: `LoadSound()` , and `CatchSound()` .

163.1 UseFLACSoundDecoder

Syntax

```
UseFLACSoundDecoder()
```

Description

Enables the FLAC (Free Lossless Audio Codec) sound support for `CatchSound()` and `LoadSound()` . Sound streaming is supported for this plug-in.

Parameters

None.

Return value

None.

Remarks

Since FLAC is a non-destructive compression codec, the file size will remain quite large, especially compared to OGG for example. This allows for a reduction in file size, while still retaining all the original sound information.

For more information: [Wikipedia - FLAC](#).

163.2 UseOGGSoundDecoder

Syntax

`UseOGGSoundDecoder()`

Description

Enables the OGG (OGG Vorbis) sound support for `CatchSound()` and `LoadSound()` . Sound streaming is supported for this plug-in.

Parameters

None.

Return value

None.

Remarks

The OGG format uses destructive compression (lossy) which entails the loss of some of the original information when it is compressed. This is actually one of the best lossy compression algorithms available, which can dramatically decrease the sound file size without the introduction of unnecessary audio artifacts.

For more information: [Wikipedia - OGG](#).

Chapter 164

SpecialEffect

Overview

The special effect library allows to apply easily several real-time effects in the 3D world like post processing and ribbon trails.

164.1 CreateCompositorEffect

Syntax

```
Result = CreateCompositorEffect(#Effect, CameraID, EffectName$)
```

Description

Create a new compositor effect. Once created, the effect is immediately applied to the rendering. It is possible to hide the effect with HideEffect() .

Parameters

#Effect The number to identify the new effect. #PB_Any can be used to auto-generate this number.

CameraID The camera to apply the effect. This effect will only affect this camera. To get a valid 'CameraID', use CameraID() .

EffectName\$ Name of effect, as described in the OGRE compositor file (usually a '.compositor' file).

Return value

Returns nonzero if the effect was created successfully and zero if there was an error. If #PB_Any was used as the #Effect parameter then the new generated number is returned on success.

See Also

FreeEffect() , HideEffect()

164.2 CreateRibbonEffect

Syntax

```
Result = CreateRibbonEffect(#Effect, MaterialID, NbChains,
                             NbElements, Length)
```

Description

Create a new ribbon trail effect. Once created, the effect has to be attached with `AttachRibbonEffect()` . It is possible to hide the effect with `HideEffect()` .

Parameters

#Effect The number to identify the new effect. `#PB_Any` can be used to auto-generate this number.

MaterialID The material to apply on the ribbon trail. To get a valid 'MaterialID', use `MaterialID()` .

NbChains Number of chains of the ribbon trail. The more chains, the more precise the ribbon will be.

NbElements Number of elements per chains. The more elements, the more precise the ribbon will be.

Length Maximum length, in world unit, of the ribbon. Once the length is reached the ribbon will be completely fade out.

Return value

Returns nonzero if the effect was created successfully and zero if there was an error. If `#PB_Any` was used as the `#Effect` parameter then the new generated number is returned on success.

See Also

`FreeEffect()` , `HideEffect()`

164.3 RibbonEffectWidth

Syntax

```
RibbonEffectWidth(#Effect, ChainIndex, Width, FadeoutWidth)
```

Description

Changed the ribbon chain width.

Parameters

#Effect The effect to use. This effect has to be created with `CreateRibbonEffect()` .

ChainIndex The index of the chain to change the width. The first index starts from zero. This index number must be lower than the number of chains created with `CreateRibbonEffect()` .

Width The new width of the chain, in world unit.

FadeoutWidth The fadeout width per second. Every second, the width of the ribbon chain will be decreased by this value up to be zero.

Return value

None.

See Also

CreateRibbonEffect()

164.4 AttachRibbonEffect

Syntax

```
AttachRibbonEffect(#Effect, NodeID)
```

Description

Attach the ribbon to the given node.

Parameters

#Effect The effect to use. This effect has to be created with CreateRibbonEffect() .

NodeID The node to attach the ribbon. A single ribbon can be attached to several nodes. To get a valid node ID, use NodeID() .

Return value

None.

See Also

CreateRibbonEffect() , DetachRibbonEffect()

164.5 DetachRibbonEffect

Syntax

```
DetachRibbonEffect(#Effect, NodeID)
```

Description

Detach the ribbon from the given node.

Parameters

#Effect The effect to use. This effect has to be created with CreateRibbonEffect() .

NodeID The node to detach the ribbon. If the ribbon was not attached to this node, the function has no effect. To get a valid node ID, use NodeID() .

Return value

None.

See Also

CreateRibbonEffect() , AttachRibbonEffect()

164.6 CreateLensFlareEffect

Syntax

```
CreateLensFlareEffect(#Effect, CameraID, NodeID, BurstSize,  
                      HaloSize, HaloMaterialID, CircleMaterialID, BurstMaterialID)
```

Description

Create a new lensflare effect for the given camera. A lensflare is always attached to a node, and will be displayed automatically depending of the node position relative to the camera view.

Parameters

#Effect The number to identify the new effect. **#PB_Any** can be used to auto-generate this number.

CameraID The camera to apply the lensflare. It will only affect this camera. To get a valid 'CameraID', use CameraID() .

NodeID The parent node to handle the lensflare. To get a valid 'NodeID', use NodeID() .

BurstSize The size of the burst, in world unit.

HaloSize The size of the halo, in world unit.

HaloMaterialID The halo material. To get a valid material ID, use MaterialID() .

CircleMaterialID The circle material. To get a valid material ID, use MaterialID() .

BurstMaterialID The burst material. To get a valid material ID, use MaterialID() .

Return value

Returns nonzero if the lensflare effect was created successfully and zero if there was an error. If **#PB_Any** was used as the **#Effect** parameter then the new generated number is returned on success.

See Also

FreeEffect() , LensFlareEffectColor()

164.7 LensFlareEffectColor

Syntax

```
LensFlareEffectColor(#Effect, ColorType, Color)
```

Description

Changes the color of the specified lens flare effect part.

Parameters

#Effect The effect to use. This effect has to be created with `CreateLensFlareEffect()` .

ColorType This can be one of the following value:

```
#PB_LensFlare_HaloColor    : changes the halo color
#PB_LensFlare_CircleColor : changes the circle color
#PB_LensFlare_BurstColor  : changes the burst color
```

Color The new color for the selected lensflare part. A valid color can be get with `RGB()` .

Return value

None.

See Also

`CreateLensFlareEffect()`

164.8 FreeEffect

Syntax

```
FreeEffect(#Effect)
```

Description

Free the specified effect.

Parameters

#Effect The effect to free. If **#PB_All** is specified, all the remaining effects are freed.

Return value

None.

Remarks

Once the effect is freed, it may not be used anymore.
All remaining effects are automatically freed when the program ends.

See Also

`CreateRibbonEffect()` , `CreateCompositorEffect()`

164.9 IsEffect

Syntax

```
Result = IsEffect(#Effect)
```

Description

Tests if the given effect number is a valid and correctly initialized effect.

Parameters

#Effect The effect to use.

Return value

Returns nonzero if #Effect is a valid effect, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a effect is ready to use.

See Also

CreateRibbonEffect() , CreateCompositorEffect()

164.10 HideEffect

Syntax

```
HideEffect(#Effect, State)
```

Description

Hide or show the specified effect.

Parameters

#Effect The effect to use.

State The new state of the effect. If State = 1, the effect will be hidden, if State = 0 it will be shown.

Return value

None.

See Also

CreateRibbonEffect() , CreateCompositorEffect()

164.11 CompositorEffectParameter

Syntax

```
CompositorEffectParameter(#Effect, TechniqueID, PassID,  
    EffectName$, DataType, *Data)
```

Description

Set real-time parameter on the specified effect.

Parameters

#Effect The effect to use. The effect has to be a compositor effect created with `CreateCompositorEffect()` .

TechniqueID The TechniqueID of the effect.

PassID The PassID of the effect.

EffectName\$ The name of the effect.

DataType The datatype of the parameter to set.

Data The data of the parameter to set. It depends of the specified datatype.

Return value

None.

See Also

`CreateCompositorEffect()`

164.12 RibbonEffectColor

Syntax

```
RibbonEffectColor(#Effect, ChainIndex, Color, FadeoutColor)
```

Description

Set the colors for the ribbon trail.

Parameters

#Effect The effect to use. It has to be a ribbon trail create with `CreateRibbonEffect()` .

ChainIndex The index of the chain to change the color. The first index starts from zero. This index number must be lower than the number of chains created with `CreateRibbonEffect()` .

Color The color of the trail. This color can be in RGB or RGBA format.

FadeoutColor The fade-out color of the trail, the destination color when the ribbon trail fade. This color can be in RGB or RGBA format.

Return value

None.

See Also

`CreateRibbonEffect()`

Chapter 165

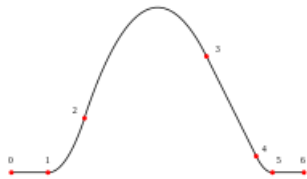
Spline

Overview

A spline contains any number of points which are smoothly interpolated. The time to go from one point to another is always the same, independent from the distance between these points. A spline doesn't exist physically in the 3D world, it is a virtual object which can be used for different purposes, like path-finding, smooth node moving (be sure to check NodeAnimation library for this as well) and more.

InitEngine3D() should be called successfully before using the spline functions.

See also the Wikipedia article about [splines](#).



165.1 CreateSpline

Syntax

```
Result = CreateSpline(#Spline)
```

Description

Creates a new spline. A spline doesn't exist physically in the 3D world, it is a virtual object which can be used for different purposes, like path-finding, smooth node moving (be sure to check NodeAnimation library for this as well) and more. To calculate the position of an intermediate point, see ComputeSpline() .

Parameters

#Spline The number to identify the new spline. #PB_Any can be used to auto-generate this number.

Return value

Returns zero if the spline can't be created. If #PB_Any is used as '#Spline' parameter, the new spline number is returned.

See Also

FreeSpline() , ComputeSpline()

165.2 FreeSpline

Syntax

```
FreeSpline(#Spline)
```

Description

Frees a spline and releases all its associated memory. This spline must not be used (by using its number with the other functions in this library) after calling this function, unless you create it again.

Parameters

#Spline The spline to free. If **#PB_All** is specified, all the remaining splines are freed.

Return value

None.

Remarks

All remaining splines are automatically freed when the program ends.

See Also

CreateSpline()

165.3 AddSplinePoint

Syntax

```
AddSplinePoint(#Spline, x, y, z)
```

Description

Add a new point to the spline. The time to go from one point to another is always the same, independent from the distance between these points.

Parameters

#Spline The spline to use.

x, y, z The new point position in the world.

Return value

None.

See Also

CreateSpline() , ComputeSpline()

165.4 ClearSpline

Syntax

```
ClearSpline(#Spline)
```

Description

Clear the spline. All points will be removed from the spline.

Parameters

#Spline The spline to clear.

Return value

None.

See Also

CreateSpline() , AddSplinePoint()

165.5 CountSplinePoints

Syntax

```
Result = CountSplinePoints(#Spline)
```

Description

Returns the number of points in the spline.

Parameters

#Spline The spline to use.

Return value

Returns the number of points in the spline.

See Also

CreateSpline() , AddSplinePoint()

165.6 SplinePointX

Syntax

```
Result = SplinePointX(#Spline, PointIndex)
```

Description

Returns the spline point 'x' position.

Parameters

#Spline The spline to use.

PointIndex The point index. The first point index is zero. The point index has to be below than the result of CountSplinePoints() .

Return value

Returns the spline point 'x' position.

See Also

CreateSpline() , AddSplinePoint() , SplinePointY() , SplinePointZ()

165.7 SplinePointY

Syntax

```
Result = SplinePointY(#Spline, PointIndex)
```

Description

Returns the spline point 'y' position.

Parameters

#Spline The spline to use.

PointIndex The point index. The first point index is zero. The point index has to be below than the result of CountSplinePoints() .

Return value

Returns the spline point 'y' position.

See Also

CreateSpline() , AddSplinePoint() , SplinePointX() , SplinePointZ()

165.8 SplinePointZ

Syntax

```
Result = SplinePointZ(#Spline, PointIndex)
```

Description

Returns the spline point 'z' position.

Parameters

#Spline The spline to use.

PointIndex The point index. The first point index is zero. The point index has to be below than the result of CountSplinePoints() .

Return value

Returns the spline point 'z' position.

See Also

CreateSpline() , AddSplinePoint() , SplinePointX() , SplinePointY()

165.9 UpdateSplinePoint

Syntax

```
UpdateSplinePoint(#Spline, PointIndex, x, y, z)
```

Description

Update the specified point in the spline. The time to go from one point to another is always the same, independent from the distance between these points.

Parameters

#Spline The spline to use.

PointIndex The point index. The first point index is zero. The point index has to be below than the result of CountSplinePoints() .

x, y, z The new point position in the world.

Return value

None.

See Also

CreateSpline() , ComputeSpline()

165.10 ComputeSpline

Syntax

```
ComputeSpline(#Spline, Offset)
```

Description

Computes the spline point position at the specified offset. Once the point has been computed, its position is available with SplineX() , SplineY() and SplineZ() .

Parameters

#Spline The spline to use.

Offset The offset in the spline. The offset value ranges from 0.0 (start of the spline) to 1.0 (end of the spline).

Return value

None.

See Also

CreateSpline() , SplineX() , SplineY() and SplineZ()

165.11 SplineX

Syntax

```
Result = SplineX(#Spline)
```

Description

Returns the spline's 'x' position in the world, after a ComputeSpline() .

Parameters

#Spline The spline to use.

Return value

Returns the spline point's 'x' position in the world, after a ComputeSpline() .

See Also

ComputeSpline() , SplineY() , SplineZ()

165.12 SplineY

Syntax

```
Result = SplineY(#Spline)
```

Description

Returns the spline's 'y' position in the world, after a ComputeSpline() .

Parameters

#Spline The spline to use.

Return value

Returns the spline point's 'y' position in the world, after a `ComputeSpline()` .

See Also

`ComputeSpline()` , `SplineX()` , `SplineZ()`

165.13 SplineZ

Syntax

```
Result = SplineZ(#Spline)
```

Description

Returns the spline's 'z' position in the world, after a `ComputeSpline()` .

Parameters

#Spline The spline to use.

Return value

Returns the spline point's 'z' position in the world, after a `ComputeSpline()` .

See Also

`ComputeSpline()` , `SplineX()` , `SplineY()`

Chapter 166

Sprite

Overview

'Sprites' are well known from game players. These are small pictures, sometimes called 'brushes', which can be displayed at any position on a screen. The sprites can move over graphics using a transparent layer. Even better, PureBasic allows you to perform real-time effects like real shadow, alphablending, coloring, zoom, rotation all this in windowed or full screen mode !

After initializing the screen and sprite environment via `InitSprite()` you can start opening a full-size or windowed screen.

Windows: DirectX 9 is used for screen creation which your programs to use hardware acceleration if available. Two additional Subsystems are also available, depending of the needs: "OpenGL" and "DirectX11" which respectively leverage OpenGL and DirectX11 to handle screen creation.

Linux: OpenGL is used to manage the screen which allows to use hardware acceleration.

MacOS X: OpenGL is used to manage the screen which allows to use hardware acceleration.

166.1 CatchSprite

Syntax

```
Result = CatchSprite(#Sprite, *MemoryAddress [, Mode])
```

Description

Loads the specified sprite from the given memory area. A screen should be opened with `OpenScreen()` or `OpenWindowedScreen()` before loading a sprite. Sprites can be in BMP format or any other format supported by the ImagePlugin library . A caught sprite can be freed by using the `FreeSprite()` function. The 'CatchSprite' function is useful when using the 'IncludeBinary ' PureBasic keyword. The images can be packed inside the executable. Nevertheless, use this option very carefully, as it will take more memory than storing the file in an external file (the file will be in both executable memory and loaded into physical memory).

The following functions can be used to enable automatically more image formats:

`UseJPEGImageDecoder()`

`UseJPEG2000ImageDecoder()`

`UsePNGImageDecoder()`

`UseTIFFImageDecoder()`

`UseTGAImageDecoder()`

Parameters

#Sprite A number to identify the new sprite. `#PB_Any` can be used to auto-generate this number.

***MemoryAddress** The memory address holding the image used to create the sprite.

Mode (optional) It can be a combination of the following values (with the '||' operator):

```
#PB_Sprite_PixelCollision: Add special information to handle
pixel collision through SpritePixelCollision()
.
#PB_Sprite_AlphaBlending : Sprite is created with per pixel
alpha-channel support, needed for DisplayTransparentSprite()
.
The image format has to support
it (only PNG and TIFF for now).
```

Return value

Nonzero if the sprite has been created, zero otherwise. If `#PB_Any` was used for the `#Sprite` parameter then the generated number is returned on success.

Example

```
1  Debug CatchSprite(0, ?Pic)
2
3  DataSection
4      Pic:
5          IncludeBinary "Sprite.bmp"
6  EndDataSection
```

Remarks

The "?>" is a pointer to a label. More information about pointers and memory access can be found in the relating chapter [here](#) .

See Also

CreateSprite() , LoadSprite()

166.2 ClipSprite

Syntax

```
ClipSprite(#Sprite, x, y, Width, Height)
```

Description

Adds a clip zone to the specified sprite. For example, if a sprite is 100*100 (Width*Height) and a clipping zone is added as x=10, y=10, Width=20, Height=20 then when the sprite is displayed, only the rectangular area starting from x=10, y=10 with width=20 and height=20 will be displayed.

Parameters

#Sprite The sprite to clip.

x, y The clipping start position (in pixels). `#PB_Default` can be used as value to remove the clipping.

Width, Height The clipping size (in pixels). `#PB_Default` can be used as value to remove the clipping.

Return value

None.

Remarks

On some older gfx cards the clipping with `ClipSprite()` doesn't work properly, if the sprite is larger than the used screen.

See Also

`DisplaySprite()` , `DisplayTransparentSprite()`

166.3 CopySprite

Syntax

```
Result = CopySprite(#Sprite1, #Sprite2 [, Mode])
```

Description

Copy the #Sprite1 to #Sprite2.

Parameters

#Sprite1 The source sprite to copy.

#Sprite2 A number to identify the new copied sprite. `#PB_Any` can be used to auto-generate this number. If #Sprite2 doesn't exists, it is created.

Mode (optional) It can be a combination of the following values (with the '||' operator):

```
#PB_Sprite_PixelCollision: Add special information to handle
                           pixel collision through SpritePixelCollision()
.
#PB_Sprite_AlphaBlending : Sprite is created with per pixel
                           alpha-channel support, needed for DisplayTransparentSprite()
.
```

Return value

Nonzero if the sprite has been copied, zero otherwise. If `#PB_Any` was used for the #Sprite2 parameter then the generated number is returned on success.

166.4 CreateSprite

Syntax

```
Result = CreateSprite(#Sprite, Width, Height [, Flags])
```

Description

Creates an empty sprite with the specified dimensions. `SpriteOutput()` can be used to draw on the sprite.

Parameters

#Sprite A number to identify the new sprite. **#PB_Any** can be used to auto-generate this number.

Width, Height The size of the new sprite (in pixels).

Flags (optional) It can be a combination of the following values (with the '||' operator):

```
#PB_Sprite_PixelCollision: Add special information to handle
pixel collision through SpritePixelCollision()
.
#PB_Sprite_AlphaBlending : Sprite is created with per pixel
alpha-channel support, needed for DisplayTransparentSprite()
.
#PB_Sprite_Transparent : Sprite is created fully transparent
(alpha channel value set to 0). If used, DrawingMode()
should be set to #PB_DrawingMode_AllChannels when using
SpriteOutput()
```

Return value

Nonzero if the sprite has been created, zero otherwise. If **#PB_Any** was used for the **#Sprite** parameter then the generated number is returned on success.

Remarks

CreateSprite() has to be called in the same thread where OpenScreen() was called.

Example: CreateSprite using transparent flag

```
1  InitSprite()
2
3  OpenWindow(0, 0, 0, 640, 480, "A screen in a window...",
   #PB_Window_SystemMenu | #PB_Window_SizeGadget |
   #PB_Window_ScreenCentered)
4  If OpenWindowedScreen(WindowID(0), 0, 0, 640, 480, #True, 0, 0)
5
6      CreateSprite(0, 64, 64, #PB_Sprite_AlphaBlending |
   #PB_Sprite_Transparent)
7      If StartDrawing(SpriteOutput())
8          DrawingMode(#PB_2DDrawing_AllChannels)
9          Circle(16, 16, 8, RGBA(255, 0, 0, 255)) ; Opaque red circle
10         Circle(32, 32, 8, RGBA(0, 255, 0, 128)) ; Semi transparent
   green circle
11         StopDrawing()
12     EndIf
13
14     Repeat
15         Repeat ; It's very important to process all the events
   remaining in the queue at each frame
16             Event = WindowEvent()
17             If Event = #PB_Event_CloseWindow
18                 End
19             EndIf
20         Until Event = 0
21
```

```

22     ClearScreen( RGB(0, 0, 0) ) ; A black background
23
24     DisplayTransparentSprite(0, 300, 200)
25
26     FlipBuffers()
27     Forever
28
29 Else
30     Debug "Can't open windowed screen!"
31 EndIf

```

See Also

GrabSprite() , SpriteOutput()

166.5 DisplaySprite

Syntax

```
DisplaySprite(#Sprite, x, y)
```

Description

Displays the #Sprite at the specified position on the current screen. As there is no transparent color or blending, this function is faster than DisplayTransparentSprite() . This function is clipped, so it's perfectly legal to display the sprite outside of the screen.

Parameters

#Sprite The sprite to display.

x, y The coordinate (in pixels) in the screen where the sprite will be display.

Return value

None.

See Also

DisplayTransparentSprite()

166.6 DisplayTransparentSprite

Syntax

```
DisplayTransparentSprite(#Sprite, x, y [, Intensity [, Color]])
```

Description

Display the #Sprite at the specified position on the current screen. The default transparent color is 0 (black - this color will not be displayed). It's possible to change the transparent color with TransparentSpriteColor() . This function is clipped, so it's perfectly legal to display the sprite outside of the screen. The sprite has to be created with #PB_Sprite_AlphaBlending flag to use this command.

Parameters

#Sprite The sprite to display.

x, y The coordinate (in pixels) in the screen where the sprite will be displayed.

Intensity (optional) The intensity level used to display the sprite. Valid value are from 0 (fully transparent) to 255 (fully opaque). Default value is 255.

Color (optional) The solid color used to display the sprite. The sprite will be rendered in only one color. To get a valid color, use RGB() .

Return value

None.

See Also

DisplaySprite()

166.7 FreeSprite

Syntax

```
FreeSprite(#Sprite)
```

Description

Removes the specified sprite from memory. It's not possible to use the sprite anymore after calling this function.

Parameters

#Sprite The sprite to free. If **#PB_All** is specified, all the remaining sprites are freed.

Return value

None.

Remarks

All remaining sprites are automatically freed when the program ends.

166.8 GrabSprite

Syntax

```
Result = GrabSprite(#Sprite, x, y, Width, Height [, Flags])
```

Description

Grabs the screen content from the area x, y, Width, Height and creates a new sprite.

Parameters

#Sprite A number to identify the new grabbed sprite. **#PB_Any** can be used to auto-generate this number.

x, y The position in the screen used to start the grab (in pixels).

Width, Height The size of the grab (in pixels).

Flags (optional) It can be a combination of the following values (with the '||' operator):

```
#PB_Sprite_PixelCollision: Add special information to handle
                           pixel collision through SpritePixelCollision()
.
#PB_Sprite_AlphaBlending : Sprite is created with per pixel
                           alpha-channel support, needed for DisplayTransparentSprite()
. The default transparent color
                           set with TransparentSpriteColor()
is automatically applied.
```

Return value

Nonzero if the sprite has been grabbed, zero otherwise. If **#PB_Any** was used for the **#Sprite** parameter then the generated number is returned on success.

Remarks

GrabSprite() should be always called outside a StartDrawing() : StopDrawing() block. The transparent sprite color is automatically applied when using AlphaBlending flag.

166.9 InitSprite

Syntax

```
Result = InitSprite()
```

Description

Initializes the sprite environment for later use. You must put this function at the top of your source code if you want to use the sprite functions.

Parameters

None.

Return value

Nonzero if the initialization of the sprite environment has succeeded, zero otherwise. You should always test this result to see if the sprite environment has been correctly initialized. If not you must quit the program or disable all the calls to the sprite related functions. This function tries to initialize DirectX 9, so if it fails, it's probably because DirectX is not found or is a previous version.

166.10 IsSprite

Syntax

```
Result = IsSprite(#Sprite)
```

Description

Tests if the given #Sprite number is a valid and correctly initialized, sprite.

Parameters

#Sprite The sprite to use.

Return value

Nonzero if #Sprite is a valid sprite and zero otherwise.

Remarks

This function is bulletproof and can be used with any value. If the 'Result' is not zero then the object is valid and initialized, otherwise it will equal zero. This is the correct way to ensure a sprite is ready to use.

See Also

CreateSprite() , LoadSprite()

166.11 LoadSprite

Syntax

```
Result = LoadSprite(#Sprite, Filename$ [, Mode])
```

Description

Load the specified sprite into memory for immediate use. A screen should be opened with OpenScreen() or OpenWindowedScreen() before loading a sprite.

The sprite can be in BMP format or any other format supported by the ImagePlugin library . The following functions can be used to enable automatically more image formats:

UseGIFImageDecoder()

UseJPEGImageDecoder()

UseJPEG2000ImageDecoder()

UsePNGImageDecoder()

UseTIFFImageDecoder()

UseTGAImageDecoder()

Parameters

#Sprite A number to identify the new loaded sprite. #PB_Any can be used to auto-generate this number.

Filename\$ Name of the image file used to create the sprite.

Mode (optional) It can be a combination of the following values (with the '|' operator):

```
#PB_Sprite_PixelCollision: Add special information to handle
pixel collision through SpritePixelCollision()
.
#PB_Sprite_AlphaBlending : Sprite is created with per pixel
alpha-channel support, needed for DisplayTransparentSprite()
.
The image format has to support it
(only PNG and TIFF for now).
```

Return value

Nonzero if the sprite has been loaded, zero otherwise. If `#PB_Any` was used for the `#Sprite` parameter then the generated number is returned on success.

Remarks

Sprites shouldn't be larger than the used screenmode. Using larger sprites possibly works on some hardware, on others not. Better split your large sprite to several smaller ones.

166.12 SaveSprite

Syntax

```
Result = SaveSprite(#Sprite, Filename$ [, ImagePlugin [, Flags]])
```

Description

Saves the specified sprite to a file. By default, the saved image will be in 24-bit BMP format. Very useful for screenshots when used with the `GrabSprite()` function.

Parameters

#Sprite The sprite to save.

Filename\$ The filename to use for the saved sprite.

ImagePlugin (optional) It can be one of the following constant:

```
#PB_ImagePlugin_BMP      : Save the image in BMP (default).
#PB_ImagePlugin_JPEG     : Save the image in JPEG (the
function UseJPEGImageEncoder()
has to be used).
#PB_ImagePlugin_JPEG2000: Save the image in JPEG2000 (the
function UseJPEG2000ImageEncoder()
has to be used).
#PB_ImagePlugin_PNG      : Save the image in PNG (the
function UsePNGImageEncoder()
has to be used).
```

Flags (optional) Additional control over saving for the specified plug-in. For now, only quality setting is supported: a number from 0 (worse quality) to 10 (maximum quality). Only the JPEG and JPEG 2000 plugins currently support it (default quality is set to '7' if no flags are specified).

Return value

Nonzero if the sprite has been successfully saved, zero otherwise.

See Also

LoadSprite()

166.13 SpriteCollision

Syntax

```
Result = SpriteCollision(#Sprite1, x1, y1, #Sprite2, x2, y2)
```

Description

Tests if the two sprites are overlapping.

Parameters

#Sprite1 The first sprite to test.

x1, y1 Coordinates of the first sprite, in pixels.

#Sprite2 The second sprite to test.

x2, y2 Coordinates of the second sprite, in pixels.

Return value

Nonzero if the two sprites are overlapping, zero otherwise.

Remarks

This routine compares the rectangular areas around the sprites, giving a very fast but not very accurate function (depending on the shapes of your sprites). Very useful for fast arcade games. Zoomed sprites are also supported.

For a more exact collision check use SpritePixelCollision() .

See Also

SpritePixelCollision()

166.14 SpriteDepth

Syntax

```
Result = SpriteDepth(#Sprite)
```

Description

Returns the color depth of the specified sprite.

Parameters

#Sprite The sprite to use.

Return value

The color depth of the specified sprite.

See Also

SpriteWidth() , SpriteHeight()

166.15 SpriteHeight

Syntax

```
Result = SpriteHeight(#Sprite)
```

Description

Returns the height (in pixels) of the specified sprite.

Parameters

#Sprite The sprite to use.

Return value

The height (in pixels) of the specified sprite.

See Also

SpriteWidth() , SpriteDepth()

166.16 SpriteID

Syntax

```
SpriteID = SpriteID(#Sprite)
```

Description

Returns the unique system identifier of the given sprite.

Parameters

#Sprite The sprite to use.

Return value

The ID of the sprite. This sometimes also known as 'Handle'. Look at the extra chapter "Handles and Numbers" for more information.

See Also

CreateSprite() , LoadSprite() , CatchSprite()

166.17 SpritePixelCollision

Syntax

```
Result = SpritePixelCollision(#Sprite1, x1, y1, #Sprite2, x2, y2)
```

Description

Tests if the two sprites are overlapping. `#PB_Sprite_PixelCollision` has to be specified at the sprite creation to have this command working.

Parameters

`#Sprite1` The first sprite to test.

`x1, y1` Coordinates of the first sprite, in pixels.

`#Sprite2` The second sprite to test.

`x2, y2` Coordinates of the second sprite, in pixels.

Return value

Nonzero if the two sprites are overlapping, zero otherwise.

Remarks

This routine performs a transparent pixel exact collision check, giving a slower but very accurate result. To optimize the comparison you should always remove as many transparent pixels as possible so that the sprite size is fully used by the sprite (i.e. do not have large transparent borders around the actual image of the sprite). Zoomed sprites are also supported.

For a faster collision check based only on rectangular bounds, use `SpriteCollision()`.

Attention, it does not work with sprites that have been rotated or transformed.

See Also

`SpriteCollision()`

166.18 SpriteWidth

Syntax

```
Result = SpriteWidth(#Sprite)
```

Description

Returns the width (in pixels) of the specified sprite.

Parameters

`#Sprite` The sprite to use.

Return value

The width (in pixels) of the specified sprite.

See Also

SpriteHeight() , SpriteDepth()

166.19 SpriteOutput

Syntax

```
OutputID = SpriteOutput(#Sprite)
```

Description

Returns the OutputID of the sprite to perform 2D rendering operation on it.

Parameters

#Sprite The sprite to draw on.

Return value

The output ID or zero if drawing is not possible. This value should be passed directly to the StartDrawing() function to start the drawing operation. The return-value is valid only for one drawing operation and cannot be reused.

Remarks

SpriteOutput() has to be called in the same thread where OpenScreen() was called.

Example

```
1 StartDrawing(SpriteOutput(#Sprite))
2 ; do some drawing stuff here...
3 StopDrawing()
```

166.20 TransparentSpriteColor

Syntax

```
TransparentSpriteColor(#Sprite, Color)
```

Description

Use the specified color as the transparent sprite color (when displayed with DisplayTransparentSprite()). Only one color can be set as the transparent color and the previous alpha channel values are lost. If the sprite already have alpha information, this command is probably not needed. The sprite has to be created with **#PB_Sprite_AlphaBlending** flag to use this command.

Parameters

#Sprite The sprite to use. If **#PB_Default** is used, then the default color (black - RGB(0,0,0)) is changed to the new given one and all future loaded or created sprites will use this color as the transparent one.

Color The new color to handle as transparent color. RGB() can be used to get a valid color value. A table with common colors is available here .

Return value

None.

See Also

`DisplayTransparentSprite()` , `RGB()`

166.21 RotateSprite

Syntax

```
RotateSprite(#Sprite, Angle.f, Mode)
```

Description

Rotates the specified `#Sprite` to the given 'Angle'.

Parameters

#Sprite The sprite to rotate.

Angle.f Angle value, in degree (from 0 to 360). The rotation is performed clockwise.

Mode It can be one the following values:

```
#PB_Absolute: the angle is set to the new angle.
#PB_Relative: the angle is added to the previous angle value.
```

Return value

None.

166.22 SpriteBlendingMode

Syntax

```
SpriteBlendingMode(SourceMode, DestinationMode)
```

Description

Changes the way the sprite are blended with the background (when using `DisplayTransparentSprite()`). This function is for advanced users only. The result can differ depending of the underlying subsystem: for example OpenGL and DirectX doesn't behave the same.

Parameters

SourceMode, DestinationMode Both source and destination mode can be one of the following values:

```
#PB_Sprite_BlendZero
#PB_Sprite_BlendOne
#PB_Sprite_BlendSourceColor
#PB_Sprite_BlendInvertSourceColor
#PB_Sprite_BlendDestinationColor
#PB_Sprite_BlendInvertDestinationColor
#PB_Sprite_BlendSourceAlpha
#PB_Sprite_BlendInvertSourceAlpha
#PB_Sprite_BlendDestinationAlpha
#PB_Sprite_BlendInvertDestinationAlpha
```

The default values are SpriteBlendingMode(#PB_Sprite_BlendSourceAlpha, #PB_Sprite_BlendInvertSourceAlpha).

Return value

None.

166.23 SpriteQuality

Syntax

```
SpriteQuality(quality)
```

Description

Changes the way the sprites are rendered.

Parameters

Quality The display sprite quality. Can be one the following values:

```
#PB_Sprite_NoFiltering      : No filtering, faster but ugly
    when zooming/rotating (default).
#PB_Sprite_BilinearFiltering: Bilinear filtering, slower but
    clean when zooming/rotating.
```

Return value

None.

166.24 TransformSprite

Syntax

```
TransformSprite(#Sprite, x1, y1, [z1], x2, y2, [z2], x3, y3, [z3],
    x4, y4, [z4])
```

Description

Transforms the sprite to the new given coordinates. This is typically used to perform real-time transformations. Warning, as a sprite is a combination of 2 triangles, the transformation could look strange. If one of the optional 'z' parameter is specified, all need to be specified.

Parameters

#Sprite The sprite to transform.

x1, y1 First point coordinate, in pixel

x2, y2 Second point coordinate, in pixel

x3, y3 Third point coordinate, in pixel

x4, y4 Fourth point coordinate, in pixel

z1, z2, z3, z4 The 'z' value, which specify the depth of the points.

Return value

None.

```
;
; x1          x2
;  - - - - -
;  /          //
;  /          /  |
;  /          /  |
;  //         |
;  //         |
;  - - - - -
; x4          x3
;
```

166.25 ZoomSprite

Syntax

```
ZoomSprite(#Sprite, Width, Height)
```

Description

Zooms the specified **#Sprite** from the given dimension.

Parameters

#Sprite The sprite to zoom.

Width New sprite width (in pixels). If **#PB_Default** is specified, the initial sprite width is restored.

Height New sprite height (in pixels). If **#PB_Default** is specified, the initial sprite height is restored.

Return value

None.

Chapter 167

StaticGeometry

Overview

A static geometry is a predefined and pre-rendered geometry form, which can be very complex and still have a very fast rendering. But once created, the geometry can't be moved anymore. `InitEngine3D()` should be called successfully before using the static geometry functions.

167.1 FreeStaticGeometry

Syntax

```
FreeStaticGeometry(#StaticGeometry)
```

Description

Free the given StaticGeometry, previously initialized by `CreateStaticGeometry()` .

Parameters

#StaticGeometry The static geometry to free. If **#PB_All** is specified, all the remaining static geometries are freed.

Return value

None.

Remarks

All remaining static geometries are automatically freed when the program ends.

See Also

`CreateStaticGeometry()`

167.2 IsStaticGeometry

Syntax

```
Result = IsStaticGeometry(#StaticGeometry)
```

Description

Tests if the given `#StaticGeometry` is a valid and correctly initialized static geometry.

Parameters

#StaticGeometry The static geometry to use.

Return value

Returns nonzero if `#StaticGeometry` is a valid static geometry and zero otherwise.

Remarks

This function is bulletproof and can be used with any value. If Result is not zero then the object is valid and initialized, otherwise it returns zero. This is a good way to check that a static geometry is ready to use.

See Also

CreateStaticGeometry()

167.3 CreateStaticGeometry

Syntax

```
Result = CreateStaticGeometry(#StaticGeometry, Width, Height,  
                             Length, EnableShadows)
```

Description

Create an empty static geometry.

Parameters

#StaticGeometry A number to identify the new static geometry. `#PB_Any` can be used to auto-generate this number.

Width Width (in world unit) of the static geometry.

Height Height (in world unit) of the static geometry.

Length Length (in world unit) of the static geometry.

EnableShadows Enables or disables dynamic shadows casting on the static geometry. Set to `#True` to enable it, `#False` otherwise.

Return value

Returns nonzero if the static geometry has been successfully created and zero if not. If `#PB_Any` was used for the `#StaticGeometry` parameter then the generated number is returned on success.

Remarks

If another static geometry was previously loaded with the same `#StaticGeometry` number, then the older static geometry will be automatically freed when creating the new one.

See Also

FreeStaticGeometry()

167.4 AddStaticGeometryEntity

Syntax

```
AddStaticGeometryEntity(#StaticGeometry, EntityID, x, y, z [,  
    ScaleX, ScaleY, ScaleZ [, RotationX, RotationY, RotationZ [,  
    RotationW, Mode]]])
```

Description

Add an entity to the specified #StaticGeometry. The original entity is left untouched by this function and can be freed after the add. The same entity can be added multiple times.

Parameters

#StaticGeometry StaticGeometry to use.

EntityID The EntityID of the entity to add to the static geometry.

x, y, z The position of the entity in the static geometry.

ScaleX, ScaleY, ScaleZ (optional) The scale factor to apply to the added entity.

RotationX, RotationY, RotationZ (optional) The rotation to apply to the added entity.

RotationW (optional) The rotation to apply to the added entity (only used for #PB_Orientation_Quaternion and #PB_Orientation_Direction).

Mode (optional) The rotation mode. It can be one of the following values:

```
#PB_Orientation_PitchYawRoll: 'RotationX' (pitch),  
    'RotationY' (yaw), 'RotationZ' (roll), applied in this order  
    (default).  
#PB_Orientation_Quaternion   : 'RotationX', 'RotationY',  
    'RotationZ', 'RotationW' for quaternion values  
#PB_Orientation_Direction    : 'RotationX', 'RotationY',  
    'RotationZ' for direction vector, and 'RotationW' for  
    rotation (roll).
```

Return value

None.

Remarks

Once all the entities have been added, the command BuildStaticGeometry() has to be called to generate the static geometry.

See Also

CreateStaticGeometry() , BuildStaticGeometry()

167.5 BuildStaticGeometry

Syntax

```
BuildStaticGeometry(#StaticGeometry)
```

Description

Build the final static geometry. Once created, a static geometry can't be modified anymore.

Parameters

`#StaticGeometry` StaticGeometry to use.

Return value

None.

See Also

CreateStaticGeometry() , AddStaticGeometryEntity()

Chapter 168

StatusBar

Overview

A status bar is the bottom bar of a window where some information are displayed on it. This bar is always visible and can be split in several parts.

168.1 AddStatusBarField

Syntax

```
AddStatusBarField(Width)
```

Description

Adds a field to the current statusbar previously created with `CreateStatusBar()` . Each new field is created after the old one.

Parameters

Width The width of the new field in pixels. If sets to `#PB_Ignore` then the field will be resized to fill the remaining free space on the statusbar. Multiple fields can have a width of `#PB_Ignore`, in which case the free space will be divided evenly among these fields.

Return value

None.

Remarks

The following commands can be used to set or change the content of the statusbar field:

- `StatusBarText()`
- `StatusBarImage()`
- `StatusBarProgress()`

See `CreateStatusBar()` for an example.

See Also

`StatusBarText()` , `StatusBarImage()` , `StatusBarProgress()` , `CreateStatusBar()`

168.2 CreateStatusBar

Syntax

```
Result = CreateStatusBar(#StatusBar, WindowID)
```

Description

Create and add an empty #StatusBar to the specified WindowID. Once the bar is created, AddStatusBarField() can be used to setup the different parts of the bar.

Parameters

#StatusBar A number to identify the new status bar. #PB_Any can be used to auto-generate this number.

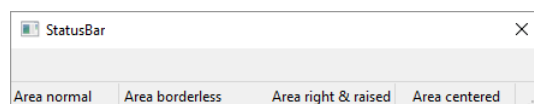
WindowID The window on which the status bar needs to be created. WindowID() can be used to get this value.

Return value

Nonzero if the status bar has been successfully created, zero otherwise.

Example

```
1  If OpenWindow(0, 0, 0, 440, 50, "StatusBar",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered |
3      #PB_Window_SizeGadget)
4      If CreateStatusBar(0, WindowID(0))
5          AddStatusBarField(90)
6          AddStatusBarField(100)
7          AddStatusBarField(#PB_Ignore) ; automatically resize this
8          field
9          AddStatusBarField(100)
10         EndIf
11         StatusBarText(0, 0, "Area normal")
12         StatusBarText(0, 1, "Area borderless", #PB_StatusBar_BorderLess)
13         StatusBarText(0, 2, "Area right", #PB_StatusBar_Right)
14         StatusBarText(0, 3, "Area centered", #PB_StatusBar_Center)
15     Repeat
16         Until WaitWindowEvent() = #PB_Event_CloseWindow
17     EndIf
```



See Also

FreeStatusBar()

168.3 FreeStatusBar

Syntax

```
FreeStatusBar (#StatusBar)
```

Description

Free the given status bar.

Parameters

#StatusBar The status bar to free. If **#PB_All** is specified, all the remaining status bars are freed.

Return value

None.

Remarks

All remaining status bars are automatically freed when the program ends.

168.4 IsStatusBar

Syntax

```
Result = IsStatusBar (#StatusBar)
```

Description

Tests if the given status bar number is a valid and correctly initialized status bar.

Parameters

#StatusBar The status bar to use.

Return value

Nonzero if the status bar is valid, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a status bar is ready to use.

See Also

CreateStatusBar()

168.5 StatusBarImage

Syntax

```
StatusBarImage(#StatusBar, Field, ImageID [, Appearance])
```

Description

Sets the specified Field to display an image.

Parameters

#StatusBar The status bar to use.

Field The field index to set the image. The first field index starts from zero.

ImageID The image to set to the specified statusbar field. It can be easily obtained by using ImageID() .

Appearance (optional) It can be used to alter the look of the field with the following values:

```
#PB_StatusBar_Raised      : raised borders (has no effect on OS
X and Windows with theme enabled)
#PB_StatusBar_BorderLess: without border
#PB_StatusBar_Center     : center the icon in the field
#PB_StatusBar_Right      : align the icon to the right of the
field
```

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 340, 50, "StatusBarImage",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered |
   #PB_Window_SizeGadget)
2      If CreateStatusBar(0, WindowID(0))
3          AddStatusBarField(120)
4          AddStatusBarField(170)
5      EndIf
6
7      If LoadImage(0, "cube16.ico")          ; change path/filename to
   your own image (or icon)
8          StatusBarImage(0, 0, ImageID(0))
9          StatusBarImage(0, 1, ImageID(0), #PB_StatusBar_Right)
10     EndIf
11
12     Repeat
13     Until WaitWindowEvent() = #PB_Event_CloseWindow
14 EndIf
```

168.6 StatusBarID

Syntax

```
StatusBarID = StatusBarID(#StatusBar)
```

Description

Returns the unique system identifier of the status bar.

Parameters

#StatusBar The status bar to use.

Return value

The system identifier. This result is sometimes also known as 'Handle'. Look at the extra chapter Handles and Numbers for more information.

168.7 StatusBarText

Syntax

```
StatusBarText(#StatusBar, Field, Text$ [, Appearance])
```

Description

Sets the text for the specified status bar field.

Parameters

#StatusBar The status bar to use.

Field The field index to set the text. The first field index starts from zero.

Text\$ The text to set to the specified statusbar field.

Appearance (optional) It can be used to alter the look of the field with the following values:

```
#PB_StatusBar_Raised      : raised borders (has no effect on OS  
X and Windows with theme enabled)  
#PB_StatusBar_BorderLess: without border  
#PB_StatusBar_Center      : center the text in the field  
#PB_StatusBar_Right       : align the text to the right of the  
field
```

The options can be combined together with the '||' (OR) operator :

```
#PB_StatusBar_BorderLess | #PB_StatusBar_Right : Borderless  
right aligned field.
```

For a working example look at CreateStatusBar() .

Return value

None.

168.8 StatusBarProgress

Syntax

```
StatusBarProgress(#StatusBar, Field, Value [, Appearance [, Min, Max]])
```

Description

Display a progress bar in the specified 'Field' in the given '#StatusBar'.

Parameters

#StatusBar The status bar to use.

Field The field index to set the progress bar. The first field index starts from zero.

Value This specifies the current progress (relative to the current minimum and maximum). To update this value simply call this command again with a different value.

Appearance (optional) It can be used to alter the look of the field with the following values:

```
#PB_StatusBar_Raised      : raised borders (has no effect on OS  
X and Windows with theme enabled)  
#PB_StatusBar_BorderLess: without border
```

Min, Max (optional) Specifies the boundaries of the progress bar. If they are not specified or have the value `#PB_Ignore` then current boundaries will be used. The default 'Min' and 'Max' values for newly created status bar fields are 0 and 100.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 340, 50, "StatusBarProgress",  
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered |  
   #PB_Window_SizeGadget)  
2      If CreateStatusBar(0, WindowID(0))  
3          AddStatusBarField(120)  
4          AddStatusBarField(170)  
5      EndIf  
6  
7      StatusBarText(0, 0, "ProgressBar !")  
8      StatusBarProgress(0, 1, 25)  
9  
10     Repeat  
11         Until WaitWindowEvent() = #PB_Event_CloseWindow  
12     EndIf
```

168.9 StatusBarHeight

Syntax

```
Result = StatusBarHeight(#StatusBar)
```


Description

Returns the height in pixel of the `#StatusBar`. This is useful for correct calculation on window height when using a statusbar.

Parameters

`#StatusBar` The status bar to use.

Return value

The height in pixels of the status bar.

Chapter 169

String

Overview

Strings are the method used in order to store a list of characters. With the functions supplied in this library, many essential actions may be performed upon strings. By default, Strings are seen as unicode strings.

169.1 Asc

Syntax

```
Result = Asc(String$)
```

Description

Return the first character value of the specified string.

Parameters

String\$ The string to get the first character value.

Return value

The string first character value.

Example

```
1  Debug Asc("!") ; will print '33'
```

Remarks

It is also possible to obtain a constant character value while placing it between apostrophes.

Example

```
1  Debug "Please check that your IDE is in UTF8: File \ File Format
   \ Encoding: Utf8 must be checked."
2  Debug '!'; will print '33'
3  Debug Asc("!"); will print '33'
4  Debug Asc(" "); will print '8364' = 20AC in hexadecimal
5
6  unicode$=""
7  Debug Asc(unicode$); will print '8364'
8  ShowMemoryViewer(@unicode$,8)
```

A table with all Ascii values and their relating figures may be found [here](#) .

See Also

Chr()

169.2 Bin

Syntax

```
Result\$ = Bin(Value.q [, Type])
```

Description

Converts a quad numeric number into a string, in binary format.

Parameters

Value.q The number to convert.

Type (optional) If the value should be handled as another type, one of the following value can be specified:

```
#PB_Quad      : The value is handled as a quad number, ranging
  from 0 to 18446744073709551615 (default)
#PB_Byte      : The value is handled as a byte number, ranging
  from 0 to 255
#PB_Ascii     : The value is handled as a ascii character,
  ranging from 0 to 255
#PB_Word      : The value is handled as a word number, ranging
  from 0 to 65535
#PB_Unicode   : The value is handled as a unicode character,
  ranging from 0 to 65535
#PB_Long      : The value is handled as a long number, ranging
  from 0 to 4294967296
```

Return value

A string holding the binary representation of the specified value.

Example

```
1   Debug Bin(32) ; Will display "100000"
2   Debug Bin(1024*1024*1024*1024) ; Will display
```

Remarks

If leading zero are needed in the output string, use the RSet() function as follows:

```
1  Debug RSet(Bin(32), 16, "0") ; Will display "0000000000100000"
```

See Also

Str() , Val() , Hex()

169.3 Chr

Syntax

Text\\$ = Chr(CharacterValue)

Description

Returns a string created with the given character value.

Parameters

CharacterValue The unicode character value. It must be a value between 0 and \$D7FF or between \$E000 and \$FFFF. The range between \$D800 and \$DFFF is reserved for surrogate characters.

Return value

Returns a string created with the given character value. If the value is invalid, it will returns an empty string.

Example

```
1  Debug Chr(33) ; Will display "!"
```

Remarks

A table with all Ascii values and their relating figures may be found here .

This command works in Unicode and then returns the related characters associated to the given value.

```
1  Debug Chr(8364) ; will display " "
2  Debug Chr($BC) ; will display "¼"
3  Debug Chr($BD) ; will display "½"
```

Example: Special symbols (icons) and a loop with output of latin letters

```
1  Debug Chr($2630) ; ?
2  Debug Chr($25BC) ; ?
3  For i = 65 To 90
4      Debug Chr(i)
5  Next
```

See Also

Asc()

169.4 CountString

Syntax

```
Result = CountString(String$, StringToCount$)
```

Description

Returns the number of occurrences of StringToCount\$ found in String\$.

Parameters

String\$ The input string to use.

StringToCount\$ The string to be counted in the input string.

Return value

The number of occurrences of 'StringToCount\$' found in 'String\$'.

