

SpiderBasic Reference Manual

2.21

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

February 4, 2019

Contents

I General	17
1 Introduction	18
2 Terms And Conditions	19
3 System requirements	20
4 Installation	21
5 Order	22
6 Contact	24
7 Acknowledgements	25
II The SpiderBasic Editor	26
8 Getting Started	27
9 Working with source files	29
10 Editing features	31
11 Managing projects	37
12 Compiling your programs	42
13 Using the built-in Tools	48
14 Using external tools	56
15 Getting Help	61
16 Customizing the IDE	63
17 Command-line options for the IDE	78
III Language Reference	80
18 Working with different number bases	81
19 Break : Continue	85
20 Using the command line compiler	86
21 Compiler Directives	88
22 Compiler Functions	92
23 Data	97

24	Debugger keywords in SpiderBasic	99
25	Define	101
26	Dim	103
27	Enumerations	105
28	For : Next	107
29	ForEach : Next	109
30	General Rules	111
31	Global	114
32	Handles and Numbers	116
33	If : Else : EndIf	118
34	Import : EndImport	119
35	Includes Functions	120
36	Inline Javascript	122
37	Interfaces	124
38	Licenses for the SpiderBasic applications (without using 3D engine)	126
39	Macros	138
40	Pointers and memory access	141
41	Module	144
42	NewList	147
43	NewMap	149
44	Others Commands	151
45	Procedures	153
46	Protected	156
47	Prototypes	158
48	Repeat : Until	159
49	Residents	160
50	Runtime	161
51	Select : EndSelect	163
52	Using several SpiderBasic versions on Windows	165
53	Shared	166
54	SpiderBasic objects	167
55	Static	170

56	Structures	172
57	Subsystems	175
58	Variables and Types	176
59	While : Wend	185
60	With : EndWith	186
IV	Library Reference	188
61	2DDrawing	189
61.1	Red	189
61.2	Green	190
61.3	Blue	190
61.4	Alpha	191
61.5	RGB	191
61.6	RGBA	192
61.7	BackColor	193
61.8	Box	193
61.9	RoundBox	194
61.10	Circle	195
61.11	DrawImage	196
61.12	DrawingFont	197
61.13	DrawingMode	198
61.14	StartDrawing	199
61.15	DrawText	200
61.16	Ellipse	201
61.17	FrontColor	202
61.18	Line	203
61.19	LineXY	204
61.20	Plot	205
61.21	Point	206
61.22	StopDrawing	207
61.23	TextHeight	207
61.24	TextWidth	208
61.25	OutputDepth	208
61.26	OutputWidth	209
61.27	OutputHeight	209
62	Accelerometer	211
62.1	StartAccelerometer	211
62.2	StopAccelerometer	211
62.3	AccelerometerX	212
62.4	AccelerometerY	213
62.5	AccelerometerZ	213
62.6	AccelerometerTime	214
63	Array	215
63.1	ArraySize	215
63.2	CopyArray	216
63.3	FreeArray	217
64	Cipher	218
64.1	AddCipherBuffer	218
64.2	AESEncoder	219
64.3	AESDecoder	221
64.4	StartFingerprint	222

64.5	FinishCipher	224
64.6	AddFingerprintBuffer	224
64.7	FinishFingerprint	225
64.8	IsFingerprint	225
64.9	Fingerprint	226
64.10	StringFingerprint	227
64.11	UseMD5Fingerprint	228
64.12	UseSHA1Fingerprint	229
64.13	UseSHA2Fingerprint	230
64.14	UseSHA3Fingerprint	231
64.15	UseCRC32Fingerprint	231
64.16	StartAESEncipher	232
65	Database	234
65.1	AffectedDatabaseRows	234
65.2	CloseDatabase	235
65.3	DatabaseColumns	235
65.4	DatabaseColumnIndex	236
65.5	DatabaseColumnName	236
65.6	DatabaseError	237
65.7	DatabaseID	238
65.8	DatabaseQuery	238
65.9	DatabaseUpdate	240
65.10	ExportDatabase	241
65.11	ExportDatabaseMemory	242
65.12	FinishDatabaseQuery	243
65.13	FirstDatabaseRow	243
65.14	GetDatabaseBlob	244
65.15	GetDatabaseDouble	245
65.16	GetDatabaseFloat	246
65.17	GetDatabaseLong	246
65.18	GetDatabaseQuad	247
65.19	GetDatabaseString	247
65.20	CheckDatabaseNull	248
65.21	IsDatabase	249
65.22	NextDatabaseRow	249
65.23	OpenDatabase	250
65.24	SetDatabaseBlob	250
65.25	SetDatabaseString	252
65.26	SetDatabaseLong	252
65.27	SetDatabaseQuad	253
65.28	SetDatabaseFloat	254
65.29	SetDatabaseDouble	255
65.30	SetDatabaseNull	255
66	Date	257
66.1	AddDate	257
66.2	Date	258
66.3	Day	259
66.4	DayOfWeek	259
66.5	DayOfYear	260
66.6	Month	261
66.7	Year	261
66.8	Hour	262
66.9	Minute	262
66.10	Second	263
66.11	FormatDate	264
66.12	ParseDate	265

67	Debugger	266
67.1	ShowDebugOutput	266
67.2	ClearDebugOutput	267
68	Desktop	268
68.1	ExamineDesktops	268
68.2	DesktopDepth	269
68.3	DesktopFrequency	269
68.4	DesktopHeight	270
68.5	DesktopX	271
68.6	DesktopY	271
68.7	DesktopMouseX	272
68.8	DesktopMouseY	273
68.9	DesktopName	274
68.10	DesktopWidth	274
69	Dialog	276
69.1	CreateDialog	276
69.2	DialogError	277
69.3	DialogGadget	277
69.4	DialogWindow	278
69.5	DialogID	278
69.6	FreeDialog	279
69.7	IsDialog	279
69.8	OpenXMLDialog	280
69.9	RefreshDialog	288
70	File	289
70.1	CloseFile	289
70.2	CreateFile	290
70.3	Eof	291
70.4	ExportFile	292
70.5	ExportFileMemory	293
70.6	FetchData	294
70.7	FileID	295
70.8	FileSeek	295
70.9	IsFile	296
70.10	FileProgress	297
70.11	Loc	298
70.12	Lof	298
70.13	ReadAsciiCharacter	299
70.14	ReadByte	299
70.15	ReadCharacter	300
70.16	ReadDouble	301
70.17	OpenFile	301
70.18	ReadFile	303
70.19	ReadFloat	305
70.20	ReadInteger	306
70.21	ReadLong	306
70.22	ReadQuad	307
70.23	ReadData	307
70.24	ReadString	308
70.25	ReadStringFormat	309
70.26	ReadUnicodeCharacter	310
70.27	ReadWord	310
70.28	WriteAsciiCharacter	311
70.29	WriteByte	311
70.30	WriteCharacter	312
70.31	WriteDouble	313

70.32	WriteFloat	313
70.33	WriteInteger	314
70.34	WriteLong	314
70.35	WriteData	315
70.36	WriteQuad	316
70.37	WriteString	317
70.38	WriteStringN	317
70.39	WriteStringFormat	318
70.40	WriteUnicodeCharacter	319
70.41	WriteWord	320
71	Font	321
71.1	FreeFont	321
71.2	FontID	322
71.3	IsFont	323
71.4	LoadFont	323
72	Gadget	325
72.1	AddGadgetColumn	325
72.2	AddGadgetItem	326
72.3	ButtonImageGadget	327
72.4	ButtonGadget	328
72.5	CalendarGadget	330
72.6	CanvasGadget	331
72.7	CanvasOutput	336
72.8	CanvasVectorOutput	336
72.9	CheckBoxGadget	337
72.10	ClearGadgetItems	338
72.11	CloseGadgetList	339
72.12	ComboBoxGadget	340
72.13	ContainerGadget	342
72.14	CountGadgetItems	343
72.15	DateGadget	344
72.16	DisableGadget	346
72.17	EditorGadget	346
72.18	FrameGadget	348
72.19	FreeGadget	349
72.20	GadgetID	350
72.21	GadgetToolTip	350
72.22	GadgetX	351
72.23	GadgetY	352
72.24	GadgetHeight	353
72.25	GadgetType	353
72.26	GadgetWidth	355
72.27	GetActiveGadget	355
72.28	GetGadgetAttribute	356
72.29	GetGadgetColor	356
72.30	GetGadgetData	358
72.31	GetGadgetItemAttribute	358
72.32	GetGadgetItemData	359
72.33	GetGadgetState	360
72.34	GetGadgetItemText	361
72.35	GetGadgetItemState	362
72.36	GetGadgetText	363
72.37	HideGadget	364
72.38	HyperLinkGadget	365
72.39	ImageGadget	366
72.40	IsGadget	367
72.41	ListIconGadget	368

72.42	ListViewGadget	371
72.43	OpenGadgetList	373
72.44	OptionGadget	373
72.45	PanelGadget	374
72.46	ProgressBarGadget	376
72.47	RemoveGadgetColumn	378
72.48	RemoveGadgetItem	378
72.49	ResizeGadget	379
72.50	ScrollAreaGadget	380
72.51	SetActiveGadget	382
72.52	SetGadgetAttribute	383
72.53	SetGadgetColor	384
72.54	SetGadgetData	385
72.55	SetGadgetFont	386
72.56	SetGadgetItemAttribute	387
72.57	SetGadgetItemData	388
72.58	SetGadgetItemImage	389
72.59	SetGadgetItemState	390
72.60	SetGadgetItemText	391
72.61	SetGadgetState	392
72.62	SetGadgetText	393
72.63	SpinGadget	394
72.64	SplitterGadget	395
72.65	StringGadget	397
72.66	TextGadget	399
72.67	TrackBarGadget	401
72.68	TreeGadget	402
72.69	UseGadgetList	404
72.70	WebGadget	405
72.71	BindGadgetEvent	406
72.72	UnbindGadgetEvent	407
73	Geolocation	409
73.1	StartGeolocation	409
73.2	StopGeolocation	410
73.3	GeolocationLatitude	410
73.4	GeolocationLongitude	411
73.5	GeolocationAltitude	411
73.6	GeolocationSpeed	412
73.7	GeolocationHeading	412
73.8	GeolocationTime	413
74	Http	414
74.1	HTTPRequest	414
74.2	URLDecoder	416
74.3	URLEncoder	416
75	Image	418
75.1	CopyImage	418
75.2	CreateImage	419
75.3	EncodeImage	420
75.4	FreeImage	420
75.5	GrabImage	421
75.6	ImageDepth	421
75.7	ImageFormat	422
75.8	ImageHeight	423
75.9	ImageID	423
75.10	ImageOutput	424
75.11	ImageVectorOutput	424

75.12	ImageWidth	425
75.13	IsImage	426
75.14	LoadImage	426
75.15	ResizeImage	428
75.16	ExportImage	428
76	InAppPurchase	430
76.1	RegisterAppProduct	430
76.2	FetchAppProducts	431
76.3	ExamineAppProducts	433
76.4	NextAppProduct	434
76.5	AppProductName	434
76.6	AppProductDescription	435
76.7	AppProductPrice	435
76.8	AppProductID	436
76.9	PurchaseAppProduct	437
77	Joystick	438
77.1	InitJoystick	438
77.2	ExamineJoystick	438
77.3	JoystickAxisX	439
77.4	JoystickAxisY	440
77.5	JoystickAxisZ	441
77.6	JoystickName	441
77.7	JoystickButton	442
78	Json	443
78.1	AddJSONElement	443
78.2	AddJSONMember	444
78.3	ClearJSONElements	445
78.4	ClearJSONMembers	446
78.5	ComposeJSON	446
78.6	CreateJSON	447
78.7	ExamineJSONMembers	448
78.8	ExportJSON	449
78.9	ExtractJSONArray	449
78.10	ExtractJSONList	451
78.11	ExtractJSONMap	452
78.12	ExtractJSONStructure	453
78.13	FreeJSON	454
78.14	GetJSONBoolean	455
78.15	GetJSONDouble	455
78.16	GetJSONElement	456
78.17	GetJSONFloat	457
78.18	GetJSONInteger	457
78.19	GetJSONMember	458
78.20	GetJSONString	459
78.21	GetJSONQuad	460
78.22	InsertJSONArray	460
78.23	InsertJSONList	461
78.24	InsertJSONMap	462
78.25	InsertJSONStructure	463
78.26	IsJSON	464
78.27	JSONArraySize	465
78.28	JSONExceptionLine	465
78.29	JSONExceptionMessage	466
78.30	JSONExceptionPosition	466
78.31	JSONMemberKey	467
78.32	JSONMemberValue	468

78.33	JSONObjectSize	468
78.34	JSONType	469
78.35	JsonValue	470
78.36	LoadJSON	471
78.37	NextJSONMember	472
78.38	ParseJSON	473
78.39	RemoveJSONElement	474
78.40	RemoveJSONMember	474
78.41	ResizeJSONElements	475
78.42	SetJSONArray	476
78.43	SetJSONBoolean	477
78.44	SetJSONDouble	477
78.45	SetJSONFloat	478
78.46	SetJSONInteger	479
78.47	SetJSONNull	480
78.48	SetJSONObject	481
78.49	SetJSONQuad	481
78.50	SetJSONString	482
79	Keyboard	484
79.1	InitKeyboard	484
79.2	ExamineKeyboard	484
79.3	KeyboardInkey	485
79.4	KeyboardPushed	486
79.5	KeyboardReleased	490
80	List	492
80.1	AddElement	492
80.2	ChangeCurrentElement	493
80.3	ClearList	494
80.4	CopyList	495
80.5	FreeList	496
80.6	ListSize	497
80.7	DeleteElement	497
80.8	FirstElement	498
80.9	InsertElement	500
80.10	LastElement	501
80.11	ListIndex	503
80.12	NextElement	504
80.13	PreviousElement	505
80.14	ResetList	505
80.15	SelectElement	506
80.16	SwapElements	507
80.17	MoveElement	509
80.18	PushListPosition	510
80.19	PopListPosition	511
80.20	MergeLists	512
80.21	SplitList	513
81	Map	515
81.1	AddMapElement	515
81.2	ClearMap	516
81.3	CopyMap	517
81.4	FreeMap	518
81.5	MapSize	518
81.6	DeleteMapElement	519
81.7	FindMapElement	520
81.8	MapKey	521
81.9	NextMapElement	522

81.10	ResetMap	523
81.11	PushMapPosition	523
81.12	PopMapPosition	525
82	Math	526
82.1	Abs	526
82.2	ACos	527
82.3	ACosH	527
82.4	ASin	528
82.5	ASinH	529
82.6	ATan	530
82.7	ATan2	530
82.8	ATanH	531
82.9	Cos	532
82.10	CosH	532
82.11	Degree	533
82.12	Exp	534
82.13	Infinity	535
82.14	Int	535
82.15	IntQ	536
82.16	IsInfinity	537
82.17	IsNaN	538
82.18	Pow	538
82.19	Log	539
82.20	Log10	540
82.21	Mod	540
82.22	NaN	541
82.23	Radian	542
82.24	Random	542
82.25	RandomSeed	543
82.26	Round	544
82.27	Sign	545
82.28	Sin	545
82.29	SinH	546
82.30	Sqr	547
82.31	Tan	547
82.32	TanH	548
83	Memory	550
83.1	AllocateMemory	550
83.2	AllocateStructure	551
83.3	CompareMemory	552
83.4	FreeMemory	553
83.5	FreeStructure	553
83.6	MemorySize	554
83.7	ReAllocateMemory	554
83.8	PeekA	555
83.9	PeekB	556
83.10	PeekC	556
83.11	PeekD	557
83.12	PeekL	558
83.13	PeekW	558
83.14	PeekF	559
83.15	PeekS	559
83.16	PeekU	560
83.17	PokeA	560
83.18	PokeB	561
83.19	PokeC	561
83.20	PokeD	562

83.21	PokeL	562
83.22	PokeW	563
83.23	PokeF	564
83.24	PokeS	564
83.25	PokeU	565
84	Menu	566
84.1	CloseSubMenu	566
84.2	CreateMenu	567
84.3	CreateImageMenu	568
84.4	CreatePopupMenu	569
84.5	CreatePopupImageMenu	570
84.6	DisplayPopupMenu	571
84.7	DisableMenuItem	572
84.8	FreeMenu	573
84.9	GetMenuItemText	573
84.10	GetMenuTitleText	574
84.11	IsMenu	574
84.12	MenuBar	575
84.13	MenuItem	576
84.14	MenuItemID	577
84.15	MenuTitle	578
84.16	OpenSubMenu	579
84.17	SetMenuItemText	580
84.18	SetMenuTitleText	580
84.19	BindMenuEvent	581
84.20	UnbindMenuEvent	582
85	Mouse	584
85.1	InitMouse	584
85.2	ExamineMouse	585
85.3	MouseButton	585
85.4	MouseDeltaX	586
85.5	MouseDeltaY	586
85.6	MouseLocate	587
85.7	MouseWheel	587
85.8	MouseX	588
85.9	MouseY	588
85.10	ReleaseMouse	589
86	RegularExpression	590
86.1	CreateRegularExpression	590
86.2	ExtractRegularExpression	591
86.3	FreeRegularExpression	592
86.4	IsRegularExpression	593
86.5	MatchRegularExpression	593
86.6	ReplaceRegularExpression	594
86.7	RegularExpressionError	595
87	Requester	597
87.1	NextSelectedFile	597
87.2	SelectedFileName	597
87.3	SelectedFileID	598
87.4	OpenFileRequester	598
87.5	UseGoogleDrive	600
88	Runtime	602
88.1	GetRuntimeInteger	602
88.2	GetRuntimeDouble	603
88.3	GetRuntimeString	603

88.4	IsRuntime	604
88.5	SetRuntimeDouble	604
88.6	SetRuntimeInteger	605
88.7	SetRuntimeString	605
89	Screen	607
89.1	ClearScreen	607
89.2	CloseScreen	607
89.3	FlipBuffers	608
89.4	IsScreenActive	609
89.5	ScreenWidth	609
89.6	ScreenHeight	610
89.7	ScreenDepth	610
89.8	SetFrameRate	611
89.9	OpenScreen	611
89.10	OpenWindowedScreen	612
89.11	ResizeScreen	613
90	Sort	615
90.1	SortArray	615
90.2	SortList	616
90.3	RandomizeArray	616
90.4	RandomizeList	617
91	Sound	619
91.1	GetSoundPosition	619
91.2	SetSoundPosition	620
91.3	FreeSound	620
91.4	InitSound	621
91.5	IsSound	621
91.6	LoadSound	622
91.7	PauseSound	623
91.8	ResumeSound	623
91.9	PlaySound	624
91.10	SoundStatus	625
91.11	SoundPan	625
91.12	SoundLength	626
91.13	SoundVolume	626
91.14	StopSound	627
92	Sprite	628
92.1	ClipSprite	628
92.2	CopySprite	629
92.3	CreateSprite	629
92.4	DisplaySprite	630
92.5	DisplayTransparentSprite	630
92.6	FreeSprite	631
92.7	InitSprite	632
92.8	IsSprite	632
92.9	LoadSprite	633
92.10	SpriteCollision	634
92.11	SpriteDepth	634
92.12	SpriteHeight	635
92.13	SpritePixelCollision	635
92.14	SpriteWidth	636
92.15	SpriteOutput	636
92.16	RotateSprite	637
92.17	SpriteQuality	638
92.18	ZoomSprite	638

93	String	639
93.1	Asc	639
93.2	Bin	640
93.3	Chr	641
93.4	CountString	641
93.5	FindString	642
93.6	Hex	643
93.7	InsertString	644
93.8	LCase	645
93.9	Left	645
93.10	Len	646
93.11	LSet	646
93.12	LTrim	647
93.13	Mid	648
93.14	RemoveString	648
93.15	ReplaceString	649
93.16	Right	650
93.17	RSet	651
93.18	RTrim	652
93.19	StringField	652
93.20	StrF	653
93.21	StrD	654
93.22	Str	654
93.23	StrU	655
93.24	ReverseString	656
93.25	Space	657
93.26	Trim	657
93.27	UCase	658
93.28	ValD	658
93.29	ValF	659
93.30	Val	660
94	System	662
94.1	ElapsedMilliseconds	662
94.2	CountProgramParameters	663
94.3	ProgramParameter	663
94.4	BatteryLevel	664
94.5	DeviceInfo	664
94.6	VibrateDevice	665
95	ToolBar	666
95.1	CreateToolBar	666
95.2	FreeToolBar	667
95.3	DisableToolBarButton	668
95.4	GetToolBarButtonState	669
95.5	IsToolBar	670
95.6	SetToolBarButtonState	670
95.7	ToolBarHeight	671
95.8	ToolBarImageButton	672
95.9	ToolBarSeparator	673
95.10	ToolBarToolTip	674
95.11	ToolBarID	675
96	TouchScreen	676
96.1	ExamineTouchEvent	676
96.2	TouchDeltaX	676
96.3	TouchDeltaY	677
96.4	TouchX	678
96.5	TouchY	678

96.6	TouchScreenPushed	679
97	VectorDrawing	681
97.1	StartVectorDrawing	682
97.2	StopVectorDrawing	683
97.3	VectorOutputWidth	683
97.4	VectorOutputHeight	684
97.5	VectorResolutionX	685
97.6	VectorResolutionY	685
97.7	VectorUnit	686
97.8	SaveVectorState	686
97.9	RestoreVectorState	687
97.10	BeginVectorLayer	688
97.11	EndVectorLayer	689
97.12	FillVectorOutput	690
97.13	ResetCoordinates	691
97.14	TranslateCoordinates	691
97.15	ScaleCoordinates	693
97.16	SkewCoordinates	694
97.17	RotateCoordinates	696
97.18	ConvertCoordinateX	698
97.19	ConvertCoordinateY	699
97.20	ResetPath	700
97.21	ClosePath	701
97.22	MovePathCursor	702
97.23	AddPathLine	703
97.24	AddPathArc	704
97.25	AddPathCurve	705
97.26	AddPathBox	707
97.27	AddPathCircle	708
97.28	AddPathEllipse	709
97.29	AddPathSegments	711
97.30	IsInsidePath	712
97.31	IsPathEmpty	714
97.32	StrokePath	714
97.33	DotPath	716
97.34	DashPath	717
97.35	CustomDashPath	718
97.36	FillPath	720
97.37	PathCursorX	721
97.38	PathCursorY	721
97.39	PathPointX	722
97.40	PathPointY	723
97.41	PathPointAngle	724
97.42	PathLength	724
97.43	PathBoundsX	725
97.44	PathBoundsY	726
97.45	PathBoundsWidth	727
97.46	PathBoundsHeight	728
97.47	PathSegments	728
97.48	VectorSourceColor	729
97.49	VectorSourceLinearGradient	730
97.50	VectorSourceCircularGradient	731
97.51	VectorSourceGradientColor	732
97.52	DrawVectorImage	733
97.53	DrawVectorText	734
97.54	VectorFont	735
97.55	VectorTextWidth	736
97.56	VectorTextHeight	738

98	Window	740
98.1	AddKeyboardShortcut	740
98.2	AddWindowTimer	742
98.3	RemoveWindowTimer	743
98.4	EventTimer	744
98.5	CloseWindow	745
98.6	DisableWindow	745
98.7	EventGadget	746
98.8	EventMenu	746
98.9	EventData	747
98.10	EventType	747
98.11	EventWindow	749
98.12	GetActiveWindow	749
98.13	GetWindowColor	750
98.14	GetWindowData	750
98.15	GetWindowTitle	751
98.16	HideWindow	751
98.17	IsWindow	752
98.18	OpenWindow	753
98.19	PostEvent	754
98.20	RemoveKeyboardShortcut	755
98.21	ResizeWindow	755
98.22	SetActiveWindow	756
98.23	SetWindowColor	756
98.24	SetWindowData	757
98.25	SetWindowTitle	757
98.26	StickyWindow	758
98.27	BindEvent	759
98.28	UnbindEvent	760
98.29	WindowBounds	761
98.30	WindowHeight	762
98.31	WindowID	763
98.32	WindowWidth	763
98.33	WindowX	764
98.34	WindowY	764
98.35	WindowMouseX	765
98.36	WindowMouseY	766
99	XML	767
99.1	IsXML	767
99.2	FreeXML	768
99.3	CreateXML	768
99.4	LoadXML	769
99.5	ParseXML	770
99.6	XMLStatus	770
99.7	XMLError	771
99.8	XMLErrorLine	772
99.9	XMLErrorPosition	772
99.10	ExportXML	773
99.11	ComposeXML	773
99.12	GetXMLEncoding	774
99.13	SetXMLEncoding	774
99.14	GetXMLStandalone	775
99.15	SetXMLStandalone	776
99.16	RootXMLNode	776
99.17	MainXMLNode	777
99.18	ChildXMLNode	777
99.19	ParentXMLNode	778
99.20	XMLChildCount	778

99.21	NextXMLNode	779
99.22	PreviousXMLNode	779
99.23	XMLNodeFromPath	780
99.24	XMLNodeFromID	781
99.25	XMLNodeType	781
99.26	GetXMLNodeText	782
99.27	SetXMLNodeText	783
99.28	GetXMLNodeOffset	783
99.29	SetXMLNodeOffset	784
99.30	GetXMLNodeName	785
99.31	XMLNodePath	785
99.32	GetXMLAttribute	786
99.33	SetXMLAttribute	786
99.34	RemoveXMLAttribute	787
99.35	ExamineXMLAttributes	787
99.36	NextXMLAttribute	788
99.37	XMLAttributeName	788
99.38	XMLAttributeValue	789
99.39	CreateXMLNode	789
99.40	DeleteXMLNode	790
99.41	ResolveXMLNodeName	791
99.42	ResolveXMLAttributeName	791

Part I

General

Chapter 1

Introduction

SpiderBasic is an "high-level" programming language based on established "BASIC" rules. It does share background with other "BASIC" compiler, but has its own syntax extensions. Learning SpiderBasic is very easy! SpiderBasic has been created for beginners and experts alike. Compilation time is extremely fast. 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 structures, procedures, dynamic lists and much more. For the experienced coder, there are no problems gaining access to external third party libraries.

The main features of SpiderBasic

- Huge set of internal commands (500+) to quickly and easily build applications or games
- BASIC based keywords
- Very fast compiler which creates optimized apps
- Procedure and structure support for advanced programming
- Built-in containers like array, list and map
- Strong types, strong syntax to avoid programming mistakes
- Full unicode support
- Namespace support for easy code reuse
- Easy but very fast 2D game support through WebGL
- Inlined JavaScript support for extensibility
- Dedicated editor and development environment
- Available on Windows, MacOS X and Linux
- Very close to PureBasic, which allow to port easily an application to the desktop

Chapter 2

Terms And Conditions

This program is provided **"AS IS"**. Fantaisie Software are NOT responsible for any damage (or damages) attributed to SpiderBasic. You are warned that you use SpiderBasic 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.

SpiderBasic 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.

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

Chapter 3

System requirements

SpiderBasic will run on Windows XP, Windows Vista, Windows 7 and Windows 8 (in both 32-bit and 64-bit edition), Linux (kernel 2.2 or above) and MacOS X (10.6 or above).

If there are any problems, please contact us.

Chapter 4

Installation

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

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 "SpiderBasic\Compilers\" directory to the PATH environment variable to make the compiler accessible from any directory.

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

Chapter 5

Order

SpiderBasic is a low-cost programming language. In buying SpiderBasic you will ensure that development will go further and faster. The updates are free until the next major version. For example, when you buy SpiderBasic 1.00, all further 1.xx updates will be free, and the 2.00 version and above will need a new registration fee. For ease of ordering, you can safely use our secure online method. Thanks a lot for your support!

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

- maximum number of source lines: about 800

Full version of SpiderBasic:

Check <http://www.spiderbasic.com> for more information about pricing.

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:

Andre Beer
Siedlung 6
09548 Deutschneudorf
Germany
e-mail: andre@spiderbasic.com

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

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

Delivering of the full version

The full version will be provided via your personal download account, which you will get on www.spiderbasic.com after successful registration. If you order from Andre, 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 contact us, to any of the following addresses:

Frederic 'AlphaSND' Laboureur

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

s-mail :

Frederic Laboureur
10, rue de Lausanne
67640 Fegersheim
France

e-mail : fred@spiderbasic.com

Andre Beer

Andre is responsible for the complete German translation of the SpiderBasic manual and website. SpiderBasic 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@spiderbasic.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. SpiderBasic wouldn't be the same without him !
- **Gaetan Dupont-Panon:** For the wonderful new visual designer, which really rocks on Windows, Linux and OS X !

Miscellaneous

- **Andre Beer:** To spend time for improving the guides (including beginners guide) and do the complete translation into German. Big thanks!

Part II

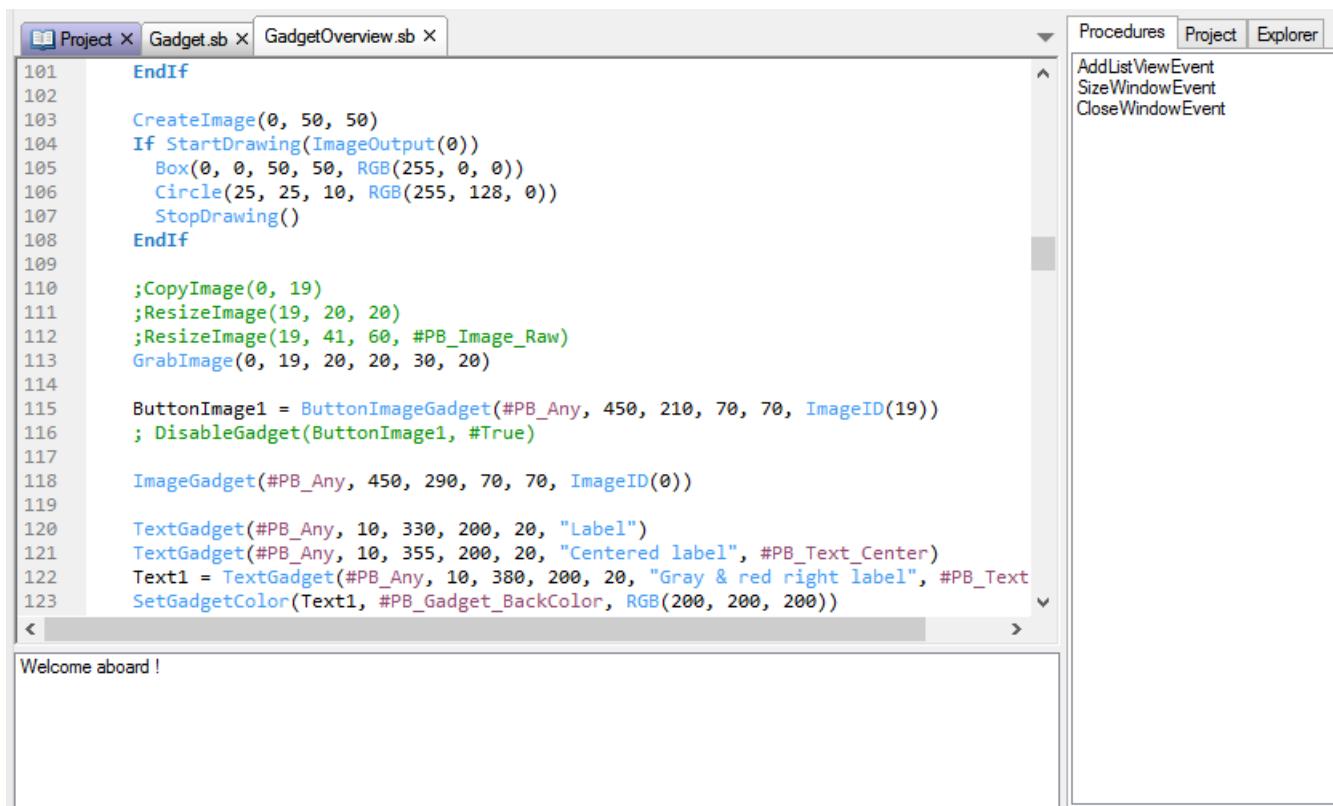
The SpiderBasic Editor

Chapter 8

Getting Started

The SpiderBasic IDE allows you to create and edit your SpiderBasic source codes, as well as run them, debug them and create the final project.

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 then 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

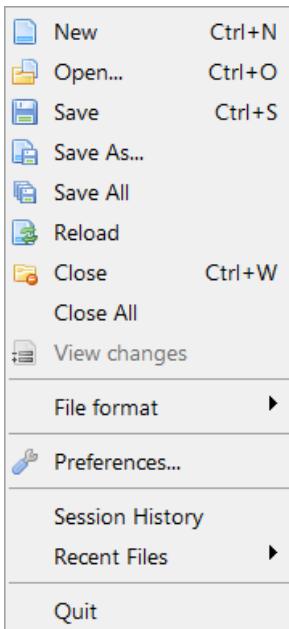
Working with source files

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 SpiderBasic file extensions (sb, sbi and sbf) 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

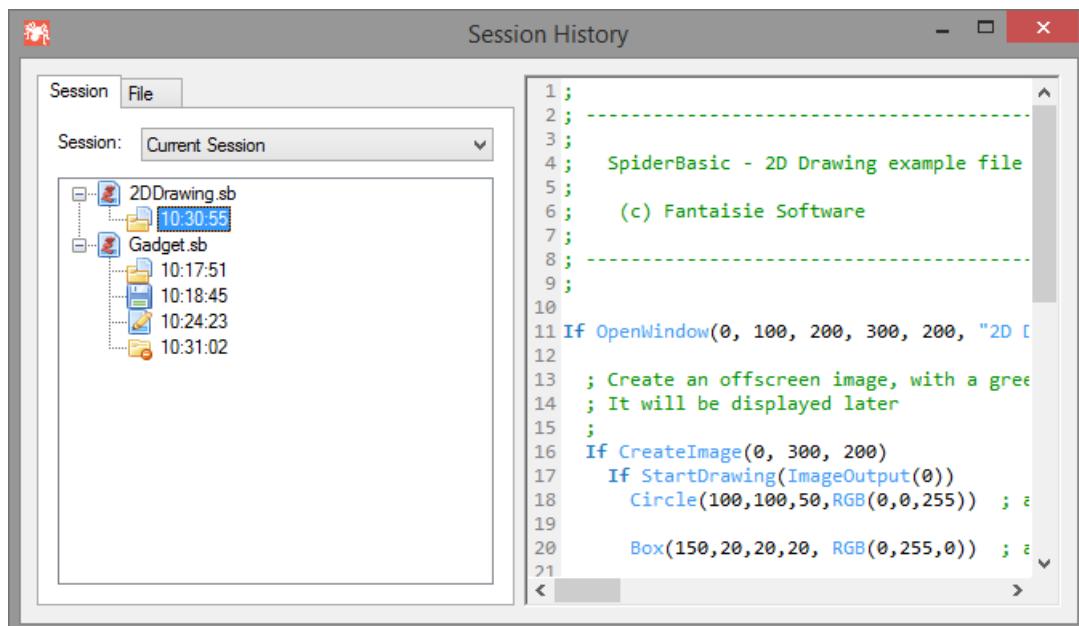
In this submenu you can select the text encoding as well as the newline format which should be used when the currently active source code is saved to disk. The IDE can handle files in Ascii or UTF-8. The newline formats it can handle are Windows (CRLF), Linux/Unix (LF) and MacOSX (CR). The defaults for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

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

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 that is defined in the current source while holding down the Ctrl Key, the cursor automatically jumps to the declaration of this procedure.

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:

```
Select EventGadget()

Case 1 ; Play
    ClearGadgetItems(4)
    DisableGadget(2,0) ; Enable the 'Stop' gadget
    DisableGadget(1,1) ; Disable the 'Play' Gadget

Case 2 ; Stop
    DisableGadget(1,0) ; Enable the 'Play' gadget
    DisableGadget(2,1) ; Disable the 'Stop' Gadget

Case 4
    If EventType() = 2
        SetGadgetText(0, GetGadgetText(4)) ; Get the current item
    EndIf

EndSelect
```

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 SpiderBasic) 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 SpiderBasic 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 or interfaces in your code as long as they are declared in the same source code or project .

Folding options:

```
⊕ Procedure GadgetEvents()  
⊖ Procedure CloseWindowEvent()  
    CloseWindow(EventWindow())  
EndProcedure
```

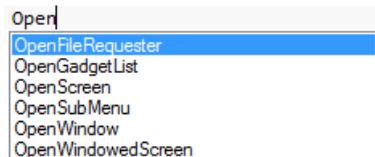
When special folding keywords are encountered ([Procedure](#) / [EndProcedure](#) by default. More can be added), the IDE marks the region between these keywords on the left side next to the line numbers with a [-] at the starting point, followed by a vertical line to the end point.

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:



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

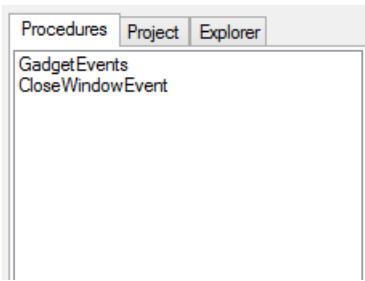
After you have typed the beginning of a command, a list of possible matches to the word start you have just typed will be displayed. A list of options is also displayed when you typed a structured variable or interface followed by a "\".

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
	Cut	Ctrl+X
	Copy	Ctrl+C
	Paste	Ctrl+V
	Insert comments	Ctrl+B
	Remove comments	Ctrl+Shift+B
	Format indentation	Ctrl+I
	Select All	Ctrl+A
	Goto...	Ctrl+G
	Goto matching Keyword	Ctrl+K
	Goto recent Line	Ctrl+L
	Toggle current fold	F4
	Toggle all folds	Ctrl+F4
	Add/Remove Marker	Ctrl+F2
	Jump to Marker	F2
	Clear Markers	
	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 Brower 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.

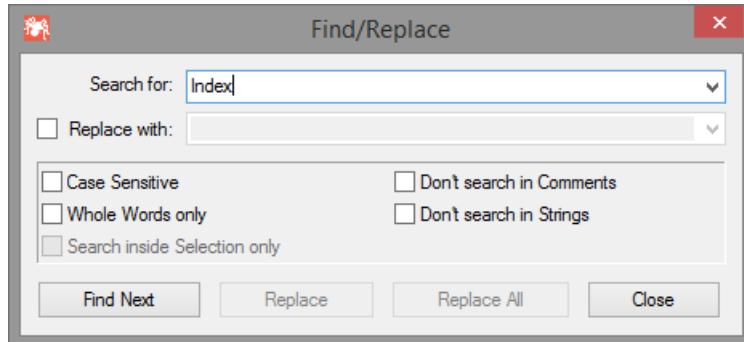
Note: You can also set/remove markers by holding down the Ctrl Key and clicking on the border that holds the markers (not the Line-number part of it).

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.

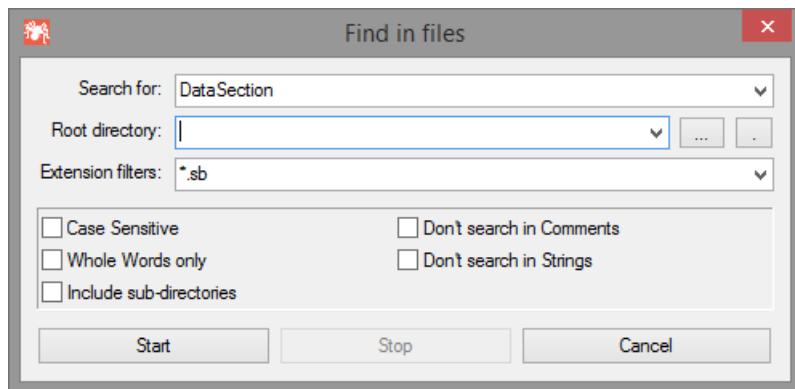
By enabling the "Replace with" checkbox, you go into replace mode. "Find Next" will still only search, but with each click on the "Replace" button, the next match of the search word will be replaced by whatever is inside the "Replace with" box.

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 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...	Ctrl+Shift+N
 Open Project...	Ctrl+Shift+O
Recent Projects	▶
 Close Project	Ctrl+Shift+W
 Project Options...	
 Add File to Project	Ctrl+Shift+A
 Remove File from Project	Ctrl+Shift+R
 Open Project Folder	

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

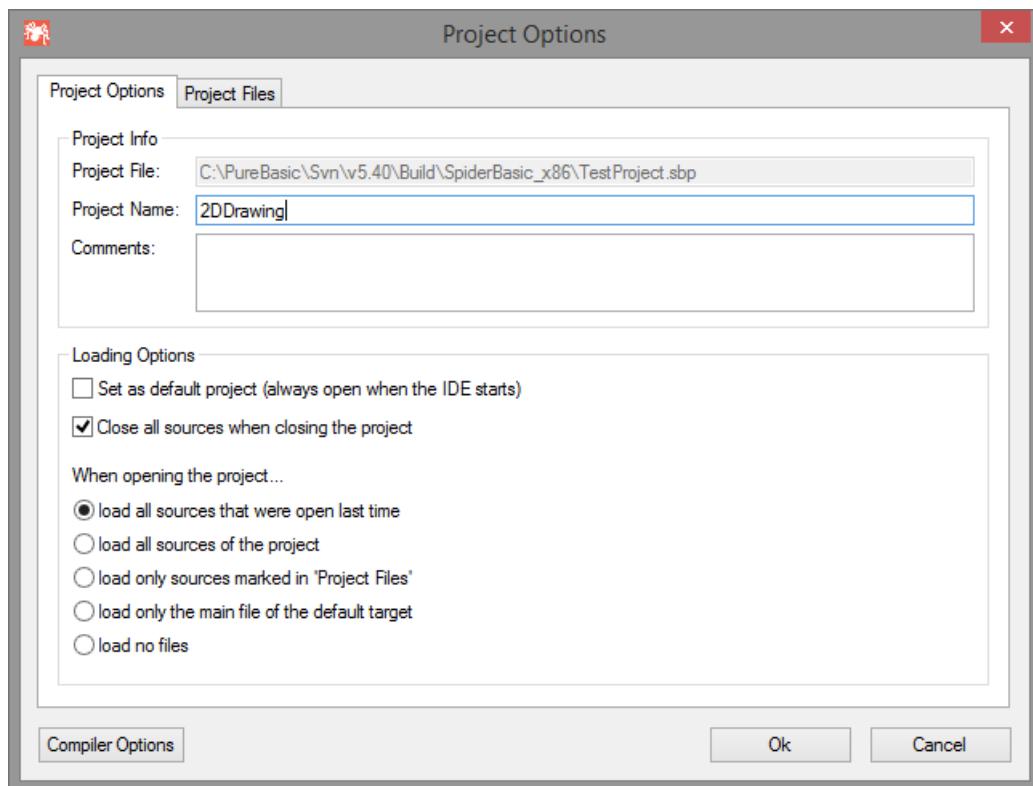
Removes the currently active source from the current project.

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 me 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.

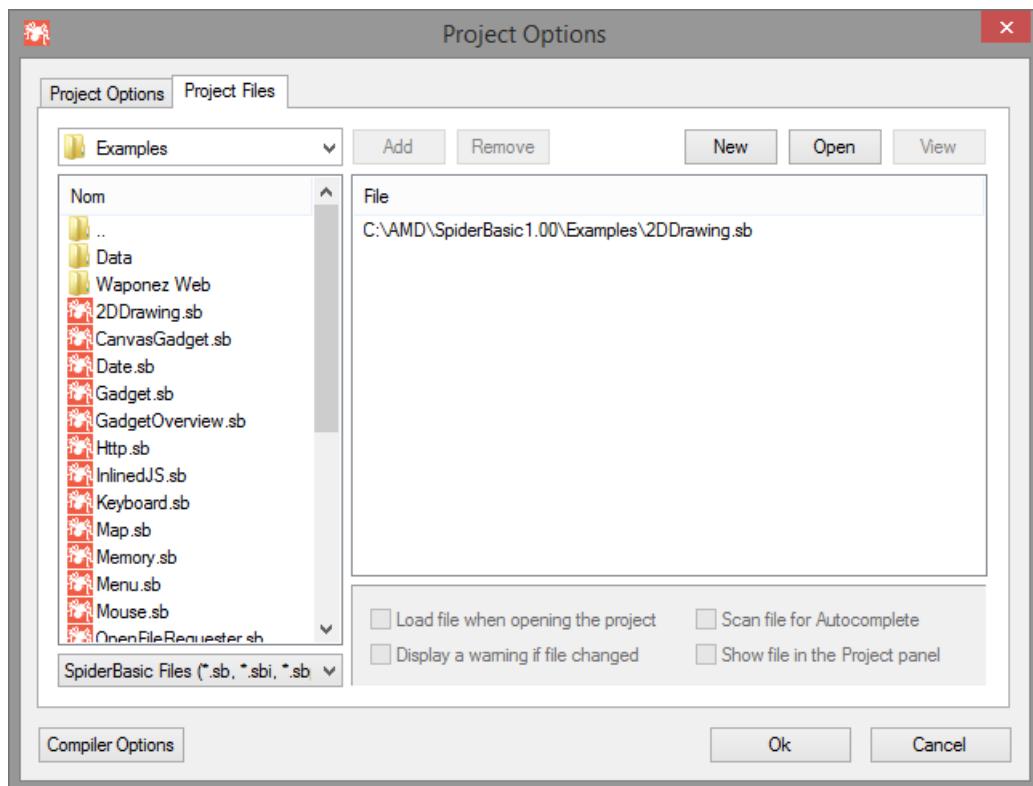
load only the main file of the default target

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.