Remarks

The counting is not word based, which means that if the 'StringToCount\$' is a part of a word, it will be counted as well, as shown in the following example.

Example

```
1  Debug CountString("How many 'ow' contains Bow ?", "ow") ; will  
    display 3
```

169.5 EscapeString

Syntax

```
Result\< = EscapeString(String$ [, Mode])
```

Description

Returns the escaped version of the string. UnescapeString() can be used to do the reverse operation.

Parameters

String\$ The string to escape.

Mode (optional) The mode to use when escaping the string.

```

#PB_String_EscapeInternal: escape the string using PureBasic
                          internal format (see General rules
                                for accepted escape sequences)
                          (default).
#PB_String_EscapeXML      : escape the string using the XML
                          escape characters. This can be useful to easily put any
                          string in a
                                XML tree.
#PB_String_EscapeJSON     : escape the string using the JSON
                          escape characters. This can be useful to easily put any
                          string in a
                                JSON value.

```

Return value

An escaped string.

Warning: On Windows, \t does not work with the graphical functions of the 2DDrawing and VectorDrawing libraries.

Example

```

1  Debug EscapeString("Test="+Chr(34)+"Hello"+Chr(34)+".") ; Will
    display "Test=\"Hello\"."
2  Debug EscapeString("<item>Hello</item>", #PB_String_EscapeXML) ;
    Will display "&lt;item&gt;Hello&lt;/item&gt;"

```

See Also

UnescapeString()

169.6 FindString

Syntax

```
Position = FindString(String$, StringToFind$ [, StartPosition [,
Mode]])
```

Description

Find the 'StringToFind\$' within the given 'String\$'.

Parameters

String\$ The string to use.

StringToFind\$ The string to find.

StartPosition (optional) The start position to begin the search. The first valid character index is 1. If this parameter isn't specified, the whole string is searched.

Mode (optional) It can be one of the following values:

```

#PB_String_CaseSensitive: case sensitive search (a=a)
                          (default).
#PB_String_NoCase       : case insensitive search (A=a).

```

Return value

Returns the position (in character) of the string to find, or zero if the string isn't found. The first character index is 1.

```
1  Debug FindString("PureBasic", "Bas") ; will display 5
```

Example

```
1  String$ = "This is a simple line...."
2
3  Repeat
4  Last = Position
5  Position = FindString(String$ , "i", Position + 1)
6  If Position > 0
7      Debug "'i' found at position: " + Position
8  EndIf
9  Until Not Position
10
11 Debug "Last position 'i' was found: " + Last
```

See Also

ReplaceString()

169.7 Hex

Syntax

```
Result\$ = Hex(Value.q [, Type])
```

Description

Converts a quad numeric number into a string, in hexadecimal format.

Parameters

Value.q The number to convert.

Type (optional) If the value should be handled as another type, one of the following value can be specified:

```
#PB_Quad    : The value is handled as a quad number, ranging
               from 0 to 18446744073709551615 (default)
#PB_Byte     : The value is handled as a byte number, ranging
               from 0 to 255
#PB_Ascii    : The value is handled as a ascii character,
               ranging from 0 to 255
#PB_Word     : The value is handled as a word number, ranging
               from 0 to 65535
#PB_Unicode  : The value is handled as a unicode character,
               ranging from 0 to 65535
#PB_Long     : The value is handled as a long number, ranging
               from 0 to 4294967296
```

Return value

A string holding the hexadecimal representation of the specified value.

Example

```
1  Debug Hex(12) ; Will display "C"
2  Debug Hex(1234567890) ; Will display "499602D2"
```

Remarks

If leading zero are needed in the output string, use the RSet() function as follows:

```
1  Debug RSet(Hex(12), 4, "0") ; Will display "000C"
```

Example

```
1  Debug Hex(-1)
2  Debug Hex(-1, #PB_Byte)
3  Debug Hex(-1, #PB_Word)
4  Debug Hex(-1, #PB_Long)
5  Debug Hex(-1, #PB_Quad) ; quad value is the default
```

See Also

Str() , Val() , Bin()

169.8 InsertString

Syntax

```
Result\$ = InsertString(String$, StringToInsert$, Position)
```

Description

Inserts 'StringToInsert\$' into 'String\$' at the specified 'Position'.

Parameters

String\$ The string to use.

StringToInsert\$ The string to insert.

Position The position in the string to insert the new string. The first position index is 1.

Return value

A new string with the inserted string at the specified position.

Example

```
1  Debug InsertString("Hello !", "World", 7) ; Will display "Hello
    World!"
2  Debug InsertString("Hello !", "World", 1) ; Will display
    "WorldHello !"
```

See Also

RemoveString()

169.9 LCase

Syntax

Result\\$_ = LCase(String\$)

Description

Returns the string converted into lower case characters.

Parameters

String\$ The string to convert into lowercase.

Return value

The string converted into lowercase.

Remarks

This function also supports accent letters, so if a upper 'É' is found, it will be transformed into 'é'.

Example

```
1  Debug LCase("This is Art") ; Will display "this is art"
```

See Also

UCase()

169.10 Left

Syntax

Result\\$_ = Left(String\$, Length)

Description

Returns the specified number of characters from the left side of the string.

Parameters

String\$ The string to use.

Length The number of characters to return. If this value exceeds the number of characters of the string, it will be returns the whole string.

Return value

A string holding the specified number of characters from the left side of the string.

Example

```
1  Debug Left("This is Art",4) ; Will display "This"
```

See Also

Right()

169.11 Len

Syntax

```
Length = Len(String$)
```

Description

Returns the character length of the string.

Parameters

String\$ The string to use.

Return value

The character length of the string.

Example

```
1  Debug Len("This is Art") ; will display 11
```

See Also

StringByteLength()

169.12 LSet

Syntax

```
Result\$\$ = LSet(String$, Length [, Character$])
```

Description

Adjusts the length of a string by adding characters at the end of the string if necessary to reach the specified length.

Parameters

String\$ The string to adjust.

Length The total length (in characters) of the new string.

Character\$ (optional) The character used to adjust the length of the new string. The default character is 'space'. 'Character\$' has to be a one character length string.

Return value

A new string holding the original string padded with the specified character to fit the length.

Remarks

If the string is longer than the specified length, it will be truncated starting from the right side of the string.

Example

```
1  Debug LSet("L", 8)           ; will display "L          "
2  Debug LSet("L", 8, "-")      ; will display "L-----"
3  Debug LSet("LongString", 4) ; will display "Long"
```

See Also

RSet() , LTrim() , RTrim()

169.13 LTrim

Syntax

```
Result\[ = LTrim(String$ [, Character$])
```

Description

Removes all the specified characters located in the front of a string.

Parameters

String\$ The string to trim.

Character\$ (optional) The character used to trim the string. The default character is 'space'. 'Character\$' has to be a one character length string.

Return value

A new string holding the original string with the front characters removed.

Example

```
1  Debug LTrim("      This is Art      ") ; Will display "This is Art
   "
2  Debug LTrim("!!Hello World!!", "!") ; Will display "Hello
   World!!"
```

See Also

RSet() , LSet() , RTrim() , Trim()

169.14 Mid

Syntax

```
Result\$ = Mid(String$, StartPosition [, Length])
```

Description

Extracts a string of specified length from the given string.

Parameters

String\$ The string to use.

StartPosition Specifies the character position to start the extracting. The first character position is 1.

Length (optional) Specifies how many characters needs to be extracted. If this parameter is omitted, characters are extracted until the end of string.

Return value

A new string holding the extracted characters.

Example

```
1  Debug Mid("Hello", 2) ; Will display "ello"
2  Debug Mid("Hello", 2, 1) ; Will display "e"
```

169.15 RemoveString

Syntax

```
String\$ = RemoveString(String$, StringToRemove$ [, Mode [,
    StartPosition [, NbOccurrences]]])
```

Description

Finds all occurrences of 'StringToRemove\$' within the specified 'String\$' and removes them.

Parameters

String\$ The string to use.

StringToRemove\$ The string to remove.

Mode (optional) It can be one of the following values:

```
#PB_String_CaseSensitive: case sensitive remove (a=a)
                        (default)
#PB_String_NoCase       : case insensitive remove (A=a)
```

StartPosition (optional) Specifies the character position to start the removing. The first character position is 1. If omitted the whole string is used.

NbOccurrences (optional) Specifies how many strings should be removed before stopping the operation. If omitted, all strings are removed.

Return value

A new string without the removed strings.

Example

```
1  Debug RemoveString("This is Art", "is") ; Will display "Th Art"
2  Debug RemoveString("This is Art", "is", #PB_String_CaseSensitive,
    1, 1) ; Will display "Th is Art"
```

See Also

[InsertString\(\)](#)

169.16 ReplaceString

Syntax

```
String\$ = ReplaceString(String$, StringToFind$, ReplacementString$
    [, Mode [, StartPosition [, NbOccurrences]]])
```

Description

Try to find any occurrences of 'StringToFind\$' in the given 'String\$' and replace them with 'ReplacementString\$'.

Parameters

String\$ The string to use.

StringToFind\$ The string to find.

ReplacementString\$ The string to use as replacement.

Mode (optional) It can be a combination of the following values:

```
#PB_String_CaseSensitive : Case sensitive search (a=a)
    (default)
#PB_String_NoCase : Case insensitive search (A=a)
#PB_String_InPlace: In-place replacing. This means that the
    string is replaced directly in the memory.
    The 'StringToFind$' and
    'ReplacementString$' parameter must have the same length.
    This is
        a dangerous option, for advanced users
    only. The advantage is the very high speed of the
        replacement. When using this option, the
    result of ReplaceString() has to be ignored.
    Only the string passed in parameter
    'String$' is changed and it's the result.
    Fixed strings
are not supported using this mode.
```

StartPosition (optional) Specifies the character position to start the replacement. The first character position is 1. If omitted the whole string is used.

NbOccurrences (optional) Specifies how many strings should be replaced before stopping the operation. If omitted, all strings are replaced.

Return value

A new string with the replaced strings (see the `#PB_String_InPlace` mode for a different behavior).

Example

```
1  Debug ReplaceString("This is Art", " is", " was") ; Will display
   "This was Art"
2
3  test$ = "Hello again, hello again"
4  Result$ = ReplaceString(test$, "HELLO", "oh no...",
   #PB_String_NoCase, 10) ; Will display "Hello again, oh no...
   again"
5  Debug Result$
6
7  test$ = "Bundy, Barbie, Buddy"
8  ReplaceString(test$, "B", "Z", #PB_String_InPlace, 1) ; all B
   gets changed to Z (directly in memory, no valid return-value
   here)
9  Debug test$ ; Output of the changed string and will display
   "Zundy, Zarbie, Zuddy"
```

See Also

`RemoveString()` , `InsertString()` , `FindString()` , `ReplaceRegularExpression()`

169.17 Right

Syntax

```
Result\$ = Right(String$, Length)
```

Description

Returns the specified number of characters from the right side of the string.

Parameters

String\$ The string to use.

Length The number of characters to return. If this value exceeds the number of characters of the string, it will be returns the whole string.

Return value

A string holding the specified number of characters from the right side of the string.

Example

```
1  Debug Right("This is Art", 3) ; Will display "Art"
```

See Also

Left()

169.18 RSet

Syntax

```
Result\$_ = RSet(String$, Length [, Character$])
```

Description

Adjusts the length of the string by adding characters at the beginning of the string if necessary to reach the specified length.

Parameters

String\$ The string to adjust.

Length The total length (in characters) of the new string.

Character\$ (optional) The character used to adjust the length of the new string. The default character is 'space'. 'Character\$' has to be a one character length string.

Return value

A new string holding the original string padded with the specified character to fit the length.

Remarks

If the string is longer than the specified length, it will be truncated starting from the right side of the string.

Example

```
1  Debug RSet("R", 8) ; will display " R"
2  Debug RSet("R", 8, "-") ; will display "-----R"
3  Debug RSet("LongString", 4) ; will display "Long"
```

See Also

LSet(), LTrim(), RTrim()

169.19 RTrim

Syntax

```
Result\$_ = RTrim(String$ [, Character$])
```

Description

Removes all the specified characters located at the end of a string.

Parameters

String\$ The string to trim.

Character\$ (optional) The character used to trim the string. The default character is 'space'. 'Character\$' has to be a one character length string.

Return value

A new string holding the original string with the end characters removed.

Example

```
1  Debug RTrim("This is Art      ")    ; Will display "This is Art"
2  Debug RTrim("Hello World!!", "!")  ; Will display "Hello World"
```

See Also

RSet() , LSet() , LTrim() , Trim()

169.20 StringByteLength

Syntax

```
Result = StringByteLength(String$ [, Format])
```

Description

Returns the number of bytes required to store the string in memory in a given format.

Parameters

String\$ The string to use.

Format (optional) The destination format to use. It can be one of the following value:

```
#PB_Ascii   : Ascii format
#PB_UTF8    : UTF8 format
#PB_Unicode : Unicode format
```

Return value

The number of bytes required to store the string in memory in a given format.

Remarks

The number of bytes returned does not include the terminating Null-Character of the string. The size of the Null-Character is 1 byte for Ascii and UTF8 and 2 bytes for Unicode.

Example

```
1  Debug StringByteLength("ä", #PB_UTF8) ; will display 2
```


See Also

Len()

169.21 StringField

Syntax

```
Result\$ = StringField(String$, Index, Delimiter$)
```

Description

Returns the string field at the specified index.

Parameters

String\$ The string to parse.

Index The field index to return. The first index is 1.

Delimiter\$ The string delimiter to use to separate the fields. It can be a multi-characters delimiter.

Example

```
1 For k = 1 To 6
2   Debug StringField("Hello I am a split string", k, " ")
3 Next
```

169.22 StrF

Syntax

```
Result\$ = StrF(Value.f [, NbDecimal])
```

Description

Converts a float number into a string.

Parameters

Value.f The value to convert.

NbDecimal (optional) The maximum number of decimal places for the converted number. If omitted, it will be set to 10 decimal places, with removing the trailing zeros. The number will be rounded, if 'NbDecimal' is smaller than existing decimal places of 'Value.f'.

Return value

A string holding the converted value.

Remarks

Signed integer numbers have to be converted with Str() and unsigned numbers with StrU() . It is possible to omit this command when concatenating string and float, it will then use the default behaviour of StrF() .

Example

```
1 value.f = 10.54
2 Debug "Result: " + StrF(value)      ; we do not use the 2nd
   parameter, so we get a float number rounded to 10 decimal places
3 Debug "Result: " + value            ; same as previous line
4 Debug "Result: " + StrF(value,2)    ; we want a result with two
   decimal places, no rounding needed as we have only two
5 Debug "Result: " + StrF(value,0)    ; we want a result with no
   decimal places, so the value is rounded
```

See Also

StrD() , Str() , StrU() , FormatNumber()

169.23 StrD

Syntax

```
Result\$ = StrD(Value.d [, NbDecimal])
```

Description

Converts a double number into a string.

Parameters

Value.d The value to convert.

NbDecimal (optional) The maximum number of decimal places for the converted number. If omitted, it will be set to 10 decimal places, with removing the trailing zeros. The number will be rounded to the nearest decimal place (compare with Round()), if 'NbDecimal' is smaller than existing decimal places of 'Value.d'.

Return value

A string holding the converted value.

Remarks

Signed integer numbers have to be converted with Str() and unsigned numbers with StrU() . It is possible to omit this command when concatenating string and double, it will then use the default behaviour of StrD() .

Example

```
1 Value.d = 10.54
2 Debug "Result: " + StrD(Value)      ; we do not use the 2nd
   parameter, so we get a float number rounded to 10 decimal places
3 Debug "Result: " + Value            ; same as previous line
4 Debug "Result: " + StrD(Value, 2)   ; we want a result with two
   decimal places, no rounding needed as we have only two
5 Debug "Result: " + StrD(Value, 0)   ; we want a result with no
   decimal places, so the value is rounded
```

See Also

StrF() , Str() , StrU() , FormatNumber()

169.24 Str

Syntax

```
Result\$ = Str(Value.q)
```

Description

Convert a signed quad number into a string.

Parameters

Value.q The value to convert.

Return value

A string holding the converted value.

Remarks

Floats must be converted with StrF() , doubles with StrD() and unsigned numbers with StrU() . It is possible to omit this command when concatenating string and integer, it will then use the default behaviour of Str() .

Example

```
1 Value.q = 10000000000000000001
2 Debug "Result: " + Str(Value)
```

See Also

Val() , Hex() , Bin() , StrF() , StrD() , StrU() , FormatNumber()

169.25 StrU

Syntax

```
Result\$ = StrU(Value.q [, Type])
```

Description

Converts an unsigned numeric number into a string.

Parameters

Value.q The value to convert.

Type (optional) It can be one of the following value:

```
#PB_Quad    : The value is handled as a quad number, ranging
               from 0 to 18446744073709551615 (default)
#PB_Byte     : The value is handled as a byte number, ranging
               from 0 to 255
#PB_Ascii    : The value is handled as a ascii character,
               ranging from 0 to 255
#PB_Word     : The value is handled as a word number, ranging
               from 0 to 65535
#PB_Unicode  : The value is handled as a unicode character,
               ranging from 0 to 65535
#PB_Long     : The value is handled as a long number, ranging
               from 0 to 4294967296
```

Return value

A string holding the converted value.

Remarks

Signed integer numbers must be converted with Str() and float numbers with StrF() or StrD() .

Example

```
1  byte.b = 255
2  Debug Str(byte) ; Will display -1
3  Debug StrU(byte, #PB_Byte) ; Will display 255
```

See Also

Str() , StrD() , StrF() , Val() , ValD() , ValF() , Hex() , Bin() , FormatNumber()

169.26 ReverseString

Syntax

```
Result\$ = ReverseString(String\$)
```

Description

Reverses all the characters in the 'String\$'. The last characters becomes the first characters, and vice-versa.

Parameters

String\$ The string to reverse.

Return value

A string holding the reversed string.

Example

```
1  Debug ReverseString("Hello") ; Will display "olleH"
```

169.27 Space

Syntax

```
Result\$ = Space(Length)
```

Description

Creates a string of the given length filled with 'space' characters.

Parameters

Length The length (in characters) of the new string.

Return value

A new string filled with 'space' characters.

Example

```
1  Debug "-" + Space(5) + "-" ; Will display "-     -"
```

169.28 Trim

Syntax

```
Result\$ = Trim(String$ [, Character$])
```

Description

Removes all the specified characters located at the beginning and at the end of a string.

Parameters

String\$ The string to trim.

Character\$ (optional) The character used to trim the string. The default character is 'space'. 'Character\$' has to be a one character length string.

Return value

A new string holding the original string without the removed characters.

Example

```
1  Debug Trim("   Hello   ") ; Will display "Hello"
2  Debug Trim("!!Hello!!", "!") ; Will display "Hello"
```

See Also

LTrim() , RTrim()

169.29 UCase

Syntax

```
Result\$ = UCase(String$)
```

Description

Returns the original string converted into upper case characters.

Parameters

String\$ The string to convert into uppercase.

Return value

The string converted into uppercase.

Remarks

This function also supports accent letters, so if a lower 'é' is found, it will be transformed into 'É'.

Example

```
1  Debug UCase("This is Art") ; Will display "THIS IS ART"
```

See Also

LCase()

169.30 UnescapeString

Syntax

```
Result\$ = UnescapeString(String$ [, Mode])
```

Description

Returns the unescaped version of the string. EscapeString() can be used to do the reverse operation.

Parameters

String\$ The string to unescape.

Mode (optional) The mode to use when unescaping the string.

```
#PB_String_EscapeInternal: unescape the string using
                          PureBasic internal format (see General rules
                          for accepted escape sequences)
                          (default).
#PB_String_EscapeXML      : unescape the string using the XML
                          escape characters. This can be useful to easily read any
                          string from a
                          XML tree.
#PB_String_EscapeJSON     : unescape the string using the JSON
                          escape characters. This can be useful to easily read any
                          string from a
                          JSON value.
```

Return value

An unescaped string.

Warning: On Windows, \t does not work with the graphical functions of the 2DDrawing and VectorDrawing libraries.

Example

```
1  Debug UnescapeString(~"Test=\"Hello\".") ; Will display
   "Test="Hello". "
2  Debug UnescapeString("&lt;item&gt;Hello&lt;/item&gt;",
   #PB_String_EscapeXML) ; Will display "<item>Hello</item>"
```

See Also

EscapeString()

169.31 ValD

Syntax

```
Result.d = ValD(String$)
```

Description

Converts a string into a double value. The string must be a double in decimal or in scientific (exponent) format. The number parsing stops at the first non numeric character.

Parameters

String\$ The string to convert.

Return value

The double value of the string.

Remarks

Strings holding an integer can also be converted with `Val()` , and 32-bit floats with `ValF()` (with less accuracy than `ValD()`).
"NaN", "-Infinity" and "+Infinity" are allowed.

Example

```
1  Debug ValD("10.000024") ; will display 10.000024
2  Debug ValD("1.2345e-2") ; will display 0.012345
```

See Also

`ValF()` , `Val()` , `Str()` , `StrF()` , `StrD()`

169.32 ValF

Syntax

```
Result.f = ValF(String$)
```

Description

Converts a string into a float value. The string must be a float in decimal or in scientific (exponent) format. The number parsing stops at the first non numeric character.

Parameters

String\$ The string to convert.

Return value

The float value of the string.

Remarks

Strings holding an integer can also be converted with `Val()` and 64-bit floats with `ValD()` (with more accuracy than `ValF()`).
"NaN", "-Infinity" and "+Infinity" are allowed.

Example

```
1  Debug ValF("10.75") ; will display 10.75
2  Debug ValF("1.2345e+3") ; will display 1234.5
```

See Also

`ValD()` , `Val()` , `Str()` , `StrF()` , `StrD()`

169.33 Val

Syntax

```
Result.q = Val(String$)
```

Description

Converts a string into a quad numeric value. The string may be an integer in decimal, hexadecimal (with '\$' prefix) or binary (with '%' prefix) format. The number parsing stops at the first non numeric character.

Parameters

String\$ The string to convert.

Return value

The numeric value of the string.

Remarks

Strings holding a 32-bit floats may be converted with ValF() and 64-bit floats with ValD() .

Example

```
1  Debug Val("1024102410241024") ; will print '1024102410241024'.
2  Debug Val("$10FFFFFFFF")       ; will print '73014444031'.
3  Debug Val("%1000")             ; will print '8'.
```

See Also

ValD() , ValF() , Str() , StrF() , StrD()

169.34 Ascii

Syntax

```
*Buffer = Ascii(String$)
```

Description

Creates an Ascii representation of the string. When the buffer is no longer needed, the buffer needs to be freed with FreeMemory().

Parameters

String\$ The string to convert.

Return value

The Ascii representation of the string.

Remarks

This function is mainly useful when interacting with third-party libraries which requires Ascii as input. Pseudotype 'p-ascii' can also be used to automated the converting process when importing external functions.

The buffer includes a null-terminated character.

Example

```
1  *Ascii = Ascii("Hél  ")
2  ShowMemoryViewer(*Ascii, 4)
```

See Also

UTF8() , FreeMemory() , MemorySize()

169.35 UTF8

Syntax

```
*Buffer = UTF8(String$)
```

Description

Creates a buffer with an UTF8 representation of the string. When the buffer is no longer needed, the buffer needs to be freed with FreeMemory().

Parameters

String\$ The string to convert.

Return value

The UTF8 representation of the string.

Remarks

This function is mainly useful when interacting with third-party libraries which requires UTF8 as input. Pseudotype 'p-utf8' can also be used to automated the converting process when importing external functions.

The buffer includes a null-terminated character.

Example

```
1  *UTF8 = UTF8("H     ")
2  ShowMemoryViewer(*UTF8, MemorySize(*UTF8))
3
4  Debug PeekS(*UTF8, -1, #PB_UTF8) ; Displays "H     "
```

Example

```
1 Macro Unicode(Mem, Type = #PB_Ascii)
2   PeekS(Mem, -1, Type)
3 EndMacro
4
5 *Mem1 = Ascii("Test - éâïöËÜ")
6 *Mem2 = UTF8("Test - éâïöËÜ")
7
8 Text.s = Unicode(*Mem1)
9 Debug Text
10
11 Text2.s = Unicode(*Mem2, #PB_UTF8)
12 Debug Text2
```

See Also

Ascii() , FreeMemory() , MemorySize()

169.36 FormatNumber

Syntax

```
Result\$ = FormatNumber(Number.d [, NbDecimals [, DecimalPoint$ [,
    ThousandSeparator$]]])
```

Description

Format a number into money-like format.

Parameters

Number The number to format

NbDecimals (optional) Number of decimals to display.

DecimalPoint\$ (optional) The string to use to split the decimal and integer parts. It can be a multiple character string. Default value is ”.”.

ThousandSeparator\$ (optional) The string to use to separate thousands. It can be a multiple character string. Default value is ”,”.

Return value

The formatted number.

Example

```
1 Debug FormatNumber(125400.25) ; Will display: 125,400.25
```

See Also

Str() , StrU() , StrF() , StrD()

Chapter 170

SysTray

Overview

The SysTray (also called the system tray) is the right part of the taskbar which contains some icons and the date/hour. PureBasic provides full access to this area and the capability to add any number of icons to it.

On some Linux distributions (like Ubuntu), the SysTray icons can be hidden by default. For more information see this [link](#).

170.1 AddSysTrayIcon

Syntax

```
Result = AddSysTrayIcon(#SysTrayIcon, WindowID, ImageID)
```

Description

Adds an icon in the SysTray area.

When an event occurs on any of the SysTray icons the `#PB_Event_SysTray` event is sent.

`EventGadget()` can be used to know which SysTrayIcon has been used. `EventType()` functions is also updated by this function.

Parameters

#SysTrayIcon A number to identify the new SysTray icon. `#PB_Any` can be used to auto-generate this number.

WindowID The window on which the SysTray events will be handled. `WindowID()` can be used to get this value.

ImageID The image to use for the SysTray icon. It is advised to use a PNG image in order to have a transparency layer. `ImageID()` may be used to get this ID easily.

Return value

Nonzero if the SysTray icon has been successfully added, zero otherwise.

Remarks

All SysTray icons are automatically removed when the program ends.

With the QT subsystem, the left-click returns a double-left-click event and the middle-click returns a left-click event.

For normal operation, a systray event must always be associated with a menu .

Example

```
1 ; Invisible window to just have the systray
2 OpenWindow(0, 0, 0, 10, 10, "", #PB_Window_Invisible)
3
4 UsePNGImageDecoder()
5 AddSysTrayIcon(0, WindowID(0), LoadImage(0, #PB_Compiler_Home +
6     "examples/sources/Data/world.png"))
7
8 ; Create a pop-up menu to be displayed by the systray with a
9     systray look
10 CreatePopupMenu(0, #PB_Menu_SysTrayLook)
11     MenuItem(0, "About PureBasic...")
12     MenuBar()
13     MenuItem(1, "Exit")
14
15 ; Associate the menu to the systray
16 SysTrayIconMenu(0, MenuID(0))
17
18 Repeat
19     Select WaitWindowEvent()
20     Case #PB_Event_Menu
21         Select EventMenu()
22         Case 1 ; Exit
23             RemoveSysTrayIcon(0)
24             End
25         EndSelect
26     EndSelect
27 EndRepeat
28 ForEver
```

See Also

RemoveSysTrayIcon() , ChangeSysTrayIcon()

170.2 ChangeSysTrayIcon

Syntax

```
ChangeSysTrayIcon(#SysTrayIcon, ImageID)
```

Description

Changes the specified icon in the SysTray area.

Parameters

#SysTrayIcon The SysTray icon to change.

ImageID The new image to use for the SysTray icon. It is advised to use a PNG image in order to have a transparency layer. ImageID() may be used to get this ID easily.

Return value

None.

Example

```
1 ; Invisible window to just have the systray
2 OpenWindow(0, 0, 0, 10, 10, "", #PB_Window_Invisible)
3
4 ; Create an image with a yellow box to be used for the Systray
  icon:
5 If CreateImage(0, 16, 16) And StartDrawing(ImageOutput(0))
6   Box(0, 0, 16, 16, $00DDDD)
7   StopDrawing()
8 EndIf
9
10 ; Associate the menu to the systray
11 AddSysTrayIcon(0, WindowID(0), ImageID(0))
12
13 ; Create a pop-up menu to be used with the Systray icon later:
14 CreatePopupMenu(0, #PB_Menu_SysTrayLook)
15   MenuItem(0, "Exit")
16
17 SysTrayIconMenu(0, MenuID(0))
18
19 AddWindowTimer(0, 0, 1000) ; Add a window timer, which should
   cause a change of the Systray icon after every 1000 ms
20
21 Repeat
22   Select WaitWindowEvent()
23     Case #PB_Event_Timer
24       ; Update the Systray icon image with a randomly sized red
       circle on the yellow background
25       If StartDrawing(ImageOutput(0))
26         Box(0, 0, 16, 16, $00DDDD)
27         Radius = Random(6, 2)
28         Circle(8, 8, Radius, $0000DD)
29         StopDrawing()
30         ChangeSysTrayIcon(0, ImageID(0))
31       EndIf
32
33     Case #PB_Event_Menu
34       Select EventMenu()
35         Case 0 ; Quit after clicking on the 'Exit' menu item
           associated to the Systray icon
36           RemoveSysTrayIcon(0)
37           End
38       EndSelect
39     EndSelect
40   ForEver
```

See Also

AddSysTrayIcon()

170.3 IsSysTrayIcon

Syntax

```
Result = IsSysTrayIcon(#SysTrayIcon)
```

Description

Tests if the given SysTray icon is valid and correctly initialized.

Parameters

#SysTrayIcon The SysTray icon to test.

Return value

Nonzero if the SysTray icon is valid, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a SysTray icon is ready to use.

See Also

AddSysTrayIcon()

170.4 SysTrayIconMenu

Syntax

```
SysTrayIconMenu(#SysTrayIcon, MenuID)
```

Description

Associates the specified popup menu with the SysTray icon. The menu should be created with CreatePopupMenu() using the #PB_Menu_SysTrayLook flag. It's the best way to associate a popup menu with a SysTray icon as it will display the popup menu at the correct position on all OS when the user click on the icon.

Parameters

#SysTrayIcon The systray icon to use.

MenuID The popup menu to associate with the SysTray. MenuID() can be used to get this value.

Return value

None.

Example

```
1 ; Invisible window to just have the systray
2 OpenWindow(0, 0, 0, 10, 10, "", #PB_Window_Invisible)
3
4 UsePNGImageDecoder()
5 AddSysTrayIcon(0, WindowID(0), LoadImage(0, #PB_Compiler_Home +
6     "examples/sources/Data/world.png"))
7
8 ; Create a pop-up menu to be displayed by the systray with a
9     systray look
```

```

8   CreatePopupMenu(0, #PB_Menu_SysTrayLook)
9   MenuItem(0, "About PureBasic...")
10  MenuBar()
11  MenuItem(1, "Exit")
12
13  ; Associate the menu to the systray
14  SysTrayIconMenu(0, MenuID(0))
15
16  Repeat
17      Select WaitWindowEvent()
18          Case #PB_Event_Menu
19              Select EventMenu()
20                  Case 1 ; Exit
21                      RemoveSysTrayIcon(0)
22                      End
23              EndSelect
24          EndSelect
25  ForEver

```

See Also

CreatePopupMenu()

170.5 SysTrayIconToolTip

Syntax

```

SysTrayIconToolTip(#SysTrayIcon, Text$)

```

Description

Associates the specified Text\$ with the SysTray icon. Tool-tip text is the text which is displayed when the mouse cursor hovers over the icon for a period of time (yellow floating box).

Parameters

#SysTrayIcon The SysTray icon to use.

Text\$ The text to use for the SysTray icon's tool-tip.

Return value

None.

170.6 RemoveSysTrayIcon

Syntax

```

RemoveSysTrayIcon(#SysTrayIcon)

```

Description

Remove the specified SysTray icon.

Parameters

#SysTrayIcon The SysTray icon to remove. If **#PB_All** is specified, all the remaining SysTray icons are freed.

Return value

None.

Remarks

All remaining SysTray icons are automatically removed when the program ends.

Chapter 171

System

Overview

The system library offers access to some system specific information, like the number of CPUs, amount of memory available, and more.

171.1 CocoaMessage

Syntax

```
Result = CocoaMessage(ReturnValueAddress, Object, Method$ [,
    ParameterValue [, Parameter$, [, ParamaterValue, ...]]])
```

Description

For advanced users. Available on OS X only, it allows to easily send an Objective-C message to the OS X framework and access any API. Usually Objective-C use brackets to have a clear syntax for messages. As PureBasic doesn't have built-in Objective-C support, it needs to emulate it, so the syntax is a bit different. Once learnt, it's easy to call the required API. To get more examples, please read the following thread on the [forums](#).

Parameters

ReturnValueAddress If the API call return a structure or a type different than 'integer', this field is used to set the returning result. An address to the structure or variable needs to be specified. If zero is specified, the result will be ignored.

Object The object on which the Objective-C methods will be called. It can be zero if the method is a static method (mostly when creating an object).

Method\$ The method to call on the object, usually followed by a semicolon (':'). If the method needs a structure as parameter, '@' needs to be appended after the semicolon. If the method expect a string as parameter, '\$' can be appended after the semicolon, so the string will be automatically converted in a temporary NSString. This is not required, but it can be useful and ease the coding. If the method isn't supported by the object, a debugger message will be raised at runtime.

ParameterValue (optional) The parameter value associated to the previous method.

Parameter\$ (optional) The next method parameter. PureBasic support up to 7 method parameters.

Return value

Integer return value. Useful for object creation id.

Remarks

PureBasic has already setup a temporary memory pool which is flushed every time WindowEvent() or WaitWindowEvent() is called. If you need to release big objects immediately, you will have to create a local memory pool around your calls.

Example: with string

```
1 ; Objective-C:
2 ;   ColorList = [NSColorList colorListNamed:@"Crayons"];
3 ;
4 ColorList = CocoaMessage(0, 0, "NSColorList colorListNamed:$",
   @"Crayons") ; Will create an NSString for "Crayons" under the
   hood
```

Example: with complex type

```
1 ; Objective-C:
2 ;   Transform = [NSAffineTransform transform];
3 ;
4 Transform = CocoaMessage(0, 0, "NSAffineTransform transform") ;
   Get an identity transform
5
6 ; Objective-C:
7 ;   [Transform scaleXBy:sx yBy:sy];
8 ;
9 sx.CGFloat = 5.5
10 sy.CGFloat = 20
11 CocoaMessage(0, Transform, "scaleXBy:@", @sx, "yBy:@", @sy) ;
   Scale x by 5.5, y by 20. As sx is not an integer, '@' needs to
   be specified
12
13 ; Objective-C:
14 ;   NSAffineTransform TransformStruct = [Transform
   transformStruct];
15 ;
16 CocoaMessage(@TransformStruct.NSAffineTransform, Transform,
   "transformStruct") ; Get the transform structure
17
18 Debug TransformStruct\m11 ; debug outputs 5.5
```

Supported OS

MacOS X

171.2 CPUName

Syntax

```
Result\$ = CPUName()
```

Description

Returns the name of the CPU.

Parameters

None.

Return value

Returns the name of the CPU (the full vendor information). There is no standardized output across CPU, so this information shouldn't be used to identify the CPU at runtime, but can be useful for logging purpose.

Example

```
1  Debug CPUName() ; Could print: "Intel(R) Core(TM) i7 CPU  
    860 @ 2.80GHz"
```

171.3 Delay

Syntax

`Delay(Time)`

Description

Halts the program execution for the specified amount of time.

Parameters

Time The delay time in milliseconds. The actual delay may be longer than the specified time.

Return value

None.

Remarks

Delay is especially useful in event loops with `WindowEvent()` or `ExamineKeyboard()`, so these functions do not "eat" the entire CPU power.
In fact, this function halts the current thread.

See Also

`ElapsedMilliseconds()`

171.4 ElapsedMilliseconds

Syntax

`Result.q = ElapsedMilliseconds()`

Description

Returns the number of milliseconds that have elapsed since a specific time in the past.

Parameters

None.

Return value

Returns the elapsed time in milliseconds.

Remarks

The absolute value returned is of no use since it varies depending on the operating system. Instead, this function should be used to calculate time differences between multiple `ElapsedMilliseconds()` calls.

This function is relatively accurate: it may have a slight variation, depending on which operating system it is executed on, this is due to the fact that some systems have a lower timer resolution than others.

Example

```
1  StartTime.q = ElapsedMilliseconds()      ; Get the actual value
2  Delay(1000)                               ; Wait 1000 milliseconds
3  Debug ElapsedMilliseconds() - StartTime ; Displayed value should
    be about 1000 milliseconds
```

See Also

`Delay()`

171.5 DoubleClickTime

Syntax

```
Result = DoubleClickTime()
```

Description

Returns the system setting for the double-click time. If two mouse clicks happen within this time, they are considered a double-click.

Parameters

None.

Return value

Returns the double-click time in milliseconds.

Remarks

This function can be used to make mouse handling in a screen or `CanvasGadget()` consistent with the overall system settings.

See Also

ElapsedMilliseconds() , OpenScreen() , OpenWindowedScreen() , CanvasGadget()

171.6 OSVersion

Syntax

```
Result = OSVersion()
```

Description

Returns the version of the operating system on which the program has been launched.

Parameters

None.

Return value

Returns one of the following values, depending on the OS on which the command run:

Windows

```
#PB_OS_Windows_NT3_51
#PB_OS_Windows_95
#PB_OS_Windows_NT_4
#PB_OS_Windows_98
#PB_OS_Windows_ME
#PB_OS_Windows_2000
#PB_OS_Windows_XP
#PB_OS_Windows_Server_2003
#PB_OS_Windows_Vista
#PB_OS_Windows_Server_2008
#PB_OS_Windows_7
#PB_OS_Windows_Server_2008_R2
#PB_OS_Windows_8
#PB_OS_Windows_Server_2012
#PB_OS_Windows_8_1
#PB_OS_Windows_Server_2012_R2
#PB_OS_Windows_10
#PB_OS_Windows_Server_2016
#PB_OS_Windows_Server_2019
#PB_OS_Windows_Server_2022
#PB_OS_Windows_11
#PB_OS_Windows_Server_2025
#PB_OS_Windows_Future ; New Windows version (not existing when the program was written)
```

Linux

```
#PB_OS_Linux_2_2
#PB_OS_Linux_2_4
#PB_OS_Linux_2_6
#PB_OS_Linux_Future ; New Linux version (not existing when the program was written)
```

Mac OSX

```
#PB_OS_MacOSX_10_0
#PB_OS_MacOSX_10_1
#PB_OS_MacOSX_10_2
#PB_OS_MacOSX_10_3
#PB_OS_MacOSX_10_4
#PB_OS_MacOSX_10_5
#PB_OS_MacOSX_10_6
#PB_OS_MacOSX_10_7
#PB_OS_MacOSX_10_8
#PB_OS_MacOSX_10_9
#PB_OS_MacOSX_10_10
#PB_OS_MacOSX_10_11
#PB_OS_MacOSX_10_12
#PB_OS_MacOSX_10_13
#PB_OS_MacOSX_10_14
#PB_OS_MacOSX_10_15
#PB_OS_MacOSX_11
#PB_OS_MacOSX_12
#PB_OS_MacOSX_13
#PB_OS_MacOSX_14
#PB_OS_MacOSX_15
#PB_OS_MacOSX_Future ; New MacOS X version (not existing
                      when the program was written)
```

Example

```
1  Select OSVersion()
2      Case #PB_OS_Windows_98
3          MessageRequester("Info", "Windows 98")
4
5      Case #PB_OS_Windows_2000
6          MessageRequester("Info", "Windows 2000")
7
8      Case #PB_OS_Windows_XP
9          MessageRequester("Info", "Windows XP")
10
11     Default
12         MessageRequester("Info", "Unsupported Windows version")
13 EndSelect
```

Note: the value of these constants are established before the release date of each version, therefore tests such as the one which follows, may be used, in order to catch all versions which are older or newer than a given one:

```
1  If OSVersion() < #PB_OS_Windows_2000
2      ;
3      ; All versions older than Windows 2000
4      ;
5  EndIf
```

171.7 ComputerName

Syntax

```
Result\$ = ComputerName()
```

Description

Returns the computer name.

Parameters

None.

Return value

Returns the computer name.

Example

```
1  Debug "Computer name: "+ComputerName()
```

171.8 UserName

Syntax

```
Result\$ = UserName()
```

Description

Returns the currently logged user name.

Parameters

None.

Return value

Returns the currently logged user name.

Example

```
1  Debug "Currently logged user: "+UserName()
```

171.9 MemoryStatus

Syntax

```
Result.q = MemoryStatus(Type)
```

Description

Returns the specified memory type information.

Parameters

Type The type of memory to get. It can be one of the following value:

```
#PB_System_TotalPhysical: the total amount of installed
memory, in bytes
#PB_System_FreePhysical : the available memory, in bytes
#PB_System_TotalVirtual : the total virtual memory size, in
bytes (Windows only)
#PB_System_FreeVirtual  : the available virtual memory size,
in bytes (Windows only)
#PB_System_TotalSwap    : the total swap memory size, in
bytes (Windows and Linux only)
#PB_System_FreeSwap     : the available swap memory size, in
bytes (Windows and Linux only)
#PB_System_PageSize     : the memory page size, in bytes
(usually 4KB).
```

Return value

Returns the specified memory type information.

171.10 CountCPUs

Syntax

```
Result = CountCPUs([Type])
```

Description

Returns the number of CPU cores available.

Parameters

Type (optional) The CPU type to request. It can be one of the following value:

```
#PB_System_CPUs          : the total number of CPU cores
installed in the computer (default).
#PB_System_ProcessCPUs   : the available CPU cores for the
current process. This is useful as the OS
                           can limit a process to use only a
small number of CPU.
```

Return value

Returns the number of CPU cores available.

Chapter 172

Terrain

Overview

Terrains are outdoor 3D scenes which simulate a realistic natural environment based on pre-calculated 2D maps. They are useful in many cases, for example 3D representations of land, simulation games and more. `InitEngine3D()` must be called successfully before using the terrain functions. Multiple terrains are supported, with multiple tiles for each terrain, to create really huge landscapes.

172.1 FreeTerrain

Syntax

```
FreeTerrain(#Terrain)
```

Description

Frees a terrain and releases all its associated memory. This terrain must not be used (by using its number with the other functions in this library) after calling this function, unless you create it again.

Parameters

#Terrain The terrain to free. If **#PB_All** is specified, all the remaining terrains are freed.

Return value

None.

Remarks

All remaining terrains are automatically freed when the program ends.

See Also

`CreateTerrain()`

172.2 FreeTerrainBody

Syntax

```
FreeTerrainBody(#Terrain)
```

Description

Free the body associated with the terrain.

Parameters

#Terrain The terrain to use.

Return value

None.

See Also

CreateTerrainBody()

172.3 SetupTerrains

Syntax

```
SetupTerrains(LightID, CompositeMapDistance.f, Flags)
```

Description

Setup the default parameters for all the future created terrains.

Parameters

LightID The light to use for the terrain rendering. To get a valid light id, use LightID() .

CompositeMapDistance The distance at which to start using a composite map if present, in world unit.

Flags Can be a combination of the following values:

```
#PB_Terrain_Lightmap:    enable the light map for the
                        terrains (a texture with static shadows).
#PB_Terrain_NormalMapping: enable the normal mapping for the
                        terrains. This is a special texture
                        which simulate relief like small
                        rocks, etc.
```

Return value

None.

See Also

CreateTerrain()

172.4 CreateTerrain

Syntax

```
Result = CreateTerrain(#Terrain, Size, WorldSize, Scale, NbLayers,  
    Filename$, Extension$)
```

Description

Creates a new terrain. SetupTerrains() has to be called before to set the default parameter for the new terrain. After the terrain creation, new tiles can be defined with DefineTerrainTile() and textures applied with AddTerrainTexture() . Once the terrain definition is finished, BuildTerrain() needs to be called to build the final terrain.

Parameters

#Terrain The number to identify the new terrain. #PB_Any can be used to auto-generate this number.

Size The size of the new terrain down one edge in vertices.

WorldSize The world size of the new terrain.

Scale The scale factor to apply to the new terrain.

NbLayers The number of texture layers the new terrain will have. To add a texture layer, use AddTerrainTexture() .

Filename\$ The filename (without the extension) to store the precomputed terrain data. As the terrain can be complex and take a lot of time to create, a cache will be written to disk and reloaded automatically if present (and not outdated).

Extension\$ The extension used by the data files.

Return value

Returns zero if the #Terrain can't be created. If #PB_Any is used as '#Terrain' parameter, the new terrain number is returned.

See Also

FreeTerrain() , SetupTerrains() , BuildTerrain() , DefineTerrainTile() , AddTerrainTexture()

172.5 CreateTerrainBody

Syntax

```
CreateTerrainBody(#Terrain, Restitution, Friction)
```

Description

Adds a static physic body to the terrain. This enable physic objects to collide with the terrain.

Parameters

#Terrain The terrain to use.

Restitution The restitution factor. If set to 0.0, the terrain will not restitute any force, which means than the colliding object will not bounce on the terrain. If sets to a value above 0.0, it will restitute some force so the colliding object will bounce when colliding to the terrain (the higher is the value, the more the bouncing will be).

Friction The friction factor. If set to 0.0, the colliding object will slide on the terrain without losing force. If sets to a value above 0.0, the colliding object will loose speed when colliding to the terrain (the higher is the value, the more the speed will decrease).

Return value

None.

See Also

CreateTerrain() , FreeTerrainBody()

172.6 DefineTerrainTile

Syntax

```
Result = DefineTerrainTile(#Terrain, TileX, TileY, HeightMap$,  
                           FlipX, FlipY)
```

Description

Defines the content of a tile in the terrain grid.

Parameters

#Terrain The terrain to use.

TileX The tile x index, relative to the center tile of the terrain. The center tile starts to 0,0. The value can be negative.

TileY The tile y index, relative to the center tile of the terrain. The center tile starts to 0,0. The value can be negative.

HeightMap\$ The name of the height map image to apply to this tile.

FlipX If sets to **#True**, the image will be flipped on horizontally. If sets to **#False**, no horizontal flip will be performed.

FlipY If sets to **#True**, the image will be flipped on vertically. If sets to **#False**, no vertical flip will be performed.

Return value

Returns zero if the **#Terrain** can't be created.

See Also

CreateTerrain()

172.7 AddTerrainTexture

Syntax

```
AddTerrainTexture(#Terrain, Layer, WorldSize, DiffuseSpecular$,  
                  NormalHeight$)
```

Description

Adds a texture to the #Terrain.

Parameters

#Terrain The terrain to use.

Layer The layer index to apply the texture. The first layer index starts from 0. The maximum number of layers is defined when creating the terrain, see CreateTerrain() .

WorldSize The world size of the applied texture.

DiffuseSpecular\$ The name of the diffuse specular map image to apply to this layer.

NormalHeight\$ The name of the normal height map image to apply to this layer.

Return value

None.

See Also

CreateTerrain()

172.8 BuildTerrain

Syntax

```
BuildTerrain(#Terrain)
```

Description

Builds the terrain. Before building a terrain, tiles have to be defined with DefineTerrainTile() , and textures added with AddTerrainTexture() .

Parameters

#Terrain The terrain to build.

Return value

None.

See Also

CreateTerrain() , DefineTerrainTile() , AddTerrainTexture()

172.9 TerrainLocate

Syntax

```
TerrainLocate(#Terrain, x, y, z)
```

Description

Changes the terrain absolute location in the world.

Parameters

#Terrain The terrain to use.

x, y, z The new absolute location in the world (in world unit).

Return value

None.

See Also

CreateTerrain()

172.10 TerrainHeight

Syntax

```
Result = TerrainHeight(#Terrain, x, z)
```

Description

Gets the terrain height at the specified position in the world.

Parameters

#Terrain The terrain to use.

x, z The position in the world (in world unit) to get the terrain height.

Return value

Returns the terrain height (in world unit) at the specified position. If the terrain is not found at the specified position, the result will be zero.

See Also

CreateTerrain()

172.11 TerrainTileHeightAtPosition

Syntax

```
Result = TerrainTileHeightAtPosition(#Terrain, TileX, TileY, Layer,  
    x, y)
```

Description

Returns the height of the terrain tile at the specified coordinates.

Parameters

#Terrain The terrain to use.

TileX The tile 'x' index.

TileY The tile 'y' index.

Layer The layer index. The first layer index starts from 0. The maximum number of layers is defined when creating the terrain, see CreateTerrain() .

x, y The position in the tile (in pixels) to get the terrain height.

Return value

Returns the terrain tile height (in world unit) at the specified coordinates.

See Also

CreateTerrain()

172.12 TerrainTilePointX

Syntax

```
Result = TerrainTilePointX(#Terrain, TileX, TileY, x, y, z)
```

Description

Returns the 'x' position of the point in the terrain tile.

Parameters

#Terrain The terrain to use.

TileX The tile 'x' index.

TileY The tile 'y' index.

x, y, z The position of the point in the world (in world unit).

Return value

Returns 'x' position (in pixels) of the point in the terrain tile.

See Also

CreateTerrain() , DefineTerrainTile()

172.13 TerrainTilePointY

Syntax

```
Result = TerrainTilePointY(#Terrain, TileX, TileY, x, y, z)
```

Description

Returns the 'y' position of the point in the terrain tile.

Parameters

#Terrain The terrain to use.

TileX The tile 'x' index.

TileY The tile 'y' index.

x, y, z The position of the point in the world (in world unit).

Return value

Returns 'y' position (in pixels) of the point in the terrain tile.

See Also

CreateTerrain() , DefineTerrainTile()

172.14 TerrainTileSize

Syntax

```
Result = TerrainTileSize(#Terrain, TileX, TileY)
```

Description

Returns the size of the terrain tile.