The screenshot shows the Project panel with the following sections:

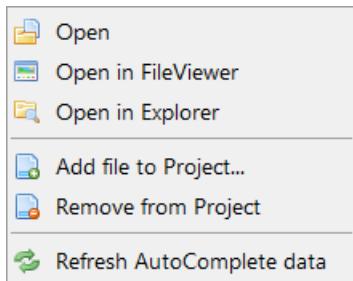
- Project Info:** Displays the project name (New Project), project file (C:\PureBasic\Svn\v5.40\Build\SpiderBasic_x86\TestProject.sbp), and last open date (03/24/2015 - 22:00 by Fred on SUPERFROG). It also shows the editor used (SpiderBasic 1.00 (Windows -x86)). Buttons for Project Options and Compiler Options are available.
- Project Files:** A table listing the project files. One file is listed: C:\AMD\SpiderBasic1.00\Examples\2DDrawing.sb. The columns are: Filename, Load, Warn, Scan, Panel, Size, and Last Modified. The file has Load: Yes, Warn: Yes, Scan: Yes, Panel: Yes, Size: 1.14 Kb, and Last Modified: 12/05/2014.
- Project Targets:** A table listing the project targets. One target is listed: Default Target. The columns are: Target, Debug, Unicode, Thread, Asm, OnError, Compile, Build, Format, and Input. The target has Debug: Yes, Unicode: Yes, and Input: Windows 2DDr.

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

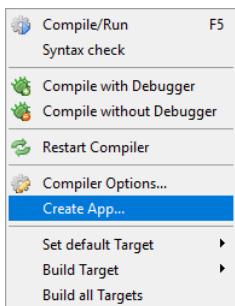
Compiling your programs

Compiling is easy. Just select "Compile/Run" (F5 by default) and your program will be compiled and launched in the default web browser.

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 SpiderBasic 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 enables 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 SpiderBasic 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.

Export

Launch the export process. The export settings can be changed in the "Compiler Options/Export" panel.

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.

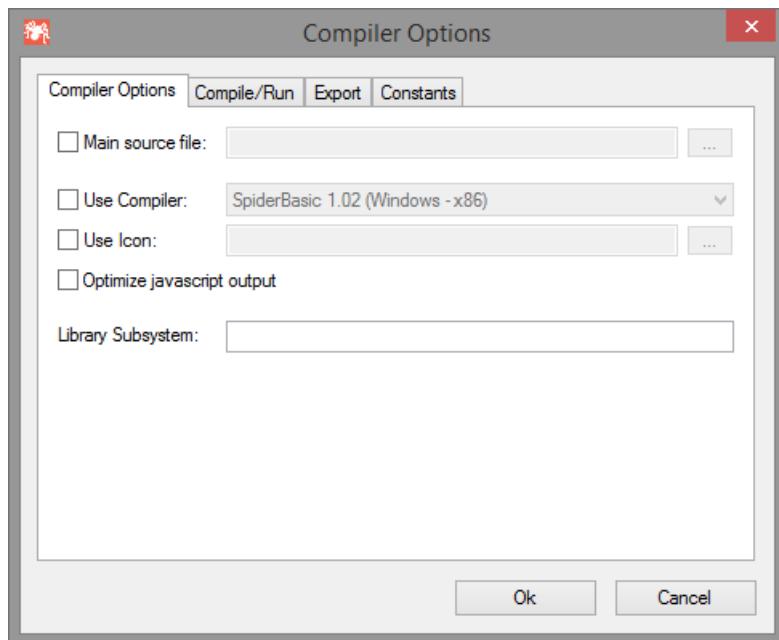
Build Target

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

SpiderBasic version. This makes it easy to compile different versions of the same program (x86 and x64 or PowerPC) 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 on Windows and Linux. The same applies to the x86 and PowerPC compilers for Mac OSX. This is especially useful as this way the fast x86 IDE and debugger can be used on a Mac with an Intel processor while still compiling programs for the PowerPC processor through the slower emulation. 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

When enabled, allow to set the "favicon" file for the web application. The icon has to be in the PNG image format. This icon is usually displayed in the browser tab, near the page title.

Optimize JavaScript output

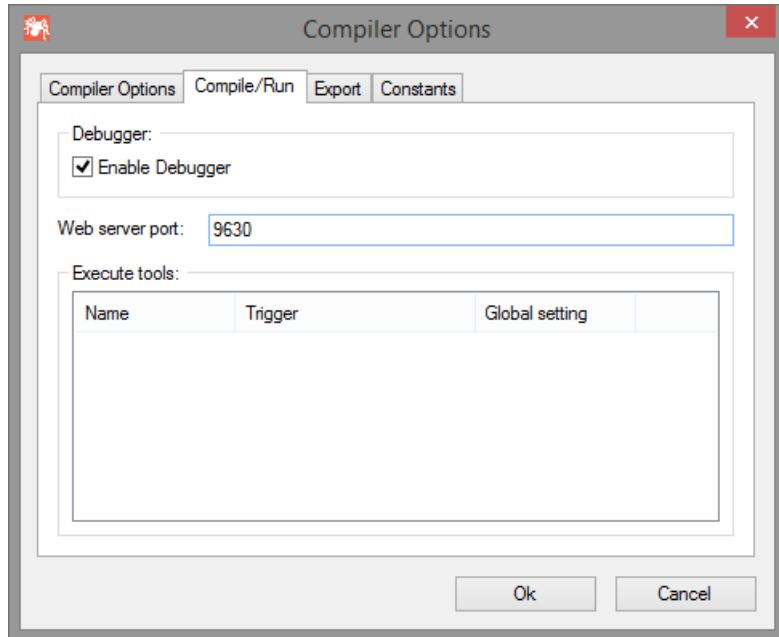
When enabled, uses the Google JavaScript closure compiler to optimize the generated JavaScript code to reduce its size. A recent Java JRE needs to be installed to have this option working. The most recent JRE version can be found here: <https://java.com/download>.

Library Subsystem

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

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.

Web server address

This allows to set a specific web server address for this file or project. The value has to be specified as 'address:port' (example: 'localhost:8080' or 'mydomain:80'). If set to an empty value, localhost will be used with a random dynamic port, starting from the value set in Preferences/Compiler.

Execute tools

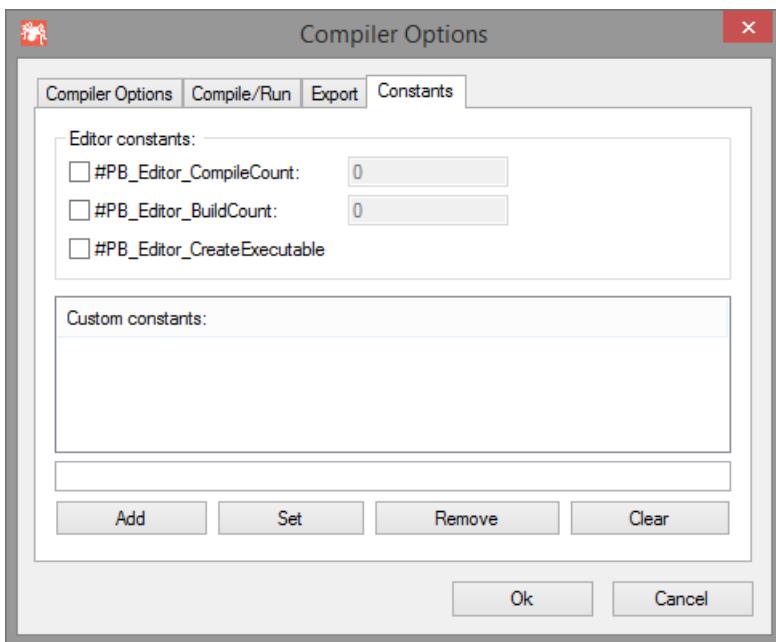
Here external tools can be enabled on a per-source basis. The "Global settings" column shows if the tool is enabled or disabled in the tools configuration . A tool will only be executed for the source if it is both

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_CompilerCount

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 .

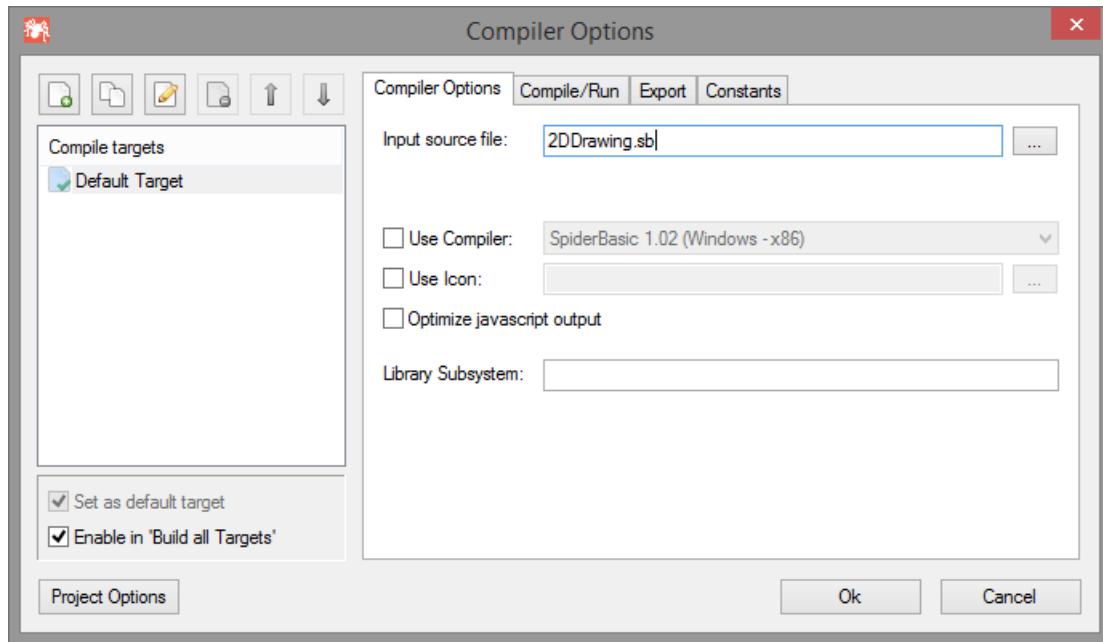
Inside the definition of these constants, environment variables can be used by specifying them in a "bash" like style with a "\$" in front. The environment variable will be replaced in the constant definition before compiling the source. This allows to pass certain options of the system that the code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

Compiler options for projects



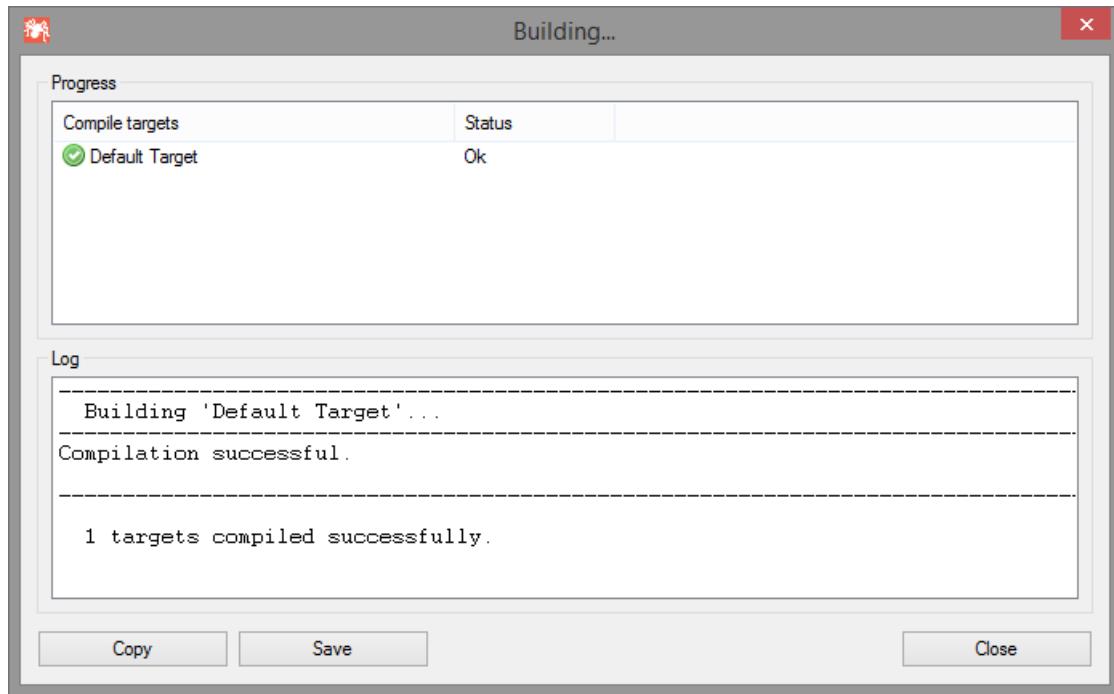
The compiler options for projects allow the definition of multiple compile targets. Each target is basically a set of compiler options with a designated source file and output executable. The left side of the compiler options window is extended with the list of the defined compile targets. The toolbar 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 is the "Input source file" on the first tab. This field has 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

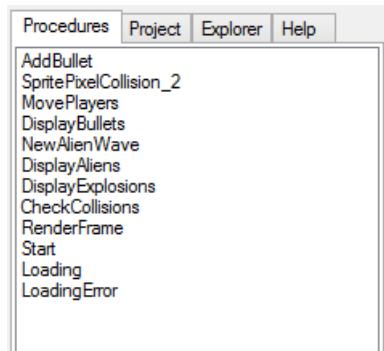
Using the built-in Tools

The SpiderBasic 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.

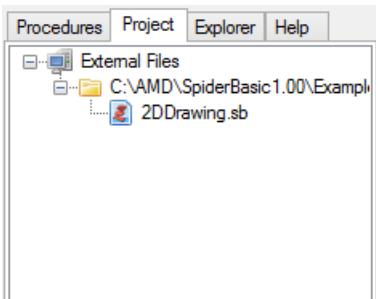
Macros will be marked in the list by a "+" sign before the name.

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.

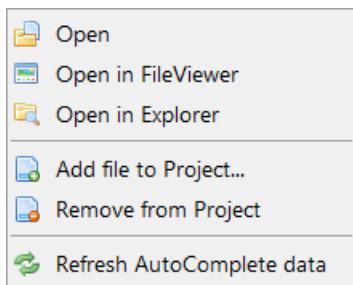
The description will be shown in the Procedure list, and clicking on it will jump to the line of this mark. Such a comment mark can be distinguished from a Procedure by the ">" that is displayed before it in the procedure list.

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.

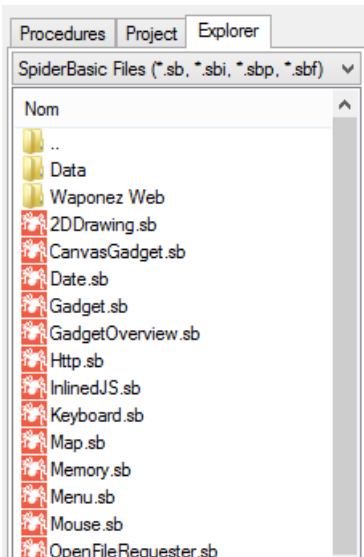
Open in Explorer - Open the file in the operating systems file manager.

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

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

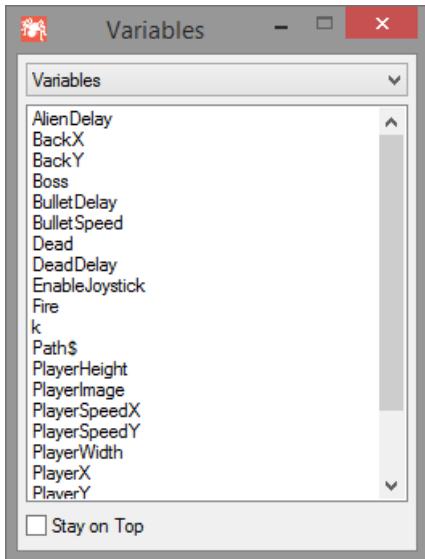
Refresh AutoComplete data - Rescan the file for AutoComplete items.

Explorer



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

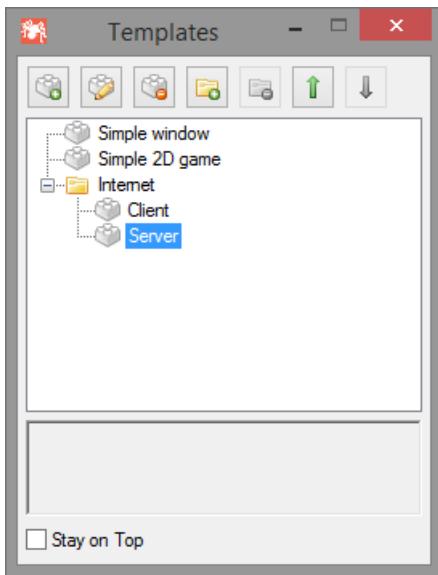
Variable Viewer



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 .

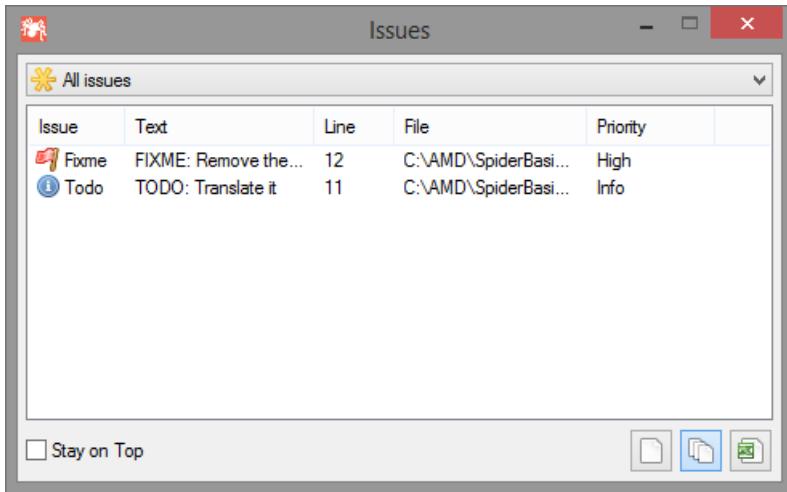
Note: The displaying of variables is somewhat limited for now. It can only detect variables explicitly declared with Define , Global , Shared , Protected or Static .

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.

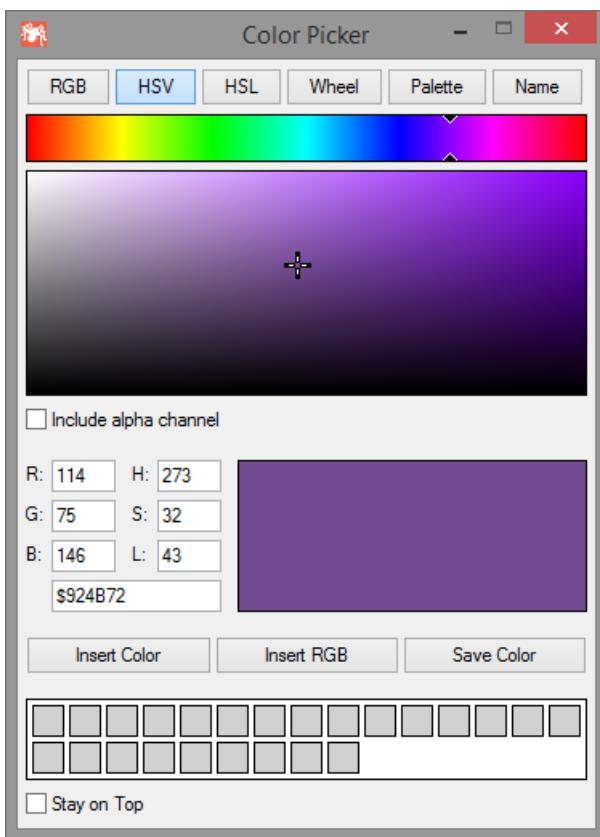
Issue Browser



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.

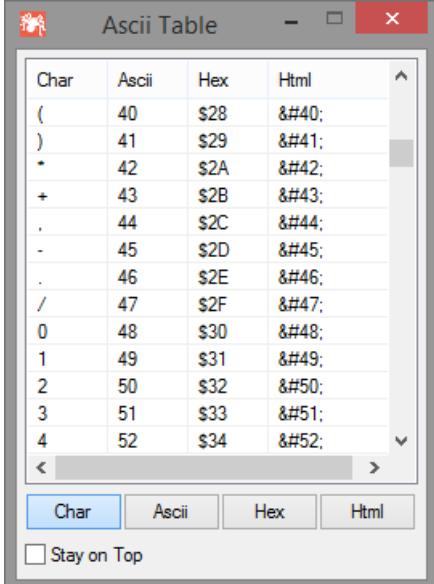
Palette: Select a color from a predefined palette.

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.

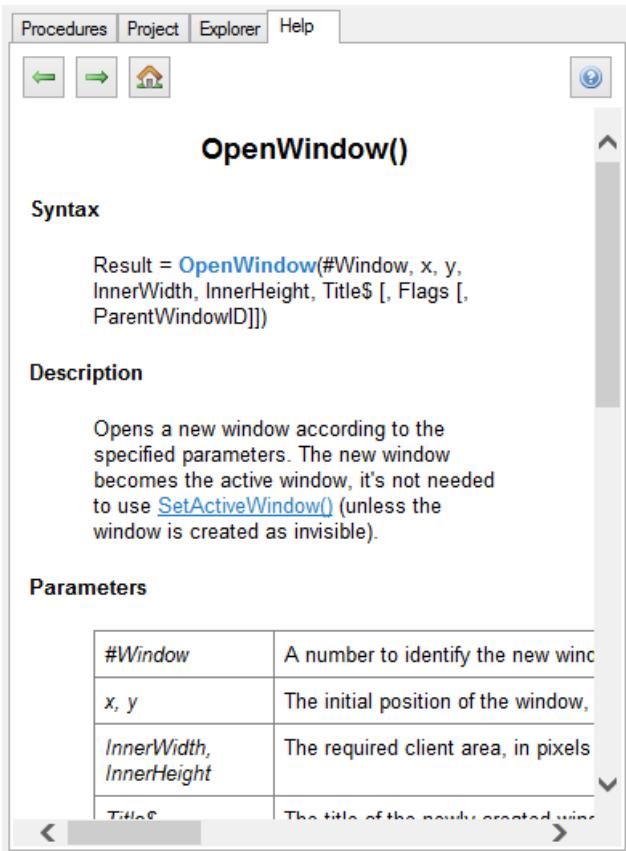
Ascii Table



Char	Ascii	Hex	Html
(40	\$28	(
)	41	\$29)
*	42	\$2A	*
+	43	\$2B	+
.	44	\$2C	,
-	45	\$2D	-
.	46	\$2E	.
/	47	\$2F	/
0	48	\$30	0
1	49	\$31	1
2	50	\$32	2
3	51	\$33	3
4	52	\$34	4

The Ascii table tool displays a table showing all the Ascii 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.

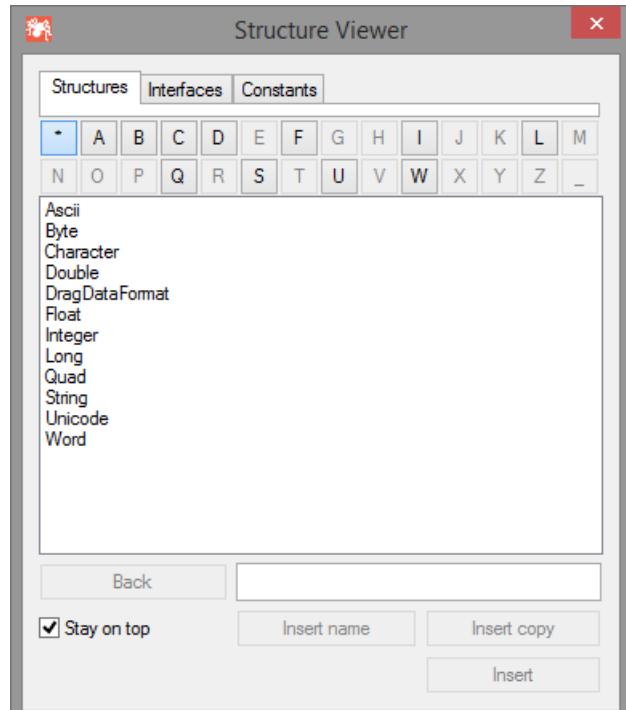
Help Tool



The Help Tool is an alternative viewer for the reference guide . It can be used to view the SpiderBasic 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 SpiderBasic. 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.

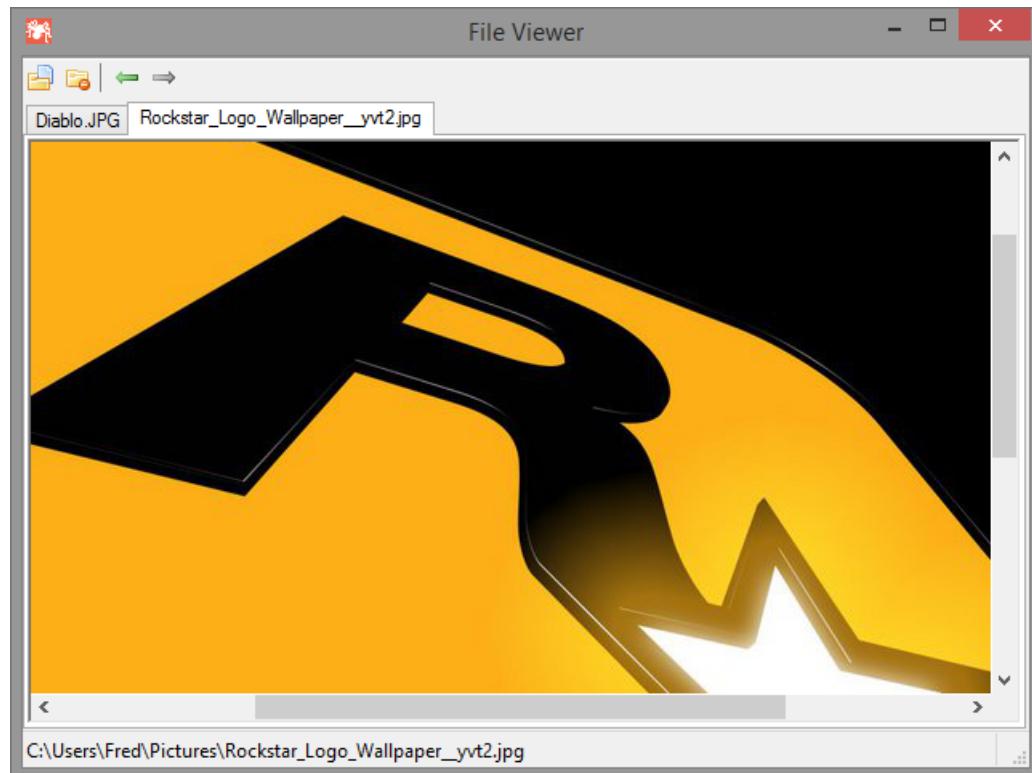
The "Back" button navigates back through the viewed entries.

"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.

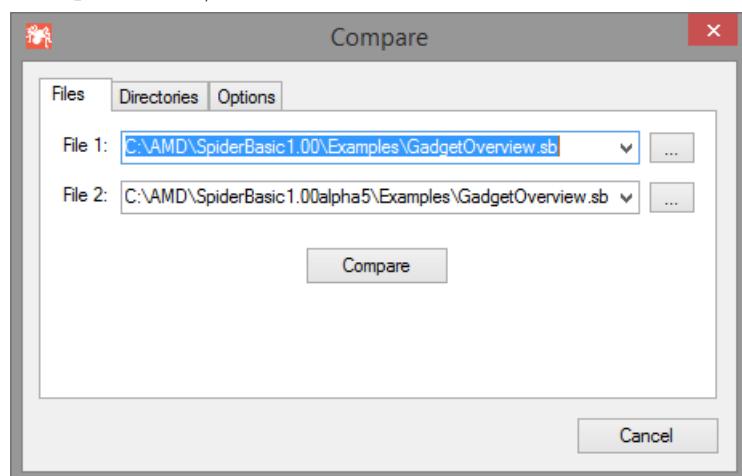
File Viewer



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.

Compare Files

```

8 ; 
9 ; 
1 
1 ◇Global ListIcon1, ListView1, Button1, Tr 
1 
1 ; Debug DesktopName(0) 
1 
1 Procedure AddListViewEvent() 
1   AddGadgetItem(ListView1, 5, "Inserted" 
1 -   ; Debug GetGadgetItemText(ListIcon1, G 
1   ;SetGadgetState(tree1, 1) 
1   ;Debug GetGadgetState(Tree1) 
1   ;Debug GetGadgetItemState(Tree1, 6) 
2   ;RemoveGadgetItem(ListView1, 2) 
2   ;ClearGadgetItems(ListView1) 
2   ;SetGadgetItemText(ListView1, 4, "Chan 
2 EndProcedure 
2 
2 
2 Procedure SizeWindowEvent() 
2   ResizeGadget(0, 10, 10, WindowWidth(0) 
2   ;ResizeGadget(Button1, 10, 10, WindowW 
2 EndProcedure 
2 
2 
2 Procedure SizeWindowEvent() 
2   ResizeGadget(0, 10, 10, WindowWidth(0) 
2   ;ResizeGadget(Button1, 10, 10, WindowW 
2 EndProcedure 
2 
2 
2 
```

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.

Compare Files

Filename	Status	Date in (1)	Date in (2)
2DDrawing.sb	Modified	12/05/2014 12:38	07/31/2014 11:11
CanvasGadget.sb	Modified	12/07/2014 14:00	02/19/2014 18:44
Date.sb	Modified	12/03/2014 17:49	01/19/2014 13:40
Gadget.sb	Modified	03/25/2015 10:18	02/02/2014 23:04
GadgetOverview.sb	Unchanged	05/02/2014 13:03	05/02/2014 13:03
Http.sb	Only in (1)	11/25/2014 14:43	
HttpRequest.sb	Only in (2)		02/09/2014 18:56
InlinedJS.sb	Only in (1)	01/29/2015 11:29	
Keyboard.sb	Only in (1)	11/25/2014 16:34	
Map.sb	Only in (1)	12/07/2014 14:07	
Memory.sb	Only in (1)	12/03/2014 17:48	
Menu.sb	Modified	12/03/2014 17:48	02/13/2014 15:12
Mouse.sb	Only in (1)	12/03/2014 17:48	
OpenFileRequester.sb	Modified	12/05/2014 14:18	07/31/2014 11:21
PopupMenu.sb	Only in (1)	12/03/2014 17:48	
RegularExpression.sb	Only in (1)	12/03/2014 17:48	
Runtime.sb	Only in (1)	12/03/2014 17:48	
Screen.sb	Modified	12/03/2014 16:50	02/18/2014 12:33
Sort.sb	Only in (1)	12/03/2014 16:51	
Sound.sb	Modified	12/03/2014 16:59	07/30/2014 15:53
Sprite.sb	Only in (1)	12/03/2014 17:47	

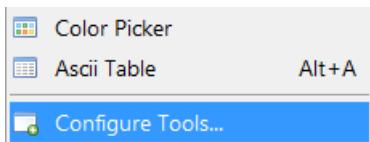
When comparing directories, the content of both directories is examined (with the option to filter the search by file extension and include subdirectories) and the files are marked in a similar way: Files in red do not exist in the second directory, files in green are new in the second directory and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

Chapter 14

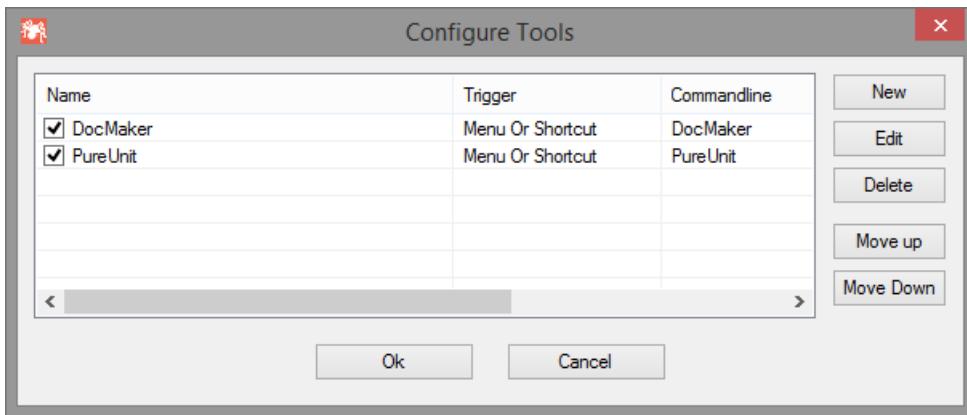
Using external tools

The SpiderBasic 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 SpiderBasic 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.



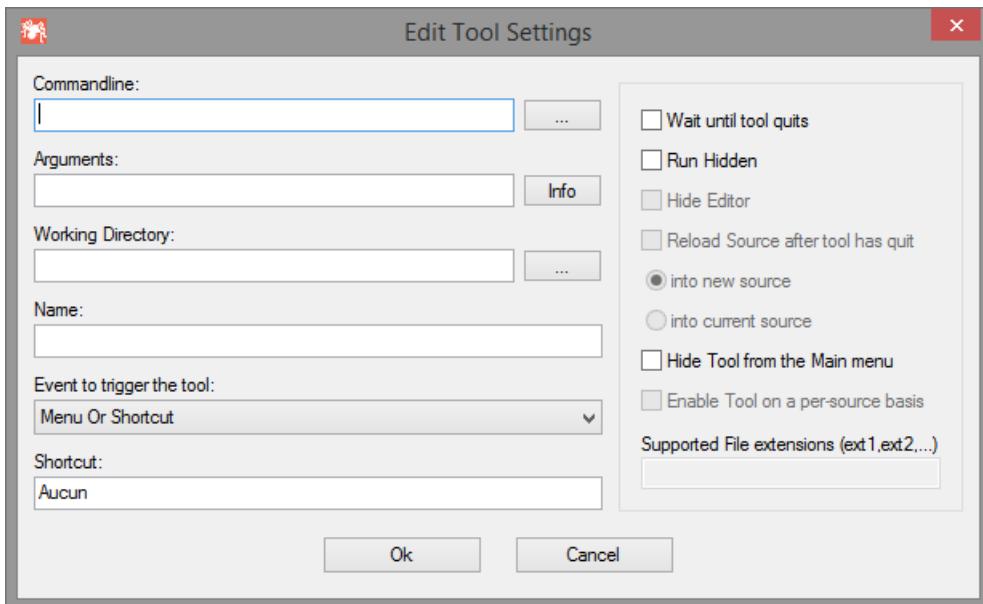
With the "Config tools" command in the Tools menu, you can configure such external tools. The list you will see displays all the configured tools in the order they appear in the Tools menu (if not hidden). You can add and remove tools here, or change the order by clicking "Move Up"/"Move Down" after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

%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 spiderbasic 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.

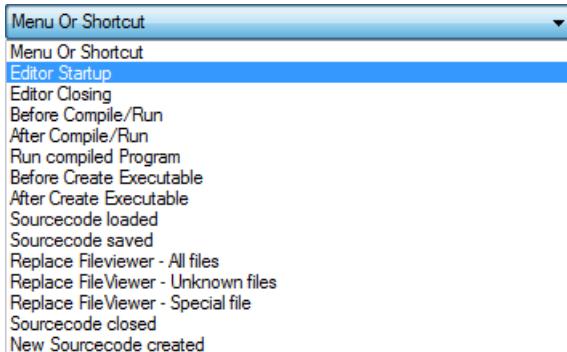
Name

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.

Other options on the right side

Wait until tool quits

The IDE will be locked for no input and cease all its actions until your 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.

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 Compiler Options	- These variables provide the settings from the window for the current source. They are set to "1" if the option is enabled, and "0" if not.
PB_TOOL_InlineASM	
PB_TOOL_Uncode	
PB_TOOL_Thread	
PB_TOOL_XPSkin	
PB_TOOL_OnError	
PB_TOOL_SubSystem compiler options	- content of the "Subsystem" field in the
PB_TOOL_Executable command-line	- same as the %COMPILEFILE token for the
PB_TOOL_Cursor	- same as the %CURSOR token for the command-line
PB_TOOL_Selection command-line	- same as the %SELECTION token for the
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 the current source	- OS handle to the Scintilla editing component of

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.

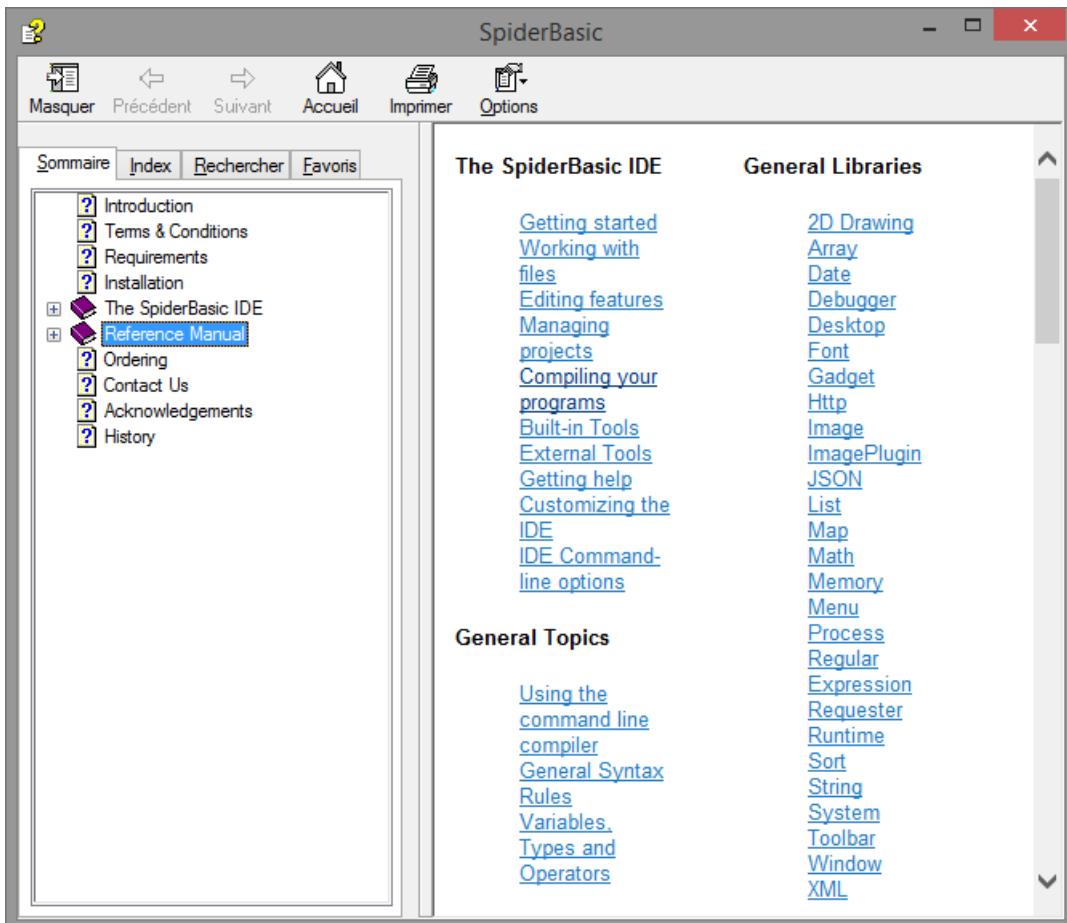
This enables your tool to read the compiler settings for this file, and take them into account for the actions your carries out.

Chapter 15

Getting Help

The SpiderBasic IDE provides ways to access the SpiderBasic 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 SpiderBasic 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 .

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 SpiderBasic 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 SpiderBasic. Now if you click the file in the "external help" menu, it will be opened in that external tool.

Chapter 16

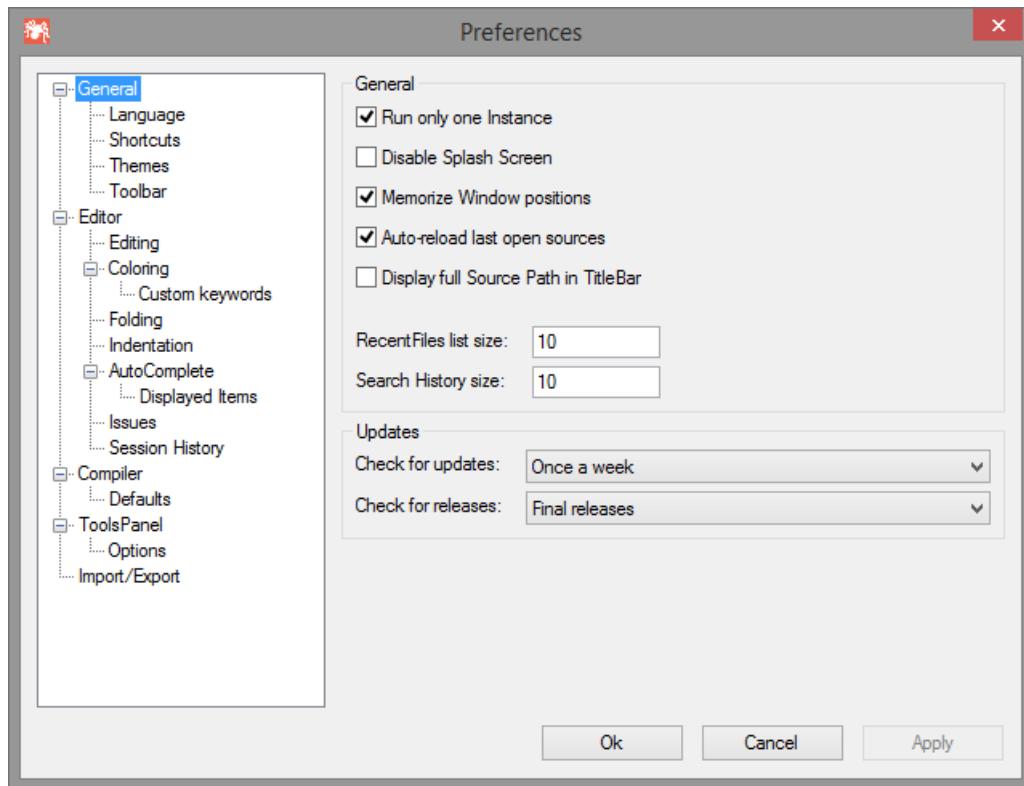
Customizing the IDE

The SpiderBasic 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.

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"

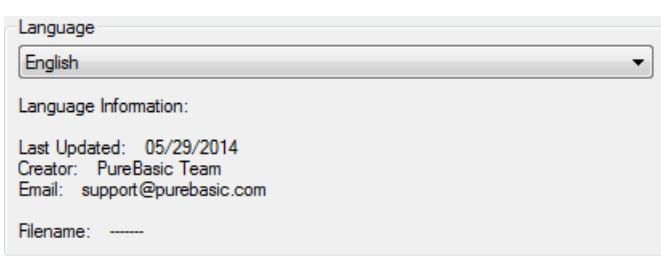
Check for updates

Specifies how often the IDE should check on the spiderbasic.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

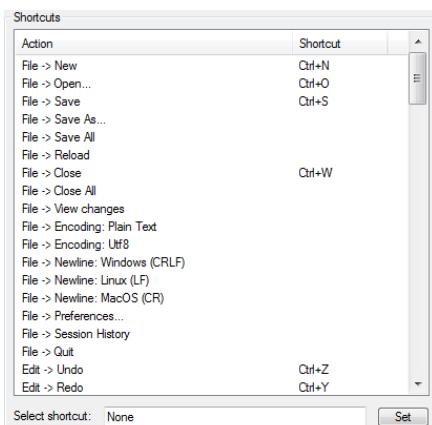
Specifies which kind of releases should cause a notification if they are available.

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

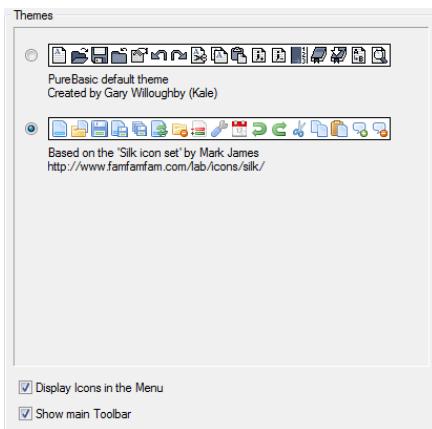


Here you can fully customize all the shortcut commands of the IDE. Select an entry from the list, select the shortcut field, enter the new key combination and click "Set" to change the entry.

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 SpiderBasic installation directory to be recognized by the IDE. The "SilkTheme.zip" file can be used as an example to create a new theme.

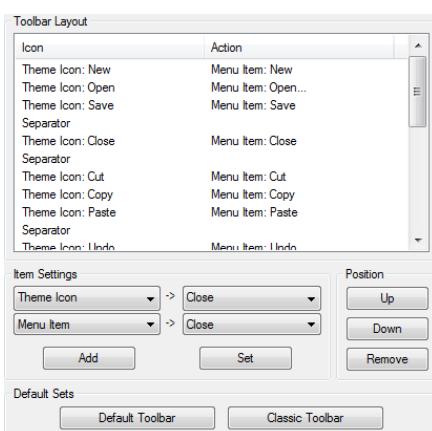
Display Icons in the Menu

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).

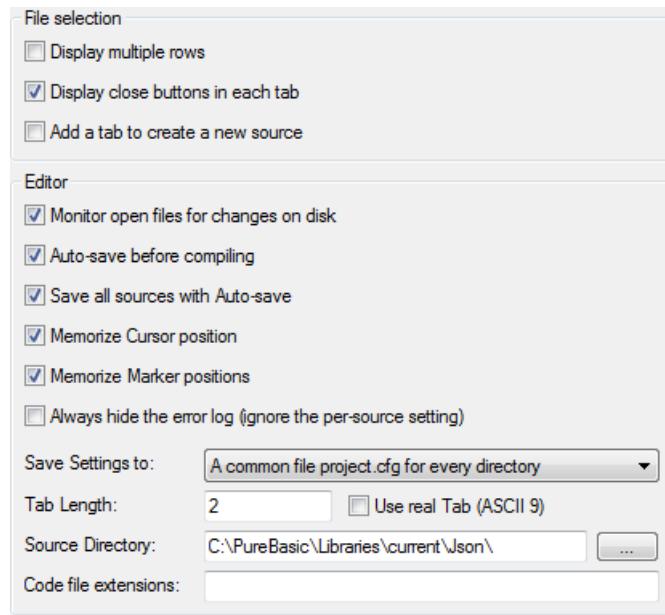
If you do not select a separator or space, you can specify an action to carry out when the button is pressed:

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



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>.sb.cfg

Creates a .sb.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

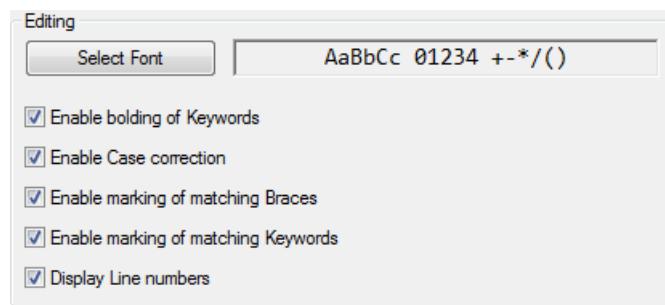
Specifies the default directory used in the Open and Save dialogs if no other files are currently open (if another file is open, its path will be used as default).