Parameters

#Terrain The terrain to use.

TileX The tile 'x' index.

TileY The tile 'y' index.

Return value

Returns the size (in pixels) of the terrain tile. As a tile is always square, the size is the width and the height of the tile.

See Also

CreateTerrain() , DefineTerrainTile()

172.15 GetTerrainTileHeightAtPoint

Syntax

```
Result = GetTerrainTileHeightAtPoint(#Terrain, TileX, TileY, x, y)
```

Description

Returns the height of the terrain tile at the specified position.

Parameters

#Terrain The terrain to use.

TileX The tile 'x' index.

TileY The tile 'y' index.

x, y The position in the tile (in pixels) to get the terrain height.

Return value

Returns the terrain tile height (in world unit) at the specified coordinates.

See Also

CreateTerrain() , SetTerrainTileHeightAtPoint()

172.16 SetTerrainTileHeightAtPoint

Syntax

```
SetTerrainTileHeightAtPoint(#Terrain, TileX, TileY, x, y, Height)
```

Description

Sets the height of the terrain tile at the specified position. The change will not be reflected immediately, UpdateTerrain() has to be called once all the modifications are done.

Parameters

#Terrain The terrain to use.

TileX The tile 'x' index.

TileY The tile 'y' index.

x, y The position in the tile (in pixels) to set the terrain height.

Height The new terrain height (in world unit) at the specified position in the tile.

Return value

None.

See Also

CreateTerrain() , GetTerrainTileHeightAtPoint() , UpdateTerrain()

172.17 UpdateTerrain

Syntax

```
UpdateTerrain(#Terrain)
```

Description

Updates the terrain. This is needs after altering the terrain with commands like SetTerrainTileHeightAtPoint() .

Parameters

#Terrain The terrain to update.

Return value

None.

See Also

CreateTerrain() , SetTerrainTileHeightAtPoint()

172.18 TerrainTileLayerMapSize

Syntax

```
Result = TerrainTileLayerMapSize(#Terrain, TileX, TileY)
```

Description

Returns the terrain tile layer blend map size.

Parameters

#Terrain The terrain to use.

TileX The tile 'x' index.

TileY The tile 'y' index.

Return value

Returns the size (in pixels) of the terrain blend map. As a blend map layer is always square, the size is the width and the height of the blend map layer.

See Also

CreateTerrain() , DefineTerrainTile()

172.19 GetTerrainTileLayerBlend

Syntax

```
Result = GetTerrainTileLayerBlend(#Terrain, TileX, TileY, Layer, x, y)
```

Description

Returns the terrain tile layer blend value at the specified position.

Parameters

#Terrain The terrain to use.

TileX The tile 'x' index.

TileY The tile 'y' index.

Layer The layer index. The first layer index starts from 0. The maximum number of layers is defined when creating the terrain, see CreateTerrain() .

x, y The position in the tile (in pixels) to get the blend value.

Return value

Returns the terrain tile layer blend value at the specified position. The blend value ranges from 0.0 (fully transparent) to 1.0 (fully opaque).

See Also

CreateTerrain()

172.20 SetTerrainTileLayerBlend

Syntax

```
SetTerrainTileLayerBlend(#Terrain, TileX, TileY, Layer, x, y, Value)
```

Description

Changes the terrain tile layer blend value at the specified position. The change will not be reflected immediately, UpdateTerrainTileLayerBlend() has to be called once all the modifications are done.

Parameters

#Terrain The terrain to use.

TileX The tile 'x' index.

TileY The tile 'y' index.

Layer The layer index. The first layer index starts from 0. The maximum number of layers is defined when creating the terrain, see CreateTerrain() .

x, y The position in the tile (in pixels) to set the blend value.

Value The new terrain tile layer blend value to set at the specified position. The blend value ranges from 0.0 (fully transparent) to 1.0 (fully opaque).

Return value

None.

See Also

CreateTerrain()

172.21 UpdateTerrainTileLayerBlend

Syntax

```
UpdateTerrainTileLayerBlend(#Terrain, TileX, TileY, Layer)
```

Description

Updates the terrain tile blend layer. This is needed after modifying the layer blend value with SetTerrainTileLayerBlend() .

Parameters

#Terrain The terrain to use.

TileX The tile 'x' index.

TileY The tile 'y' index.

Layer The layer index to update. The first layer index starts from 0. The maximum number of layers is defined when creating the terrain, see CreateTerrain() .

Return value

None.

See Also

CreateTerrain() , SetTerrainTileLayerBlend()

172.22 TerrainMousePick

Syntax

```
Result = TerrainMousePick(#Terrain, CameraID, x, y)
```

Description

Simulates a mouse click on the terrain under the specified 2D point (x,y - in pixels) on the specified camera.

Parameters

#Terrain The terrain to use.

CameraID The camera to use. To get a valid camera id, use CameraID() .

x, y The position (in pixels) in the camera view of the point.

Return value

If the terrain has been hit by the simulated mouse click, it will return `#True`, else it will return `#False`. To get information about the pick position, use `PickX()` , `PickY()` and `PickZ()` .

See Also

`CreateTerrain()` , `CreateCamera()`

172.23 SaveTerrain

Syntax

```
SaveTerrain(#Terrain, ModifiedOnly)
```

Description

Saves the terrain to disk, using the filename and extension defined with `CreateTerrain()` .

Parameters

#Terrain The terrain to save.

ModifiedOnly If sets to `#True`, the terrain will be saved only if it has been modified since the last save. If sets to `#False`, it will always be saved.

Return value

None.

See Also

`CreateTerrain()`

172.24 TerrainRenderMode

Syntax

```
TerrainRenderMode(#Terrain, Flag)
```

Description

Changes the way the terrain is rendered.

Parameters

#Terrain The terrain to use.

Flag It can be one of the following values:

```
#PB_Terrain_CastShadows: enables dynamic shadow casting on  
the terrain (can be slow).  
#PB_Terrain_LowLODShadows: enables low quality shadow casting  
(to have a faster rendering).
```

Return value

None.

See Also

CreateTerrain()

Chapter 173

Text3D

Overview

Text3D are meant to ease the need to display text in the 3D world. It can use any font, and be moved and scaled according to the object to track.

InitEngine3D() must be called successfully before using these functions.

173.1 CreateText3D

Syntax

```
Result = CreateText3D(#Text3D, Caption$ [, Font$, Height, Color])
```

Description

Creates a new 3D text. To be displayed, the text needs to be attached to a LibraryLink "node" "node" or an entity .

Parameters

#Text3D The number to identify the new 3D text. #PB_Any can be used to auto-generate this number.

Caption\$ The text caption to display. It can be changed with Text3DCaption() .

Font\$ (optional) The font name to use. The font must be present in the Add3DArchive() path and must be defined as it is in the 'proper-definitions.fontdef' file.

Height (optional) The font height to use.

Color (optional) The RGBA color to use. To get a valid color, use RGBA() . The color can be changed with Text3DColor() .

Return value

Returns nonzero if the text was created successfully and zero if there was an error. If #PB_Any was used as the #Text3D parameter then the new generated number is returned on success.

See Also

FreeText3D()

173.2 FreeText3D

Syntax

```
FreeText3D(#Text3D)
```

Description

Free the specified text.

Parameters

#Text3D The text to free. If **#PB_All** is specified, all the remaining texts are freed.

Return value

None.

Remarks

Once the text is freed, it may not be used anymore.
All remaining texts are automatically freed when the program ends.

See Also

CreateText3D()

173.3 Text3DID

Syntax

```
Text3DID = Text3DID(#Text3D)
```

Description

Returns the unique system identifier of the text.

Parameters

#Text3D The text to use.

Return value

Returns the unique system identifier of the text.

173.4 IsText3D

Syntax

```
Result = IsText3D(#Text3D)
```

Description

Tests if the given text number is a valid and correctly initialized text.

Parameters

#Text3D The text to test.

Return value

Returns nonzero if #Text3D is a valid text, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a text is ready to use.

See Also

CreateText3D()

173.5 MoveText3D

Syntax

```
MoveText3D(#Text3D, x, y, z [, Mode])
```

Description

Move the specified text.

Parameters

#Text3D The text to use.

x, y, z The new position of the text.

Mode (optional) The move mode. It can be one of the following values:

```
#PB_Relative: relative move, from the current text position  
(default).  
#PB_Absolute: absolute move to the specified position.
```

combined with one of the following values:

```
#PB_Local : local move.  
#PB_Parent: move relative to the parent position.  
#PB_World : move relative to the world.
```

Return value

None.

See Also

ScaleText3D()

173.6 ScaleText3D

Syntax

```
ScaleText3D(#Text3D, x, y, z [, Mode])
```

Description

Scales the text according to the specified x,y,z values. When using **#PB_Relative** mode, this is a factor based scale which means the text size will be multiplied with the given value to obtain the new size.

Parameters

#Text3D The text to use.

x, y, z The scale vector.

Mode (optional) The scale mode. It can be one of the following values:

```
#PB_Relative: relative scale, based on the previous size  
                (default). Using 1.0 for scale value will let this value  
                unchanged.  
#PB_Absolute: absolute scale, in world unit.
```

Return value

None.

See Also

MoveText3D()

173.7 Text3DCaption

Syntax

```
Text3DCaption(#Text3D, Caption$)
```

Description

Change the displayed text.

Parameters

#Text3D The text to use.

Caption\$ The new text caption to display.

Return value

None.

See Also

CreateText3D()

173.8 Text3DColor

Syntax

```
Text3DColor(#Text3D, Color)
```

Description

Change displayed text color.

Parameters

#Text3D The text to use.

Color The new text color to display. To get a valid color, use RGBA() .

Return value

None.

See Also

CreateText3D()

173.9 Text3DAlignment

Syntax

```
Text3DAlignment(#Text3D, Alignment)
```

Description

Change displayed text alignment.

Parameters

#Text3D The text to use.

Alignment The new text alignment. It can be a combination of the following values:

```
#PB_Text3D_Left    : the text will be left aligned
#PB_Text3D_Top     : the text will be top aligned
#PB_Text3D_Bottom  : the text will be bottom aligned
#PB_Text3D_HorizontallyCentered: the text will be centered
                    horizontally
#PB_Text3D_VerticallyCentered  : the text will be centered
                    vertically
```

Return value

None.

See Also

CreateText3D()

173.10 Text3DX

Syntax

```
Result = Text3DX(#Text3D)
```

Description

Returns the absolute 'x' position of the text in the world.

Parameters

#Text3D The text to use.

Return value

Returns the absolute 'x' position of the text.

See Also

Text3DY() , Text3DZ() , MoveText3D()

173.11 Text3DY

Syntax

```
Result = Text3DY(#Text3D)
```

Description

Returns the absolute 'y' position of the text in the world.

Parameters

#Text3D The text to use.

Return value

Returns the absolute 'y' position of the text.

See Also

Text3DX() , Text3DZ() , MoveText3D()

173.12 Text3DZ

Syntax

```
Result = Text3DZ(#Text3D)
```

Description

Returns the absolute 'z' position of the text in the world.

Parameters

#Text3D The text to use.

Return value

Returns the absolute 'z' position of the text.

See Also

Text3DX() , Text3DY() , MoveText3D()

Chapter 174

Texture

Overview

Textures are useful to have realistic looking meshes . It's possible to create a texture on fly (with regular 2DDrawing functions) or load it from disk. InitEngine3D() must be called successfully before using the Texture functions.

174.1 CopyTexture

Syntax

```
Result = CopyTexture(#Texture , #NewTexture)
```

Description

Creates a new texture which is the exact copy of the specified texture.

Parameters

#Texture The texture to copy.

#NewTexture A number to identify the new texture. **#PB_Any** can be used to auto-generate this number.

Return value

Nonzero if the texture was successfully duplicated, zero otherwise. If **#PB_Any** was used for the **#NewTexture** parameter then the generated number is returned on success.

174.2 CreateTexture

Syntax

```
Result = CreateTexture(#Texture , Width , Height [, TextureName$])
```

Description

Creates a new blank texture with the specified dimension.

Parameters

#Texture A number to identify the new texture. **#PB_Any** can be used to auto-generate this number.

Width, Height The size of the new texture, in pixels. The width and height of the texture should be each preferably be powers of 2, and if possible, make them square because this will look best on the most hardware. Examples: 64x64, 128x128, 256x256, but also 128x64, 16x32... Old GFX cards can have limitation, so if possible limit the texture size to 256x256.

TextureName\$ (optional) The name of the new texture in the OGRE system. This allow to use this name in script to assign a shader to this texture

Return value

Nonzero if the texture was successfully created, zero otherwise. If **#PB_Any** was used for the **#Texture** parameter then the generated number is returned on success.

Example

```
1 CreateTexture(0, 256, 256) ; Creates a new 256x256 texture
```

174.3 CreateCubicTexture

Syntax

```
Result = CreateCubicTexture(#Texture, #Texture1, #Texture2,  
    #Texture3, #Texture4, #Texture5, #Texture6)
```

Description

Creates a new cubic texture using the specified textures. Cubic textures are useful to create world like reflections. **#PB_Material_EnvironmentMap** should be specified with **SetMaterialAttribute()** to enable cubic reflection.

Parameters

#Texture A number to identify the new texture. **#PB_Any** can be used to auto-generate this number.

#Texture1 The texture to use for the first face of the cubic texture.

#Texture2 The texture to use for the second face of the cubic texture.

#Texture3 The texture to use for the third face of the cubic texture.

#Texture4 The texture to use for the fourth face of the cubic texture.

#Texture5 The texture to use for the fifth face of the cubic texture.

#Texture6 The texture to use for the sixth face of the cubic texture.

Return value

Nonzero if the cubic texture was successfully created, zero otherwise. If **#PB_Any** was used for the **#Texture** parameter then the generated number is returned on success.

Example

```
1  InitEngine3D() : InitSprite() : InitKeyboard()
2
3  OpenWindow(0, 0,0, 800, 600, "VertexColour - [Esc] quit",
   #PB_Window_ScreenCentered)
4  OpenWindowedScreen(WindowID(0), 0, 0, WindowWidth(0),
   WindowHeight(0), 0, 0, 0)
5
6  Add3DArchive(#PB_Compiler_Home + "Examples/3D/Data/Textures",
   #PB_3DArchive_FileSystem)
7  Add3DArchive(#PB_Compiler_Home +
   "Examples/3D/Data/Packs/desert.zip", #PB_3DArchive_Zip)
8  Parse3DScripts()
9
10 CreateCamera(0, 0, 0, 100, 100)
11 MoveCamera(0,0,0,-8)
12 CameraLookAt(0,0,0,0)
13
14 CreateLight(0,$ffffff, -100, 100, 50)
15 AmbientColor($111111*2)
16 CameraBackColor(0,$880044)
17
18 SkyBox("desert07.jpg")
19
20 LoadTexture(0,"desert07_RT.jpg")
21 LoadTexture(1,"desert07_LF.jpg")
22 LoadTexture(2,"desert07_UP.jpg")
23 LoadTexture(3,"desert07_DN.jpg")
24 LoadTexture(4,"desert07_FR.jpg")
25 LoadTexture(5,"desert07_BK.jpg")
26 CreateCubicTexture(10, 0, 1, 2, 3, 4, 5)
27
28 LoadTexture(11, "dirt.jpg")
29 CreateMaterial(0, TextureID(11))
30 AddMaterialLayer(0, TextureID(10), #PB_Material_Add)
31 SetMaterialAttribute(0, #PB_Material_EnvironmentMap,
   #PB_Material_ReflectionMap, 1)
32
33 CreateTorus(0,2,1,32,32)
34 CreateEntity(0, MeshID(0), MaterialID(0))
35
36 Repeat
37   While WindowEvent():Wend
38   ExamineKeyboard()
39   RotateEntity(0,1,1,1, #PB_Relative)
40   RenderWorld()
41   FlipBuffers()
42 Until KeyboardReleased(#PB_Key_Escape)
```

174.4 CreateRenderTexture

Syntax

```
Result = CreateRenderTexture(#Texture, CameraID, Width, Height [,
   Flags [, RenderTextureName$]])
```

Description

Creates a new render texture. The camera associated to the texture will render its view directly on the texture, without being displayed on screen. This can be very useful to have objects which display a part of the scene (like a TV screen, a mirror etc).

Parameters

#Texture A number to identify the new texture. **#PB_Any** can be used to auto-generate this number.

CameraID The ID of the camera to associate with the texture. This ID can be get with `CameraID()` .

Width The width of the new texture (in pixels).

Height The height of the new texture (in pixels).

Flags (optional) The flags can be one of the following value:

```
#PB_Texture_AutomaticUpdate: the texture is updated at every
RenderWorld()
automatically (default)
#PB_Texture_ManualUpdate:    the texture is not updated
automatically, UpdateRenderTexture()
has to be called manually.
#PB_Texture_CameraViewPort: the camera viewport won't be
removed, useful to still be able to do a capture from the
camera.
```

RenderTextureName\$ (optional) The name of the new texture in the OGRE system. This allow to use this name in script to assign a shader to this texture.

Return value

Returns zero if the texture can't be created. If **#PB_Any** is used as '**#Texture**' parameter, the new texture number is returned.

Remarks

`TextureOutput()` is not supported on rendered textures.

The Automatic update only happen when `RenderWorld()` is called. If another texture has the same name, the texture creation will fail.

See Also

`UpdateRenderTexture()`

174.5 UpdateRenderTexture

Syntax

```
UpdateRenderTexture(#Texture)
```

Description

Updates the texture content with the current camera view. If the render texture has been created with the **#PB_Texture_AutomaticUpdate** flag, this function is not needed.

Parameters

#Texture The texture to update.

Return value

None.

See Also

CreateRenderTexture()

174.6 SaveRenderTexture

Syntax

```
Result = SaveRenderTexture(#Texture, Filename$)
```

Description

Save the render texture content. It can be useful to do screenshots of a particular scene. The save format can only be PNG.

Parameters

#Texture The texture to use.

Filename\$ The filename where the texture will be saved. This can be an absolute or relative (to the current directory) path.

Return value

If the texture can't be saved (permission error, disk full etc), the returned value will be zero.

See Also

CreateRenderTexture()

174.7 CreateCubeMapTexture

Syntax

```
Result = CreateCubeMapTexture(#Texture, Width, Height, TextureName$  
[, BackColor])
```

Description

Creates a new cube map texture. A cube map texture use the surrounding to render itself as reflection on it. This texture has to exist in an OGRE script.

Parameters

#Texture The number to identify the new texture. **#PB_Any** can be used to auto-generate this number.

Width The width of the new texture (in pixels).

Height The height of the new texture (in pixels).

TextureName\$ The name of the texture in the OGRE scripts. This texture has to be defined in the script with the correct value to have a working cube mapping. If 'TextureName\$' is set to 'CubeMapTexture', a material script should contain the following definition (which can be adapted depending on your needs):

```
material CubeMapMaterial
{
    technique
    {
        pass
        {
            texture_unit
            {
                cubic_texture CubeMapTexture combinedUVW
                tex_address_mode clamp
                env_map cubic_reflection
            }
        }
    }
}
```

BackColor (optional) The back color of the cube mapping scene, if no SkyBox() is used. To get a valid color, use RGB() .

Return value

Returns zero if the texture can't be created. If **#PB_Any** is used as '**#Texture**' parameter, the new texture number is returned.

See Also

EntityCubeMapTexture()

174.8 EntityCubeMapTexture

Syntax

```
Result = EntityCubeMapTexture(#Texture , #Entity)
```

Description

Applies the cube map texture to the entity . The entity will reflect the world around it.

Parameters

#Texture The texture to use. The texture must have been created with CreateCubeMapTexture() .

#Entity The entity to apply the texture. One texture can be applied to many entities.

Return value

None.

See Also

CreateCubeMapTexture()

174.9 FreeTexture

Syntax

```
FreeTexture(#Texture)
```

Description

Frees the specified texture. All its associated memory is released and this object can't be used anymore.

Parameters

#Texture The texture to free. If **#PB_All** is specified, all the remaining textures are freed.

Return value

None.

Remarks

All remaining textures are automatically freed when the program ends.

174.10 IsTexture

Syntax

```
Result = IsTexture(#Texture)
```

Description

Tests if the given texture is valid and correctly initialized.

Parameters

#Texture The texture to test.

Return value

Nonzero if the texture is valid, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a texture is ready to use.

174.11 GetScriptTexture

Syntax

```
Result = GetScriptTexture(#Texture, Name$)
```

Description

Get a texture defined in an OGRE script file. Scripts are loaded and parsed when calling Parse3DScripts() .

Parameters

#Texture A number to identify the new texture. #PB_Any can be used to auto-generate this number.

Name\$ The name of the texture in the script files.

Return value

Nonzero if the texture was successfully created, zero otherwise. If #PB_Any was used for the #Texture parameter then the generated number is returned on success.

174.12 LoadTexture

Syntax

```
Result = LoadTexture(#Texture, Filename$)
```

Description

Loads a new texture from the disk. Before loading a texture, an archive has to be specified with Add3DArchive() .

Texture format can be in PNG, TGA or JPG. It's strongly recommended to make the texture dimension square and with power of 2 width/height: 64x64, 128x128, 256x256... Old GFX cards can have limitation, so if possible limit the texture size to 256x256.

Parameters

#Texture A number to identify the new texture. #PB_Any can be used to auto-generate this number.

Filename\$ The filename of the texture.

Return value

Nonzero if the texture was successfully loaded, zero otherwise. If #PB_Any was used for the #Texture parameter then the generated number is returned on success.

See Also

FreeTexture()

174.13 TextureID

Syntax

```
TextureID = TextureID(#Texture)
```

Description

Returns the unique system identifier of the texture.

Parameters

#Texture The texture to use.

Return value

Returns the unique system identifier of the texture.

174.14 TextureHeight

Syntax

```
Height = TextureHeight(#Texture)
```

Description

Returns the height of the specified texture.

Parameters

#Texture The texture to use.

Return value

The height of the texture, in pixels.

See Also

TextureWidth()

174.15 TextureOutput

Syntax

```
OutputID = TextureOutput(#Texture)
```

Description

Returns the OutputID of the image to perform 2D rendering operation on it. Textures created with CreateRenderTexture() are not supported.

Parameters

#Texture The texture to draw on.

Return value

Returns the output ID or zero if drawing is not possible. This value should be passed directly to the `StartDrawing()` function to start the drawing operation. The return-value is valid only for one drawing operation and cannot be reused.

Example

```
1  ...
2  StartDrawing(TextureOutput(#Texture))
3      ; do some drawing stuff here...
4  StopDrawing()
```

See Also

`StartDrawing()` , `CreateRenderTexture()`

174.16 TextureWidth

Syntax

```
Width = TextureWidth(#Texture)
```

Description

Returns the width of the specified texture.

Parameters

#Texture The texture to use.

Return value

The width of the texture, in pixels.

See Also

`TextureHeight()`

Chapter 175

Thread

Overview

A thread is a part of a program which runs asynchronously, in the background of this program. This means it's possible to perform long operations (compression, image processing, etc) without halting the whole program and let the user continue to do other things. A thread runs within your program, it's not another process. When the main program exits, all the threads are destroyed. Under PureBasic, threads are simply a procedure which is called asynchronously. The thread runs until the procedure exits.

Examples of places of programs where threads are useful are when you need to be able to handle multiple situations with different response times or which occur at different intervals. In the above paragraph, the response times of image processing and the user interface are quite different (you would want to wait for the image to be processed but always have the user interface to respond).

PureBasic has a special compiler setting to create thread-safe executables. (/THREAD command-line switch or "create thread-safe executable" in the IDE compiler options). Without this mode, certain functions (and also the string access) are faster, but not safe to use in threads. It is still possible to create threads without this mode but it is not recommended, as even something simple like a local string access can be dangerous and needs to be protected. Enabling this option makes these things safe inside threads, but comes at the price of some speed. The decision on whether or not to use threads should therefore be done with care, and the threadmode should only be used when there is a real need for it.

Note: Threads need to be used carefully because it is possible that you can have multiple access to shared resources (memory, variables, files, etc) and you need to manually ensure that you do not run into trouble because of this. The Mutex functions in this library can be used to synchronize access to such shared resources.

Using the Threaded keyword it's possible to create thread-based persistent objects (variables, arrays, lists, maps).

Note: Don't use DirectX inside threads (MS Windows limitation)! If you need to display graphics in threads use Images and 2DDrawing instead.

175.1 IsThread

Syntax

```
Result = IsThread(Thread)
```

Description

Tests if the given thread number is a valid thread created with the CreateThread() function, and if it is still running.

Parameters

Thread The thread to use.

Return value

Nonzero if the thread is still valid and running, zero otherwise.

175.2 ThreadID

Syntax

```
ThreadID = ThreadID(Thread)
```

Description

Returns the unique system identifier of the thread.

Parameters

Thread The thread to use.

Return value

The system identifier. This result is sometimes also known as 'Handle'. Look at the extra chapter Handles and Numbers for more information.

175.3 CreateMutex

Syntax

```
Mutex = CreateMutex()
```

Description

Creates a new mutex object. The mutex is initially unlocked.

The main objective of the mutex functions is thread synchronization. They do not create too much overhead, but they only work within one program, not system-wide. A mutex is an object that can only be "owned" or locked by one thread at a time, so it is used to protect shared resources. Only the thread that has the mutex locked may access a certain file, memory area, ...

See LockMutex() and UnlockMutex() for locking/unlocking a mutex.

Parameters

None.

Return value

The new mutex object, or zero if the mutex creation has failed.

Example

```
1 ; Run this code once as it is. You will see that the lines
   printed are
2 ; mixed from between the threads. Now uncomment the Mutex
   functions and
3 ; the strings are printed in order, because only one thread at a
   time
```

```

4      ; has the right to execute the printing functions.
5      ;
6      Procedure WithoutMutex(*Number)
7          Shared Mutex
8
9          For a = 1 To 5
10             ;LockMutex(Mutex)      ; uncomment this to see the difference
11
12             PrintN("Thread "+Str(*Number)+": Trying to print 5x in a
row:")
13             For b = 1 To 5
14                 Delay(50)
15                 PrintN("Thread "+Str(*Number)+" Line "+Str(b))
16             Next b
17
18             ;UnlockMutex(Mutex) ; uncomment this to see the difference
19         Next a
20     EndProcedure
21
22     OpenConsole()
23     Mutex = CreateMutex()
24
25     thread1 = CreateThread(@WithoutMutex(), 1)
26     Delay(25)
27     thread2 = CreateThread(@WithoutMutex(), 2)
28     Delay(25)
29     thread3 = CreateThread(@WithoutMutex(), 3)
30
31     WaitThread(thread1)
32     WaitThread(thread2)
33     WaitThread(thread3)
34
35     Input()

```

175.4 CreateThread

Syntax

```
Thread = CreateThread(@ProcedureName(), *Value)
```

Description

Creates a new thread running in the application background. If the thread is correctly created, it returns the Thread number which is used with the other thread functions, such as KillThread() , PauseThread() , etc. The procedure which you use as a thread must take one parameter and cannot return anything. The '*Value' argument of CreateThread() is passed as the parameter to the procedure. If you do try to return a value from your thread it will simply be lost.

Parameters

@ProcedureName() The address of the procedure you want to use as the code for the new thread. Remember to put the @ in front to get the name and the () afterwards so it gets the address of the procedure.

***Value** The value passed to the thread procedure as parameter. It is up to you to decide what this is used for.

Return value

The number for the newly created thread, or zero if a new thread could not be created. This number is required if you want to control the thread using the other functions in this library.

Example

The example below shows the basic way to create a thread, although in this case it does not do anything.

```
1  Procedure YourProcedure(*Value)
2      ; The variable '*Value' will contain 23
3  EndProcedure
4
5  CreateThread(@YourProcedure(), 23)
```

Example: Passing multiple parameters to a thread

```
1  Structure Person
2      Name$
3      Age.b
4      Phone.l
5  EndStructure
6
7  Procedure Thread(*Parameters.Person)
8
9      ; Display the parameters
10     ;
11     Debug *Parameters\Name$
12     Debug *Parameters\Age
13     Debug *Parameters\Phone
14
15     ; Once we don't need them anymore, use ClearStructure() to
16     ; ensure dynamic
17     ; objects (if any) are correctly cleared, and release the
18     ; dynamic memory block
19     ClearStructure(*Parameters, Person)
20     FreeMemory(*Parameters)
21
22 EndProcedure
23
24 ; We use a dynamically allocated block, so even if we call it
25 ; from a procedure, it will
26 ; still work. The memory block will be freed by the thread, when
27 ; the thread ends.
28 ;
29 *Parameters.Person = AllocateMemory(SizeOf(Person))
30 *Parameters\Name$ = "John"
31 *Parameters\Age = 30
32 *Parameters\Phone = 10203040
33
34 CreateThread(@Thread(), *Parameters) ; Send the thread a pointer
35 ; to our structure
36
37 Delay(2000)
```

175.5 FreeMutex

Syntax

```
FreeMutex(Mutex)
```

Description

Frees a mutex object and the memory it requires.

Parameters

Mutex The mutex to free.

Return value

None.

Remarks

The mutex object should be unlocked by the time it is freed and it may no longer be used after it was freed. To ensure this a mutex should only be freed after all threads that use it have either ended, or are otherwise sure not to use this mutex again.

See Also

CreateMutex()

175.6 KillThread

Syntax

```
KillThread(Thread)
```

Description

Immediately kills the specified thread, which had previously been created with CreateThread() . This is a very dangerous function, and should only be used rarely. The problem is that the thread is killed immediately and has no chance to perform any cleanup code (for example, freeing memory, releasing items, de-allocating its own stack).

If possible, a flag like a global variable should be used to tell the thread to quit itself (which does the needed cleanup) and this function should only be used if this is not possible for some reason.

Parameters

Thread The thread to kill. This value is returned by CreateThread() .

Return value

None.

Example

```
1 ; A procedure/thread which will never exit. Not good, but it
2 ; shows how KillThread works
3 Procedure PrintStuff(*Interval)
4     Repeat
5         PrintN(".")
6         Delay(*Interval)
7     ForEver
8 EndProcedure
9
10 If OpenConsole()
11     thread = CreateThread(@PrintStuff(), 500)
12     If thread
13         For i=0 To 10
14             PrintN("A")
15             Delay(999)
16
17             If i=5
18                 KillThread(thread)
19             EndIf
20         Next
21     EndIf
22 EndIf
```

175.7 LockMutex

Syntax

`LockMutex(Mutex)`

Description

Waits until the mutex object is available (not locked by another thread) and then locks the object so no other thread can get a lock on the object.

After this function returns, it is assured that this thread is the only one with a locked state on the mutex. The thread can now freely access the shared resource that is protected by this mutex, as it is the only one with exclusive access to the mutex.

If another thread calls `LockMutex()` while this one has the lock, it will wait inside the `LockMutex()` function until this thread calls `UnlockMutex()` to release its lock on the mutex.

Note: Since the `LockMutex()` function waits until the mutex is available, it can easily lead to lockup situations if a `UnlockMutex()` call is forgotten.

See `CreateMutex()` for a code example.

Parameters

Mutex The mutex to lock.

Return value

None.

See Also

`UnlockMutex()` , `CreateMutex()`

175.8 PauseThread

Syntax

`PauseThread(Thread)`

Description

Pauses the execution of the specified thread, previously created with `CreateThread()` . The thread can be resumed with `ResumeThread()` .

Parameters

Thread The thread to pause. This value is returned by `CreateThread()` .

Return value

None.

Example

```
1  Procedure PrintStuff(*Dummy)
2      For i = 0 To 10
3          PrintN(" ")
4          Delay(200)
5      Next
6  EndProcedure
7
8  If OpenConsole()
9      thread = CreateThread(@PrintStuff(), 0)
10     If thread
11         Delay(100)
12         PauseThread(thread)
13         For i = 0 To 10
14             PrintN("A")
15             Delay(50)
16         Next
17
18         ; Resume thread and give it enough time to complete
19         ResumeThread(thread)
20         Delay(3000)
21     EndIf
22 EndIf
```

See Also

`ResumeThread()` , `CreateThread()`

175.9 ResumeThread

Syntax

`ResumeThread(Thread)`

Description

Resumes execution of the specified thread, previously paused with `PauseThread()` .

Parameters

Thread The thread to resume. This value is returned by `CreateThread()` .

Return value

None.

Remarks

See the `PauseThread()` function for an example code.

See Also

`PauseThread()` , `CreateThread()`

175.10 ThreadPriority

Syntax

```
OldPriority = ThreadPriority(Thread, Priority)
```

Description

Change the priority of the specified thread and returns the old priority.

The priority value can go from 1 to 32. 1 is the lowest priority available, 16 is the normal priority and 32 is the time critical priority (highest, please don't use it unless you know what you're doing). If the priority is 0, then the priority isn't changed (useful to only retrieve the thread priority without change it). All threads are created with a normal priority.

Typically you would give a thread which is always running (for example, an image processing thread) a priority which is not greater than any other thread in your system. The reason for this is that if it is always running, and it has a high priority, then no other thread will get a chance to run. If a thread needs to be highly responsive but spends most of its time waiting for some event you might consider giving it a higher than normal priority.

Windows schedules threads (chooses which one to run for a short length of time) using a preemptive priority based scheduling strategy, with round robin scheduling within the same priority level. This means that if it is time for execution of a different thread (known as a context switch) then the thread with the highest priority that is available to run will be the next one to be executed. If there is more than one thread with the highest priority, and they are available to run, then each thread will be cycled through on successive context switches.

Parameters

Thread The thread to change the priority of. This value is returned by `CreateThread()` .

Priority The new priority to assign to the thread. The priority can be zero (meaning not to change the priority) or range from 1 (lowest priority) to 32 (highest priority). 16 is normal priority. Windows doesn't support 32 different level of priority, here is the corresponding table:

- 1: lowest
- between 2 and 15: below normal
- 16: normal
- between 17 and 30: above normal
- 31: highest
- 32: time critical

Return value

The priority of the thread before this function was called. This can be useful if you only want to boost the priority of the thread for a short time and then return it to its previous level. The returned value is not necessarily the same than the one set with `ThreadPriority()`, as it depends of the granularity of the priority tuning offered by the system.

Example

```
1  ; Procedure which always runs (note, no Delay function as
2  ; this would cause the thread to stop running while it was
3  ; being delayed
4  Procedure PrintStuff(*Interval)
5      For i = 0 To 1000000000
6          ; Nasty busy wait
7          Next
8  EndProcedure
9
10 If OpenConsole()
11     thread = CreateThread(@PrintStuff(), 500)
12     If thread
13         ; Increase the priority above the main thread
14         ; You should notice a delay before the print function
15         ; is executed. Now change the 17 to 15 (lower than normal
16         priority)
17         ; and see that the print executes immediately
18         ThreadPriority(thread, 17)
19         PrintN("Waits for higher priority thread to finish")
20     EndIf
21     PrintN("Press return to exit")
22     Input()
23 EndIf
```

Supported OS

Windows

175.11 TryLockMutex

Syntax

```
Result = TryLockMutex(Mutex)
```

Description

Tries to lock the specified mutex. Unlike `LockMutex()`, this function does not stop execution until the mutex is available. It returns immediately and the return-value indicates if the lock was successful or not. This is useful in situations where the thread should not wait for the mutex to be available but rather do other things in the meantime.

Parameters

Mutex The mutex to lock.

Return value

Nonzero if the mutex was successfully locked, zero otherwise.

Remarks

If the lock was successful, the `UnlockMutex()` function must be called to make the mutex available to other threads again. If this is not done, this could easily lead to a lockup situation.

Example

```
1  Procedure ThreadProcedure(*Value)
2      Shared Mutex
3
4      Repeat
5          If TryLockMutex(Mutex)
6              PrintN("Mutex successfully locked.")
7
8              UnlockMutex(Mutex)
9              Break ; quit the loop and thread
10         Else
11             PrintN("Still waiting for mutex access...")
12             Delay(200)
13         EndIf
14     ForEver
15 EndProcedure
16
17 OpenConsole()
18
19 Mutex = CreateMutex()
20 LockMutex(Mutex) ; main program has the mutex locked at first
21 Thread = CreateThread(@ThreadProcedure(), 0)
22
23 Delay(4000)
24 UnlockMutex(Mutex) ; now release the mutex, so the thread can get
   it
25
26 Input()
```

See Also

`UnlockMutex()`

175.12 UnlockMutex

Syntax

```
UnlockMutex(Mutex)
```

Description

Unlocks a mutex previously locked by `LockMutex()` . The mutex is then available again for other threads to lock it.

Parameters

Mutex The mutex to unlock.

Return value

None.

Remarks

A mutex can only be unlocked by the thread that also locked it.
See `CreateMutex()` for a code example.

See Also

`LockMutex()` , `CreateMutex()`

175.13 WaitThread

Syntax

```
Result = WaitThread(Thread [, Timeout])
```

Description

Stop the program execution until the specified 'Thread' exits, or the optional timeout (in milliseconds) is reached. If the thread is already finished, it returns immediately.

Parameters

Thread The thread to wait for. This value is returned by `CreateThread()` .

Timeout (optional) Timeout to wait, in milliseconds.

Return value

Nonzero if the thread has ended, or zero if the timeout was reached.

Example

```
1  Procedure PrintStuff(*Interval)
2      For i = 0 To 10
3          PrintN(".")
4          Delay(*Interval)
5      Next
6  EndProcedure
7
8  If OpenConsole()
9      thread = CreateThread(@PrintStuff(), 500)
10     If thread
11         ; Wait for thread to finish before we continue
12         ; Try commenting the WaitThread function out and seeing what
           happens
13         WaitThread(thread)
14     
```

```

15         For i = 0 To 10
16             PrintN("A")
17             Delay(1000)
18         Next
19     EndIf
20 EndIf

```

175.14 CreateSemaphore

Syntax

Semaphore = CreateSemaphore([InitialCount])

Description

Creates a new semaphore object.

A semaphore is a thread synchronization object that keeps an internal count. It has two kinds of operations: signal and wait . A wait operation decreases the count of the semaphore by one. If the count would drop below 0, the wait operation will block until a signal call is made. A signal operation increases the count one, releasing a blocking thread if there is one. A semaphore allows to enforce minimum or maximum counts across threads for example to prevent a queue from running out or getting too many items.

Unlike a mutex , a semaphore object is not "owned" by a particular thread, which means that signal/wait calls do not need to be done from the same thread as it is the case with LockMutex() and UnlockMutex() . In fact a common use for a semaphore objects is for one thread to do the SignalSemaphore() calls and for another one to do all WaitSemaphore() calls to implement a producer/consumer pattern.

Parameters

InitialCount (optional) It has be a positive value that specifies the initial count of the semaphore. If it is not specified, the initial count is zero.

Return value

The new semaphore, or zero if the semaphore creation has failed.

Example

This example shows a "producer" thread populating a queue with elements and the main thread reading them. The semaphore is used to make sure the queue never runs out of elements. Note that this could also be achieved with only a mutex and waiting/polling for queue elements with a Delay() in between, but the semaphore commands do a more efficient wait (returning immediately at the SignalSemaphore() call and not just on the next time a polling loop would check the queue).

```

1 Global Semaphore = CreateSemaphore()
2 Global Mutex      = CreateMutex()
3 Global NewList Queue()
4
5 Procedure Producer(Total)
6
7     For i = 1 To Total
8         Delay(Random(750) + 250)
9

```

```

10      ; The queue access still needs a normal mutex lock to be
      thread-safe
11      LockMutex(Mutex)
12      LastElement(Queue())
13      AddElement(Queue())
14      Queue() = i
15      UnlockMutex(Mutex)
16
17      ; Signal that there is a new queue element
18      SignalSemaphore(Semaphore)
19  Next i
20
21 EndProcedure
22
23 If CreateThread(@Producer(), 30)
24
25     For i = 1 To 30
26         ; wait for one element to be available
27         WaitSemaphore(Semaphore)
28
29         ; display the queue state
30         LockMutex(Mutex)
31         Queue$ = "Queue:"
32         ForEach Queue()
33             Queue$ + " " + Str(Queue())
34         Next Queue()
35         Debug Queue$
36
37         ; remove head element from the queue
38         FirstElement(Queue())
39         DeleteElement(Queue())
40         UnlockMutex(Mutex)
41
42     Next i
43
44 EndIf

```

See Also

FreeSemaphore()

175.15 FreeSemaphore

Syntax

```
FreeSemaphore(Semaphore)
```

Description

Destroys the given Semaphore object and frees all resources used by it.

Parameters

Semaphore The semaphore to free.

Return value

None.

See Also

CreateSemaphore()

175.16 SignalSemaphore

Syntax

```
SignalSemaphore (Semaphore)
```

Description

Increases the internal count of the semaphore by one, releasing a waiting thread if there is one.

Parameters

Semaphore The semaphore to signal.

Return value

None.

Remarks

The semaphore count is limited to a signed 32-bit value , so a maximum of 2147483647
SignalSemaphore() calls can be made without being balanced by WaitSemaphore() calls in between.
See CreateSemaphore() for a code example.

175.17 WaitSemaphore

Syntax

```
WaitSemaphore (Semaphore)
```

Description

Decreases the internal count of the semaphore by one, blocking thread execution if the count would fall below zero. A blocked thread is resumed as soon as another thread calls SignalSemaphore() .

Parameters

Semaphore The semaphore to wait for.

Return value

None.

Remarks

The semaphore count is limited to a signed 32-bit value , so a maximum of 2147483647
WaitSemaphore() calls can be made without being balanced by SignalSemaphore() calls in between.
See CreateSemaphore() for a code example.

175.18 TrySemaphore

Syntax

```
Result = TrySemaphore(Semaphore)
```

Description

Decreases the internal count of the semaphore by one only if the count is above 0. This is the same as a WaitSemaphore() operation, but without blocking if the count would fall below 0.

Parameters

Semaphore The semaphore to use.

Return value

Nonzero if the semaphore count was decreased, or zero if the count could not be decreased because it was already zero.

Chapter 176

ToolBar

Overview

Toolbars are very useful to access some functions of the application quickly, with the help of small icons. It's often the shortcuts of menus items. PureBasic allows to create any number of toolbar and to handle them as if it was a menu.

176.1 CreateToolBar

Syntax

```
Result = CreateToolBar(#ToolBar, WindowID [, Flags])
```

Description

Creates a new empty toolbar on the given window.

Parameters

#ToolBar A number to identify the new toolbar. #PB_Any can be used to auto-generate this number.

WindowID The window for the new toolbar. It can be obtained using the WindowID() function.

Flags (optional) It can be a combination of the following values:

```
#PB_ToolBar_Small      : Small icon (16x16 pixels) toolbar  
                        (default)  
#PB_ToolBar_Large      : Large icon (24x24 pixels) toolbar  
#PB_ToolBar_Text       : Text will be displayed below the  
                        button  
#PB_ToolBar_InlineText : Text will be displayed at the right  
                        of the button (Windows only)
```

Return value

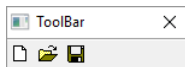
Returns nonzero if the toolbar was created successfully and zero if not. If #PB_Any was used for the #ToolBar parameter then the generated number is returned on success.

Remarks

This toolbar become the default toolbar for creation and it's possible to use `ToolBarImageButton()` and `ToolBarSeparator()` to add some items to this toolbar. The events are handled the same way than menu events, using the function `EventMenu()` . ToolBars are often used as shortcut for menu items, so when assigning the same menu item number to a toolbar button, both events are handled using the same code.

Example

```
1  If OpenWindow(0, 0, 0, 150, 25, "ToolBar", #PB_Window_SystemMenu |
    #PB_Window_ScreenCentered)
2      If CreateToolBar(0, WindowID(0))
3          UsePNGImageDecoder()
4          Path$ = #PB_Compiler_Home + "Examples" + #PS$ + "Sources" +
    #PS$ + "Data" + #PS$ + "ToolBar" + #PS$ + ""
5          ToolBarImageButton(0, LoadImage(0, Path$ + "New.png"))
6          ToolBarImageButton(1, LoadImage(1, Path$ + "Open.png"))
7          ToolBarImageButton(2, LoadImage(2, Path$ + "Save.png"))
8      EndIf
9      Repeat
10         Event = WaitWindowEvent()
11         If Event = #PB_Event_Menu
12             Debug "ToolBar ID: "+Str(EventMenu())
13         EndIf
14     Until Event = #PB_Event_CloseWindow
15 EndIf
```



See Also

`ToolBarImageButton()` , `ToolBarSeparator()` , `FreeToolBar()`

176.2 FreeToolBar

Syntax

```
FreeToolBar(#ToolBar)
```

Description

Free the specified #Toolbar.

Parameters

#ToolBar The toolbar to free. If **#PB_All** is specified, all the remaining toolbar are freed.

Return value

None.

Remarks

All remaining toolbars are automatically freed when the program ends.

See Also

CreateToolBar()

176.3 DisableToolBarButton

Syntax

```
DisableToolBarButton(#ToolBar, Button, State)
```

Description

Disable (or enable) a toolbar button in the given toolbar.

Parameters

#ToolBar The toolbar to use.

Button The toolbar button to disable or enable.

State The new state for the toolbar button. A value of 1 disables the toolbar button and a value of 0 enables it.

Return value

None.

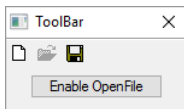
Example

```
1 If OpenWindow(0, 0, 0, 180, 60, "ToolBar", #PB_Window_SystemMenu |
  #PB_Window_ScreenCentered)
2   If CreateToolBar(0, WindowID(0))
3     UsePNGImageDecoder()
4     Path$ = #PB_Compiler_Home + "Examples" + #PS$ + "Sources" +
      #PS$ + "Data" + #PS$ + "ToolBar" + #PS$ + ""
5     ToolBarImageButton(0, LoadImage(0, Path$ + "New.png"))
6     ToolBarImageButton(1, LoadImage(1, Path$ + "Open.png"))
7     ToolBarImageButton(2, LoadImage(2, Path$ + "Save.png"))
8     DisableToolBarButton(0, 1, 1) : Disabled = #True
9   EndIf
10
11   ButtonGadget(0, 10, 30, 130, 20, "Enable OpenFile")
12   ResizeGadget(0, #PB_Ignore, #PB_Ignore, GadgetWidth(0,
      #PB_Gadget_RequiredSize), #PB_Ignore)
13
14   Repeat
15     Event = WaitWindowEvent()
16     If Event = #PB_Event_Gadget
17       If EventGadget() = 0
18         If Disabled = #True
19           DisableToolBarButton(0, 1, 0)
20           SetGadgetText(0, "Disable OpenFile")
```

```

21         ResizeGadget(0, #PB_Ignore, #PB_Ignore, GadgetWidth(0,
22         #PB_Gadget_RequiredSize), #PB_Ignore)
23         Disabled = #False
24     Else
25         DisableToolBarButton(0, 1, 1)
26         SetGadgetText(0, "Enable OpenFile")
27         ResizeGadget(0, #PB_Ignore, #PB_Ignore, GadgetWidth(0,
28         #PB_Gadget_RequiredSize), #PB_Ignore)
29         Disabled = #True
30     EndIf
31 EndIf
32 Until Event = #PB_Event_CloseWindow
EndIf

```



See Also

ToolBarImageButton()

176.4 GetToolBarButtonState

Syntax

```
State = GetToolBarButtonState(#ToolBar, Button)
```

Description

Get the state of the specified toolbar button. The button has to be created using the `#PB_ToolBar_Toggle` mode.

Parameters

#ToolBar The toolbar to use.

Button The toolbar button to get the state.

Return value

Returns nonzero if the toolbar button is toggled (pushed) and zero otherwise.

Remarks

Use SetToolBarButtonState() to change the state of a toolbar button.

See Also

SetToolBarButtonState()

176.5 IsToolBar

Syntax

```
Result = IsToolBar(#ToolBar)
```

Description

Tests if the given #ToolBar number is a valid and correctly initialized, toolbar.

Parameters

#ToolBar The toolbar to use.

Return value

Returns nonzero if #ToolBar is a valid toolbar and zero otherwise.

Remarks

This function is bulletproof and can be used with any value. If the 'Result' is not zero then the object is valid and initialized, otherwise it will equal zero. This is the correct way to ensure a toolbar is ready to use.

See Also

CreateToolBar()

176.6 SetToolBarButtonState

Syntax

```
SetToolBarButtonState(#ToolBar, Button, State)
```

Description

Set the state of the specified toolbar button. The button has to be created using the #PB_ToolBar_Toggle mode.

Parameters

#ToolBar The toolbar to use.

Button The toolbar button to set the state.

State The new state value for the toolbar button. If the state value is nonzero, the toolbar button will be pushed, else it will be unpushed.

Return value

None.

Remarks

Use GetToolBarButtonState() to get the state of a toolbar button.

See Also

GetToolBarButtonState()

176.7 ToolBarHeight

Syntax

```
Result = ToolBarHeight(#ToolBar)
```

Description

Returns the height (in pixels) of the toolbar. This is useful for correct calculation on window height when using a toolbar.

Parameters

#ToolBar The toolbar to use.

Return value

Returns the height (in pixels) of the toolbar.

Remarks

On OS X this command returns 0, as the toolbar is not part of the window inner height so no calculation is needed.

See Also

CreateToolBar()

176.8 ToolBarImageButton

Syntax

```
ToolBarImageButton(#Button, ImageID [, Mode [, Text$]])
```

Description

Add an image button to the toolbar being constructed. CreateToolBar() must be called before to use this function.

Parameters

#Button The new toolbar button identifier.

ImageID The image to use for the button. It can be easily obtained by using ImageID() from the Image library. It can be an image loaded with LoadImage() or created in memory with CreateImage(). To have a real transparent background, use the 'icon' (.ico) file format on Windows, or the PNG file format on Linux/MacOS X.

Mode (optional) The mode value can be one of the following constants:

`#PB_ToolBar_Normal`: the button will act as standard button
(default)
`#PB_ToolBar_Toggle`: the button will act as toggle button

`GetToolBarButtonState()` and `SetToolBarButtonState()` can be used to retrieve or modify a toggle button state.

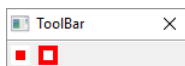
Text\$ (optional) The text to display with this button. The toolbar has to be created with the `#PB_ToolBar_Text` flag, or the text won't be displayed.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 150, 25, "ToolBar", #PB_Window_SystemMenu
2    | #PB_Window_ScreenCentered)
3      CreateImage(0,16,16)
4      StartDrawing(ImageOutput(0))
5      Box(0,0,16,16,RGB(255,255,255))
6      Box(4,4,8,8,RGB(255,0,0))
7      StopDrawing()
8      CreateImage(1,16,16)
9      StartDrawing(ImageOutput(1))
10     Box(0,0,16,16,RGB(255,0,0))
11     Box(4,4,8,8,RGB(255,255,255))
12     StopDrawing()
13     If CreateToolBar(0, WindowID(0))
14         ToolBarImageButton(0,ImageID(0))
15         ToolBarImageButton(1,ImageID(1))
16     EndIf
17     Repeat
18         Until WaitWindowEvent() = #PB_Event_CloseWindow
19     EndIf
```



See Also

`CreateToolBar()` , `ToolBarSeparator()`

176.9 ToolBarSeparator

Syntax

```
ToolBarSeparator()
```

Description

Add a vertical separator to toolbar being constructed. `CreateToolBar()` must be called before to use this function.

Parameters

None.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 150, 25, "ToolBar", #PB_Window_SystemMenu
    | #PB_Window_ScreenCentered)
2      If CreateToolBar(0, WindowID(0))
3          UsePNGImageDecoder()
4          Path$ = #PB_Compiler_Home + "Examples" + #PS$ + "Sources" +
    #PS$ + "Data" + #PS$ + "ToolBar" + #PS$ + ""
5          ToolBarImageButton(0, LoadImage(0, Path$ + "New.png"))
6          ToolBarSeparator()
7          ToolBarImageButton(1, LoadImage(1, Path$ + "Open.png"))
8          ToolBarSeparator()
9          ToolBarImageButton(2, LoadImage(2, Path$ + "Save.png"))
10         ToolBarSeparator()
11         ToolBarSeparator()
12     EndIf
13     Repeat
14         Event = WaitWindowEvent()
15         If Event = #PB_Event_Menu
16             Debug "ToolBar ID: "+Str(EventMenu())
17         EndIf
18     Until Event = #PB_Event_CloseWindow
19 EndIf
```



See Also

CreateToolBar() , ToolBarImageButton()

176.10 ToolBarButtonText

Syntax

```
ToolBarButtonText(#ToolBar, Button, Text$)
```

Description

Change the text for the specified #ToolBar button. The toolbar had to be created with the #PB_ToolBar_Text flag.

Parameters

#ToolBar The toolbar to use.

Button The toolbar button to change the text.

Text\$ The new text to display.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 200, 80, "ToolBar", #PB_Window_SystemMenu
   | #PB_Window_ScreenCentered)
2  If CreateToolBar(0, WindowID(0), #PB_ToolBar_Large |
   #PB_ToolBar_Text)
3      UsePNGImageDecoder()
4      Path$ = #PB_Compiler_Home + "Examples" + #PS$ + "Sources" +
   #PS$ + "Data" + #PS$ + "ToolBar" + #PS$ + ""
5      ToolBarImageButton(0, LoadImage(0, Path$ + "New.png"),
   #PB_ToolBar_Normal, "New")
6      ToolBarImageButton(1, LoadImage(1, Path$ + "Open.png"),
   #PB_ToolBar_Normal, "Open")
7      ToolBarImageButton(2, LoadImage(2, Path$ + "Save.png"),
   #PB_ToolBar_Normal, "Save")
8      ToolBarButtonText(0, 0, "Old !")
9  EndIf
10 Repeat
11     Until WaitWindowEvent() = #PB_Event_CloseWindow
12 EndIf
```

See Also

ToolBarImageButton(), ToolBarSeparator(), CreateToolBar()

176.11 ToolBarToolTip

Syntax

```
ToolBarToolTip(#ToolBar, Button, Text$)
```

Description

Associates the specified text to the #ToolBar button. A tool-tip text is a text which is displayed when the mouse cursor is over the button for a few time (usually a small yellow floating box).

Parameters

#ToolBar The toolbar to use.

Button The toolbar button to set the tooltip.

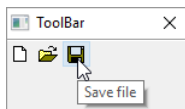
Text\$ The new text to associate with the toolbar button. If the text is empty, the tooltip is removed.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 150, 60, "ToolBar", #PB_Window_SystemMenu
   | #PB_Window_ScreenCentered)
2      If CreateToolBar(0, WindowID(0))
3          UsePNGImageDecoder()
4          Path$ = #PB_Compiler_Home + "Examples" + #PS$ + "Sources" +
#PS$ + "Data" + #PS$ + "ToolBar" + #PS$ + ""
5          ToolBarImageButton(0, LoadImage(0, Path$ + "New.png"))
6          ToolBarImageButton(1, LoadImage(1, Path$ + "Open.png"))
7          ToolBarImageButton(2, LoadImage(2, Path$ + "Save.png"))
8          ToolBarToolTip(0, 0, "New document")
9          ToolBarToolTip(0, 1, "Open file")
10         ToolBarToolTip(0, 2, "Save file")
11     EndIf
12     Repeat
13     Until WaitWindowEvent() = #PB_Event_CloseWindow
14 EndIf
```



See Also

ToolBarImageButton(), ToolBarSeparator()

176.12 ToolBarID

Syntax

```
ToolBarID = ToolBarID(#ToolBar)
```

Description

Returns the unique system identifier of the given toolbar.

Parameters

#ToolBar The toolbar to use.

Return value

Returns the ID of the toolbar. This sometimes also known as 'Handle'. Look at the extra chapter "Handles and Numbers" for more information.

See Also

CreateToolBar()

Chapter 177

VectorDrawing

Overview

The VectorDrawing library provides resolution independent, high-quality drawing operations for display, image manipulation or printing. Unlike the 2DDrawing library, function in this library can operate in a variety of measurement units and allows for arbitrary coordinate transformations. This allows to easily write drawing routines that are independent of the actual output resolution and can easily scale to different sizes. The VectorDrawing library supports alpha transparency in all its operations. Only vector fonts are allowed, like TrueType, bitmap fonts are not allowed. You also cannot use a registered font with RegisterFontFile() .

Drawing sequence

Drawing operations in this library involve three basic steps:

- 1) Construct a path with functions such as AddPathLine() , AddPathCurve() , etc.
- 2) Select a drawing source such as VectorSourceColor()
- 3) stroke , fill , dot or dash the path

After stroking or filling a path, the path is reset and a new path can be constructed for the next drawing operation. The selection of the drawing source (step 2) does not need to be repeated every time, as the drawing source is not reset.

The path based drawing model allows the drawing complex shapes with properties such as thick lines with rounded/diagonal corners and dot/dash patterns without introducing any visible artifacts in the places where segments of the figures meet. Since the entire path is drawn at once, such artifacts can be avoided.

See the AddPathLine() function for a basic example of the drawing steps.

Measurement units

Every drawing output has a default unit of measurement. The default unit is pixels for screen or raster image outputs and points for printer or vector image outputs. It is however possible to select a different unit of measurement for the output when creating it with the ImageVectorOutput() , PrinterVectorOutput() or similar function. All drawing operations will use the selected unit of measurement and internally convert the values to the actual device coordinates. This allows to write the drawing code in the preferred unit of measurement independent of the used output. The selected unit of measurement for an output can be checked with VectorUnit() .

Coordinate transformation

It is possible to move , scale , rotate , flip or skew the coordinate system used for drawing. The transformations can be freely combined. Such transformations affect all drawing operations. Possible uses of coordinate transformations is to draw figures in a rotated or stretched manner without the need to modify the actual drawing code. For example, printing code can easily switch

to landscape printing by simply rotating the coordinates (and therefore all output) at the start of the drawing options.

There are four different coordinate systems and some functions take an optional parameter to select which system should be used. These are the available options:

`#PB_Coordinate_Device`

This coordinate system represents the physical coordinates of the output device. It cannot be transformed. This coordinate system is useful when converting values between the device and the actual drawing coordinate system with `ConvertCoordinateX()` and `ConvertCoordinateY()`.

`#PB_Coordinate_Output`

This coordinate system represents the initial output coordinates in the selected unit of measurement. This coordinate system is equal to `#PB_Coordinate_Device` except for possible scaling by a different measurement unit. This coordinate system cannot be transformed.

`#PB_Coordinate_User`

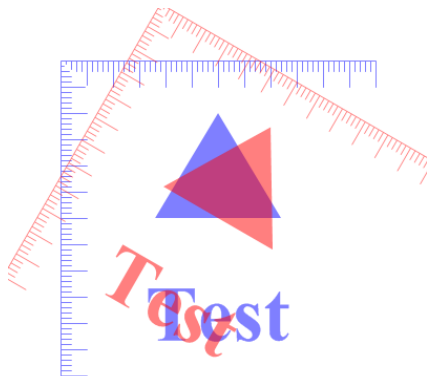
This is the coordinate system used for all drawing operations. This coordinate system is used whenever a different system is not explicitly specified. It can be freely transformed. Initially, this coordinate system is equal to the `#PB_Coordinate_Output` system and can be reset that way with `ResetCoordinates()`.

`#PB_Coordinate_Source`

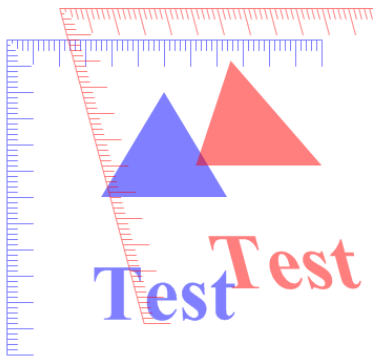
This coordinate system is used by the commands that select the vector drawing source. It is most useful together with the `VectorSourceImage()` command to transform the used source image. This coordinate system is relative to the `#PB_Coordinate_User` system, so any transformation to the `#PB_Coordinate_User` system will affect this system as well.

For most purposes, the `#PB_Coordinate_User` is the interesting coordinate system and is therefore the default. The other systems are useful mainly for coordinate conversion or for special purposes such as transforming the source image.

Example: Rotating the coordinate system



Example: Combining coordinate transformations (translation & skew)



Drawing state and layers

A number of properties of the drawing output such as coordinate transformations, clipping or the drawing source can be saved and later restored with `SaveVectorState()` and `RestoreVectorState()` respectively. This allows to make temporary modifications to the drawing output and later restoring the previous state. The commands work in a stack, so it is possible to save/restore multiple drawing states.

The `BeginVectorLayer()` allows to save the current drawing state, constructs a new virtual drawing layer. Future drawing operations will be directed to that layer. A call to `EndVectorLayer()` will combine the layer with the below drawing output and restore the previous drawing state. This allows to combine a number of drawing operations and then applying them as a layer to the output. Multiple temporary layers can be created this way.

177.1 StartVectorDrawing

Syntax

```
Result = StartVectorDrawing(Output)
```

Description

Prepares the vector drawing library to draw to the specified output.

Parameters

Output The output to draw on. These functions can be used to get an output for vector drawing:

`WindowVectorOutput()` : Drawing will be rendered directly on the Window

`ImageVectorOutput()` : Drawing will be rendered directly on the Image data (see `CreateImage()`)

`PrinterVectorOutput()` : Drawing will be rendered directly on the Printer

`CanvasVectorOutput()` : Drawing will be rendered directly on the CanvasGadget()

`PdfVectorOutput()` : Drawing will be rendered to a PDF file

`SvgVectorOutput()` : Drawing will be rendered to an SVG file

Return value

Returns nonzero if drawing is possible or zero if the operation failed.

Remarks

Drawing must be finished with `StopVectorDrawing()` .

If "Create thread-safe executable" is enabled in the compiler options then every thread has its own current drawing output, which means two threads can do drawing on separate outputs at the same time.

See Also

`StopVectorDrawing()`

177.2 StopVectorDrawing

Syntax

```
StopVectorDrawing()
```

Description

Finishes a sequence of drawing operations and frees all resources allocated by it.

Parameters

None.

Return value

None.

See Also

`StartVectorDrawing()`

177.3 VectorOutputWidth

Syntax

```
Result.d = VectorOutputWidth()
```

Description

Returns the width of the vector drawing output area.

Parameters

None.

Return value

Returns the output width.

Example

See Also

`VectorOutputHeight()` , `VectorUnit()` , `VectorResolutionX()` , `VectorResolutionY()`

177.4 VectorOutputHeight

Syntax

```
Result.d = VectorOutputHeight()
```

Description

Returns the height of the vector drawing output area.

Parameters

None.

Return value

Returns the output height.

Example

See Also

`VectorOutputWidth()` , `VectorUnit()` , `VectorResolutionX()` , `VectorResolutionY()`

177.5 VectorResolutionX

Syntax

```
Result.d = VectorResolutionX()
```

Description

Returns the horizontal resolution of the vector drawing output area.

Parameters

None.

Return value

Returns the horizontal resolution in DPI (dots per inch).

See Also

`VectorResolutionY()`

177.6 VectorResolutionY

Syntax

```
Result.d = VectorResolutionY()
```

Description

Returns the vertical resolution of the vector drawing output area.

Parameters

None.

Return value

Returns the vertical resolution in DPI (dots per inch).

Remarks

The vertical resolution can differ from the horizontal resolution in the case of a printer output.

See Also

VectorResolutionX()

177.7 VectorUnit

Syntax

```
Result = VectorUnit()
```

Description

Returns the unit in which all coordinates and sizes are measured on the current vector drawing output. This unit has been specified when the output was created.

Parameters

None.

Return value

Returns one of the following values:

```
#PB_Unit_Pixel      : Values are measured in pixels (or dots in  
  case of a printer)  
#PB_Unit_Point      : Values are measured in points (1/72 inch)  
#PB_Unit_Inch       : Values are measured in inches  
#PB_Unit_Millimeter : Values are measured in millimeters
```

177.8 SaveVectorState

Syntax

```
SaveVectorState()
```

Description

Saves the current vector drawing state to be restored later. Multiple states can be saved on a stack and restored in the reverse order they were saved.

The following information is saved with this command:

- The coordinate transformations
- The drawing source
- The drawing font
- The clipping path

Note that the current path is not saved by this command.

Parameters

None.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     CanvasGadget(0, 0, 0, 400, 200)
4     LoadFont(0, "Times New Roman", 20, #PB_Font_Bold)
5
6     If StartVectorDrawing(CanvasVectorOutput(0))
7         VectorSourceColor(RGBA(255, 0, 0, 255))
8         VectorFont(FontID(0))
9
10        MovePathCursor(20, 20)
11        DrawVectorText("Normal text")
12
13        ; Changes made to the drawing state within this block do not
14        affect the other commands
15        SaveVectorState()
16        MovePathCursor(120, 160)
17        RotateCoordinates(120, 160, -50)
18        VectorSourceColor(RGBA(0, 0, 255, 255))
19        DrawVectorText("Rotated text")
20        RestoreVectorState()
21
22        MovePathCursor(220, 140)
23        DrawVectorText("Normal text")
24
25        StopVectorDrawing()
26    EndIf
27
28    Repeat
29        Event = WaitWindowEvent()
```



```
29     Until Event = #PB_Event_CloseWindow
30 EndIf
```

See Also

RestoreVectorState() , BeginVectorLayer()

177.9 RestoreVectorState

Syntax

```
RestoreVectorState()
```

Description

Restores the vector drawing state that was stored in the corresponding call to SaveVectorState() .

Parameters

None.

Return value

None.

Example

See SaveVectorState() for an example.

See Also

SaveVectorState()

177.10 BeginVectorLayer

Syntax

```
BeginVectorLayer([Alpha])
```

Description

Begins a new empty layer on top of the current vector drawing output. All future drawing operations will be performed on this layer until EndVectorLayer() is called. This command also saves the current drawing state in the same way as SaveVectorState() . Multiple layers can be created.

Parameters

Alpha (optional) Specifies the alpha transparency of the new vector layer. Allowed values are from 0 (fully transparent) to 255 (fully opaque). The default is 255 (fully opaque).

Return value

None.

Remarks

The resources needed to create the temporary layer depend on the size of the current clipping path . It is therefore recommended to set a clipping path that covers only the area where the temporary layer will contain any drawing to save resources and improve drawing performance.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     CanvasGadget(0, 0, 0, 400, 200)
4
5     If StartVectorDrawing(CanvasVectorOutput(0))
6
7         ; Semi-transparent drawing on the base layer
8         AddPathCircle(75, 100, 60)
9         VectorSourceColor(RGBA(255, 0, 0, 127))
10        FillPath()
11        AddPathCircle(125, 100, 60)
12        VectorSourceColor(RGBA(0, 0, 255, 127))
13        FillPath()
14
15        ; Opaque drawing on a semi-transparent layer
16        BeginVectorLayer(127)
17        AddPathCircle(275, 100, 60)
18        VectorSourceColor(RGBA(255, 0, 0, 255))
19        FillPath()
20        AddPathCircle(325, 100, 60)
21        VectorSourceColor(RGBA(0, 0, 255, 255))
22        FillPath()
23        EndVectorLayer()
24
25        StopVectorDrawing()
26    EndIf
27
28    Repeat
29        Event = WaitWindowEvent()
30    Until Event = #PB_Event_CloseWindow
31 EndIf
```



See Also

EndVectorLayer() , SaveVectorState()

177.11 EndVectorLayer

Syntax

```
EndVectorLayer()
```

Description

Finishes drawing on a temporary layer created by `BeginVectorLayer()` . The contents of the layer are drawn to the next lower layer using the alpha transparency of the temporary layer. This command also restores the drawing state that was in effect when `BeginVectorLayer()` was called.

Parameters

None.

Return value

None.

Example

See `BeginVectorLayer()` for an example.

See Also

`BeginVectorLayer()` , `SaveVectorState()`

177.12 NewVectorPage

Syntax

```
NewVectorPage()
```

Description

Finishes the current page on the vector drawing output and starts a fresh page. The following outputs support multiple pages:

`PrinterVectorOutput()`

`PdfVectorOutput()`

Parameters

None.

Return value

None.

See Also

`NewPrinterPage()`

177.13 FillVectorOutput

Syntax

```
FillVectorOutput()
```

Description

Fills the entire drawing area (except areas outside the clipping path) with the current drawing source. This operation is equivalent to constructing a path that covers the entire drawing area and calling FillPath() on it.

Parameters

None.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",  
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
3      CanvasGadget(0, 0, 0, 400, 200)  
4      If StartVectorDrawing(CanvasVectorOutput(0))  
5  
6          ; make the entire output red  
7          VectorSourceColor(RGBA(255, 0, 0, 255))  
8          FillVectorOutput()  
9  
10         StopVectorDrawing()  
11     EndIf  
12  
13     Repeat  
14         Event = WaitWindowEvent()  
15     Until Event = #PB_Event_CloseWindow  
16 EndIf
```

See Also

FillPath() , ClipPath()

177.14 ResetCoordinates

Syntax

```
ResetCoordinates([System])
```

Description

Reset any coordinate transformations that were applied to the current vector drawing output and restore the coordinate system that was in effect when StartVectorDrawing() was called.

Parameters

System (optional) Specifies the coordinate system to change. This can be one of the following values:

```
#PB_Coordinate_User : Change the coordinate system for
points in the drawing path (default)
#PB_Coordinate_Source: Change the coordinate system for the
vector drawing source
```

Return value

None.

Remarks

See the vectordrawing overview for an introduction to the different coordinate systems.

See Also

TranslateCoordinates() , ScaleCoordinates() , RotateCoordinates() , SkewCoordinates() ,
FlipCoordinatesX() , FlipCoordinatesY() , ConvertCoordinateX() , ConvertCoordinateY()

177.15 TranslateCoordinates

Syntax

```
TranslateCoordinates(x.d, y.d [, System])
```

Description

Move the origin of the vector drawing coordinate system. The move will be applied along the x/y axis of the current coordinate system. All future drawing operations will be relative to the new origin.

Parameters

x.d, y.d Specifies the amount to move the coordinate origin along the x/y axis.

System (optional) Specifies the coordinate system to change. This can be one of the following values:

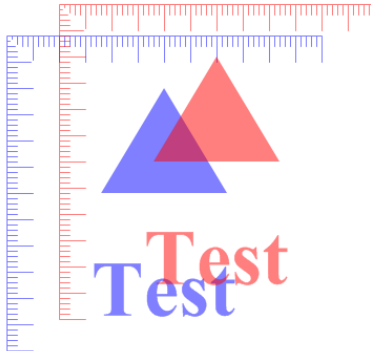
```
#PB_Coordinate_User : Change the coordinate system for
points in the drawing path (default)
#PB_Coordinate_Source: Change the coordinate system for the
vector drawing source
```

Return value

None.

Remarks

See the vectordrawing overview for an introduction to the different coordinate systems. The following image demonstrates the effect of translated coordinates. The same figure is drawn twice, the original is in blue, and the version with translated coordinates is in red.



Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      CanvasGadget(0, 0, 0, 400, 200)
4
5      If StartVectorDrawing(CanvasVectorOutput(0))
6          VectorFont(LoadFont(0, "Times New Roman", 60, #PB_Font_Bold))
7
8          VectorSourceColor(RGBA(0, 0, 255, 128))
9          MovePathCursor(50, 50)
10         DrawVectorText("Test")
11
12         TranslateCoordinates(30, 30) ; all coordinates are moved 30
13         pixels in each direction
14
15         VectorSourceColor(RGBA(255, 0, 0, 128))
16         MovePathCursor(50, 50)
17         DrawVectorText("Test")
18
19         StopVectorDrawing()
20     EndIf
21
22     Repeat
23         Event = WaitWindowEvent()
24     Until Event = #PB_Event_CloseWindow
25 EndIf
```

See Also

ResetCoordinates() , ScaleCoordinates() , RotateCoordinates() , SkewCoordinates() ,
FlipCoordinatesX() , FlipCoordinatesY() , ConvertCoordinateX() , ConvertCoordinateY()

177.16 ScaleCoordinates

Syntax

```
ScaleCoordinates(ScaleX.d, ScaleY.d [, System])
```

Description

Scale the vector drawing coordinate system by stretching it in the x/y direction.

Parameters

ScaleX.d, ScaleY.d The scale factor for each direction. A factor of 1.0 leaves the coordinates unchanged while factors above and below 1.0 stretch the coordinate system. A negative factor mirrors the output coordinates.

System (optional) Specifies the coordinate system to change. This can be one of the following values:

```
#PB_Coordinate_User   : Change the coordinate system for  
                        points in the drawing path (default)  
#PB_Coordinate_Source: Change the coordinate system for the  
                        vector drawing source
```

Return value

None.

Remarks

See the vectordrawing overview for an introduction to the different coordinate systems. The following image demonstrates the effect of scaled coordinates. The same figure is drawn twice, the original is in blue, and the version with scaled coordinates is in red.



Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     CanvasGadget(0, 0, 0, 400, 200)
4
5     If StartVectorDrawing(CanvasVectorOutput(0))
6         VectorFont(LoadFont(0, "Times New Roman", 60, #PB_Font_Bold))
7
8         VectorSourceColor(RGBA(0, 0, 255, 128))
9         MovePathCursor(50, 50)
10        DrawVectorText("Test")
11
12        ScaleCoordinates(0.7, 0.9)
13
14        VectorSourceColor(RGBA(255, 0, 0, 128))
15        MovePathCursor(50, 50)
16        DrawVectorText("Test")
17
18        StopVectorDrawing()
19    EndIf
20
21    Repeat
22        Event = WaitWindowEvent()
23    Until Event = #PB_Event_CloseWindow
24 EndIf
```



See Also

ResetCoordinates(), TranslateCoordinates(), RotateCoordinates(), SkewCoordinates(),
FlipCoordinatesX(), FlipCoordinatesY(), ConvertCoordinateX(), ConvertCoordinateY()

177.17 RotateCoordinates

Syntax

```
RotateCoordinates(x.d, y.d, Angle.d [, System])
```

Description

Rotate the vector drawing coordinate system around the given center point. The center point is expressed in terms of the current coordinate system.

Parameters

x.d, y.d Specifies the center point for the rotation.

Angle.d Specifies the rotation angle in degrees. A positive angle rotates the coordinate system clockwise.

System (optional) Specifies the coordinate system to change. This can be one of the following values:

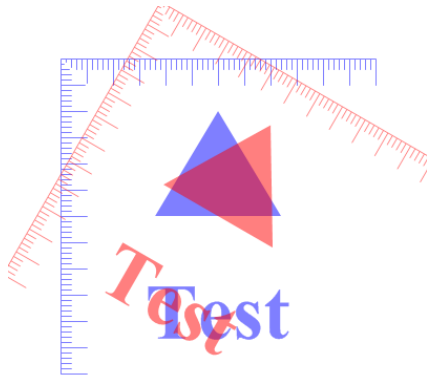
```
#PB_Coordinate_User : Change the coordinate system for
                     points in the drawing path (default)
#PB_Coordinate_Source: Change the coordinate system for the
                     vector drawing source
```

Return value

None.

Remarks

See the vectordrawing overview for an introduction to the different coordinate systems. The following image demonstrates the effect of rotated coordinates. The same figure is drawn twice, the original is in blue, and the version with rotated coordinates is in red.



Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     CanvasGadget(0, 0, 0, 400, 200)
4
5     If StartVectorDrawing(CanvasVectorOutput(0))
6         VectorFont(LoadFont(0, "Times New Roman", 60, #PB_Font_Bold))
7
8         VectorSourceColor(rgba(0, 0, 255, 128))
9         MovePathCursor(50, 50)
10        DrawVectorText("Test")
11
12        RotateCoordinates(50, 50, -20) ; rotate by -20 degrees around
13        the (50, 50) point
14
15        VectorSourceColor(rgba(255, 0, 0, 128))
16        MovePathCursor(50, 50)
17        DrawVectorText("Test")
18
19        StopVectorDrawing()
20    EndIf
21
22    Repeat
23        Event = WaitWindowEvent()
```

```
22     Until Event = #PB_Event_CloseWindow
23 EndIf
```



See Also

ResetCoordinates() , TranslateCoordinates() , ScaleCoordinates() , SkewCoordinates() ,
FlipCoordinatesX() , FlipCoordinatesY() , ConvertCoordinateX() , ConvertCoordinateY()

177.18 SkewCoordinates

Syntax

```
SkewCoordinates(AngleX.d, AngleY.d [, System])
```

Description

Apply a shearing angle in the x and/or y direction to the vector drawing coordinate system.

Parameters

AngleX.d, AngleY.d Specifies the shearing angle in each direction in degrees.

System (optional) Specifies the coordinate system to change. This can be one of the following values:

```
#PB_Coordinate_User   : Change the coordinate system for
                        points in the drawing path (default)
#PB_Coordinate_Source: Change the coordinate system for the
                        vector drawing source
```

Return value

None.

Remarks

See the vectordrawing overview for an introduction to the different coordinate systems.
The following image demonstrates the effect of skewed coordinates. The same figure is drawn twice, the original is in blue, and the version with skewed coordinates is in red.



Example

```

1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      CanvasGadget(0, 0, 0, 400, 200)
4
5      If StartVectorDrawing(CanvasVectorOutput(0))
6          VectorFont(LoadFont(0, "Times New Roman", 60, #PB_Font_Bold))
7
8          VectorSourceColor(RGBA(0, 0, 255, 128))
9          MovePathCursor(50, 50)
10         DrawVectorText("Test")
11
12         SkewCoordinates(45, 0)
13
14         VectorSourceColor(RGBA(255, 0, 0, 128))
15         MovePathCursor(50, 50)
16         DrawVectorText("Test")
17
18         StopVectorDrawing()
19     EndIf
20
21     Repeat
22         Event = WaitWindowEvent()
23     Until Event = #PB_Event_CloseWindow
24 EndIf

```



See Also

ResetCoordinates(), TranslateCoordinates(), ScaleCoordinates(), RotateCoordinates(), FlipCoordinatesX(), FlipCoordinatesY(), ConvertCoordinateX(), ConvertCoordinateY()

177.19 FlipCoordinatesX

Syntax

```
FlipCoordinatesX(AxisX.d [, System])
```

Description

Mirrors the vector drawing coordinate system at the specified X axis.

Parameters

AxisX.d The X coordinate at which the coordinate system should be mirrored.

System (optional) Specifies the coordinate system to change. This can be one of the following values:

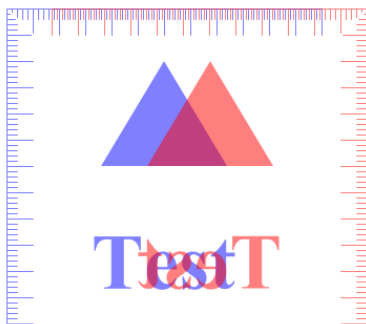
```
#PB_Coordinate_User : Change the coordinate system for  
points in the drawing path (default)  
#PB_Coordinate_Source: Change the coordinate system for the  
vector drawing source
```

Return value

None.

Remarks

See the vectordrawing overview for an introduction to the different coordinate systems. The following image demonstrates the effect of flipped coordinates. The same figure is drawn twice, the original is in blue, and the version with flipped coordinates is in red.



Example

```
1 If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",  
2   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
3   CanvasGadget(0, 0, 0, 400, 200)  
4   If StartVectorDrawing(CanvasVectorOutput(0))  
5     VectorFont(LoadFont(0, "Times New Roman", 60, #PB_Font_Bold))  
6  
7     VectorSourceColor(RGBA(0, 0, 255, 128))  
8     MovePathCursor(50, 50)
```

```

9      DrawVectorText("Test")
10
11     FlipCoordinatesX(200)
12
13     VectorSourceColor(RGBA(255, 0, 0, 128))
14     MovePathCursor(50, 50)
15     DrawVectorText("Test")
16
17     StopVectorDrawing()
18 EndIf
19
20 Repeat
21     Event = WaitWindowEvent()
22 Until Event = #PB_Event_CloseWindow
23 EndIf

```



See Also

ResetCoordinates() , TranslateCoordinates() , ScaleCoordinates() , RotateCoordinates() ,
SkewCoordinates() , FlipCoordinatesY() , ConvertCoordinateX() , ConvertCoordinateY()

177.20 FlipCoordinatesY

Syntax

```
FlipCoordinatesY(AxisY.d [, System])
```

Description

Mirrors the vector drawing coordinate system at the specified Y axis.

Parameters

AxisY.d The Y coordinate at which the coordinate system should be mirrored.

System (optional) Specifies the coordinate system to change. This can be one of the following values:

```

#PB_Coordinate_User   : Change the coordinate system for
                        points in the drawing path (default)
#PB_Coordinate_Source: Change the coordinate system for the
                        vector drawing source

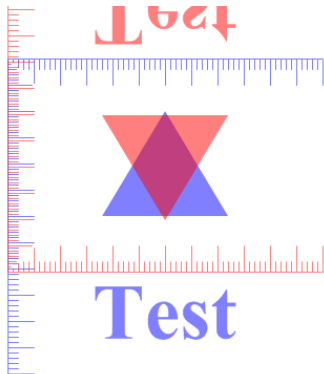
```

Return value

None.

Remarks

See the vectordrawing overview for an introduction to the different coordinate systems. The following image demonstrates the effect of flipped coordinates. The same figure is drawn twice, the original is in blue, and the version with flipped coordinates is in red.



Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      CanvasGadget(0, 0, 0, 400, 200)
4
5      If StartVectorDrawing(CanvasVectorOutput(0))
6          VectorFont(LoadFont(0, "Times New Roman", 60, #PB_Font_Bold))
7
8          VectorSourceColor(RGBA(0, 0, 255, 128))
9          MovePathCursor(50, 50)
10         DrawVectorText("Test")
11
12         FlipCoordinatesY(120)
13
14         VectorSourceColor(RGBA(255, 0, 0, 128))
15         MovePathCursor(50, 50)
16         DrawVectorText("Test")
17
18         StopVectorDrawing()
19     EndIf
20
21     Repeat
22         Event = WaitWindowEvent()
23     Until Event = #PB_Event_CloseWindow
24 EndIf
```



See Also

ResetCoordinates() , TranslateCoordinates() , ScaleCoordinates() , RotateCoordinates() ,
SkewCoordinates() , FlipCoordinatesX() , ConvertCoordinateX() , ConvertCoordinateY()

177.21 ConvertCoordinateX

Syntax

```
Result.d = ConvertCoordinateX(x.d, y.d [, Source, Target])
```

Description

Convert a point from one coordinate system to another in the vector drawing output. This function returns the X coordinate of the conversion. The Y coordinate can be retrieved with the ConvertCoordinateY() function.

Parameters

x.d, y.d Specifies the coordinates of the point to convert in terms of the source coordinate system.

Source, Target (optional) Specifies the source and target coordinates for the conversion. Each can be one of these values:

```
#PB_Coordinate_Device: The coordinate system of the output  
device  
#PB_Coordinate_Output: The coordinate system as it was  
created with the drawing output function  
#PB_Coordinate_User : The coordinate system for points in  
the drawing path  
#PB_Coordinate_Source: The coordinate system for the vector  
drawing source
```

The default conversion is from #PB_Coordinate_User to #PB_Coordinate_Output.

Return value

Returns the X coordinate of the point in the target coordinate system.

Remarks

See the vectordrawing overview for an introduction to the different coordinate systems.

Example

```
1 ; This example draws a dot at the mouse location even in a  
  modified coordinate system  
2 ; by mapping the coordinates from the device system (pixels) to  
  the user system (points)  
3 ;  
4 If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",  
  #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
5   CanvasGadget(0, 0, 0, 400, 200)  
6  
7   Repeat  
8     Event = WaitWindowEvent()  
9
```

```

10      If Event = #PB_Event_Gadget And EventGadget() = 0 And
      EventType() = #PB_EventType_LeftButtonDown
11
12      If StartVectorDrawing(CanvasVectorOutput(0, #PB_Unit_Point))
13          RotateCoordinates(0, 0, 30)
14
15          CanvasX = GetGadgetAttribute(0, #PB_Canvas_MouseX)
16          CanvasY = GetGadgetAttribute(0, #PB_Canvas_MouseY)
17
18          DrawingX = ConvertCoordinateX(CanvasX, CanvasY,
      #PB_Coordinate_Device, #PB_Coordinate_User)
19          DrawingY = ConvertCoordinateY(CanvasX, CanvasY,
      #PB_Coordinate_Device, #PB_Coordinate_User)
20
21          AddPathCircle(DrawingX, DrawingY, 10)
22          VectorSourceColor(RGBA(Random(255), Random(255),
      Random(255), 255))
23          FillPath()
24
25          StopVectorDrawing()
26      EndIf
27
28  EndIf
29
30  Until Event = #PB_Event_CloseWindow
31  EndIf

```

See Also

ResetCoordinates() , TranslateCoordinates() , ScaleCoordinates() , RotateCoordinates() ,
SkewCoordinates() , FlipCoordinatesX() , FlipCoordinatesY() , ConvertCoordinateY()

177.22 ConvertCoordinateY

Syntax

Result.d = ConvertCoordinateY(x.d, y.d [, Source, Target])

Description

Convert a point from one coordinate system to another in the vector drawing output. This function returns the Y coordinate of the conversion. The X coordinate can be retrieved with the ConvertCoordinateX() function.

Parameters

x.d, y.d Specifies the coordinates of the point to convert in terms of the source coordinate system.

Source, Target (optional) Specifies the source and target coordinates for the conversion. Each can be one of these values:

```

#PB_Coordinate_Device: The coordinate system of the output
device
#PB_Coordinate_Output: The coordinate system as it was
created with the drawing output function
#PB_Coordinate_User   : The coordinate system for points in
the drawing path

```


#PB_Coordinate_Source: The coordinate system for the vector drawing source

The default conversion is from #PB_Coordinate_User to #PB_Coordinate_Output.

Return value

Returns the Y coordinate of the point in the target coordinate system.

Remarks

See the vectordrawing overview for an introduction to the different coordinate systems.

Example

See ConvertCoordinateX() for an example.

See Also

ResetCoordinates() , TranslateCoordinates() , ScaleCoordinates() , RotateCoordinates() , SkewCoordinates() , FlipCoordinatesX() , FlipCoordinatesY() , ConvertCoordinateX()

177.23 ResetPath

Syntax

ResetPath()

Description

Resets the vector drawing path to an empty path and moves to cursor to position (0, 0).

Parameters

None.

Return value

None.

See Also

IsPathEmpty()

177.24 ClosePath

Syntax

ClosePath()

Description

Closes the current figure in the vector drawing path by adding a straight line to the starting point of the figure. The starting point is the location of the last MovePathCursor() call. When a path is filled , only closed figures are taken into account.

Parameters

None.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     CanvasGadget(0, 0, 0, 400, 200)
4
5     If StartVectorDrawing(CanvasVectorOutput(0))
6
7         ; Create a path with two closed triangles
8         MovePathCursor(20, 160)
9         AddPathLine(100, 20)
10        AddPathLine(180, 160)
11        ClosePath()
12
13        MovePathCursor(220, 160)
14        AddPathLine(300, 20)
15        AddPathLine(380, 160)
16        ClosePath()
17
18        ; fill the path
19        VectorSourceColor(RGBA(0, 0, 255, 255))
20        FillPath()
21
22        StopVectorDrawing()
23    EndIf
24
25    Repeat
26        Event = WaitWindowEvent()
27    Until Event = #PB_Event_CloseWindow
    EndIf
```



See Also

FillPath() , IsInsidePath() , MovePathCursor() , AddPathLine()

177.25 MovePathCursor

Syntax

```
MovePathCursor(x.d, y.d [, Flags])
```

Description

Moves the cursor of the vector drawing path to a new location. This also starts a new figure within the path, which means that a call to `ClosePath()` will draw a line back to this location.

Parameters

x.d, y.d The new position for the path cursor.

Flags (optional) Can be one of the following values:

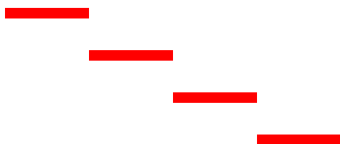
```
#PB_Path_Default : The new position is absolute (default)
#PB_Path_Relative: The new position is relative to the last
                  cursor position.
```

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     CanvasGadget(0, 0, 0, 400, 200)
4
5     If StartVectorDrawing(CanvasVectorOutput(0))
6
7         MovePathCursor(40, 40)
8         For i = 1 To 4
9             AddPathLine(80, 0, #PB_Path_Relative)
10            MovePathCursor(0, 40, #PB_Path_Relative)
11        Next i
12
13        VectorSourceColor(RGBA(255, 0, 0, 255))
14        StrokePath(10)
15
16        StopVectorDrawing()
17    EndIf
18
19    Repeat
20        Event = WaitWindowEvent()
21    Until Event = #PB_Event_CloseWindow
22 EndIf
```



See Also

`ClosePath()` , `AddPathLine()` , `FillPath()` , `StrokePath()`

177.26 AddPathLine

Syntax

```
AddPathLine(x.d, y.d [, Flags])
```

Description

Adds a straight line to the vector drawing path. The line starts at the current cursor position and ends at the given coordinates.

Parameters

x.d, y.d The position for the end of the line. This will become the new position of the path cursor.

Flags (optional) Can be one of the following values:

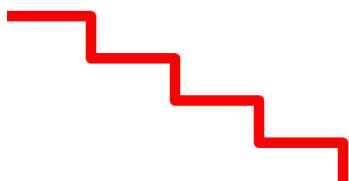
```
#PB_Path_Default : The new position is absolute (default)
#PB_Path_Relative: The new position is relative to the last
                  cursor position.
```

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     CanvasGadget(0, 0, 0, 400, 200)
4
5     If StartVectorDrawing(CanvasVectorOutput(0))
6         MovePathCursor(40, 20)
7         For i = 1 To 4
8             AddPathLine(80, 0, #PB_Path_Relative)
9             AddPathLine(0, 40, #PB_Path_Relative)
10        Next i
11
12        VectorSourceColor(RGBA(255, 0, 0, 255))
13        StrokePath(10, #PB_Path_RoundCorner)
14
15        StopVectorDrawing()
16    EndIf
17
18    Repeat
19        Event = WaitWindowEvent()
20        Until Event = #PB_Event_CloseWindow
21    EndIf
```



See Also

MovePathCursor() , ClosePath() , AddPathArc() , AddPathCurve() , AddPathCircle() , AddPathEllipse() , AddPathBox()

177.27 AddPathArc

Syntax

```
AddPathArc(x1.d, y1.d, x2.d, y2.d, Radius.d, [, Flags])
```

Description

Adds a straight line towards (x1, y2) followed by an arc in the direction of (x2, y2) to the vector drawing path. This function can be used to create paths with rounded corners. The new cursor position will be the endpoint of the arc.

Parameters

x1.d, y1.d The target position for the straight line.

x2.d, y2.d The target position to indicate the direction of the arc.

Radius.d The radius for the rounded corner.

Flags (optional) Can be one of the following values:

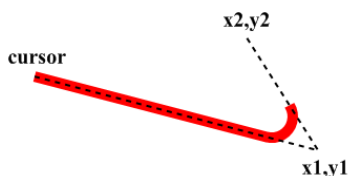
```
#PB_Path_Default : The positions are absolute (default)
#PB_Path_Relative: The positions are relative to the last
                    cursor position.
```

Return value

None.

Remarks

The following image illustrates the meaning of the two reference points and the the segments that are added to the path. Note that no second straight line is added towards the (x2, y2) point by the command. This makes it possible to use AddPathArc() again to add a further rounded corner also at the (x2, y2) position.



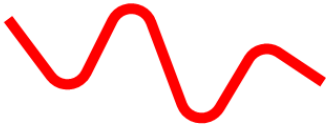
Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     CanvasGadget(0, 0, 0, 400, 200)
4     If StartVectorDrawing(CanvasVectorOutput(0))
```

```

5
6     MovePathCursor(40, 60)
7     AddPathArc(100, 140, 160, 20, 20)
8     AddPathArc(160, 20, 220, 180, 20)
9     AddPathArc(220, 180, 280, 80, 20)
10    AddPathArc(280, 80, 340, 120, 20)
11    AddPathLine(340, 120)
12
13    VectorSourceColor(RGBA(255, 0, 0, 255))
14    StrokePath(10)
15
16    StopVectorDrawing()
17 EndIf
18
19 Repeat
20     Event = WaitWindowEvent()
21 Until Event = #PB_Event_CloseWindow
22 EndIf

```



See Also

MovePathCursor(), AddPathLine(), AddPathCurve(), AddPathCircle(), AddPathEllipse(), AddPathBox()

177.28 AddPathCurve

Syntax

```
AddPathCurve(x1.d, y1.d, x2.d, y2.d, x3.d, y3.d [, Flags])
```

Description

Adds a cubic bezier curve to the vector drawing path. The curve starts at the current path position and ends at (x3, y3). The other two points determine the shape of the curve.

Parameters

x1.d, y1.d The first control point of the curve.

x2.d, y2.d The second control point of the curve.

x3.d, y3.d The endpoint of the curve. This point will become the new path position.

Flags (optional) Can be one of the following values:

```

#PB_Path_Default : The positions are absolute (default)
#PB_Path_Relative: The positions are relative to the last
                  cursor position.

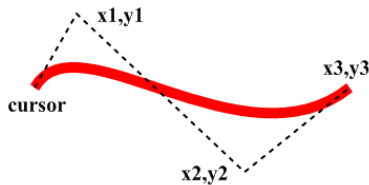
```

Return value

None.

Remarks

The below image shows the position of the reference points. See [here](#) for more information on bezier curves.



Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     CanvasGadget(0, 0, 0, 400, 200)
4
5     If StartVectorDrawing(CanvasVectorOutput(0))
6
7         MovePathCursor(50, 100)
8         AddPathCurve(90, 30, 250, 180, 350, 100)
9         VectorSourceColor(RGBA(255, 0, 0, 255))
10        StrokePath(10)
11
12        StopVectorDrawing()
13    EndIf
14
15    Repeat
16        Event = WaitWindowEvent()
17    Until Event = #PB_Event_CloseWindow
18 EndIf
```

See Also

MovePathCursor() , AddPathLine() , AddPathArc() , AddPathCircle() , AddPathEllipse() , AddPathBox()

177.29 AddPathBox

Syntax

```
AddPathBox(x.d, y.d, Width.d, Height.d [, Flags])
```

Description

Add a box to the vector drawing path. This is a convenience function that combines the needed AddPathLine() calls to create a simple box shape.

By default, this function ends the current figure in the path and adds the box as an unconnected and closed figure to the path (i.e. a box that can be filled). This behavior can be changed with the appropriate flags.

Parameters

x.d, y.d Specifies the origin of the box.

Width.d, Height.d Specifies the width and height of the box.

Flags (optional) This can be a combination of the following values:

```
#PB_Path_Default    : No special behavior (default value)
#PB_Path_Relative   : The positions are relative to the last
                      cursor position.
#PB_Path_Connected  : The box is connected to the existing path
                      with a line and not automatically a closed figure.
```

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     CanvasGadget(0, 0, 0, 400, 200)
4
5     If StartVectorDrawing(CanvasVectorOutput(0))
6         AddPathBox(50, 50, 200, 50)
7         AddPathBox(150, 75, 200, 50)
8         VectorSourceColor(RGBA(255, 0, 0, 255))
9         StrokePath(10)
10
11        StopVectorDrawing()
12    EndIf
13
14    Repeat
15        Event = WaitWindowEvent()
16    Until Event = #PB_Event_CloseWindow
17 EndIf
```



See Also

MovePathCursor() , AddPathLine() , AddPathArc() , AddPathCircle() , AddPathEllipse() , AddPathCurve()

177.30 AddPathCircle

Syntax


```
AddPathCircle(x.d, y.d, Radius.d [, StartAngle.d, EndAngle.d [,
Flags]])
```

Description

Add a circle (or a partial circle) to the vector drawing path.

By default, this function ends the current figure in the path and adds the circle as an unconnected figure to the path (full circles are marked as closed). This behavior can be changed with the appropriate flags.

Parameters

x.d, y.d Specifies the center point for the circle.

Radius.d Specifies the radius for the circle.

StartAngle.d, EndAngle.d (optional) Specifies the angle for start and end of the circle in degrees. The angle 0 marks at the positive X axis. The defaults are 0 and 360 degrees respectively.

Flags (optional) This can be a combination of the following values:

```
#PB_Path_Default          : No special behavior (default
value)
#PB_Path_Relative         : The positions are relative to the
last cursor position.
#PB_Path_Connected        : The circle is connected to the
existing path with a line and not automatically a closed
figure.
#PB_Path_CounterClockwise : The drawing direction between the
start/end angles is counter-clockwise.
```

Return value

None.

Remarks

This function is intended for drawing standalone circles or arcs. To draw figures with rounded corners, the AddPathArc() function can be used which automatically calculates the proper angles and center point to draw rounded corners.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     CanvasGadget(0, 0, 0, 400, 200)
4
5     If StartVectorDrawing(CanvasVectorOutput(0))
6
7         ; partial circle
8         AddPathCircle(100, 100, 75, 0, 235)
9
10        ; partial circle with lines to the center
11        MovePathCursor(300, 100)
12        AddPathCircle(300, 100, 75, 0, 235, #PB_Path_Connected)
13        ClosePath()
```

```

14     VectorSourceColor(RGBA(255, 0, 0, 255))
15     StrokePath(10)
16
17     StopVectorDrawing()
18 EndIf
19
20 Repeat
21     Event = WaitWindowEvent()
22 Until Event = #PB_Event_CloseWindow
23 EndIf

```



See Also

MovePathCursor() , AddPathLine() , AddPathArc() , AddPathBox() , AddPathEllipse() , AddPathCurve()

177.31 AddPathEllipse

Syntax

```

AddPathEllipse(x.d, y.d, RadiusX.d, RadiusY.d [, StartAngle.d,
EndAngle.d [, Flags]])

```

Description

Add an ellipse (or a partial ellipse) to the vector drawing path.

By default, this function ends the current figure in the path and adds the ellipse as an unconnected figure to the path (full ellipses are marked as closed). This behavior can be changed with the appropriate flags.

Parameters

x.d, y.d Specifies the center point for the ellipse.

RadiusX.d, RadiusY.d Specifies the radius for the ellipse in the X and Y direction.

StartAngle.d, EndAngle.d (optional) Specifies the angle for start and end of the circle in degrees. The angle 0 marks at the positive X axis. The defaults are 0 and 360 degrees respectively.

Flags (optional) This can be a combination of the following values:

```

#PB_Path_Default           : No special behavior (default
value)
#PB_Path_Relative          : The positions are relative to the
last cursor position.
#PB_Path_Connected         : The circle is connected to the
existing path with a line and not automatically a closed
figure.
#PB_Path_CounterClockwise : The drawing direction between the
start/end angles is counter-clockwise.

```

Return value

None.

Remarks

This function draws an ellipse shape with a defined radius at the X and Y axis of the current coordinate system. To draw an ellipse at any rotation, rotate the coordinate system around the ellipse's center point before adding the ellipse as shown in the example below. The current coordinate system can be preserved by using `SaveVectorState()` and `RestoreVectorState()` .