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

Code file extensions

The IDE detects code files by their extension (sb, sbi or sbf 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. "sbx, 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 SpiderBasic keywords, SpiderBasic 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.

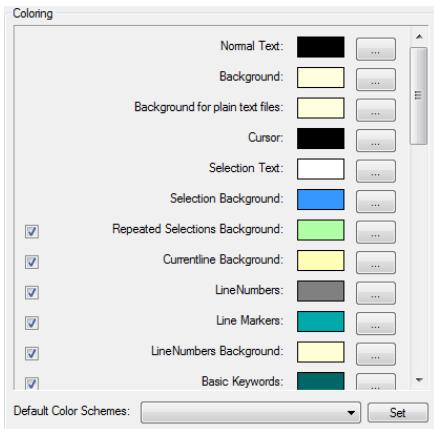
Enable marking of matching Keywords

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.

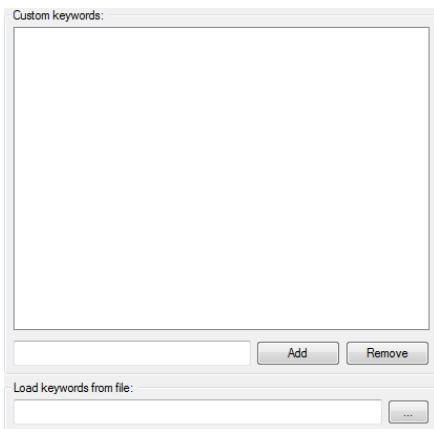
Editor - Coloring



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.

Editor - Coloring - Custom Keywords

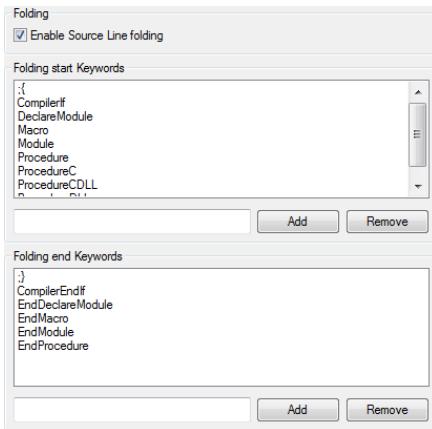


In this section, a list of custom keywords can be defined. These keywords can have a special color assigned to them in the coloring options and the IDE will apply case-correction to them if this feature is enabled. This allows for applying a special color to special keywords by preprocessor tools or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

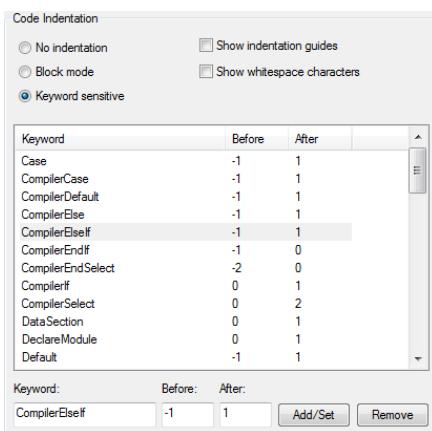


Here you can set the keywords in the source code that start/end a foldable section of code. You can add any number of words that will mark such a sections. You can also choose to completely disable the folding feature.

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

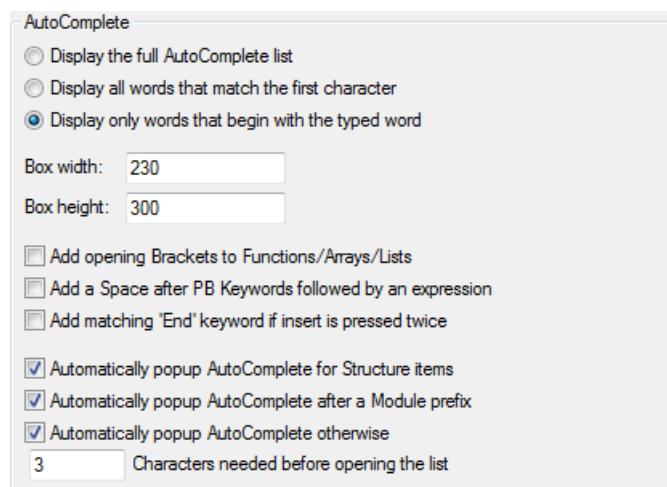
Causes vertical lines to be shown to visualize the indentation on each line. This makes it easier to see which source lines are on the same level of indentation.

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 Tab/Enter is pressed twice

If you press Tab or Enter twice, it will insert the corresponding end keyword (for example "EndSelect" to "Select" or "EndIf" to "If") to the keyword you have just inserted. 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).

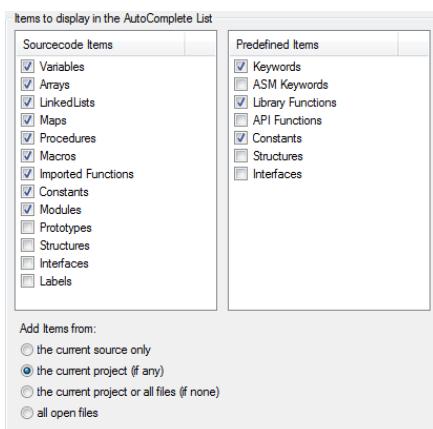
Automatically popup AutoComplete outside of Structures

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 SpiderBasic, such as the SpiderBasic 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.

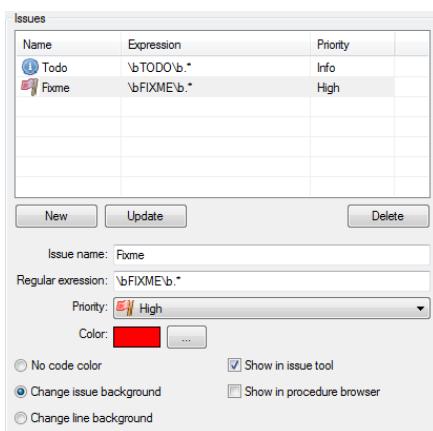
Add Items from: the current project or all files (if none)

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.

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.

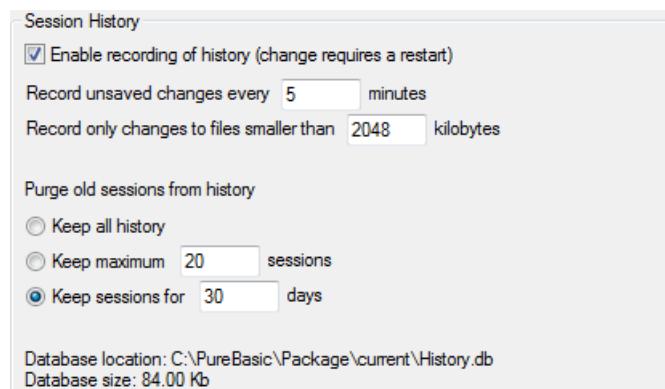
Show in issue tool

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

The screenshot shows a configuration interface for compilers. At the top, there is a field for the 'Default web server port' set to '9083'. Below this, the 'Default Compiler' section lists 'SpiderBasic 1.00 (Windows - x86)' with the path 'C:\PureBasic\Svn\w5.40\Build\SpiderBasic_x86\Compilers\sbcompiler.exe'. The 'Additional Compilers' section contains a table with columns 'Version' and 'Path', which is currently empty. Below the table are buttons for 'Add', 'Remove', 'Clear', and an ellipsis (...).

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 SpiderBasic 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.

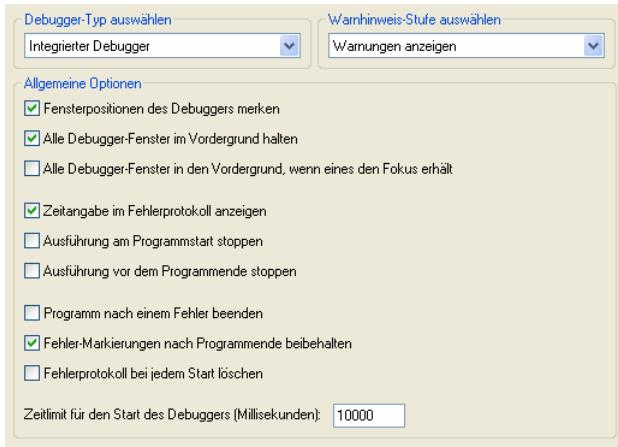
Compiler - Defaults

The screenshot shows the 'Default Settings for new Files' section. It includes a list of checkboxes for various options: 'Show Error Log' (checked), 'Enable Debugger' (checked), 'Enable Purifier' (unchecked), 'Enable inline ASM syntax coloring' (unchecked), 'Create unicode executable' (checked), 'Enable modern theme support (for Windows XP and above)' (checked), 'Request Administrator mode for Windows Vista and above' (unchecked), 'Request User mode for Windows Vista and above (no virtualisation)' (unchecked), and 'Create temporary executable in the source directory' (unchecked). Below this, there are dropdown menus for 'Current directory', 'Library Subsystem', 'Executable format' (set to 'Windows'), 'CPU Optimisation' (set to 'All CPU'), 'Sourcefile Text encoding' (set to 'UTF-8'), and 'Sourcefile Newline format' (set to 'Windows (CRLF)').

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



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.

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.



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

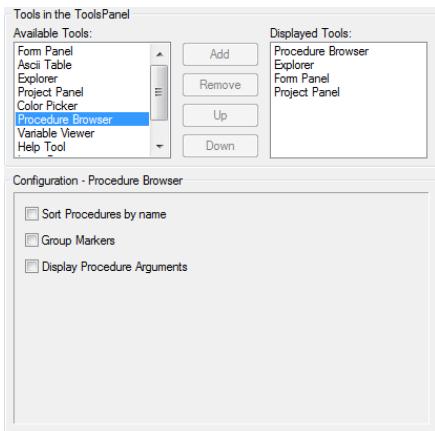
If the memory area is displayed in array view, this option selects whether it should be multi-column (with 16 bytes displayed in each column) or with only one column.

Debugger - Default Windows



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

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

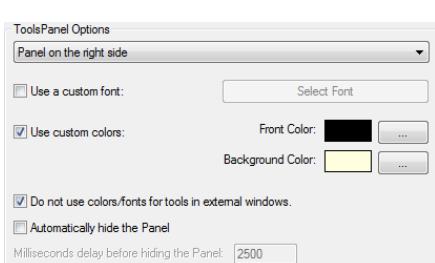
The "Display Elements from all open sources" option determines whether the list should only include items from this code, or from all open codes.

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.

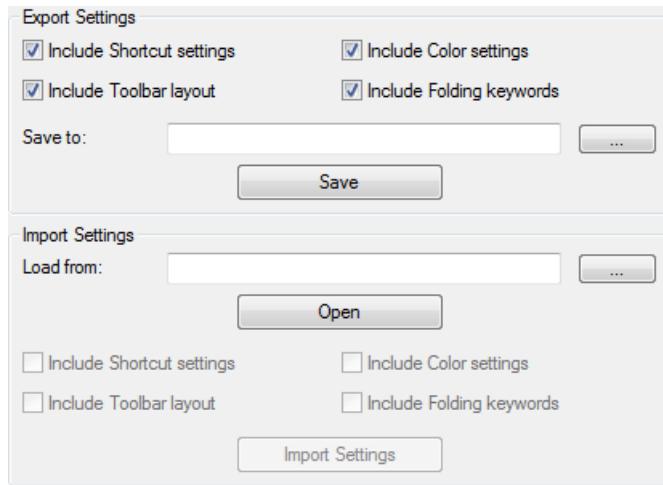
Automatically hide the Panel

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 SpiderBasic 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 17

Command-line options for the IDE

The SpiderBasic 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 SpiderBasic 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:

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

Options for launching the IDE:

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

Options for building projects:

/BUILD <file>	specifies the project file to build
/TARGET <target>	specifies the target to build (the default is to build all targets)
/QUIET	hides all build messages except errors
/READONLY	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%\SpiderBasic\ directory on the system.

The /NOEXT command is useful when you have several different SpiderBasic versions at once (for testing of beta versions for example), but want the .sb 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 SpiderBasic to different computers (or run it from USB sticks for example).

Example:

```
1 SpiderBasic.exe Example.sb /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 “*.sb” you can load a whole directory).

Part III

Language Reference

Chapter 18

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, SpiderBasic does not currently have a power-of operator! Use the SpiderBasic 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

$$\begin{array}{rcl} 1 * 1000 &=& 1000 \\ + \quad 2 * \quad 100 &=& 200 \\ + \quad 3 * \quad 10 &=& 30 \\ + \quad 4 * \quad 1 &=& 4 \\ \hline \text{Total} &=& 1234 \end{array}$$

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, \wedge 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):

$$\begin{aligned} a * B^4 &= a * B * B * B * B \\ b * B^3 &= b * B * B * B \\ c * B^2 &= c * B * B \\ d * B^1 &= d * B \\ e * B^0 &= e \end{aligned}$$

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 SpiderBasic'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
Weight (index)	2^7	2^6	2^5	2^4	2^3	2^2	2^1
Weight (actual value)	128	64	32	16	8	4	2

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

$$\begin{aligned} & 0 * 128 \\ + & 0 * 64 \\ + & 1 * 32 \\ + & 1 * 16 \\ + & 0 * 8 \\ + & 0 * 4 \\ + & 1 * 2 \\ + & 1 * 1 \\ = & \quad 51 \end{aligned}$$

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, SpiderBasic 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
Weight (index)	-2^7	2^6	2^5	2^4	2^3	2^2	2^1
Weight (actual value)	-128	64	32	16	8	4	2

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 SpiderBasic 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 SpiderBasic 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:

$$\begin{aligned}
 B * 4096 &= 11 * 4096 \\
 + E * 256 &= 14 * 256 \\
 + E * 16 &= 14 * 16 \\
 + F * 1 &= 15 * 1 \\
 = & 48879
 \end{aligned}$$

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

```

666 / 16 = 41 r 10      ^
41 / 16 = 2 r 9      /|\     Read digits in this direction,
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 SpiderBasic, you put a dollar sign in front of the number, for example \$BEEF.

Chapter 19

Break : Continue

Syntax

`Break`

Description

`Break` provides the ability to exit during any iteration, for the following loops: Repeat , For , ForEach and While .

Example

```
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
```

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
4        iteration
5    EndIf
6    Debug k
7  Next
```

Chapter 20

Using the command line compiler

Introduction

The command line compiler is located in the subdirectory 'Compilers\' from the SpiderBasic folder. The easier way to access this is to add this directory in the windows PATH variable, which will give access to all the commands of this directory at all times. The Linux/OS X switches equivalent are also accepted on Windows to ease script creation across all platforms.

Command switches

/? (-h, -help): display a quick help about the compiler.
/COMMENTED (-c, -commented): create a commented javascript output file.
/DEBUGGER (-d, -debugger): enable the debugger support.
/OUTPUT (-o, -output) "filename": output name for the created web, iOS or Android app.
/JAVASCRIPT (-js, -javascript) "filename.js": name of the created javascript file.
/LIBRARYPATH (-lp, -librarypath) "path": path of the SpiderBasic dependencies.
/COPYLIBRARIES (-cl, -copylibraries) "path": copy the SpiderBasic dependencies to the specified local path.
/RESOURCEDIRECTORY (-rd, -resourcedirectory) "path": the directory where all the app resources are located.
/OPTIMIZEJS (-z, -optimizejs): creates an optimized javascript output, including needed javascript. A recent Java JRE needs to be installed to have this option working. The most recent JRE version can be found here: <https://java.com/download>.
/ICON (-n, -icon) \"filename.png\": icon to display in the browser tab. Has to be in PNG image format.
/APPNAME (-an, -appname): set the app name.
/APPVERSION (-av, -appversion): set the app version.
/APPPERMISSION (-ap, -apppermission) "permission": add a permission to the app (it can be specified several time). Available permissions:
- geolocation: add GPS access and other location API support to the app.
/STARTUPIMAGE (-si, -startupimage) "image.png": set the startup image to use when launching the app (iOS and Android only)
/PACKAGEID (-pi, -packageid) "com.yourcompany.appid": set the package id for the app (iOS and Android only)
/IAPKEY (-ik, -iapkey) "MIIB.....AQAB": set InAppPurchase API key for the Android app
/DEPLOY (-p, -deploy): automatically deploy the app on USB connected device (iOS and Android only)
/FULLSCREEN (-fs, -fullscreen): activate fullscreen mode for the app (iOS and Android only)
/ORIENTATION (-w, -orientation) "orientation": set the app orientation (iOS and Android only):
- "any": the app can be in landscape or portrait mode, depending of the device orientation (default).
- "portrait": the app will always launch in portait mode.
- "landscape": the app will always launch in landscape mode.

/ANDROID: create an Android app (Windows only).
/JDK "jdk path": set the path to full JDK 1.8+ used to build Android app (Windows only).
/RESIDENT (-r, --resident) "filename": create a resident file specified by the filename.
/QUIET (-q, --quiet): Disable all unnecessary text output, very useful when using another editor.
/STANDBY (-sb, --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 SpiderBasic 'SDK' directory.
/IGNORERESIDENT (-ir, --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.
/CONSTANT (-t, --constant) Name=Value: Creates the specified constant with the given expression.
Example: 'sbcompiler test.sb /CONSTANT MyConstant=10'. The constant **#MyConstant** will be created on the fly with a value of 10 before the program gets compiled.
/SUBSYSTEM (-s, --subsystem) "Name": Uses the specific subsystem to replace a set of internal functions. For more information, see subsystems .
/CHECK (-k, --check) : Check the syntax only, doesn't create or launch the executable.
/PREPROCESS (-pp, --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 SpiderBasic 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 SpiderBasic source file to create the final output.
/LANGUAGE (-g, --language) "Language": uses the specified language for the compiler.
/VERSION (-v, --version): Displays the compiler version.

Example

```
CLI> sbcompiler sourcecode.sb /OUTPUT "test.html"
```

The compiler will compile the source code to test.html.

```
CLI> sbcompiler sourcecode.sb /DEBUGGER /OUTPUT "test.html"
```

The compiler will compile the source code to test.html with debugger support.

Chapter 21

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_AmigaOS
3         ; some Amiga specific code
4     CompilerCase #PB_OS_Linux
5         ; some Linux specific code
6 CompilerEndSelect
```

Syntax

```
CompilerError <string constant>
```

Description

Generates an error, 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.

Example

```
1 CompilerIf #PB_Compiler_OS = #PB_OS_AmigaOS
2     CompilerError "AmigaOS isn't supported, sorry."
3 CompilerElse
4     CompilerError "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

```
EnableJS  
DisableJS
```

Description

Enables or disables the inline javascript. Inside this block, the line are left untouched and put as is in the generated code. See the inline javascript section for more information.

Example

```
1      ;  
2      ;  
3      Test = 10  
4  
5      EnableJS  
6          v_test = 20  
7      DisableJS  
8  
9      Debug Test ; Will be 20
```

Reserved Constants

The SpiderBasic 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 creating Windows executable  
(PureBasic)  
    #PB_OS_Linux   : The compiler is creating Linux executable  
(PureBasic)  
    #PB_OS_AmigaOS : The compiler is creating AmigaOS executable  
(PureBasic)  
    #PB_OS_MacOS   : The compiler is creating OS X executable  
(PureBasic)  
    #PB_OS_Web     : The compiler is generating a JavaScript file  
(SpiderBasic)  
  
#PB_Compiler_Processor : Determines the processor type for which the  
output is created. It can be one of the following:  
    #PB_Processor_x86   : x86 processor architecture (also called  
IA-32 or x86-32) (PureBasic)  
    #PB_Processor_x64   : x86-64 processor architecture (also called  
x64, AMD64 or Intel64) (PureBasic)  
    #PB_Processor_PowerPC : PowerPC processor architecture (PureBasic)  
    #PB_Processor_mc68000 : Motorola 68000 processor architecture  
(PureBasic)  
    #PB_Processor_JavaScript : JavaScript output (SpiderBasic)  
  
#PB_Compiler_Date      : Current date, at the compile time, in the  
SpiderBasic date  
format.  
#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 SpiderBasic 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 a final project is created
is created, the debugger is always disabled
(this constant will be 0).
#PB_Compiler_InlineJavaScript : Set to 1 if the code is inside
a EnableJS/DisableJS block.
#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.

```

Chapter 22

Compiler Functions

Syntax

```
Size = SizeOf(Type)
```

Description

`SizeOf` can be used to find out the size of any complex Structure (it does not work on the simple built-in types such as word and float), Interface or even variables . This can be useful in many areas such as calculating memory requirements for operations, using API commands, etc.

Example

```
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 (4+4+2)
8
9  John.Person\Name = "John"
10
11 Debug SizeOf(John) ; will be also 10
```

Note: if a variable and a structure have the same name, the structure will have the priority over the variable.

Syntax

```
Index = OffsetOf(Structure\Field)
Index = OffsetOf(Interface\Function())
```

Description

`OffsetOf` can be used to find out the index of a Structure field or the index of an Interface function. When used with an Interface , the function index is the memory offset, so it will be `IndexOfTheFunction*SizeOf(Integer)`.

Example

```
1  Structure Person
2      Name.s
3      ForName.s
4      Age.w
5  EndStructure
6
7  Debug OffsetOf(Person\Age) ; will be 8 as a string is 4 byte in memory
8          ; (16 with the 64-bit compiler, as a
9          string is 8 bytes there)
10
11 Interface ITest
12     Create()
13     Destroy(Flags)
14 EndInterface
15
16 Debug OffsetOf(ITest\Destroy()) ; will be 4
```

Syntax

```
Type = TypeOf(Object)
```

Description

`TypeOf` can be used to find out the type of a variable , or a structure field . 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_Qquad
#PB_List
#PB_Array
#PB_Integer
#PB_Map
#PB_Ascii
#PB_Uunicode
```

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 name of the subsystem is case sensitive.

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. 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).

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_OFunction
#PB_Label
#PB_Prototype
#PB_Module
#PB_Enumeration

```

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

```
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 ensure that the structure match the two memory area. This function is for advanced users only and should be used with care.

Example

```
1 Structure People
2   Name$
3   LastName$
4   Map Friends$()
5   Age.l
6 EndStructure
7
8 Student.People\Name$ = "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` free the memory of a structured memory area. This is useful when the structure contains strings, array, list or map which have been allocated internally by SpiderBasic. 'Structure' is 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 ensure the structure matches the memory area. This function is for advanced users only and should be used with care.

Example

```
1 Structure People
2   Name$
3   LastName$
4   Age.1
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.
14 ;
15 Debug Student\Name$
16 Debug Student\LastName$
17 Debug Student\Age
```

Syntax

```
Bool(<boolean expression>)
```

Description

`Bool` can be used to evaluate a boolean expression outside of regular conditional operator like `If`, `While`, `Until` etc. If the boolean expression is true, it will return `#True`, otherwise it will return `#False`.

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 23

Data

Introduction

SpiderBasic 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.

Commands

Syntax

`DataSection`

Description

Start a data section.

Syntax

`EndDataSection`

Description

End a data section.

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 ','.

Example

```
1 Data.l 100, 200, -250, -452, 145
2 Data.s "Hello", "This", "is ", "What ?"
```

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 DataSection
13   NumericalData:
14     Data.l 100, 200, -250, -452, 145
15
16   StringData:
17     Data.s "Hello", "This", "is ", "What ?"
18 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.

Chapter 24

Debugger keywords in SpiderBasic

Overview

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.

Syntax

`CallDebugger`

Description

This invokes the "debugger" and freezes the program immediately. This will invoke the web browser debugger.

Syntax

`Debug <expression> [, DebugLevel]`

Description

Display the DebugOutput window and the result inside it. The expression can be any valid SpiderBasic 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.

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.

Chapter 25

Define

Syntax

```
Define.<type> [<variable> [= <expression>], <variable> [= <expression>], ...]
```

Description

If no <variables> are specified, **Define** is used to change the default type for future untyped variables (including procedure parameters and interface method parameters). The initial default type is integer (.i). Each variable can have a default value directly assigned to it.

Define may also be used with arrays , lists and maps .

Example

```
1   d = e + f
2   Define.w
3   a = b + c
```

d, e and f will be created as integer type variables, since there was no type specified. a, b and c will be signed word typed (.w) as no type is specified and the default type had been changed to the word type. If variables are specified, **Define** only declares these variables as "defined type" and will not change the default type. If you don't assign a value to the variables at this time, the value will be 0.

Example

```
1   Define.b a, b = 10, c = b*2, d ; these 4 variables will be byte typed
    (.b)
```

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 1 in ASCII mode, and 2 in unicode
   mode
6 Debug SizeOf(MyLong)      ; will print 4
7 Debug SizeOf(MyWord)      ; will print 2
```

Chapter 26

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 SpiderBasic 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 SpiderBasic (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 .

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 stays for the
2   number of dimensions in the array
3   For i = 0 To Length
4     Numbers(i) = i
5   Next
6   EndProcedure
7
8   Dim Numbers(10)
9   fill(Numbers(), 10) ; the array A() will be passed as parameter here
10
11  Debug Numbers(5)
12  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 can not 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
```

Chapter 27

Enumerations

Syntax

```
Enumeration [name] [<constant> [Step <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.

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
```

Chapter 28

For : Next

Syntax

```
For <variable> = <expression1> To <expression2> [Step <constant>]  
...  
Next [<variable>]
```

Description

The **For : Next** command 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 SpiderBasic 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 29

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. 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 |         Next
```

Chapter 30

General Rules

SpiderBasic has established rules which never change. These are:

Comments

Comments are marked by ;. All text entered after ; is ignored by the compiler.

Example

```
1 If a = 10 ; This is a comment to indicate something.
```

Keywords

All **keywords** are used for general things inside SpiderBasic, 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 for SpiderBasic **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

All **functions** must be followed by a (or else it will not be considered as a function, (even for null parameter functions).

Example

```
1 EventWindow() ; it is a function.  
2 EventWindow    ; it is a variable.
```

Functions are regularly included in the SpiderBasic "Command libraries", described on the right side of the index page in the reference manual.

Constants

All constants are preceded by #. 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.
```

Labels

All labels must be followed by a : (colon). Label names may not contain any operators (+,-,...) as well special characters (ß,ä,ö,ü,...). When defined in a procedure , the label will be only available in this procedure.

Example

```
1 I_am_a_label:
```

Expressions

An expression is something which can be evaluated. An expression can mix any variables, constants, or functions, of the same type. When you use numbers in an expression, you can add the Keyword \$ sign in front of it to mean a hexadecimal number, or a Keyword % sign to mean a binary number. Without either of those, the number will be treated as decimal. Strings must be enclosed by inverted commas.

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 : Debug "Not Crazy" : Else : Debug "Crazy" : EndIf
```

Line continuation

If the line contains a big expression, it can be split on several lines. A split line has to end with one of the following operator: 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 OpenWindow(0,
6             10, 10,
7             320, 200,
8             "Image viewer")
```

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 SpiderBasic documentation are used the terms "commands" and "functions" as well - this is one and the same, independently of a return-value. If a value is returned by the command or function, you can read in the related command description.

Chapter 31

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. 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  
2 ; specified type  
3 Global.f Angle, Length.b, Position
```

Chapter 32

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 SpiderBasic 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 33

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.

Example

```
1  If a=10
2    Debug "a=10"
3  Else
4    Debug "a<>10"
5  EndIf
```

Example

```
1  If a=10 And b>=10 Or c=20
2    If b=15
3      Debug "b=15"
4    Else
5      Debug "Other possibility"
6    EndIf
7  Else
8    Debug "Test failure"
9  EndIf
```

Chapter 34

Import : EndImport

Syntax

```
Import "cordova command"
EndImport
```

Description

For advanced programmers. **Import : EndImport** allows to easily invoke a cordova command at build time. It's an Android and iOS functionality only. The command has to start with 'cordova' followed by any supported parameters.

Example

```
1 ; Will add the camera plugin to the app, so it can be accessed with
  inline javascript
2 ;
3 Import "cordova plugin add cordova-plugin-camera"
4 EndImport
```

Chapter 35

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.sb" ; 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.sb" ; This file will be inserted.  
2  XIncludeFile "Sources\myfile.sb" ; 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

```
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.sb"
3  XIncludeFile "Music.sb"
```

Chapter 36

Inline Javascript

Introduction

SpiderBasic allows you to include raw JavaScript directly into the source code. To activate the inline JavaScript input, you can use the compiler directives EnableJS and DisableJS . Another way is to put '!' (exclamation mark) at the beginning of the line, and the whole line will be put as is in the generated source.

Please note than the JavaScript code isn't checked by the SpiderBasic compiler, so if a syntax error occurs in this code, it will be reported in the browser when executing the code.

The IDE coloring can also be wrong sometimes with JavaScript input, but it won't impact the generated code.

Rules

Here are the naming rules to use when accessing SpiderBasic items:

- JavaScript variable name is the same in lowercase with a 'v_' prefix. It's the same for local variable, global variable and function parameter.

Example

```
1 MyVariable = 10
2 !v_myvariable++;
3 Debug MyVariable
```

- JavaScript pointer name is the same in lowercase with a 'p_' prefix. It's the same for local pointer, global pointer and function parameter.

Example

```
1 *Pointer = 10
2 !p_pointer = 20;
3 Debug *Pointer
```

- JavaScript procedure name is the same in lowercase with a 'f_' prefix.

Example

```

1 Procedure MyProcedure()
2   Debug "Hello world"
3 EndProcedure
4
5 ; Call in SpiderBasic
6 MyProcedure()
7
8 ; Call using Javascript
9 !f_myprocedure();

```

- JavaScript module name is the same in lowercase and using the '\$' sign to access module items.
Example

```

1 DeclareModule MyModule
2   MyModuleVariable.i
3 EndDeclareModule
4
5 Module MyModule
6 EndModule
7
8 !mymodule$v_mymodulevariable = 12
9 Debug MyModule::MyModuleVariable

```

Chapter 37

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 SpiderBasic, 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.

Note: The concept of objects, and the capability provided within SpiderBasic 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 ;
23 ; - 'Cube' which has the: Get/SetPosition() and Get/SetWidth()
24 ; functions.
25 ; - 'ColoredCube' which has the: Get/SetPosition(), Get/SetWidth()
26 ; and Get/SetColor() functions.
27 ; - 'TexturedCube' which has the: Get/SetPosition(), Get/SetWidth()
28 ; and Get/SetTexture() functions.
29 ;

```

Chapter 38

Licenses for the SpiderBasic 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 Verschueren
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: 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 developer:

- 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

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: 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 numebr of colors arguement.

Copyright (c) Stuart Coyle 2004-2006

Rewritten by Kornel LesiÅski (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.

Chapter 39

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 as 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.

The 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). The macros can be used recursively, but if the parameter passed contain the concatenation character '#', it won't be expanded.

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$+" - ") ; Here the line will be expanded like
7                                     ; that:
                                     ; 'MessageRequester("Hello",
                                     ; UCASE(" - "+Text$+" - "), 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
6                                     ; 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 + "): " +
9                 DoubleQuote#Expression#DoubleQuote
10            EndIf
11        CompilerEndIf
12    EndMacro
13
14    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 40

Pointers and memory access

Pointers

Using pointers is possible by placing one '*' (asterix) in front of the name of a variable , array , list or map . A pointer is a placeholder for a memory address which is usually associated to a structure .

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 SpiderBasic 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 guaranteed address integrity at the compilation time whatever the CPU address mode is.

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 | Structure Point
```

```

2     x.l
3     y.l
4 EndStructure
5
6 Define Point1.Point, Point2.Point
7
8 *CurrentPoint.Point = @Point1 ; Pointer declaration, associated to a
9   structure and initialized with Point1's address
10  *CurrentPoint \x = 10          ; Assign value 10 to Point1\x
11  *CurrentPoint.Point = @Point2 ; move to Point2's address
12  *CurrentPoint \x = 20          ; Assign value 20 to Point2\x
13
14 Debug Point1\x
15 Debug Point2\x

```

Example: Pointers and array

```

1 Structure Point
2   x.l
3   y.l
4 EndStructure
5
6 Define Point1.Point, Point2.Point
7
8 Dim *Points.Point(1) ; 2 slots array
9 *Points(0) = @Point1 ; Assign the first point variable to the first
10   array slot
11 *Points(1) = @Point2 ; Same for second
12
13 *Points(0)\x = 10 ; Modify the variables trough the pointers
14 *Points(1)\x = 20 ;
15
16 Debug Point1\x
17 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 Arithmetic

In SpiderBasic, a pointer can't be moved by adding or subtracting a number. A pointer is only allowed on a structured element. A memory buffer allocated with AllocateMemory() is not allowed.

AllocateStructure() needs to be used if a pointer is a required on a dynamically allocated element.

Address of variable

To find 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 People
2     Age.b
3     Name$
4 EndStructure
5
6 Procedure SetInfo(*People.People)
7     *People\Age = 69
8     *People\Name$ = "John"
9 EndProcedure
10
11 Define.People King
12
13 SetInfo(@King)
14
15 Debug King\Age
16 Debug King\Name$
```

Address of procedure

The most common reason to get the address of a procedure is when dealing with callbacks. Many commands in SpiderBasic requires callbacks as parameter, like BindEvent() , BindGadgetEvent() , ReadFile() etc. The address of a procedure is got using the 'at' symbol (@) in front of the procedure name including the '()'.

Example

```

1 Procedure ButtonHandler()
2     Debug "Button click event on gadget #" + EventGadget()
3 EndProcedure
4
5 OpenWindow(0, 100, 100, 200, 50, "Click test", #PB_Window_SystemMenu)
6     ButtonGadget(0, 10, 10, 180, 30, "Click me")
7
8 BindGadgetEvent(0, @ButtonHandler())
```

Chapter 41

Module

Syntax

```
DeclareModule <name>
...
EndDeclareModule

Module <name>
...
EndModule

UseModule <name>
UnuseModule <name>
```

Description

Modules are an easy way to isolate code part 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 this case, if a module name conflict, the module items won't be imported and a compiler error will be raised. `UnuseModule` remove 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 modules which needs it. It's the common way have global data available for all modules.

Default items available in modules are all SpiderBasic commands, structure and constants. Therefore module items can not be named like internal SpiderBasic commands, structure 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 SpiderBasic 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 ; Every items in this sections will be available from outside
2 ;
3 DeclareModule Ferrari
4     #FerrariName$ = "458 Italia"
5
6     Declare CreateFerrari()
7 EndDeclareModule
8
9
10 ; Every items in this sections will be private. Every names can be
11 ; used without conflict
12 ;
13 Module Ferrari
14
15     Global Initialized = #False
16
17     Procedure Init() ; Private init procedure
18         If Initialized = #False
19             Initialized = #True
20             Debug "InitFerrari()"
21         EndIf
22     EndProcedure
23
24     Procedure CreateFerrari()
25         Init()
26         Debug "CreateFerrari()"
27     EndProcedure
28
29
30
31     Procedure Init() ; Main init procedure, doesn't conflict with the
32         Ferrari Init() procedure
33         Debug "Main init()"
34     EndProcedure
35
36     Init()
37
38     Ferrari::CreateFerrari()
39     Debug Ferrari::#FerrariName$
40
41     Debug "-----"
42
43     UseModule Ferrari ; Now import all public item directly in the main
44     ; program scope
45
46     CreateFerrari()
47     Debug #FerrariName$
```

Example: With a common module

```
1 ; The common module, which will be used by others to share data
2 ;
3 DeclareModule Cars
4     Global NbCars = 0
5 EndDeclareModule
6
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 Porche
25 EndDeclareModule
26
27 Module Porche
28     UseModule Cars
29
30     NbCars+1
31 EndModule
32
33 Debug Cars::NbCars
```

Chapter 42

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  AddElement Test()
4  Test() = 1
5  AddElement Test()
```

```
6 | Test() = 2
7 |
8 | Procedure DebugList(c, List ParameterList())
9 |
10| AddElement(ParameterList())
11| ParameterList() = 3
12|
13| ForEach ParameterList()
14|   MessageRequester("List", Str(ParameterList()))
15| Next
16|
17| EndProcedure
18|
19| DebugList(10, Test())
```

Chapter 43

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 for an assignment, a new element is automatically added to the map. 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 need 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 of 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.1
3     Speed.1
4     Price.1
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
```

Chapter 44

Others Commands

Syntax

```
End [ExitCode]
```

Description

Ends the program execution correctly. If the application is a mobile application, it quits the app. If the app runs in a browser, it will try to close the current window or tab (but can fail depending of the browser).

'ExitCode' is currently ignored.

Syntax

```
Swap <expression>, <expression>
```

Description

Swaps the value of the both expression, in an optimized way. The both <expression> have to be a variable , array , list or a map element (structured or not) and have to be one of the SpiderBasic 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 45

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 it's own variables . In SpiderBasic, 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 them by using Shared or Global keywords (see also the Protected and Static keywords).

The last parameters can have a default value (need 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 returned value will be undefined.

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("SpiderBasic", "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
7     support 2 dimensions
8
9     ParameterTable(1, 2)\x = 3
10    ParameterTable(2, 2)\x = 4
```

```

11  EndProcedure
12
13  TestIt(10, Table())
14
15  MessageRequester("Table", Str(Table(1, 2)\x))

```

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 46

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 47

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).

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.

Example

```
1 Prototype ProtoMax(Value, Print = #False)
2
3 Procedure Max1(Value, Print)
4     Debug "Max1 : " + Value
5 EndProcedure
6
7 Procedure Max2(Value, Print)
8     Debug "Max2 : " + Value
9 EndProcedure
10
11 Max.ProtoMax = @Max1(); Set the function pointer
12 Max(10)
13
14 Max.ProtoMax = @Max2(); Change the function pointer
15 Max(20); the function call doesn't change
```

Chapter 48

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     Debug a
5   Until a > 10
```

This will loop until "a" takes a value > to 10, (it will loop 11 times).

Chapter 49

Residents

Description

Residents are precompiled files which are loaded when the compiler starts. They can be found in the 'residents' folder of the SpiderBasic installation path. A resident file must have the extension '.res' and can contain the following items: structures , interfaces , 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 'SpiderBasic.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 SpiderBasic 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` (-ir or -ignoreresident) and `/RESIDENT` (-r or -resident) 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 50

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.

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
```

Chapter 51

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). 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 52

Using several SpiderBasic versions on Windows

Overview

It is possible to install several SpiderBasic versions on your hard disk at the same time. This is useful to finish one project with an older SpiderBasic version, and start developing a new project with a new SpiderBasic version.

How to do it

Create different folders like "SpiderBasic_1.00" and "SpiderBasic_1.10" and install the related SpiderBasic version in each folders.

When one "SpiderBasic.exe" is started, it will assign all ".sb" files with this version of SpiderBasic. So when a source code is loaded by double-clicking on the related file, the currently assigned SpiderBasic version will be started. Beside SpiderBasic will change nothing, which can affect other SpiderBasic versions in different folders.

To avoid the automatic assignment of ".sb" files when starting the IDE, a shortcut can be created from SpiderBasic.exe with "/NOEXT" as parameter. The command line options for the IDE are described here .

Note: The settings for the IDE are saved in the %APPDATA%\SpiderBasic 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 SpiderBasic directory and disabled the creation of the .sb extension.

Chapter 53

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 54

SpiderBasic objects

Introduction

The purpose of this section is to describe the behavior, creation, and handling of objects in SpiderBasic. For the demonstration, we will use the Image object, but the same logic applies to all other SpiderBasic 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
```

Example

```
1 CreateImage(2, 640, 480) ; Create an image, the n°2
2 ResizeImage(2, 320, 240) ; Resize the n°2 image
```

```

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) ; Create a new image in the n°2 index, the
   |   old one is automatically free'ed
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. SpiderBasic 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

```

Overview of the different SpiderBasic objects

Different SpiderBasic 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 SpiderBasic differs them by their type:

- Font
- Gadget
- JSON
- Image

- Menu (not MenuItem() , as this is no object)
- RegularExpression
- Sound
- Sprite
- ToolBar
- Window
- XML

Chapter 55

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 | Procedure Change()
4 |   Static Dim Array(2)
5 |   Array(0)+1
6 |   Debug "In Procedure: " + Array(0) ; Will print 1, 2, 3 as the value
7 |   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.
```

Chapter 56

Structures

Syntax

```
Structure <name> [Extends <name>]  
...  
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. Statics arrays are supported inside structures.

Dynamic objects like arrays, lists and maps are 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**.

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.

SizeOf can be used with structures to get the size of the structure and **OffsetOf** can be used to retrieve the index 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++/JavaScript 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. 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 SpiderBasic 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.

It's possible to perform a full structure copy by using the equal affectation between two structure element of the same type.

ClearStructure can be used to clear a structured memory area. It's for advanced use only, when pointers are involved.

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)\Fornname = "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: 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'

```

Chapter 57

Subsystems

Introduction

SpiderBasic integrated commandset relies on specific libraries. Sometimes, there is different way to achieve the same goal and when it makes sense, SpiderBasic offers the possibility to change the used underlying libraries for specific commands, without changing one line of source code.

The available subsystems are located in the SpiderBasic 'subsystems' folder. Any residents or libraries found in this drawer will have precedence 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

For now, SpiderBasic doesn't have any additional subsystems.

Chapter 58

Variables and Types

Variables declaration

To declare a variable in SpiderBasic, simply type its name. You can also specify the type you want this variable to be. 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 !
```

Notes:

Variable names must not start with a number (0,1,...), contain operators (+,-,...) or special characters (ß,ä,ö,ü,...).

The variables in SpiderBasic 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 SpiderBasic 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

SpiderBasic 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	.b	1 byte	-128 to +127
Ascii	.a	1 byte	0 to +255
Character	.c	1 byte (in ascii mode)	0 to +255
Character	.c	2 bytes (in unicode mode)	0 to +65535

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: SpiderBasic 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 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 0.00000000000000001235
```

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 SpiderBasic, 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  
2      "price" and uses the result with the equals operator  
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'd 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 () (Brackets)

You can use sets of brackets 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=$8000000 << 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)	<code>~, -</code>
7	<code><<, >>, %, !</code>
6	<code> , &</code>
5	<code>*, /</code>
4	<code>+, -</code>
3	<code>>, >=, <, <=, =, <></code>
2	<code>Not</code>
1 (low)	<code>And, Or, XOr</code>

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 SpiderBasic.

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 59

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 $\neq 10$. The value of 'a' becomes 11 when b=10, the program will loop 10 times.

Chapter 60

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 anti-slash '\' character which does not 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.l
5 | EndStructure
6 |
7 | Structure Person
8 |     Name$
9 |     Age.l
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 61

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 sprite, a canvas or an image. More information can be found at [StartDrawing\(\)](#).

Note: The drawing process starts after calling [StartDrawing\(\)](#) and must end with [StopDrawing\(\)](#).

61.1 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()

61.2 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()

61.3 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()

61.4 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()

61.5 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()

61.6 RGBA

Syntax

```
Color = 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 .

See Also

Red() , Green() , Blue() , Alpha() , RGB()

61.7 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()

61.8 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.

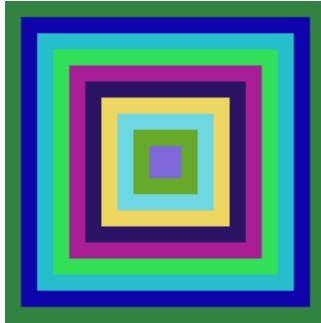
Color (optional) The color to be used for the box. If this parameter is not specified, the default color set with FrontColor() will be used. This color can be in RGB or RGBA format.

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 EndIf
```



See Also

RoundBox() , Line() , Circle() , Ellipse() FrontColor() , RGB() , RGBA() , DrawingMode()

61.9 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 EndIf
```



See Also

Box() , Line() , Circle() , Ellipse() FrontColor() , RGB() , RGBA() , DrawingMode()

61.10 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.

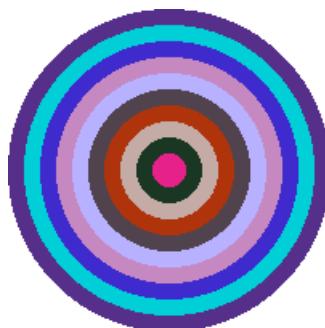
Color (optional) The color to be used for the circle. If this parameter is not specified, the default color set with FrontColor() will be used. This color can be in RGB or RGBA format.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 200, 200, "2DDrawing Example",
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 Radius = 100 To 10 Step -10
6              Circle(100, 100, Radius, RGB(Random(255), Random(255),
7                  Random(255)))
8          Next
9          StopDrawing()
10         ImageGadget(0, 0, 0, 200, 200, ImageID(0))
11     EndIf
12 EndIf
```



See Also

Box() , RoundBox() , Line() , Ellipse() FrontColor() , RGB() , RGBA() , DrawingMode()

61.11 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 size with which the image will be drawn. If these parameters are not specified then the image will be drawn in its original size.

Return value

None.

See Also

ImageID()

61.12 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()

61.13 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.



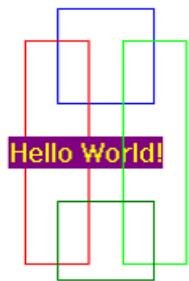
#PB_2DDrawing_Transparent

If this flag is set then the background will be transparent with the DrawText() command.



#PB_2DDrawing_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.



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()

61.14 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:

SpriteOutput() : Graphics will be rendered directly on the Sprite (for games)

ImageOutput() : Graphics will be rendered directly on the Image data (see CreateImage())

CanvasOutput() : Graphics will be rendered directly on the CanvasGadget()

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.

See Also

StopDrawing()

61.15 DrawText

Syntax

```
Result = DrawText(x, y, 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() .