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",  
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
3      CanvasGadget(0, 0, 0, 400, 200)  
4  
5      If StartVectorDrawing(CanvasVectorOutput(0))  
6          ; regular ellipse  
7          AddPathEllipse(100, 100, 80, 30)  
8  
9          ; rotated ellipse  
10         SaveVectorState()  
11         RotateCoordinates(300, 100, 45)  
12         AddPathEllipse(300, 100, 80, 30)  
13         RestoreVectorState()  
14  
15         VectorSourceColor(RGBA(255, 0, 0, 255))  
16         StrokePath(10)  
17  
18         StopVectorDrawing()  
19     EndIf  
20  
21     Repeat  
22         Event = WaitWindowEvent()  
23     Until Event = #PB_Event_CloseWindow  
24 EndIf
```



See Also

`MovePathCursor()` , `AddPathLine()` , `AddPathArc()` , `AddPathBox()` , `AddPathCircle()` , `AddPathCurve()`

177.32 AddPathText

Syntax

AddPathText(Text\$)

Description

Add the outline of the characters in the given text to the current cursor position in the vector drawing path. The current position can be set with `MovePathCursor()` . After the call to this function the cursor is moved to the end of the added text.

The `DrawVectorText()` function should be preferred if possible. See the below remarks for details.

Parameters

Text\$ Specifies the text (single-line) to add to the drawing path.

Return value

None.

Remarks

Only vector fonts are allowed, like TrueType, bitmap fonts are not allowed.

Converting text to a path is an expensive operation and may result in a loss of text quality and even a slightly different text form (depending on the font) as compared to directly drawing the text to the output with `DrawVectorText()` . The `DrawVectorText()` function is more efficient and can make use of methods such as sub-pixel rendering to improve the text quality. Therefore, the `AddPathText()` function should only be used if the text is explicitly needed as a path and not for simple text drawing.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",  
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
3      CanvasGadget(0, 0, 0, 400, 200)  
4      LoadFont(0, "Times New Roman", 20, #PB_Font_Bold)  
5  
6      If StartVectorDrawing(CanvasVectorOutput(0))  
7          VectorFont(FontID(0), 150)  
8  
9          MovePathCursor(50, 25)  
10         AddPathText("Text")  
11  
12         VectorSourceColor(RGBA(255, 0, 0, 255))  
13         DashPath(3, 6)  
14  
15         StopVectorDrawing()  
16     EndIf  
17  
18     Repeat  
19         Event = WaitWindowEvent()  
20     Until Event = #PB_Event_CloseWindow  
21 EndIf
```



See Also

`DrawVectorText()` , `DrawVectorParagraph()` , `VectorTextWidth()` , `VectorTextHeight()`

177.33 AddPathSegments

Syntax

```
AddPathSegments(Segments$ [, Flags])
```

Description

Add multiple segments described in string format to the vector drawing path. This command can be used to reproduce the path commands recorded with the `PathSegments()` command.

Parameters

Segments\$ Specifies the path commands to execute.

The segment description consists of one-letter commands followed by the appropriate number of coordinates for the command. Values can be separated by whitespace or comma. Commands in uppercase interpret their arguments as absolute coordinates, the equivalent command in lowercase interprets its arguments as relative the most recent added path segment.

M x y	<code>MovePathCursor()</code>
L x y	<code>AddPathLine()</code>
C x1 y1 x2 y2 x3 y3	<code>AddPathCurve()</code>
Z	<code>ClosePath()</code>

In addition to this simplified segments syntax, the command also accepts path descriptions in the format defined by the [SVG Tiny standard](#) which contains some additional command letters.

Flags (optional) This can be a combination of the following values:

<code>#PB_Path_Default</code>	: No special behavior (default value)
<code>#PB_Path_Relative</code>	: Interpret all coordinates as relative to the current path cursor

Return value

None.

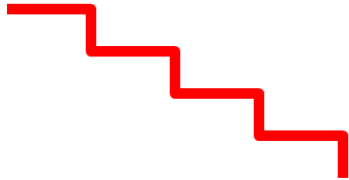
Example

```
1 If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3   CanvasGadget(0, 0, 0, 400, 200)
4
5   If StartVectorDrawing(CanvasVectorOutput(0))
6       AddPathSegments("M 40 20 L 120 20 L 120 60 L 200 60 L 200 100
          L 280 100 L 280 140 L 360 140 L 360 180")
```

```

7      VectorSourceColor(RGBA(255, 0, 0, 255))
8      StrokePath(10, #PB_Path_RoundCorner)
9
10     StopVectorDrawing()
11 EndIf
12
13 Repeat
14     Event = WaitWindowEvent()
15 Until Event = #PB_Event_CloseWindow
16 EndIf

```



See Also

PathSegments()

177.34 IsInsidePath

Syntax

```
Result = IsInsidePath(x.d, y.d [, CoordinateSystem])
```

Description

Tests if the given coordinates are within a closed figure in the current vector drawing path. That is, this function returns non-zero if the given point would be filled by a call to FillPath() .

Parameters

x.d, y.d Specifies the coordinates of the point to test.

CoordinateSystem (optional) Specifies the coordinate system for the point to test. This can be one of the following values:

```

#PB_Coordinate_Device: The coordinate system of the output
                        device
#PB_Coordinate_Output: The coordinate system as it was
                        created with the drawing output function
#PB_Coordinate_User   : The coordinate system for points in
                        the drawing path (default)
#PB_Coordinate_Source: The coordinate system for the vector
                        drawing source

```

Return value

Returns non-zero if the point is within the path and zero if not.

Remarks

See the vectordrawing overview for an introduction to the different coordinate systems.

Example

```
1
2 ; This example uses the IsInsidePath() function to color the
   figure in green
3 ; while the mouse is inside of it and blue otherwise
4 ;
5 Procedure Draw()
6     x = GetGadgetAttribute(0, #PB_Canvas_MouseX)
7     y = GetGadgetAttribute(0, #PB_Canvas_MouseY)
8
9     If StartVectorDrawing(CanvasVectorOutput(0))
10         VectorSourceColor(RGBA(255, 255, 255, 255)) ; erase
previous content
11         FillVectorOutput()
12
13         AddPathEllipse(200, 100, 150, 75) ; prepare path
14
15         If IsInsidePath(x, y, #PB_Coordinate_Device) ; check if the
mouse is inside
16             VectorSourceColor(RGBA(0, 255, 0, 255))
17         Else
18             VectorSourceColor(RGBA(0, 0, 255, 255))
19         EndIf
20
21         FillPath() ; fill path
22         StopVectorDrawing()
23     EndIf
24 EndProcedure
25
26 If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
27     CanvasGadget(0, 0, 0, 400, 200)
28     LoadFont(0, "Times New Roman", 20, #PB_Font_Bold)
29     Draw()
30
31     Repeat
32         Event = WaitWindowEvent()
33
34         If Event = #PB_Event_Gadget And EventGadget() = 0 And
EventType() = #PB_EventType_MouseMove
35             Draw()
36         EndIf
37
38     Until Event = #PB_Event_CloseWindow
39 EndIf
```

See Also

IsInsideStroke() , FillPath() , ClosePath() , ResetPath()

177.35 IsInsideStroke

Syntax

```
Result = IsInsideStroke(x.d, y.d, Width.d [, Flags [,
   CoordinateSystem]])
```

Description

Tests if the given coordinates are within an area that will be drawn to by a call to `StrokePath()` .

Parameters

x.d, y.d Specifies the coordinates of the point to test.

Width.d Specifies the line width to use for the test.

Flags (optional) Possible flags for the line characteristics as described in the `StrokePath()` function.

CoordinateSystem (optional) Specifies the coordinate system for the point to test. This can be one of the following values:

```
#PB_Coordinate_Device: The coordinate system of the output
                        device
#PB_Coordinate_Output: The coordinate system as it was
                        created with the drawing output function
#PB_Coordinate_User   : The coordinate system for points in
                        the drawing path (default)
#PB_Coordinate_Source: The coordinate system for the vector
                        drawing source
```

Remarks

See the vectordrawing overview for an introduction to the different coordinate systems.

Return value

Returns non-zero if the point is within the stroke and zero if not.

Example

```
1  ; This example uses the IsInsideStroke() function to color the
   ; figure in green
2  ; while the mouse on its outline and blue otherwise
3  ;
4  Procedure Draw()
5      x = GetGadgetAttribute(0, #PB_Canvas_MouseX)
6      y = GetGadgetAttribute(0, #PB_Canvas_MouseY)
7
8      If StartVectorDrawing(CanvasVectorOutput(0))
9          VectorSourceColor(RGBA(255, 255, 255, 255)) ; erase
previous content
10         FillVectorOutput()
11
12         AddPathEllipse(200, 100, 150, 75) ; prepare
path
13
14         If IsInsideStroke(x, y, 20, #PB_Path_Default,
#PB_Coordinate_Device) ; check if the mouse is inside
15             VectorSourceColor(RGBA(0, 255, 0, 255))
16         Else
17             VectorSourceColor(RGBA(0, 0, 255, 255))
18         EndIf
19
```



```

20         StrokePath(20)                                ; stroke
    path
21         StopVectorDrawing()
22     EndIf
23 EndProcedure
24
25 If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
26     CanvasGadget(0, 0, 0, 400, 200)
27     LoadFont(0, "Times New Roman", 20, #PB_Font_Bold)
28     Draw()
29
30     Repeat
31         Event = WaitWindowEvent()
32
33         If Event = #PB_Event_Gadget And EventGadget() = 0 And
    EventType() = #PB_EventType_MouseMove
34             Draw()
35         EndIf
36
37     Until Event = #PB_Event_CloseWindow
38 EndIf

```

See Also

IsInsidePath() , StrokePath() , ResetPath()

177.36 IsPathEmpty

Syntax

```
Result = IsPathEmpty()
```

Description

Tests if the current vector drawing path is empty.

Parameters

None.

Return value

Returns non-zero if the path is empty and zero if the path contains any line segments.

See Also

ResetPath() , IsInsidePath() , IsInsideStroke()

177.37 StrokePath

Syntax

```
StrokePath(Width.d [, Flags])
```

Description

Stroke the current drawing path with the current drawing source. This draws the path as a solid line.

By default, the path is reset after calling this function. This can be prevented with the appropriate flags.

Parameters

Width.d Specifies the width for the stroked line.

Flags (optional) Specifies optional characteristics for the drawn stroke. This can be a combination of the following values:

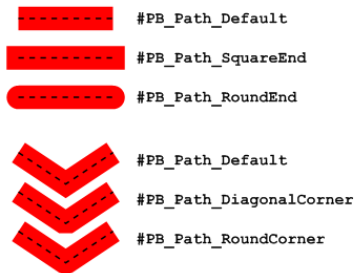
<code>#PB_Path_Default</code>	: No special behavior (default value)
<code>#PB_Path_Preserve</code>	: Don't reset the path after this function
<code>#PB_Path_RoundEnd</code>	: Draw the line(s) with a rounded ends
<code>#PB_Path_SquareEnd</code>	: Draw the line(s) with a square box at the ends
<code>#PB_Path_RoundCorner</code>	: Draw the line(s) with rounded corners
<code>#PB_Path_DiagonalCorner</code>	: Draw the line(s) with diagonally cut corners

Return value

None.

Remarks

The following image demonstrates the effect of the different flags. The Corner and End flags can be combined with the binary or ('||') operator to combine the effects.



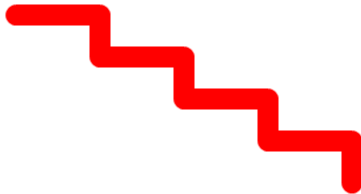
Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      CanvasGadget(0, 0, 0, 400, 200)
4
5      If StartVectorDrawing(CanvasVectorOutput(0))
6
7          MovePathCursor(40, 20)
8          For i = 1 To 4
9              AddPathLine(80, 0, #PB_Path_Relative)
10             AddPathLine(0, 40, #PB_Path_Relative)
11         Next i
```

```

12     VectorSourceColor(RGBA(255, 0, 0, 255))
13     StrokePath(20, #PB_Path_RoundCorner|#PB_Path_RoundEnd)
14
15     StopVectorDrawing()
16 EndIf
17
18 Repeat
19     Event = WaitWindowEvent()
20 Until Event = #PB_Event_CloseWindow
21 EndIf

```



See Also

FillPath() , DotPath() , DashPath() , CustomDashPath() , IsInsideStroke() , ResetPath()

177.38 DotPath

Syntax

```
DotPath(Width.d, Distance.d [, Flags [, StartOffset.d]])
```

Description

Draw the current drawing path as a line of dots.

By default, the path is reset after calling this function. This can be prevented with the appropriate flags.

Parameters

Width.d Specifies the width for the dotted line.

Distance.d Specifies the distance between the center of each dot.

Flags (optional) Specifies optional characteristics for the drawn dots. This can be a combination of the following values:

#PB_Path_Default	: No special behavior (default value)
#PB_Path_Preserve function	: Don't reset the path after this function
#PB_Path_RoundEnd	: Draw the dots round
#PB_Path_SquareEnd	: Draw the dots as squares

StartOffset.d (optional) Specifies the distance to skip within the dot pattern before starting to draw the path. The default value is 0.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",  
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
2      CanvasGadget(0, 0, 0, 400, 200)  
3  
4      If StartVectorDrawing(CanvasVectorOutput(0))  
5  
6          MovePathCursor(40, 20)  
7          For i = 1 To 4  
8              AddPathLine(80, 0, #PB_Path_Relative)  
9              AddPathLine(0, 40, #PB_Path_Relative)  
10             Next i  
11  
12             VectorSourceColor(RGBA(255, 0, 0, 255))  
13             DotPath(5, 10, #PB_Path_RoundEnd)  
14  
15             StopVectorDrawing()  
16         EndIf  
17  
18         Repeat  
19             Event = WaitWindowEvent()  
20         Until Event = #PB_Event_CloseWindow  
21     EndIf
```



See Also

FillPath() , StrokePath() , DashPath() , CustomDashPath() , IsInsideStroke() , ResetPath()

177.39 DashPath

Syntax

```
DashPath(Width.d, Length.d [, Flags [, StartOffset.d]])
```

Description

Draw the current drawing path as a series of dashes of equal length and distance. By default, the path is reset after calling this function. This can be prevented with the appropriate flags.

Parameters

Width.d Specifies the width for the dashed line. This value does not include any round/square line ends.

Length.d Specifies the length of each dash (and the space between the dashes).

Flags (optional) Specifies optional characteristics for the drawn dashes. This can be a combination of the following values:

```

#PB_Path_Default      : No special behavior (default value)
#PB_Path_Preserve     : Don't reset the path after this
function
#PB_Path_RoundEnd     : Draw the dashes with a rounded ends
#PB_Path_SquareEnd    : Draw the dashes with a square box at
the ends
#PB_Path_RoundCorner   : Draw the dashes with rounded corners
#PB_Path_DiagonalCorner : Draw the dashes with diagonally cut
corners

```

StartOffset.d (optional) Specifies the distance to skip within the dash pattern before starting to draw the path. The default value is 0.

Return value

None.

Example

```

1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      CanvasGadget(0, 0, 0, 400, 200)
3
4      If StartVectorDrawing(CanvasVectorOutput(0))
5
6          MovePathCursor(40, 20)
7          For i = 1 To 4
8              AddPathLine(80, 0, #PB_Path_Relative)
9              AddPathLine(0, 40, #PB_Path_Relative)
10             Next i
11
12             VectorSourceColor(RGBA(255, 0, 0, 255))
13             DashPath(5, 15)
14
15             StopVectorDrawing()
16         EndIf
17
18         Repeat
19             Event = WaitWindowEvent()
20             Until Event = #PB_Event_CloseWindow
21         EndIf

```



See Also

FillPath() , StrokePath() , DotPath() , CustomDashPath() , IsInsideStroke() , ResetPath()

177.40 CustomDashPath

Syntax

```
CustomDashPath(Width.d, Array.d() [, Flags [, StartOffset.d]])
```

Description

Draw the current drawing path with a custom dashing pattern.

By default, the path is reset after calling this function. This can be prevented with the appropriate flags.

Parameters

Width.d Specifies the width for the dashed line.

Array.d() Specifies the length of each dash and each space to the next dash. The array must have an even number of entries. When the drawing operation reaches the end of the array, the pattern is repeated. A dash length of 0 will draw a single dot.

Flags (optional) Specifies optional characteristics for the drawn dashes. This can be a combination of the following values:

<code>#PB_Path_Default</code>	: No special behavior (default value)
<code>#PB_Path_Preserve</code>	: Don't reset the path after this function
<code>#PB_Path_RoundEnd</code>	: Draw the dashes with a rounded ends
<code>#PB_Path_SquareEnd</code>	: Draw the dashes with a square box at the ends
<code>#PB_Path_RoundCorner</code>	: Draw the dashes with rounded corners
<code>#PB_Path_DiagonalCorner</code>	: Draw the dashes with diagonally cut corners

StartOffset.d (optional) Specifies the distance to skip within the dash pattern before starting to draw the path. The default value is 0.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",  
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
3      CanvasGadget(0, 0, 0, 400, 200)  
4  
5      If StartVectorDrawing(CanvasVectorOutput(0))  
6  
7          MovePathCursor(40, 20)  
8          For i = 1 To 4  
9              AddPathLine(80, 0, #PB_Path_Relative)  
10             AddPathLine(0, 40, #PB_Path_Relative)  
11         Next i  
12  
13         VectorSourceColor(RGBA(255, 0, 0, 255))  
14  
15         Dim dashes.d(7)  
16         dashes(0) = 20
```

```

16      dashes(1) = 10
17      dashes(2) = 0 ; draw a dot
18      dashes(3) = 10
19      dashes(4) = 0
20      dashes(5) = 10
21      dashes(6) = 20
22      dashes(7) = 10
23      CustomDashPath(5, dashes())
24
25      StopVectorDrawing()
26  EndIf
27
28  Repeat
29      Event = WaitWindowEvent()
30  Until Event = #PB_Event_CloseWindow
31  EndIf

```



See Also

FillPath() , StrokePath() , DotPath() , DashPath() , IsInsideStroke() , ResetPath()

177.41 FillPath

Syntax

```
FillPath([Flags])
```

Description

Fill all closed figures in the current vector drawing path with color from the drawing source. By default, the path is reset after calling this function. This can be prevented with the appropriate flags.

Parameters

Flags (optional) Can be a combination of the following values:

```

#PB_Path_Default      : No special behavior (default value).
#PB_Path_Preserve     : Don't reset the path after this
                        function.
#PB_Path_Winding      : Fill the whole path, including
                        overlapped figures (no odd/even mode).

```

Return value

None.

Remarks

If the path has overlapping figures, it is filled in an odd/even fashion, unless `#PB_Path_Winding` is specified. Areas enclosed in an odd number of borders are filled, while areas enclosed in an even number of borders are not filled. That is, everything within the outer border is filled, while enclosed figures are not filled. If the enclosed figure again contains another figure, that 3rd figure will be filled again, and so on.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      CanvasGadget(0, 0, 0, 400, 200)
4
5      If StartVectorDrawing(CanvasVectorOutput(0))
6          AddPathBox(50, 50, 200, 50)
7          AddPathBox(150, 75, 200, 50)
8          VectorSourceColor(RGBA(0, 0, 255, 255))
9          FillPath()
10
11         StopVectorDrawing()
12     EndIf
13
14     Repeat
15         Event = WaitWindowEvent()
16     Until Event = #PB_Event_CloseWindow
17 EndIf
```



See Also

`StrokePath()` , `DotPath()` , `DashPath()` , `CustomDashPath()` , `ResetPath()` , `ClipPath()`

177.42 ClipPath

Syntax

```
ClipPath([Flags])
```

Description

Clip the vector drawing output to the area defined by the current vector drawing path. Future drawing operations will only affect areas within the current path. The clipping will be combined with any clipping that previously existed on the drawing output.

By default, the path is reset after calling this function. This can be prevented with the appropriate flags.

Parameters

Flags (optional) Can be one of the following values:

<code>#PB_Path_Default</code>	: No special behavior (default value)
<code>#PB_Path_Preserve</code>	: Don't reset the path after this function

Return value

None.

Remarks

There is no "UnclipPath()" function: The clipping region of the drawing output can only be made smaller by adding further clipping, it cannot be made larger again. However, the clipping region can be saved and restored using the `SaveVectorState()` and `RestoreVectorState()` functions respectively. So in order to apply temporary clipping to the drawing output, first save the drawing state and later restore it to go back to the original clipping regions. On Windows, the result will be pixelated on the edges, without antialiasing, which is a limitation of GDI+.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      CanvasGadget(0, 0, 0, 400, 200)
3      LoadFont(0, "Times New Roman", 20, #PB_Font_Bold)
4
5      If StartVectorDrawing(CanvasVectorOutput(0))
6
7          ; Setup a complex clipping path with nested ellipses (every
          second one will be clipped)
8          For i = 10 To 150 Step 5
9              AddPathEllipse(200, 100, 2*i, i)
10             Next i
11             ClipPath()
12
13             ; Draw some text with this clipping
14             VectorFont(FontID(0), 150)
15             VectorSourceColor(RGBA(255, 0, 0, 255))
16
17             MovePathCursor(50, 25)
18             DrawVectorText("Text")
19
20             StopVectorDrawing()
21         EndIf
22
23     Repeat
24         Event = WaitWindowEvent()
25     Until Event = #PB_Event_CloseWindow
26     EndIf
```



See Also

177.43 PathCursorX

Syntax

```
Result.d = PathCursorX()
```

Description

Returns the current X coordinate of the vector drawing cursor. This is the location where new path segments will be added or text will be drawn.

Parameters

None.

Return value

The X coordinate of the path cursor.

See Also

PathCursorY() , MovePathCursor() , ResetPath()

177.44 PathCursorY

Syntax

```
Result.d = PathCursorY()
```

Description

Returns the current Y coordinate of the vector drawing cursor. This is the location where new path segments will be added or text will be drawn.

Parameters

None.

Return value

The Y coordinate of the path cursor.

See Also

PathCursorX() , MovePathCursor() , ResetPath()

177.45 PathPointX

Syntax

```
Result.d = PathPointX(Distance.d)
```

Description

Returns the X coordinate of the point at the given distance from the start of the current vector drawing path.

Parameters

Distance.d Specifies the distance from the start of the path. If this parameter is negative or larger than the total path length, the start/endpoint of the path is returned. The full length of the path can be determined with PathLength() .

Return value

The X coordinate of the point of the path.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     CanvasGadget(0, 0, 0, 400, 200)
4
5     If StartVectorDrawing(CanvasVectorOutput(0))
6
7         ; construct path
8         MovePathCursor(150, 125)
9         AddPathCurve(0, 270, 0, -150, 350, 180)
10
11        ; get location & angle of point on the path
12        x = PathPointX(200)
13        y = PathPointY(200)
14        a = PathPointAngle(200)
15
16        ; stroke the path
17        VectorSourceColor($FF0000FF)
18        StrokePath(5)
19
20        ; draw a marker at the path point
21        AddPathCircle(x, y, 10)
22        VectorSourceColor($FFFF0000)
23        FillPath()
24
25        MovePathCursor(x, y)
26        AddPathLine(30*Cos(Radian(a)), 30*Sin(Radian(a)),
27        #PB_Path_Relative)
28        StrokePath(5)
29
30        StopVectorDrawing()
31    EndIf
32
33    Repeat
34        Event = WaitWindowEvent()
```

```
33     Until Event = #PB_Event_CloseWindow
34 EndIf
```



See Also

PathPointY() , PathPointAngle() , PathLength()

177.46 PathPointY

Syntax

```
Result.d = PathPointY(Distance.d)
```

Description

Returns the Y coordinate of the point at the given distance from the start of the current vector drawing path.

Parameters

Distance.d Specifies the distance from the start of the path. If this parameter is negative or larger than the total path length, the start/endpoint of the path is returned. The full length of the path can be determined with PathLength() .

Return value

The Y coordinate of the point of the path.

Example

See PathPointX() for an example.

See Also

PathPointX() , PathPointAngle() , PathLength()

177.47 PathPointAngle

Syntax

```
Result.d = PathPointAngle(Distance.d)
```

Description

Returns the angle of the path at the point at the given distance from the start of the current vector drawing path.

Parameters

Distance.d Specifies the distance from the start of the path. If this parameter is negative or larger than the total path length, the start/endpoint of the path is returned. The full length of the path can be determined with `PathLength()` .

Return value

The angle of the path at the given point in degrees. The angle 0 marks at the positive X axis.

Example

See `PathPointX()` for an example.

See Also

`PathPointX()` , `PathPointY()` , `PathLength()`

177.48 PathLength

Syntax

```
Result.d = PathLength()
```

Description

Returns the total length of the current vector drawing path.

Parameters

None.

Return value

Returns the length of the current path.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      CanvasGadget(0, 0, 0, 400, 200)
4
5      If StartVectorDrawing(CanvasVectorOutput(0))
6          ; construct path
7          MovePathCursor(150, 125)
8          AddPathCurve(0, 270, 0, -150, 350, 180)
9
10         ; get length
11         Debug "Path length: " + PathLength()
12
13         ; stroke the path
14         VectorSourceColor($FF0000FF)
15         StrokePath(5)
16     EndIf
```

```

17         StopVectorDrawing()
18     EndIf
19
20     Repeat
21         Event = WaitWindowEvent()
22     Until Event = #PB_Event_CloseWindow
23     EndIf

```

See Also

PathPointX() , PathPointY() , PathPointAngle()

177.49 PathBoundsX

Syntax

```
Result.d = PathBoundsX()
```

Description

Returns the X coordinate (top/left corner) of the bounding box for the current vector drawing path. The result is the lowest X coordinate that stroking/filling the current path would reach.

Parameters

None.

Return value

The X coordinate of the bounding box.

Example

```

1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      CanvasGadget(0, 0, 0, 400, 200)
4
5      If StartVectorDrawing(CanvasVectorOutput(0))
6
7          ; construct path
8          MovePathCursor(150, 125)
9          AddPathCurve(0, 270, 0, -150, 350, 180)
10
11         ; get path bounds
12         x = PathBoundsX()
13         y = PathBoundsY()
14         w = PathBoundsWidth()
15         h = PathBoundsHeight()
16
17         ; stroke the path
18         VectorSourceColor($FF0000FF)
19         StrokePath(5)
20
21         ; draw bounding box
22         AddPathBox(x, y, w, h)

```

```

22     VectorSourceColor($FF000000)
23     DashPath(2, 5)
24
25     StopVectorDrawing()
26 EndIf
27
28 Repeat
29     Event = WaitWindowEvent()
30 Until Event = #PB_Event_CloseWindow
31 EndIf

```



See Also

PathBoundsY() , PathBoundsWidth() , PathBoundsHeight()

177.50 PathBoundsY

Syntax

```
Result.d = PathBoundsY()
```

Description

Returns the Y coordinate (top/left corner) of the bounding box for the current vector drawing path. The result is the lowest Y coordinate that stroking/filling the current path would reach.

Parameters

None.

Return value

The Y coordinate of the bounding box.

Example

See PathBoundsX() for an example.

See Also

PathBoundsX() , PathBoundsWidth() , PathBoundsHeight()

177.51 PathBoundsWidth

Syntax

```
Result.d = PathBoundsWidth()
```

Description

Returns the width of the bounding box for the current vector drawing path. The result is the difference between the lowest & highest X coordinate that stroking/filling the current path would reach.

Parameters

None.

Return value

The width of the bounding box.

Example

See PathBoundsX() for an example.

See Also

PathBoundsX() , PathBoundsY() , PathBoundsHeight()

177.52 PathBoundsHeight

Syntax

```
Result.d = PathBoundsHeight()
```

Description

Returns the height of the bounding box for the current vector drawing path. The result is the difference between the lowest & highest Y coordinate that stroking/filling the current path would reach.

Parameters

None.

Return value

The height of the bounding box.

Example

See PathBoundsX() for an example.

See Also

PathBoundsX() , PathBoundsY() , PathBoundsWidth()

177.53 PathSegments

Syntax

```
Result\$_ = PathSegments()
```

Description

Returns a string description of the current vector drawing path. The result can be used to examine the current path or in the AddPathSegments() command to reproduce the same path later.

Parameters

None.

Return value

The returned string contains one letter commands followed by the appropriate number of coordinate parameters. Each value is separated by a single space. All coordinates are absolute.

M x y	MovePathCursor()
L x y	AddPathLine()
C x1 y1 x2 y2 x3 y3	AddPathCurve()
Z	ClosePath()

There are no string representations for commands like AddPathCircle() or AddPathEllipse() , as their results are internally converted to curves by the vector drawing library.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     CanvasGadget(0, 0, 0, 400, 200)
4
5     If StartVectorDrawing(CanvasVectorOutput(0))
6
7         MovePathCursor(40, 20)
8         For i = 1 To 4
9             AddPathLine(80, 0, #PB_Path_Relative)
10            AddPathLine(0, 40, #PB_Path_Relative)
11        Next i
12
13        ; Show the path segments
14        Debug PathSegments()
15
16        VectorSourceColor(RGBA(255, 0, 0, 255))
17        StrokePath(10, #PB_Path_RoundCorner)
18
19        StopVectorDrawing()
20    EndIf
```

```

20
21     Repeat
22         Event = WaitWindowEvent()
23     Until Event = #PB_Event_CloseWindow
24 EndIf

```

See Also

AddPathSegments()

177.54 VectorSourceColor

Syntax

```
VectorSourceColor(Color)
```

Description

Selects a single color as the source for vector drawing operations such as FillPath() , StrokePath() and others.

Parameters

Color The 32bit RGBA color including alpha transparency.

Return value

None.

See Also

VectorSourceLinearGradient() , VectorSourceCircularGradient() , VectorSourceImage() , FillPath() , FillVectorOutput() , StrokePath() , DotPath() , DashPath() , CustomDashPath()

177.55 VectorSourceLinearGradient

Syntax

```
VectorSourceLinearGradient(x1.d, y1.d, x2.d, y2.d)
```

Description

Selects a linear color gradient as the source for vector drawing operations such as FillPath() or StrokePath() . Initially, the gradient is solid black. Color stops have to be added with the VectorSourceGradientColor() after this function.

Parameters

x1.d, y1.d Specifies the point that represents the start (Position 0.0) of the gradient. The coordinates are specified in terms of the `#PB_Coordinate_Source` coordinate system.

x2.d, y2.d Specifies the point that represents the end (Position 1.0) of the gradient. The coordinates are specified in terms of the `#PB_Coordinate_Source` coordinate system.

Return value

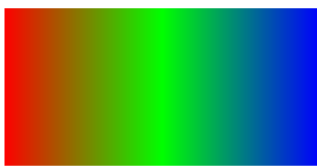
None.

Remarks

See the vectordrawing overview for an introduction to the different coordinate systems.
The color gradient is only defined in the area between the (x1, y1) and (x2, y2) points. Outside of these points, the used source color is depending on the operating system, so drawing operations outside of the defined gradient's area should be avoided.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      CanvasGadget(0, 0, 0, 400, 200)
3
4      If StartVectorDrawing(CanvasVectorOutput(0))
5
6          VectorSourceLinearGradient(50, 0, 350, 0)
7          VectorSourceGradientColor(rgba(255, 0, 0, 255), 0.0)
8          VectorSourceGradientColor(rgba(0, 255, 0, 255), 0.5)
9          VectorSourceGradientColor(rgba(0, 0, 255, 255), 1.0)
10
11         AddPathBox(50, 25, 300, 150)
12         FillPath()
13
14         StopVectorDrawing()
15     EndIf
16
17     Repeat
18         Event = WaitWindowEvent()
19     Until Event = #PB_Event_CloseWindow
20 EndIf
```



See Also

VectorSourceGradientColor(), VectorSourceCircularGradient(), VectorSourceColor(),
VectorSourceImage()

177.56 VectorSourceCircularGradient

Syntax

```
VectorSourceCircularGradient(x.d, y.d, Radius.d, [CenterX.d,  
    CenterY.d])
```

Description

Selects a circular gradient as the source for vector drawing operations such as `FillPath()` or `StrokePath()`. Initially, the gradient is solid black. Color stops have to be added with the `VectorSourceGradientColor()` after this function.

Parameters

x.d, y.d Specifies the center point of the circle that defines the gradient. The coordinates are specified in terms of the `#PB_Coordinate_Source` coordinate system.
The center point of the circle represents the start (Position 0.0) of the gradient and the perimeter of the circle represents the end (Position 1.0) of the gradient.

Radius.d Specifies the radius of the circle that defines the gradient.

CenterX.d, CenterY.d (optional) Specifies an optional offset for the starting point of the gradient from the center of the circle. With these parameters, the gradient can be started at any point within the circle.

Return value

None.

Remarks

See the `vectordrawing` overview for an introduction to the different coordinate systems.
The color gradient is only defined in the area inside the circle. Outside of the circle, the used source color is depending on the operating system, so drawing operations outside of the defined gradient's area should be avoided.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      CanvasGadget(0, 0, 0, 400, 200)
4
5      If StartVectorDrawing(CanvasVectorOutput(0))
6          VectorSourceCircularGradient(200, 100, 150, -50, -50)
7          VectorSourceGradientColor(RGBA(255, 255, 255, 255), 0.0)
8          VectorSourceGradientColor(RGBA(0, 0, 0, 255), 1.0)
9
10         FillVectorOutput()
11
12         StopVectorDrawing()
13     EndIf
14
15     Repeat
16         Event = WaitWindowEvent()
17     Until Event = #PB_Event_CloseWindow
18 EndIf
```



See Also

VectorSourceGradientColor() , VectorSourceLinearGradient() , VectorSourceColor() , VectorSourceImage()

177.57 VectorSourceGradientColor

Syntax

```
VectorSourceGradientColor(Color , Position.d)
```

Description

Add a new color stop (a defined color position) to the gradient defined by VectorSourceLinearGradient() or VectorSourceCircularGradient() .
A gradient must at least have a color at position 0.0 and 1.0. If no colors are added for these positions then they default to solid black. Any number of color positions can be added to a gradient.

Parameters

Color The 32bit RGBA color including alpha transparency.

Position.d The position at which to add the color. The value must be between and including 0.0 (the gradient start) and 1.0 (the gradient end).

Return value

None.

Example

See VectorSourceLinearGradient() for an example.

See Also

VectorSourceLinearGradient() , VectorSourceCircularGradient()

177.58 VectorSourceImage

Syntax

```
VectorSourceImage(ImageID [, Alpha [, Width.d, Height.d [, Flags]]])
```

Description

Selects an image as the source for vector drawing operations such as FillPath() or StrokePath() . These functions will apply pixels from the specified image to the drawing output wherever they draw something.

Parameters

ImageID Specifies the image to use as the source. Use the ImageID() function to get this value from an image.

Alpha (optional) Specifies an optional alpha transparency to apply to the source image. This transparency is applied in addition to any transparent pixels already present in the source image. The default is value is 255 (no additional transparency).

Width.d, Height.d (optional) Specifies an optional with and height for the image. The values are interpreted in terms of the #PB_Coordinate_Source coordinate system. If no width and height are specified, then the dimensions of the source image (in pixels) are converted into the unit of the vector drawing output and used (i.e. the image has its original size).

Flags (optional) Can be one of the following values:

```
#PB_VectorImage_Default: Areas outside of the source image
                        are transparent (default)
#PB_VectorImage_Repeat : The source image is repeated to
                        cover the entire drawing area
```

Return value

None.

Remarks

See the vectordrawing overview for an introduction to the different coordinate systems. By transforming the #PB_Coordinate_Source coordinate system, the used source image can be transformed (moved, rotated, stretched, skewed). See the second example below for a demonstration.

Example: Repeated source image

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      CanvasGadget(0, 0, 0, 400, 200)
4
5      LoadImage(0, #PB_Compiler_Home +
6      "examples/sources/Data/PureBasicLogo.bmp")
7
8      If StartVectorDrawing(CanvasVectorOutput(0))
9          AddPathBox(50, 50, 200, 50)
10         AddPathBox(150, 75, 200, 50)
11
12         VectorSourceImage(ImageID(0), 255, ImageWidth(0),
13         ImageHeight(0), #PB_VectorImage_Repeat)
14         StrokePath(20)
15
16         StopVectorDrawing()
17     EndIf
18
19     Repeat
20         Event = WaitWindowEvent()
21     Until Event = #PB_Event_CloseWindow
22 EndIf
```



Example: Rotated and flipped source image

```

1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      CanvasGadget(0, 0, 0, 400, 200)
3
4      LoadImage(0, #PB_Compiler_Home +
    "examples/sources/Data/PureBasicLogo.bmp")
5      LoadFont(0, "Impact", 20, #PB_Font_Bold)
6
7      If StartVectorDrawing(CanvasVectorOutput(0))
8
9          FlipCoordinatesY(50, #PB_Coordinate_Source)
10         RotateCoordinates(50, 50, -45, #PB_Coordinate_Source)
11         VectorSourceImage(ImageID(0), 255, ImageWidth(0),
    ImageHeight(0), #PB_VectorImage_Repeat)
12
13         VectorFont(FontID(0), 150)
14         MovePathCursor(20, 20)
15         DrawVectorText("TEXT")
16
17         StopVectorDrawing()
18     EndIf
19
20     Repeat
21         Event = WaitWindowEvent()
22     Until Event = #PB_Event_CloseWindow
23 EndIf

```



See Also

VectorSourceColor() , VectorSourceLinearGradient() , VectorSourceCircularGradient()

177.59 DrawVectorImage

Syntax

DrawVectorImage(ImageID [, Alpha [, Width.d, Height.d]])

Description

Draw the specified image directly to the vector drawing output.

The image will be drawn at the location of the path cursor . The cursor will be moved to the location of the bottom/right corner of the image after the image is drawn.

Parameters

ImageID (optional) Specifies the image to draw. Use the ImageID() function to get this value from an image.

Alpha (optional) Specifies an optional alpha transparency to apply to the image. This transparency is applied in addition to any transparent pixels already present in the image. The default is value is 255 (no additional transparency).

Width.d, Height.d (optional) Specifies an optional with and height for the image. If no width and height are specified, then the dimensions of the source image (in pixels) are converted into the unit of the vector drawing output and used (i.e. the image has its original size).

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     CanvasGadget(0, 0, 0, 400, 200)
4
5     LoadImage(0, #PB_Compiler_Home +
6     "examples/sources/Data/PureBasicLogo.bmp")
7
8     If StartVectorDrawing(CanvasVectorOutput(0))
9
10        MovePathCursor(50, 50)
11        DrawVectorImage(ImageID(0), 127)
12
13        MovePathCursor(75, 75)
14        DrawVectorImage(ImageID(0), 127, ImageWidth(0) / 2,
15        ImageHeight(0))
16
17        MovePathCursor(120, 0)
18        RotateCoordinates(120, 0, 35)
19        DrawVectorImage(ImageID(0), 127)
20
21        StopVectorDrawing()
22    EndIf
23
24    Repeat
25        Event = WaitWindowEvent()
26    Until Event = #PB_Event_CloseWindow
27    EndIf
```




See Also

MovePathCursor() , PathCursorX() , PathCursorY() , VectorSourceImage()

177.60 DrawVectorText

Syntax

```
DrawVectorText(Text$)
```

Description

Draw the given text at the current location of the path cursor . The cursor will be moved horizontally to the end of the drawn text. The font to use can be set with VectorFont() .

Parameters

Text\$ The text to draw (single line).

Return value

None.

Remarks

Only vector fonts are allowed, like TrueType, bitmap fonts are not allowed.
This function draws single lines of text only. Multiple calls must be made to draw multiple lines.
Use VectorTextWidth() and VectorTextHeight() to determine the dimensions of the text to draw in order to properly align the text with other content.
The DrawVectorParagraph() function can be used to draw a larger block of text with automatic layout such as line breaks. This function is more suited for drawing multiline text.

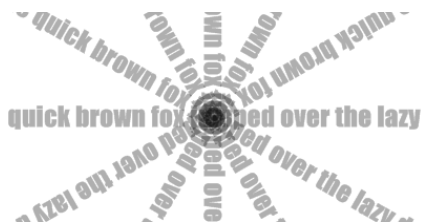
Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",  
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
3      CanvasGadget(0, 0, 0, 400, 200)  
4      LoadFont(0, "Impact", 20, #PB_Font_Bold)  
5  
6      If StartVectorDrawing(CanvasVectorOutput(0))  
7          VectorFont(FontID(0), 25)  
8          VectorSourceColor(RGBA(0, 0, 0, 80))  
9          Text$ = "The quick brown fox jumps over the lazy dog"  
10  
11      For i = 1 To 6
```

```

12         MovePathCursor(200 - VectorTextWidth(Text$)/2, 100 -
VectorTextHeight(Text$)/2)
13         DrawVectorText(Text$)
14         RotateCoordinates(200, 100, 30)
15     Next i
16
17     StopVectorDrawing()
18 EndIf
19
20 Repeat
21     Event = WaitWindowEvent()
22 Until Event = #PB_Event_CloseWindow
23 EndIf

```



See Also

VectorTextWidth() , VectorTextHeight() , DrawVectorParagraph() , AddPathText() , VectorFont()

177.61 DrawVectorParagraph

Syntax

```
DrawVectorParagraph(Text$, Width.d, Height.d [, Flags])
```

Description

Draw a paragraph of text (multiple lines) within a given bounding box with automatic layout for linebreaks. If the text does not fit the defined box, it will be cut at the end. The font to use can be set with VectorFont() .

Parameters

Text\$ Specifies the text to draw.

Width.d Specifies the width for the paragraph. Line breaks will be added if text is longer than the specified width.

Height.d Specifies the maximum height for the paragraph. If the text does not fit within this height it will be cut. The required height for a paragraph can be calculated with VectorParagraphHeight() .

Flags (optional) Can be one of the following values:

```

#PB_VectorParagraph_Left   : The paragraph is aligned to the
left (default)
#PB_VectorParagraph_Right  : The paragraph is aligned to the
right
#PB_VectorParagraph_Center : The paragraph is centered
#PB_VectorParagraph_Block  : Lines in the paragraph are spaced
to form a block of text (not supported on Windows)

```

Return value

None.

Remarks

Only vector fonts are allowed, like TrueType, bitmap fonts are not allowed.

Example

```
1  If OpenWindow(0, 0, 0, 400, 250, "VectorDrawing",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      CanvasGadget(0, 0, 0, 400, 250)
4      LoadFont(0, "Times New Roman", 20)
5
6      If StartVectorDrawing(CanvasVectorOutput(0))
7
8          Text$ = "Every drawing output has a default unit of
9              measurement. The default unit is pixels " +
10                 "for screen or raster image outputs and points for
11                 printer or vector image outputs. " +
12                 "It is however possible to select a different unit of
13                 measurement for the output when " +
14                 "creating it with the ImageVectorOutput(),
15                 PrinterVectorOutput() or similar function."
16
17         VectorFont(FontID(0), 18)
18
19         MovePathCursor(25, 25)
20         DrawVectorParagraph(Text$, 350, 200)
21
22     StopVectorDrawing()
23 EndIf
24
25 Repeat
26     Event = WaitWindowEvent()
27 Until Event = #PB_Event_CloseWindow
28 EndIf
```

Every drawing output has a default unit of measurement. The default unit is pixels for screen or raster image outputs and points for printer or vector image outputs. It is however possible to select a different unit of measurement for the output when creating it with the ImageVectorOutput(), PrinterVectorOutput() or similar function.

See Also

VectorParagraphHeight() , DrawVectorText() , AddPathText() , VectorFont()

177.62 VectorFont

Syntax

```
VectorFont(FontID [, Size.d])
```

Description

Specifies the font to use for vector drawing.

Only vector fonts are allowed, like TrueType, bitmap fonts are not allowed.

Parameters

FontID The FontID() of the font to use for drawing.

Size.d (optional) Specifies the size for the font. The size is measured in the units of the vector drawing output. If no size is specified, then the size used in the LoadFont() command for the font will be converted to the current vector drawing unit.

Return value

None.

Remarks

You cannot use a registered font with RegisterFontFile() .

See Also

DrawVectorText() , DrawVectorParagraph() , VectorTextWidth() , VectorTextHeight() , VectorParagraphHeight()

177.63 VectorTextWidth

Syntax

```
Result.d = VectorTextWidth(Text$ [, Flags])
```

Description

Measures the width of the given text in the current vector drawing font.

Parameters

Text\$ The text (single-line) to measure.

Flags (optional) Can be a combination of the following values:

```
#PB_VectorText_Default: Return the logical bounding box of
                        the text
#PB_VectorText_Visible: Return the visible bounding box of
                        the text
#PB_VectorText_Offset : Return the offset of the bounding box
                        from the current position rather than the width
```

Return value

Returns the text width in units of the vector drawing output.

Remarks

The dimensions of drawn text can be defined in terms of two bounding boxes:

The "logical bounding box" of a character or text defines the space that the cursor must move to properly draw text next to each other. However, the actual drawn characters may extend beyond that box (for example in case of cursive or serif fonts). When determining where to draw text, the logical bounding box is the interesting one.

The "visible bounding box" of a character or text defines the area in which the text is actually drawn. This area is usually larger than the logical bounding box. The visible dimensions of the text can be retrieved by specifying the `#PB_VectorText_Visible` flag. The visible dimensions of the text can be at an offset to the logical ones. This offset can be calculated by specifying the `#PB_VectorText_Offset` flag.

The following example shows a sample text with the logical bounding box in blue, the visible bounding box in red and the location of the baseline in green. The origin at which the text is drawn is the upper left corner of the logical bounding box (blue).

Example

```
1  If OpenWindow(0, 0, 0, 500, 250, "VectorDrawing",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      CanvasGadget(0, 0, 0, 500, 250)
3      LoadFont(0, "Monotype Corsiva", 20, #PB_Font_Italic)
4
5      If StartVectorDrawing(CanvasVectorOutput(0))
6
7          VectorFont(FontID(0), 125)
8          Text$ = "Sample"
9
10         ; draw text
11         MovePathCursor(25, 25)
12         DrawVectorText(Text$)
13
14         ; draw logical bounding box
15         AddPathBox(25, 25, VectorTextWidth(Text$),
VectorTextHeight(Text$))
16         VectorSourceColor(RGBA(0, 0, 255, 255))
17         DashPath(2, 10)
18
19         ; draw visible bounding box
20         AddPathBox(25 + VectorTextWidth(Text$,
#PB_VectorText_Visible | #PB_VectorText_Offset),
21                 25 + VectorTextHeight(Text$,
#PB_VectorText_Visible | #PB_VectorText_Offset),
22                 VectorTextWidth(Text$, #PB_VectorText_Visible),
23                 VectorTextHeight(Text$, #PB_VectorText_Visible))
24         VectorSourceColor(RGBA(255, 0, 0, 255))
25         DashPath(2, 10)
26
27         ; draw baseline
28         MovePathCursor(25, 25 + VectorTextHeight(Text$,
#PB_VectorText_Baseline))
29         AddPathLine(VectorTextWidth(Text$), 0, #PB_Path_Relative)
30         VectorSourceColor(RGBA(0, 255, 0, 255))
31         DashPath(2, 10)
```

```

32
33     StopVectorDrawing()
34 EndIf
35
36 Repeat
37     Event = WaitWindowEvent()
38 Until Event = #PB_Event_CloseWindow
39 EndIf

```



See Also

VectorTextHeight() , DrawVectorText() , DrawVectorParagraph() , VectorParagraphHeight() , VectorFont()

177.64 VectorTextHeight

Syntax

```
Result.d = VectorTextHeight(Text$ [, Flags])
```

Description

Measures the height of the given text in the current vector drawing font.

Parameters

Text\$ The text (single-line) to measure.

Flags (optional) Can be a combination of the following values:

```

#PB_VectorText_Default : Return the logical bounding box of
the text
#PB_VectorText_Visible : Return the visible bounding box of
the text
#PB_VectorText_Offset  : Return the offset of the bounding
box from the current position rather than the height
#PB_VectorText_Baseline: Return the offset of the text
baseline from the current position

```

Return value

Returns the text height in units of the vector drawing output.

Remarks

The dimensions of drawn text can be defined in terms of two bounding boxes:

The "logical bounding box" of a character or text defines the space that the cursor must move to properly draw text next to each other. However, the actual drawn characters may extend beyond that box (for example in case of cursive or serif fonts). When determining where to draw text, the logical bounding box is the interesting one.

The "visible bounding box" of a character or text defines the area in which the text is actually drawn. This area is usually larger than the logical bounding box. The visible dimensions of the text can be retrieved by specifying the `#PB_VectorText_Visible` flag. The visible dimensions of the text can be at an offset to the logical ones. This offset can be calculated by specifying the `#PB_VectorText_Offset` flag.

The "baseline" defines the vertical distance from the origin of the drawn text to the line where the letters "sit". It is useful to draw text of different heights on a single line. The baseline value for the text can be retrieved by specifying the `#PB_VectorText_Baseline` flag.

The following example shows a sample text with the logical bounding box in blue, the visible bounding box in red and the location of the baseline in green. The origin at which the text is drawn is the upper left corner of the logical bounding box (blue).

Example

```
1  If OpenWindow(0, 0, 0, 500, 250, "VectorDrawing",
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      CanvasGadget(0, 0, 0, 500, 250)
3      LoadFont(0, "Monotype Corsiva", 20, #PB_Font_Italic)
4
5      If StartVectorDrawing(CanvasVectorOutput(0))
6
7          VectorFont(FontID(0), 125)
8          Text$ = "Sample"
9
10         ; draw text
11         MovePathCursor(25, 25)
12         DrawVectorText(Text$)
13
14         ; draw logical bounding box
15         AddPathBox(25, 25, VectorTextWidth(Text$),
VectorTextHeight(Text$))
16         VectorSourceColor(RGBA(0, 0, 255, 255))
17         DashPath(2, 10)
18
19         ; draw visible bounding box
20         AddPathBox(25 + VectorTextWidth(Text$,
#PB_VectorText_Visible | #PB_VectorText_Offset),
21             25 + VectorTextHeight(Text$,
#PB_VectorText_Visible | #PB_VectorText_Offset),
22             VectorTextWidth(Text$, #PB_VectorText_Visible),
23             VectorTextHeight(Text$, #PB_VectorText_Visible))
24         VectorSourceColor(RGBA(255, 0, 0, 255))
25         DashPath(2, 10)
26
27         ; draw baseline
28         MovePathCursor(25, 25 + VectorTextHeight(Text$,
#PB_VectorText_Baseline))
29         AddPathLine(VectorTextWidth(Text$), 0, #PB_Path_Relative)
30         VectorSourceColor(RGBA(0, 255, 0, 255))
31         DashPath(2, 10)
32
33         StopVectorDrawing()
```

```

34     EndIf
35
36     Repeat
37         Event = WaitWindowEvent()
38     Until Event = #PB_Event_CloseWindow
39 EndIf

```



See Also

VectorTextWidth() , DrawVectorText() , DrawVectorParagraph() , VectorParagraphHeight() , VectorFont()

177.65 VectorParagraphHeight

Syntax

```
Result.d = VectorParagraphHeight(Text$, Width.d, Height.d)
```

Description

Returns the height needed to draw the given paragraph of text using the DrawVectorParagraph() function.

Parameters

Text\$ The paragraph of text to measure (can be multiple lines).

Width.d The width to use for the paragraph.

Height.d The maximum height available for the paragraph.

Return value

The actual height needed to draw the paragraph of text.

Remarks

If the text does not fit within the defined bounding box, the result will be equal to the value of "Height.d". This indicates that calling DrawVectorParagraph() would cut off parts of the text.

See Also

DrawVectorParagraph() , DrawVectorText() , VectorFont() , VectorTextWidth() , VectorTextHeight()

177.66 PdfVectorOutput

Syntax

```
Result = PdfVectorOutput(FileName$, Width.d, Height.d [, Unit])
```

Description

Creates a PDF file and returns the OutputID to perform vector drawing operations. The actual drawing operations must be enclosed in a StartVectorDrawing() / StopVectorDrawing() block. The PDF file can have multiple pages using the NewVectorPage() command.

Parameters

FileName\$ Specifies the filename of the PDF to create. If the file exists, it will be overwritten.

Width.d, Height.d Specifies the dimensions of a page in the PDF in units of the vector drawing output.

Unit (optional) Specifies the units for the vector drawing output. The default unit for PDF files is #PB_Unit_Point.

```
#PB_Unit_Pixel      : Values are measured in pixels (or dots  
                     in case of a printer)  
#PB_Unit_Point     : Values are measured in points (1/72 inch)  
#PB_Unit_Inch      : Values are measured in inches  
#PB_Unit_Millimeter: Values are measured in millimeters
```

Return value

The OutputID of the given file to perform 2D rendering operation on it using StartVectorDrawing() .

Example

```
1  LoadFont(0, "Times New Roman", 20)  
2  
3  If StartVectorDrawing(PdfVectorOutput("test.pdf", 595, 842))  
4      VectorFont(FontID(0), 25)  
5  
6      MovePathCursor(20, 20)  
7      DrawVectorText("This is page 1...")  
8  
9      NewVectorPage()  
10  
11     MovePathCursor(20, 20)  
12     DrawVectorText("This is page 2...")  
13  
14     StopVectorDrawing()  
15 EndIf
```

See Also

SvgVectorOutput() , ImageVectorOutput() , PrinterVectorOutput() , WindowVectorOutput() ,
CanvasVectorOutput()

177.67 SvgVectorOutput

Syntax

```
Result = SvgVectorOutput(FileName$, Width.d, Height.d [, Unit])
```

Description

Creates an SVG (scalable vector graphics) file and returns the OutputID to perform vector drawing operations. The actual drawing operations must be enclosed in a StartVectorDrawing() / StopVectorDrawing() block. The SVG file can have multiple pages using the NewVectorPage() command.

Parameters

FileName\$ Specifies the filename of the SVG file to create. If the file exists, it will be overwritten.

Width.d, Height.d Specifies the dimensions of a page in the SVG file in units of the vector drawing output.

Unit (optional) Specifies the units for the vector drawing output. The default unit for SVG files is #PB_Unit_Point.

```
#PB_Unit_Pixel      : Values are measured in pixels (or dots  
                     in case of a printer)  
#PB_Unit_Point     : Values are measured in points (1/72 inch)  
#PB_Unit_Inch      : Values are measured in inches  
#PB_Unit_Millimeter: Values are measured in millimeters
```

Return value

The OutputID of the given file to perform 2D rendering operation on it using StartVectorDrawing() .

Example

```
1  If StartVectorDrawing(SvgVectorOutput("test.svg", 400, 200))  
2  
3      AddPathBox(50, 50, 200, 50)  
4      AddPathBox(150, 75, 200, 50)  
5      VectorSourceColor(RGBA(255, 0, 0, 255))  
6      StrokePath(10)  
7  
8      StopVectorDrawing()  
9  EndIf
```

See Also

PdfVectorOutput() , ImageVectorOutput() , PrinterVectorOutput() , WindowVectorOutput() , CanvasVectorOutput()

Chapter 178

Vehicle

Overview

Vehicle are usually composed of one body and one or many wheels (cars, trucks, bicycle, etc.). The new created vehicle is a regular entity , so all entity functions can be used to manipulate a vehicle. `InitEngine3D()` should be called successfully before using the vehicle functions.

178.1 AddVehicleWheel

Syntax

```
AddVehicleWheel(#Entity, #WheelEntity, ConnectX.f, ConnectY.f,  
                ConnectZ.f, AxisX.f, AxisY.f, AxisZ.f, MaxSuspensionLength.f,  
                WheelRadius.f, TractionWheel, RollInfluence.f)
```

Description

Add a new wheel to a vehicle previous created with `CreateVehicle()` .

Parameters

#Entity The vehicle entity to use.

#WheelEntity The entity to use for the wheel.

ConnectX, ConnectY, ConnectZ The starting point of the ray, where the suspension connects to the chassis.

AxisX, AxisY, AxisZ The axis the wheel rotates around.

MaxSuspensionLength The maximum length of the suspension (in meters).

WheelRadius Radius of the wheel.

TractionWheel If sets to `#True`, this wheel will be a traction wheel: `ApplyVehicleForce()` and `ApplyVehicleSteering()` can be used on it. If sets to `#False`, no force can be applied to the wheel.

RollInfluence Reduces the rolling torque applied from the wheels that cause the vehicle to roll over. (0.0: no roll, 1.0: physical behaviour). If friction is too high, it could be needed to reduce this value to stop the vehicle rolling over.

Return value

None.

See Also

`ApplyVehicleBrake()` , `ApplyVehicleForce()` , `CreateVehicle()`

178.2 ApplyVehicleForce

Syntax

```
ApplyVehicleForce(#Entity, Wheel, Force.f)
```

Description

Apply the specified traction force to the vehicle wheel. The new traction force value replace any previous force previously applied to the vehicle wheel.

Parameters

#Entity The vehicle entity to use.

Wheel The wheel index, starting from 0.

Force The traction force to apply on the wheel.

Return value

None.

See Also

`ApplyVehicleSteering()` , `ApplyVehicleBrake()` , `AddVehicleWheel()`

178.3 ApplyVehicleBrake

Syntax

```
ApplyVehicleBrake(#Entity, Wheel, Brake.f)
```

Description

Apply the specified brake force to the vehicle wheel. The new brake force value replace any previous force previously applied to the vehicle wheel.

Parameters

#Entity The vehicle entity to use.

Wheel The wheel index, starting from 0.

Brake The brake force to apply on the wheel.

Return value

None.

See Also

ApplyVehicleSteering() , ApplyVehicleForce() , AddVehicleWheel()

178.4 ApplyVehicleSteering

Syntax

```
ApplyVehicleSteering(#Entity, Wheel, Steering.f)
```

Description

Apply the specified steering force to the vehicle wheel. The new steering force value replace any previous force previously applied to the vehicle wheel.

Parameters

#Entity The vehicle entity to use.

Wheel The wheel index, starting from 0.

Steering The steering force to apply on the wheel.

Return value

None.

See Also

ApplyVehicleBrake() , ApplyVehicleForce() , AddVehicleWheel()

178.5 CreateVehicle

Syntax

```
Result = CreateVehicle(#Entity)
```

Description

Creates a new vehicle #Entity.

Parameters

#Entity The number to identify the new entity. #PB_Any can be used to auto-generate this number.

Return value

Returns zero if the vehicle entity can't be created. If #PB_Any is used as '#Entity' parameter, the new vehicle entity number is returned.

See Also

FreeEntity()

178.6 CreateVehicleBody

Syntax

```
CreateVehicleBody(#Entity, Mass.f, Restitution.f, Friction.f [,  
    SuspensionStiffness.f, SuspensionCompression.f,  
    SuspensionDamping.f, MaxSuspensionCompression.f, Friction.f)
```

Description

Creates a physic body associated with the vehicle #Entity.

To have its collisions managed by the physic engine, an entity has to set a body. In fact, only the body is known by the physic engine, which will do all the calculation about the entity, check the mass, friction and if it collides will move back the real entity.

To have any effect, the physic engine needs to be activated with the EnableWorldPhysics() .

Parameters

#Entity The vehicle entity to use.

Mass Mass of the vehicle. Don't use too big value or it could produce physic incoherencies (1 is the preferred value).

Restitution Restitution of the vehicle. This value can also be get or set via GetEntityAttribute() and SetEntityAttribute()

Friction Friction of the vehicle. This value can also be get or set via GetEntityAttribute() and SetEntityAttribute()

SuspensionStiffness (optional) The stiffness value for the suspension (10: Offroad buggy, 50: Sports car, 200: F1 Car)

SuspensionCompression (optional) The damping coefficient to use when the suspension is compressed. Sets to value * 2 * SquareRoot(SuspensionStiffness), so it is proportional to critical damping. Value examples:

```
value = 0: undamped & bouncy  
value = 1: critical damping
```

Recommended values are from 0.1 to 0.3.

SuspensionDamping (optional) The damping when the suspension is expanding. See the SuspensionCompression to know how to set this value. SuspensionDamping should be slightly larger than SuspensionCompression. Recommended values are from 0.2 to 0.5.

MaxSuspensionCompression (optional) The length the suspension can be compressed (in centimeters).

Friction (optional) The friction between the tyre and the ground. Should be about 0.8 for realistic cars, but can increased for better handling. A large value (10000.0) can be used for kart racers like handling.

Return value

None.

See Also

FreeEntityBody()

178.7 GetVehicleAttribute

Syntax

```
Result.f = GetVehicleAttribute(#Entity, Attribute, Wheel)
```

Description

Get the specified attribute of the given vehicle entity.

Parameters

#Entity The vehicle entity to use.

Attribute The attribute to get. The following attributes are available:

```
#PB_Vehicle_Friction: get the wheel friction value (see
CreateVehicleBody()
for more info).
#PB_Vehicle_MaxSuspensionForce: get the wheel maximum
suspension force value (see CreateVehicleBody()
for more info).
#PB_Vehicle_SuspensionStiffness: get the stiffness value for
the suspension (see CreateVehicleBody()
for more info).
#PB_Vehicle_MaxSuspensionCompression: get the wheel maximum
suspension compression value (see CreateVehicleBody()
for more info).
#PB_Vehicle_MaxSuspensionLength: get the maximum length of
the suspension (meters).
#PB_Vehicle_WheelDampingCompression: get the wheel damping
compression value.
#PB_Vehicle_WheelDampingRelaxation: get the wheel damping
relaxation value.
#PB_Vehicle_RollInfluence: get the wheel roll influence value
(see AddVehicleWheel())
for more info).
#PB_Vehicle_IsInContact: return #True if the vehicle is in
contact with another object, #False otherwise.
#PB_Vehicle_CurrentSpeed : get the current vehicle speed in
Km/Hour.
#PB_Vehicle_ContactPointX: get the X contact point coordinate.
#PB_Vehicle_ContactPointY: get the Y contact point coordinate.
#PB_Vehicle_ContactPointZ: get the Z contact point coordinate.
#PB_Vehicle_ContactPointNormalX: get the X normal value of
the contact point.
#PB_Vehicle_ContactPointNormalY: get the Y normal value of
the contact point.
#PB_Vehicle_ContactPointNormalZ: get the Z normal value of
the contact point.
#PB_Vehicle_ForwardVectorX: get the X forward vector value of
the contact point.
#PB_Vehicle_ForwardVectorY: get the Y forward vector value of
the contact point.
#PB_Vehicle_ForwardVectorZ: get the Z forward vector value of
the contact point.
```

Wheel The wheel index, starting from 0.

Return value

Returns the value of the specified attribute or 0 if the vehicle does not support the attribute.

See Also

SetVehicleAttribute()

178.8 SetVehicleAttribute

Syntax

```
SetVehicleAttribute(#Entity, Attribute, Value.f [, Wheel])
```

Description

Set the specified attribute value to the given vehicle entity. For more info about attributes possible value, .

Parameters

#Entity The vehicle entity to use.

Attribute The attribute to set. The following attributes are available:

```
#PB_Vehicle_Friction: set the wheel friction value (see
CreateVehicleBody()
for more info).
#PB_Vehicle_MaxSuspensionForce: set the wheel maximum
suspension force value (see CreateVehicleBody()
for more info).
#PB_Vehicle_SuspensionStiffness: set the stiffness value for
the suspension (see CreateVehicleBody()
for more info).
#PB_Vehicle_MaxSuspensionCompression: set the wheel maximum
suspension compression value (see CreateVehicleBody()
for more info).
#PB_Vehicle_MaxSuspensionLength: set the maximum length of
the suspension (meters).
#PB_Vehicle_WheelDampingCompression: set the wheel damping
compression value.
#PB_Vehicle_WheelDampingRelaxation: set the wheel damping
relaxation value.
#PB_Vehicle_RollInfluence: set the wheel roll influence value
(see AddVehicleWheel()
for more info).
```

Value Value of the attribute to set.

Wheel (optional) The wheel index, starting from 0. If not specified, or sets to `#PB_All`, the new attribute value is applied to all wheels.

Return value

None.

See Also

GetVehicleAttribute()

Chapter 179

VertexAnimation

Overview

Vertex animation allows to handle mesh pose animations. The mesh must contain a list of predefined poses.

179.1 CreateVertexAnimation

Syntax

```
Result = CreateVertexAnimation(#Mesh, Animation$, Length)
```

Description

Creates a new vertex animation on the specified mesh.

Parameters

#Mesh The mesh to use to create the new animation.

Animation\$ The new animation name. This name is case-sensitive.

Length The length of new animation (in milliseconds).

Return value

Returns nonzero if the animation has been successfully created, zero otherwise.

See Also

CreateVertexTrack()

179.2 CreateVertexTrack

Syntax

```
Result = CreateVertexTrack(#Mesh, Animation$, Index)
```

Description

Creates a new vertex animation track on the specified mesh. The animation should already be created with CreateVertexAnimation() , or be predefined in the mesh. Every track has the same length, as defined in CreateVertexAnimation() .

Parameters

#Mesh The mesh to use to create the new animation track.

Animation\$ The animation name. This name is case-sensitive. The animation should already be created with `CreateVertexAnimation()` .

Index The new index of the track (should start from zero).

Return value

Returns nonzero if the animation track has been successfully created, zero otherwise.

See Also

`CreateVertexAnimation()`

179.3 CreateVertexPoseKeyFrame

Syntax

```
Result = CreateVertexPoseKeyFrame(#Mesh, Animation$, Track, Time)
```

Description

Creates a new keyframe in the vertex animation. The animation should already be created with `CreateVertexAnimation()` , or be predefined in the mesh.

Parameters

#Mesh The mesh to use.

Animation\$ The animation name. This name is case-sensitive. The animation should already be created with `CreateVertexAnimation()` .

Track The track index. The track has to be previously created with `CreateVertexTrack()` .

Time The time in the animation to set the keyframe (in milliseconds). This value has to be between zero and the 'Length' defined in `CreateVertexAnimation()` .

Return value

Returns the new keyframe index.

See Also

`CreateVertexAnimation()` , `CreateVertexTrack()`

179.4 AddVertexPoseReference

Syntax

```
AddVertexPoseReference(#Mesh, Animation$, Track, Keyframe,  
PoseIndex, Influence)
```

Description

Adds a new pose reference to the animation. The animation should already be created with `CreateVertexAnimation()` .

Parameters

#Mesh The mesh to use.

Animation\$ The animation name. This name is case-sensitive. The animation should already be created with `CreateVertexAnimation()` .

Track The track index. The track has to be previously created with `CreateVertexTrack()` .

Keyframe The keyframe index. The keyframe has to be previously created with `CreateVertexPoseKeyFrame()` .

PoseIndex The pose index. The first pose index starts from zero.

Influence The influence of the pose. This value ranges from 0.0 (no influence) to 1.0 (full influence).

Return value

None.

See Also

`CreateVertexAnimation()` , `CreateVertexTrack()`

179.5 UpdateVertexPoseReference

Syntax

```
UpdateVertexPoseReference(#Mesh, Animation$, Track, Keyframe,  
PoseIndex, Influence)
```

Description

Updates a new pose reference to the animation. The animation should already be created with `CreateVertexAnimation()` , or be predefined in the mesh.

Parameters

#Mesh The mesh to use.

Animation\$ The animation name. This name is case-sensitive. The animation should already be created with `CreateVertexAnimation()` , or be predefined in the mesh.

Track The track index. The track has to be previously created with `CreateVertexTrack()` .

Keyframe The keyframe index. The keyframe has to be previously created with `CreateVertexPoseKeyFrame()` .

PoseIndex The pose index. The first pose index starts from zero.

Influence The new influence of the pose. This value ranges from 0.0 (no influence) to 1.0 (full influence).

Return value

None.

See Also

CreateVertexAnimation() , CreateVertexTrack()

179.6 VertexPoseReferenceCount

Syntax

```
Result = VertexPoseReferenceCount(#Mesh, Animation$, Track,  
    Keyframe)
```

Description

Returns the number of pose references in the specified keyframe. The animation should already be created with CreateVertexAnimation() .

Parameters

#Mesh The mesh to use.

Animation\$ The animation name. This name is case-sensitive. The animation should already be created with CreateVertexAnimation() , or be predefined in the mesh.

Track The track index. The track has to be previously created with CreateVertexTrack() .

Keyframe The keyframe index. The keyframe has to be previously created with CreateVertexPoseKeyFrame() .

Return value

Returns the number of pose references in the specified keyframe.

See Also

CreateVertexAnimation() , CreateVertexTrack() , AddVertexPoseReference()

179.7 MeshPoseCount

Syntax

```
Result = MeshPoseCount(#Mesh)
```

Description

Returns the number of pose's in the mesh. A pose is a predefined vertex animation in the mesh.

Parameters

#Mesh The mesh to use.

Return value

Returns the number of pose's in the specified mesh.

See Also

MeshPoseName() , AddVertexPoseReference()

179.8 MeshPoseName

Syntax

```
Result\$_ = MeshPoseName(#Mesh, PoseIndex)
```

Description

Returns the pose name in the mesh. A pose is a predefined vertex animation in the mesh.

Parameters

#Mesh The mesh to use.

PoseIndex The index of the pose to get the name of. The first pose index starts from zero. The index has to be lower than the result of MeshPoseCount() .

Return value

Returns the pose name at the specified index.

See Also

MeshPoseName() , AddVertexPoseReference()

Chapter 180

WebView

Overview

The WebView library is a modern web browser gadget which can be used to easily create HTML/CSS UI in PureBasic. It allows to have rich cross-platform UI which behaves and looks the same, while having access to advanced CSS functions to enhance the user experience. The component is directly embedded in the PureBasic program for minimal footprint and using native callbacks to interact between the web content (which is JavaScript) and the PureBasic code. The WebView gadget is leveraging the built-in OS web components (WebView2 on Windows, WebKit-GTK on Linux/Raspberry and WebKit on macOS). On Windows 10 and above, no additional component is required but if older Windows support is needed, the WebView2 redistributable might need to be installed: [distribution guide](#).

As the WebView component is a PureBasic gadget library, standard commands like `ResizeGadget()` or `HideGadget()` can be used to control it.

180.1 WebViewGadget

Syntax

```
Result = WebViewGadget(#Gadget, x, y, Width, Height [, Flags])
```

Description

Creates a new web view gadget in the current GadgetList. If needed a proxy can be set with `WebViewProxy()`.

Parameters

#Gadget A number to identify the new gadget. `#PB_Any` can be used to auto-generate this number.

x, y, Width, Height The position and dimensions of the new gadget.

Flags (optional) Flags to modify the gadget behavior. It can be composed (using the '||' operator) of one of the following constants:

```
#PB_WebView_Debug: Enable the 'inspect' menu item in the
                    context menu to display the web inspector and console.
```

Return value

Nonzero on success, zero otherwise. If `#PB_Any` was used as the `#Gadget` parameter then the return-value is the auto-generated gadget number on success.

Remarks

After creation, use `BindWebViewCallback()` to interact with the JavaScript code of the UI. In addition common gadget commands like `ResizeGadget()` or `HideGadget()` may be used with the control as well.

The following functions can be used to act on a `WebViewGadget`:

- `SetGadgetText()` : Change the current URL. It can be a local file URL like `'file://c:/purebasic/svn/webview.html'` or a regular HTTP URL. When using a local file URL, it needs to be the full pathname. You can easily construct the local URL by adding `'file://'` to the pathname.
- `SetGadgetItemText()` : With `#PB_WebView_HtmlCode` as 'Item' html code can be streamed into the gadget. The html is treated as untrusted local content and some permission are not allowed.

Example: using local HTML and CSS files

```
1  Procedure IncrementJS(JsonParameters$)
2      Static i
3      Debug "IncrementJS "+JsonParameters$
4      i+1
5      ProcedureReturn UTF8(~"{ \"count\": "+Str(i)+ "}")
6  EndProcedure
7
8  Procedure ComputeJS(JsonParameters$)
9      Debug "ComputeJS "+JsonParameters$
10     ProcedureReturn UTF8(~"150")
11 EndProcedure
12
13 OpenWindow(0, 100, 100, 400, 400, "Hello", #PB_Window_SystemMenu)
14
15 WebViewGadget(0, 0, 0, 400, 400)
16 SetGadgetText(0, "file://" + #PB_Compiler_Home +
17     "examples/sources/Data/WebView/webview.html")
18
19 BindWebViewCallback(0, "increment", @IncrementJS())
20 BindWebViewCallback(0, "compute", @ComputeJS())
21
22 Repeat
23     Event = WaitWindowEvent()
```

Example: using inline HTML code

```
1  Procedure IncrementJS(JsonParameters$)
2      Static i
3      Debug "IncrementJS "+JsonParameters$
4      i+1
5      ProcedureReturn UTF8(~"{ \"count\": "+Str(i)+ "}")
6  EndProcedure
7
8  Procedure ComputeJS(JsonParameters$)
9      Debug "ComputeJS "+JsonParameters$
10     ProcedureReturn UTF8(~"150")
11 EndProcedure
12
13 OpenWindow(0, 100, 100, 400, 400, "Hello", #PB_Window_SystemMenu)
14
```

```

15  WebViewGadget(0, 0, 0, 400, 400)
16
17  SetGadgetItemText(0, #PB_WebView_HtmlCode,
18      ~"<button id=\"increment\">Tap me</button>\n"+
19      ~"<div>You tapped <span id=\"count\">0</span>
    time(s).</div>\n"+
20      ~"<button id=\"compute\">Compute</button>\n"+
21      ~"<div>Result of computation: <span
    id=\"compute-result\">0</span></div>\n"+
22      ~"<script>\n"+
23      ~"    const [incrementElement, countElement,
    computeElement, "+
24      ~"computeResultElement] =\n"+
25      ~"        document.querySelectorAll(\"#increment,
    #count, #compute, "+
26      ~"#compute-result\");\n"+
27      ~"
    document.addEventListener(\"DOMContentLoaded\", () => {\n"+
28      ~"
    incrementElement.addEventListener(\"click\", () => {\n"+
29      ~"        window.increment().then(result => {\n"+
30      ~"            countElement.textContent =
    result.count;\n"+
31      ~"        });\n"+
32      ~"    });\n"+
33      ~"
    computeElement.addEventListener(\"click\", () => {\n"+
34      ~"        computeElement.disabled = true;\n"+
35      ~"        window.compute(6, 7).then(result =>
    {\n"+
36      ~"            computeResultElement.textContent =
    result;\n"+
37      ~"            computeElement.disabled = false;\n"+
38      ~"        });\n"+
39      ~"    });\n"+
40      ~"    });\n"+
41      ~"</script>")
42
43  BindWebViewCallback(0, "increment", @IncrementJS())
44  BindWebViewCallback(0, "compute", @ComputeJS())
45
46  Repeat
47      Event = WaitWindowEvent()
48  Until Event = #PB_Event_CloseWindow

```

Example: with disabled right-click menu

```

1  Html$ = ~"<html style=\"margin:0px;height:100%;width:100%\">"+
2      ~"<body oncontextmenu=\"return false;\"
    style=\"margin:0px;min-height:100%;width:100%\">"+
3      ~"<button id=\"displayInfo\">Display Info</button>"+
4      ~"</body></html>\n"
5
6  OpenWindow(0, 100, 100, 400, 400, "No right click UI",
    #PB_Window_SystemMenu)
7
8  WebViewGadget(0, 0, 0, 400, 400)
9  SetGadgetItemText(0, #PB_WebView_HtmlCode, Html$)

```



```

10
11 Repeat
12     Event = WaitWindowEvent()
13 Until Event = #PB_Event_CloseWindow

```

See Also

WebViewExecuteScript() , BindWebViewCallback() , WebViewProxy()

180.2 BindWebViewCallback

Syntax

```
BindWebViewCallback(#Gadget, JavaScriptFunction$, @Callback())
```

Description

Bind a PureBasic callback to a new JavaScript function. The JavaScript function will be automatically created in the web view and will be available in the JavaScript code. When the JavaScript function will be called in the webview gadget, the PureBasic 'Callback' will be called. The JavaScript code will be halted until the PureBasic callback finish its execution.

Parameters

#Gadget The web view gadget to bind the event to.

JavaScriptFunction\$ The case-sensitive name of the JavaScript function to create and to bind the callback to.

@Callback() The callback procedure to call when the JavaScript function is called. It has to be declared like this:

```

1 Procedure JavaScriptCallback(JsonParameters$)
2     ;
3     ; 'JsonParameters$' contains all the parameters which were
    specified when the JavaScript function was called in JSON
    format.
4     ;
5
6     ; Code
7
8     ProcedureReturn UTF8(JsonResult$)
9 EndProcedure

```

The callback procedure can return a result which can be used by the JavaScript function.

Return value

None.

Remarks

If your JavaScript callback function returns a UTF8() generated result instead of 0, PureBasic will automatically free the memory used to return it when it is no longer needed.

Example: with a 2 parameters Javascript function

```
1      ; A simple button which will change its label by the result of
      the PureBasic callback
2      ;
3      Html$ = ~"<button id=\"displayInfo\">Display Info</button>\n"+
4              ~"<script>\n"+
5              ~"    const
      displayInfoElement=document.getElementById(\"displayInfo\");\n"+
6              ~"    document.addEventListener(\"DOMContentLoaded\", ()
      => {\n"+
7              ~"        displayInfoElement.addEventListener(\"click\", ()
      => {\n"+
8              ~"            window.displayInfo(1000, 2000).then(result =>
      {\n"+
9              ~"                displayInfoElement.textContent =
      result.sum;\n"+
10             ~"            });\n"+
11             ~"        });\n"+
12             ~"    });\n"+
13             ~"</script>";
14
15      Procedure IncrementJS(JsonParameters$)
16
17          Dim Parameters(0)
18
19          ParseJSON(0, JsonParameters$)
20          ExtractJSONArray(JSONValue(0), Parameters())
21
22          Debug "Nb Parameters: " + ArraySize(Parameters())
23
24          Debug "Parameter 1: " + Parameters(0)
25          Debug "Parameter 2: " + Parameters(1)
26
27          ProcedureReturn UTF8(~"{ \"sum\": "+Str(Parameters(0) +
      Parameters(1))+ "}")
28      EndProcedure
29
30
31      OpenWindow(0, 100, 100, 400, 400, "Hello", #PB_Window_SystemMenu)
32
33      WebViewGadget(0, 0, 0, 400, 400)
34      SetGadgetItemText(0, #PB_WebView_HtmlCode, Html$)
35
36      BindWebViewCallback(0, "displayInfo", @IncrementJS())
37
38      Repeat
39          Event = WaitWindowEvent()
40      Until Event = #PB_Event_CloseWindow
```

See Also

WebViewGadget() , UnbindWebViewCallback()

180.3 UnbindWebViewCallback

Syntax

```
UnbindWebViewCallback(#Gadget, JavaScriptFunction$)
```

Description

Unbind a JavaScript function previously bound with BindWebViewCallback(). The JavaScript function will be automatically removed from the web view and won't be available anymore in the JavaScript code.

Parameters

#Gadget The web view gadget to use.

JavaScriptFunction\$ The case-sensitive name of the JavaScript function to unbind.

Return value

None.

See Also

WebViewGadget(), BindWebViewCallback()

180.4 WebViewExecuteScript

Syntax

```
WebViewExecuteScript(#Gadget, JavaScript$)
```

Description

Executes a JavaScript expression asynchronously in the web view.

Parameters

#Gadget The web view gadget to use.

JavaScript\$ The JavaScript expression to execute.

Return value

None.

Example

```
1  Html$ = ~"<button id=\"displayInfo\">Display Info</button>\n"+
2          ~"<script>\n"+
3          ~"    const
4          displayInfoElement=document.getElementById(\"displayInfo\");\n"+
          ~"    document.addEventListener(\"DOMContentLoaded\", ()
          => {\n"+
```

```

5      ~"      displayInfoElement.addEventListener(\"click\", ()
=> {\n"+
6      ~"      window.displayInfo(1000, 2000).then(result =>
{\n"+
7      ~"      });\n"+
8      ~"      });\n"+
9      ~"      });\n"+
10     ~"</script>";
11
12     Procedure DisplayInfoCallback(JsonParameters$)
13         ; Execute a script in the webview
14         WebViewExecuteScript(0, ~"const
element=document.getElementById(\"displayInfo\");
element.textContent=\"Hello from Callback !\";");
15     EndProcedure
16
17     OpenWindow(0, 100, 100, 400, 400, "Hello", #PB_Window_SystemMenu)
18
19     WebViewGadget(0, 0, 0, 400, 400)
20     SetGadgetItemText(0, #PB_WebView_HtmlCode, Html$)
21
22     BindWebViewCallback(0, "displayInfo", @DisplayInfoCallback())
23
24     Repeat
25         Event = WaitWindowEvent()
26     Until Event = #PB_Event_CloseWindow

```

See Also

WebViewGadget()

180.5 WebViewProxy

Syntax

```
WebViewProxy(URL$, Port [Username$, Password$])
```

Description

Setup a proxy for all the future WebViewGadget() created. This means than all the external link and URL will goes trough this proxy before reaching the destination. It can be useful in a company environment where security level is high.

Parameters

URL\$ The proxy server URL.

Port The proxy server port.

Username\$, Password\$ (optional) If the proxy server requires authentication, a username and password can be set.

Return value

None.

Remarks

When used on MacOS, the minimum requirement is MacOS 14 (Sonoma).

See Also

`WebViewGadget()`

Chapter 181

Window

Overview

Windows are essential components of modern interfaces. PureBasic provides full access to them.

181.1 AddKeyboardShortcut

Syntax

```
AddKeyboardShortcut(#Window, Shortcut, Event)
```

Description

Add or replace a keyboard shortcut to the specified window. A shortcut generates a menu event (like a menu item) as most of them are used in conjunction with menus.

Parameters

#Window The window to use.

Shortcut It can be one of the following constants:

```
#PB_Shortcut_Back  
#PB_Shortcut_Tab  
#PB_Shortcut_Clear  
#PB_Shortcut_Return  
#PB_Shortcut_Menu  
#PB_Shortcut_Pause  
#PB_Shortcut_Print  
#PB_Shortcut_Capital  
#PB_Shortcut_Escape  
#PB_Shortcut_Space  
#PB_Shortcut_PageUp  
#PB_Shortcut_PageDown  
#PB_Shortcut_End  
#PB_Shortcut_Home  
#PB_Shortcut_Left  
#PB_Shortcut_Up  
#PB_Shortcut_Right  
#PB_Shortcut_Down  
#PB_Shortcut_Select  
#PB_Shortcut_Execute  
#PB_Shortcut_Snapshot  
#PB_Shortcut_Insert
```

#PB_Shortcut_Delete
#PB_Shortcut_Help
#PB_Shortcut_0
#PB_Shortcut_1
#PB_Shortcut_2
#PB_Shortcut_3
#PB_Shortcut_4
#PB_Shortcut_5
#PB_Shortcut_6
#PB_Shortcut_7
#PB_Shortcut_8
#PB_Shortcut_9
#PB_Shortcut_A
#PB_Shortcut_B
#PB_Shortcut_C
#PB_Shortcut_D
#PB_Shortcut_E
#PB_Shortcut_F
#PB_Shortcut_G
#PB_Shortcut_H
#PB_Shortcut_I
#PB_Shortcut_J
#PB_Shortcut_K
#PB_Shortcut_L
#PB_Shortcut_M
#PB_Shortcut_N
#PB_Shortcut_O
#PB_Shortcut_P
#PB_Shortcut_Q
#PB_Shortcut_R
#PB_Shortcut_S
#PB_Shortcut_T
#PB_Shortcut_U
#PB_Shortcut_V
#PB_Shortcut_W
#PB_Shortcut_X
#PB_Shortcut_Y
#PB_Shortcut_Z
#PB_Shortcut_LeftWindows
#PB_Shortcut_RightWindows
#PB_Shortcut_Apps
#PB_Shortcut_Pad0
#PB_Shortcut_Pad1
#PB_Shortcut_Pad2
#PB_Shortcut_Pad3
#PB_Shortcut_Pad4
#PB_Shortcut_Pad5
#PB_Shortcut_Pad6
#PB_Shortcut_Pad7
#PB_Shortcut_Pad8
#PB_Shortcut_Pad9
#PB_Shortcut_Multiply
#PB_Shortcut_Add
#PB_Shortcut_Separator
#PB_Shortcut_Subtract
#PB_Shortcut_Decimal
#PB_Shortcut_Divide
#PB_Shortcut_F1
#PB_Shortcut_F2

```

#PB_Shortcut_F3
#PB_Shortcut_F4
#PB_Shortcut_F5
#PB_Shortcut_F6
#PB_Shortcut_F7
#PB_Shortcut_F8
#PB_Shortcut_F9
#PB_Shortcut_F10
#PB_Shortcut_F11
#PB_Shortcut_F12
#PB_Shortcut_F13
#PB_Shortcut_F14
#PB_Shortcut_F15
#PB_Shortcut_F16
#PB_Shortcut_F17
#PB_Shortcut_F18
#PB_Shortcut_F19
#PB_Shortcut_F20
#PB_Shortcut_F21
#PB_Shortcut_F22
#PB_Shortcut_F23
#PB_Shortcut_F24
#PB_Shortcut_Numlock
#PB_Shortcut_Scroll

```

The above key can be combined with any of the following constants:

```

#PB_Shortcut_Shift
#PB_Shortcut_Control
#PB_Shortcut_Alt
#PB_Shortcut_Command

```

Event The number which will be returned by the EventMenu() function. This value has a limited range, from 0 to 64000. By default, a window already has the `#PB_Shortcut_Tab` and `#PB_Shortcut_Tab|#PB_Shortcut_Shift` shortcuts to handle tab and shift-tab correctly through the gadgets . A shortcut can be removed with RemoveKeyboardShortcut() .

Return value

None.

Remarks

The `#PB_Shortcut_Command` constant is only useful on Mac OSX and allow to use the 'Apple' key (left or right) to define shortcuts. This constant is also supported on others OS (to ease portability), but will act like `#PB_Shortcut_Control`. The shortcuts Apple+Q and Apple+P, are predefined on Mac OSX for the `#PB_Menu_Quit` and `#PB_Menu_Preferences` menu entries in the application menu and cannot be reassigned. MDI child windows are not supported.

Example

```

1  AddKeyboardShortcut(0, #PB_Shortcut_Control | #PB_ShortCut_F, 15)
   ; Will create a keyboard shortcut CTRL+F on the window 0
2
   which will fires a menu event '15'

```


Example: Extended example with keyboard shortcuts for menu items and standalone

```
1  #Window = 0
2  Enumeration Menu
3      #Menu
4      #PopupMenu
5  EndEnumeration
6  Enumeration Menu_items
7      #mOpen
8      #mCopy
9      #mDummy
10 EndEnumeration
11
12 If OpenWindow(#Window, 200, 200, 200, 100, "Press Ctrl+D")
13     If CreateMenu(#Menu, WindowID(#Window)) ; Create a regular
        menu with title and one item
14         MenuTitle("File")
15         MenuItem(#mOpen, "Open" + #TAB$ + "Ctrl+O")
16     EndIf
17     If CreatePopupMenu(#PopupMenu) ; Create an additional pop-up
        menu
18         MenuItem(#mCopy, "Copy" + #TAB$ + "Ctrl+Shift+C")
19     EndIf
20     AddKeyboardShortcut(#Window, #PB_Shortcut_Control |
        #PB_Shortcut_D, #mDummy) ; keyboard shortcut standalone
        (without menu item)
21     AddKeyboardShortcut(#Window, #PB_Shortcut_Control |
        #PB_Shortcut_O, #mOpen) ; shortcut for the menu item
22     AddKeyboardShortcut(#Window, #PB_Shortcut_Control |
        #PB_Shortcut_Shift | #PB_Shortcut_C, #mCopy) ; shortcut for the
        pop-up menu item
23     Repeat
24         Select WaitWindowEvent()
25             Case #PB_Event_RightClick ; Display the pop-up menu on
                right mouse-click
26                 DisplayPopupMenu(#PopupMenu, WindowID(#Window))
27             Case #PB_Event_Menu
28                 Select EventMenu()
29                     Case #mDummy : Debug "Dummy"
30                     Case #mOpen : Debug "Open"
31                     Case #mCopy : Debug "Copy"
32                 EndSelect
33             Case #PB_Event_CloseWindow
34             End
35         EndSelect
36     ForEver
37 EndIf
```

See Also

RemoveKeyboardShortcut()

181.2 AddWindowTimer

Syntax

```
AddWindowTimer(#Window, Timer, Timeout)
```

Description

Adds a new timer to the specified window. This will cause `#PB_Event_Timer` events to be received periodically in the `WindowEvent()` or `WaitWindowEvent()` functions. The `RemoveWindowTimer()` function can be used to remove the timer again.

Parameters

#Window The window to use. A timer is always attached to a window and will be removed if that window is closed.

Timer An user-defined number that identifies this timer. Timers on separate windows may have overlapping numbers. This value will later be returned from `EventTimer()` when a `#PB_Event_Timer` is received. It can also be used to remove the timer again with the `RemoveWindowTimer()` function.

Timeout Specifies the amount of time (in milliseconds) between each `#PB_Event_Timer` events. The timer events will only be generated when there are no other events to be processed (timers are low-priority events). This means that the time that elapses between two timer events may be larger than the specified `Timeout` value. Timers are therefore not suited for precise timing but are rather intended to perform periodic tasks such as updating a gadget content or similar.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 400, 100, "Timer Example",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      ProgressBarGadget(0, 10, 10, 380, 20, 0, 100)
4      AddWindowTimer(0, 123, 250)
5
6      Value = 0
7      Repeat
8          Event = WaitWindowEvent()
9
10         If Event = #PB_Event_Timer And EventTimer() = 123
11             Value = (Value + 5) % 100
12             SetGadgetState(0, Value)
13         EndIf
14
15     Until Event = #PB_Event_CloseWindow
16 EndIf
```

Remarks

To change the timer duration you have to remove the timer first, then add the same timer with a new 'Timeout' value again:

```
1  RemoveWindowTimer(#Window, Timer)
2  AddWindowTimer(#Window, Timer, Timeout)
```

See Also

`RemoveWindowTimer()` , `EventTimer()`

181.3 RemoveWindowTimer

Syntax

```
RemoveWindowTimer(#Window, Timer)
```

Description

Removes the timer from the specified window.

Parameters

#Window The window to use.

Timer The same value that was used in `AddWindowTimer()` to create the timer. There will be no further events received for this timer.

Return value

None.

See Also

`AddWindowTimer()`

181.4 EventTimer

Syntax

```
Timer = EventTimer()
```

Description

After an event of type `#PB_Event_Timer` (returned by `WindowEvent()` or `WaitWindowEvent()`), use this function to determine which timer caused the event.

Parameters

None.

Return value

The same value that was used in `AddWindowTimer()` to create the timer.

See Also

`AddWindowTimer()`

181.5 CloseWindow

Syntax

```
CloseWindow(#Window)
```

Description

Close the specified window.

Parameters

#Window The window to close. If **#PB_All** is specified, all the remaining windows are closed.

Return value

None.

Remarks

All remaining opened windows are automatically closed when the program ends.

Note: The following items created for a window are also automatically freed when the window is closed: Gadgets , Keyboard shortcuts , Menus , StatusBars , Timers , Toolbars and bound events (with BindEvent()).

See Also

OpenWindow()

181.6 DisableWindow

Syntax

```
DisableWindow(#Window, State)
```

Description

Enables or disables user input to the specified Window.

Parameters

#Window The window to disable or enable.

State It can be one of the following values:

```
#True : the window is disabled.  
#False: The window is enabled.
```

Return value

None.

181.7 Event

Syntax

```
Event = Event()
```

Description

Return the current event. It is the same value returned by WindowEvent() and WaitWindowEvent() , it is mainly useful when using a callback to determine which event triggered it.

Parameters

None.

Return value

Returns the current event.

Example

```
1  Procedure EventHandler()  
2      Select Event()  
3          Case #PB_Event_CloseWindow  
4              End  
5  
6          Case #PB_Event_Gadget  
7              Debug "Gadget #" + EventGadget() + " pushed"  
8          EndSelect  
9  EndProcedure  
10  
11  OpenWindow(0, 100, 100, 300, 100, "", #PB_Window_SizeGadget |  
12      #PB_Window_SystemMenu | #PB_Window_MaximizeGadget)  
13  ButtonGadget(0, 10, 10, 100, 30, "Push me")  
14  
15  ; Use a single callback for all events  
16  BindGadgetEvent(0, @EventHandler())  
17  BindEvent(#PB_Event_CloseWindow, @EventHandler())  
18  
19  ; We don't process events here, so we can run the loop forever  
20  ; and forget about it  
21  Repeat  
22      WaitWindowEvent()  
23  ForEver
```

See Also

WindowEvent() , WaitWindowEvent()

181.8 EventGadget

Syntax

```
GadgetNumber = EventGadget()
```

Description

After an event of type `#PB_Event_Gadget` (returned by `WindowEvent()` or `WaitWindowEvent()`), use this function to determine which gadget has been triggered.

Parameters

None.

Return value

Returns the `#Gadget` number associated to the event.

See Also

`WindowEvent()` , `WaitWindowEvent()`

181.9 EventMenu

Syntax

```
MenuItem = EventMenu()
```

Description

After an event of type `#PB_Event_Menu` (returned by `WindowEvent()` or `WaitWindowEvent()`), use this function to determine which menu item, toolbar item or keyboard shortcut has been selected.

Parameters

None.

Return value

Returns the menu item, toolbar item or keyboard shortcut number associated to the event.

Remarks

A toolbar event is like a menu event (as toolbars are shortcuts for menu items most of the time). So it's a good idea, if the toolbar buttons and the menu items have the same ID, then the same operation can be done on both without any additional code.

See Also

`WindowEvent()` , `WaitWindowEvent()`

181.10 **EventData**

Syntax

```
Data = EventData()
```

Description

Get the data associated with the current event. The event needs to be a custom event sent with `PostEvent()` .

Parameters

None.

Return value

Returns the value associated with the current event. If the current event isn't a custom event sent with `PostEvent()` , this value is undefined.

See Also

`PostEvent()` , `WindowEvent()`

181.11 **EventType**

Syntax

```
EventType = EventType()
```

Description

After a `WindowEvent()` or `WaitWindowEvent()` function, use this function to determine of which type the event is. The following gadgets support `EventType()`:

- `CanvasGadget()` - The `CanvasGadget` has a special set of event types.
- `ComboBoxGadget()`
- `DateGadget()`
- `EditorGadget()`
- `ExplorerListGadget()`
- `ExplorerTreeGadget()`
- `ImageGadget()`
- `ListViewGadget()`
- `ListIconGadget()`
- `MDIGadget()`
- `OpenGLGadget()`
- `SpinGadget()`
- `StringGadget()`
- `TreeGadget()`
- `WebGadget()` - The `WebGadget` has a special set of event types.
(See the gadget definition to see which events are supported.)

Parameters

None.

Return value

The following values are possible, if an event of the type `#PB_Event_Gadget` (library `Gadget`) or `#PB_Event_SysTray` (library `SysTray`) occurs:

<code>#PB_EventType_LeftClick</code>	: Left mouse button click
<code>#PB_EventType_RightClick</code>	: right mouse button click
<code>#PB_EventType_LeftDoubleClick</code>	: Left mouse button double click
<code>#PB_EventType_RightDoubleClick</code>	: Right mouse button double click
<code>#PB_EventType_Focus</code>	: Get the focus.
<code>#PB_EventType_LostFocus</code>	: Lose the focus.
<code>#PB_EventType_Change</code>	: Content change.
<code>#PB_EventType_DragStart</code>	: The user tries to start a Drag & Drop operation.

Example

```
1  If OpenWindow(0, 0, 0, 230, 120, "Eventtypes example...",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      ListIconGadget(1, 10, 10, 150, 100, "ListIcon", 140,
4      #PB_ListIcon_GridLines)
5      For a = 1 To 4
6          AddGadgetItem(1, -1, "Line "+Str(a))
7      Next
8      Repeat
9          Event = WaitWindowEvent()
10         Select Event
11             Case #PB_Event_Gadget
12                 Select EventGadget()
13                     Case 1
14                         Select EventType()
15                             Case #PB_EventType_LeftClick           : Debug "Click
16 with left mouse button"
17                             Case #PB_EventType_RightClick          : Debug "Click
18 with right mouse button"
19                             Case #PB_EventType_LeftDoubleClick      : Debug
20 "Double-click with left mouse button"
21                             Case #PB_EventType_RightDoubleClick     : Debug
22 "Double-click with right mouse button"
23                         EndSelect
24                     EndSelect
25                 EndSelect
26             Until Event = #PB_Event_CloseWindow
27         EndIf
```

See Also

`WaitWindowEvent()` , `WindowEvent()`

181.12 EventWindow

Syntax

```
WindowNumber = EventWindow()
```

Description

After a WindowEvent() or WaitWindowEvent() function, use this function to determine on which window the event has occurred.

Parameters

None.

Return value

The window number on which the event has occurred.

See Also

WaitWindowEvent() , WindowEvent()

181.13 GetActiveWindow

Syntax

```
WindowNumber = GetActiveWindow()
```

Description

Returns the number of the window which currently has the keyboard focus or -1 if no window within the program is active.

Parameters

None.

Return value

The number of the window which currently has the keyboard focus or -1 if no window is active.

Remarks

The function will only report the window ID within the program. A window within the program can be activated (set the focus on it) with the SetActiveWindow() function.

See Also

SetActiveWindow()

181.14 GetWindowColor

Syntax

```
Color = GetWindowColor(#Window)
```

Description

Returns the background color of the specified window that was set with SetWindowColor() .

Parameters

#Window The window to use.

Return value

The background color of the specified window that was set with SetWindowColor() . If no background color was set yet, -1 is returned.

See Also

SetWindowColor()

181.15 GetWindowData

Syntax

```
Result = GetWindowData(#Window)
```

Description

Returns the 'Data' value that has been stored for this window with the SetWindowData() function. This allows to associate a custom value with any window.

Parameters

#Window The window to use.

Return value

Returns the current data value. If there was never a data value set for this window, the return-value will be 0.

See Also

SetWindowData() , GetGadgetData() , SetGadgetData()

181.16 GetWindowState

Syntax

```
State = GetWindowState(#Window)
```

Description

Checks whether the specified window is maximized, minimized or displayed normally.

Parameters

#Window The window to use.

Return value

It can be one of the following values:

```
#PB_Window_Normal    : The window is neither maximized nor
                        minimized.
#PB_Window_Maximize  : The window is maximized.
#PB_Window_Minimize  : The window is minimized.
```

Remarks

The status of a window can be changed with the `SetWindowState()` function.

See Also

`SetWindowState()`

181.17 GetWindowTitle

Syntax

```
Result\$_ = GetWindowTitle(#Window)
```

Description

Returns the text currently displayed in the specified window's title bar.

Parameters

#Window The window to use.

Return value

The text currently displayed in the specified window's title bar.

Remarks

The title of a window can be changed with `SetWindowTitle()` .

Example

```
1  If OpenWindow(2, 100, 100, 200, 100, "My cool title")
2      Title$ = GetWindowTitle(2) ; Will return "My cool title"
3  EndIf
```

See Also

`SetWindowTitle()`

181.18 HideWindow

Syntax

```
HideWindow(#Window, State [, Flags])
```

Description

Hides or shows the specified window.

Parameters

#Window The window to use.

State It can be one of the following values:

```
#True : the window is hidden.  
#False: the window is shown. The window is automatically  
        activated (gets the focus),  
        unless the #PB_Window_NoActivate flag is set.
```

Flags It can be a combination of the following values:

```
#PB_Window_NoActivate      : the window will be shown but not  
        activated (only valid when un-hiding the window).  
#PB_Window_ScreenCentered: the window will be screen centered  
        (only valid when un-hiding the window).  
#PB_Window_WindowCentered: the window will be window centered  
        (only valid when un-hiding the window).
```

Return value

None.