Parameters

x, y 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
```

```

6   DrawText(Random(200), Random(200), "Hello World!", 
7   RGB(Random(255), Random(255), Random(255)))
8   Next i
9   StopDrawing()
10  ImageGadget(0, 0, 0, 200, 200, ImageID(0))
11  EndIf
EndIf

```



See Also

[DrawingFont\(\)](#) , [FrontColor\(\)](#) , [BackColor\(\)](#)

61.16 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

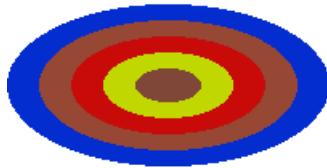
None.

Example

```

1  If OpenWindow(0, 0, 0, 200, 200, "2DDrawing Example",
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 radius=50 To 10 Step -10
6        Ellipse(100, 100, radius*2, radius, RGB(Random(255),
7          Random(255), Random(255)))
8      Next radius
9      StopDrawing()
10     ImageGadget(0, 0, 0, 200, 200, ImageID(0))
11   EndIf
12 EndIf

```



See Also

`Box()` , `RoundBox()` , `Line()` , `Circle()` `FrontColor()` , `RGB()` , `RGBA()` , `DrawingMode()`

61.17 FrontColor

Syntax

`FrontColor(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()

61.18 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

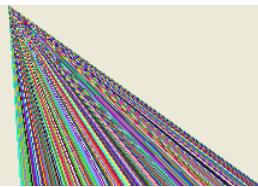
None.

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 EndIf
```



See Also

[LineXY\(\)](#) , [Box\(\)](#) , [RoundBox\(\)](#) , [Ellipse\(\)](#) , [Circle\(\)](#) [FrontColor\(\)](#) , [RGB\(\)](#) , [RGBA\(\)](#)

61.19 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.

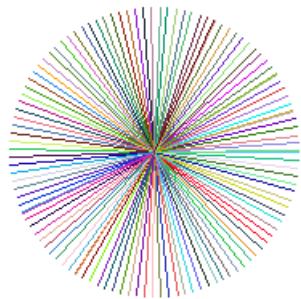
Example

```
1  If OpenWindow(0, 0, 0, 200, 200, "2DDrawing Example",
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 3
6              LineXY(100, 100, 100+Cos(Radian(Angle))*90,
7                  100+Sin(Radian(Angle))*90, RGB(Random(255), Random(255),
8                  Random(255)))
9          Next Angle
```

```

7   StopDrawing()
8 EndIf
9
10 ; Display the picture
11 ImageGadget(0, 0, 0, 200, 200, ImageID(0))
12 EndIf

```



See Also

[Line\(\)](#) , [Box\(\)](#) , [RoundBox\(\)](#) , [Ellipse\(\)](#) , [Circle\(\)](#) [FrontColor\(\)](#) , [RGB\(\)](#) , [RGBA\(\)](#)

61.20 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.

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",
2    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3    If CreateImage(0, 200, 200) And StartDrawing(ImageOutput(0))
4      For x = 0 To 199
5        For y = 0 To 199
6          Plot(x, y, RGB(Random(255), Random(255), Random(255)))
7        Next y
8      Next x
9      StopDrawing()
10     ImageGadget(0, 0, 0, 200, 200, ImageID(0))
11   EndIf

```

See Also

[Point\(\)](#) , [FrontColor\(\)](#)

61.21 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.

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.

See Also

[Plot\(\)](#) , [Red\(\)](#) , [Green\(\)](#) , [Blue\(\)](#) , [Alpha\(\)](#)

61.22 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

```
1  If OpenWindow(0, 0, 0, 200, 200, "2DDrawing Example",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      If CreateImage(0, 200, 200)
4
5          If StartDrawing(ImageOutput(0))
6              Box(50, 50, 50, 50, RGB(255, 0, 0))
7
8              ; Further drawing operation goes there
9              ; and then don't forget to call StopDrawing() to finish 2D
10             rendering
11
12             StopDrawing()
13             EndIf
14             EndIf
15
16             ; Display the picture
17             ImageGadget(0, 0, 0, 200, 200, ImageID(0))
18             EndIf
```

See Also

StartDrawing()

61.23 TextHeight

Syntax

```
Height = 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 in pixels.

See Also

[TextWidth\(\)](#) , [DrawingFont\(\)](#)

61.24 TextWidth

Syntax

```
Width = 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 in pixels.

See Also

[TextHeight\(\)](#) , [DrawingFont\(\)](#)

61.25 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()`

61.26 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()`

61.27 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\(\)](#)

Chapter 62

Accelerometer

Overview

SpiderBasic provides an easy access to accelerometer data, which are mainly used on phone and tablets.

62.1 StartAccelerometer

Syntax

```
Result = StartAccelerometer(Frequency)
```

Description

Starts accelerometer data monitoring.

Parameters

Frequency The frequency (in millisecond) to refresh the accelerometer data. The lower the frequency is, the more CPU consuming it is. This is a reference value, depending of the device it can lack of precision. AccelerometerTime() can be used to monitor the delay between 2 different fetches.

Return value

Nonzero if accelerometer function are availables (ie: the device have an accelerometer), zero otherwise.

See Also

StopAccelerometer() , AccelerometerX() , AccelerometerY() , AccelerometerZ() , AccelerometerTime()

62.2 StopAccelerometer

Syntax

```
StopAccelerometer()
```

Description

Stops accelerometer data monitoring.

Parameters

None.

Return value

None.

See Also

StartAccelerometer()

62.3 AccelerometerX

Syntax

```
Result.d = AccelerometerX()
```

Description

Returns the current acceleration on 'x' axis (in m/s²). StartAccelerometer() has to be called successfully before using this command.

Parameters

None.

Return value

Returns the current acceleration on 'x' axis (in m/s²).

See Also

StartAccelerometer() , AccelerometerY() , AccelerometerZ() , AccelerometerTime()

62.4 AccelerometerY

Syntax

```
Result.d = AccelerometerY()
```

Description

Returns the current acceleration on 'y' axis (in m/s²). StartAccelerometer() has to be called successfully before using this command.

Parameters

None.

Return value

Returns the current acceleration on 'y' axis (in m/s²).

See Also

StartAccelerometer() , AccelerometerX() , AccelerometerZ() , AccelerometerTime()

62.5 AccelerometerZ

Syntax

```
Result.d = AccelerometerZ()
```

Description

Returns the current acceleration on 'z' axis (in m/s²). StartAccelerometer() has to be called successfully before using this command.

Parameters

None.

Return value

Returns the current acceleration on 'z' axis (in m/s²).

See Also

StartAccelerometer() , AccelerometerX() , AccelerometerY() , AccelerometerTime()

62.6 AccelerometerTime

Syntax

```
Result = AccelerometerTime()
```

Description

Returns the time (in millisecond) when the current acceleration data has been fetched. It can be used to ensure the time between two different fetches is not too wide. StartAccelerometer() has to be called successfully before using this command.

Parameters

None.

Return value

Returns the time (in millisecond) when the current acceleration data has been fetched.

See Also

[StartAccelerometer\(\)](#) , [AccelerometerX\(\)](#) , [AccelerometerY\(\)](#) , [AccelerometerZ\(\)](#)

Chapter 63

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 other hand, it provides a 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 . Arrays can be sorted using SortArray() , and can also be randomized using RandomizeArray() .

63.1 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 (or its allocation has failed), it will return -1.

Example

```
1  Dim MyArray(10)
2  Debug ArraySize(MyArray()) ; will print '10'
3
```

```
4 | Dim MultiArray(10, 20, 30)
5 | Debug ArraySize(MultiArray(), 2) ; will print '20'
```

Example

```
1 | Dim Test.q(9999999999999999)
2 |
3 | If ArraySize(Test()) <> -1
4 |   Test(12345) = 123 ; everything fine
5 | Else
6 |   Debug "Array 'Test()', couldn't be initialized."
7 | EndIf
```

See Also

ListSize() , MapSize()

63.2 CopyArray

Syntax

```
Result = CopyArray(SourceArray(), DestinationArray())
```

Description

Copy every elements 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
```

```
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\(\)](#)

63.3 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 64

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.

64.1 AddCipherBuffer

Syntax

```
AddCipherBuffer(#Cipher, *Input, InputOffset, *Output, OutputOffset,  
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. It has to be allocated with AllocateMemory() .

InputOffset The offset (in bytes) in the input buffer.

***Output** The output buffer. It has to be allocated with AllocateMemory() and be different than the input buffer.

InputOffset The offset (in bytes) in 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()

64.2 AESEncoder

Syntax

```
Result = AESEncoder(*Input, InputOffset, *Output, OutputOffset, Size,
                     *Key, Bits, *InitializationVector [, Mode])
```

Description

Encodes the specified input buffer using the AES algorithm into the output buffer.

Parameters

***Input** The input buffer. It has to be allocated with AllocateMemory() .

InputOffset The offset (in bytes) in the input buffer.

***Output** The output buffer. It has to be allocated with AllocateMemory() and be different than the input buffer.

InputOffset The offset (in bytes) in the output 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.

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 ; Easy procedure to convert a datasection to an allocated memory block
2 ;
3 Procedure DataSectionToMemory(Size)
4     Protected *Buffer
5
6     *Buffer = AllocateMemory(Size)
7     For k = 0 To Size-1
8         Read.b value
9         PokeB(*Buffer, k, value)
10    Next
11
12    ProcedureReturn *Buffer
13 EndProcedure
14
15 Text$ = "Single block msg"
16 Debug "Original text: " + Text$
17
18 ; AES encoder can only work with memory buffer, so put our text in a
19 ; memory buffer
20 *AsciiText = AllocateMemory(128)
21 PokeS(*AsciiText, 0, Text$, -1, #PB_Ascii)
22
23 ; Get the InitializationVector value (needed for CBC encoding)
24 ;
25 Restore AESInitializationVector1
26 *AESInitializationVector1 = DataSectionToMemory(16)
27
28 ; Our private key
29 ;
30 Restore AESKey1
31 *AESKey1 = DataSectionToMemory(16)
32
33 ; The output buffer, needs to be input buffer size+1, aligned to the
34 ; next 16-byte block
35 *Output = AllocateMemory(16+16)
36 AESEncoder(*AsciiText, 0, *Output, 0, 16, *AESKey1, 128,
37             *AESInitializationVector1)
38
39 ; Display the encoded message
40 ;
41 For k = 0 To 15
42     HexView$ + Hex(PeekB(*Output, k), #PB_Byte) + " "
43 Next
```

```

43 |     Debug "Text encoded (hex view): " + HexView$ 
44 | 
45 | ; Decode it with the same InitializationVector and private key
46 | ;
47 | *DecodedText = AllocateMemory(128)
48 | AESDecoder(*Output, 0, *DecodedText, 0, 32, *AESKey1, 128,
49 |             *AESInitializationVector1)
50 | Debug "Decoded text: " + PeekS(*DecodedText, 0, 16, #PB_Ascii)
51 | 
52 | DataSection
53 |     AESInitializationVector1:
54 |         Data.b $3d, $af, $ba, $42, $9d, $9e, $b4, $30, $b4, $22, $da,
55 |         $80, $2c, $9f, $ac, $41
56 | 
57 |     AESKey1:
58 |         Data.b $06, $a9, $21, $40, $36, $b8, $a1, $5b, $51, $2e, $03,
59 |         $d5, $34, $12, $00, $06
60 | 
61 | EndDataSection

```

See Also

AESDecoder() , StartAESEncipher()

64.3 AESDecoder

Syntax

```
Result = AESDecoder(*Input, InputOffset, *Output, OutputOffset, Size,
                     *Key, Bits, *InitializationVector [, Mode])
```

Description

Decodes the specified input buffer using the AES algorithm into the output buffer.

Parameters

***Input** The input buffer. It has to be allocated with AllocateMemory() .

InputOffset The offset (in bytes) in the input buffer.

***Output** The output buffer. It has to be allocated with AllocateMemory() and be different than the input buffer.

InputOffset The offset (in bytes) in the output 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.

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() , StartAESEncrypt()

64.4 StartFingerprint

Syntax

```
Result = StartFingerprint(#Fingerprint, Plugin [, Bits])
```

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 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.

```

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.

```

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(128) ; Prepare a buffer with data
4  If *Buffer
5    Text$ = "The quick brown fox jumps over the lazy dog."
6    PokeS(*Buffer, 0, Text$, -1, #PB_Ascii)
7    Length = Len(Text$)
8
9    If StartFingerprint(0, #PB_Cipher_MD5) ; start the
calculation
10   AddFingerprintBuffer(0, *Buffer, 0, Length/2) ; calculate part 1
11   AddFingerprintBuffer(0, *Buffer, Length/2, Length/2) ; calculate part 2
12
13   MD5$ = FinishFingerprint(0) ; finish calculation
14   Debug "MD5 checksum = " + MD5$
15
16   MD5$ = Fingerprint(*Buffer, 0, Length, #PB_Cipher_MD5) ; compare
to a calculation in 1 step
17   Debug "MD5 checksum = " + MD5$
18 EndIf
19
20   FreeMemory(*Buffer)
21 EndIf

```

See Also

Fingerprint() , StringFingerprint()

64.5 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()

64.6 AddFingerprintBuffer

Syntax

```
AddFingerprintBuffer(#Fingerprint, *Buffer, Offset, 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.

Offset The offset (in bytes) in the buffer.

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()

64.7 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()

64.8 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()

64.9 Fingerprint

Syntax

```
Result\$ = Fingerprint(*Buffer, Offset, Size, Plugin [, Bits])
```

Description

Returns a fingerprint for the given buffer.

Parameters

***Buffer** The buffer containing the data.

Offset The offset (in bytes) in the buffer.

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.
```

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.
```

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

StartFingerprint() , StringFingerprint()

64.10 StringFingerprint

Syntax

```
Result\$ = StringFingerprint(String$, Plugin [, Bits [, Format]])
```

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.

```

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.

```

Return value

Returns the fingerprint as an hexadecimal string.

Example

```

1  UseMD5Fingerprint()
2
3  Debug StringFingerprint("yourpassword", #PB_Cipher_MD5)

```

See Also

StartFingerprint() , Fingerprint()

64.11 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 severals 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() ()

64.12 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() ()

64.13 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() ()

64.14 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() ()

64.15 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() ()

64.16 StartAESEncipher

Syntax

```
Result = StartAESEncipher(#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.

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()`

Chapter 65

Database

Overview

The database library is an easy set of functions to access and create databases. For now, it only allows to access client side database in the SQLite format. 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](#)

65.1 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\(\)](#)

65.2 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()

65.3 DatabaseColumns

Syntax

```
Result = DatabaseColumns (#Database)
```

Description

Returns the numbers of columns (fields) from the last executed database query with DatabaseQuery(). This command is only valid after a successful FirstDatabaseRow() or NextDatabaseRow() .

Parameters

#Database The database to use.

Return value

Returns the number of columns from the last database query.

See Also

DatabaseColumnName()

65.4 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. This command is only valid after a successful FirstDatabaseRow() or NextDatabaseRow() .

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. This is only valid after having executed a query with DatabaseQuery() .

See Also

DatabaseQuery() , GetDatabaseBlob() , GetDatabaseDouble() , GetDatabaseFloat() ,
GetDatabaseString() , GetDatabaseQuad()

65.5 DatabaseColumnName

Syntax

```
Text\$ = DatabaseColumnName(#Database, Column)
```

Description

Return the name of the specified column in the #Database. This command is only valid after a successful FirstDatabaseRow() or NextDatabaseRow() .

Parameters

#Database The database to use.

Column The column to use.

Return value

Returns the name of the column.

See Also

[DatabaseColumns\(\)](#)

65.6 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   Debug "Can't execute the query: "+DatabaseError()
8 EndIf
```

See Also

[DatabaseQuery\(\)](#) , [DatabaseUpdate\(\)](#)

65.7 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.

65.8 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 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 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

```
1 ; SQLite 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
```

See Also

DatabaseUpdate(), NextDatabaseRow(), SetDatabaseString(), SetDatabaseLong(), SetDatabaseQuad(), SetDatabaseFloat(), SetDatabaseDouble(), SetDatabaseBlob(), SetDatabaseNull()

65.9 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

```
1 ; SQLite 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=?")
```

65.10 ExportDatabase

Syntax

```
ExportDatabase(#Database , Filename$)
```

Description

Exports the specified file to the user through a download.

Parameters

#File The file to export.

Filename\$ The filename to use for the download.

Return value

None.

Example

```
1 ; Create a new empty database in memory
2 If OpenDatabase(0)
3
4 ; Add new table in it
5 DatabaseUpdate(0, "CREATE TABLE food (name CHAR(50) , weight REAL ,
image BLOB )")
6
7 ; Add some records
8 DatabaseUpdate(0, "INSERT INTO food (name , weight) VALUES ('apple' ,
'10.5')")
9 DatabaseUpdate(0, "INSERT INTO food (name , weight) VALUES ('pear' ,
'5')")
10
11 ; Export it as a download
12 ExportDatabase(0, "Food.db")
13 EndIf
```

See Also

[ExportDatabaseMemory\(\)](#)

65.11 ExportDatabaseMemory

Syntax

```
*Buffer = ExportDatabaseMemory(#Database)
```

Description

Exports the full content of the specified database to a new memory buffer.

Parameters

#Database The database to export.

Return value

A new memory buffer internally allocated with AllocateMemory() containing the database content as binary form, or zero if it failed. Once the buffer is no more needed FreeMemory() must be called to release the memory.

Example

```
1 ; Create a new empty database in memory
2 If OpenDatabase(0)
3
4 ; Add new table in it
5 DatabaseUpdate(0, "CREATE TABLE food (name CHAR(50), weight REAL,
6 image BLOB)")
7
8 ; Add some records
9 DatabaseUpdate(0, "INSERT INTO food (name, weight) VALUES ('apple',
10 '10.5')")
11 DatabaseUpdate(0, "INSERT INTO food (name, weight) VALUES ('pear',
12 '5')")
13
14 ; Export it as a memory buffer
15 *Buffer = ExportDatabaseMemory(0)
16 If *Buffer
17   Debug "Export success (size: " + MemorySize(*Buffer) + ") bytes"
18 Else
19   Debug "Export failed"
20 EndIf
21 EndIf
```

See Also

ExportDatabase()

65.12 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\(\)](#)

65.13 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() , GetDatabaseLong()

65.14 GetDatabaseBlob

Syntax

```
*Buffer = GetDatabaseBlob(#Database, Column)
```

Description

Returns the content of the specified database column as a new memory buffer. This command is only valid after a successful FirstDatabaseRow() 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 new memory buffer contained the blob, or zero if it failed. The returned memory buffer is allocated with AllocateMemory() () and must be freed with FreeMemory() () when no more used.

Remarks

To determine the size of the blob, MemorySize() can be used.

Example

```

1 ; Create a new empty database in memory
2 If OpenDatabase(0)
3
4 ; Add new table in it
5 DatabaseUpdate(0, "CREATE TABLE food (name CHAR(50), image BLOB)")
6
7
8 ; Create a dummy blob (125 kb)
9 *Buffer = AllocateMemory(125000)
10
11 ; Add a record with a blob. For this, we need to use the bind
method using '?'
12 SetDatabaseString(0, 0, "blob test")
13 SetDatabaseBlob(0, 1, *Buffer, MemorySize(*Buffer)) ; Assign it to
the 2nd parameter
14
15 DatabaseUpdate(0, "INSERT INTO food (name, image) VALUES (?, ?)")
16
17 ; Now check if it's really in the db
18 If DatabaseQuery(0, "SELECT * FROM food")
19
20 While NextDatabaseRow(0)
21     Debug "name: '" + GetDatabaseString(0, 0) + "', blob size: " +
MemorySize(GetDatabaseBlob(0, 1))
22     Wend
23
24 FinishDatabaseQuery(0)
25 Else
26     Debug "query error: " + DatabaseError()
27 EndIf
28 EndIf

```

See Also

[GetDatabaseDouble\(\)](#) , [GetDatabaseFloat\(\)](#) , [GetDatabaseLong\(\)](#) , [GetDatabaseString\(\)](#) , [GetDatabaseQuad\(\)](#)

65.15 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() 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.

See Also

`GetDatabaseBlob()` , `GetDatabaseFloat()` , `GetDatabaseLong()` , `GetDatabaseString()` ,
`GetDatabaseQuad()`

65.16 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()` 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.

See Also

`GetDatabaseBlob()` , `GetDatabaseDouble()` , `GetDatabaseLong()` , `GetDatabaseString()` ,
`GetDatabaseQuad()`

65.17 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()` 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.

See Also

GetDatabaseBlob() , GetDatabaseDouble() , GetDatabaseFloat() , GetDatabaseString() ,
GetDatabaseQuad()

65.18 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() 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.

See Also

GetDatabaseBlob() , GetDatabaseDouble() , GetDatabaseFloat() , GetDatabaseString() ,
GetDatabaseLong()

65.19 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() 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.

See Also

GetDatabaseBlob() , GetDatabaseDouble() , GetDatabaseFloat() , GetDatabaseLong() ,
GetDatabaseQuad()

65.20 CheckDatabaseNull

Syntax

```
Text\$ = CheckDatabaseNull(#Database, Column)
```

Description

Checks if the content of the specified database column is null. This command is only valid after a successful FirstDatabaseRow() 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 is the data is null, #False otherwise.

See Also

GetDatabaseBlob() , GetDatabaseDouble() , GetDatabaseFloat() , GetDatabaseLong() ,
GetDatabaseQuad()

65.21 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()

65.22 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()

65.23 OpenDatabase

Syntax

```
Result = OpenDatabase(#Database [, *DatabaseBuffer])
```

Description

Opens or create a new database. If '*DatabaseBuffer' is not specified, a new empty database is created. A database is always created in memory.

Parameters

#Database A number to identify the new database. #PB_Any can be used to auto-generate this number.

*DatabaseBuffer (optional) A memory buffer containing a SQLite database. The database will be initialized using the specified database data. ExportDatabaseMemory() can be used to get a raw memory buffer of a database.

Return value

Returns nonzero if the database has been successfully created, zero otherwise. 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

CloseDatabase() , ExportDatabaseMemory()

65.24 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

```
1 ; Create a new empty database in memory
2 If OpenDatabase(0)
3
4 ; Add new table in it
5 DatabaseUpdate(0, "CREATE TABLE food (name CHAR(50), image BLOB)")
6
7
8 ; Create a dummy blob (125 kb)
9 *Buffer = AllocateMemory(125000)
10
11 ; Add a record with a blob. For this, we need to use the bind
method using '?
12 SetDatabaseString(0, 0, "blob test")
13 SetDatabaseBlob(0, 1, *Buffer, MemorySize(*Buffer)) ; Assign it to
the 2nd parameter
14
15 DatabaseUpdate(0, "INSERT INTO food (name, image) VALUES (?, ?)")
16
17 ; Now check if it's really in the db
18 If DatabaseQuery(0, "SELECT * FROM food")
19
20 While NextDatabaseRow(0)
21   Debug "name: '" + GetDatabaseString(0, 0) + "', blob size: " +
MemorySize(GetDatabaseBlob(0, 1))
22   Wend
23
24 FinishDatabaseQuery(0)
25 Else
26   Debug "query error: " + DatabaseError()
27 EndIf
28 EndIf
```

See Also

DatabaseUpdate() , GetDatabaseBlob()

65.25 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()

65.26 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()

65.27 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()

65.28 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()`

65.29 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()`

65.30 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) 2014 - Fantaisie Software ;

Chapter 66

Date

Overview

The Date library allows for the manipulation of Date and Time from 1970 up to 2038 using the Unix method (i.e. the number of seconds elapsed since the 1st of January 1970).

Note: supported date/time values are 1970-01-01, 00:00:00 for the minimum and 2038-01-19, 03:14:07 for the maximum.

66.1 AddDate

Syntax

```
Date = AddDate(Date, Type, Value)
```

Description

Add an amount of time to a date.

Parameters

Date 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.

Example

```
1 Debug FormatDate("%yyyy/%mm/%dd", AddDate(Date(), #PB_Date_Year, 2))
2 ; Returns the current date + 2 years
3
4 Debug FormatDate("%mm/%dd/%yyyy", AddDate(Date(), #PB_Date_Year, 2))
5 ; Returns the current date + 2 years
```

See Also

Date() , FormatDate()

66.2 Date

Syntax

```
Date = Date([Year, Month, Day, Hour, Minute, Second])
```

Description

Returns the date value expressing the given parameters, or the current system date if no parameters are specified.

Parameters

Year, Month, Day, Hour, Minute, Second (optional) The components of the date to return. If these parameters are not specified, then the current system date and time are used.

Return value

Returns the specified date value.

Example

```
1 Debug Date() / (3600*24*365) ; will print the number of years since
2 01/01/1970 and now
3 Debug Date(1999, 12, 31, 23, 59, 59) ; will print '946684799'
4 (number of seconds between 01/01/1970 0:00:00 and 12/31/1999
5 23:59:59)
```

See Also

FormatDate() , Year() , Month() , Day() , Hour() , Minute() , Second()

66.3 Day

Syntax

```
Result = Day(Date)
```

Description

Returns the day component of the specified date.

Parameters

Date The date value from which to extract the day.

Return value

Returns the day component. 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()

66.4 DayOfWeek

Syntax

```
Result = DayOfWeek(Date)
```

Description

Returns the weekday of the specified date.

Parameters

Date The date value from which to extract the weekday.

Return value

Returns a number between 0 and 6 representing the day of the week:

```
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()

66.5 DayOfYear

Syntax

```
Result = DayOfYear(Date)
```

Description

Returns the number of days elapsed since the beginning of the year of the specified date.

Parameters

Date The date value from which to extract the number of days.

Return value

Returns the number of days since the beginning of the year. 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()

66.6 Month

Syntax

```
Result = Month(Date)
```

Description

Returns the month value of the specified date.

Parameters

Date The date from which to extract the month.

Return value

Returns the month component of the date. 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()

66.7 Year

Syntax

```
Result = Year(Date)
```

Description

Returns the year value of the specified date.

Parameters

Date The date from which to extract the year.

Return value

Returns the year component of the date.

Example

```
1 Debug Year(Date(2002, 10, 3, 0, 0)) ; Outputs '2002'.
```

See Also

FormatDate() , Month() , Day() , Hour() , Minute() , Second()

66.8 Hour

Syntax

```
Result = Hour(Date)
```

Description

Returns the hour value of the specified date.

Parameters

Date The date from which to extract the hour.

Return value

Returns the hour component of the date. 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()

66.9 Minute

Syntax

```
Result = Minute(Date)
```

Description

Returns the minute value of the specified date.

Parameters

Date The date from which to extract the minutes.

Return value

Returns the minute component of the date. 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()

66.10 Second

Syntax

```
Result = Second(Date)
```

Description

Returns the second value of the specified date.

Parameters

Date The date from which to extract the seconds.

Return value

Returns the second component of the date. 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()

66.11 FormatDate

Syntax

```
Text\$ = FormatDate(Mask$, Date)
```

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 The date value to use.

Return value

Returns the mask string with all tokens replaced by the date values.

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()

66.12 ParseDate

Syntax

```
Date = 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 Debug ParseDate("%yy/%mm/%dd", "10/01/07") ; Returns the date  
2   value of "10/01/07"  
3  
4 Debug ParseDate("%mm/%dd/%yyyy", "01/07/2010") ; Returns the date  
5   value of "01/07/2010"
```

See Also

Date() , FormatDate()

Chapter 67

Debugger

Overview

The Debugger library provides functions for controlling the debugger, for example to empty the debug output window.

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.

67.1 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

[Debug](#) , [ClearDebugOutput\(\)](#)

67.2 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

Debug ShowDebugOutput()

Chapter 68

Desktop

Overview

The desktop library allows access to information about the user's desktop environment. In SpiderBasic the desktop is the current web browser page. It can be useful to get information about browser width, height, depth, mouse position etc.

68.1 ExamineDesktops

Syntax

```
Result = ExamineDesktops()
```

Description

This function is available for PureBasic compatibility only. It always returns 1, as the web browser can only have one web page active at once. This function must be called before using the following functions: DesktopDepth() , DesktopFrequency() , DesktopHeight() , DesktopName() and DesktopWidth() .

Parameters

None.

Return value

Always 1.

Example

```
1 MessageRequester("Desktop Information", "You have " +  
ExamineDesktops() + " active web page")
```

See Also

DesktopDepth() , DesktopFrequency() , DesktopHeight() , DesktopName() , DesktopWidth()

68.2 DesktopDepth

Syntax

```
Result = DesktopDepth(#Desktop)
```

Description

Returns the color depth of the specified desktop.

Parameters

#Desktop The index of the desktop. Should be always zero as only one web page is active at once.

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 = "+
    DesktopWidth(0) + "x" + DesktopHeight(0) + "x" + DesktopDepth(0))
```

See Also

ExamineDesktops() , DesktopFrequency() , DesktopHeight() , DesktopName() , DesktopWidth()

68.3 DesktopFrequency

Syntax

```
Result = DesktopFrequency(#Desktop)
```

Description

Returns the frequency of the specified desktop.

Parameters

#Desktop The index of the desktop. Should be always zero as only one web page is active at once.

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.

Example

```
1 ExamineDesktops()
2 Debug DesktopFrequency(0)
```

See Also

ExamineDesktops() , DesktopDepth() , DesktopHeight() , DesktopName() , DesktopWidth()

68.4 DesktopHeight

Syntax

```
Result = DesktopHeight(#Desktop)
```

Description

Returns the height of the specified desktop.

Parameters

#Desktop The index of the desktop. Should be always zero as only one web page is active at once.

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 = " +
    DesktopWidth(0) + "x" + DesktopHeight(0) + "x" + DesktopDepth(0))
```

See Also

ExamineDesktops() , DesktopDepth() , DesktopX() , DesktopY() , DesktopWidth()

68.5 DesktopX

Syntax

```
Result = DesktopX(#Desktop)
```

Description

Returns the x coordinate of the specified desktop.

Parameters

#Desktop The index of the desktop. Should be always zero as only one web page is active at once.

Return value

Will always return zero, as the web page has no offset.

Remarks

ExamineDesktops() must be called before using this function to retrieve information about the available desktops.

See Also

ExamineDesktops() , DesktopDepth() , DesktopY() , DesktopHeight() , DesktopWidth()

68.6 DesktopY

Syntax

```
Result = DesktopY(#Desktop)
```

Description

Returns the y coordinate of the specified desktop.

Parameters

#Desktop The index of the desktop. Should be always zero as only one web page is active at once.

Return value

Will always return zero, as the web page has no offset.

Remarks

ExamineDesktops() must be called before using this function to retrieve information about the available desktops.

See Also

ExamineDesktops() , DesktopDepth() , DesktopX() , DesktopHeight() , DesktopWidth()

68.7 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 web page. If the mouse cursor is not inside the web page, it will return -1.

Example

```
1 Procedure TimerEvent()
2     SetGadgetText(0, "Desktop mouse position: " + DesktopMouseX() + ", "
+ DesktopMouseY())
3 EndProcedure
4
5 If OpenWindow(0, 0, 0, 300, 30, "Desktop mouse monitor",
#PB_Window_SystemMenu | #PB_Window_ScreenCentered)
    TextGadget(0, 10, 6, 200, 20, "")
6
7
```

```

8 |     AddWindowTimer(0, 1, 50) ; Uses a 50 ms timer for the realtime
9 |     refresh
10|     BindEvent(#PB_Event_Timer, @TimerEvent(), 0, 1)
10| EndIf

```

See Also

[DesktopMouseY\(\)](#) , [DesktopX\(\)](#) , [DesktopWidth\(\)](#) , [WindowMouseX\(\)](#)

68.8 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 web page. If the mouse cursor is not inside the web page, it will returns -1.

Example

```

1 Procedure TimerEvent()
2     SetGadgetText(0, "Desktop mouse position: " + DesktopMouseX() + ", "
3     + DesktopMouseY())
4     EndProcedure
5
6 If OpenWindow(0, 0, 0, 300, 30, "Desktop mouse monitor",
7     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
8     TextGadget(0, 10, 6, 200, 20, "")
9
10    AddWindowTimer(0, 1, 50) ; Uses a 50 ms timer for the realtime
10    refresh
10    BindEvent(#PB_Event_Timer, @TimerEvent(), 0, 1)
10 EndIf

```

See Also

[DesktopMouseX\(\)](#) , [DesktopY\(\)](#) , [DesktopHeight\(\)](#) , [WindowMouseY\(\)](#)

68.9 DesktopName

Syntax

```
Result\$ = DesktopName(#Desktop)
```

Description

Returns the web browser full name for the active web page.

Parameters

#Desktop The index of the desktop. Should be always zero as only one web page is active at once.

Return value

Returns a string with the web browser full name. If this information can't be retrieved, 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()

68.10 DesktopWidth

Syntax

```
Result = DesktopWidth(#Desktop)
```

Description

Returns the width for the specified desktop.

Parameters

#Desktop The index of the desktop. Should be always zero as only one web page is active at once.

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 = " +
    DesktopWidth(0) + "x" + DesktopHeight(0) + "x" + DesktopDepth(0))
```

See Also

ExamineDesktops() , DesktopDepth() , DesktopX() , DesktopY() , DesktopHeight()

Chapter 69

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.

69.1 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()

69.2 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()

69.3 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()

69.4 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()

69.5 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()

69.6 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()

69.7 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()

69.8 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() .

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() or ParseXML() . That means it's possible to create dialogs on the fly with CreateXML() 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). All the XML attributes are case-sensitive.

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

Note: all the XML attributes are case-sensitive.

```
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/left     - only applied when expand="no", top/left
is the default
    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>
<frame> - single element container
<hyperlink>
<image>
<listicon>
<listview>
<option group> - use the same 'group' number to create linked
OptionGadget()

<panel> - can contain <tab> items only
<progressbar min max value>
<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
<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>

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='Dialog example',
5    minwidth='auto' minheight='auto' flags='#PB_Window_ScreenCentered |
6    #PB_Window_SystemMenu | #PB_Window_SizeGadget'>" +
7    "  <panel>" +
8    "    <tab text='First tab'>" +
9    "      <vbox expand='item:2'>" +
10   "        <hbox>" +
11     "          <button text='button 1' />" +
12     "          <checkbox text='checkbox 1' />" +
13     "          <button text='button 2' />" +
14     "        </hbox>" +
15     "        <editor text='content' height='150' />" +
16     "      </vbox>" +
17     "    </tab>" +
18     "    <tab text='Second tab'>" +
19     "    </tab>" +
20     "  </panel>" +
21   "</window>"
22
23  If ParseXML(#Xml, XML$) And XMLStatus(#Xml) = #PB_XML_Success
24
25  If CreateDialog(#Dialog) And OpenXMLDialog(#Dialog, #Xml, "test")
26    Debug "Dialog created"
27  Else
28    Debug "Dialog error: " + DialogError(#Dialog)
29  EndIf
30  Else
31    Debug "XML error: " + XMLError(#Xml) + " (Line: " +
32    XMLErrorLine(#Xml) + ")"
33 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 |
35     #PB_Window_SystemMenu | #PB_Window_SizeGadget'>" +
36         "    <hbox expand='item:2'>" +
37             "        <listview id='#ListView', width='100',
38                 oneevent='OnListViewEvent()'>" +
39                 "            <multibox>" +
40                     "                <container id='#GeneralContainer'>" +
41                         "                    <frame text='General'>" +
42                             "                                <vbox expand='no'>" +
43                                 "                                    <checkbox text='Enable red light' />" +
44                                 "                                    <checkbox text='Enable green light' />" +
45                                 "                                </vbox>" +
46                                 "                            </frame>" +
47                             "                        </container>" +
48                     "                <container id='#EditorContainer'>" +
49                         "                    <frame text='Editor'>" +
50                             "                                <vbox expand='no'>" +
51                                 "                                    <checkbox text='Set read only mode' />" +
52                                 "                                    <checkbox text='Duplicate line automatically' />" +
53                                 "                                    <checkbox text='Enable monospace font' />" +
54                                 "                                </vbox>" +
55                             "                            </frame>" +
56                         "                     </container>" +
57                 "             </multibox>" +
58             "         </hbox>" +
59         "     </listview>" +
60     " </window>"
```

```

55      "      </container>" +
56      " " +
57      "      <container id='#BackupContainer'>" +
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 Else
81     Debug "Dialog error: " + DialogError(#Dialog)
82 EndIf
83 Else
84     Debug "XML error: " + XMLError(#Xml) + " (Line: " +
85         XMLErrorLine(#Xml) + ")"
86 EndIf

```

Example: Gridbox example

```

1 #Dialog = 0
2 #Xml = 0
3
4 XML$ = "<window id='#PB_Any' name='test' text='Gridbox' "
5     minwidth='auto' minheight='auto' maxheight='auto'
6     flags='#PB_Window_ScreenCentered | #PB_Window_SystemMenu |
7     #PB_Window_SizeGadget'>" +
8         "<gridbox columns='6'>" +
9             "<button text='Button 1' />" +
10            "<button text='Button 2' />" +
11            "<button text='Button 3' colspan='3' />" +
12            "<button text='Button 4' />" +
13            "<button text='Button 5' rowspan='2' />" +
14            "<button text='Button 6' />" +
15            "<button text='Button 7' />" +
16            "<button text='Button 8' />" +
17            "<button text='Button 9' />" +
18            "<button text='Button 10' />" +
19            "<button text='Button 11' />" +
20            "<button text='Button 12' />" +
21        "</gridbox>" +
22    "</window>"

```

```

20
21  If ParseXML(#Xml, XML$) And XMLStatus(#Xml) = #PB_XML_Success
22
23  If CreateDialog(#Dialog) And OpenXMLDialog(#Dialog, #Xml, "test")
24    Debug "Dialog created"
25  Else
26    Debug "Dialog error: " + DialogError(#Dialog)
27  EndIf
28 Else
29  Debug "XML error: " + XMLError(#Xml) + " (Line: " +
30  XMLErrorLine(#Xml) + ")"
EndIf

```

See Also

CreateDialog()

69.9 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()

Chapter 70

File

Overview

Files in SpiderBasic can be either locals (result of OpenFileRequester()), or remote. Both are handled the same way, in a binary form, allowing any type of file manipulation.

70.1 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.

All remaining opened files are automatically closed when the program ends.
For an example see the ReadFile() or the CreateFile() functions.

See Also

CreateFile() , ReadFile()

70.2 CreateFile

Syntax

```
Result = CreateFile(#File, Filename$, Callback [, Flags])
```

Description

Create an empty file. In SpiderBasic, the file are always created in memory. To be saved, the file needs to be exported using ExportFile() or use the #PB_LocalStorage flag. The file will automatically grows when write commands are used.

Parameters

#File The number to identify the new file. #PB_Any can be used to auto-generate this number.

Filename\$ The filename to the new file. It will be used when using ExportFile()

Callback The callback to be called when data has been saved to the file. It has to use the following syntax:

```
1 Procedure Callback(Status, Filename$, File, SizeRead)
2   Select Status
3     Case #PB_Status_Saved
4       ; File correctly saved
5
6     Case #PB_Status_Error
7       ; File saving has failed
8   EndSelect
9 EndProcedure
```

Flags (optional) It can be a combination (using the '||' operand) of the following values (affects 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_Uncode: all read/write string operation will use Unicode if
not specified otherwise.
#PB_LocalStorage: will save the file on client side using its
filename when @closefile() is called. This file
could be opened again in another session in the
same browser later, but it can be wiped if the
user clear its local cache. It's also domain
related, so if the application domain name change,
the files won't be accessible anymore.
```

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.

Example

```
1 Procedure Callback(Status, Filename$, File, SizeRead)
2   Select Status
3     Case #PB_Status_Saved
4       ; File correctly saved
5
6     Case #PB_Status_Error
7       ; File saving has failed
8   EndSelect
9 EndProcedure
10
11 If CreateFile(0, "Text.txt", @Callback())           ; we create a new text
12   file
13   For a=1 To 10
14     WriteStringN(0, "Line "+Str(a))    ; we write 10 lines (each with
15     'end of line' character)
16   Next
17   For a=1 To 10
18     WriteString(0, "String"+Str(a))   ; and now we add 10 more strings
19     on the same line (because there is no 'end of line' character)
20   Next
21
22   ExportFile(0, "text/plain")
23   CloseFile(0)
24 EndIf
```

See Also

ReadFile() , CloseFile()

70.3 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() , ReadFile()

70.4 ExportFile

Syntax

```
ExportFile(#File, MimeType$ [, Flags])
```

Description

Exports the specified file to the user through a download. The file filename will be used as download name.

Parameters

#File The file to export.

MimeType\$ The mimetype to use for the file. Common types are:

```
text/plain  
text/html  
application/octet-stream  
application/json  
image/jpeg  
image/png
```

Full list can be found here: <http://www.freeformatter.com/mime-types-list.html>.

Flags (optional) Can be one the following value:

```
#PB_LocalFile: the file will be exported as a download (default).  
#PB_GoogleDriveFile: the file will be exported on google drive.  
    UseGoogleDrive()  
() should have been successfully called before using this flag.  
        If the flag #PB_GoogleDriveFile has been  
        specified with OpenFile() or CreateFile(),  
            the file on google drive will be updated. If  
            the flag wasn't specified, then a new file will be created using  
            the specified filename.
```

Return value

None.

Example

```
1 If CreateFile(0, "SomeText.txt")
2   WriteStringN(0, "First line")
3   WriteStringN(0, "Second line")
4
5   ExportFile(0, "text/plain")
6
7   CloseFile(0)
8 EndIf
```

See Also

ExportFileMemory()

70.5 ExportFileMemory

Syntax

```
*Buffer = ExportFileMemory(#File)
```

Description

Exports the full content of the specified file to a new memory buffer.

Parameters

#File The file to export.

Return value

A new memory buffer internally allocated with AllocateMemory() containing the file content. Once the buffer is no more needed FreeMemory() must be called to release the memory.

Example

```
1 If CreateFile(0, "SomeText.txt")
2   WriteStringN(0, "First line")
3   WriteStringN(0, "Second line")
4
5   *Buffer = ExportFileMemory(0)
6   Debug PeekS(*Buffer, 0, 3) ; Will display 'Fir'
7
8   CloseFile(0)
9 EndIf
```

See Also

ExportFile()

70.6 FetchData

Syntax

```
FetchData(#File, Size)
```

Description

Load data from the file, without modifying the file pointer position. This is only useful when reading local file with #PB_File_Streaming flag.

Parameters

#File The file to use.

Return value

None.

Example

```
1 Procedure ReadCallback(Status, Filename$, File, Size)
2   If Status = #PB_Status_Loaded
3     Debug "File: " + Filename$ + " - Size read: " + Size + " bytes"
4
5   ; Do something with the data
6   FileSeek(0, Size, #PB_Relative) ; Skip the read data and continue
7   the streaming
8
9   If Not Eof(0)
10    FetchData(0, 1024) ; Read the next 1024 bytes of the file
11  EndIf
12
13 ElseIf Status = #PB_Status_Error
14   Debug "Error when loading the file: " + Filename$
15 EndIf
16 EndProcedure
17
18 Procedure OpenFileRequesterCallback()
19   If NextSelectedFile()
20     ReadFile(0, SelectedFileID(), @ReadCallback(), #PB_LocalFile |
21     #PB_File_Streaming)
22     FetchData(0, 1024) ; Read the first 1024 bytes of the file
23   EndIf
24 EndProcedure
25
26 Procedure ChooseFileEvent()
27   OpenFileRequester("*.txt", @OpenFileRequesterCallback())
28 EndProcedure
29
30 OpenWindow(0, 0, 0, 300, 50, "Read file example",
31   #PB_Window_ScreenCentered)
32   ButtonGadget(0, 10, 10, 280, 30, "Choose a file...")
```

31 | `BindGadgetEvent(0, @ChooseFileEvent())`

See Also

`CreateFile()` , `CloseFile()`

70.7 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()` , `ReadFile()`

70.8 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  If CreateFile(0, "Text.txt")           ; we create a new text file
2    WriteStringN(0, "Hello world")
3
4    ; Rollback to the start of "world"
5    FileSeek(0, 6, #PB_Absolute)
6    Debug ReadString(0) ; will print "world"
7
8    CloseFile(0)
9  EndIf
```

See Also

Loc() , Lof()

70.9 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() , ReadFile()

70.10 FileProgress

Syntax

```
Result.f = FileProgress(#File)
```

Description

Return the current progress (in percent) of the current file loading. It is only available with ReadFile() without the #PB_File_Streaming flag. Can be useful to monitor big files loading.

Parameters

#File The file to use.

Return value

Return the current progress (in percent) of the current file loading.

Example

```
1 Procedure ReadCallback(Status, Filename$, File, Size)
2   If Status = #PB_Status_Loaded
3     Debug "File: " + Fillename$ + " - Size read: " + Size + " bytes"
4
5   ElseIf Status = #PB_Status_Progress
6     Debug "Loading file: " + Fillename$ + "(" + FileProgress(0) + "%)"
7
8   ElseIf Status = #PB_Status_Error
9     Debug "Error when loading the file: " + Fillename$
10    EndIf
11  EndProcedure
12
13  ReadFile(0, "yourbigfilehere", @ReadCallback())
```

See Also

ReadFile()

70.11 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.

Example

```
1  If CreateFile(0, "Text.txt") ; we create a new text file
2    WriteString(0, "Hello world")
3
4    Debug Loc(0) ; will print '11'
5
6    FileSeek(0, 2, #PB_Absolute) ; Change the file pointer position
7    Debug Loc(0) ; will print '2'
8
9    CloseFile(0)
10   EndIf
```

See Also

FileSeek() , Lof()

70.12 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  If CreateFile(0, "Text.txt") ; we create a new text file
2    WriteString(0, "Hello world")
3
4    Debug "Length of file: " + Lof(0) ; will print '11'
5    CloseFile(0)
6  EndIf
```

See Also

Loc() , FileSeek()

70.13 ReadAsciiCharacter

Syntax

```
Number.a = ReadAsciiCharacter(#File)
```

Description

Read an ascii character (1 byte) from a file.

Parameters

#File The file to read from.

Return value

Returns the read ascii character or zero if there was an error.

See Also

WriteAsciiCharacter() , ReadUnicodeCharacter() , ReadCharacter() , ReadFile()

70.14 ReadByte

Syntax

```
Number.b = ReadByte(#File)
```

Description

Read a byte (1 byte) from a file.

Parameters

#File The file to read from.

Return value

Returns the read byte or zero if there was an error.

See Also

WriteByte() , ReadFile()

70.15 ReadCharacter

Syntax

```
Result.c = ReadCharacter(#File [, Format])
```

Description

Read a character from a file.

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_Uncode : 2 bytes character UTF-16.  
#PB_UTF8    : multi-bytes character (default).
```

Return value

Returns the read character or zero if there was an error.

See Also

WriteCharacter() , ReadAsciiCharacter() , ReadUnicodeCharacter() , ReadFile()

70.16 ReadDouble

Syntax

```
Number.d = ReadDouble(#File)
```

Description

Read a double (8 bytes) from a file.

Parameters

#File The file to read from.

Return value

Returns the read double value or zero if there was an error.

See Also

WriteDouble() , ReadFile()

70.17 OpenFile

Syntax

```
Result = OpenFile(#File, Filename$, Callback [, Flags])
```

Description

Open an existing file for read and write operations.

Parameters

#File The number to identify the file. #PB_Any can be used to auto-generate this number.

Filename\$ The name of the file to read. The filename can be an URL or a local file (if the flag #PB_LocalFile is set).

Callback The callback to be called when data has been read from the file. If the flag #PB_File_Streaming is not set, the callback will be called only when the whole file has been read. It has to use the following syntax:

```
1 Procedure Callback(Status, Filename$, File, SizeRead)
2     Select Status
3         Case #PB_Status_Loaded
4             ; File correctly loaded
5
6         Case #PB_Status_Saved
7             ; File correctly saved
```

```

8      Case #PB_Status_Progress
9          ; File loading in progress, use FileProgress() get the
10         current progress
11
12     Case #PB_Status_Error
13         ; File loading has failed
14     EndSelect
15 EndProcedure

```

Flags (optional) It can be a combination (using the '||' operand) of the following values:

```

#PB_LocalFile: the filename is a local file. OpenFileRequester()
() needs to be called before
                to have access to local files. SelectedFileID()
() is used to get the
                local file identifier.
#PB_GoogleDriveFile: the filename is a google drive file
identifier. OpenFileRequester()
() can be called with the #PB_Requester_GoogleDrive
flag, SelectedFileID()
() can be used to get the google drive file identifier.
#PB_File_Streaming: the file will be read chunk by chunk, using
FetchData()
. It is only
                supported with #PB_LocalFile.
#PB_LocalStorage: will save the file on client side using its
filename when @closefile() is called. This file
                could be opened again in another session in the
same browser later, but it can be wiped if the
                user clear its local cache. It's also domain
related, so if the application domain name change,
                the files won't be accessible anymore.

```

combined with one of the following values (the following flags affect the ReadString() and ReadCharacter() behaviour):

```

#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 create a new and empty file, use the CreateFile() function.

Example

```

1 Procedure ReadCallback(Status, Filename$, File, Size)
2   If Status = #PB_Status_Loaded
3     Debug "File: " + Filename$ + " - Size: " + Size + " bytes"
4
5   ; Read the first 10 lines
6   ;
7   While Eof(0) = 0 And NbLine < 10
8     Debug ReadString(0)
9     NbLine+1
10    Wend
11
12  CloseFile(0)
13
14  ElseIf Status = #PB_Status_Error
15    Debug "Error when loading the file: " + Filename$
16  EndIf
17 EndProcedure
18
19 Procedure OpenFileDialogRequesterCallback()
20   If NextSelectedFile()
21     OpenFile(0, SelectedFileDialog(), @ReadCallback(), #PB_LocalFile)
22   EndIf
23 EndProcedure
24
25 Procedure ChooseFileEvent()
26   OpenFileDialogRequester("*.txt", @OpenFileDialogRequesterCallback())
27 EndProcedure
28
29 OpenWindow(0, 0, 0, 300, 50, "Read file example",
30   #PB_Window_ScreenCentered)
31   ButtonGadget(0, 10, 10, 280, 30, "Choose a file...")
32
33 BindGadgetEvent(0, @ChooseFileEvent())

```

See Also

CreateFile() , ReadFile() , CloseFile() , FileProgress() ()

70.18 ReadFile

Syntax

```
Result = ReadFile(#File, Filename$, Callback [, 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 name of the file to read. The filename can be an URL or a local file (if the flag **#PB_LocalFile** is set).

Callback The callback to be called when data has been read from the file. If the flag **#PB_File_Streaming** is not set, the callback will be called only when the whole file has been read. It has to use the following syntax:

```
1 Procedure Callback(Status, Filename$, File, SizeRead)
2   Select Status
3     Case #PB_Status_Loaded
4       ; File correctly loaded
5
6     Case #PB_Status_Progress
7       ; File loading in progress, use FileProgress() get the
8       current progress
9
10    Case #PB_Status_Error
11      ; File loading has failed
12  EndSelect
13 EndProcedure
```

Flags (optional) It can be a combination (using the '||' operand) of the following values:

#PB_LocalFile: the filename is a local file. **OpenFileRequester()** needs to be called before to have access to local files. **SelectedFileID()** is used to get the local file identifier.
#PB_GoogleDriveFile: the filename is a google drive file identifier. **OpenFileRequester()** can be called with the **#PB_Requester_GoogleDrive** flag, **SelectedFileID()**.
#PB_File_Streaming: the file will be read chunk by chunk, using **FetchData()**.
. It is only supported with **#PB_LocalFile**.
#PB_LocalStorage: will read the file on client side using its filename. The file should have been created before with **OpenFile()** or **CreateFile()** using the **#PB_LocalStorage** flag.

combined with one of the following values (the following flags affect the **ReadString()** and **ReadCharacter()** behaviour):

```
#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_Uncode: 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 create a new and empty file, use the CreateFile() function.

Example

```
1 Procedure ReadCallback(Status, Filename$, File, Size)
2   If Status = #PB_Status_Loaded
3     Debug "File: " + Filename$ + " - Size: " + Size + " bytes"
4
5   ; Read the first 10 lines
6   ;
7   While Eof(0) = 0 And NbLine < 10
8     Debug ReadString(0)
9     NbLine+1
10    Wend
11
12  CloseFile(0)
13
14
15 ElseIf Status = #PB_Status_Error
16   Debug "Error when loading the file: " + Filename$
17 EndIf
18 EndProcedure
19
20 Procedure OpenFileDialogerCallback()
21   If NextSelectedFile()
22     ReadFile(0, SelectedFileDialog(), @ReadCallback(), #PB_LocalFile)
23   EndIf
24 EndProcedure
25
26 Procedure ChooseFileEvent()
27   OpenFileDialoger("*.txt", @OpenFileDialogerCallback())
28 EndProcedure
29
30 OpenWindow(0, 0, 0, 300, 50, "Read file example",
#PB_Window_ScreenCentered)
31 ButtonGadget(0, 10, 10, 280, 30, "Choose a file...")
32 BindGadgetEvent(0, @ChooseFileEvent())
```

See Also

CreateFile() , CloseFile() , FileProgress() ()

70.19 ReadFloat

Syntax

```
Number.f = ReadFloat(#File)
```

Description

Read a float (4 bytes) from a file.

Parameters

#File The file to read from.

Return value

Returns the read float value or zero if there was an error.

See Also

WriteFloat() , ReadDouble() , ReadFile()

70.20 ReadInteger

Syntax

```
Number.i = ReadInteger(#File)
```

Description

Read an integer (8 bytes) from a file. In SpiderBasic 'integer' is not 64-bit but 53-bit type (Javascript standard). If a number read is greater than 2^{53} , it will be rounded.

Parameters

#File The file to read from.

Return value

Returns the read value or zero if there was an error.

See Also

WriteInteger() , ReadFile()

70.21 ReadLong

Syntax

```
Number.l = ReadLong(#File)
```

Description

Read a long (4 bytes) from a file.

Parameters

#File The file to read from.

Return value

Returns the read value or zero if there was an error.

See Also

WriteLong() , ReadFile()

70.22 ReadQuad

Syntax

```
Number.q = ReadQuad(#File)
```

Description

Read a quad (8 bytes) from a file.

Parameters

#File The file to read from.

Return value

Returns the read number or zero if there was an error.

See Also

WriteQuad() , ReadFile()

70.23 ReadData

Syntax

```
Result = ReadData(#File, *Buffer, SizeToRead)
```

Description

Read the content from the file to the specified memory buffer.

Parameters

#File The file to read from.

***Buffer** The buffer to write the read data to. It has to be allocated with AllocateMemory() .

LengthToRead The number of bytes to read.

Return value

Returns the number of bytes actually read from the file. If there is an error, the return value is zero.

See Also

WriteData() , ReadFile()

70.24 ReadString

Syntax

```
Text\$ = ReadString(#File [, Flags [, Length]])
```

Description

Read a string from a file until an 'End Of Line' 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 (default).  
#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.
```

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.

See Also

[WriteString\(\)](#) , [ReadStringFormat\(\)](#) , [ReadFile\(\)](#)

70.25 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 SpiderBasic 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](#).

See Also

WriteStringFormat() , ReadString() , ReadFile()

70.26 ReadUnicodeCharacter

Syntax

```
Number.u = ReadUnicodeCharacter(#File)
```

Description

Read a unicode character (2 bytes) from a file.

Parameters

#File The file to read from.

Return value

Returns the read character or zero if there was an error.

See Also

WriteUnicodeCharacter() , ReadAsciiCharacter() , ReadCharacter() , ReadFile()

70.27 ReadWord

Syntax

```
Number.w = ReadWord(#File)
```

Description

Read a word (2 bytes) from a file.

Parameters

#File The file to read from.

Return value

Returns the read number or zero if there was an error.

See Also

WriteWord() , ReadFile()

70.28 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

The file must be opened using a write-capable function (i.e. not with ReadFile()).

See Also

ReadAsciiCharacter() , WriteUnicodeCharacter() , WriteCharacter() , CreateFile()

70.29 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

The file must be opened using a write-capable function (i.e. not with ReadFile()).

See Also

ReadByte() , CreateFile()

70.30 WriteCharacter

Syntax

```
Result = WriteCharacter(#File, Character.c [, Format])
```

Description

Write a character value 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 UTF-16.  
#PB_UTF8    : multi-bytes character (default).
```

Return value

Returns nonzero if the operation was successful or zero if it failed.

Remarks

The file must be opened using a write-capable function (i.e. not with ReadFile()).

See Also

ReadCharacter() , WriteAsciiCharacter() , WriteUnicodeCharacter() , CreateFile()

70.31 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

The file must be opened using a write-capable function (i.e. not with ReadFile()).

See Also

ReadDouble() , WriteFloat() , CreateFile()

70.32 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

The file must be opened using a write-capable function (i.e. not with ReadFile()).

See Also

ReadFloat() , WriteDouble() , CreateFile()

70.33 WriteInteger

Syntax

```
Result = WriteInteger(#File, Number)
```

Description

Write an integer number (8 bytes) to a file. In SpiderBasic 'integer' is not 64-bit but 53-bit type (Javascript standard). If a number written is greater than 2^{53} , it will be rounded.

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

The file must be opened using a write-capable function (i.e. not with ReadFile()).

See Also

ReadInteger() , CreateFile()

70.34 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

The file must be opened using a write-capable function (i.e. not with ReadFile()).

See Also

ReadLong() , CreateFile()

70.35 WriteData

Syntax

```
Result = WriteData(#File, *Buffer, Size)
```

Description

Write the content of the specified memory buffer to a file.

Parameters

#File The file to write to.

***Buffer** The memory address of the data to write to the file. It has to be allocated with AllocateMemory() .

Size The number of bytes to write to the file.

Return value

Returns the number of bytes actually written to the file. If there is an error, the return-value is zero.

Remarks

The file must be opened using a write-capable function (i.e. not with ReadFile()).

Example

```
1 *Buffer = AllocateMemory(128)           ; allocating a memory block
2 If *Buffer
3   PokeS(*Buffer, 0, "Store this string in the memory area") ; we
4     write a string into the memory block (in UTF-8 format)
5   If CreateFile(0, "Text.txt")
6     WriteData(0, *Buffer, 10)        ; write the first 10 chars from
7     the memory block into the file
8
9   ExportFile(0, "text/plain")
10  CloseFile(0)                   ; close the previously opened
11  file and so store the written data
12  EndIf
13 EndIf
```

See Also

ReadData() , CreateFile()

70.36 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

The file must be opened using a write-capable function (i.e. not with ReadFile()).

See Also

ReadQuad() , CreateFile()

70.37 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 (default).  
#PB_Unicode: Writes the string in UTF16 format.
```

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

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.

For an example see the CreateFile() function.

See Also

ReadString() , WriteStringN() , WriteStringFormat() , CreateFile()

70.38 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 (default).  
#PB_Unicode : Writes the string in UTF16 format.
```

Return value

Returns nonzero if the operation was successful and zero if it failed.

Remarks

The file must be opened using a write-capable function (i.e. not with ReadFile()).
For an example see the CreateFile() function.

See Also

ReadString() , WriteString() , WriteStringFormat() , CreateFile()

70.39 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  
an 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

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](#).

See Also

ReadStringFormat() , WriteString() , WriteStringN() , CreateFile()

70.40 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

The file must be opened using a write-capable function (i.e. not with ReadFile()).

See Also

ReadUnicodeCharacter() , WriteAsciiCharacter() , WriteCharacter() , CreateFile()

70.41 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

The file must be opened using a write-capable function (i.e. not with ReadFile()).

See Also

ReadWord() , CreateFile()

Chapter 71

Font

Overview

Fonts are widely used on computers since it is the only way to render text in different sizes and forms.
Note: with SpiderBasic using a colored font is as yet not possible. This may be achieved by using the StartDrawing() function and then drawing directly on the visual area.

71.1 FreeFont

Syntax

```
FreeFont(#Font)
```

Description

Free the given Font, previously initialized by 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()

71.2 FontID

Syntax

```
FontID = FontID(#Font)
```

Description

Returns the unique system identifier of the #Font.

Parameters

#Font The font to use.

Return value

Returns the 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 DrawingFont() :

```
1  If OpenWindow(0, 100, 200, 300, 200, "2D Drawing Test")
2
3  If CreateImage(0, 300, 200)
4    If StartDrawing(ImageOutput(0))
5      DrawingMode(#PB_2DDrawing_Transparent)
6
7      LoadFont(0, "times", 30)
8      DrawingFont(FontID(0))
9
10     DrawText(10, 50, "Hello world !", RGB(255, 0, 0))
11
12     LoadFont(0, "verdana", 12)
13     DrawingFont(FontID(0))
14
15     DrawText(10, 150, "This is a green verdana text", RGB(0, 255,
16     0))
17
18     StopDrawing()
19   EndIf
20
21 ; Create a gadget to display the image
22 ;
23   ImageGadget(0, 0, 0, 300, 200, ImageID(0))
24 EndIf
```

See Also

LoadFont() , DrawingFont()

71.3 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()

71.4 LoadFont

Syntax

```
Result = LoadFont(#Font, Name$, Height [, 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.

Height The vertical size of the font in points.

Flags (optional) Not used.

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.

Example

```
1  If OpenWindow(0, 100, 200, 300, 200, "2D Drawing Test")
2
3  If CreateImage(0, 300, 200)
4    If StartDrawing(ImageOutput(0))
5      DrawingMode(#PB_2DDrawing_Transparent)
6
7      LoadFont(0, "times", 30)
8      DrawingFont(FontID(0))
9      DrawText(10, 50, "Hello world !", RGB(255, 0, 0))
10
11     StopDrawing()
12   EndIf
13 EndIf
14
15 ; Create a gadget to display the image
16 ;
17 ImageGadget(0, 0, 0, 300, 200, ImageID(0))
18 EndIf
```

See Also

FontID() , FreeFont()

Chapter 72

Gadget

Overview

The Gadgets in SpiderBasic (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 and toolbars 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.

72.1 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()

Example

```
1 Global Index = 1      ; "Standard column" has already index 0
2
3 Procedure AddNewColumnEvent()
4     AddGadgetColumn(0, index, "Column " + Index, 80)
5     Index + 1
6 EndProcedure
7
8 If OpenWindow(0, 0, 0, 400, 150, "ListIcon - Add Columns",
9     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
10    ListIconGadget(0, 10, 10, 380, 100, "Standard Column", 150,
11        #PB_ListIcon_GridLines)
12    ButtonGadget(1, 10, 120, 150, 20, "Add new column")
13 BindGadgetEvent(1, @AddNewColumnEvent())
EndIf
```

See Also

RemoveGadgetColumn() , ListIconGadget()

72.2 AddGadgetItem

Syntax

```
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.

#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.

Return value

None.

Remarks

The following gadgets are supported:

- ListIconGadget() : supports the ImageID.
- ListViewGadget()
- PanelGadget() : supports the ImageID.
- TreeGadget() : supports the ImageID. 'Flags' is required and specifies the new sublevel.

See Also

RemoveGadgetItem() , ClearGadgetItems() , CountGadgetItems() , ComboBoxGadget() ,
ListIconGadget() , ListViewGadget() , PanelGadget() , TreeGadget()

72.3 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.  
#PB_Button_PressedImage: Get the image displayed when the button is  
pressed.
```

- 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, 120, 100, "ButtonImage", #PB_Window_SystemMenu  
| #PB_Window_ScreenCentered)  
2  
3    CreateImage(0, 16, 16, 32, #PB_Image_Transparent)  
4    If StartDrawing(ImageOutput(0))  
5      Circle(8, 8, 7, RGB(255, 0, 0))  
6      StopDrawing()  
7    EndIf  
8  
9    ButtonImageGadget(0, 10, 10, 100, 83, ImageID(0))  
10  EndIf
```



See Also

[GetGadgetState\(\)](#) , [SetGadgetState\(\)](#) , [GetGadgetAttribute\(\)](#) , [SetGadgetAttribute\(\)](#) , [ButtonGadget\(\)](#) , [ImageID\(\)](#) , [EventGadget\(\)](#)

72.4 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.

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.  
#PB_Button_Left       : Aligns the button text at the left.  
#PB_Button_Default    : Makes the button look as if it is the  
                         default button in the window  
#PB_Button_MultiLine  : If the text is too long, it will be  
                         displayed on several lines.  
#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 EndIf
```



See Also

`SetGadgetText()` , `GetGadgetText()` , `SetGadgetState()` , `GetGadgetState()` , `ButtonImageGadget()`

72.5 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.

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.
- `SetGadgetItemState()` : Make a specific date appear bold (Windows only).
- `GetGadgetItemState()` : Get the bold state of a specific date (Windows only).
- `SetGadgetAttribute()` : With the following attributes:

```
#PB_Calendar_Minimum: Set the minimum selectable date
#PB_Calendar_Maximum: Set the maximum selectable date in this gadget.
```

- GetGadgetAttribute(): With the following attributes:

```
#PB_Calendar_Minimum: Get the minimum selectable date
#PB_Calendar_Maximum: Get the maximum selectable date of this gadget.
```

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, 230, 260, "CalendarGadget",
2   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3   CalendarGadget(0, 10, 10, 210, 240)
EndIf
```



See Also

SetGadgetState() , GetGadgetState() , SetGadgetItemState() , GetGadgetItemState() ,
 SetGadgetAttribute() , GetGadgetAttribute() , SetGadgetColor() , GetGadgetColor() , DateGadget() ,
 Date() , FormatDate()

72.6 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 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.  
#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_Transparent: The canvas background will be transparent.
```

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 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.

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_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. 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_BitButtons`

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.
```

`#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. 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
#PB_Cursor_LeftRight     : left and right arrows
#PB_CursorUpDown          : up and down arrows
#PB_Cursor_LeftUpRightDown: diagonal arrows
#PB_Cursor_LeftDownRightUp: diagonal arrows
#PB_Cursor_Invisible      : hides the cursor

```

A 'mini help' can be added to this gadget using GadgetToolTip() .

Example

```

1 Procedure CanvasEvent()
2
3   If EventType() = #PB_EventType_LButtonDown Or (EventType() =
#PB_EventType_MouseMove And GetGadgetAttribute(0,
#PB_Canvas.Buttons) & #PB_Canvas_LeftButton)
4     If StartDrawing(CanvasOutput(0))
5       x = GetGadgetAttribute(0, #PB_Canvas_MouseX)
6       y = GetGadgetAttribute(0, #PB_Canvas_MouseY)
7       Circle(x, y, 10, RGB(Random(255), Random(255), Random(255)))
8       StopDrawing()
9     EndIf
10    EndIf
11
12  EndProcedure
13
14  If OpenWindow(0, 0, 0, 220, 220, "CanvasGadget",
#PB_Window_SystemMenu | #PB_Window_ScreenCentered)
15    CanvasGadget(0, 10, 10, 200, 200)
16
17  BindGadgetEvent(0, @CanvasEvent())
18 EndIf

```



See Also

[CanvasOutput\(\)](#) , [GetGadgetAttribute\(\)](#) , [SetGadgetAttribute\(\)](#) , [EventType\(\)](#) , [StartDrawing\(\)](#)

72.7 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

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()

72.8 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
#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()

72.9 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.

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.

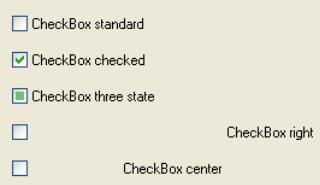
Remarks

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, 130, "CheckBoxGadget",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      CheckBoxGadget(0, 10, 10, 250, 20, "CheckBox standard")
4      CheckBoxGadget(1, 10, 40, 250, 20, "CheckBox checked") :
5          SetGadgetState(1, #PB_Checkbox_Checked)
6          CheckBoxGadget(2, 10, 70, 250, 20, "CheckBox right",
7              #PB_CheckBox_Right)
8          CheckBoxGadget(3, 10, 100, 250, 20, "CheckBox center",
9              #PB_CheckBox_Center)
10     EndIf
```



See Also

GetGadgetState() , SetGadgetState() , OptionGadget()

72.10 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()
- ListIconGadget()
- ListViewGadget()
- PanelGadget()
- TreeGadget()

Example

```
1 Procedure ClearEvent()
2     ClearGadgetItems(0)
3 EndProcedure
4
5 If OpenWindow(0, 0, 0, 220, 150, "ClearGadgetItems",
6     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
7     ListViewGadget(0,10,10,200,100)
8
9     ; add 10 items
10    For a = 1 To 10 : AddGadgetItem (0, -1, "Item " + a) : Next
11
12    ButtonGadget(1, 10, 120, 150, 20, "Clear Listview contents")
13    BindGadgetEvent(1, @ClearEvent())
14 EndIf
```

See Also

AddGadgetItem() , RemoveGadgetItem() , CountGadgetItems()

72.11 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:

- ContainerGadget()
- PanelGadget()
- ScrollAreaGadget()

See Also

OpenGadgetList() , ContainerGadget() , PanelGadget() , ScrollAreaGadget()

72.12 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.

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.
```

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:

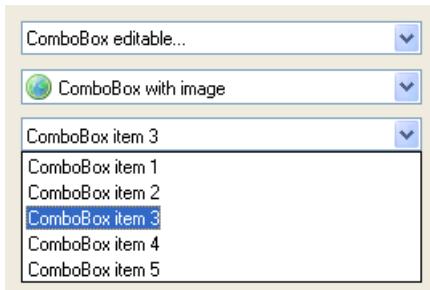
- AddGadgetItem() : Add an item.
- GetGadgetItemText() : Returns the gadget item text.
- CountGadgetItems() : Count the items in the current combobox.
- ClearGadgetItems() : Remove all the items.
- RemoveGadgetItem() : Remove an item.
- SetGadgetItemText() : Changes the gadget item text.
- SetGadgetItemImage() : 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.
- 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  If OpenWindow(0, 0, 0, 270, 85, "ComboBoxGadget",
2    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3    ComboBoxGadget(0, 10, 10, 250, 25, #PB_ComboBox_Editable)
4    AddGadgetItem(0, -1, "ComboBox editable...")
5
6    ComboBoxGadget(1, 10, 50, 250, 25)
7    For Index = 0 To 5
8      AddGadgetItem(1, -1, "ComboBox item " + Index)
9    Next
10
11   SetGadgetState(0, 0)
12   SetGadgetState(1, 2)      ; set (beginning with 0) the third item as
                               active one
13 EndIf
```



See Also

AddGadgetItem() , RemoveGadgetItem() , CountGadgetItems() , ClearGadgetItems() , GetGadgetState() , SetGadgetState() , GetGadgetText() , SetGadgetText() GetGadgetItemText() , SetGadgetItemText() , SetGadgetItemImage() GetGadgetItemData() , SetGadgetItemData()

72.13 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() :

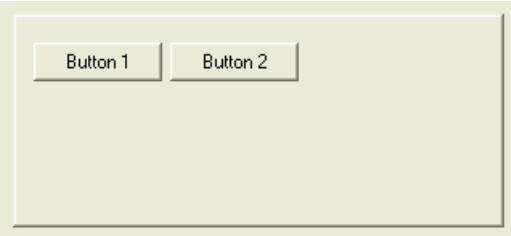
```
#PB_EventType_Resize: The gadget has been resized.
```

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()
EndIf
```



See Also

OpenGadgetList() , CloseGadgetList() , SetGadgetColor() , GetGadgetColor()

72.14 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()
- ListIconGadget()
- ListViewGadget()
- PanelGadget()
- TreeGadget()

See Also

AddGadgetItem() , RemoveGadgetItem() , ClearGadgetItems()

72.15 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 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. Only %yyyy, %mm and %dd tags are supported. No extra text characters can be set in the date format. 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
```

- SetGadgetAttribute() : With the following attributes:

```
#PB_EventType_Change: Set the minimum date that can be entered  
#PB_EventType_Change: Set the maximum date that can be entered
```

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, 200, 250, "DateGadget", #PB_Window_SystemMenu  
| #PB_Window_ScreenCentered)  
2    DateGadget(0, 10, 10, 180, 25, "%mm/%dd/%yyyy")  
3  EndIf
```



See Also

GetGadgetState() , SetGadgetState() , GetGadgetText() , SetGadgetText() , GetGadgetAttribute() , SetGadgetAttribute() , GetGadgetColor() , SetGadgetColor() , CalendarGadget() , Date() , FormatDate()

72.16 DisableGadget

Syntax

```
DisableGadget(#Gadget, State)
```

Description

Disable or enable the gadget.

Parameters

#Gadget The gadget to enable or disable.

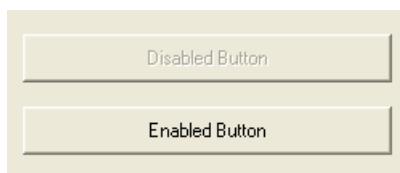
State The new state of the gadget. If State = 1, the gadget will be disabled, if State = 0 it will be enabled.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 250, 105, "Disable/enable buttons...",  
2    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
3    ButtonGadget(0, 10, 15, 230, 30, "Disabled Button") :  
4      DisableGadget(0, 1)  
5      ButtonGadget(1, 10, 60, 230, 30, "Enabled Button") :  
6      DisableGadget(1, 0)  
7  EndIf
```



See Also

HideGadget()

72.17 EditorGadget

Syntax

```
Result = EditorGadget(#Gadget, x, y, Width [, 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 the following value:

```
#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
```

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:

- GetGadgetText() : Get the text content of the editor gadget.
- 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  
#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    SetGadgetText(0, "This is a multiline text to edit" + #LF$ +  
5      "in the EditorGadget()." + #LF$ +  
6      "Third line")  
7  EndIf
```



See Also

AddGadgetItem() , RemoveGadgetItem() , CountGadgetItems() , ClearGadgetItems() ,
GetGadgetText() , SetGadgetText() , GetGadgetItemText() , SetGadgetItemText() ,
GetGadgetAttribute() , SetGadgetAttribute() , GetGadgetColor() , SetGadgetColor() , StringGadget()

72.18 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_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

```
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 EndIf
```



See Also

[GetGadgetText\(\)](#) , [SetGadgetText\(\)](#) , [ContainerGadget\(\)](#)

72.19 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.

See Also

IsGadget() , CloseWindow()

72.20 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.

72.21 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.

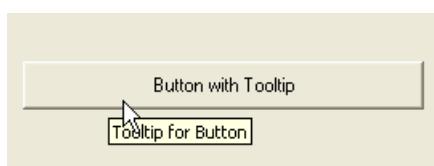
Remarks

The following gadgets are supported:

- ButtonGadget()
- ButtonImageGadget()
- CalendarGadget()
- CanvasGadget()
- CheckBoxGadget()
- ComboBoxGadget()
- ContainerGadget()
- DateGadget()
- EditorGadget()
- HyperLinkGadget()
- ImageGadget()
- ListIconGadget()
- ListViewGadget()
- OptionGadget()
- PanelGadget()
- ProgressBarGadget()
- SpinGadget()
- SplitterGadget()
- StringGadget()
- TrackBarGadget()
- TreeGadget()

Example

```
1  If OpenWindow(0, 0, 0, 270, 100, "GadgetTooltip",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      ButtonGadget(0, 10, 30, 250, 30, "Button with tooltip")
4      GadgetToolTip(0, "Tooltip for Button")
5  EndIf
```



72.22 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()

72.23 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()

72.24 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()

72.25 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:

#PB_GadgetType_Button	: ButtonGadget()
#PB_GadgetType_ButtonImage	: ButtonImageGadget()
#PB_GadgetType_Calendar	: CalendarGadget()
#PB_GadgetType_Canvas	: CanvasGadget()
#PB_GadgetType_CheckBox	: CheckBoxGadget()
#PB_GadgetType_ComboBox	: ComboBoxGadget()
#PB_GadgetType_Container	: ContainerGadget()
#PB_GadgetType_Date	: DateGadget()
#PB_GadgetType_Editor	: EditorGadget()
#PB_GadgetType_Frame	: FrameGadget()
#PB_GadgetType_HyperLink	: HyperLinkGadget()
#PB_GadgetType_Image	: ImageGadget()
#PB_GadgetType_ListIcon	: ListIconGadget()
#PB_GadgetType_ListView	: ListViewGadget()
#PB_GadgetType_Option	: OptionGadget()
#PB_GadgetType_Panel	: PanelGadget()
#PB_GadgetType_ProgressBar	: ProgressBarGadget()
#PB_GadgetType_ScrollArea	: ScrollAreaGadget()
#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_Unknown not a PB gadget at all.	: The type is unknown. Most likely it is

72.26 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()

72.27 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.

72.28 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()
- ListIconGadget()
- PanelGadget()
- ProgressBarGadget()
- ScrollAreaGadget()
- SpinGadget()
- SplitterGadget()
- StringGadget()
- TrackBarGadget()
- WebGadget()

See Also

SetGadgetAttribute() , GetGadgetItemAttribute() , SetGadgetItemAttribute()

72.29 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 -1.

Remarks

This function is supported by the following gadgets: (See each gadget description for the supported ColorType values.)

- CalendarGadget()
- ContainerGadget()
- DateGadget()
- EditorGadget()
- HyperLinkGadget()
- ListViewGadget()
- ListIconGadget()
- ProgressBarGadget()
- ScrollAreaGadget()
- SpinGadget()
- StringGadget()
- TextGadget()
- TreeGadget()

See Also

[SetGadgetColor\(\)](#)

72.30 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 SpiderBasic gadgets. See SetGadgetData() for an example.

See Also

[SetGadgetData\(\)](#) , [GetGadgetItemData\(\)](#) , [SetGadgetItemData\(\)](#)

72.31 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:

- 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()

72.32 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\(\)](#)

72.33 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.
```

- 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.
- ImageGadget() : returns the ImageID of the currently displayed image.
- ListIconGadget() : returns the first selected item index, -1 if none is selected.
- ListViewGadget() : returns the currently selected item index, -1 if none is selected.
- 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.
- 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\(\)](#)

72.34 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.
- 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.
- 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\(\)](#)

72.35 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:

- CalendarGadget() : returns **#PB_Calendar_Bold** when the specified date is displayed bold, **#PB_Calendar_Normal** otherwise. 'Item' must be a SpiderBasic date value.
- 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).
```

- 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).
```

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  If  GetGadgetItemState(#ListIcon, n) & #PB_ListIcon_Checked
2      ; Item n is checked (no matter if selected)
3  EndIf
4
5  If  GetGadgetItemState(#ListIcon, n) & #PB_ListIcon_Selected
6      ; Item n is selected (no matter if checked)
7  EndIf
8
9  If  GetGadgetItemState(#ListIcon, n) = #PB_ListIcon_Checked | 
10     #PB_ListIcon_Selected
11     ; Item n is checked AND selected
12  EndIf
13
14  If  GetGadgetItemState(#ListIcon, n) & #PB_ListIcon_Checked | 
15     #PB_ListIcon_Selected
16     ; Item n is checked OR selected OR both
17  EndIf

```

See Also

[SetGadgetItemState\(\)](#) , [GetGadgetState\(\)](#) , [SetGadgetState\(\)](#)

72.36 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 normally separated with "Chr(13)+Chr(10)".
- 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\(\)](#)

72.37 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 Procedure HideEvent()
2   Static Visible = #True
3
4   If Visible = #True      ; ButtonGadget is displayed
5     HideGadget(0, 1)      ; => hide it
6     Visible = #False
7     SetGadgetText(1, "Show Button 1")
8   Else                   ; ButtonGadget is hidden
9     HideGadget(0, 0)      ; => show it
10    Visible = #True
11    SetGadgetText(1, "Hide Button 1")
12  EndIf
13
14 EndProcedure
15
16 If OpenWindow(0, 0, 0, 180, 120, "HideGadget", #PB_Window_SystemMenu
| #PB_Window_ScreenCentered)

```

```

17 |     ButtonGadget(0, 10, 10, 160, 50, "Button 1")           : button = #True
18 |         ; Button is displayed
19 |     ButtonGadget(1, 10, 80, 160, 30, "Hide Button 1")
20 |     BindGadgetEvent(1, @HideEvent())
21 | EndIf

```

See Also

[DisableGadget\(\)](#)

72.38 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 type to change the text color.

Example

```
1 If OpenWindow(0, 0, 0, 270, 160, "HyperlinkGadget",
2   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3   HyperLinkGadget(0, 10, 10, 250, 20, "Red HyperLink", RGB(255,0,0))
4   HyperLinkGadget(1, 10, 30, 250, 20, "Arial Underlined Green
5     HyperLink", RGB(0,255,0), #PB_HyperLink_Underline)
6 EndIf
```



See Also

[GetGadgetText\(\)](#) , [SetGadgetText\(\)](#) , [GetGadgetColor\(\)](#) , [SetGadgetColor\(\)](#)

72.39 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.

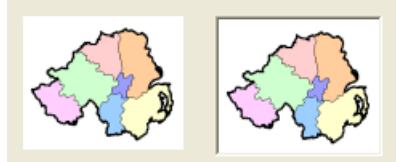
The following events are supported through EventType() :

```
#PB_EventType_LeftClick  
#PB_EventType_RightClick  
#PB_EventType_LeftDoubleClick  
#PB_EventType_RightDoubleClick
```

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    CreateImage(0, 16, 16, 32, #PB_Image_Transparent)  
3    If StartDrawing(ImageOutput(0))  
4      Circle(8, 8, 7, RGB(255, 0, 0))  
5      StopDrawing()  
6    EndIf  
7  
8    ImageGadget(0, 10, 10, 100, 83, ImageID(0));  
imagegadget standard  
9    ImageGadget(1, 130, 10, 100, 83, ImageID(0), #PB_Image_Border);  
imagegadget with border  
10 EndIf
```



See Also

GetGadgetState() , SetGadgetState() , ButtonImageGadget() , ImageID() , CanvasGadget()

72.40 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\(\)](#)

72.41 ListIconGadget

Syntax

```
Result = ListIconGadget(#Gadget, x, y, Width, Height, Title$,  
TitleWidth [, 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.

Title\$ The title for the first column in the gadget. The gadget is created with one initial column.

TitleWidth 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_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 only).
#PB_ListIcon_AlwaysShowSelection	: The selection is still visible, even when the gadget is not activated (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

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.
- 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, no more item will be selected.
- GetGadgetAttribute() / SetGadgetAttribute() : With the following attribute:

```
#PB_ListIcon_DisplayMode : Changes the display of the gadget. 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() / SetGadgetItemAttribute() : With the following attribute:

```
#PB_ListIcon_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_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
```

Example

```
1 If OpenWindow(0, 100, 100, 300, 100, "ListItemIcon Example",
2 #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3 ListIconGadget(0, 5, 5, 290, 90, "Name", 100,
4 #PB_ListIcon_FullRowSelect | #PB_ListIcon_AlwaysShowSelection)
5 AddGadgetColumn(0, 1, "Address", 250)
6 AddGadgetItem(0, -1, "Harry Rannit"+Chr(10)+"12 Parliament Way,
7 Battle Street, By the Bay")
8 AddGadgetItem(0, -1, "Ginger Brokeit"+Chr(10)+"130 SpiderBasic Road,
9 BigTown, CodeCity")
10 EndIf
```

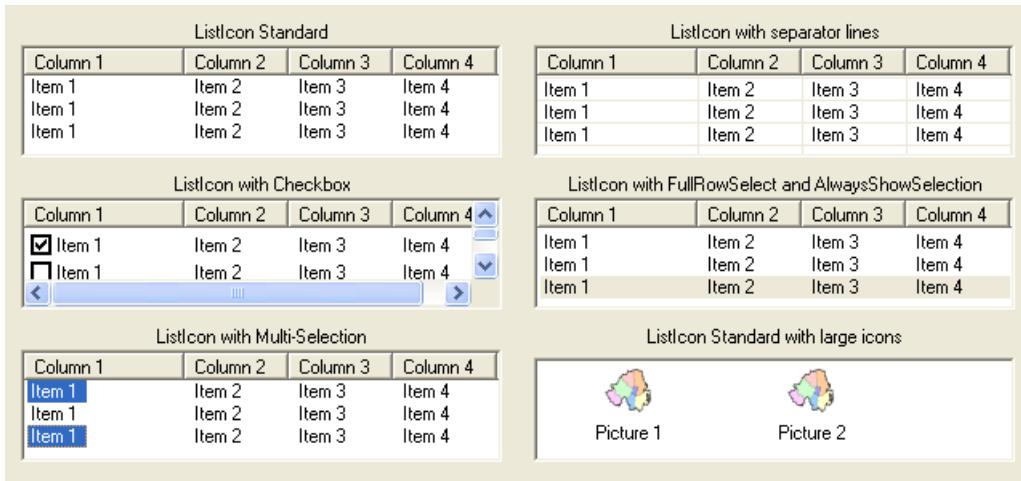
Example

```
1 ; Shows possible flags of ListIconGadget in action...
2 If OpenWindow(0, 0, 0, 640, 300, "ListItemIconGadgets",
3 #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
4 ; left column
5 TextGadget (6, 10, 10, 300, 20, "ListItemIcon Standard",
6 #PB_Text_Center)
7 ListIconGadget(0, 10, 25, 300, 70, "Column 1", 100)
8 TextGadget (7, 10, 105, 300, 20, "ListItemIcon with Checkbox",
9 #PB_Text_Center)
10 ListIconGadget(1, 10, 120, 300, 70, "Column 1", 100,
11 #PB_ListIcon_CheckBoxes) ; ListIcon with checkbox
12 TextGadget (8, 10, 200, 300, 20, "ListItemIcon with
13 Multi-Selection", #PB_Text_Center)
14 ListIconGadget(2, 10, 215, 300, 70, "Column 1", 100,
15 #PB_ListIcon_MultiSelect) ; ListIcon with multi-selection
16 ; right column
17 TextGadget (9, 330, 10, 300, 20, "ListItemIcon with separator
18 lines",#PB_Text_Center)
19 ListIconGadget(3, 330, 25, 300, 70, "Column 1", 100,
20 #PB_ListIcon_GridLines)
21 TextGadget (10, 330, 105, 300, 20, "ListItemIcon with FullRowSelect
22 and AlwaysShowSelection",#PB_Text_Center)
23 ListIconGadget(4, 330, 120, 300, 70, "Column 1", 100,
24 #PB_ListIcon_FullRowSelect | #PB_ListIcon_AlwaysShowSelection)
25 TextGadget (11, 330, 200, 300, 20, "ListItemIcon Standard with large
26 icons",#PB_Text_Center)
27 ListIconGadget(5, 330, 220, 300, 65, "", 200,#PB_ListIcon_GridLines)
28 For a = 0 To 4 ; add columns to each of the first 5
29 listicons
30 For b = 2 To 4 ; add 3 more columns to each listicon
31 AddGadgetColumn(a, b, "Column " + Str(b), 65)
32 Next
33 For b = 0 To 2 ; add 4 items to each line of the
34 listicons
35 AddGadgetItem(a, b, "Item 1"+Chr(10)+"Item 2"+Chr(10)+"Item
36 3"+Chr(10)+"Item 4")
37 Next
38 Next
39 ; Here we change the ListIcon display to large icons and show an
40 image
41 If LoadImage(0, "map2.bmp") ; change path/filename to your own
42 32x32 pixel image
```

```

27     SetGadgetAttribute(5, #PB_ListIcon_DisplayMode,
28     #PB_ListIcon_LargeIcon)
29     AddGadgetItem(5, 1, "Picture 1", ImageID(0))
30     AddGadgetItem(5, 2, "Picture 2", ImageID(0))
31 EndIf
EndIf

```



See Also

[AddGadgetColumn\(\)](#) , [RemoveGadgetColumn\(\)](#) , [AddGadgetItem\(\)](#) , [RemoveGadgetItem\(\)](#) , [ClearGadgetItems\(\)](#) , [CountGadgetItems\(\)](#) , [GetGadgetState\(\)](#) , [SetGadgetState\(\)](#) , [GetGadgetAttribute\(\)](#) , [SetGadgetAttribute\(\)](#) , [GetGadgetItemText\(\)](#) , [SetGadgetItemText\(\)](#) , [SetGadgetItemImage\(\)](#) , [GetGadgetItemState\(\)](#) , [SetGadgetItemState\(\)](#) , [GetGadgetItemData\(\)](#) , [SetGadgetItemData\(\)](#) , [GetGadgetItemAttribute\(\)](#) , [SetGadgetItemAttribute\(\)](#) , [GetGadgetColor\(\)](#) , [SetGadgetColor\(\)](#) , [ListViewGadget\(\)](#)

72.42 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
```

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.
- 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.
- GetGadgetItemSelected() : 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.
- SetGadgetItemSelected() : Selects or deselects the given item.
- SetGadgetItemText() : Set the text of the given item.
- SetGadgetState() : Change the selected item. If -1 is specified, the selection 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  
#PB_EventType_LeftDoubleClick  
#PB_EventType_RightClick
```

Example

```
1  If OpenWindow(0, 0, 0, 270, 140, "ListViewGadget",  
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
3      ListViewGadget(0, 10, 10, 250, 120)  
4      For a = 1 To 12  
5          AddGadgetItem (0, -1, "Item " + Str(a) + " of the Listview") ;  
6          define listview content  
7      Next  
8      SetGadgetState(0, 4) ; set (beginning with 0) the fifth item as the  
9      active one  
10 EndIf
```



See Also

AddGadgetItem() , RemoveGadgetItem() , ClearGadgetItems() , CountGadgetItems() ,
GetGadgetState() , SetGadgetState() , GetGadgetText() , SetGadgetText() , GetGadgetItemState() ,
SetGadgetItemState() , GetGadgetItemText() , SetGadgetItemText() , GetGadgetItemData() ,
SetGadgetItemData() , GetGadgetColor() , SetGadgetColor() , ListIconGadget()

72.43 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()

72.44 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

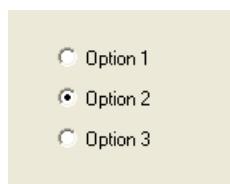
Returns nonzero on success and zero on failure. If #PB_Any was used as the #Gadget parameter then the return-value is the auto-generated gadget number on success.