Example

```
1  If OpenWindow(0, 200, 200, 220, 100, "HideWindow()",  
2      #PB_Window_SystemMenu)  
3  
4      ButtonGadget (1, 10, 60, 200, 30, "Hide the window")  
5  
6      Repeat  
7          Event = WaitWindowEvent()  
8          Select Event  
9  
10             Case #PB_Event_Gadget  
11                 Select EventGadget()  
12                     Case 1  
13  
14                         HideWindow(0, #True)  
15                         Debug "Window hidden."  
16                         MessageRequester("Info", "Press OK to show the window")  
17                         HideWindow(0, #False)  
18                         Debug "Window is visible."  
19  
20                     EndSelect  
21
```

```
22     EndSelect
23     Until Event = #PB_Event_CloseWindow
24 EndIf
```

See Also

OpenWindow()

181.19 IsWindow

Syntax

```
Result = IsWindow(#Window)
```

Description

Tests if the given #Window number is a valid and correctly initialized, window.

Parameters

#Window The window to use.

Return value

Returns nonzero if #Window is a valid window and zero otherwise.

Remarks

This function is bulletproof and can be used with any value. If the 'Result' is not zero then the object is valid and initialized, otherwise it will equal zero. This is the correct way to ensure a window is ready to use.

See Also

OpenWindow()

181.20 OpenWindow

Syntax

```
Result = OpenWindow(#Window, x, y, InnerWidth, InnerHeight, Title$  
[, Flags [, ParentID]])
```

Description

Opens a new window according to the specified parameters. The new window becomes the active window, it's not needed to use SetActiveWindow() (unless the window is created as invisible).

Parameters

#Window A number to identify the new window. **#PB_Any** can be used to auto-generate this number.

x, y The initial position of the window, in pixels (unless one of the center flags is used). If 'x' or 'y' is set to **#PB_Ignore**, the OS will choose a position for the window.

InnerWidth, InnerHeight The required client area, in pixels (without borders and window decorations).

Title\$ The title of the newly created window.

Flags (optional) Can be a combination of the following values:

```
#PB_Window_SystemMenu      : Enables the system menu on the
                             window title bar (default).
#PB_Window_MinimizeGadget : Adds the minimize gadget to the
                             window title bar. #PB_Window_SystemMenu is automatically
                             added.
#PB_Window_MaximizeGadget : Adds the maximize gadget to the
                             window title bar. #PB_Window_SystemMenu is automatically
                             added.
                             (MacOS only: #PB_Window_SizeGadget
                             will be also automatically added).
#PB_Window_SizeGadget      : Adds the sizeable feature to a
                             window.
#PB_Window_Invisible       : Creates the window but don't
                             display.
#PB_Window_TitleBar        : Creates a window with a titlebar.
#PB_Window_Tool            : Creates a window with no taskbar
                             entry.
#PB_Window_BorderLess      : Creates a window without any
                             borders.
#PB_Window_ScreenCentered : Centers the window in the middle
                             of the screen. x,y parameters are ignored.
#PB_Window_WindowCentered : Centers the window in the middle
                             of the parent window ('ParentID' parameter must be
                             specified).
                             x,y parameters are
                             ignored.
#PB_Window_Maximize        : Opens the window maximized. (Note:
                             on Linux, not all Windowmanagers support this)
#PB_Window_Minimize        : Opens the window minimized.
#PB_Window_NoGadgets       : Prevents the creation of a
                             GadgetList. UseGadgetList()
                             can be used to do this later.
#PB_Window_NoActivate      : Don't activate the window after
                             opening.
```

ParentID (optional) The parent window identifier. A valid window identifier can be retrieved with `WindowID()`.

Return value

Nonzero if the window was successfully created, zero otherwise. If **#PB_Any** was used for the **#Window** parameter then the generated number is returned on success.

Remarks

All possible events in a window are handled with the `WindowEvent()` and `WaitWindowEvent()` functions.

Windows only: for special situations callbacks are used, see description at `SetWindowCallback()` .

On Windows, the height and width of a window with a titlebar can't be less than around 100 pixels. To open smaller windows, use the "`#PB_Window_BorderLess`" flag. On Windows, the `#PB_Window_Maximize` flag without the maximize gadget will get the full screen size covering the task bar.

Important: A window should not be opened in a thread , as there is some limitation on OS X and Linux. A debugger error will be raised.

See Also

`CloseWindow()`

181.21 PostEvent

Syntax

```
Result = PostEvent(Event [, Window, Object [, Type [, Data]])
```

Description

Posts an event at the end of the internal event queue and continues program execution without waiting for message processing.

Parameters

Event The event to post. For a list of PureBasic event, see `WindowEvent()` . When using custom event, the first value must be at least `#PB_Event_FirstCustomValue`, to not clash with internal events.

Window (optional) The window number associated with the event. When using a custom event, it can be any integer number. This value can be retrieved with `EventWindow()` .

Object (optional) The object number associated with the event. It can be for example a gadget or menu number. When using a custom event, it can be any positive integer number (zero included). This value can be retrieved with `EventGadget()` .

Type (optional) The type associated with the event. When using custom event, the first value must be at least `#PB_EventType_FirstCustomValue`, to not clash with internal values. This value can be retrieved with `EventType()` .

Data (optional) A data associated with the event. It is only valid when using a custom event and can be any integer number. This value can be retrieved with `EventData()` .

Return value

Returns a non-zero value if the event was sent without error, zero otherwise.

Remarks

This command can be very useful to communicate between threads and the main event loop. For example, a thread can send a custom event when it has finished its processing (with an associated data), so the main loop will get it and further processing can be done.

All PureBasic generated events are always handled before systems events (so as you add a new event without processing others it always comes first).

```

1  ; All our custom events
2  Enumeration #PB_Event_FirstCustomValue
3      #EventBeginProcessing
4      #EventProcessingFinished
5  EndEnumeration
6
7
8  Procedure Thread(Value)
9      PostEvent(#EventBeginProcessing)
10
11     Delay(3000)
12     PostEvent(#EventProcessingFinished)
13 EndProcedure
14
15 OpenWindow(0, 200, 200, 100, 100, "PostEvent")
16
17 CreateThread(@Thread(), 0)
18
19 Repeat
20     Event = WaitWindowEvent()
21
22     Select Event
23     Case #EventBeginProcessing
24         Debug "Thread begin processing "
25
26     Case #EventProcessingFinished
27         Debug "Thread processing finished"
28     EndSelect
29
30 Until Event = #PB_Event_CloseWindow

```

See Also

WindowEvent() , EventWindow() , EventGadget() , EventType() , EventData()

181.22 RemoveKeyboardShortcut

Syntax

```
RemoveKeyboardShortcut(#Window, Shortcut)
```

Description

Removes a keyboard shortcut previously defined with AddKeyboardShortcut() from the specified #Window.

Parameters

#Window The window to use.

Shortcut The shortcut to remove. For a full list of the available shortcut, see AddKeyboardShortcut() . If this parameter is set to #PB_Shortcut_All, all the shortcuts are removed from the specified window.

Return value

None.

Example

```
1 RemoveKeyboardShortcut(0, #PB_Shortcut_All) ; Remove all keyboard  
   shortcuts of the window 0
```

See Also

AddKeyboardShortcut()

181.23 ResizeWindow

Syntax

```
ResizeWindow(#Window, x, y, Width, Height)
```

Description

Move and resize the given window to the given position and size. If any of the parameters should be ignored (not be changed) `#PB_Ignore` can be passed at this place.

Parameters

#Window The window to resize.

x, y, Width, Height The new position and dimensions of the window, in pixel. `#PB_Ignore` can be passed as any parameter (x, y, Width or Height) and this parameter will not be changed.

Return value

None.

See Also

ResizeGadget() , BindEvent()

181.24 SetActiveWindow

Syntax

```
SetActiveWindow(#Window)
```

Description

Activates the specified window, which means the focus has been put on this window.

Parameters

#Window The window to activate.

Return value

None.

Remarks

The function will only change the focus within the program. It can not bring the program to the foreground when another program has the focus.

See Also

GetActiveWindow()

181.25 SetWindowCallback

Syntax

```
SetWindowCallback(@ProcedureName() [, #Window [, Mode]])
```

Description

For experienced programmers only. It's only supported on Microsoft Windows.

Normal events should be handled with the regular WaitWindowEvent() or WindowEvent() .

This function associates a callback to handle the events of the all open windows. All the events are caught by this callback and can be processed here. To set a callback for a specific window only, the optional parameter can be used to pass the PB window number.

To remove/disable a previous set callback just call SetWindowCallback(0 [, #Window]).

Warning: this way is lowlevel. Incorrect handling of the messages in the callback can interfere with PB's own message processing.

Parameters

@ProcedureName() The callback procedure to use. If a previous callback was set it will be replaced. The callback must have 4 parameters. If the event isn't processed, the result must be set to `#PB_ProcessPureBasicEvents` so the event will be properly handled by PureBasic. Here is a sample code to use a callback correctly:

```
1 Procedure MyWindowCallback(WindowID, Message, WParam, LParam)
2     Result = #PB_ProcessPureBasicEvents
3     ;
4     ; your code here
5     ;
6     ProcedureReturn Result
7 EndProcedure
```

#Window (optional) A specific window to associate the callback to. If omitted the callback will be called for any window.

Mode (optional) The callback mode, can be one of the following value:

```
#PB_Window_ProcessChildEvents: the child gadgets events
                              (mostly #WM_NOTIFY) will be sent to the main window.
                              Doesn't work for all gadgets
                              (default).
#PB_Window_NoChildEvents      : the child gadgets events won't
                              be sent to the main window (WinAPI default behavior).
```

Return value

None.

Example

```
1  Procedure WinCallback(hWnd, uMsg, WParam, LParam)
2      ; Windows fills the parameter automatically, which we will use
   in the callback...
3
4      If uMsg = #WM_SIZE
5          Select WParam
6              Case #SIZE_MINIMIZED
7                  Debug "Window was minimized"
8              Case #SIZE_RESTORED
9                  Debug "Window was restored"
10             Case #SIZE_MAXIMIZED
11                 Debug "Window was maximized"
12             EndSelect
13         EndIf
14
15         ProcedureReturn #PB_ProcessPureBasicEvents
16     EndProcedure
17
18
19     If OpenWindow(0, 0, 0, 200, 100, "Messages",
20         #PB_Window_MinimizeGadget | #PB_Window_MaximizeGadget)
21
22         SetWindowCallback(@WinCallback(), 0) ; set the callback
23
24         Repeat
25             Select WaitWindowEvent()
26                 Case #PB_Event_CloseWindow
27                     End
28             EndSelect
29         Forever
30     EndIf
```

Supported OS

Windows

181.26 SetWindowColor

Syntax

```
SetWindowColor(#Window, Color)
```

Description

Changes the background color of the specified window.

Parameters

#Window The window to use.

Color The new color to use for the window background. RGB() can be used to get a valid color value. A color table with common colors is available here . If color is set to #PB_Default, it will reset the background to the default color.

Return value

None.

Remarks

GetWindowColor() can be used to get the current background color of the window.

See Also

GetWindowColor()

181.27 SetWindowData

Syntax

```
SetWindowData(#Window, Value)
```

Description

Stores the given value with the specified window. This value can later be read with the GetWindowData() function. This allows to associate a custom value with any window.

Parameters

#Window The window to use.

Value The value to set.

Return value

None.

See Also

GetWindowData() , SetGadgetData() , GetGadgetData()

181.28 SetWindowState

Syntax

```
SetWindowState(#Window, State)
```

Description

Changes the minimized/maximized of the specified window.

Parameters

#Window The window to use.

State Can have one of the following values:

```
#PB_Window_Normal    : The window will be neither maximized nor
                        minimized.
#PB_Window_Maximize  : The window will be maximized. (Note: on
                        Linux, not all Windowmanagers support this)
#PB_Window_Minimize  : The window will be minimized.
```

Remarks

GetWindowState() can be used to get the current state of the window.

See Also

GetWindowState()

181.29 SetWindowTitle

Syntax

```
SetWindowTitle(#Window, Title$)
```

Description

Changes the text which is currently displayed in the window title bar.

Parameters

#Window The window to use.

Title\$ The new title to set.

Return value

None.

Remarks

GetWindowTitle() can be used to get the current window title.

Example

```
1  If OpenWindow(2, 100, 100, 200, 100, "My cool title")
2      SetWindowTitle(2, "Even cooler title !")
3
4      Repeat
5          Until WaitWindowEvent() = #PB_Event_CloseWindow
6      EndIf
```

See Also

GetWindowTitle()

181.30 SmartWindowRefresh

Syntax

```
SmartWindowRefresh(#Window, State)
```

Description

Enables a smart way to refresh the window to reduce the flickering when resizing the window. If the window isn't resizable, the function isn't needed. This function just try to help with the flickering problems, but it won't always give good results. The only way to see if it will work for a specific window is to enable it, see if there is any difference and see if the window content isn't trashed when a resize is done.

Parameters

#Window The window to use.

State It can be one of the following value:

```
#True : smart refresh is enabled
#False: smart refresh is disabled
```

Return value

None.

Supported OS

Windows

181.31 StickyWindow

Syntax

```
StickyWindow(#Window, State)
```

Description

Makes the specified window stay on top of all other open windows (also from other programs), even if it does not have the focus.

Parameters

#Window The window to use.

State It can be one the following values:

```
#True : the window will stay on top of all others.
#False: the window will not stay on top of all others when it
        does not have the focus.
```

Return value

None.

181.32 WindowEvent

Syntax

```
Event = WindowEvent()
```

Description

Checks if an event has occurred on any of the opened windows.

Parameters

None.

Return value

The next event from the event queue or zero if there are no more events. Unlike WaitWindowEvent() it doesn't wait for the next event - it always returns immediately. Event() can be used to get back this value.

This makes it useful for window event loops, where other processing needs to be done without waiting for an event to happen (e.g. Network transactions) and therefore WaitWindowEvent() can't be used.

It must be handled with care though if used on a continuing basis, because unlike WaitWindowEvent(), it will not give CPU time to other programs while waiting for an event and therefore consume all CPU power. In this case, either Delay() should be used somewhere in the loop or WaitWindowEvent() with a small timeout value.

To get the window number in which the event occurred, use the EventWindow() function.

Possible Events are :

```
#PB_Event_Menu           : a menu
has been selected
#PB_Event_Gadget         : a gadget
has been pushed
#PB_Event_SysTray        : an icon in the systray
zone was clicked
#PB_Event_Timer          : a timer
has reached its timeout
#PB_Event_CloseWindow    : the window close gadget has been
pushed
#PB_Event_Repaint        : the window content has been destroyed
and must be repainted (useful for 2D graphics operations
)
#PB_Event_SizeWindow     : the window has been resized
#PB_Event_MoveWindow     : the window has been moved
#PB_Event_MinimizeWindow : the window has been minimized
#PB_Event_MaximizeWindow : the window has been maximized
#PB_Event_RestoreWindow  : the window has been restored to
normal size (either from a minimum or maximum size)
#PB_Event_ActivateWindow : the window has been activated (got
the focus)
#PB_Event_DeactivateWindow : the window has been deactivated (lost
the focus)
#PB_Event_WindowDrop     : a Drag & Drop
operation was finished on a window
```

```

#PB_Event_GadgetDrop      : a Drag & Drop
operation was finished on a gadget

#PB_Event_RightClick      : a right mouse button click has
occurred on the window. This can be useful to display a popup
menu
#PB_Event_LeftClick       : a left mouse button click has
occurred on the window
#PB_Event_LeftDoubleClick : a left mouse button double-click has
occurred on the window

```

A basic example for event handling can be found in the `WaitWindowEvent()` description.

Remarks

After a `#PB_Event_WindowDrop` or `#PB_Event_GadgetDrop` Event, the event functions of the Drag & Drop library can be used to examine and read the dropped data.

Important: The window event loop should not be processed in a thread , as there is some limitation on OS X and Linux. A debugger error will be raised.

Correct way to handle an infinite `WindowEvent()` loop:

```

1  Repeat
2      Event = WindowEvent()
3
4      If Event      ; An event was in the queue so process it
5          ....
6      Else
7          Delay(1)  ; No event, let the others apps get some CPU time
            too !
8      EndIf
9  Until Event = #PB_Event_CloseWindow

```

Important: The 'Delay' shouldn't be put after each event, because when lot of events will come (like refresh, gadgets updates etc..) the app will wait between each event. So the delay need to be put when no events are received. Other (recommended) ways is to use `WaitWindowEvent()` with a small timeout value, or setup a timer with `AddWindowTimer()` .

Important: If you use a `FlipBuffers()` somewhere in your code, so the 'Delay(1)' is not necessary.

Example: n°1 With Gadget

```

1  If OpenWindow(0, 0, 0, 600, 100, "Position of the mouse on the
    window: ", #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      TextGadget(0, 10, 6, 200, 20, "")
3
4      Repeat
5          Event = WindowEvent()
6
7          If Event <> 0 ; All events are treated, so we can display the
            coordinates of the mouse
8              SetWindowTitle(0, "Position of the mouse on the window: " +
                Str(WindowMouseX(0)) + "," + Str(WindowMouseY(0)))
9          Else
10
11              Delay(1) ; Without a FlipBuffers(), Delay() frees the CPU for
                the multitasking
12          EndIf
13      Until Event = #PB_Event_CloseWindow
14  EndIf

```


Example: n°2 With Gadget

```
1  If OpenWindow(0, 0, 0, 300, 30, "Position of the mouse on the
    desktop", #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      TextGadget(0, 10, 6, 200, 20, "")
3
4      Repeat
5          Event = WindowEvent()
6
7          If Event = 0 ; No more events in the queue, so we can display
            the coordinates of the mouse
8              SetGadgetText(0, "Coordinates:
                "+Str(DesktopMouseX())+", "+Str(DesktopMouseY()))
9              EndIf
10
11             Delay(20) ; Without a FlipBuffers(), Delay() frees the CPU
                for the multitasking
12
13             Until Event = #PB_Event_CloseWindow
14             EndIf
```

Example: Without Gadgets (General case)

```
1      ;Few variables
2      BallX = 400
3      BallY = 200
4      BallSpeedY.f = 5
5      Gravitation.f = 2
6
7      ;Initialization
8      If InitSprite()
9          InitKeyboard()
10         InitMouse()
11     EndIf
12
13     ;Window
14     OpenWindow(0, 0, 0, 800, 600, "WindowEvent",
        #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
15     OpenWindowedScreen(WindowID(0), 0, 0, 800, 600)
16     SetFrameRate(30)
17
18     ;Ground
19     Ground = CreateSprite(#PB_Any, 800, 30)
20     StartDrawing(SpriteOutput(Ground))
21     Box(0,0,800,30,RGB(128, 0, 0))
22     StopDrawing()
23
24     ;Ball
25     Ball = CreateSprite(#PB_Any, 16, 16)
26     StartDrawing(SpriteOutput(Ball))
27     Box(0,0,16,16,RGB(135, 206, 235))
28     Circle(8,8,8,RGB(255, 255, 0))
29     StopDrawing()
30
31     ;Gauge
32     Image = CreateImage(#PB_Any, 8, 8, 24, RGB(255, 255, 255))
33     *Buffer=EncodeImage(Image, #PB_ImagePlugin_BMP)
34     Gauge = CatchSprite(#PB_Any, *Buffer)
```

```

35
36
37 ;Main Loop
38 Repeat
39
40     Repeat
41         ;Window events
42         ;=====
43         ;Try all possibilities but only one at a time
44         Event = WindowEvent() ; Animation
45         ;Event = WaitWindowEvent() ; Blocking animation
46         ;Event = WaitWindowEvent(100) ; Animation but every time an
event occurs like moving the mouse for example,
47                                     ; the animation will be suspended
for 100 ms
48         Select Event
49             Case #PB_Event_CloseWindow
50             End
51         EndSelect
52         Until Event=0
53
54         FlipBuffers() ; ==> With WindowEvent(), FlipBuffers() frees the
CPU for the multitasking so Delay(1) is not necessary
55         ClearScreen(RGB(135, 206, 235))
56
57         ExamineKeyboard() ;Keyboard
58
59         DisplaySprite(Gauge, 50, 570-BallY) ;Display the gauge
60         ZoomSprite(Gauge, 20, 570)
61
62         DisplaySprite(Ground, 0, 570) ;Display the ground
63
64         DisplaySprite(Ball, BallX, BallY) ;Display the ball
65
66         ;Ball Movement
67         BallSpeedY = BallSpeedY + Gravitation
68         BallY = BallY + BallSpeedY
69
70         ;Management of the collision of the ball with the ground
71         If SpriteCollision(Ball, BallX, BallY+16, Ground, 0, 570)
72             BallY= 554
73             BallSpeedY = -BallSpeedY
74         EndIf
75
76     Until KeyboardPushed(#PB_Key_Escape)

```

See Also

WaitWindowEvent(), Event(), EventWindow(), EventGadget(), EventMenu(), EventTimer(), EventData(), EventType(), PostEvent(), BindEvent(), UnbindEvent()

181.33 WaitWindowEvent

Syntax

```
Event = WaitWindowEvent([Timeout])
```

Description

Wait until an event occurs. It's the same function as `WindowEvent()` but locks the program execution, which is very important in a multitasking environment.

Parameters

Timeout (optional) The timeout (in milliseconds) which causes the function to return if no events are occurring. If no timeout is specified, it will wait infinitely until an event occurs.

Return value

Return the event which occurred, see `WindowEvent()` for more information. `Event()` can be used to get back this value.

Remarks

An application should always use this function instead of `WindowEvent()` if possible, as it doesn't takes an CPU time while waiting for an event.

The window event loop should not be processed in a thread , as there is some limitation on OS X and Linux. A debugger error will be raised.

`WaitWindowEvent()` can only be called once per event loop, because else events will be "lost" (every event can only be processed once and isn't available anymore for a second time after first processing).

Example: General case

```
1  If OpenWindow(0, 0, 0, 230, 90, "Event handling example...",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3
4      ButtonGadget (1, 10, 10, 200, 20, "Click me")
5      CheckBoxGadget(2, 10, 40, 200, 20, "Check me")
6
7      If CreateMenu(0, WindowID(0))
8          MenuItem("Menu")
9          MenuItem(1, "Item 1")
10         MenuItem(2, "Item 2")
11         MenuItem(3, "Item 3")
12     EndIf
13
14     Repeat
15         Event = WaitWindowEvent()
16
17         Select Event
18
19             Case #PB_Event_Gadget
20                 Select EventGadget()
21                     Case 1 : Debug "Button 1 clicked!"
22                     Case 2 : Debug "Button 2 clicked!"
23                 EndSelect
24
25             Case #PB_Event_Menu
26                 Select EventMenu()
27                     Case 1 : Debug "Menu item 1 clicked!"
28                     Case 2 : Debug "Menu item 2 clicked!"
29                     Case 3 : Debug "Menu item 3 clicked!"
30                 EndSelect
31         EndRepeat
```

```

30
31     EndSelect
32     Until Event = #PB_Event_CloseWindow
33 EndIf

```

Example: With a TimeOut

```

1  If OpenWindow(0, 0, 0, 300, 30, "Position of the mouse on the
   desktop", #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      TextGadget(0, 10, 6, 200, 20, "")
3
4      Repeat
5          Event = WaitWindowEvent(20)
6
7          If Event = 0 ; No more events in the queue, so let's display
           the mouse coordinates
8              SetGadgetText(0, "Coordinates: " + Str(DesktopMouseX()) +
               ", " + Str(DesktopMouseY()))
9              EndIf
10
11     Until Event = #PB_Event_CloseWindow
12 EndIf

```

Example: With a Timer

```

1  If OpenWindow(0, 0, 0, 300, 30, "Position of the mouse on the
   desktop", #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
2      TextGadget(0, 10, 6, 200, 20, "")
3      AddWindowTimer(0, 0, 10) ; Timeout = 10 ms
4
5      Repeat
6          Event = WaitWindowEvent()
7          If Event = #PB_Event_Timer ; Each 10 ms => Let's display the
           coordinates
8              SetGadgetText(0, "Coordinates: " + Str(DesktopMouseX()) + ", "
               + Str(DesktopMouseY()))
9              EndIf
10
11     Until Event = #PB_Event_CloseWindow
12 EndIf

```

See Also

WindowEvent() , Event() , EventWindow() , EventGadget() , EventMenu() , EventTimer() , EventData() , EventType() , PostEvent() , BindEvent() , UnbindEvent()

181.34 BindEvent

Syntax

```
BindEvent(Event, @Callback() [, Window [, Object [, EventType]]])
```

Description

Bind an event to a callback. It's an additional way to handle events in PureBasic, which works without problem with the regulars `WindowEvent()` / `WaitWindowEvent()` commands. It also allows to have real-time event notifications as the callback can be invoked as soon as the event occurs (useful for `ScrollBarGadget()` , live window resize, etc.). An event can be unbound with `UnbindEvent()` .

Parameters

Event The event to bind. For a full list of events, see `WindowEvent()` . Custom events are also supported, when using `PostEvent()` .

@Callback() The callback procedure to call when the event occurs. It has to be declared like this:

```
1  Procedure EventHandler()  
2      ; Code  
3  EndProcedure
```

Regular functions like `EventGadget()` , `EventWindow()` , `EventMenu()` , `EventType()` and `EventData()` are available within the callback to get more information about the event. Note: `WindowEvent()` and `WaitWindowEvent()` should never be called from inside the callback or the program can be locked or have wrong behavior.

Window (optional) The #Window number to bind the event to. The event will be dispatched only when occurring on this window. `#PB_All` can be specified to bind the event to all windows (if specified, 'Object' and 'EventType' parameters have to be set to `#PB_All` as well).

Object (optional) The object number to bind the event to. It can be a gadget , menuitem or systray number. `#PB_All` can be used to bind the event to any objects (if specified, 'EventType' parameter has to be set to `#PB_All` as well).

EventType (optional) The event type to bind the event to. For a full list of supported type, see `EventType()` . `#PB_All` can be used to bind the event to any type.

Return value

None.

Example

```
1  Procedure SizeWindowHandler()  
2      Debug "Size event on window #" + EventWindow()  
3  
4      ; Resize the gadget to fit the new window dimensions  
5      ;  
6      ResizeGadget(0, #PB_Ignore, #PB_Ignore,  
7      WindowWidth(EventWindow())-20, WindowHeight(EventWindow())-20)  
8  EndProcedure  
9  
10 OpenWindow(0, 100, 100, 200, 200, "Live resize test",  
11 #PB_Window_SizeGadget | #PB_Window_SystemMenu)  
12 EditorGadget(0, 10, 10, 180, 180)  
13  
14 BindEvent(#PB_Event_SizeWindow, @SizeWindowHandler())  
15 Repeat
```

```

15     Event = WaitWindowEvent()
16 Until Event = #PB_Event_CloseWindow

```

See Also

BindGadgetEvent() , BindMenuEvent() , UnbindEvent() , WindowEvent() , WaitWindowEvent() , ResizeWindow() , ResizeGadget()

181.35 UnbindEvent

Syntax

```
UnbindEvent(Event, @Callback() [, Window [, Object [, EventType]]])
```

Description

Unbind an event from a callback. If no matching event callback is found, this command has no effect.

Parameters

Event The event to unbind. For a full list of events, see WindowEvent() . Custom events are also supported, when using PostEvent() .

@Callback() The callback procedure to unbind.

Window (optional) The #Window number to unbind the event.

Object (optional) The object number to unbind the event. It can be a gadget, menuitem or stray number.

EventType (optional) The event type to unbind the event from. For a full list of supported types, see EventType() .

Return value

None.

Example

```

1  Procedure SizeWindowHandler()
2      Debug "Size event on window #" + EventWindow()
3  EndProcedure
4
5  OpenWindow(0, 100, 100, 200, 200, "Resize test",
6      #PB_Window_SizeGadget | #PB_Window_SystemMenu)
7
8  BindEvent(#PB_Event_SizeWindow, @SizeWindowHandler())
9  UnbindEvent(#PB_Event_SizeWindow, @SizeWindowHandler()) ; Unbind
10 it immediately
11
12 Repeat
13     Event = WaitWindowEvent()
14 Until Event = #PB_Event_CloseWindow

```

See Also

`BindEvent()` , `BindGadgetEvent()` , `BindMenuEvent()` , `WindowEvent()` , `WaitWindowEvent()`

181.36 WindowBounds

Syntax

```
WindowBounds(#Window, MinimumWidth, MinimumHeight, MaximumWidth,  
             MaximumHeight)
```

Description

Changes the minimal and maximal #Window dimensions (in pixels). This is useful to prevent a window from becoming too small or too big when the user resizes it.

Parameters

#Window The window to use.

MinimumWidth The minimum width of the window. If sets to `#PB_Ignore`, the current minimum width value is not changed. If sets to `#PB_Default`, the minimum width value is reset to the system default (as it was before calling this command).

MinimumHeight The minimum height of the window. If sets to `#PB_Ignore`, the current minimum height value is not changed. If sets to `#PB_Default`, the minimum height value is reset to the system default (as it was before calling this command).

MaximumWidth The maximum width of the window. If sets to `#PB_Ignore`, the current maximum width value is not changed. If sets to `#PB_Default`, the maximum width value is reset to the system default (as it was before calling this command).

MaximumHeight The maximum height of the window. If sets to `#PB_Ignore`, the current maximum height value is not changed. If sets to `#PB_Default`, the maximum height value is reset to the system default (as it was before calling this command).

Return value

None.

Remarks

It doesn't work on a borderless window.

Example

```
1  If OpenWindow(0, 0, 0, 300, 300, "Resize me !",  
    #PB_Window_SystemMenu | #PB_Window_ScreenCentered |  
    #PB_Window_SizeGadget)  
2      WindowBounds(0, 200, 200, 400, 400)  
3  
4      Repeat  
5          Event = WaitWindowEvent()  
6          Until Event = #PB_Event_CloseWindow  
7  EndIf
```

181.37 WindowHeight

Syntax

```
Result = WindowHeight(#Window [, Mode])
```

Description

Returns the height (in pixels) of the given window.

Parameters

#Window The window to use.

Mode (optional) The mode used to calculate the height of the window. It can be one of the following value:

```
#PB_Window_InnerCoordinate: height of the window inner area
                             (where gadget can be added),
                             excluding borders (default).
#PB_Window_FrameCoordinate: height of the window, including
                             borders.
```

Return value

Returns the height (in pixels) of the given window.

See Also

OpenWindow() , WindowWidth()

181.38 WindowID

Syntax

```
WindowID = WindowID(#Window)
```

Description

Returns the unique system identifier of the window.

Parameters

#Window The window to use.

Return value

The system identifier. This result is sometimes also known as 'Handle'. Look at the extra chapter Handles and Numbers for more information.

181.39 WindowWidth

Syntax

```
Result = WindowWidth(#Window [, Mode])
```

Description

Returns the width (in pixels) of the given window.

Parameters

#Window The window to use.

Mode (optional) The mode used to calculate the width of the window. It can be one of the following value:

```
#PB_Window_InnerCoordinate: width of the window inner area
                             (where gadget can be added),
                             excluding borders (default).
#PB_Window_FrameCoordinate: width of the window, including
                             borders.
```

Return value

Returns the width (in pixels) of the given window.

See Also

OpenWindow() , WindowHeight()

181.40 WindowX

Syntax

```
Result = WindowX(#Window [, Mode])
```

Description

Returns the x position on the screen of the given window.

Parameters

#Window The window to use.

Mode (optional) The mode used to calculate the x position of the window. It can be one of the following value:

```
#PB_Window_FrameCoordinate: x position of the window,
                             including borders (default).
#PB_Window_InnerCoordinate: x position of the window inner
                             area (where gadget can be added),
                             excluding borders.
```

Return value

Returns the x position (from the left) on the screen (in pixels) of the given window.

See Also

OpenWindow() , WindowY()

181.41 WindowY

Syntax

```
Result = WindowY(#Window [, Mode])
```

Description

Returns the y position on the screen of the given window.

Parameters

#Window The window to use.

Mode (optional) The mode used to calculate the y position of the window. It can be one of the following value:

```
#PB_Window_FrameCoordinate: y position of the window,  
                           including borders (default).  
#PB_Window_InnerCoordinate: y position of the window inner  
                           area (where gadget can be added),  
                           excluding borders.
```

Return value

Returns the y position (from the top) on the screen (in pixels) of the given window.

See Also

OpenWindow() , WindowX()

181.42 WindowMouseX

Syntax

```
x = WindowMouseX(#Window)
```

Description

Returns the mouse x position in the inner area of the specified window.

Parameters

#Window The window to use.

Return value

The mouse x position in the inner area of the given window. If the mouse is outside of the window area, it will return -1.

Remarks

To get the absolute x position of the mouse on the desktop, use DesktopMouseX() .

Example

```
1  If OpenWindow(0, 0, 0, 300, 30, "Window mouse monitor",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      TextGadget(0, 10, 6, 200, 20, "")
4
5      Repeat
6          Event = WaitWindowEvent(20) ; return at least every 20ms for
7          an update
8          SetGadgetText(0, "Window mouse position: " +
9              Str(WindowMouseX(0)) + "," + Str(WindowMouseY(0)))
10         Until Event = #PB_Event_CloseWindow
11     EndIf
```

181.43 WindowMouseY

Syntax

```
y = WindowMouseY(#Window)
```

Description

Returns the mouse y position in the inner area of the given window.

Parameters

#Window The window to use.

Return value

The mouse y position in the inner area of the specified window. If the mouse is outside of the window area, it will return -1.

Remarks

To get the absolute y position of the mouse on the desktop, use DesktopMouseY() .

```
1  If OpenWindow(0, 0, 0, 300, 30, "Window mouse monitor",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      TextGadget(0, 10, 6, 200, 20, "")
4
5      Repeat
6          Event = WaitWindowEvent(20) ; return at least every 20ms for
7          an update
8          SetGadgetText(0, "Window mouse position: " +
9              Str(WindowMouseX(0)) + "," + Str(WindowMouseY(0)))
10         Until Event = #PB_Event_CloseWindow
11     EndIf
```

181.44 WindowOutput

Syntax

```
OutputID = WindowOutput(#Window)
```

Description

Returns the OutputID of the given window to perform 2D rendering operation on it. It will use the PureBasic 2DDrawing library and can only be used within a StartDrawing() / StopDrawing() block. The memory allocated in WindowOutput() is released on StopDrawing().

Parameters

#Window The window to use.

Return value

The OutputID of the given window to perform 2D rendering operation on it using StartDrawing() .

Example

```
1  If OpenWindow(0, 0, 0, 220, 100, "Example...",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      Repeat
4          Event = WaitWindowEvent()
5
6          If Event = #PB_Event_Repaint ; Redraw on the window every
            time the window is repainted
7              StartDrawing(WindowOutput(0))
8              Box(10, 10, 50, 50, RGB(255, 0, 0))
9              StopDrawing()
10         EndIf
11     Until Event = #PB_Event_CloseWindow
12 EndIf
```

Remarks

Content drawn on a window will be erased whenever the window or a part of it is covered by another window, moved outside of the screen or when the window is hidden or minimized. So to keep the drawn content visible, it must be redrawn after every **#PB_Event_Repaint** event. A more convenient alternative is to draw the content to an image via ImageOutput() and display it as ImageGadget() in the application window and if necessary, update it with SetGadgetState() . This way all needed refreshing will be handled by the ImageGadget.

Example

```
1  If OpenWindow(0, 0, 0, 220, 100, "Example...",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      ButtonGadget (1, 10, 60, 200, 30, "Draw on window")
4
5      Repeat
6          Event = WaitWindowEvent()
```

```

7
8     Select Event
9
10    Case #PB_Event_Gadget
11        Select EventGadget()
12            Case 1
13                ; Draw a red box on the window
14                If StartDrawing(WindowOutput(0))
15                    Box(10,10, 200, 30, RGB(255, 0, 0))
16                    StopDrawing()
17                EndIf
18            EndSelect
19        EndSelect
20
21    EndSelect
22    Until Event = #PB_Event_CloseWindow
23 EndIf

```

See Also

StartDrawing() , WindowVectorOutput()

181.45 WindowVectorOutput

Syntax

```
VectorOutputID = WindowVectorOutput(#Window [, Unit])
```

Description

Returns the OutputID of the given window to perform vector drawing operations. It will use the PureBasic VectorDrawing library and can only be used within a StartVectorDrawing() / StopVectorDrawing() block. The memory allocated in WindowVectorOutput() is released on StopVectorDrawing() .

Parameters

#Window The window to use.

Unit (optional) Specifies the unit used for measuring distances on the drawing output. The default unit for windows is **#PB_Unit_Pixel**.

```

#PB_Unit_Pixel      : Values are measured in pixels (or dots
                     in case of a printer)
#PB_Unit_Point      : Values are measured in points (1/72 inch)
#PB_Unit_Inch       : Values are measured in inches
#PB_Unit_Millimeter : Values are measured in millimeters

```

Return value

The OutputID of the given window to perform 2D rendering operation on it using StartVectorDrawing() .

Remarks

Content drawn on a window will be erased whenever the window or a part of it is covered by another window, moved outside of the screen or when the window is hidden or minimized. So to keep the drawn content visible, it must be redrawn after every `#PB_Event_Repaint` event. A more convenient alternative is to draw the content to an image via `ImageVectorOutput()` and display it as `ImageGadget()` in the application window and if necessary, update it with `SetGadgetState()` . This way all needed refreshing will be handled by the `ImageGadget`.

See Also

`StartVectorDrawing()` , `WindowOutput()`

181.46 EventwParam

Syntax

```
Result = EventwParam()
```

Description

This function is not supported anymore and shouldn't used in new project. Use a callback to get full control over Windows message with `SetWindowCallback()` .

Supported OS

Windows

181.47 EventlParam

Syntax

```
Result = EventlParam()
```

Description

This function is not supported anymore and shouldn't used in new project. Use a callback to get full control over Windows message with `SetWindowCallback()` .

Supported OS

Windows

Chapter 182

Window3D

Overview

The Window3D library allows you to create complex Graphical User Interfaces (GUIs) directly over the screen area, using the 3D engine. This is mainly intended for games or applications which need user input while running in fullscreen mode. This library is based on the regular PureBasic window library , and offers similar syntax and behavior. The GUI engine used is CEGUI, which offers some nice options like skinning, good speed and a lot of built-in gadgets. More information about CEGUI can be found here: <http://www.cebui.org.uk>. This library uses the 3D engine, so InitEngine3D() has to be called successfully before using these functions.

182.1 CloseWindow3D

Syntax

```
CloseWindow3D (#Window3D)
```

Description

Close the specified window.

Parameters

#Window3D The 3D window to close. If **#PB_All** is specified, all the remaining 3D windows are closed.

Return value

None.

Remarks

All remaining opened 3D windows are automatically closed when the program ends.

182.2 DisableWindow3D

Syntax

```
DisableWindow3D (#Window3D , State)
```

Description

Enables or disables user input to the specified Window.

Parameters

#Window3D The 3D window to use.

State It can take the following values:

```
#True : The window is disabled.  
#False: The window is enabled.
```

Return value

None.

182.3 EventGadget3D

Syntax

```
Result = EventGadget3D()
```

Description

After an event of type **#PB_Event3D_Gadget** (returned by `WindowEvent3D()`), use this function to determine which gadget has been triggered.

Parameters

None.

Return value

The 3D gadget number associated to the last **#PB_Event3D_Gadget** event.

See Also

`WindowEvent3D()`

182.4 EventType3D

Syntax

```
Result = EventType3D()
```

Description

After a `WindowEvent3D()` function, use this function to determine of which type the event is.

Parameters

None.

Return value

The following values are possible, if an event of the type `#PB_Event3D_Gadget` (library `Gadget3D`) occurs:

```
#PB_EventType3D_Focus      : Get the focus.
#PB_EventType3D_LostFocus : Lose the focus.
#PB_EventType3D_Change     : Content change.
```

The following gadgets support `EventType3D()`:

- `SpinGadget3D()`
- `StringGadget3D()`

(See the gadget definition to see which events are supported).

See Also

`WindowEvent3D()`

182.5 EventWindow3D

Syntax

```
Result = EventWindow3D()
```

Description

After a `WindowEvent3D()` function, use this function to determine on which window the event has occurred.

Parameters

None.

Return value

The 3D window number associated with the last event.

182.6 InputEvent3D

Syntax

```
InputEvent3D(MouseX, MouseY, LeftMouseButton [, Text$, SpecialKey])
```

Description

Send an event to the 3D GUI system. It is required to have `WindowEvent3D()` working.

Parameters

MouseX, MouseY The new mouse coordinate to use.

LeftMouseButton Left mouse button state. It can be one of the following values:

```
0: left mouse button is released.
1: left mouse button is pushed.
```

Text\$ (optional) The text to use as input.

SpecialKey (optional) The special key to use as key press. It can be one of the following values:

```
#PB_Key_Back  
#PB_Key_Delete  
#PB_Key_Return  
#PB_Key_Up  
#PB_Key_Down  
#PB_Key_Left  
#PB_Key_Right
```

Return value

None.

Example

```
1 ;  
2 ; Typical code to use to send to the 3D GUI system with actual  
   mouse and keyboard events  
3 ;  
4 If ExamineKeyboard() And ExamineMouse()  
5     Input$ = KeyboardInkey()  
6  
7     InputEvent3D(MouseX(), MouseY(),  
6     MouseButton(#PB_MouseButton_Left), Input$, 0)  
8 EndIf
```

See Also

WindowEvent3D()

182.7 GetActiveWindow3D

Syntax

```
Result = GetActiveWindow3D()
```

Description

Returns the 3D window number which currently has the keyboard focus.

Parameters

None.

Return value

The 3D window number which currently has the keyboard focus. If no window has the focus, -1 is returned.

Remarks

A window can be activated (have the keyboard focus set to it) with the `SetActiveWindow3D()` function.

See Also

`SetActiveWindow3D()`

182.8 GetWindowTitle3D

Syntax

```
Result\$_ = GetWindowTitle3D(#Window3D)
```

Description

Returns the text which is currently displayed in the title bar of the specified 3D window.

Parameters

#Window3D The 3D window to use.

Return value

The text which is currently displayed in the specified 3D window title bar.

Remarks

The title of a window can be changed with `SetWindowTitle3D()` .

See Also

`SetWindowTitle3D()`

182.9 HideWindow3D

Syntax

```
HideWindow3D(#Window3D, State)
```

Description

Hides or shows the specified `#Window3D`.

Parameters

#Window3D The 3D window to use.

State It can take the following values:

```
#True : the #Window3D is hidden
#False: the #Window3D is shown
```

Return value

None.

182.10 IsWindow3D

Syntax

```
Result = IsWindow3D(#Window3D)
```

Description

Tests if the given 3D window is valid and correctly initialized.

Parameters

#Window3D The 3D window to test.

Return value

Nonzero if the 3D window is valid, zero otherwise.

Remarks

This function is bulletproof and may be used with any value. This is the correct way to ensure a 3D window is ready to use.

See Also

OpenWindow3D()

182.11 OpenWindow3D

Syntax

```
Result = OpenWindow3D(#Window3D, x, y, InnerWidth, InnerHeight,  
    Title$ [, Flags])
```

Description

Opens a new window on the current screen according to the specified parameters. The new window becomes the active window, it's not needed to use SetActiveWindow3D() (unless the window is created as invisible). All possible events in a window are handled with WindowEvent3D() .

Parameters

#Window3D A number to identify the new 3D window. **#PB_Any** can be used to auto-generate this number.

x, y The initial position of the 3D window on the screen (unless one of the center flags is used).

InnerWidth, InnerHeight The initial size of the client area of the 3D window (without borders and window decorations).

Flags (optional) It can be a combination of the following values:

```
#PB_Window3D_SizeGadget : Adds the sizeable feature to a
window.
#PB_Window3D_Invisible : Creates the window but doesn't
display it.
#PB_Window3D_BorderLess : Creates a window without any
borders.
```

Return value

Nonzero if the 3D window has been successfully opened, zero otherwise.

See Also

CloseWindow3D() , WindowEvent3D()

182.12 ResizeWindow3D

Syntax

```
ResizeWindow3D(#Window3D, x, y, Width, Height)
```

Description

Move and resize the given window to the given position and size.

Parameters

#Window3D The 3D window to resize.

x, y The new 3D window position. If 'x' or 'y' is set to `#PB_Ignore`, the current value of 'x' or 'y' won't be changed.

Width, Height The new 3D window size. If 'Width' or 'Height' is set to `#PB_Ignore`, the current value of 'Width' or 'Height' won't be changed.

Return value

None.

182.13 SetActiveWindow3D

Syntax

```
SetActiveWindow3D(#Window3D)
```

Description

Activate the specified window, which means the focus has been put on this window.

Parameters

#Window3D The 3D window to activate.

Return value

None.

See Also

GetActiveWindow3D()

182.14 SetWindowTitle3D

Syntax

```
SetWindowTitle3D(#Window3D, Title$)
```

Description

Changes the text which is currently displayed in the specified 3D window title bar.

Parameters

#Window3D The 3D window to use.

Title\$ The new title to set.

Return value

None.

See Also

GetWindowTitle3D()

182.15 WindowEvent3D

Syntax

```
Result = WindowEvent3D()
```

Description

Checks if an event has occurred on any of the opened 3D windows. InputEvent3D() needs to be used to send events to the 3D GUI system to be able to have window events.

WindowEvent3D() returns the next event from the event queue and returns zero when there are no more events. It doesn't wait for the next event - it always returns immediately.

To get the number of the window on which the event occurred, use the EventWindow3D() function.

Parameters

None.

Return value

Possible Events are:

```
#PB_Event3D_Gadget          : a Gadget3D
has been used
#PB_Event3D_CloseWindow     : the window close gadget has been
    pushed
#PB_Event3D_SizeWindow      : the window has been resized
#PB_Event3D_MoveWindow      : the window has been moved
#PB_Event3D_ActivateWindow  : the window has been activated (got
    the focus)
```

See Also

EventWindow3D() , InputEvent3D()

182.16 WindowHeight3D

Syntax

```
Result = WindowHeight3D(#Window3D)
```

Description

Returns the height of the given window.

Parameters

#Window3D The 3D window to use.

Return value

The height, in pixels, of the given window.

See Also

WindowWidth3D()

182.17 WindowID3D

Syntax

```
Result = WindowID3D(#Window3D)
```

Description

Returns the unique system identifier of the 3D window.

Parameters

#Window3D The 3D window to use.

Return value

The unique system identifier of the 3D window.

182.18 WindowWidth3D

Syntax

```
Result = WindowWidth3D(#Window3D)
```

Description

Return the width of the given window.

Parameters

#Window3D The 3D window to use.

Return value

The width, in pixels, of the given window.

See Also

WindowHeight3D()

182.19 WindowX3D

Syntax

```
Result = WindowX3D(#Window3D)
```

Description

Returns the left position on the screen of the given window.

Parameters

#Window3D The 3D window to use.

Return value

The left position on the screen, in pixels, of the given window.

See Also

WindowY3D()

182.20 WindowY3D

Syntax

```
Result = WindowY3D(#Window3D)
```

Description

Returns the top position on the screen, of the given window.

Parameters

#Window3D The 3D window to use.

Return value

The top position on the screen, in pixels, of the given window.

See Also

WindowX3D()

Chapter 183

XML

Overview

The XML library provides set of functions to easily add XML parsing and creating capability to applications. It is based on the [expat XML parser](#), which is licensed under the MIT license which can be viewed here . expat is used in many projects (like Mozilla or Perl). It is very stable and very fast.

Important: The expat license requires that a copyright notice and the license text itself be included in any software that includes the parser. So if this library (or the API import) are used in software that is to be made public, the above linked license must be included with the software. The expat functions can also be called directly from PureBasic like other API functions. However you will have to add "pb_" at the beginning of the function name in addition to the ending 'underscore'. This allows to take advantage of expat features not directly provided by this library. The constants and structures defined in expat.h are directly available in PureBasic. For detailed information on the expat functions, refer to the documentation provided in the download package on <http://expat.sourceforge.net/>

This library has partial support for Document Type Definitions (DTD) and Namespaces. The goal is to keep the commandset very simple while still allowing this library to handle any XML compliant document.

The expat parser is a non-validating parser. This means it checks the parsed documents for errors in the markup (a document must be well-formed according to the XML specification), but it does not validate the document against a DTD. When parsing a document, this library places DTDs inside a special node in the XML tree with the type `#PB_XML_DTD`. The content of this node is the full DOCTYPE tag. This way it can be easily accessed and manipulated but is also save to ignore if this information is not needed. The tag is simply written back when exporting/saving the document.

Namespaces do not get resolved when parsing a document. This means that in a document using namespaces, the namespace declarations are accessible as normal node attributes, and node/attribute names using namespaces will be visible as "namespace:tagname". This allows a document using namespaces to be read and also saved back like any other document without destroying its structure. To make working with namespaces simpler, the functions `ResolveXMLNodeName()` and `ResolveXMLAttributeName()` are provided to resolve names inside documents that use namespaces.

The PureBasic Debugger provides the possibility to examine `#XML` objects during runtime with the Library Viewer tool.

The official specification of XML and XML Namespaces by the W3C can be found here:

[XML specification](#)

[XML Namespaces](#)

[Various translations of XML related documents](#)

Also the [Wikipedia article on XML](#) provides a good starting point for people new to XML.

183.1 IsXML

Syntax

```
Result = IsXML (#XML)
```

Description

Tests if the given #XML number is a valid and correctly initialized, XML.

Parameters

#XML The XML to use.

Return value

Nonzero if #XML is a valid XML, zero otherwise.

Remarks

This function is bulletproof and can be used with any value. If the 'Result' is not zero then the object is valid and initialized, otherwise it will equal zero. This is the correct way to ensure a XML is ready to use.

See Also

LoadXML() , CreateXML()

183.2 FreeXML

Syntax

```
FreeXML (#XML)
```

Description

Frees the XML object and all data it contains.

Parameters

#XML The XML object to free. If #PB_All is specified, all the remaining XML objects are freed.

Return value

None.

Remarks

All remaining XML objects are automatically freed when the program ends.

183.3 CreateXML

Syntax

```
Result = CreateXML(#XML [, Encoding])
```

Description

Creates a new empty XML tree identified by the #XML number.

Parameters

#XML A number to identify the new XML. #PB_Any can be used to auto-generate this number.

Encoding (optional) The encoding to use for the XML tree. Valid values are:

```
#PB_UTF8 (default)
#PB_Ascii
#PB_Unicode
```

Return value

Nonzero if the XML tree was created successfully, zero otherwise. If #PB_Any was used for the #XML parameter then the generated number is returned on success.

Remarks

The new XML tree will only have a root node which can be accessed with RootXMLNode() . To add new nodes, CreateXMLNode() can be used.

Example

```
1  ; Create xml tree
2  xml = CreateXML(#PB_Any)
3  mainNode = CreateXMLNode(RootXMLNode(xml), "Zoo")
4
5  ; Create first xml node (in main node)
6  item = CreateXMLNode(mainNode, "Animal")
7  SetXMLAttribute(item, "id", "1")
8  SetXMLNodeText(item, "Elephant")
9
10 ; Create second xml node (in main node)
11 item = CreateXMLNode(mainNode, "Animal")
12 SetXMLAttribute(item, "id", "2")
13 SetXMLNodeText(item, "Tiger")
14
15 ; Save the xml tree into a xml file
16 SaveXML(xml, "demo.xml")
```

See Also

FreeXML() , CreateXMLNode() , RootXMLNode()

183.4 LoadXML

Syntax

```
Result = LoadXML(#XML, Filename$ [, Encoding])
```

Description

Loads a XML tree from the specified file.

Parameters

#XML A number to identify the new XML. #PB_Any can be used to auto-generate this number.

Filename\$ The filename to load the XML from.

Encoding (optional) The encoding to use when loading the XML tree (this overwrites the encoding set in the XML declaration). Valid values are:

```
#PB_UTF8 (default)
#PB_Ascii
#PB_Unicode
```

This parameter should be used when the document does not have an XML declaration, or the encoding information is provided outside of the XML document, for example through a mime type header in a communication protocol.

Return value

Nonzero if the file could be opened and read. Note that this does not mean that the XML contained in the file was valid. To check for parser errors XMLStatus() should be used. In case of a parsing error, all data parsed before the error is accessible in the XML tree. If #PB_Any was used for the #XML parameter then the generated number is returned on success.

See Also

CreateXML() , FreeXML() , ParseXML() , CatchXML()

183.5 CatchXML

Syntax

```
Result = CatchXML(#XML, *Address, Size [, Flags [, Encoding]])
```

Description

Creates a new XML tree from XML data in the given memory area. The markup can be parsed in blocks by multiple calls to this function to allow parsing XML data while it arrives from the network for example.

Parameters

#XML A number to identify the new XML. **#PB_Any** can be used to auto-generate this number.

***Address** A readable memory location.

Size Size (in bytes) of the memory location.

Flags (optional) If omitted, the memory location must contain all XML data. To parse XML in multiple blocks, the following flags can be used:

```
#PB_XML_StreamStart: Start parsing the first block
#PB_XML_StreamNext : Continue parsing with a new block
#PB_XML_StreamEnd  : End parsing after this block
```

When calling this function with **#PB_XML_StreamStart** or **#PB_XML_StreamEnd**, the 'Size' parameter can be 0 to start/end a parsing operation without actually parsing more data.

Note that when parsing in blocks, all data already parsed is accessible in the XML tree even before a call with **#PB_XML_StreamEnd** is made.

Encoding (optional) The encoding to use when loading the XML tree (this overwrites the encoding set in the XML declaration). Valid values are:

```
#PB_UTF8 (default)
#PB_Ascii
#PB_Unicode
```

This parameter should be used when the document does not have an XML declaration, or the encoding information is provided outside of the XML document, for example through a mime type header in a communication protocol.

Return value

This function only returns zero on memory errors or invalid Flags. To check for parser errors **XMLStatus()** should be used. In case of a parsing error, all data parsed before the error is accessible in the XML tree.

See Also

FreeXML() , **CreateXML()** , **LoadXML()** , **ParseXML()**

183.6 ParseXML

Syntax

```
Result = ParseXML(#XML, Input$)
```

Description

Creates a new XML tree from XML data in the string. The XML is expected to be encoded in the string format of the executable (Ascii or Unicode). If another encoding needs to be parsed, the **CatchXML()** function can be used instead.

Parameters

#XML A number to identify the new XML. **#PB_Any** can be used to auto-generate this number.

Input\$ The string containing the XML to parse.

Return value

This function only returns zero on memory errors. To check for parser errors XMLStatus() should be used. In case of a parsing error, all data parsed before the error is accessible in the XML tree.

See Also

FreeXML() , CreateXML() , LoadXML()

183.7 XMLStatus

Syntax

```
Result = XMLStatus(#XML)
```

Description

Returns the status of the last parsing operation done on this XML tree (using LoadXML() or CatchXML()). This function should be called after every LoadXML() or CatchXML() call to ensure that the parsing succeeded. A string representation of the parsing status (ie a readable error-message) is returned by the XMLError() function.

Parameters

#XML The XML to use.

Return value

A value of zero (#PB_XML_Success) indicates a successful parsing, all other values indicate various error conditions.

The following returnvalues are possible:

#PB_XML_Success	: no error
#PB_XML_NoMemory	: out of memory
#PB_XML_Syntax	: syntax error
#PB_XML_NoElements	: no element found
#PB_XML_InvalidToken	: not well-formed (invalid token)
#PB_XML_UnclosedToken	: unclosed token
#PB_XML_PartialCharacter	: partial character
#PB_XML_TagMismatch	: mismatched tag
#PB_XML_DuplicateAttribute	: duplicate attribute
#PB_XML_JunkAfterDocElement	: junk after document element
#PB_XML_ParamEntityRef	: illegal parameter entity reference
#PB_XML_UndefinedEntity	: undefined entity
#PB_XML_RecursiveEntityRef	: recursive entity reference
#PB_XML_AsyncEntity	: asynchronous entity
#PB_XML_BadCharacterRef	: reference to invalid character
number	
#PB_XML_BinaryEntityRef	: reference to binary entity
#PB_XML_AttributeExternalEntityRef	: reference to external entity in attribute
#PB_XML_MisplacedXML	: XML or text declaration not at start of entity
#PB_XML_UnknownEncoding	: unknown encoding
#PB_XML_IncorrectEncoding	: encoding specified in XML declaration is incorrect
#PB_XML_UnclosedCDATASection	: unclosed CDATA section

```

#PB_XML_ExternalEntityHandling: error in processing external
    entity reference
#PB_XML_NotStandalone           : document is not standalone
#PB_XML_UnexpectedState        : unexpected parser state
#PB_XML_EntityDeclaredInPE     : entity declared in parameter entity
#PB_XML_FeatureRequiresDTD     : requested feature requires XML_DTD
    support in Expat
#PB_XML_CantChangeFeatures     : cannot change setting once parsing
    has begun
#PB_XML_UnboundPrefix          : unbound prefix
#PB_XML_UndeclaringPrefix      : must not undeclare prefix
#PB_XML_IncompletePE          : incomplete markup in parameter
    entity
#PB_XML_XMLDeclaration         : XML declaration not well-formed
#PB_XML_TextDeclaration        : text declaration not well-formed
#PB_XML_PublicID               : illegal character(s) in public id
#PB_XML_Suspended              : parser suspended
#PB_XML_NotSuspended           : parser not suspended
#PB_XML_Aborted                : parsing aborted
#PB_XML_Finished               : parsing finished
#PB_XML_SuspendedPE           : cannot suspend in external
    parameter entity
#PB_XML_ReservedPrefixXML      : reserved prefix (xml) must not be
    undeclared or bound to another namespace name
#PB_XML_ReservedPrefixXMLNS    : reserved prefix (xmlns) must not be
    declared or undeclared
#PB_XML_ReservedNamespaceURI  : prefix must not be bound to one of
    the reserved namespace names

```

See Also

ParseXML()

183.8 XML_Error

Syntax

```
Result\[ $ = XML_Error( #XML )
```

Description

In case of an error while parsing XML data this function returns an error-message describing the error. XML_Status() can be used to detect parsing errors.

Parameters

#XML The XML to use.

Return value

The english readable error string.

Remarks

To get more information about the error, XML_ErrorLine() or XML_ErrorPosition() can be used.

See Also

XMLErrorLine() , XMLErrorPosition() , XMLStatus()

183.9 XMLErrorLine

Syntax

```
Result = XMLErrorLine(#XML)
```

Description

In case of an error while parsing XML data this function returns the line in the input that caused the error (one based). XMLStatus() can be used to detect parsing errors.

Parameters

#XML The XML to use.

Return value

The line in the XML where the error occurred. The first line index starts from 1.

Remarks

To get the position within the line at which the error happened, XMLErrorPosition() can be used.

See Also

XMLError() , XMLErrorPosition() , XMLStatus()

183.10 XMLErrorPosition

Syntax

```
Result = XMLErrorPosition(#XML)
```

Description

In case of an error while parsing XML data this function returns character position within the line returned by XMLErrorLine() at which the error was caused. (The first character of the line is at position 1) XMLStatus() can be used to detect parsing errors.

Parameters

#XML The XML to use.

Return value

The character position in the XML where the error occurred. The first character index starts from 1.

See Also

XML_Error() , XML_ErrorLine() , XML_Status()

183.11 SaveXML

Syntax

```
Result = SaveXML(#XML, Filename$ [, Flags])
```

Description

Saves the #XML tree to the given file.

Parameters

#XML The XML to save.

Filename\$ The filename where the XML should be saved.

Flags (optional) It can be a combination of the following values (with the '||' operator):

```
#PB_XML_StringFormat : Includes a byte order mark. See
WriteStringFormat()
for more information.
#PB_XML_NoDeclaration: Does not include the XML declaration.
```

Note: According to the XML specification, the XML declaration can only be omitted if the document is encoded in UTF-8 or UTF-16 or if the encoding information is provided externally through a transfer protocol for example. Even then, it is advised to keep the declaration in the document.

Return value

Nonzero if the file was successfully saved, zero otherwise.

Remarks

The created XML markup is not reformatted. It is written back as it was initially parsed/created. The amount of newline/whitespace written between the tags is stored in the 'text' of each XML node. (see GetXMLNodeText() for more information) To reformat the XML markup before saving, the 'text' for each XML node can be altered or FormatXML() can be used to apply some common reformatting options to the tree.

See Also

LoadXML() , CreateXML() , FormatXML() , ExportXML() , ComposeXML()

183.12 ExportXMLSize

Syntax

```
Result = ExportXMLSize(#XML [, Flags])
```

Description

Returns the size in bytes that will be needed to export the given XML tree to a memory buffer . This function should be used to determine the needed buffersize for the ExportXML() command.

Parameters

#XML The XML to use.

Flags (optional) It can be used to specify the same options as accepted by the `ExportXML()` command. It allows to include these options in the size calculation.

Return value

The size in bytes that will be needed to export the given XML tree to a memory buffer .

See Also

`ExportXML()`

183.13 ExportXML

Syntax

```
Result = ExportXML(#XML, *Address, Size [, Flags])
```

Description

Writes the XML tree as markup to the given memory buffer .

Parameters

#XML The XML to export.

***Address** A writable memory location.

Size Size (in bytes) of the memory location.

Flags (optional) It can be a combination of the following values (with the '|' operator):

```
#PB_XML_StringFormat : Includes a byte order mark. See  
WriteStringFormat()  
for more information.  
#PB_XML_NoDeclaration: Does not include the XML declaration.
```

Note: According to the XML specification, the XML declaration can only be omitted if the document is encoded in UTF-8 or UTF-16 or if the encoding information is provided externally through a transfer protocol for example. Even then, it is advised to keep the declaration in the document.

Return value

Nonzero if the specified length was large enough to hold the entire markup of the tree. `ExportXMLSize()` can be used to determine the needed size for this buffer.

Remarks

The created XML markup is not reformatted. It is written back as it was initially parsed/created. The amount of newline/whitespace written between the tags is stored in the 'text' of each XML node. (see `GetXMLNodeText()` for more information) To reformat the XML markup before saving, the 'text' for each XML node can be altered or `FormatXML()` can be used to apply some common reformatting options to the tree.

See Also

ExportXMLSize() , FormatXML() , ComposeXML() , SaveXML()

183.14 ComposeXML

Syntax

```
Result\$_ = ComposeXML(#XML [, Flags])
```

Description

Returns the XML tree as markup in a single string. The XML will be returned in the string format of the executable (Ascii or Unicode) independent of the setting returned by GetXMLEncoding() . The ExportXML() function can be used to create markup in a different encoding.

Parameters

#XML The XML to export.

Flags (optional) It can be a combination of the following values (with the '|' operator):

```
#PB_XML_StringFormat : Includes a byte order mark. See  
WriteStringFormat()  
for more information.  
#PB_XML_NoDeclaration: Does not include the XML declaration.
```

Note: According to the XML specification, the XML declaration can only be omitted if the document is encoded in UTF-8 or UTF-16 or if the encoding information is provided externally through a transfer protocol for example. Even then, it is advised to keep the declaration in the document.

Return value

Returns the markup as a string.

Remarks

The created XML markup is not reformatted. It is returned as it was initially parsed/created. The amount of newline/whitespace written between the tags is stored in the 'text' of each XML node. (see GetXMLNodeText() for more information) To reformat the XML markup before saving, the 'text' for each XML node can be altered or FormatXML() can be used to apply some common reformatting options to the tree.

See Also

FormatXML() , ExportXML() , SaveXML()

183.15 FormatXML

Syntax

```
FormatXML(#XML, Flags [, IndentStep])
```

Description

Cleans up or reformats the XML tree for a better look when exporting /saving . It can be used to have a very compact output for efficient transfer or a more formatted output for better reading. The formatting of the parsed XML document is stored in the 'text' and 'offset' fields of each node in the tree (see `GetXMLNodeText()` and `GetXMLNodeOffset()` for more information).

Parameters

#XML The XML to format.

Flags It can be a combination of the following values (with the '|' operator):

```
#PB_XML_WindowsNewline : Changes all newline to CRLF
#PB_XML_LinuxNewline   : Changes all newline to LF
#PB_XML_MacNewline     : Changes all newline to CR

#PB_XML_CutNewline     : Removes all newline
#PB_XML_ReduceNewline  : Removes all empty lines

#PB_XML_CutSpace       : Removes all spaces
#PB_XML_ReduceSpace    : Removes all multiple spaces

#PB_XML_ReFormat       : Completely reformats the tree
                        structure
#PB_XML_ReIndent       : Changes the indentation of the lines
```

For `#PB_XML_ReFormat` and `#PB_XML_ReIndent` the 'IndentStep' parameter specifies how many spaces of indentation to add for each level.

Note: There is no reformatting in CDATA sections and Processing Instructions except for the newline changes, as the whitespace contained inside these sections may be important depending on what is contained in the section.

Note: Since MacOSX, the CR newline has become less common and the LF newline is used mostly like on other Unix systems. The `#PB_XML_MacNewline` is provided for completeness, but it is usually better to use `#PB_XML_LinuxNewline` even on MacOSX.

IndentStep (optional) The indent to apply (in characters) when using the `#PB_XML_ReFormat` or `#PB_XML_ReIndent` flags

Return value

None.

See Also

`ExportXML()` , `SaveXML()`

183.16 GetXMLEncoding

Syntax

```
Result = GetXMLEncoding(#XML)
```

Description

Returns the text encoding used for exporting/saving the given XML tree.

Parameters

#XML The XML to use.

Return value

It can be one of the following values:

```
#PB_Ascii  
#PB_Unicode (UTF16)  
#PB_UTF8
```

See Also

ExportXML() , SaveXML() , SetXMLEncoding()

183.17 SetXMLEncoding

Syntax

```
SetXMLEncoding(#XML, Encoding)
```

Description

Changes the text encoding used for exporting/saving the given XML tree.

Parameters

#XML The XML to use.

Encoding It can be one of the following values:

```
#PB_Ascii  
#PB_Unicode (UTF16)  
#PB_UTF8
```

This only affects the exporting /saving of the tree. The data in the #XML object is always stored in the PB string format (unicode). The encoding can be safely changed to #PB_Ascii for saving and then back to something else without losing any information in the tree in memory.

Return value

None.

See Also

ExportXML() , SaveXML() , GetXMLEncoding()

183.18 GetXMLStandalone

Syntax

```
Result = GetXMLStandalone(#XML)
```

Description

Returns the value of the "standalone" attribute in the XML declaration of the document.

Parameters

#XML The XML to use.

Return value

It can be one of the following values:

```
#PB_XML_Stand-aloneYes    : The document mode is standalone
#PB_XML_Stand-aloneNo     : The document mode is not standalone
#PB_XML_Stand-aloneUnset : The standalone mode is not specified in
the declaration
```

See Also

SetXMLStand-alone()

183.19 SetXMLStand-alone

Syntax

```
SetXMLStand-alone(#XML, Stand-alone)
```

Description

Changes the "standalone" attribute of the XML declaration when exporting/saving the document.

Parameters

#XML The XML to use.

Stand-alone It can be one of these values:

```
#PB_XML_Stand-aloneYes    : The document mode is standalone
#PB_XML_Stand-aloneNo     : The document mode is not standalone
#PB_XML_Stand-aloneUnset : The standalone mode is not specified
in the declaration
```

Since this library does not validate document type definitions (DTDs), the value of this attribute has no effect on the parsing/saving of documents with this library except that it is read from and written to the XML declaration. This value is however important when working with XML documents intended for validating parsers, that's why this command exists.

See Also

GetXMLStand-alone()

183.20 RootXMLNode

Syntax

```
Result = RootXMLNode(#XML)
```

Description

Returns the root node of the XML tree. This node is always present. It represents the XML document itself. The text contained in this node represents the whitespace outside of any XML node (there can be no text outside of nodes). The children of this node are the main node and any comments outside the main node. The type of this node is `#PB_Xml_Root`.

Parameters

`#XML` The XML to use.

Return value

Always returns a valid XML node if `#XML` is an existing XML tree.

See Also

`XMLNodeType()` , `MainXMLNode()`

183.21 MainXMLNode

Syntax

```
Result = MainXMLNode(#XML)
```

Description

Returns the main XML node of the tree. A valid XML document must have one "main" or "document" node which contains all other nodes. Except this node, there can only be comments on the first level below the root node . The type of this node is `#PB_Xml_Normal`.

Parameters

`#XML` The XML to use.

Return value

The main node, or zero if the tree has no main node (which happens if the tree is empty or the main node was deleted).

See Also

`XMLNodeType()` , `RootXMLNode()`

183.22 ChildXMLNode

Syntax

```
Result = ChildXMLNode(Node [, Index])
```

Description

Returns a child node of the given XML node.

Parameters

Node The XML node to get the child.

Index (optional) The index of the child node. The first index starts from 1. If omitted, the first node is returned.

Return value

The requested child node or zero if there are no children or index is out of range.

See Also

ParentXMLNode()

183.23 ParentXMLNode

Syntax

```
Result = ParentXMLNode(Node)
```

Description

Returns the parent node of the given XML node. Every XML node has a parent, except the root node .

Parameters

Node The XML node to get the parent.

Return value

The parent node or zero if 'Node' was the root node.

See Also

ChildXMLNode() , RootXMLNode()

183.24 XMLChildCount

Syntax

```
Result = XMLChildCount(Node)
```

Description

Returns the number of child nodes inside the specified XML node.

Parameters

Node The XML node to count the children.

Return value

The number of child nodes inside the specified XML node.

183.25 NextXMLNode

Syntax

```
Result = NextXMLNode(Node)
```

Description

Returns the next XML node after the given one (inside their parent node).

Parameters

Node The XML node to use.

Return value

The node after the specified node or zero if there are no more nodes.

See Also

PreviousXMLNode()

183.26 PreviousXMLNode

Syntax

```
Result = PreviousXMLNode(Node)
```

Description

Returns the previous XML node from the given one (inside their parent node).

Parameters

Node The XML node to use.

Return value

The node before the specified node or zero if the given node was the first child of its parent.

See Also

NextXMLNode()

183.27 XMLNodeFromPath

Syntax

```
Result = XMLNodeFromPath(ParentNode, Path$)
```

Description

Returns the XML node inside ParentNode who's relation to ParentNode is described through 'Path\$'. XMLNodePath() can be used to get such a path to a node.

Parameters

ParentNode The parent XML node.

Path\$ Contains a list of node names separated by '/' to indicate the way to follow from the parent to the target node. For example "childtag/subchildtag" specifies the first node with name "subchildtag" inside the first node with name "childtag" inside ParentNode. A node name can have an index (one based) to specify which of multiple child tags of the same name should be selected. "childtag/subchildtag[3]" specifies the 3rd "subchildtag" inside the first "childtag" of ParentNode.

Other rules:

- If a path starts with '/' it is relative to the tree's root. No matter which node ParentNode specifies.
- A Wildcard "*" can be used instead of a tag name to specify that any tag is to be selected.
- A Comment node has the tagname "#comment"
- A CDATA node has the tagname "#cdata"
- A DTD node has the tagname "#dtd"
- A Processing Instruction node has the tagname "#instruction"

Some examples of valid paths:

```
"/mainnode/#comment[4]" - the 4th comment inside the
"mainnode" node inside the root of the tree
"*[10]"                  - the 10th node (of any type) inside
ParentNode
"*/*/*"                  - the 1st node 3 levels below
ParentNode independent of its type
"node[3]/*[3]/#cdata"    - the first CDATA section inside the
3rd node of any kind inside the 3rd "node" node inside
ParentNode
```

Note: This command is no implementation of the XPath specification. The syntax used and understood by this command is only a small subset of XPath. This means a path returned from XMLNodePath() is a valid XPath query, but this command only understands the syntax described here, not just any XPath query.

Return value

The target node or zero if the path did not lead to a valid node.

See Also

XMLNodePath()

183.28 XMLNodeFromID

Syntax

`Result = XMLNodeFromID(#XML, ID$)`

Description

In valid XML, if a node has an attribute called "ID", the value of this attribute must be unique within the XML document. This function can be used to search for a node in the document based on its ID attribute.

Parameters

#XML The XML to use.

ID\$ The ID value to look for.

Return value

The node with the given ID tag or zero if no such node exists within the tree.

183.29 XMLNodeType

Syntax

`Result = XMLNodeType(Node)`

Description

Returns the type of the given XML node.

Parameters

Node The XML node to use.

Return value

It can be one of the following:

#PB_XML_Root

This is the tree's root node. It represents the document itself. This node cannot be created or deleted manually. Inside the root node, there can be only one node of type **#PB_XML_Normal** and also no plain text. (this is required to be a well-formed XML document)

#PB_XML_Normal

This is a normal node in the tree. It can have a list of attributes and contain text and/or child nodes.

Example: `<node attribute="hello"> contained text </node>`

#PB_XML_Comment

This node represents a comment. It can have no children or attributes. Its text represents the content of the comment.

Example: `<!-- comment text -->`

#PB_XML_CData

This is a CData section. A CData section contains only text. Its content is not interpreted by the parser so it can contain unescaped "<" and ">" characters for example. CData sections can be used to include other markup or code inside a document without having to escape all characters that could be interpreted as XML.

Example: <![CDATA[cdata content]]>

#PB_XML_DTD

This is a document type declaration (DTD). This library does not use a validating parser, so these declarations are actually ignored when parsing a document. In order to save them back correctly, they are contained within such a DTD node. The text content of the node is the entire DTD tag. It can be read and modified through commands like SetXMLNodeText() and will be written back to the document when exporting/saving without modification. The SetXMLStandalone() command could be useful as well when working with DTDs.

Example: <!DOCTYPE name SYSTEM "external dtd uri">

#PB_XML_Instruction

This node represents a Processing Instruction. Processing Instructions contain information that is intended to be interpreted/executed by the target application. They have a name to specify the content of the instruction and the instruction data which can be accessed with GetXMLNodeText() .

Example: <?php if (...) ... ?>

(here "php" is the node name, and the rest up to the "?>" is the node text.)

183.30 GetXMLNodeText

Syntax

Result\\$_ = GetXMLNodeText(Node)

Description

Returns the text inside the given XML node.

Parameters

Node The XML node to use.

For a node of type #PB_XML_Normal, this is all text and whitespace within the node that is not contained within a child node.

For the root node , this is all whitespace outside of the main node. (there can be no text outside of the main node)

For #PB_XML_Comment or #PB_XML_CData nodes, this is all text contained in the node.

Return value

The text inside the given XML node.

See Also

SetXMLNodeText()

183.31 SetXMLNodeText

Syntax

```
SetXMLNodeText(Node, Text$)
```

Description

Changes the text contained within the given XML node. See GetXMLNodeText() for more information.

Parameters

Node The XML node to set the text.

Text\$ The new text to set.

Return value

None.

Remarks

If the node contains children, changing its contained text may require an adjustment of the child nodes offset values as well.

See Also

GetXMLNodeText()

183.32 GetXMLNodeOffset

Syntax

```
Result = GetXMLNodeOffset(Node)
```

Description

Returns the character offset of this node within its parent.

Parameters

Node The XML node to get the offset.

Return value

The number of characters in the parent nodes text data that lie between this node and the previous child node. So, if this node directly follows the previous one, this value will be zero.

See Also

SetXMLNodeOffset()

183.33 SetXMLNodeOffset

Syntax

```
SetXMLNodeOffset(Node, Offset)
```

Description

Changes the character offset of the given XML node within its parent nodes text data . See GetXMLNodeOffset() for more information.

Parameters

Node The XML node to set the offset.

Offset The new offset, in characters.

Return value

None.

See Also

GetXMLNodeOffset()

183.34 GetXMLNodeName

Syntax

```
Result\[extract_itex] = GetXMLNodeName(Node)
```

Description

Returns the tagname of the given XML node.

Parameters

Node The XML node to get the name.

Return value

The tagname of the given XML node. If the node is not of type `#PB_XML_Normal` or `#PB_XML_Instruction`, an empty string is returned.

See Also

SetXMLNodeName()

183.35 SetXMLNodeName

Syntax

```
SetXMLNodeName(Node, Name$)
```

Description

Changes the tagname of the given XML node. If the node is not of type `#PB_XML_Normal` or `#PB_XML_Instruction`, this function is ignored.

Parameters

Node The XML node to set the name.

Name\$ The new tagname.

Return value

None.

Remarks

Reminder, according to the XML standard, the name of a node must follow these rules:

- Case sensitive
- Must start with a letter or an underscore '_'
- Do not start with "XML" (Xml, xml, etc.)
- Allowed are: letters, numbers, hyphen '-' or dot '.' but not () * + , / "# \$ % & ! ' ; < = > @ [\] ^ { | } * *' :
- Spaces are not allowed

See Also

GetXMLNodeName()

183.36 XMLNodePath

Syntax

```
Result\$$ = XMLNodePath(Node [, ParentNode])
```

Description

Returns a string representing the relation between Node and ParentNode.

Parameters

Node The XML node to get the path.

ParentNode (optional) It has to be a parent or grandparent of 'Node'. If omitted, the root node of the tree is used.

Return value

A string representing the relation between 'Node' and 'ParentNode'. See XMLNodeFromPath() for a description of the returned path string.

See Also

XMLNodeFromPath()

183.37 GetXMLAttribute

Syntax

```
Result\$_ = GetXMLAttribute(Node, Attribute$)
```

Description

Returns the value of an attribute in the given XML node.

Parameters

Node The XML node to get the attribute.

Attribute\$ The attribute name.

Return value

The value of the attribute in the specified XML node. If the attribute does not exist an empty string is returned. Only nodes of type `#PB_XML_Normal` can have attributes. For all other node types the compiler raises an error.

See Also

SetXMLAttribute() , RemoveXMLAttribute()

183.38 SetXMLAttribute

Syntax

```
SetXMLAttribute(Node, Attribute$, Value$)
```

Description

Sets the value of the attribute on the given XML node. If the attribute does not exist yet, it will be added.

Parameters

Node The XML node to set the attribute.

Attribute\$ The attribute name to change or create.

Value\$ The new value to set.

Return value

None.

Remarks

Only nodes of type `#PB_XML_Normal` can have attributes. For all other node types this function is ignored.

See Also

`GetXMLAttribute()` , `RemoveXMLAttribute()`

183.39 RemoveXMLAttribute

Syntax

```
RemoveXMLAttribute(Node, Attribute$)
```

Description

Removes the attribute from the given XML node.

Parameters

Node The XML node to remove the attribute.

Attribute\$ The attribute name to remove. If the attribute doesn't exists, nothing happens.

Return value

None.

Remarks

Only nodes of type `#PB_XML_Normal` can have attributes. For all other node types this function is ignored.

See Also

`GetXMLAttribute()` , `SetXMLAttribute()`

183.40 ExamineXMLAttributes

Syntax

```
Result = ExamineXMLAttributes(Node)
```

Description

Starts to examine the attributes of the given XML node.

Parameters

Node The XML node to examine.

Return value

Nonzero if the node is of type `#PB_XML_Normal` and zero else (as such nodes cannot have attributes).

See Also

`NextXMLAttribute()` , `XMLAttributeName()` , `XMLAttributeValue()`

183.41 NextXMLAttribute

Syntax

```
Result = NextXMLAttribute(Node)
```

Description

This function must be called after `ExamineXMLAttributes()` to move step by step through the attributes of the given XML node.

Parameters

Node The XML node being examined with `ExamineXMLAttributes()` .

Return value

Zero if there are no more attributes or nonzero if there still is one.

See Also

`ExamineXMLAttributes()` , `XMLAttributeName()` , `XMLAttributeValue()`

183.42 XMLAttributeName

Syntax

```
Result\$_ = XMLAttributeName(Node)
```

Description

After calling `ExamineXMLAttributes()` and `NextXMLAttribute()` this function returns the attribute name of the currently examined attribute on the given XML node.

Parameters

Node The XML node being examined with `ExamineXMLAttributes()` .

Return value

The attribute name of the currently examined attribute on the given XML node.

See Also

`ExamineXMLAttributes()` , `NextXMLAttribute()` , `XMLAttributeValue()`

183.43 XMLAttributeValue

Syntax

```
Result\$$ = XMLAttributeValue(Node)
```

Description

After calling `ExamineXMLAttributes()` and `NextXMLAttribute()` this function returns the attribute value of the currently examined attribute on the given XML node.

Parameters

Node The XML node being examined with `ExamineXMLAttributes()` .

Return value

The attribute value of the currently examined attribute on the given XML node.

See Also

`ExamineXMLAttributes()` , `NextXMLAttribute()` , `XMLAttributeName()`

183.44 CreateXMLNode

Syntax

```
Result = CreateXMLNode(ParentNode, Name$ [, PreviousNode [, Type]])
```

Description

Creates a new XML node and inserts it into the given parent node.

Parameters

ParentNode The node into which to insert the new node. To insert the new node at the root of the tree, `RootXMLNode()` can be used here.

Name\$ The node name\$. Can be an empty string if the node name isn't required.

PreviousNode (optional) A childnode of 'ParentNode' after which the new node should be inserted. If this value is 0 or not specified, the new node is inserted as the first child of its parent. If this value is -1, the node is inserted as the last child of its parent.

Type (optional) The type for the new node. The default is `#PB_XML_Normal`. Note that the node type cannot be changed after the node was created.

Return value

The new XML node if it was created successfully or zero if no node could be inserted at this point.

Remarks

The following rules must be followed for a successful insertion:

- ParentNode may not be of type `#PB_XML_Comment` or `#PB_XML_CData`
- PreviousNode must be a direct child of ParentNode (if it is specified)
- A node of type `#PB_XML_Root` cannot be created manually
- If the XML tree already has a main node , only nodes other than `#PB_XML_Normal` and `#PB_XML_CData` can be inserted at the root level

See Also

DeleteXMLNode()

183.45 CopyXMLNode

Syntax

```
Result = CopyXMLNode(Node, ParentNode [, PreviousNode])
```

Description

Copies the given XML node and all its contained text and children to a new location. This function can even be used to copy nodes into a different XML tree. For moving a complete node to a new location MoveXMLNode() can be used.

Parameters

Node The node to copy.

ParentNode The node into which to insert the new node. To insert the new node at the root of the tree, RootXMLNode() can be used here.

PreviousNode (optional) A childnode of 'ParentNode' after which the new node should be inserted. If this value is 0 or not specified, the new node is inserted as the first child of its parent. If this value is -1, the node is inserted as the last child of its parent.

Return value

The new XML node if it was copied successfully or zero if copying was not possible.

Remarks

The following rules must be followed for a successful copying:

- The root node of a tree cannot be copied
- ParentNode may not be of type `#PB_XML_Comment` or `#PB_XML_CData`
- PreviousNode must be a direct child of ParentNode (if it is specified)
- If the XML tree already has a main node , only nodes other than `#PB_XML_Normal` and `#PB_XML_CData` can be inserted at the root level

See Also

DeleteXMLNode() , MoveXMLNode()

183.46 MoveXMLNode

Syntax

```
Result = MoveXMLNode(Node, ParentNode [, PreviousNode])
```

Description

Moves the given XML node and all its contained text and children to a new location. This function can even be used to move nodes into a different XML tree. For copying a complete node to a new location CopyXMLNode() can be used.

Parameters

Node The node to move.

ParentNode The node into which to insert the node. To insert the node at the root of the tree, RootXMLNode() can be used here.

PreviousNode (optional) A childnode of 'ParentNode' after which the node should be inserted. If this value is 0 or not specified, the node is inserted as the first child of its parent. If this value is -1, the node is inserted as the last child of its parent.

Return value

Nonzero if the move was successful or zero if the node could not be moved.

Remarks

The following rules must be followed for a successful move:

- The root node of a tree cannot be moved
- ParentNode may not be of type #PB_XML_Comment or #PB_XML_CData
- PreviousNode must be a direct child of ParentNode (if it is specified)
- Node and PreviousNode cannot be equal
- ParentNode cannot be equal to, or a child of Node (a node cannot be moved into itself)
- If the XML tree already has a main node, only nodes other than #PB_XML_Normal and #PB_XML_CData can be inserted at the root level

See Also

DeleteXMLNode() , CopyXMLNode()

183.47 DeleteXMLNode

Syntax

```
DeleteXMLNode(Node)
```

Description

Deletes the specified XML node and all its contained text and children from its XML tree.

Parameters

Node The node to delete.

Remarks

The root node of a tree cannot be deleted.

See Also

CreateXMLNode()

183.48 ResolveXMLNodeName

Syntax

```
Result\$_ = ResolveXMLNodeName(Node [, Separator$])
```

Description

Returns the expanded name of the given node in a document that uses XML namespaces. The expanded name consists of the namespace uri (if any) and the local node name, separated by the separator character given in 'Separator\$'.

Parameters

Node The node to use.

Separator\$ (optional) The separator to use when concatenating the namespace and the local node name. The default separator character is "/".

Return value

In a document using namespaces, returns the expanded name of the node if it could be correctly resolved or an empty string if a namespace prefix is used that is never declared (which is invalid). In a document without namespaces, returns the node name itself.

See Also

ResolveXMLAttributeName()

183.49 ResolveXMLAttributeName

Syntax

```
Result\$_ = ResolveXMLAttributeName(Node, Attribute$ [, Separator$])
```

Description

Returns the expanded name of the given node's attribute in a document that uses XML namespaces. The expanded name consists of the namespace uri (if any) and the local attribute name, separated by the separator character given in 'Separator\$'.

Parameters

Node The node to use.

Attribute\$ The attribute to resolve.

Separator\$ (optional) The separator to use when concatenating the namespace and the local attribute name. The default separator character is "/".

Return value

In a document using namespaces, returns the expanded name of the attribute if it could be correctly resolved or an empty string if a namespace prefix is used that is never declared (which is invalid).

In a document without namespaces, returns the attribute name itself.

Remarks

Note: Unlike with node names, the default namespace is not applied to attribute names that do not have a namespace prefix. So attribute names without a namespace prefix simply get their local name returned.

See Also

ResolveXMLNodeName()

183.50 InsertXMLArray

Syntax

```
Result = InsertXMLArray(ParentNode, Array() [, PreviousNode])
```

Description

Insert the specified Array() as a new XML node into the given parent node.

Parameters

ParentNode The node into which to insert the new node. To insert the new node at the root of the tree, RootXMLNode() can be used here.

Array() The array to insert into the XML.

PreviousNode (optional) A childnode of 'ParentNode' after which the new node should be inserted. If this value is 0 or not specified, the new node is inserted as the first child of its parent. If this value is -1, the node is inserted as the last child of its parent.

Return value

The new XML node if it was created successfully or zero if no node could be inserted at this point.

Remarks

The rules specified in the CreateXMLNode() for where a new node can be inserted also apply to this function.

The inserted node is named "array" and the contained element nodes are named "element". If the array has multiple dimension, each element will have attributes indicating the coordinate of the element within the array, with each coordinate named "a", "b" and so forth. See below for an example of the created XML.

Example

```
1      ; This example produces the following XML tree:
2      ;
3      ; <array>
4      ;   <element>red</element>
5      ;   <element>green</element>
6      ;   <element>blue</element>
7      ; </array>
8      ;
9      Dim Colors$(2)
10     Colors$(0) = "red"
11     Colors$(1) = "green"
12     Colors$(2) = "blue"
13
14     If CreateXML(0)
15         InsertXMLArray(RootXMLNode(0), Colors$())
16         FormatXML(0, #PB_XML_ReFormat)
17         Debug ComposeXML(0)
18     EndIf
```

Example

```
1      ; This example produces the following XML tree:
2      ;
3      ; <array>
4      ;   <element a="0" b="0">0</element>
5      ;   <element a="0" b="1">1</element>
6      ;   <element a="1" b="0">10</element>
7      ;   <element a="1" b="1">11</element>
8      ;   <element a="2" b="0">20</element>
9      ;   <element a="2" b="1">21</element>
10     ; </array>
11     ;
12     Dim MultiArray(2, 1)
13     For a = 0 To 2
14         For b = 0 To 1
15             MultiArray(a, b) = a * 10 + b
16         Next b
17     Next a
18
19     If CreateXML(0)
20         InsertXMLArray(RootXMLNode(0), MultiArray())
21         FormatXML(0, #PB_XML_ReFormat)
22         Debug ComposeXML(0)
23     EndIf
```

See Also

[ExtractXMLArray\(\)](#) , [InsertXMLList\(\)](#) , [InsertXMLMap\(\)](#) , [InsertXMLStructure\(\)](#) ,

183.51 InsertXMLList

Syntax

```
Result = InsertXMLList(ParentNode, List() [, PreviousNode])
```

Description

Insert the specified List() as a new XML node into the given parent node.

Parameters

ParentNode The node into which to insert the new node. To insert the new node at the root of the tree, RootXMLNode() can be used here.

List() The list to insert into the XML.

PreviousNode (optional) A childnode of 'ParentNode' after which the new node should be inserted. If this value is 0 or not specified, the new node is inserted as the first child of its parent. If this value is -1, the node is inserted as the last child of its parent.

Return value

The new XML node if it was created successfully or zero if no node could be inserted at this point.

Remarks

The rules specified in the CreateXMLNode() for where a new node can be inserted also apply to this function.

The inserted node is named "list" and the contained element nodes are named "element". See below for an example of the created XML.

Example

```
1      ; This example produces the following XML tree:
2      ;
3      ; <list>
4      ;   <element>square</element>
5      ;   <element>circle</element>
6      ;   <element>triangle</element>
7      ; </list>
8      ;
9      NewList Shapes$()
10     AddElement(Shapes$()): Shapes$() = "square"
11     AddElement(Shapes$()): Shapes$() = "circle"
12     AddElement(Shapes$()): Shapes$() = "triangle"
13
14     If CreateXML(0)
15         InsertXMLList(RootXMLNode(0), Shapes$())
16         FormatXML(0, #PB_XML_ReFormat)
17         Debug ComposeXML(0)
18     EndIf
```

Example

```
1      ; This example produces the following XML tree:
2      ;
3      ; <list>
4      ;   <element>
5      ;       <x>100</x>
```

```

6      ;      <y>200</y>
7      ;      </element>
8      ;      <element>
9      ;      <x>200</x>
10     ;      <y>400</y>
11     ;      </element>
12     ; </list>
13     ;
14     Structure Position
15         x.l
16         y.l
17     EndStructure
18
19     NewList Positions.Position()
20
21     For i = 1 To 2
22         AddElement(Positions())
23         Positions()\x = 100 * i
24         Positions()\y = 200 * i
25     Next i
26
27     If CreateXML(0)
28         InsertXMLList(RootXMLNode(0), Positions())
29         FormatXML(0, #PB_XML_ReFormat)
30         Debug ComposeXML(0)
31     EndIf

```

See Also

ExtractXMLList() , InsertXMLArray() , InsertXMLMap() , InsertXMLStructure()

183.52 InsertXMLMap

Syntax

Result = InsertXMLMap(ParentNode, Map() [, PreviousNode])

Description

Insert the specified Map() as a new XML node into the given parent node.

Parameters

ParentNode The node into which to insert the new node. To insert the new node at the root of the tree, RootXMLNode() can be used here.

Map() The map to insert into the XML.

PreviousNode (optional) A childnode of 'ParentNode' after which the new node should be inserted. If this value is 0 or not specified, the new node is inserted as the first child of its parent. If this value is -1, the node is inserted as the last child of its parent.

Return value

The new XML node if it was created successfully or zero if no node could be inserted at this point.

Remarks

The rules specified in the CreateXMLNode() for where a new node can be inserted also apply to this function.

The inserted node is named "map" and the contained element nodes are named "element". Each element node will have an attribute named "key" containing the map key of the element. See below for an example of the created XML.

Example

```
1      ; This example produces the following XML tree:
2      ;
3      ; <map>
4      ;   <element key="DE">Germany</element>
5      ;   <element key="US">United States</element>
6      ;   <element key="FR">France</element>
7      ; </map>
8      ;
9      NewMap Countries.s()
10     Countries("DE") = "Germany"
11     Countries("FR") = "France"
12     Countries("US") = "United States"
13
14     If CreateXML(0)
15         InsertXMLMap(RootXMLNode(0), Countries())
16         FormatXML(0, #PB_XML_ReFormat)
17         Debug ComposeXML(0)
18     EndIf
```

See Also

ExtractXMLMap() , InsertXMLArray() , InsertXMLList() , InsertXMLStructure()

183.53 InsertXMLStructure

Syntax

```
Result = InsertXMLStructure(ParentNode, *Buffer, Structure [,
    PreviousNode])
```

Description

Insert the specified structure memory as a new XML node into the given parent node.

Parameters

ParentNode The node into which to insert the new node. To insert the new node at the root of the tree, RootXMLNode() can be used here.

***Buffer** The address of the structure to insert into the XML tree.

Structure The type of the structure to insert.

PreviousNode (optional) A childnode of 'ParentNode' after which the new node should be inserted. If this value is 0 or not specified, the new node is inserted as the first child of its parent. If this value is -1, the node is inserted as the last child of its parent.

Return value

The new XML node if it was created successfully or zero if no node could be inserted at this point.

Remarks

The rules specified in the CreateXMLNode() for where a new node can be inserted also apply to this function.

The inserted node is named like the structure. Each structure element is added as a sub-node inside the structure node. Any '*' or '\$' character is stripped from the name of the structure element. If the structure element contains an array, list, map or structure, more nodes are added recursively. See below for an example of the created XML.

Example

```
1      ; This example produces the following XML tree:
2      ;
3      ; <Person>
4      ;   <Name>John Smith</Name>
5      ;   <Age>42</Age>
6      ;   <Books>
7      ;       <element>Investing For Dummies</element>
8      ;       <element>A Little Bit of Everything For Dummies</element>
9      ;   </Books>
10     ; </Person>
11     ;
12     Structure Person
13         Name$
14         Age.l
15         List Books.s()
16     EndStructure
17
18     Define P.Person
19
20     P\Name$ = "John Smith"
21     P\Age   = 42
22     AddElement(P\Books()): P\Books() = "Investing For Dummies"
23     AddElement(P\Books()): P\Books() = "A Little Bit of Everything
        For Dummies"
24
25     If CreateXML(0)
26         InsertXMLStructure(RootXMLNode(0), @P, Person)
27         FormatXML(0, #PB_XML_ReFormat)
28         Debug ComposeXML(0)
29     EndIf
```

See Also

ExtractXMLStructure() , InsertXMLArray() , InsertXMLList() , InsertXMLMap()

183.54 ExtractXMLArray

Syntax

```
ExtractXMLArray(Node, Array() [, Flags])
```

Description

Extract elements from the given XML node into the specified `Array()`. The array will be resized to the number of elements contained in the node.

Parameters

Node The XML node containing the array data.

Array() The array to fill with the XML elements. The array will be resized to have the same size as the number of element nodes. Any previous content of the array will be lost.

Flags (optional) If this parameter is set to `#PB_XML_NoCase` then the comparison of XML node and attribute names is performed case insensitive. The default is to be case sensitive.

Return value

None.

Remarks

The extraction is performed recursively if the array has a structure type. The XML nodes must have the form described in the `InsertXMLArray()` function. Nodes with different names are ignored by the extraction. If the array has more than one dimension, each element is expected to have attributes indicating the coordinates of the element named "a", "b" and so forth.

Example

```
1  Xml$ =  
   "<array><element>1</element><element>10</element><element>100</element></array>"  
2  
3  If ParseXML(0, Xml$) And XMLStatus(0) = #PB_XML_Success  
4      Dim MyArray(0) ; will be resized by the next call  
5      ExtractXMLArray(MainXMLNode(0), MyArray())  
6  
7      For i = 0 To ArraySize(MyArray())  
8          Debug MyArray(i)  
9      Next i  
10 Else  
11     Debug XMLError(0)  
12 EndIf
```

See Also

`InsertXMLArray()` , `ExtractXMLList()` , `ExtractXMLMap()` , `ExtractXMLStructure()`

183.55 ExtractXMLList

Syntax

```
ExtractXMLList(Node, List() [, Flags])
```

Description

Extract elements from the given XML node into the specified `List()`. The list will be cleared before extracting the elements.

Parameters

Node The XML node containing the list data.

List() The list to fill with the XML elements. Any previous content of the list will be lost.

Flags (optional) If this parameter is set to `#PB_XML_NoCase` then the comparison of XML node and attribute names is performed case insensitive. The default is to be case sensitive.

Return value

None.

Remarks

The extraction is performed recursively if the list has a structure type. The XML nodes must have the form described in the `InsertXMLList()` function. Nodes with different names are ignored by the extraction.

Example

```
1  Xml$ =  
   "<list><element>1</element><element>10</element><ELEMENT>100</ELEMENT></list>  
2  
3  If ParseXML(0, Xml$) And XMLStatus(0) = #PB_XML_Success  
4      NewList Values()  
5      ExtractXMLList(MainXMLNode(0), Values(), #PB_XML_NoCase)  
6  
7      ForEach Values()  
8          Debug Values()  
9      Next  
10 Else  
11     Debug XMLError(0)  
12 EndIf
```

See Also

`InsertXMLList()` , `ExtractXMLArray()` , `ExtractXMLMap()` , `ExtractXMLStructure()`

183.56 ExtractXMLMap

Syntax

```
ExtractXMLMap(Node, Map() [, Flags])
```

Description

Extract elements from the given XML node into the specified `Map()`. The map will be cleared before extracting the elements.

Parameters

Node The XML node containing the map data.

Map() The map to fill with the XML elements. Any previous content of the map will be lost.

Flags (optional) If this parameter is set to `#PB_XML_NoCase` then the comparison of XML node and attribute names is performed case insensitive. The default is to be case sensitive.

Return value

None.

Remarks

The extraction is performed recursively if the map has a structure type. The XML nodes must have the form described in the InsertXMLMap() function. Nodes with different names are ignored by the extraction.

Example

```
1  Xml$ = "<map><element key=" + Chr(34) + "theKey" + Chr(34) +  
    ">the value</element></map>"  
2  
3  If ParseXML(0, Xml$) And XMLStatus(0) = #PB_XML_Success  
4      NewMap Test.s()  
5      ExtractXMLMap(MainXMLNode(0), Test())  
6  
7      ForEach Test()  
8          Debug MapKey(Test()) + " -> " + Test()  
9      Next  
10 Else  
11     Debug XMLError(0)  
12 EndIf
```

See Also

InsertXMLMap() , ExtractXMLArray() , ExtractXMLList() , ExtractXMLStructure()

183.57 ExtractXMLStructure

Syntax

```
ExtractXMLStructure(Node, *Buffer, Structure [, Flags])
```

Description

Extract elements from the given XML node into the specified structure memory. The structure will be cleared before extracting XML nodes.

Parameters

Node The XML node containing the structure data.

***Buffer** The address of the structure memory to fill.

Structure The type of the structure to fill.

Flags (optional) If this parameter is set to #PB_XML_NoCase then the comparison of XML node and attribute names is performed case insensitive. The default is to be case sensitive.

Return value

None.

Remarks

The content of the structure memory are cleared before extracting the XML nodes. If a structure element does not have a corresponding node in the XML, it is left empty.

The XML nodes must have the form described in the `InsertXMLStructure()` function. Namely, each node must be named after a structure member with any '*' or '\$' characters stripped from the name. If a node for the same structure element exists multiple times, the first node will be used.

Example

```
1  Structure Person
2      Name$
3      Age.l
4  EndStructure
5
6  Xml$ = "<Person><Name>John Smith</Name><Age>42</Age></Person>"
7
8  If ParseXML(0, Xml$) And XMLStatus(0) = #PB_XML_Success
9      Define P.Person
10         ExtractXMLStructure(MainXMLNode(0), @P, Person)
11
12         Debug P\Name$
13         Debug P\Age
14     Else
15         Debug XMLError(0)
16     EndIf
```

See Also

`InsertXMLStructure()` , `ExtractXMLArray()` , `ExtractXMLList()` , `ExtractXMLMap()`