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",
2    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3    OptionGadget(0, 30, 20, 100, 20, "Option 1")
4    OptionGadget(1, 30, 45, 100, 20, "Option 2")
5    OptionGadget(2, 30, 70, 100, 20, "Option 3")
6    SetGadgetState(1, 1)      ; set second option as active one
7  EndIf
```



See Also

GetGadgetText(), SetGadgetText(), GetGadgetState(), SetGadgetState(), CheckBoxGadget()

72.45 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.
- GetGadgetItemData() : Retrieve the value associated with the specified item.
- SetGadgetItemData() : Associate a value with the specified item.
- 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)

```
#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.
```

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...

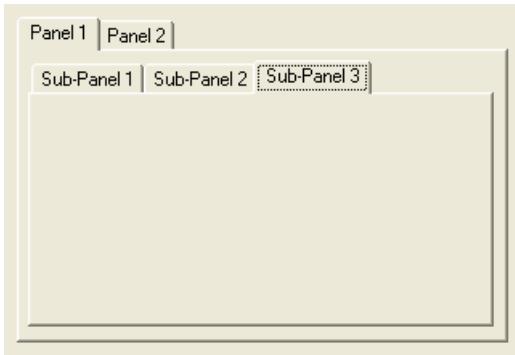
Example

```
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| EndIf

```



See Also

[AddGadgetItem\(\)](#) , [RemoveGadgetItem\(\)](#) , [CountGadgetItems\(\)](#) , [ClearGadgetItems\(\)](#) ,
[GetGadgetItemText\(\)](#) , [SetGadgetItemText\(\)](#) , [GetGadgetState\(\)](#) , [SetGadgetState\(\)](#) ,
[GetGadgetAttribute\(\)](#) , [CloseGadgetList\(\)](#) , [OpenGadgetList\(\)](#) , [SetGadgetItemImage\(\)](#)

72.46 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.

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 (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

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.
- GetGadgetState() : Get the current progress bar value.
- SetGadgetAttribute() : With the following attributes:

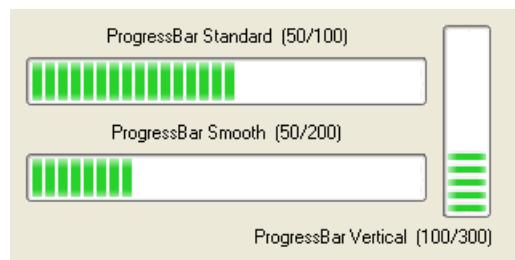
```
#PB_ProgressBar_Minimum      : Changes the minimum value.  
#PB_ProgressBar_Maximum      : Changes the maximum value.
```

- GetGadgetAttribute() : With the following attributes:

```
#PB_ProgressBar_Minimum      : Returns the minimum value.  
#PB_ProgressBar_Maximum      : Returns the maximum value.
```

Example

```
1  If OpenWindow(0, 0, 0, 320, 160, "ProgressBarGadget",  
2    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
3    TextGadget      (3, 10, 10, 250, 20, "ProgressBar Standard  
4    (50/100)", #PB_Text_Center)  
5    ProgressBarGadget(0, 10, 30, 250, 30, 0, 100)  
6    SetGadgetState  (0, 50) ; set 1st progressbar (ID = 0) to 50 of  
7    100  
8    TextGadget      (4, 10, 70, 250, 20, "ProgressBar Smooth  
9    (50/200)", #PB_Text_Center)  
10   ProgressBarGadget(1, 10, 90, 250, 30, 0, 200,  
11     #PB_ProgressBar_Smooth)  
12   SetGadgetState  (1, 50) ; set 2nd progressbar (ID = 1) to 50 of  
13   200  
14 EndIf
```



See Also

GetGadgetState() , SetGadgetState() , GetGadgetAttribute() , SetGadgetAttribute() ,
GetGadgetColor() , SetGadgetColor()

72.47 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.

Return value

None.

Remarks

The gadget type can be one of the following:

- ListIconGadget()

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     EndIf
```

See Also

AddGadgetColumn() , ListIconGadget()

72.48 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()
- PanelGadget()
- ListViewGadget()
- ListIconGadget()
- TreeGadget() - If the removed item is a node, all child-items will be removed too.

See Also

AddGadgetItem() , ClearGadgetItems() , CountGadgetItems()

72.49 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

```
1 [...]  
2  
3 ResizeGadget(0, #PB.Ignore, #PB.Ignore, 300, #PB.Ignore) ; Only  
   change the gadget width.  
4  
5 ResizeGadget(0, 150, 100, #PB.Ignore, #PB.Ignore) ; Only move the  
   gadget to a new position.
```

See Also

GadgetX() , GadgetY() , GadgetWidth() , GadgetHeight()

72.50 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.

ScrollStep (optional) The amount of pixels to scroll when the user presses the scroll bar arrows. This value is currently ignored.

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).
```

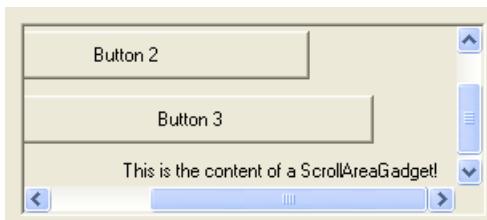
SetGadgetAttribute() : With one of the following attribute:

```
#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).
```

This gadget supports the SetGadgetColor() and GetGadgetColor() functions with the #PB_Gadget_BackColor type to change the background color.

Example

```
1 Procedure GadgetEvent()
2   Debug "Button " + EventGadget() + " pressed!"
3 EndProcedure
4
5 If OpenWindow(0, 0, 0, 305, 140, "ScrollAreaGadget",
6   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
7   ScrollAreaGadget(0, 10, 10, 290, 120, 375, 155, 30)
8     ButtonGadget (1, 10, 10, 230, 30, "Button 1")
9     ButtonGadget (2, 50, 50, 230, 30, "Button 2")
10    ButtonGadget (3, 90, 90, 230, 30, "Button 3")
11    TextGadget (4, 130, 130, 230, 20, "This is the content of a
12      ScrollAreaGadget!", #PB_Text_Right)
13    CloseGadgetList()
14
15 BindEvent(#PB_Event_Gadget, @GadgetEvent())
16 EndIf
```



See Also

GetGadgetAttribute() , SetGadgetAttribute()

72.51 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.

Return value

None.

Example

```
1 Procedure ActivateStringGadgetEvent()
2     SetActiveGadget(0)      ; Activate StringGadget
3 EndProcedure
4
5 Procedure ActivateComboBoxGadgetEvent()
6     SetActiveGadget(1)      ; Activate ComboBoxGadget
7 EndProcedure
8
9 If OpenWindow(0, 0, 0, 270, 140, "SetActiveGadget",
10    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
11    StringGadget (0, 10, 10, 250, 20, "bla bla...")
12    ComboBoxGadget(1, 10, 40, 250, 21)
13    For a = 1 To 5 : AddGadgetItem(1, -1, "ComboBox item " + Str(a)) :
14    Next
15    SetGadgetState(1, 2)          ; set (beginning with 0) the
16    third item as active one
17    ButtonGadget (2, 10, 90, 250, 20, "Activate StringGadget")
18    ButtonGadget (3, 10, 115, 250, 20, "Activate ComboBox")
19
20 BindGadgetEvent(2, @ActivateStringGadgetEvent())
21 BindGadgetEvent(3, @ActivateComboBoxGadgetEvent())
22 EndIf
```

See Also

[GetActiveGadget\(\)](#) , [SetActiveWindow\(\)](#)

72.52 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\(\)](#)
- [ListItemIconGadget\(\)](#)
- [ProgressBarGadget\(\)](#)
- [ScrollAreaGadget\(\)](#)
- [SpinGadget\(\)](#)
- [SplitterGadget\(\)](#)
- [StringGadget\(\)](#)
- [TrackBarGadget\(\)](#)
- [WebGadget\(\)](#)

See Also

[GetGadgetAttribute\(\)](#) , [GetGadgetItemAttribute\(\)](#) , [SetGadgetItemAttribute\(\)](#)

72.53 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 -1.

Return value

None.

Remarks

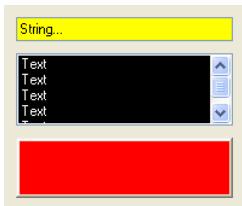
This function is supported by the following gadgets:

- CalendarGadget()
- ContainerGadget()
- DateGadget()
- EditorGadget()
- HyperLinkGadget()
- ListViewGadget()
- ListIconGadget()
- ProgressBarGadget() (Windows only)
- ScrollAreaGadget()
- SpinGadget()
- StringGadget()
- TextGadget()
- TreeGadget()

Note: With activated Windows XP style the color settings will probably be ignored or overwritten by the style.

Example

```
1 If OpenWindow(0, 0, 0, 200, 170, "SetGadgetColor",
2     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3     StringGadget(0, 10, 10, 180, 20, "String...")
4     ListViewGadget(1, 10, 40, 180, 60)
5     For i = 0 To 4
6         AddGadgetItem(1, -1, "Text")
7     Next i
8     ContainerGadget(2, 10, 110, 180, 50, #PB_Container_Raised)
9
10    SetGadgetColor(0, #PB_Gadget_BackColor, $00FFFF)
11    SetGadgetColor(1, #PB_Gadget_FrontColor, $FFFFFF)
12    SetGadgetColor(1, #PB_Gadget_BackColor, $000000)
13    SetGadgetColor(2, #PB_Gadget_BackColor, $0000FF)
13 EndIf
```



See Also

[GetGadgetColor\(\)](#)

72.54 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
2 ; Messages() array with each button. This makes the event loop simpler as not
3 ; every
4 ; gadget needs to be handled separately.
5 ;
6 Global Dim Messages.s(2)
7
7 Procedure GadgetEvent()
8   Value = GetGadgetData(EventGadget())
9   Debug "Message: " + Messages(Value)
10 EndProcedure
11
12 Messages(0) = "Good morning"
13 Messages(1) = "Hello World"
14 Messages(2) = "Nothing to say"
15 If OpenWindow(0, 0, 0, 190, 100, "SetGadgetData",
16   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
17   ButtonGadget(0, 10, 10, 80, 20, "Button"): SetGadgetData(0, 1)
18   ButtonGadget(1, 10, 40, 80, 20, "Button"): SetGadgetData(1, 2)
19   ButtonGadget(2, 10, 70, 80, 20, "Button"): SetGadgetData(2, 1)
20   ButtonGadget(3, 100, 10, 80, 20, "Button"): SetGadgetData(3, 2)
21   ButtonGadget(4, 100, 40, 80, 20, "Button") ; will have the value 0
22 because nothing was set yet
21   ButtonGadget(5, 100, 70, 80, 20, "Button")
22
23 BindEvent(#PB_Event_Gadget, @GadgetEvent())
24 EndIf
```

See Also

GetGadgetData() , GetGadgetItemData() , SetGadgetItemData()

72.55 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",
2      #PB_Window_ScreenCentered)
3      LoadFont(0, "Arial", 16)
4      LoadFont(1, "Times", 16)
5
6      ButtonGadget(0, 10, 10, 200, 30, "Arial 16")
7      SetGadgetFont(0, FontID(0))
8
9      ButtonGadget(1, 10, 50, 200, 30, "Times 32")
10     SetGadgetFont(1, FontID(1))
11
12     TextGadget(2, 10, 90, 200, 40, "Default font", #PB_Text_Center)
13 EndIf
```

See Also

FontID() , LoadFont()

72.56 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:

- ListIconGadget() :

```
#PB_ListIcon_ColumnWidth : Changes the width of the given 'Column'.
The 'Item' parameter is ignored.
```

See Also

[GetGadgetItemAttribute\(\)](#) , [GetGadgetAttribute\(\)](#) , [SetGadgetAttribute\(\)](#)

72.57 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 Procedure GadgetEvent()
5     item = GetGadgetState(3)
6 
```

```

7   Select EventGadget()
8     Case 0 ; Add
9       AddGadgetItem(3, item, "New Item")
10      If item <> -1
11        SetGadgetItemData(3, item, -1)
12      Else
13        SetGadgetItemData(3, CountGadgetItems(3)-1, -1)
14      EndIf
15
16      Case 1 ; Remove
17      If item <> -1
18        RemoveGadgetItem(3, item)
19      EndIf
20
21      Case 2 ; Test
22      If item <> -1
23        value = GetGadgetItemData(3, item)
24        If value = -1
25          Debug "Its a new item"
26        Else
27          Debug "It was item number " + value
28        EndIf
29      EndIf
30
31    EndSelect
32  EndProcedure
33
34  If OpenWindow(0, 0, 0, 280, 250, "SetGadgetItemData",
35    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
36    ButtonGadget(0, 10, 10, 80, 20, "Add")
37    ButtonGadget(1, 100, 10, 80, 20, "Remove")
38    ButtonGadget(2, 190, 10, 80, 20, "Test")
39    ListViewGadget(3, 10, 40, 260, 200)
40    For i = 0 To 10
41      AddGadgetItem(3, i, "Old Item "+Str(i))
42      SetGadgetItemData(3, i, i)
43    Next i
44
45    BindEvent(#PB_Event_Gadget, @GadgetEvent())
EndIf

```

See Also

[GetGadgetItemData\(\)](#) , [GetGadgetData\(\)](#) , [SetGadgetData\(\)](#)

72.58 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()
- TreeGadget()

See Also

AddGadgetItem()

72.59 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:

- CalendarGadget() : #PB_Calendar_Bold to make a date appear bold, #PB_Calendar_Normal otherwise.

- 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).
```

- 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.
```

1 `SetGadgetItemState(0, 1, #PB_Tree_Expanded | #PB_Tree_Selected) ; The
2nd item is selected and expanded.`

See Also

GetGadgetItemState(), GetGadgetState(), SetGadgetState()

72.60 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()
- ListIconGadget() : If Item = -1, the header text of the given column is changed.
- ListViewGadget()
- PanelGadget()
- TreeGadget()
- WebGadget() : Change the html code in the gadget with `#PB_Web_HtmlCode` as 'Item'.

See Also

`GetGadgetItemText()` , `GetGadgetText()` , `SetGadgetText()`

72.61 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.
```

- 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.
- 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.
- OptionGadget() : 1 to activate it, 0 otherwise.
- PanelGadget() : change the current panel.
- ProgressBarGadget() : change progress bar value.
- 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\(\)](#)

72.62 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)".
- 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.

See Also

[GetGadgetText\(\)](#) , [GetGadgetItemText\(\)](#) , [SetGadgetItemText\(\)](#)

72.63 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()** :

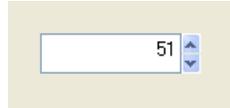
```
#PB_EventType_Change: The text in the edit area has been modified by
the user.
#PB_EventType_Up      : The 'Up' button was pressed.
#PB_EventType_Down    : The 'Down' button was pressed.
```

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 |
2   #PB_Window_ScreenCentered)
3   SpinGadget (0, 20, 20, 100, 25, 0, 1000)
4   SetGadgetState (0, 5) : SetGadgetText(0, "5") ; set initial value
EndIf
```



See Also

[GetGadgetState\(\)](#) , [SetGadgetState\(\)](#) , [GetGadgetText\(\)](#) , [SetGadgetText\(\)](#) , [GetGadgetAttribute\(\)](#) , [SetGadgetAttribute\(\)](#) , [GetGadgetColor\(\)](#) , [SetGadgetColor\(\)](#)

72.64 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:

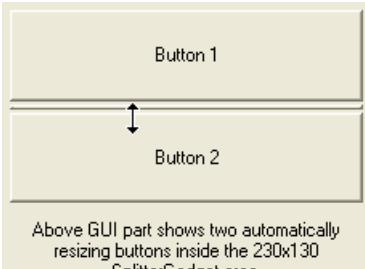
```
#PB_Splitter_FirstMinimumSize : Sets the minimum size (in pixels)  
                               than the first gadget can have.  
#PB_Splitter_SecondMinimumSize: Sets the minimum size (in pixels)  
                                than the second gadget can have.  
#PB_Splitter_FirstGadget      : 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 Enumeration
2     #Button1
3     #Button2
4     #Splitter
5 EndEnumeration
6
7 If OpenWindow(0, 0, 0, 230, 180, "SplitterGadget",
8     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
9     ButtonGadget(#Button1, 0, 0, 0, 0, "Button 1"); No need to specify
10    size or coordinates
11    ButtonGadget(#Button2, 0, 0, 0, 0, "Button 2"); as they will be
12    sized automatically
13    SplitterGadget(#Splitter, 5, 5, 220, 120, #Button1, #Button2,
#PB_Splitter_Separator)
14
15    TextGadget(3, 10, 135, 210, 40, "Above GUI part shows two
16    automatically resizing buttons inside the 220x120 SplitterGadget
17    area.",#PB_Text_Center )
18
19 EndIf
```



See Also

GetGadgetState() , SetGadgetState() , GetGadgetAttribute() , SetGadgetAttribute()

72.65 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.  
#PB_String_PlaceHolder: 'Content$' parameter will be used as a placeholder. A placeholder is  
                        a light gray string which describe the gadget use and disappear as soon as  
                        text is entered in 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",
3   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
4   StringGadget(0, 8, 10, 306, 20, "Normal StringGadget...")
5   StringGadget(1, 8, 35, 306, 20, "1234567", #PB_String_Numeric)
6   StringGadget(2, 8, 60, 306, 20, "Read-only StringGadget",
7     #PB_String_ReadOnly)
8   StringGadget(3, 8, 85, 306, 20, "lowercase...", 
9     #PB_String_LowerCase)
10  StringGadget(4, 8, 110, 306, 20, "uppercase...", 
11    #PB_String_UpperCase)
12  StringGadget(5, 8, 140, 306, 20, "Borderless StringGadget",
13    #PB_String_BorderLess)
14  StringGadget(6, 8, 170, 306, 20, "Password", #PB_String_Password)
15 EndIf
```



See Also

[GetGadgetText\(\)](#) , [SetGadgetText\(\)](#) , [GetGadgetColor\(\)](#) , [SetGadgetColor\(\)](#) , [EditorGadget\(\)](#)

72.66 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 horizontally centered in the
                            gadget.
#PB_Text_VerticalCenter: The text is vertically centered in the
                            gadget.
#PB_Text_Right            : The text is right aligned.
#PB_Text_Border           : A 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().

This gadget supports the SetGadgetColor() and GetGadgetColor() functions with the following values as 'ColorType':

```

#PB_Gadget_FrontColor: Textcolor
#PB_Gadget_BackColor : Backgroundcolor

```

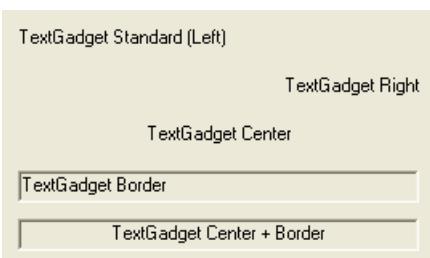
Note: This Gadget doesn't receive any user events, and GadgetToolTip() can't be used with it.

Example

```

1  If OpenWindow(0, 0, 0, 270, 160, "TextGadget", #PB_Window_SystemMenu
2    | #PB_Window_ScreenCentered)
3    TextGadget(0, 10, 10, 250, 20, "TextGadget Standard (Left)")
4    TextGadget(1, 10, 70, 250, 20, "TextGadget Center",
5      #PB_Text_Center)
6    TextGadget(2, 10, 40, 250, 20, "TextGadget Right", #PB_Text_Right)
7    TextGadget(3, 10, 100, 250, 20, "TextGadget Border",
8      #PB_Text_Border)
9    TextGadget(4, 10, 130, 250, 20, "TextGadget Center + Border",
10      #PB_Text_Center | #PB_Text_Border)
11  EndIf

```



See Also

[GetGadgetText\(\)](#) , [SetGadgetText\(\)](#) , [GetGadgetColor\(\)](#) , [SetGadgetColor\(\)](#)

72.67 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_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.
```

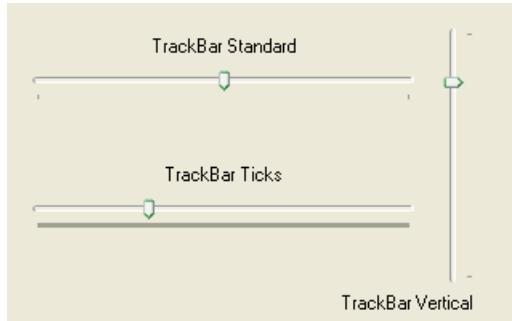
Example

```
1  If OpenWindow(0, 0, 0, 320, 200, "TrackBarGadget",  
2    #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
   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(5, 90, 180, 200, 20, "TrackBar Vertical",
#PB_Text_Right)
6   TrackBarGadget(2, 270, 10, 20, 170, 0, 10000, #PB_TrackBar_Vertical)
7   SetGadgetState(2, 8000)
8 EndIf

```



See Also

[GetGadgetState\(\)](#) , [SetGadgetState\(\)](#) , [GetGadgetAttribute\(\)](#) , [SetGadgetAttribute\(\)](#)

72.68 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.

```

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.
- 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.
```

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
```

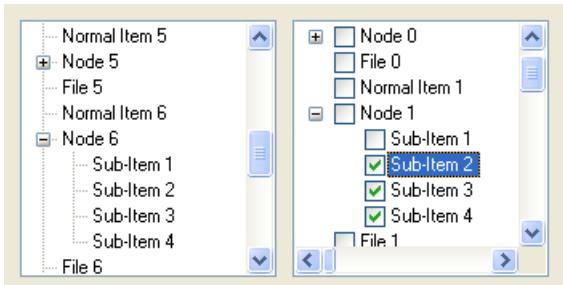
Example

```
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_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
9      1st sublevel
10     AddGadgetItem(ID, -1, "Sub-Item 2", 0, 1)
11     AddGadgetItem(ID, -1, "Sub-Item 3", 0, 1)
12     AddGadgetItem(ID, -1, "Sub-Item 4", 0, 1)
13     AddGadgetItem (ID, -1, "File "+Str(a), 0, 0) ; sublevel 0 again
14   Next
15 EndIf

```



See Also

[AddGadgetItem\(\)](#) , [RemoveGadgetItem\(\)](#) , [ClearGadgetItems\(\)](#) , [CountGadgetItems\(\)](#) , [GetGadgetItemState\(\)](#) , [SetGadgetItemState\(\)](#) , [GetGadgetItemText\(\)](#) , [SetGadgetItemText\(\)](#) , [SetGadgetItemImage\(\)](#) , [GetGadgetItemData\(\)](#) , [SetGadgetItemData\(\)](#) , [GetGadgetState\(\)](#) , [SetGadgetState\(\)](#) , [GetGadgetText\(\)](#) , [SetGadgetText\(\)](#) , [GetGadgetItemAttribute\(\)](#) , [GetGadgetColor\(\)](#) , [SetGadgetColor\(\)](#)

72.69 UseGadgetList

Syntax

```
Result = UseGadgetList(WindowID)
```

Description

Select the GadgetList window to which gadgets will be added. If there is no GadgetList on this window so far it will be created. (because it was created with the `#PB_Window_NoGadgets` flag in `OpenWindow()` or because it is not a PB window)

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", #PB_Window_SystemMenu  
| #PB_Window_ScreenCentered)  
2    ButtonGadget(0, 10, 10, 150, 25, "Button 1")  
3  
4    ; Create Window with #PB_Window_NoGadgets to prevent automatic  
GadgetList creation  
5    If OpenWindow(1, 0, 0, 300, 200, "Child Window",  
#PB_Window_TitleBar | #PB_Window_WindowCentered |  
#PB_Window_NoGadgets, WindowID(0))  
6      OldGadgetList = UseGadgetList(WindowID(1)) ; Create GadgetList  
and store old GadgetList  
7  
8      ButtonGadget(10, 10, 10, 150, 25, "Child Window Button")  
9  
10     UseGadgetList(OldGadgetList) ; Return to previous  
GadgetList  
11   EndIf  
12  
13   ButtonGadget(1, 10, 45, 150, 25, "Button 2") ; This will be on the  
main window again  
14 EndIf
```

See Also

OpenWindow() , WindowID()

72.70 WebGadget

Syntax

```
Result = WebGadget(#Gadget, x, y, Width, Height, URL$)
```

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.

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.

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.
```

- GetGadgetItemText() : The following constants can be used to get information (only works on the same page domain):

```
#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.
```

Example

```
1 If OpenWindow(0, 0, 0, 600, 300, "WebGadget", #PB_Window_SystemMenu |  
#PB_Window_ScreenCentered)  
2   WebGadget(0, 10, 10, 580, 280, "http://www.spiderbasic.com")  
3 EndIf
```



See Also

GetGadgetText() , SetGadgetText() , GetGadgetItemText() , SetGadgetState()

72.71 BindGadgetEvent

Syntax

```
BindGadgetEvent(#Gadget, @Callback() [, EventType])
```

Description

Bind a gadget event to a callback. It allows to have real-time event notifications as the callback can be invoked as soon as the event occurs (useful for ScrollAreaGadget() etc.). A gadget event can be unbinded 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.

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, 50, "Click test", #PB_Window_SystemMenu)
6     ButtonGadget(0, 10, 10, 180, 30, "Click me")
7
8 BindGadgetEvent(0, @ButtonHandler())
```

See Also

BindGadgetEvent() , BindMenuEvent() , UnbindEvent()

72.72 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 EndProcedure
4
5 OpenWindow(0, 100, 100, 200, 50, "Click test", #PB_Window_SystemMenu)
6     ButtonGadget(0, 10, 10, 180, 30, "Click me")
7
8 BindGadgetEvent(0, @ButtonHandler())
9     UnbindGadgetEvent(0, @ButtonHandler()) ; Unbind it immediately
```

See Also

`BindEvent()` , `BindGadgetEvent()` , `BindMenuEvent()`

Chapter 73

Geolocation

Overview

SpiderBasic provides an easy access to geolocation data. This can use either a real GPS or any other geolocation approximation methods (like internet connection, wifi position etc.).

73.1 StartGeolocation

Syntax

```
StartGeolocation([Timeout])
```

Description

Starts geolocation data monitoring.

Parameters

Timeout (optional) The maximum timeout (in millisecond) to have valid geolocation data. If no data can be updated during this period, all geolocation functions will returns the error value. This can be useful where up-to-date geolocation information are mandatory (ex: for a car navigation application). If not specified, no timeout is set.

Return value

None. @remarkWhen Safari is used to access the app (on OSX or iOS), it requiers the web app to be hosted using HTTPS protocol (geolocation is considered as sensitive information). If not, it won't work.

See Also

[StopGeolocation\(\)](#) , [AccelerometerY\(\)](#) , [AccelerometerZ\(\)](#) , [AccelerometerTime\(\)](#)

73.2 StopGeolocation

Syntax

```
StopGeolocation()
```

Description

Stops geolocation data monitoring.

Parameters

None.

Return value

None.

See Also

[StartGeolocation\(\)](#)

73.3 GeolocationLatitude

Syntax

```
Result.d = GeolocationLatitude()
```

Description

Returns the current latitude (in decimal degrees). StartGeolocation() has to be called successfully before using this command.

Parameters

None.

Return value

Returns the current latitude (in decimal degrees), or -1 if an error has occurred.

See Also

[StartGeolocation\(\)](#) , [GeolocationLongitude\(\)](#) , [GeolocationAltitude\(\)](#) , [GeolocationSpeed\(\)](#) , [GeolocationHeading\(\)](#) , [GeolocationTime\(\)](#)

73.4 GeolocationLongitude

Syntax

```
Result.d = GeolocationLongitude()
```

Description

Returns the current longitude (in decimal degrees). StartGeolocation() has to be called successfully before using this command.

Parameters

None.

Return value

Returns the current longitude (in decimal degrees), or -1 if an error has occurred.

See Also

StartGeolocation() , GeolocationLatitude() , GeolocationAltitude() , GeolocationSpeed() , GeolocationHeading() , GeolocationTime()

73.5 GeolocationAltitude

Syntax

```
Result.d = GeolocationAltitude()
```

Description

Returns the current altitude (in meter). StartGeolocation() has to be called successfully before using this command.

Parameters

None.

Return value

Returns the current altitude (in meter), or -1 if an error has occurred.

See Also

StartGeolocation() , GeolocationLatitude() , GeolocationLongitude() , GeolocationSpeed() ,
GeolocationHeading() , GeolocationTime()

73.6 GeolocationSpeed

Syntax

```
Result.d = GeolocationSpeed()
```

Description

Returns the current ground speed (in meters per second). StartGeolocation() has to be called successfully before using this command.

Parameters

None.

Return value

Returns the current ground speed (in meters per second), or -1 if an error has occurred.

See Also

StartGeolocation() , GeolocationLatitude() , GeolocationLongitude() , GeolocationAltitude() ,
GeolocationHeading() , GeolocationTime()

73.7 GeolocationHeading

Syntax

```
Result.d = GeolocationHeading()
```

Description

Returns the direction of the travel (in degrees). StartGeolocation() has to be called successfully before using this command.

Parameters

None.

Return value

Returns the direction of the travel (in degrees) or -1 if an error has occurred.

See Also

StartGeolocation() , GeolocationLatitude() , GeolocationLongitude() , GeolocationAltitude() ,
GeolocationSpeed() , GeolocationTime()

73.8 GeolocationTime

Syntax

```
Result = GeolocationTime()
```

Description

Returns the time (in millisecond) when the current geolocation data has been fetched. It can be used to ensure the time between two different fetches is not too wide. StartGeolocation() has to be called successfully before using this command.

Parameters

None.

Return value

Returns the time (in millisecond) when the current geolocation data has been fetched.

See Also

StartGeolocation() , GeolocationLatitude() , GeolocationLongitude() , GeolocationAltitude() ,
GeolocationSpeed() , GeolocationHeading()

Chapter 74

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 an 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.

74.1 HTTPRequest

Syntax

```
HTTPRequest(Type, URL$, Parameters, Callback [, UserData [, Headers()]])
```

Description

Download a file to disk from the given URL\$.

Parameters

Type The type of the request. Can be one of the following value:

```
#PB_HTTP_Get: executes a "GET" type request. The result will be
               returned in the callback
               once the request has been fully executed.
#PB_HTTP_Post: executes a "POST" type request. The result will be
                returned in the callback
                once the request has been fully executed.
#PB_HTTP_Put: executes a "PUT" type request. The result will be
                 returned in the callback
                 once the request has been fully executed.
#PB_HTTP_Patch: executes a "PATCH" type request. The result will
                  be returned in the callback
                  once the request has been fully executed.
#PB_HTTP_Delete: executes a "DELETE" type request. The result
                   will be returned in the callback
                   once the request has been fully executed.
```

URL\$ The URL to execute the request. Due to security constraints, it is only possible to execute a request on the same domain.

Parameters The parameter list.

Callback The callback to be called once the request has been executed. It has to use the following syntax:

```
1 Procedure Callback(Success, Result$, UserData)
2     ; Success : #True if the request has been correctly executed,
3     ; #False otherwise
4     ; Result$ : The result of the request, as a text object
5     ; UserData: The value specified in UserData parameter of
      ; HTTPRequest() or zero if not specified.
6 EndProcedure
```

UserData (optional) A custom user value which will be passed to the callback in the "UserData" parameter.

Headers() (optional) A string map of custom headers to set when sending the request.

Return value

None.

Example

```
1 Procedure HttpGetEvent(Success, Result$, UserData)
2     If Success
3         Debug Result$
4     Else
5         Debug "HTTPRequest(): Error"
6     EndIf
7 EndProcedure
8
9     ; Get the content of this file, and display it in the debug window
10    ;
11    HTTPRequest(#PB_HTTP_Get, #PB_Compiler_Filename, "", @HttpGetEvent())
```

Example: With custom headers

```
1 Procedure HttpGetEvent(Success, Result$, UserData)
2     If Success
3         Debug Result$
4     Else
5         Debug "HTTPRequest(): Error"
6     EndIf
7 EndProcedure
8
9     NewMap Headers$()
10    Headers$("x-customheader") = "test"
11    Headers$("x-customvalue") = "10"
12
13    HTTPRequest(#PB_HTTP_Get, #PB_Compiler_Filename, "", @HttpGetEvent(),
      0, Headers())
```

See Also

URLEncoder()

74.2 URLDecoder

Syntax

```
Result\$ = URLDecoder(URL$)
```

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() .

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.spiderbasic.com/test%20with%20space.php3")
2 ; Will print "http://www.spiderbasic.com/test with space.php3"
```

See Also

URLEncoder()

74.3 URLEncoder

Syntax

```
Result\$ = URLEncoder(URL$)
```

Description

Returns the URL\$ encoded to the HTTP format.

Parameters

URL\$ The URL to encode.

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.

Example

```
1 Debug URLEncoder("http://www.spiderbasic.com/test with space.php3")
2 ; Will print "http://www.spiderbasic.com/test%20with%20space.php3"
```

See Also

URLDecoder()

Chapter 75

Image

Overview

Images are graphics objects which can be displayed in a window or in several gadgets. SpiderBasic supports all image types supported by the host browser, the most common are PNG (lossless compression) and JPG (lossy compression).

Transparent PNG images can be used to enable transparency in the gadgets , menu and toolbars images.

75.1 CopyImage

Syntax

```
Result = CopyImage(#Image1, #Image2)
```

Description

Creates an identical copy of an image.

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()

75.2 CreateImage

Syntax

```
Result = CreateImage(#Image, Width, Height [, Depth [, BackColor]])
```

Description

Create an empty canvas image (with black background) which can be used to do rendering on it.

Parameters

#Image A number to identify the new canvas 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 are can be handled by all operating systems supported by SpiderBasic.

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() , FreeImage()

75.3 EncodeImage

Syntax

```
URL\$ = EncodeImage(#Image [, ImagePlugin [, Flags]])
```

Description

Encode the specified canvas image into an image URL.

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_JPEG: encode the image in JPEG  
#PB_ImagePlugin_PNG : encode the image in PNG
```

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 plugins currently support it (default quality is set to '7' if no flags are specified).

Depth (optional) The depth in which to save the image. Valid values are 24 and 32. The default value is the original image depth.

Return value

Returns the URL containing the encoded image, or an empty string if the encoding has failed.

See Also

[LoadImage\(\)](#)

75.4 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() , CopyImage() , GrabImage()

75.5 GrabImage

Syntax

```
Result = GrabImage(#Image1, #Image2, x, y, Width, Height)
```

Description

Create a new image with the selected area on the source image.

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() , CopyImage()

75.6 ImageDepth

Syntax

```
Result = ImageDepth(#Image [, Flags])
```

Description

Returns the depth of the #Image, as it is stored internally by SpiderBasic.

Parameters

#Image The image to use.

Flags (optional) Not used.

Return value

Returns always '32' as all images are 32-bit in a browser.

See Also

ImageWidth() , ImageHeight()

75.7 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_PNG
```

If the image was not loaded from one of this format, it will return zero. This is the case for images created with CreateImage() or GrabImage() .

See Also

LoadImage() , CreateImage() , GrabImage()

75.8 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()

75.9 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()

75.10 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.

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()
```

See Also

StartDrawing() , ImageVectorOutput()

75.11 ImageVectorOutput

Syntax

```
VectorOutputID = ImageVectorOutput(#Image [, Unit])
```

Description

Returns the OutputID of the image to perform 2D vector drawing operations on it.

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
#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()
```

See Also

StartVectorDrawing() , ImageOutput()

75.12 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()

75.13 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() , CopyImage() , GrabImage()

75.14 LoadImage

Syntax

```
Result = LoadImage(#Image, Filename$, [Flags])
```

Description

Load the specified image from an URL or a local 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. The filename can be an URL or a local file (if the flag #PB_LocalFile is set).

Flags (optional) It can be one of the following value:

```

#PB_LocalFile: the filename is a local file, OpenFileRequester()
() needs to be called before
          to have access to local files. SelectedFileID()
() is used to get the
          local file identifier.

```

Return value

Returns nonzero if a temporary image has been created or zero otherwise. The image is still not loaded, the callbacks binded to #PB_Event_Loading and #PB_Event>LoadingError will be called once the loading is done. If #PB_Any was specified as the #Image parameter then the auto-generated number is returned on success.

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.
- ImageGadget() for displaying image on an application window
- ButtonImageGadget() for creating an image button on an application window

Example: with an URL

```

1 Procedure Loaded(Type, Filename$, ObjectId)
2
3     ; Display the image in a new window
4     OpenWindow(#PB_Any, 10, 10, 300, 300, "Image",
5     #PB_Window_SizeGadget)
6     ImageGadget(#PB_Any, 0, 0, ImageWidth(ObjectId),
7     ImageHeight(ObjectId), ImageID(ObjectId))
8
9 EndProcedure
10
11 Procedure LoadingError(Type, Filename$, ObjectId)
12     Debug Filename$ + ": loading error"
13 EndProcedure
14
15 ; Register the loading event before calling any resource load command
16 BindEvent(#PB_Event_Loading, @Loaded())
17 BindEvent(#PB_Event>LoadingError, @LoadingError())
18
19 LoadImage(0, "Data/SpiderBasicLogo.png")

```

Example: with local file

See Also

[CreateImage\(\)](#) , [CopyImage\(\)](#) , [GrabImage\(\)](#) , [ExportImage\(\)](#)

75.15 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() .

See Also

ImageWidth(), ImageHeight()

75.16 ExportImage

Syntax

```
ExportImage(#Image, Filename$ [, ImagePlugin])
```

Description

Exports the specified image to the user through a download.

Parameters

#Image The image to export.

Filename\$ The filename to use. This is the name which will be shown to the user when the download starts.

ImagePlugin (optional) The format to save the image in. This can be one of the following values:

```
#PB_ImagePlugin_JPEG: export the image as JPEG.  
#PB_ImagePlugin_PNG : export the image as PNG.
```

Return value

None.

See Also

LoadImage()

Chapter 76

InAppPurchase

Overview

In-App purchase is a system supported by GooglePlay and iOS AppStore to buy items directly from inside an application. It is mostly used for games, but can be used for any kind of application to unlock special features against a fee.

Remarks

For Android, the GooglePlay public app API key needs to be specified in the create app dialog. It can be found in the google play console 'Service & API' -> 'Public key RSA for this app' (the key which looks like: MIIB.....AQAB).

76.1 RegisterAppProduct

Syntax

```
RegisterAppProduct(ProductID$, Callback [, Type])
```

Description

Register a new product for In-App purchase store. This product has to be existing in the app store. Once all needed products have been registered, FetchAppProducts() has to be called to contact the store and retrieve information about each products (like price, description etc.)

Parameters

ProductID\$ The unique product identifier.

Callback A callback which will be called when a state change occurs on this product. It has to respect the following declaration:

```
1 Procedure OnPurchase(State, ProductID$)
2   Select State
3     Case #PB_Product_Approved
4       ; A product has been successfully purchased. Do the logic
          and then call FinishAppProductPurchase() to finish the order
          process
```

```

5      Case #PB_Product_Owned
6          ; A non-consummable product has been already purchased. It
7          will trigger this at program start
8
9      Case #PB_Product_Cancelled
10         ; An order has been initialized, but cancelled by the user.
11         Mostly for statistical purpose
12
13      Case #PB_Product_Refunded
14         ; An order has been refunded, you should take appropriate
15         action to remove the previously purchased item
16
17      EndSelect
18  EndProcedure

```

State can be one of the following value:

```

#PB_Product_Approved : Product successfully purchased.
#PB_Product_Owned    : Product already owned (only for
                      non-consummable product).
#PB_Product_Cancelled: Product paiement cancelled (paiement was
                      initialized, but cancelled.
                      Mainly useful for statistical purpose.
#PB_Product_Refunded : Product paiement has been refunded.
                      Previously purchased item should removed from the user account.

```

Note: the same callback can be used for more than one product.

Type (optional) Type can be one of the following value:

```

#PB_Product_Consumable   : Product that can be purchased many
                           times (like rubies, gems, wings for a game).
#PB_Product_NonConsumable: Product that can be purchased only
                           once (like sword, helmet for a game).
                           If a non-consummable item has been
                           already purchased, the #PB_Product_Owned state
                           will be triggered at program start.

```

Return value

None.

See Also

[FetchAppProducts\(\)](#)

76.2 FetchAppProducts

Syntax

`FetchAppProducts(Callback)`

Description

Fetch information from the store about all previously registered products with RegisterAppProduct() .

Parameters

Callback A callback which will be called when the fetch will be finish. It has to respect the following declaration:

```
1 Procedure OnFetchAppProducts(Success)
2     ; Success will be #True if fetch has been successful, or #False
3     ; if it failed.
4 EndProcedure
```

Return value

None.

Example

```
1 Procedure OnFetchAppProducts(Success)
2     If Success
3         If ExamineAppProducts()
4             While NextAppProduct()
5                 Debug "id: " + AppProductId() + ", name: " + AppProductName()
6                 + ", price: " + AppProductPrice()
7                 Wend
8             Else
9                 Debug "ExamineAppProducts() failed."
10            EndIf
11        Else
12            Debug "FetchAppProducts() failed."
13        EndIf
14    EndProcedure
15
16 Procedure OnPurchaseAppProduct(State, ProductId$)
17     Debug "Product: " + ProductId$ + ", state : " + State
18 EndProcedure
19
20 RegisterAppProduct("gems", @OnPurchaseAppProduct()) ; can be
21     purchased more than once
22 RegisterAppProduct("helmet", @OnPurchaseAppProduct(),
23     #PB_Product_NonConsumable) ; can be purchased only once
24
25 FetchAppProducts(@OnFetchAppProducts())
```

See Also

RegisterAppProduct()

76.3 ExamineAppProducts

Syntax

```
Result = ExamineAppProducts()
```

Description

Start to examine the products previously fetch with FetchAppProducts() . To step trough all products, use NextAppProduct() .

Parameters

None.

Return value

Nonzero if the examine is successful, zero otherwise.

Example

```
1 Procedure OnFetchAppProducts(Success)
2   If Success
3     If ExamineAppProducts()
4       While NextAppProduct()
5         Debug "id: " + AppProductId() + ", name: " + AppProductName()
6         + ", price: " + AppProductPrice()
7         Wend
8       Else
9         Debug "ExamineAppProducts() failed."
10      EndIf
11    Else
12      Debug "FetchAppProducts() failed."
13    EndIf
14  EndProcedure
15
16 Procedure OnPurchaseAppProduct(State, ProductId$)
17   Debug "Product: " + ProductId$ + ", state : " + State
18 EndProcedure
19
20 RegisterAppProduct("gems", @OnPurchaseAppProduct()) ; can be
21 purchased more than once
22 RegisterAppProduct("helmet", @OnPurchaseAppProduct(),
#PB_Product_NonConsumable) ; can be purchased only once
23
24 FetchAppProducts(@OnFetchAppProducts())
```

See Also

NextAppProduct()

76.4 NextAppProduct

Syntax

```
Result = NextAppProduct()
```

Description

Go to the next product, if any. ExamineAppProducts() needs to be called before using this function. AppProductName() , AppProductDescription() , AppProductPrice() and AppProductID() can be used to get information about the current product.

Parameters

None.

Return value

Nonzero if there is a next product, zero otherwise (meaning it's the end of the product list).

Remarks

For a full code example, see ExamineAppProducts()

See Also

ExamineAppProducts() , AppProductName() , AppProductDescription() , AppProductPrice() , AppProductID()

76.5 AppProductName

Syntax

```
Result\$ = AppProductName()
```

Description

Get the name of the current product, listed with ExamineAppProducts() and NextAppProduct() .

Parameters

None.

Return value

Returns the name of the current product.

Remarks

For a full code example, see ExamineAppProducts()

See Also

AppProductDescription() , AppProductPrice() , AppProductID()

76.6 AppProductDescription

Syntax

```
Result\$ = AppProductDescription()
```

Description

Get the description of the current product, listed with ExamineAppProducts() and NextAppProduct() .

Parameters

None.

Return value

Returns the description of the current product.

Remarks

For a full code example, see ExamineAppProducts()

See Also

AppProductName() , AppProductPrice() , AppProductID()

76.7 AppProductPrice

Syntax

```
Result\$ = AppProductPrice()
```

Description

Get the price of the current product, listed with ExamineAppProducts() and NextAppProduct() .

Parameters

None.

Return value

Returns the price of the current product (including the currency).

Remarks

For a full code example, see ExamineAppProducts()

See Also

AppProductDescription() , AppProductName() , AppProductID()

76.8 AppProductID

Syntax

```
Result\$ = AppProductID()
```

Description

Get the identifier of the current product, listed with ExamineAppProducts() and NextAppProduct() .

Parameters

None.

Return value

Returns the identifier of the current product. This is the same identifier used when registering a product with RegisterAppProduct() .

Remarks

For a full code example, see ExamineAppProducts()

See Also

AppProductDescription() , AppProductName() , AppProductPrice()

76.9 PurchaseAppProduct

Syntax

```
PurchaseAppProduct( ProductID$ )
```

Description

Launch the purchase process of the specified product. A system window will be opened automatically with the product price and description asking for the user payment. The callback defined with RegisterAppProduct() will be called if the product state change.

Parameters

ProductID\$ The product identifier to use for the purchase.

Return value

None.

See Also

RegisterAppProduct() , AppProductID()

Chapter 77

Joystick

Overview

SpiderBasic provides a 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.

77.1 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 regular basis to detect if new joystick have been connected.

See Also

ExamineJoystick()

77.2 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, returns zero otherwise.

See Also

JoystickButton() , JoystickAxisX() , JoystickAxisY() , JoystickAxisZ() .

77.3 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 value:

```
#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()

77.4 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 value:

```
#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()

77.5 JoystickAxisZ

Syntax

```
Result = JoystickAxisZ(#Joystick [, Pad [, Mode]])
```

Description

Returns the joystick Z axis state. This axis is often referred as trigger on new gamepad.

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 value:

```
#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()

77.6 JoystickName

Syntax

```
Result\$ = JoystickName(#Joystick)
```

Description

Returns the joystick name. It can be useful when having several joystick 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()

77.7 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.

Return 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 78

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](#).

78.1 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   ; add element at the end
4   For i = 1 To 5
5     NumValue = AddJSONElement(ArrayValue)
6     SetJSONInteger(NumValue, i)
7   Next i
8
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\(\)](#)

78.2 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\(\)](#)

78.3 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\(\)](#)

78.4 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" +
2   Chr(34) + ": 20 }"
3 Debug Input$
4 ParseJSON(0, Input$)
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()

78.5 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

ParseJSON()

78.6 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) Not used.

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.

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

LoadJSON() , ParseJSON() , JSONValue() , FreeJSON()

78.7 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  If ExamineJSONMembers( ObjectValue )
8    While NextJSONMember( ObjectValue )
9      Debug JSONMemberKey( ObjectValue ) + " = " +
10     GetJSONInteger( JSONMemberValue( ObjectValue ) )
11   Wend
12 EndIf

```

See Also

[NextJSONMember\(\)](#) , [JSONMemberKey\(\)](#) , [JSONMemberValue\(\)](#) , [GetJSONMember\(\)](#) ,
[SetJSONObject\(\)](#) , [JSONType\(\)](#)

78.8 ExportJSON

Syntax

```
ExportJSON(#JSON, Filename$ [, Flags])
```

Description

Export the given JSON data to the user through a download. The JSON data will be encoded in UTF-8 format.

Parameters

#JSON The JSON to export.

Filename\$ The filename to use. This is the name which will be shown to the user when the download starts.

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

None.

See Also

[ComposeJSON\(\)](#)

78.9 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()

78.10 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\" +
2    Chr(34) + \": 20}, \" +
3      \" + Chr(34) + \"x\" + Chr(34) + \": 30, \" + Chr(34) + \"y\" +
4      Chr(34) + \": 50}, \" +
5        \" + Chr(34) + \"x\" + Chr(34) + \": -5, \" + Chr(34) + \"y\" +
6        Chr(34) + \": 100} ]"
7
8  Structure Location
9    x.l
10   y.l
11 EndStructure
12
13  NewList Locations.Location()
14
15  ParseJSON(0, Input$)
16  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\(\)](#)

78.11 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()

78.12 ExtractJSONStructure

Syntax

```
ExtractJSONStructure(JSONValue, *Buffer, Structure)
```

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.

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.

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.

Example

```
1 |     Structure Person
2 |         Name$ 
3 |         Age.1
4 |         List Books.s()
5 |     EndStructure
6 | 
```

```

7 | Input$ = "{" + Chr(34) + "Name" + Chr(34) + ": " + Chr(34) + "John
8 |   Smith" + Chr(34) + ", " +
9 |     Chr(34) + "Age" + Chr(34) + ": 42, " +
10|     Chr(34) + "Books" + Chr(34) + ": [" +
11|       Chr(34) + "Investing For Dummies" + Chr(34)
12|     + ", " +
13|       Chr(34) + "A Little Bit of Everything For
14|       Dummies" + Chr(34) + "] }"
15|
16| Debug P\Name$
17| Debug P\Age
18| Debug ListSize(P\Books())

```

See Also

[ExtractJSONArray\(\)](#) , [ExtractJSONList\(\)](#) , [ExtractJSONMap\(\)](#) , [InsertJSONArray\(\)](#) ,
[InsertJSONList\(\)](#) , [InsertJSONMap\(\)](#) , [InsertJSONStructure\(\)](#) , [SetJSONObject\(\)](#) , [JSONType\(\)](#)

78.13 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\(\)](#)

78.14 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()

78.15 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\(\)](#)

78.16 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()

78.17 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()

78.18 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()

78.19 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.

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), "y"))
```

See Also

SetJSONObject() , AddJSONMember() , RemoveJSONMember() , ClearJSONMembers() , ExamineJSONMembers() , JSONObjectSize() , JSONType()

78.20 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()

78.21 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()

78.22 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()` ,

78.23 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 contant 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()

78.24 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()

78.25 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 For
   Dummies"
15
16 If CreateJSON(0)
17   InsertJSONStructure(JSONValue(0), @P, Person)
18   Debug ComposeJSON(0, #PB_JSON_PrettyPrint)
19 EndIf
```

See Also

InsertJSONArray() , InsertJSONList() , InsertJSONMap() , ExtractJSONArray() , ExtractJSONList() ,
ExtractJSONMap() , ExtractJSONStructure()

78.26 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() , LoadJSON() , ParseJSON() , FreeJSON()

78.27 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()

78.28 JSONErrorLine

Syntax

```
Result = JSONErrorLine()
```

Description

Returns the line number within the JSON input of the last failed JSON parsing operation with ParseJSON() or LoadJSON() .

Parameters

None.

Return value

The line number (1-based) of the last JSON parser error.

See Also

`JSONExceptionPosition()` , `JSONExceptionMessage()` , `ParseJSON()` , `LoadJSON()`

78.29 JSONExceptionMessage

Syntax

```
Result\$ = JSONExceptionMessage()
```

Description

Returns a message describing the cause for the failure at the last JSON parsing operation with `ParseJSON()` 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 JSONExceptionMessage()
5  EndIf
```

See Also

`JSONExceptionLine()` , `JSONExceptionPosition()` , `ParseJSON()` , `LoadJSON()`

78.30 JSONExceptionPosition

Syntax

```
Result = JSONExceptionPosition()
```

Description

Returns the character position within the line of the last failed JSON parsing operation with ParseJSON() 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() , LoadJSON()

78.31 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()

78.32 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()

78.33 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()

78.34 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   "])")
16 For i = 0 To JSONArraySize(JSONValue(0)) - 1
17   Debug GetAnyValue(GetJSONElement(JSONValue(0), i))
18 Next i
```

See Also

`JSONValue()` , `SetJSONArray()` , `SetJSONBoolean()` , `SetJSONDouble()` , `SetJSONFloat()` , `SetJSONInteger()` , `SetJSONNull()` , `SetJSONObject()` , `SetJSONString()` , `SetJSONQuad()`

78.35 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()

78.36 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. It can be an URL or a local file identifier got with SelectedFileID() .

Flags (optional) It can be one of the following value:

```
#PB_LocalFile: the filename is a local file, OpenFileRequester()
() needs to be called before
                  to have access to local files. SelectedFileID()
() is used to get the
                  local file identifier.
```

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.

See Also

CreateJSON() , ParseJSON() , JSONValue() , FreeJSON() , JSONErrorMessage() , JSONErrorLine() ,
JSONErrorPosition()

78.37 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()

78.38 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) Not used.

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.

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    JSONErrorMessage()
7  EndIf
```

See Also

CreateJSON() , LoadJSON() , JSONValue() , FreeJSON() , JSONErrorMessage() , JSONErrorLine() ,
JSONExceptionPosition() , ExportJSON()

78.39 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()

78.40 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()

78.41 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\(\)](#)

78.42 SetJSONArray

Syntax

```
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

None.

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\(\)](#)

78.43 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()`

78.44 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()

78.45 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()

78.46 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   JSONArrayValue = 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()

78.47 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()

78.48 SetJSONObject

Syntax

```
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

None.

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()

78.49 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()

78.50 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" +
Chr(13) + Chr(10) + "line")
4     SetJSONString(AddJSONElement(ArrayValue), "with escaped \
backslash")
5
6     Debug ComposeJSON(0)
7 EndIf
```

See Also

[GetJSONString\(\)](#) , [SetJSONArray\(\)](#) , [SetJSONBoolean\(\)](#) , [SetJSONDouble\(\)](#) , [SetJSONFloat\(\)](#) ,
[SetJSONInteger\(\)](#) , [SetJSONNull\(\)](#) , [SetJSONObject\(\)](#) , [SetJSONQuad\(\)](#)

Chapter 79

Keyboard

Overview

SpiderBasic 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.

79.1 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.

79.2 ExamineKeyboard

Syntax

```
Result = ExamineKeyboard()
```

Description

Updates the keyboard state. This function has to be called before using KeyboardInkey() , KeyboardPushed() or KeyboardReleased() .

Parameters

None.

Return value

None.

See Also

KeyboardInkey() , KeyboardPushed() KeyboardReleased() .

79.3 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 OpenScreen(800, 600, 32, "Test")
2
3 Procedure RenderFrame()
4     Static x, y, FullText$
5
6     ClearScreen(RGB(0, 0, 0))
7
8     If ExamineKeyboard()
9         ; If we press the 'Back' key, we will delete the last character
```

```

10   ;
11   If KeyboardReleased(#PB_Key_Back)
12     FullText$ = Left(FullText$, Len(FullText$)-1)
13   Else
14     Result$ = KeyboardInkey()
15     If FindString("1234567890
'ABCDEFGHIJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz", Result$) ;
Or your chosen valid characters
16       FullText$ + Result$
17       Debug FullText$
18     EndIf ; Add the new text to the current one (if any)
19   EndIf
20 EndIf
21
22   FlipBuffers(); // continue the rendering
23 EndProcedure
24
25 Procedure Loading(Type, Filename$, ObjectId)
26   Static NbLoadedElements
27
28   NbLoadedElements+1
29   If NbLoadedElements = 1 ; The loading of all images and sounds is
finished, we can start the rendering
30     FlipBuffers(); // start the rendering
31   EndIf
32 EndProcedure
33
34 Procedure LoadingError(Type, Filename$, ObjectId)
35   Debug Filename$ + ": loading error"
36 EndProcedure
37
38 ; Register the loading event before calling any resource load command
39 BindEvent(#PB_Event>Loading, @Loading())
40 BindEvent(#PB_Event>LoadingError, @LoadingError())
41 BindEvent(#PB_Event>RenderFrame, @RenderFrame())
42
43 LoadSprite(0, "Data/SpiderBasicLogo.png")

```

See Also

[ExamineKeyboard\(\)](#)

79.4 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. 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  OpenScreen(800, 600, 32, "Test")
2
3  Procedure RenderFrame()
4      Static x, y
5
6      ClearScreen(RGB(0, 0, 0))
7
8      If ExamineKeyboard()
9          If KeyboardPushed(#PB_Key_Left)
10             x-2
11         ElseIf KeyboardPushed(#PB_Key_Right)
12             x+2
13         EndIf
14
15         If KeyboardPushed(#PB_Key_Up)
16             y-2
17         ElseIf KeyboardPushed(#PB_Key_Down)
18             y+2
19         EndIf
20
21         DisplaySprite(0, x, y)
22     EndIf
23
24     FlipBuffers(); // continue the rendering
25 EndProcedure
26
27 Procedure Loading(Type, Filename$, ObjectId)
28     Static NbLoadedElements
29
30     NbLoadedElements+1
31     If NbLoadedElements = 1 ; The loading of all images and sounds is
32         finished, we can start the rendering
33         FlipBuffers(); // start the rendering
34     EndIf
35 EndProcedure
36
37 Procedure LoadingError(Type, Filename$, ObjectId)
38     Debug Filename$ + ": loading error"
39 EndProcedure
40
41 ; Register the loading event before calling any resource load command
42 BindEvent(#PB_Event_Loading, @Loading())
43 BindEvent(#PB_Event>LoadingError, @LoadingError())
44 BindEvent(#PB_Event_RenderFrame, @RenderFrame())
45 LoadSprite(0, "Data/SpiderBasicLogo.png")
```

See Also

ExamineKeyboard() , KeyboardReleased()

79.5 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.

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 Debug "The sprite will only move when the key are released"
2
3 OpenScreen(800, 600, 32, "Test")
4
5 Procedure RenderFrame()
6     Static x, y
7
8     ClearScreen(RGB(0, 0, 0))
9
10    If ExamineKeyboard()
11        If KeyboardReleased(#PB_Key_Left)
12            x-10
13        ElseIf KeyboardReleased(#PB_Key_Right)
14            x+10
15        EndIf
16
17        If KeyboardReleased(#PB_Key_Up)
18            y-10
19        ElseIf KeyboardReleased(#PB_Key_Down)
20            y+10
21        EndIf
22
23        DisplaySprite(0, x, y)
24    EndIf
25
26    FlipBuffers(); // continue the rendering
27 EndProcedure
28
29 Procedure Loading(Type, Filename$, ObjectId)
30     Static NbLoadedElements
31
32     NbLoadedElements+1
```

```

33 |     If NbLoadedElements = 1 ; The loading of all images and sounds is
34 |         finished, we can start the rendering
35 |             FlipBuffers(); // start the rendering
36 |         EndIf
37 |     EndProcedure
38 |
39 | Procedure LoadingError(Type, Filename$, ObjectId)
40 |     Debug Filename$ + ": loading error"
41 | EndProcedure
42 |
43 | ; Register the loading event before calling any resource load command
44 | BindEvent(#PB_Event_Loading, @Loading())
45 | BindEvent(#PB_Event>LoadingError, @LoadingError())
46 | BindEvent(#PB_Event_RenderFrame, @RenderFrame())
47 | LoadSprite(0, "Data/SpiderBasicLogo.png")

```

See Also

[ExamineKeyboard\(\)](#) , [KeyboardPushed\(\)](#)

Chapter 80

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() and can 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 .

Other possibilities for storing data are the use of Arrays and Maps .

80.1 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 brackets 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
32 Debug Programmers()\Name
33 Debug Programmers()\Strength
34 Else
35     MessageRequester("Error!", "Unable to allocate memory for new
        element", #PB_MessageRequester_OK)
36 EndIf
```

See Also

InsertElement() , DeleteElement() , ClearList()

80.2 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 brackets 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

```
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)
```

See Also

SelectElement()

80.3 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 brackets after the list name.

Return value

None.

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
11   "+Str(ListSize(Numbers()))+" elements in the list",
12   #PB_MessageRequester_OK)
13
14  ; Clear the list and show that the list really is empty
15  ClearList(Numbers())
16  MessageRequester("Information", "There are
17   "+Str(ListSize(Numbers()))+" elements in the list",
18   #PB_MessageRequester_OK)
```

See Also

DeleteElement() , FreeList()

80.4 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\(\)](#)

80.5 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\(\)](#)

80.6 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 brackets after the list name.

Return value

The total number of elements in the list.

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\(\)](#)

80.7 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 brackets 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(),
11   #PB_MessageRequester_Ok)
12
13 LastElement(people()); Moves to "Bob"
14 PreviousElement(people()); Moves to "Harry"
15 DeleteElement(people()); And deletes him. there is an element
16   before Harry, so it becomes the current
17 MessageRequester("Information", "Current person in list is
18   "+people(), #PB_MessageRequester_Ok)
```

See Also

AddElement() , InsertElement() , ClearList()

80.8 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 brackets 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
11 "+Str(Numbers()), #PB_MessageRequester_OK)
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
18   "+Str(Numbers()), #PB_MessageRequester_OK)
19 Else
20   MessageRequester("Information", "List is empty",
21   #PB_MessageRequester_OK)
22 EndIf
23
24 AddElement(Numbers())
25 Numbers() = 5
26 AddElement(Numbers())
27 Numbers() = 8
28
29 If FirstElement(Numbers()) <> 0
30   MessageRequester("Information", "First element value is
31   "+Str(Numbers()), #PB_MessageRequester_OK)
32 Else
33   MessageRequester("Information", "List is empty",
34   #PB_MessageRequester_OK)
35 EndIf
36
37 ; 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)
EndIf

```

See Also

[LastElement\(\)](#) , [PreviousElement\(\)](#) , [NextElement\(\)](#) , [SelectElement\(\)](#) , [ListIndex\(\)](#)

80.9 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 brackets 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 list
4 simple() = 23
5
6 InsertElement(simple())      ; Current position is the first element,
    so we add this element to the start of the list

```

```

7 | simple() = 45           ; The old first element is now the second
8 |   element in the list
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
16|   element", #PB_MessageRequester_OK)
17| EndIf
18|
19| ; A small structure to demonstrate the use of the pointer to the new
20|   element
21| Structure Programmer
22|   Name.s
23|   Strength.b
24| EndStructure
25|
26| NewList Programmers.Programmer(); The list for storing the elements
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
33|   element", #PB_MessageRequester_OK)

```

See Also

AddElement() , DeleteElement() , ClearList()

80.10 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 brackets after the list name.

Return value

Returns the address of the data in the last 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 LastElement(Numbers())
10 MessageRequester("Information", "Last element value is
11 "+Str(Numbers()), #PB_MessageRequester_OK)
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
18   "+Str(Numbers()), #PB_MessageRequester_OK)
19 Else
20   MessageRequester("Information", "List is empty",
21   #PB_MessageRequester_OK)
22 EndIf
23
24 AddElement(Numbers())
25 Numbers() = 5
26 AddElement(Numbers())
27 Numbers() = 8
28
29 If LastElement(Numbers()) <> 0
30   MessageRequester("Information", "Last element value is
31   "+Str(Numbers()), #PB_MessageRequester_OK)
32 Else
33   MessageRequester("Information", "List is empty",
34   #PB_MessageRequester_OK)
35 EndIf
36
37
38 ; An example which is only for advanced users
39 NewList Numbers.w()
40
41 AddElement(Numbers())
42 Numbers() = 5
43 AddElement(Numbers())
44 Numbers() = 8
45
46 *Element.Word = LastElement(Numbers())
47 If *Element
48   MessageRequester("Information", "Last element value is
49   "+Str(*Element\w), #PB_MessageRequester_OK)
50 Else
```

```

46     MessageRequester("Information", "List is empty",
47     #PB_MessageRequester_OK)
47 EndIf

```

See Also

[FirstElement\(\)](#) , [PreviousElement\(\)](#) , [NextElement\(\)](#) , [SelectElement\(\)](#) , [ListIndex\(\)](#)

80.11 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 brackets 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
16 |         "+Str(ListIndex(fruit())), #PB_MessageRequester_OK)
17 |
18 |     NextElement(fruit())
19 |     MessageRequester("Fruit: "+fruit(), "Now at position
20 |         "+Str(ListIndex(fruit())), #PB_MessageRequester_OK)

```

See Also

SelectElement() , ListSize()

80.12 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 brackets 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
11 ; current element to the first item in the list
12     MessageRequester("Score", Str(Scores()), #PB_MessageRequester_OK)
13 Wend

```

See Also

ResetList() , PreviousElement() , FirstElement() , LastElement() , SelectElement() , ListIndex()

80.13 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 brackets 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()), #PB_MessageRequester_OK)
10     Until PreviousElement(Numbers()) = 0
```

See Also

NextElement() , FirstElement() , LastElement() , SelectElement() , ListIndex()

80.14 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 brackets 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()

80.15 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 using an own loop for this.

Parameters

List() The name of your list variable, created with the NewList function. You must include the brackets 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.1()
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()

80.16 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 brackets 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()

80.17 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 brackets 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\(\)](#)

80.18 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 structure, 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 brackets 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.

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](#)

80.19 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 brackets 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

80.20 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_First : Insert the elements at the beginning of  
DestinationList()  
#PB_List_Last : Append the elements at the end 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\(\)](#)

80.21 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 81

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 are 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, using 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 ”.

81.1 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 State"
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\(\)](#)

81.2 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.

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 Debug "There is " + MapSize(Country()) + " elements in the map"
8
9 Debug "Clearing the map..."
10
11 ; Clear the map and show that the map really is empty
12 ClearMap(Country())
13 Debug "There is " + MapSize(Country()) + " element in the map"
```

See Also

AddMapElement() , DeleteMapElement()

81.3 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\(\)](#)

81.4 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\(\)](#)

81.5 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.

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: " + MapSize(Country())
```

See Also

[MapSize\(\)](#)

81.6 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.

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()`

81.7 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 new 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()

81.8 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()

81.9 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()

81.10 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()

81.11 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

81.12 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 82

Math

Overview

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.

82.1 Abs

Syntax

```
Result.f = Abs(Number.f)
```

Description

Returns the absolute value of the given float value.

Parameters

Number.f 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 can handle float and double values.

Example

```
1 Debug Abs(3.14159) ; Will display '3.14159'  
2 Debug Abs(-3.14159) ; will display '3.14159'
```

See Also

Sign()

82.2 ACos

Syntax

```
Result.f = ACos(Value.f)
```

Description

Returns the arc-cosine of the specified value.

Parameters

Value.f The input value. It must be between -1 and 1 (-1 and 1 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 can handle float and double values.

Example

```
1 Debug ACos(1) ; will display '0'  
2 Debug ACos(-1) ; will display 'pi' (approximately 3.14159)
```

See Also

Cos() , ACosH() , Degree()

82.3 ACosH

Syntax

```
Result.f = ACosH(Value.f)
```

Description

Returns the area hyperbolic cosine of the specified value.

Parameters

Value.f The input value. It must be between greater than or equal to 1.

Return value

Returns the hyperbolic angle.

Remarks

This is the inverse function of CosH() . This function can handle float and double values.

Example

```
1 Debug ACosh(1) ; will display '0'  
2 Debug Exp(ACosh(0.5 * Sqr(5))) ; will display '1.618033' (the golden ratio)
```

See Also

CosH() , @acos

82.4 ASin

Syntax

```
Result.f = ASin(Value.f)
```

Description

Returns the arc-sine of the specified value.

Parameters

Value.f The input value. It must be between -1 and 1 (-1 and 1 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 can handle float and double values.

Example

```
1 Debug ASin(1) ; will display '1.570796' (pi/2)
2 Debug ASin(0) ; will display '0'
```

See Also

Sin() , ASinH() , Degree()

82.5 ASinH

Syntax

```
Result.f = ASinH(Value.f)
```

Description

Returns the area hyperbolic sine of the specified value.

Parameters

Value.f The input value. The range of Value.f is not limited.

Return value

Returns the hyperbolic angle.

Remarks

This is the inverse function of SinH() . This function can handle float and double values.

Example

```
1 Debug ASinH(0) ; will display '0'
2 Debug Exp(ASinH(0.5)) ; will display '1.618033' (the golden ratio)
```

See Also

Sinh() , ASin()

82.6 ATan

Syntax

```
Result.f = ATan(Value.f)
```

Description

Returns the arc-tangent of the specified value.

Parameters

Value.f 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 can handle float and double values.

Example

```
1 Debug ATan(1) ; will display '0.785398' (pi/4)
```

See Also

Tan() , ATanH() , Degree()

82.7 ATan2

Syntax

```
Result.f = ATan2(x.f, y.f)
```

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 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 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. This function can handle float and double values.

Example

```
1 Debug ATan2(10, 10) ; will display #PI/4 (45 degrees in radian)
```

See Also

ATan() , Degree()

82.8 ATanH

Syntax

```
Result.f = ATanH(Value.f)
```

Description

Returns the area hyperbolic tangent of the specified value.

Parameters

Value.f The input value. It must be between -1 and 1 (not including -1 and 1).

Return value

Returns the hyperbolic angle.

Remarks

This is the inverse function of TanH() . This function can handle float and double values.

Example

```
1 Debug Exp(ATanH(0.2 * Sqr(5))) ; will display '1.618033' (the golden ratio)
```

See Also

TanH() , ATan()

82.9 Cos

Syntax

```
Result.f = Cos(Angle.f)
```

Description

Returns the cosine of the specified angle.

Parameters

Angle.f 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 and 1.

Remarks

The inverse function of Cos() is ACos() . This function can handle float and double values.

Example

```
1 Debug Cos(3.141593) ; will display '-1'
```

See Also

ACos() , Cosh() , Radian()

82.10 Cosh

Syntax

```
Result.f = Cosh(Angle.f)
```

Description

Returns the hyperbolic cosine of the specified hyperbolic angle.

Parameters

Angle.f 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 can handle float and double values.

Example

```
1 Debug CosH(0) ; will display 1
```

See Also

ACosH() , Cos()

82.11 Degree

Syntax

```
Result.f = Degree(Angle.f)
```

Description

Converts the given angle from radian to degree.

Parameters

Angle.f 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 can handle float and double values.

Example

```
1 Debug Degree(#PI/4) ; will display 45
```

See Also

`Radian()`

82.12 Exp

Syntax

```
Result.f = Exp(Number.f)
```

Description

Returns the result of the exponential function. This is the value e raised to the power 'Number'.

Parameters

`Number.f` 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 can handle float and double values.

See Also

`Log()`

82.13 Infinity

Syntax

```
Result.f = 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
```

See Also

IsInfinity() , NaN()

82.14 Int

Syntax

```
Result = Int(Number.f)
```

Description

Returns the integer part of a float number.

Parameters

Number.f 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()

82.15 IntQ

Syntax

```
Result = IntQ(Number.f)
```

Description

Returns the integer part of a float number as a quad.

Parameters

Number.f 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()

82.16 IsInfinity

Syntax

```
Result.f = IsInfinity(Value.f)
```

Description

Returns nonzero if the given value represents positive or negative infinity.

Parameters

Value.f 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 can handle float and double values.

Example

```
1 Debug IsInfinity(Infinity()) ; infinity
2 Debug IsInfinity(Log(0)) ; -infinity
3 Debug IsInfinity(1234.5) ; a finite number
4 Debug IsInfinity(NaN()) ; NaN is not the same as infinity
```

See Also

Infinity() , isNaN()

82.17 IsNaN

Syntax

```
Result.f = IsNaN(Value.f)
```

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 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 can handle float and 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()

82.18 Pow

Syntax

```
Result.f = Pow(Number.f, Power.f)
```

Description

Returns the given number, raised to the given power.

Parameters

Number.f The mantissa.

Power.f The exponent. If 'Number.f' is negative, the exponent must be a whole number.

Return value

Returns the value of 'Number' raised to the power 'Power'.

Remarks

This function can handle float and 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'
```

See Also

Sqr()

82.19 Log

Syntax

```
Result.f = Log(Number.f)
```

Description

Returns the natural Log (ie log to the base e) of the given number.

Parameters

Number.f 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 can handle float and double values.

See Also

Exp() , Log10()

82.20 Log10

Syntax

```
Result.f = Log10(Number.f)
```

Description

Returns the log in base 10 of the given number.

Parameters

Number.f The number to calculate the log from. This must be a positive value.

Return value

Returns the log in base 10.

Remarks

This function can handle float and double values.

See Also

Log()

82.21 Mod

Syntax

```
Result.f = Mod(Number.f, Divisor.f)
```

Description

Returns the remainder of the division of Number.f by Divisor.f.

Parameters

Number.f The number to divide. The input does not need to be a whole number.

Divisor.f 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 parameter.

Remarks

This is the floating-point version of the '%' operator for integers. This function can handle float and double values.

82.22 NaN

Syntax

```
Result.f = 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
```

See Also

IsNaN() , Infinity()

82.23 Radian

Syntax

```
Result.f = Radian(Angle.f)
```

Description

Converts the given angle from degrees into radian.

Parameters

Angle.f 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 can handle float and double values.

Example

```
1 Debug Radian(90) ; will display #PI/2
```

See Also

Degree()

82.24 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. 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.

Example

```
1 Repeat
2   Result = Random(6, 1) ; get a value between 1 and 6, including 1
3   and 6
4   Debug Result
5 Until Result = 6 ; Exit when 6 is found
```

See Also

RandomSeed() , RandomizeArray() , RandomizeList()

82.25 RandomSeed

Syntax

```
RandomSeed(Value)
```

Description

Changes the random number seed for the values returned with Random() .

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() , RandomizeArray() , RandomizeList()

82.26 Round

Syntax

```
Result.f = Round(Number.f, Mode)
```

Description

Round the specified float number according to the given mode.

Parameters

Number.f 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 can handle float and double values.

Example

```
1  Debug Round(11.6, #PB_Round_Down)      ; Will print '11'  
2  Debug Round(-3.6, #PB_Round_Down)      ; Will print '-4'  
3  
4  Debug Round(11.6, #PB_Round_Up)        ; Will print '12'  
5  Debug Round(-1.6, #PB_Round_Up)        ; Will print '-1'  
6  
7  Debug Round(11.6, #PB_Round_Nearest); Will print '12'  
8  Debug Round(11.4, #PB_Round_Nearest); Will print '11'
```

```
9 | Debug Round(11.5, #PB_Round_Nearest) ; Will print '12'
10 | Debug Round(-7.5, #PB_Round_Nearest) ; Will print '-8'
```

See Also

[Int\(\)](#) , [IntQ\(\)](#)

82.27 Sign

Syntax

```
Result.f = Sign(Number.f)
```

Description

Returns a floating-point value representing the sign of the given number.

Parameters

Number.f 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 can handle float and double values.

Example

```
1 | Debug Sign(-7)      ; will display -1.0
2 | Debug Sign(0)       ; will display 0.0
3 | Debug Sign(7)       ; will display 1.0
```

See Also

[Abs\(\)](#)

82.28 Sin

Syntax

```
Result.f = Sin(Angle.f)
```

Description

Returns the sine of the specified angle.

Parameters

Angle.f 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 and 1.

Remarks

The inverse function of Sin() is ASin() . This function can handle float and double values.

Example

```
1 Debug Sin(1.5708) ; will display '1'
```

See Also

ASin() , SinH() , Radian()

82.29 SinH

Syntax

```
Result.f = SinH(Angle.f)
```

Description

Returns the hyperbolic sine of the specified hyperbolic angle.

Parameters

Angle.f The input angle.

Return value

Returns the hyperbolic sine of the angle.

Remarks

The inverse function of SinH() is ASinH() . This function can handle float and double values.

Example

```
1 Debug SinH(Log(1.618033)) ; Will be approximately 0.5
```

See Also

ASinH() , Sin()

82.30 Sqr

Syntax

```
Result.f = Sqr(Number.f)
```

Description

Returns the square root of the specified number.

Parameters

Number.f The input number. This must be a positive value.

Return value

Returns the square root of the number.

Remarks

This function can handle float and double values.

See Also

Pow()

82.31 Tan

Syntax

```
Result.f = Tan(Angle.f)
```

Description

Returns the tangent of the specified angle.

Parameters

Angle.f 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 can handle float and double values.

Example

```
1 Debug Tan(0.785398) ; will display '1'
```

See Also

ATan() , ATan2() , TanH() , Radian()

82.32 TanH

Syntax

```
Result.f = TanH(Angle.f)
```

Description

Returns the hyperbolic tangent of the specified hyperbolic angle.

Parameters

Angle.f The input angle.

Return value

Returns the hyperbolic tangent of the angle.

Remarks

The inverse function of TanH() is ATanH() . This function can handle float and double values.

Example

```
1 Debug TanH(Log(1.618033)) ; will display '0.447213' (1/5 * Sqr(5))
```

See Also

ATanH() , Tan()

Chapter 83

Memory

Overview

Sometimes, it's useful have dynamically allocated block to deal with unknown number of data. SpiderBasic already provides built-in list and map for this, but this library allows for more flexibility.

83.1 AllocateMemory

Syntax

```
*Buffer = 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. All the allocated memory areas are automatically freed when the programs ends.

Example

```
1 *Buffer = AllocateMemory(5000)
2 If *Buffer
3   PokeS(*Buffer, 0, "Store this string in the memory buffer")
4
5   ; Read it back
6   Debug PeekS(*Buffer, 0)
7
8   FreeMemory(*Buffer) ; will also be done automatically at the end
9   of program
10 Else
11   Debug "Couldn't allocate the requested memory!"
12 EndIf
```

See Also

FreeMemory() , MemorySize()

83.2 AllocateStructure

Syntax

```
*Item.StructureName = AllocateStructure(StructureName)
```

Description

Allocates a new dynamic structure item. This dynamic structure item is properly initialized and ready to use. 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 of 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.

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\(\)](#)

83.3 CompareMemory

Syntax

```
Result = CompareMemory(*Buffer1, Offset1, *Buffer2, Offset2, Size)
```

Description

Compares the content of two memory buffers.

Parameters

***Buffer1** The addresses of the first memory buffer to compare. This buffer must have been allocated with [AllocateMemory\(\)](#).

Offset1 The '`*Buffer1`' offset (in bytes) to compare.

***Buffer2** The addresses of the second memory buffer to compare. This buffer must have been allocated with [AllocateMemory\(\)](#).

Offset2 The '`*Buffer2`' offset (in bytes) 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\(\)](#) , [MemorySize\(\)](#)

83.4 FreeMemory

Syntax

```
FreeMemory (*Buffer)
```

Description

Free the memory previously allocated with AllocateMemory() or ReAllocateMemory() .

Parameters

***Buffer** The address of the memory area to free. This must be a value returned from either AllocateMemory() or ReAllocateMemory() .

Return value

None.

Remarks

All remaining allocated memory blocks are automatically freed when the program ends.

See Also

AllocateMemory() , ReAllocateMemory()

83.5 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

All remaining allocated dynamic structure items are automatically freed when the program ends.

See Also

AllocateStructure()

83.6 MemorySize

Syntax

```
Result = MemorySize(*Buffer)
```

Description

Returns the size of the given memory buffer.

Parameters

***Buffer** The address of the memory area to get the size from. This must be a value returned from either AllocateMemory() or ReAllocateMemory() .

Return value

Returns the size of the given memory area in bytes.

See Also

AllocateMemory() , ReAllocateMemory() , FreeMemory()

83.7 ReAllocateMemory

Syntax

```
*Buffer = ReAllocateMemory(*Buffer, 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

***Buffer** 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 '`*Buffer`' 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 '`*Buffer`' 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.

All remaining allocated memory blocks are automatically freed when the program ends.

Example

```
1  *Buffer = AllocateMemory(1000)
2  PokeS(*Buffer, 0, "Store this string")
3
4  Debug "Buffer size: " + MemorySize(*Buffer)
5
6  *NewBuffer = ReAllocateMemory(*Buffer, 2000) ; need more memory
7  If *NewBuffer
8      Debug "New buffer size: " + MemorySize(*NewBuffer)
9
10     Debug "The old content is still here:"
11     Debug PeekS(*NewBuffer, 0)
12
13     FreeMemory(*NewBuffer)
14 EndIf
```

See Also

`AllocateMemory()` , `FreeMemory()` , `MemorySize()`

83.8 PeekA

Syntax

```
Value.a = PeekA(*Buffer, Offset)
```

Description

Reads an ascii character (1 byte) from the specified memory address.

Parameters

***Buffer** The address to read from.

Offset The offset (in bytes) in the buffer.

Return value

Returns the value of the ascii character.

See Also

PokeA()

83.9 PeekB

Syntax

```
Value.b = PeekB(*Buffer, Offset)
```

Description

Reads a byte (1 byte) number from the specified memory address.

Parameters

***Buffer** The address to read from.

Offset The offset (in bytes) in the buffer.

Return value

Returns the value of the byte.

See Also

PokeB()

83.10 PeekC

Syntax

```
Value.c = PeekC(*Buffer, Offset)
```

Description

Reads a character (UTF-16) from the specified memory address.

Parameters

***Buffer** The address to read from.

Offset The offset (in bytes) in the buffer.

Return value

Returns the value of the character.

See Also

PokeC()

83.11 PeekD

Syntax

```
Value.d = PeekD(*Buffer, Offset)
```

Description

Reads a double (8 bytes) number from the specified memory address.

Parameters

***Buffer** The address to read from.

Offset The offset (in bytes) in the buffer.

Return value

Returns the value of the double.

See Also

PokeD()

83.12 PeekL

Syntax

```
Value.l = PeekL(*Buffer, Offset)
```

Description

Reads a long (4 bytes) number from the specified memory address.

Parameters

***Buffer** The address to read from.

Offset The offset (in bytes) in the buffer.

Return value

Returns the value of the long.

See Also

PokeL()

83.13 PeekW

Syntax

```
Value.w = PeekW(*Buffer, Offset)
```

Description

Reads a word (2 bytes) number from the specified memory address.

Parameters

***Buffer** The address to read from.

Offset The offset (in bytes) in the buffer.

Return value

Returns the value of the word.

See Also

PokeW()

83.14 PeekF

Syntax

```
Value.f = PeekF(*Buffer, Offset)
```

Description

Reads a float (4 bytes) from the specified memory address.

Parameters

***Buffer** The address to read from.

Offset The offset (in bytes) in the buffer.

Return value

Returns the value of the float.

See Also

PokeF()

83.15 PeekS

Syntax

```
Text\$ = PeekS(*Buffer [, Length [, Format]])
```

Description

Reads a string from the specified memory address.

Parameters

***Buffer** The address to read from.

Offset The offset (in bytes) in the buffer.

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 (default).  
#PB_Unicode: Reads the strings as unicode.
```

Return value

Returns the read string.

See Also

PokeS()

83.16 PeekU

Syntax

```
Value.u = PeekU(*Buffer, Offset)
```

Description

Reads an unicode character (2 bytes) from the specified memory address.

Parameters

***Buffer** The address to read from.

Offset The offset (in bytes) in the buffer.

Return value

Returns the value of the unicode character.

See Also

PokeU()

83.17 PokeA

Syntax

```
PokeA(*Buffer, Number)
```

Description

Writes an ascii character (1 byte) to the specified memory address.

Parameters

***Buffer** The address to write to.

Offset The offset (in bytes) in the buffer.

Number The value to write.

Return value

None.

See Also

[PeekA\(\)](#)

83.18 PokeB

Syntax

```
PokeB(*Buffer, Offset, Number)
```

Description

Writes a byte (1 byte) number to the specified memory address.

Parameters

***Buffer** The address to write to.

Offset The offset (in bytes) in the buffer.

Number The value to write.

Return value

None.

See Also

[PeekB\(\)](#)

83.19 PokeC

Syntax

```
PokeC(*Buffer, Offset, Number)
```

Description

Writes a character (UTF-16) to the specified memory address.

Parameters

***Buffer** The address to write to.

Offset The offset (in bytes) in the buffer.

Number The character value to write.

Return value

None.

See Also

[PeekC\(\)](#)

83.20 PokeD

Syntax

```
PokeD(*Buffer, Offset, Number)
```

Description

Writes a double (8 bytes) number to the specified memory address.

Parameters

***Buffer** The address to write to.

Offset The offset (in bytes) in the buffer.

Number The value to write.

Return value

None.

See Also

[PeekD\(\)](#)

83.21 PokeL

Syntax

```
PokeL(*Buffer, Offset, Number)
```

Description

Writes a long (4 bytes) number to the specified memory address.

Parameters

***Buffer** The address to write to.

Offset The offset (in bytes) in the buffer.

Number The value to write.

Return value

None.

See Also

PeekL()

83.22 PokeW

Syntax

```
PokeW(*Buffer, Offset, Number)
```

Description

Writes a word (2 bytes) number to the specified memory address.

Parameters

***Buffer** The address to write to.

Offset The offset (in bytes) in the buffer.

Number The value to write.

Return value

None.

See Also

PeekW()

83.23 PokeF

Syntax

```
PokeF(*Buffer, Offset, Number.f)
```

Description

Writes a float (4 bytes) to the specified memory address.

Parameters

***Buffer** The address to write to.

Offset The offset (in bytes) in the buffer.

Number The value to write.

Return value

None.

See Also

PeekF()

83.24 PokeS

Syntax

```
Result = PokeS(*Buffer, Offset, Text$, [Length [, Flags]])
```

Description

Writes a string to the specified memory address, followed by a null-character for termination.

Parameters

***Buffer** The address to write to.

Offset The offset (in bytes) in the buffer.

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 (default).  
#PB_Unicode : Writes the strings in unicode.
```

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()`

83.25 PokeU

Syntax

```
PokeU(*Buffer, Offset, Number)
```

Description

Writes an unicode character (2 bytes) to the specified memory address.

Parameters

***Buffer** The address to write to.

Offset The offset (in bytes) in the buffer.

Number The value to write.

Return value

None.

See Also

`PeekU()`

Chapter 84

Menu

Overview

Creating and managing menus in SpiderBasic is easy. 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.

84.1 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()

84.2 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

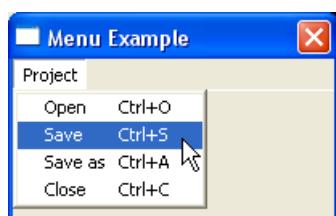
To create a menu with support for images, use CreateImageMenu() .

Once created, this menu becomes the current menu for further item additions. It's now possible to use functions such as MenuTitle() , MenuItem() , MenuBar() , OpenSubMenu() to populate the menu.

To handle menu events, use BindEvent() , EventWindow() and EventMenu() .

Example

```
1 Procedure MenuEvents()
2   Debug "Menu item selected: " + EventMenu()
3 EndProcedure
4
5 If OpenWindow(0, 200, 200, 200, 100, "Menu Example")
6   If CreateMenu(0, WindowID(0))      ; menu creation starts....
7     MenuTitle("Project")
8     MenuItem(1, "Open"    + #TAB$ + "Shift+O")
9     MenuItem(2, "Save"    + #TAB$ + "Shift+S")
10    MenuItem(3, "Save as" + #TAB$ + "Shift+A")
11    MenuItem(4, "Close"   + #TAB$ + "Shift+C")
12  EndIf
13
14  BindEvent(#PB_Event_Menu, @MenuEvents())
15 EndIf
```



See Also

CreateImageMenu() , CreatePopupMenu() , CreatePopupImageMenu() , FreeMenu() , MenuTitle() , MenuItem() , MenuBar() , OpenSubMenu()

84.3 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) Not used.

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, use BindEvent() , EventWindow() and EventMenu() .

Example

```
1 Procedure MenuEvents()
2   Debug "Menu item selected: " + EventMenu()
3 EndProcedure
4
5 CreateImage(0, 16, 16)
6 If StartDrawing(ImageOutput(0))
7   Box(0, 0, 16, 16, RGB(255, 0, 0)) ; red box
8   StopDrawing()
9 EndIf
10
11
12 If OpenWindow(0, 200, 200, 200, 100, "Menu Example")
13   If CreateMenu(0, WindowID(0))      ; menu creation starts...
14     MenuTitle("Project")
15     MenuItem(1, "Open", ImageID(0))
```

```

16     MenuItem(2, "Save")
17     MenuItem(3, "Save as")
18     MenuItem(4, "Close", ImageID(0))
19 EndIf
20
21 BindEvent(#PB_Event_Menu, @MenuEvents())
22 EndIf

```

See Also

[CreateMenu\(\)](#) , [CreatePopupMenu\(\)](#) , [CreatePopupImageMenu\(\)](#) , [FreeMenu\(\)](#) , [MenuTitle\(\)](#) , [MenuItem\(\)](#) , [MenuBar\(\)](#) , [OpenSubMenu\(\)](#)

84.4 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 [CreatePopupImageMenu\(\)](#) . 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, use [BindEvent\(\)](#) , [EventWindow\(\)](#) and [EventMenu\(\)](#) .

Example

```

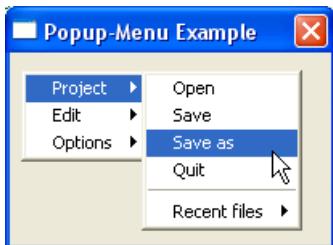
1 If CreatePopupMenu(0)
2   MenuItem(1, "Cut")
3   MenuItem(2, "Copy")
4   MenuItem(3, "Paste")
5   MenuBar()
6   OpenSubMenu("Options")

```

```

7     MenuItem(4, "Window...")
8     MenuItem(5, "Gadget...")
9     CloseSubMenu()
10    MenuBar()
11    MenuItem( 6, "Quit")
12  EndIf
13
14  Procedure GadgetEvents()
15    If EventGadget() = 0 And EventType() = #PB_EventType_RightClick
16      DisplayPopupMenu(0, WindowID(0))
17    EndIf
18  EndProcedure
19
20  Procedure MenuEvents()
21    Debug EventMenu() ; To see which menu has been selected
22  EndProcedure
23
24 ;
25 ; We just have to open a window and see when an event happen on the
26 ; menu
27 If OpenWindow(0, 100, 100, 300, 260, "SpiderBasic - PopupMenu
Example")
28
29   ListIconGadget(0, 10, 10, 280, 240, "Tools", 200)
30     AddGadgetItem(0, -1, "Hammer")
31     AddGadgetItem(0, -1, "Screwdriver")
32
33   BindEvent(#PB_Event_Menu, @MenuEvents())
34   BindEvent(#PB_Event_Gadget, @GadgetEvents())
35 EndIf

```



See Also

[CreatePopupMenu\(\)](#) , [DisplayPopupMenu\(\)](#) , [CreateMenu\(\)](#) , [CreateImageMenu\(\)](#) , [FreeMenu\(\)](#) , [MenuTitle\(\)](#) , [MenuItem\(\)](#) , [MenuBar\(\)](#) , [OpenSubMenu\(\)](#)

84.5 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) Not used.

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, use BindEvent() , EventWindow() and EventMenu() .

See Also

CreatePopupMenu() , DisplayPopupMenu() , CreateMenu() , CreateImageMenu() , FreeMenu() , MenuTitle() , MenuItem() , MenuBar() , OpenSubMenu()

84.6 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 CreatePopupMenu() .

See Also

CreatePopupMenu() , CreatePopupImageMenu()

84.7 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, "Menu Example")
2  If CreateMenu(0, WindowID(0))      ; menu creation starts....
3    MenuTitle("Project")
4    MenuItem(1, "Open")
5    MenuItem(2, "Save")
6    MenuItem(3, "Save as")
7    MenuItem(4, "Close")
8
9    DisableMenuItem(0, 2, 1)        ; disable the second menu item (Save)
10   EndIf
11   EndIf
```



See Also

`MenuItem()`, `SetMenuItemText()`

84.8 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()`, `CreatePopupImageMenu()`

84.9 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\(\)](#)

84.10 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\(\)](#)

84.11 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() , CreatePopupImageMenu()

84.12 MenuBar

Syntax

MenuBar()

Description

Creates a separator bar in the current menu.

Parameters

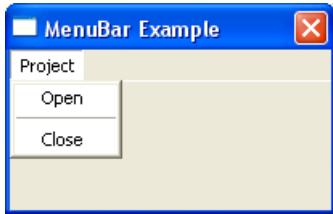
None.

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    MenuTitle("Project")
4    MenuItem(1, "Open")
5    MenuBar() ; here the separator bar will be
6      inserted
7      MenuItem(4, "Close")
8    EndIf
EndIf
```



See Also

`MenuTitle()` , `MenuItem()` , `OpenSubMenu()` , `CreateMenu()` , `CreatePopupMenu()`

84.13 MenuItem

Syntax

```
MenuItem(MenuItemID, Text$, [ImageID])
```

Description

Creates a new item on the current menu.

Parameters

MenuItemID A number to identify this menu item. This value should be between 0 and 65535.

Text\$ The text for the menu item. 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

To have a keyboard shortcut aligned to the right side of the menu (for example, "Save Ctrl+S") use the tab character (#TAB\$ or Chr(9)) to give the correct space. The code may look something like this:

```
1 MenuItem(1, "&Open" + #TAB$ + "Ctrl+O")
```

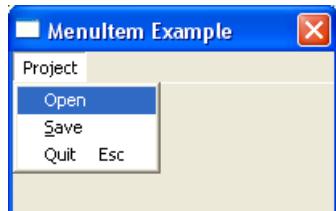
The supported modifiers are:

```
1 - "Ctrl"   : Control key
2 - "Shift"  : Shift key
3 - "Alt"    : Alt key
```

They can be mixed together with the "+" character: "Save As" + #TAB\$ + "Ctrl+Shift+S".

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" + #TAB$ + "Shift+Q")    ; item with separate
shortcut text
8   EndIf
9 EndIf
```



See Also

[MenuTitle\(\)](#) , [MenuBar\(\)](#) , [OpenSubMenu\(\)](#)

84.14 MenuItemID

Syntax

```
MenuItemID = MenuItemID(#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\(\)](#) , [CreatePopupImageMenu\(\)](#)

84.15 MenuTitle

Syntax

```
MenuTitle(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:
"&File" will actually display: File

Return value

None.

Example

```
1 If OpenWindow(0, 200, 200, 200, 100, "MenuTitle Example")
2   If CreateMenu(0, WindowID(0))
3     MenuTitle("&Project")           ; normal menu title with following item
4       MenuItem(1, "Open")
5       MenuItem(2, "Close")
6     MenuTitle("&Edit")            ; menu title with underlined character,
7       the underline               ; will only be displayed, when called
8       with F10 key
9       MenuItem(3, "Undo")
10      MenuItem(4, "Redo")
11    MenuTitle("&About")          ; only menu title
12  EndIf
12 EndIf
```



See Also

MenuItem() , MenuBar() , OpenSubMenu()

84.16 OpenSubMenu

Syntax

```
OpenSubMenu(Text$ [, ImageID])
```

Description

Creates an empty submenu in the current menu.

Parameters

Text\$ The text for the submenu.

In the Text\$ argument, the special '&' character can be used to underline a specific letter: "&File" will actually display: File

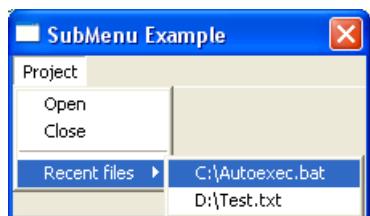
ImageID (optional) An optional image to display next to the submenu. This parameter only has an effect if the current menu was created using the CreateImageMenu() or CreatePopupImageMenu() command. This value can be obtained using the ImageID() function.

Return value

None.

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, "Cookie1.txt")
9         MenuItem( 4, "Cookie2.txt")
10        CloseSubMenu()                ; end submenu
11    EndIf
12  EndIf
```



See Also

[CloseSubMenu\(\)](#) , [MenuTitle\(\)](#) , [MenuItem\(\)](#) , [MenuBar\(\)](#)

84.17 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()

84.18 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()

84.19 BindMenuEvent

Syntax

```
BindMenuEvent (#Menu, MenuItem, @Callback())
```

Description

Bind a menu event to a callback. A menu event can be unbinded 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.

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", #PB_Window_SystemMenu)
11
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())
```

See Also

BindGadgetEvent() , BindMenuEvent() , UnbindEvent()

84.20 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 EndProcedure
8
9 OpenWindow(0, 100, 100, 200, 50, "Click test", #PB_Window_SystemMenu)
10
11 CreateMenu(0, WindowID(0))
12   MenuTitle("File")
13   MenuItem(0, "Test")
14   MenuItem(1, "Quit")
15
16   BindMenuEvent(0, 0, @TestHandler())
17   BindMenuEvent(0, 1, @QuitHandler())
18
19   UnbindMenuEvent(0, 1, @QuitHandler()); Unbind the quit event
```

See Also

[BindEvent\(\)](#) , [BindGadgetEvent\(\)](#) , [BindMenuEvent\(\)](#)

Chapter 85

Mouse

Overview

SpiderBasic provides a full access to mouses. It supports standard mouses with up to 3 buttons. This library is optimized and uses 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() .

85.1 InitMouse

Syntax

```
Result = InitMouse([LockMode])
```

Description

Initializes the Mouse environment for later use. You should call this function before any other functions in this library. If the result is null, no mouse is available.

Parameters

LockMode (optional) It can be one of the following value:

```
#PB_Mouse_Locked: the mouse will be used in exclusive mode, and  
the pointer hidden. Useful for games (default).  
#PB_Mouse_Unlocked: the mouse won't be locked, and the pointer  
will be still visible.
```

Return value

Nonzero if a mouse is available, zero otherwise.

85.2 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.

See Also

ExamineMouse() , MouseDeltaX() , MouseDeltaY() , MouseX() , MouseY() , MouseButton()

85.3 MouseButton

Syntax

```
Result = MouseButton(Button)
```

Description

Returns zero if the specified button number is not pressed, else 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\(\)](#)

85.4 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 left or right since the last call. [ExamineMouse\(\)](#) should be called before this function to update the actual mouse position.

See Also

[ExamineMouse\(\)](#) , [MouseDeltaY\(\)](#)

85.5 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 to up or down since the last call. ExamineMouse() should be called before this function to update the actual mouse position.

See Also

ExamineMouse() , MouseDeltaX()

85.6 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()

85.7 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\(\)](#)

85.8 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\(\)](#)

85.9 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()

85.10 ReleaseMouse

Syntax

```
ReleaseMouse(State)
```

Description

Locks or releases the mouse to allow its use under standard OS.

Parameters

State If set to 1, the mouse is released, else the mouse is locked by the SpiderBasic program.

Return value

None.

See Also

ExamineMouse()

Chapter 86

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 SpiderBasic and programming in general to be able to use this library efficiently.

86.1 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_Any.NewLine: 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
2 ; lower case letter,
3 ; followed with the character 'b' and which ends with an uppercase
4 ; letter. ex: abC
5
6 If CreateRegularExpression(0, "[a-z]b[A-Z]")
7   Debug MatchRegularExpression(0, "abC") ; Will print 1
8   Debug MatchRegularExpression(0, "abc") ; Will print 0
9 Else
10  Debug RegularExpressionError()
11 EndIf
```

See Also

`RegularExpressionError()` , `FreeRegularExpression()`

86.2 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
2 ; lower case letter,
3 ; followed with the character 'b' and which ends with an uppercase
4 ; letter. ex: abC
5 ;
6 If CreateRegularExpression(0, "[a-z]b[A-Z]")
7 Dim Result$(0)
8 NbFound = ExtractRegularExpression(0, "abC ABC zbA abc", Result$())
9 For k = 0 To NbFound-1
10    Debug Result$(k)
11 Next
12 Else
13    Debug RegularExpressionError()
14 EndIf
```

See Also

CreateRegularExpression()

86.3 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()

86.4 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()

86.5 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
2 ; lower case letter,
3 ; followed with the character 'b' and which ends with an uppercase
4 ; letter. ex: abC
5 ;
6 If CreateRegularExpression(0, "[a-z]b[A-Z]")
7   If MatchRegularExpression(0, "abC ABC zbA abc")
8     Debug "The string match !"
9   Else
10    Debug "No pattern found in the string"
11  EndIf
12 Else
13  Debug RegularExpressionError()
14 EndIf
```

See Also

CreateRegularExpression()

86.6 ReplaceRegularExpression

Syntax

```
Result\$ = 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
2 ; lower case letter,
3 ; followed with the character 'b' and which ends with an uppercase
4 ; letter. ex: abC
5 ;
6 If CreateRegularExpression(0, "[a-z]b[A-Z]")
7   Result$ = ReplaceRegularExpression(0, "abC ABC zbA abc", "---")
8   Debug Result$ ; Will print "--- ABC --- abc"
9 Else
10   Debug RegularExpressionError()
11 EndIf
```

See Also

CreateRegularExpression()

86.7 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
2 ; regular expression
3 ;
4 If CreateRegularExpression(0, "[a-z]b[[A-Z] []")
5   Debug "Success"
```

```
5 |     Else
6 |         Debug RegularExpressionError()
7 |     EndIf
```

See Also

[CreateRegularExpression\(\)](#)

Chapter 87

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) are all performed through standard windows called 'requesters'.

87.1 NextSelectedFile

Syntax

```
Result = NextSelectedFile()
```

Description

After a call to OpenFileRequester() it returns the next selected file (if any). It has to be called in the callback specified in OpenFileRequester(). SelectedFileName() () and SelectedFileID() () can be used to get info about the current selected file.

Parameters

None.

Return value

Returns #True if there is another file, #False otherwise.

87.2 SelectedFileName

Syntax

```
Result\$ = SelectedFileName()
```

Description

Returns the current selected filename. It has to be called after NextSelectedFile()(). The filename shouldn't be used directly with other SpiderBasic command, see SelectedFileDialog() for more information.

Parameters

None.

Return value

The selected filename.

87.3 SelectedFileDialog

Syntax

```
Result\$ = SelectedFileDialog()
```

Description

Returns the current selected filename identifier. It has to be called after NextSelectedFile()(). Remote drives like Google Drive can have several files with the exact same filename, that's why a file identifier is needed to be able to select the right one.

Parameters

None.

Return value

A filename identifier. This identifier can be used in the following commands: ReadFile(), OpenFile(), LoadJSON(), LoadImage() and LoadXML() .

87.4 OpenFileRequester

Syntax

```
Filename\$ = OpenFileRequester(Pattern$, Callback [, Flags])
```

Description

Opens the standard requester for the user to choose one or several local files. For security reason, the requester can be opened only from an event callback triggered by a real user action (ie: when the user click on a gadget). This command needs to be put in an event procedure, like in the example below. The selected file is get with NextSelectedFile() and SelectedFileName() in the callback.

Parameters

Pattern\$ A standard filter which allow to display only the files which end with such or such extension.
It has to be in the MIME form : "image/*", "audio/*" etc. A list of most common MIME types can be found here: <http://www.freeformatter.com>.

Callback The callback to be called if the user has selected one or several file. It won't be called if the user canceled the requester. It has to use the following syntax:

```
1 Procedure Callback()
2     ; Code here
3 EndProcedure
```

Flags (optional) It can be a combination of one of the following values:

```
#PB_Requester_MultiSelection: Enable the multiselection (see
NextSelectedFile())
).

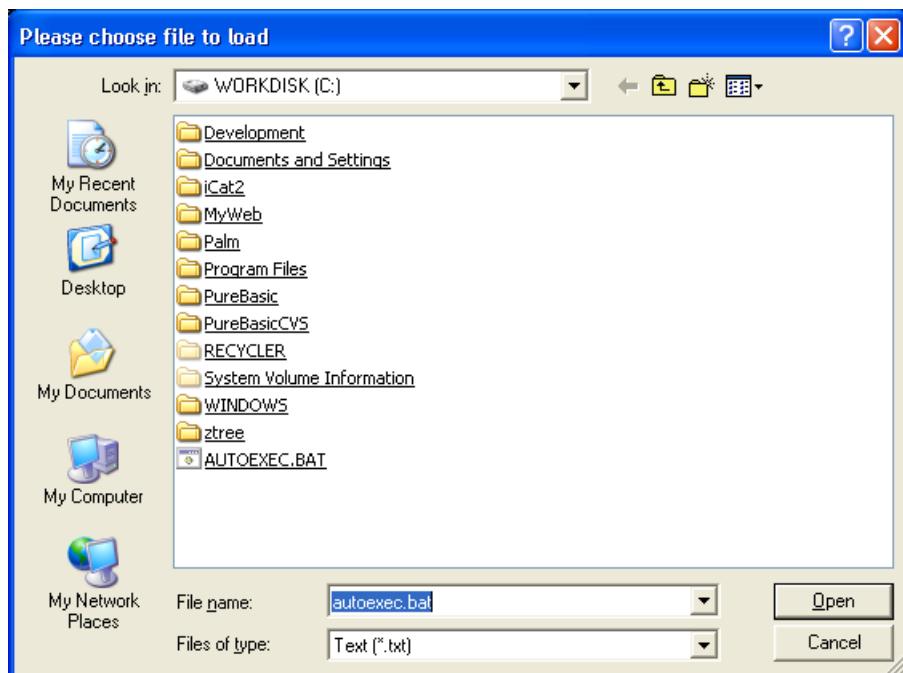
#PB_Requester_GoogleDrive: Uses the Google file requester to
select a file. UseGoogleDrive()
has to be
                                successfully initialized before using
this requester. Note: to be able to open
                                the requester, a domain name needs to
be used (ip address won't work), and this domain
                                needs to be allowed in the google
developper console. For testing purpose,
                                you can modify your /etc/hosts file
and put a domain alias for 127.0.0.1, and
                                then set the 'web server address'
compiler option of the SpiderBasic IDE.
```

Return value

None.

Example

```
1 Procedure RequesterSuccess()
2
3     ; Process all the selected filename
4 ;
5     While NextSelectedFile()
6         Debug "Filename: " + SelectedFileName()
7     Wend
8 EndProcedure
9
10 Procedure ButtonEvent()
11     OpenFileRequester("", @RequesterSuccess(),
12     #PB_Requester_MultiSelection)
13 EndProcedure
14
15 If OpenWindow(0, 100, 100, 200, 55, "File")
16     ButtonGadget(0, 10, 10, 170, 25, "Open local file...")
17     BindGadgetEvent(0, @ButtonEvent())
18 EndIf
```



See Also

`NextSelectedFile()`

87.5 UseGoogleDrive

Syntax

```
UseGoogleDrive(ClientID$, ApiKey$, Callback)
```

Description

Initializes Google Drive access for a specific application. Only one application can be registered at once.

Parameters

ClientID\$ The google client ID of the application which is used to access the drive. Each application has it's own ID, for example "568545051626-vlffi940ra93pmb3pl3tq1eaueejje7h" for the test application 'SpiderEditor'. To create your own id, you need to go to the google developer console.

ApiKey\$ The API key of the application which is used to access the drive. Each application has it's own API key, for example "AIzaSyAp_978UH3YhI4czaGaFmWWVvN14AJgNfVQ" for the test application 'SpiderEditor'. To create your own API key, you need to go to the google developer console.

Callback The callback which will be called when everything is ready to use Google Drive. If it is not called, then Google Drive is not available. It has to be declared like this:

```
1  Procedure Callback()
2      ; Google drive is ready to use.
3  EndProcedure
```

Return value

None.

Example

```
1 Procedure GoogleDriveReady()
2     Debug "Google drive is ready"
3 EndProcedure
4
5 ; Uses the SpiderEditor test application
6 ;
7 UseGoogleDrive("568545051626-vlffi940ra93pmb3pl3tq1eaueejje7h",
8                 "AIzaSyAp_978UH3YhI4czaGaFmWWvN14AJgNfVQ",
9                 @GoogleDriveReady())
```

See Also

[OpenFileRequester\(\)](#)

Chapter 88

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](#).

88.1 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 '#Constantame' (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 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()

88.2 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 '#Constantame' (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 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()

88.3 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 '#Constantame' (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 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()`

88.4 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 and zero otherwise.

88.5 SetRuntimeDouble

Syntax

```
SetRuntimeDouble(Object$ , Value)
```

Description

Changes the double value of the runtime object.

Parameters

Object\$ Name of the object to set the value to. 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\(\)](#)

88.6 SetRuntimeInteger

Syntax

```
SetRuntimeInteger( Object$ , Value )
```

Description

Changes the integer value of the runtime object.

Parameters

Object\$ Name of the object to set the value to. 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\(\)](#)

88.7 SetRuntimeString

Syntax

`SetRuntimeString(Object$, Value$)`

Description

Changes the string value of the runtime object.

Parameters

Object\$ Name of the object to set the value to. 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

`GetRuntimeInteger()` , `IsRuntime()`

Chapter 89

Screen

Overview

A screen is a surface used to display game accelerated content like sprites . A screen can be created either in a regular window , or using the whole display (fullscreen mode).

89.1 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.

89.2 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()`

89.3 FlipBuffers

Syntax

`FlipBuffers()`

Description

Flip the back and front buffers of the current screen. The invisible area is now visible and vice versa, which allows 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.

See Also

OpenScreen() , OpenWindowedScreen()

89.4 IsScreenActive

Syntax

```
Result = IsScreenActive()
```

Description

Checks if the screen still has the main input focus. If not, appropriate actions should be taken such as ReleaseMouse() , pause the game, stop the sounds etc.

Parameters

None.

Return value

Nonzero if the screen is still active, zero otherwise.

See Also

OpenScreen() , OpenWindowedScreen() , ReleaseMouse() , FlipBuffers()

89.5 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()

89.6 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()

89.7 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()

89.8 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()

89.9 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.

Depth Ignored.

Title\$ Text to be display in the brower title bar.

FlipMode (optional) Ignored.

RefreshRate (optional) Ignored.

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() , ResizeScreen()

89.10 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.

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) Ignored.

RightOffset, BottomOffset (optional) Ignored.

FlipMode (optional) Ignored.

Return value

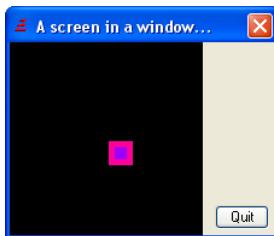
Nonzero if the screen has been successfully opened, zero otherwise.

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 OpenWindow(0, 20, 20, 840, 640, "A screen in a window...",  
2 #PB_Window_SystemMenu | #PB_Window_ScreenCentered)  
3  
4 OpenWindowedScreen(WindowID(0), 20, 20, 800, 600)  
5  
6 Procedure RenderFrame()  
7   Static x, y  
8  
9   ClearScreen(RGB(0, 0, 0))  
10  
11  x+1  
12  DisplaySprite(0, x, 30)  
13  
14  FlipBuffers() ; continue the rendering  
15 EndProcedure  
16  
17 ; Register the render event so FlipBuffers() will trigger the  
18 ; associated procedure  
19 BindEvent(#PB_Event_RenderFrame, @RenderFrame())  
20  
21 CreateSprite(0, 64, 64)  
22 If StartDrawing(SpriteOutput(0))  
23  Circle(32, 32, 25, RGB(255, 0, 0)) ; Red circle  
24  StopDrawing()  
25 EndIf  
26  
27 FlipBuffers() ; trigger the rendering
```



For a more detailed example look at

See Also

OpenScreen(), FlipBuffers(), ResizeScreen()

89.11 ResizeScreen

Syntax

ResizeScreen(Width, Height)

Description

Resizes the screen previously opened with OpenScreen() or OpenWindowedScreen() .

Parameters

Width, Height The new screen resolution, in pixels.

Return value

None.

Remarks

The screen dimension can't be greater than the window size or artefacts can occurs.

See Also

OpenScreen() , OpenWindowedScreen()

Chapter 90

Sort

Overview

Sometimes, elements has to be sorted to be usable or more convenient. SpiderBasic offers highly optimized functions in order to sort arrays and lists , either in ascending or descending order. Furthermore there are functions to reorder the elements of an array or a list in a random order.

90.1 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. Structured and 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 sensitive
(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

If an array is not fully filled then null elements will be sorted first in ascending order and last in descending order.

See Also

RandomizeArray()

90.2 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.

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 sensitive
                     (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.

Return value

None.

See Also

RandomizeList()

90.3 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() , RandomizeList() , Random() , RandomSeed()

90.4 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.

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() , RandomizeArray() , Random() , RandomSeed()

Chapter 91

Sound

Overview

SpiderBasic provides an easy way to have sound inside application or game. It uses special functions to get the maximum speed of available hardware.

91.1 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.

See Also

`SetSoundPosition()`

91.2 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.

See Also

`GetSoundPosition()`

91.3 FreeSound

Syntax

```
FreeSound(#Sound)
```

Description

Stops and removes a sound previously loaded with LoadSound() 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.

91.4 InitSound

Syntax

```
Result = InitSound()
```

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

None.

Return value

Nonzero if the sound environment has been setup correctly, zero otherwise.

91.5 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.

See Also

FreeSound()

91.6 LoadSound

Syntax

```
Result = LoadSound(#Sound, Filename$, [Flags])
```

Description

Load any sound format supported by the web browser from an URL.

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) Ignored.

Return value

Nonzero if the sound has been created, zero otherwise. The sound is still not loaded, the callbacks binded to #PB_Event_Loading and #PB_Event_LoadingError will be called once the loading is done. If #PB_Any was used for the #Sound parameter then the generated number is returned on success.

Example

```
1  InitSound()
2
3  Procedure Loading(Type, Filename$, ObjectId)
4      Debug Filename$ + " loaded (id = " + ObjectId + ")"
5
```

```

6   PlaySound(0) ; play the sound
7 EndProcedure
8
9 Procedure LoadingError(Type, Filename$)
10  Debug Filename$ + ": loading error"
11 EndProcedure
12
13 ; Register the loading event before calling any resource load command
14 BindEvent(#PB_Event_Loading, @Loading())
15 BindEvent(#PB_Event>LoadingError, @LoadingError())
16
17 LoadSound(0, "Data/Lazer.wav") ; Be sure this sound is available (the
      source file should be saved in the SpiderBasic/Examples drawer)

```

See Also

[FreeSound\(\)](#) , [PlaySound\(\)](#)

91.7 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.

See Also

[LoadSound\(\)](#) , [ResumeSound\(\)](#)

91.8 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.

See Also

LoadSound() , PauseSound()

91.9 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:

```
#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.

See Also

`StopSound()` , `FreeSound()` , `PauseSound()` , `ResumeSound()`

91.10 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).
```

91.11 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.

91.12 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.

91.13 SoundVolume

Syntax

```
SoundVolume(#Sound, Volume [, 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 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.

See Also

`LoadSound()`

91.14 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.

See Also

`PlaySound()`

Chapter 92

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, SpiderBasic 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.

92.1 ClipSprite

Syntax

```
ClipSprite(#Sprite, x, y, Width, Height)
```

Description

Creates a clip zone for 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.

See Also

DisplaySprite() , DisplayTransparentSprite()

92.2 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.

92.3 CreateSprite

Syntax

```
Result = CreateSprite(#Sprite, Width, Height [, Mode])
```

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).

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 created, zero otherwise. If #PB_Any was used for the #Sprite parameter then the generated number is returned on success.

See Also

SpriteOutput()

92.4 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()

92.5 DisplayTransparentSprite

Syntax

```
DisplayTransparentSprite(#Sprite, x, y [, Intensity [, Color]])
```

Description

Display the #Sprite at the specified position on the current screen. This command will use the sprite alpha-channel information to have transparency effect (32-bit PNG can be used to get sprites with alpha-channel). 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()`

92.6 FreeSprite

Syntax

```
FreeSprite(#Sprite)
```

Description

Removes the specified sprite from memory. It's not possible to 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.

92.7 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.

92.8 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()

92.9 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 any image format supported by the web browser.

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 (PNG
only for now).
```

Return value

Nonzero if the sprite has been created, zero otherwise. The sprite is still not loaded, the callbacks binded to #PB_Event_Loading and #PB_Event>LoadingError will be called once the loading is done. 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.

92.10 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()`

92.11 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()

92.12 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()

92.13 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()`.

See Also

`SpriteCollision()`

92.14 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()`

92.15 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.

Example

```
1 StartDrawing(SpriteOutput(#Sprite))
2 ; do some drawing stuff here...
3 StopDrawing()
```

92.16 RotateSprite

Syntax

```
RotateSprite(#Sprite, Angle.f, Mode)
```

Description

Rotates the specified #Sprite to the given 'Angle'.

Parameters

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.

92.17 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.

92.18 ZoomSprite

Syntax

```
ZoomSprite(#Sprite, Width, Height)
```

Description

Zooms the specified #Sprite from the given dimension.

Parameters

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 93

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.

93.1 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 '!'; will print '33'
```

A table with all Ascii values and their relating figures may be found [here](#).

See Also

Chr()

93.2 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_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  
#PB_Quad    : The value is handled as a quad number, ranging from  
0 to 18446744073709551615
```

Return value

A string holding the binary representation of the specified value.

Example

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 "00000000000100000"
```

See Also

Str() , Val() and Hex() .

93.3 Chr

Syntax

```
Text\$ = Chr(CharacterValue)
```

Description

Returns a string created with the given character value.

Parameters

CharacterValue The character value.

Return value

Returns a string created with the given character value.

Example

```
1 Debug Chr(33) ; Will display "!"
```

Remarks

A table with all Ascii values and their relating figures may be found here .

See Also

Asc()

93.4 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
```

93.5 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 is the string isn't found. The first character index is 1.

```
1 Debug FindString("SpiderBasic", "Bas") ; will display 7
```

93.6 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_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  
#PB_Quad    : The value is handled as a quad number, ranging from  
0 to 18446744073709551615
```

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() and Bin() .

93.7 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()

93.8 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()

93.9 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()

93.10 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
```

93.11 LSet

Syntax

```
Result\$ = LSet(String$, Length [, Character$])
```

Description

Pads a string to the left by adding extra characters to fit the specified length.

Parameters

String\$ The string to pad.

Length The total length (in characters) of the new string.

Character\$ (optional) The character used to pad extra space in 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 left 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()

93.12 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 Word", "!!") ; Will display "Hello World"
```

See Also

RSet() , LSet() , RTrim() , Trim()

93.13 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"
```

93.14 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\(\)](#)

93.15 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)
```

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.

Example

```
1 Debug ReplaceString("This is Art", " is", " was") ; Will display
   "This was Art"
2 Debug ReplaceString("Hello again, hello again", "HELLO", "oh no...", 
   1, 10) ; Will display "Hello again, oh no... again"
3
4 test$ = "Bundy, Barbie, Buddy"
5 ReplaceString(test$, "B", "Z", 2, 1) ; all B gets changed to Z
   (directly in memory, no valid return-value here)
6 Debug test$      ; Output of the changed string
```

See Also

RemoveString() , InsertString()

93.16 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()

93.17 RSet

Syntax

```
Result\$ = RSet(String$, Length [, Character$])
```

Description

Pads a string to the right by adding extra characters to fit the specified length.

Parameters

String\$ The string to pad.

Length The total length (in characters) of the new string.

Character\$ (optional) The character used to pad extra space in 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()

93.18 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 Word!!", " !") ; Will display "Hello World"
```

See Also

RSet() , LSet() , LTrim() , Trim()

93.19 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 splitted string", k, " ")
3 Next
```

93.20 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,
3                                ; so we get a float number rounded to 10 decimal places
4 Debug "Result: " + value           ; same as previous line
5 Debug "Result: " + StrF(value,2)    ; we want a result with two decimal
6                                ; places, no rounding needed as we have only two
7 Debug "Result: " + StrF(value,0)    ; we want a result with no decimal
8                                ; places, so the value is rounded
```

See Also

StrD() , Str() , StrU()

93.21 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, 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()

93.22 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 = 10000000000000001
2 Debug "Result: " + Str(Value)
```

See Also

Val() , Hex() , Bin() , StrF() , StrD() , StrU()

93.23 StrU

Syntax

```
Result\$ = StrU(Value.q, Type)
```

Description

Converts an unsigned numeric number into a string.

Parameters

Value.q The value to convert.

Type It can be one of the following value:

```
#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_Uncode: 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
#PB_Quad    : The value is handled as a quad number, ranging from
0 to 18446744073709551615

```

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()

93.24 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"
```

93.25 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 " -      - "
```

93.26 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()

93.27 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()

93.28 ValD

Syntax

```
Result.d = ValD(String$)
```

Description

Converts a string into a double value. The string must be a double in decimal 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()).

Example

```
1 Debug ValD("10.000024") ; will display 10.000024.
```

See Also

ValF() , Val() , Str() , StrF() , StrD()

93.29 ValF

Syntax

```
Result.f = ValF(String$)
```

Description

Converts a string into a float value. The string must be a float in decimal 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()).

Example

```
1 Debug ValF("10.24") ; Will display 10.24.
```

See Also

ValD() , Val() , Str() , StrF() , StrD()

93.30 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("$10FFFFFFF") ; will print '73014444031'.
3 Debug Val("%1000") ; will print '8'.
```

See Also

`ValD()` , `ValD()` , `Str()` , `StrF()` , `StrD()`

Chapter 94

System

Overview

The system library offers access to specialized commands.

94.1 ElapsedMilliseconds

Syntax

```
Result = 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 Global Start = ElapsedMilliseconds()
2
3 Procedure TimerEvent()
4   Debug "Elapsed time in ms: " + Str(ElapsedMilliseconds() - Start)
5 EndProcedure
6
7 OpenWindow(0, 0, 0, 100, 100, "Timer")
8 AddWindowTimer(0, 0, 1000)
9 BindEvent(#PB_Event_Timer, @TimerEvent())

```

94.2 CountProgramParameters

Syntax

```
Result = CountProgramParameters()
```

Description

Returns the number of parameters specified when calling the web page. Parameters are usually set in the URL: <http://www.website.com/index.html?param1=hello¶m2=world>.

Parameters

None.

Return value

The number of parameters specified when calling the web page.

Remarks

ProgramParameter() may be used to read the individual parameters.

See Also

[ProgramParameter\(\)](#)

94.3 ProgramParameter

Syntax

```
Result\$ = ProgramParameter(Index)
```

Description

Gets the specified parameter that was set in the URL.

Parameters

Index The parameter to get. The first parameter index starts from 0.

Return value

The parameter and it's associated value, if any. StringField() can be used to split the parameter and its value.

Example

```
1 For Index = 0 To CountProgramParameters() - 1
2   Debug ProgramParameter(Index)
3   Debug "name: " + StringField(ProgramParameter(Index), 1, "=")
4   Debug "value: " + StringField(ProgramParameter(Index), 2, "=")
5 Next
```

94.4 BatteryLevel

Syntax

```
Result = BatteryLevel()
```

Description

Returns the current battery level of the device.

Return value

The current battery level between 0 (empty) and 100 (fully charged). If the device is plugged, #@pb_battery_plugged will be returned. If an error occurred while retrieving the battery status, #PB_Battery_Unknown will be returned.

94.5 DeviceInfo

Syntax

```
Result\$ = DeviceInfo(Type)
```

Description

Returns information about the device.

Parameters

Type It can be one of the following value:

```

#PB_Device_Model      : The device codename model. This value can be
                        different across versions of the same product.
#PB_Device_Platform: The underlying platform ("Android" or "iOS").
#PB_Device_UUID       : The unique ID identifier of the device
#PB_Device_Version   : The OS version (ie: "4" for iOS 4, "4.1" for
                        Android 4.1 etc.)
#PB_Device_Manufacturer: The manufacturer ("Apple", "Motorola"
                        etc.)
#PB_Device_Serial     : The device hardware serial number

```

Return value

The required system information, or an empty string if it can't be retrieved.

Example

```

1 Debug "Model: " + DeviceInfo(#PB_Device_Model)
2 Debug "Platform: " + DeviceInfo(#PB_Device_Platform)
3 Debug "UUID: " + DeviceInfo(#PB_Device_UUID)
4 Debug "Version: " + DeviceInfo(#PB_Device_Version)
5 Debug "Manufacturer: " + DeviceInfo(#PB_Device_Manufacturer)
6 Debug "Serial number: " + DeviceInfo(#PB_Device_Serial)

```

94.6 VibrateDevice

Syntax

```
VibrateDevice(Time)
```

Description

Vibrates the device for a specific length of time.

Parameters

Time The time (in milliseconds) to vibrate the device.

Return value

None. @remarkWhen used in an Android application, the user needs to interact at least once to enable vibration. For example a 'Start' button can be set to force a first interaction. Once unlocked, it will work for the whole app lifetime. It's a Chrome protection to avoid nasty app to vibrate without any user consent.

Example

```

1 VibrateDevice(1000) ; Vibrate the device for 1 second.

```

Chapter 95

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. SpiderBasic allows to create any number of toolbar and to handle them as if it was a menu.

95.1 CreateToolBar

Syntax

```
Result = CreateToolBar(#ToolBar, WindowID)
```

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.

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 Procedure MenuEvents()
2   Debug "ToolBar item selected: " + EventMenu()
3 EndProcedure
4
5 If OpenWindow(0, 0, 0, 295, 260, "ToolBar example",
6   #PB_Window_SystemMenu | #PB_Window_SizeGadget |
7   #PB_Window_ScreenCentered)
8
9   CreateImage(0, 16, 16, 32, #PB_Image_Transparent)
10  If StartDrawing(ImageOutput(0))
11    Circle(8, 8, 7, RGB(255, 0, 0))
12    StopDrawing()
13  EndIf
14
15  If CreateToolBar(0, WindowID(0))
16    ToolBarImageButton(0, ImageID(0))
17    ToolBarImageButton(2, ImageID(0))
18    ToolBarSeparator()
19    ToolBarImageButton(3, ImageID(0))
20    ToolBarToolTip(0, 3, "Cut")
21
22    ToolBarImageButton(4, ImageID(0))
23    ToolBarToolTip(0, 4, "Copy")
24  EndIf
25
26  DisableToolBarButton(0, 2, 1) ; Disable the button '2'
27  BindEvent(#PB_Event_Menu, @MenuEvents())
EndIf
```



See Also

ToolBarImageButton() , ToolBarSeparator() , FreeToolBar()

95.2 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()

95.3 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.

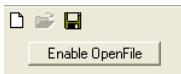
Example

```
1 Procedure MenuEvents()
2   Debug "ToolBar item selected: " + EventMenu()
3 EndProcedure
4
5 If OpenWindow(0, 0, 0, 295, 260, "ToolBar example",
6   #PB_Window_SystemMenu | #PB_Window_SizeGadget |
7   #PB_Window_ScreenCentered)
8
9   CreateImage(0, 16, 16, 32, #PB_Image_Transparent)
10  If StartDrawing(ImageOutput(0))
11    Circle(8, 8, 7, RGB(255, 0, 0))
12    StopDrawing()
```

```

11     EndIf
12
13 If CreateToolBar(0, WindowID(0))
14   ToolBarImageButton(0, ImageID(0))
15   ToolBarImageButton(2, ImageID(0))
16   ToolBarSeparator()
17   ToolBarImageButton(3, ImageID(0))
18   ToolBarToolTip(0, 3, "Cut")
19
20   ToolBarImageButton(4, ImageID(0))
21   ToolBarToolTip(0, 4, "Copy")
22 EndIf
23
24 DisableToolBarButton(0, 2, 1) ; Disable the button '2'
25
26 BindEvent(#PB_Event_Menu, @MenuEvents())
27 EndIf

```



See Also

[ToolBarImageButton\(\)](#)

95.4 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()`

95.5 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()`

95.6 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()

95.7 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.

See Also

CreateToolBar()

95.8 ToolBarImageButton

Syntax

```
ToolBarImageButton(#Button, ImageID [, Mode])
```

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 PNG file format.

Mode (optional) The mode value can be one of the following constants:

```
#PBToolBar_Normal: the button will act as standard button  
(default)  
#PBToolBar_Toggle: the button will act as toggle button
```

GetToolBarButtonState() and SetToolBarButtonState() can be used to retrieve or modify a toggle button state.

Return value

None.

Example

```
1 Procedure MenuEvents()  
2   Debug "ToolBar item selected: " + EventMenu()  
3 EndProcedure  
4  
5 If OpenWindow(0, 0, 0, 295, 260, "ToolBar example",  
6   #PB_Window_SystemMenu | #PB_Window_SizeGadget |  
7   #PB_Window_ScreenCentered)  
8  
9   CreateImage(0, 16, 16, 32, #PB_Image_Transparent)  
10  If StartDrawing(ImageOutput(0))  
11    Circle(8, 8, 7, RGB(255, 0, 0))  
12    StopDrawing()  
13  EndIf  
14  
15  If CreateToolBar(0, WindowID(0))  
16    ToolBarImageButton(0, ImageID(0), #PBToolBar_Toggle)  
17    ToolBarToolTip(0, 0, "Open")  
18    ToolBarImageButton(1, ImageID(0))  
19    ToolBarToolTip(0, 1, "Save")  
20  EndIf
```

```
20     BindEvent(#PB_Event_Menu, @MenuEvents())
21 EndIf
```



See Also

CreateToolBar() , ToolBarSeparator()

95.9 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 Procedure MenuEvents()
2     Debug "ToolBar item selected: " + EventMenu()
3 EndProcedure
4
5 If OpenWindow(0, 0, 0, 295, 260, "ToolBar example",
6     #PB_Window_SystemMenu | #PB_Window_SizeGadget |
7     #PB_Window_ScreenCentered)
8
9     CreateImage(0, 16, 16, 32, #PB_Image_Transparent)
10    If StartDrawing(ImageOutput(0))
11        Circle(8, 8, 7, RGB(255, 0, 0))
12        StopDrawing()
13    EndIf
14
15    If CreateToolBar(0, WindowID(0))
16        ToolBarImageButton(0, ImageID(0))
17        ToolBarSeparator()
18        ToolBarImageButton(2, ImageID(0))
19        ToolBarSeparator()
20        ToolBarImageButton(3, ImageID(0))
```

```

19     EndIf
20
21     BindEvent(#PB_Event_Menu, @MenuEvents())
22 EndIf

```



See Also

CreateToolBar() , ToolBarImageButton()

95.10 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 Procedure MenuEvents()
2   Debug "ToolBar item selected: " + EventMenu()
3 EndProcedure
4
5 If OpenWindow(0, 0, 0, 295, 260, "ToolBar example",
6   #PB_Window_SystemMenu | #PB_Window_SizeGadget |
7   #PB_Window_ScreenCentered)
8
9   CreateImage(0, 16, 16, 32, #PB_Image_Transparent)
10  If StartDrawing(ImageOutput(0))
11    Circle(8, 8, 7, RGB(255, 0, 0))
12    StopDrawing()
13  EndIf
14
15  If CreateToolBar(0, WindowID(0))

```

```

14     ToolBarImageButton(0, ImageID(0))
15     ToolBarToolTip(0, 0, "Open")
16     ToolBarImageButton(1, ImageID(0))
17     ToolBarToolTip(0, 1, "Save")
18 EndIf
19
20 BindEvent(#PB_Event_Menu, @MenuEvents())
21 EndIf

```



See Also

[ToolBarImageButton\(\)](#) , [ToolBarSeparator\(\)](#)

95.11 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 96

TouchScreen

Overview

SpiderBasic provides an easy access to the screen touches. This library is mainly designed for game or fullscreen applications.

96.1 ExamineTouchScreen

Syntax

```
Result = ExamineTouchScreen()
```

Description

Updates the screen touches state. This function has to be called successfully before using other touches commands like TouchX() , TouchY() , TouchDeltaX() , TouchDeltaY() or TouchScreenPushed() .

Parameters

None.

Return value

Nonzero if touchscreen commands are availables (ie: the device have a touchscreen), zero otherwise.

See Also

TouchX() , TouchY() , TouchDeltaX() , TouchDeltaY() TouchScreenPushed()

96.2 TouchDeltaX

Syntax

```
Result = TouchDeltaX(Finger)
```

Description

Returns the finger 'x' movement (in pixels) since the last call of this function.

Parameters

Finger The finger to get the value. Up to five fingers are supported at once, value can be between 0 and 4. TouchScreenPushed() can be used to know which fingers are currently on the screen.

Return value

The finger '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 finger has been moved to left or right since the last call. ExamineTouchScreen() should be called before this function to update the actual finger position.

See Also

ExamineTouchScreen() , TouchDeltaY()

96.3 TouchDeltaY

Syntax

```
Result = TouchDeltaY(Finger)
```

Description

Returns the finger 'y' movement (in pixels) since the last call of this function.

Parameters

Finger The finger to get the value. Up to five fingers are supported at once, value can be between 0 and 4. TouchScreenPushed() can be used to know which fingers are currently on the screen.

Return value

The finger '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 finger has been moved to top or bottom since the last call. ExamineTouchScreen() should be called before this function to update the actual finger position.

See Also

ExamineTouchScreen() , TouchDeltaX()

96.4 TouchX

Syntax

```
Result = TouchX(Finger)
```

Description

Returns the finger 'x' position (in pixels) on the current screen. ExamineTouchScreen() should be called before this function to update the actual finger position.

Parameters

Finger The finger to get the value. Up to five fingers are supported at once, value can be between 0 and 4. TouchScreenPushed() can be used to know which fingers are currently on the screen.

Return value

The actual finger 'x' position (in pixels) on the current screen.

See Also

ExamineTouchScreen() , TouchY()

96.5 TouchY

Syntax

```
Result = TouchY(Finger)
```

Description

Returns the finger 'y' position (in pixels) on the current screen. ExamineTouchScreen() should be called before this function to update the actual finger position.

Parameters

Finger The finger to get the value. Up to five fingers are supported at once, value can be between 0 and 4. TouchScreenPushed() can be used to know which fingers are currently on the screen.

Return value

The finger 'y' position (in pixels) on the current screen.

See Also

ExamineTouchScreen() , TouchX()

96.6 TouchScreenPushed

Syntax

```
Result = TouchScreenPushed(Finger)
```

Description

Checks if the specified finger is pressed.

Parameters

Finger The finger to get the value. Up to five fingers are supported at once, value can be between 0 and 4.

Return value

Nonzero if the specified finger is pushed, zero otherwise.

Example

```
1  OpenScreen(800, 600, 32, "Test")
2
3  Procedure RenderFrame()
4      Static SpriteFinger = -1
5      Static x, y
6
7      ClearScreen(RGB(0, 0, 0))
8
9      If ExamineTouchScreen() ; TouchScreen is detected and available
10
11         For k = 0 To 4 ; Up to 5 possible fingers at once. We need to
12             check them all, as some finger can be removed in between
13
14         If TouchScreenPushed(k) And SpriteFinger = -1 ; One finger
15             press detected, use it to move the sprite
16             SpriteFinger = k
17             EndIf
18             Next
19
20         If SpriteFinger <> -1 And TouchScreenPushed(SpriteFinger) ;
21             Ensure the finger used to move the sprite is still pressed
22             x = TouchX(SpriteFinger) - SpriteWidth(0) / 2 ; We want our
23             finger centered in the sprite
24             y = TouchY(SpriteFinger) - SpriteHeight(0) / 2
25             Else
26                 SpriteFinger = -1
27             EndIf
28
29             DisplaySprite(0, x, y)
30
31             FlipBuffers(); // continue the rendering
32             Else
33                 Debug "No touchscreen device detected"
```

```

30     EndIf
31
32 EndProcedure
33
34 Procedure Loading(Type, Filename$, ObjectId)
35     Static NbLoadedElements
36
37     NbLoadedElements+1
38     If NbLoadedElements = 1 ; The loading of all images and sounds is
        finished, we can start the rendering
        FlipBuffers(); // start the rendering
39     EndIf
40 EndProcedure
41
42 Procedure LoadingError(Type, Filename$, ObjectId)
43     Debug Filename$ + ": loading error"
44 EndProcedure
45
46 ; Register the loading event before calling any resource load command
47 BindEvent(#PB_Event_Loading, @Loading())
48 BindEvent(#PB_Event>LoadingError, @LoadingError())
49 BindEvent(#PB_Event_RenderFrame, @RenderFrame())
50
51 LoadSprite(0, "Data/SpiderBasicLogo.png")
52

```

See Also

[ExamineKeyboard\(\)](#) , [KeyboardReleased\(\)](#)

Chapter 97

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.

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 vector image outputs. 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 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() .

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.

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.

97.1 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:
ImageVectorOutput() : Drawing will be rendered directly on the Image data (see CreateImage())
CanvasVectorOutput() : Drawing will be rendered directly on the CanvasGadget()

Return value

Returns nonzero if drawing is possible or zero if the operation failed.

Remarks

Drawing must be finished with StopVectorDrawing() .

See Also

StopVectorDrawing()

97.2 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()

97.3 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\(\)](#)

97.4 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()

97.5 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()

97.6 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).

See Also

`VectorResolutionX()`

97.7 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
#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
```

97.8 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

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", 30, #PB_Font_Bold)
5
6  If StartVectorDrawing(CanvasVectorOutput(0))
7
8      VectorSourceColor(RGBA(255, 0, 0, 255))
9      VectorFont(FontID(0))
10
11     MovePathCursor(20, 20)
12     DrawVectorText("Normal text")
13
14     ; Changes made to the drawing state within this block do not
15     ; affect the other commands
16     SaveVectorState()
17     MovePathCursor(120, 160)
18     RotateCoordinates(120, 160, -50)
19     VectorSourceColor(RGBA(0, 0, 255, 255))
20     DrawVectorText("Rotated text")
21     RestoreVectorState()
22
23     MovePathCursor(220, 140)
24     DrawVectorText("Normal text")
25
26     StopVectorDrawing()
27 EndIf
28 EndIf
```

See Also

[RestoreVectorState\(\)](#) , [BeginVectorLayer\(\)](#)

97.9 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()

97.10 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.

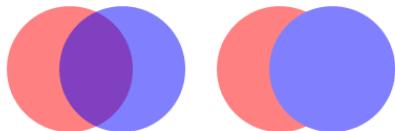
Example

```
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         ; Semi-transparent drawing on the base layer
7         AddPathCircle(75, 100, 60)
```

```

8     VectorSourceColor(RGBA(255, 0, 0, 127))
9     FillPath()
10    AddPathCircle(125, 100, 60)
11    VectorSourceColor(RGBA(0, 0, 255, 127))
12    FillPath()
13
14    ; Opaque drawing on a semi-transparent layer
15    BeginVectorLayer(127)
16        AddPathCircle(275, 100, 60)
17        VectorSourceColor(RGBA(255, 0, 0, 255))
18        FillPath()
19        AddPathCircle(325, 100, 60)
20        VectorSourceColor(RGBA(0, 0, 255, 255))
21        FillPath()
22    EndVectorLayer()
23
24    StopVectorDrawing()
25 EndIf
26 EndIf

```



See Also

`EndVectorLayer()` , `SaveVectorState()`

97.11 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()

97.12 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
5      If StartVectorDrawing(CanvasVectorOutput(0))
6          ; make the entire output red
7          VectorSourceColor(RGBA(255, 0, 0, 255))
8          FillVectorOutput()
9
10     StopVectorDrawing()
11 EndIf
12 EndIf
```

See Also

FillPath()

97.13 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()
ConvertCoordinateX() , ConvertCoordinateY()

97.14 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
```

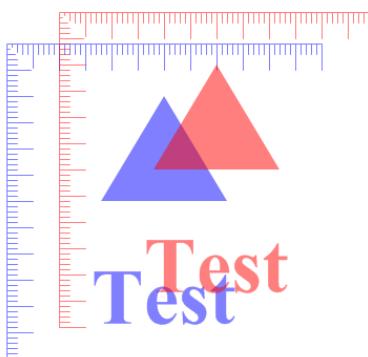
Return value

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  
pixels in each direction  
13  
14    VectorSourceColor(RGBA(255, 0, 0, 128))  
15    MovePathCursor(50, 50)  
16    DrawVectorText("Test")
```

```
17     StopVectorDrawing()
18 EndIf
19 EndIf
```



See Also

`ResetCoordinates()` , `ScaleCoordinates()` , `RotateCoordinates()` , `SkewCoordinates()`
`ConvertCoordinateX()` , `ConvertCoordinateY()`

97.15 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:

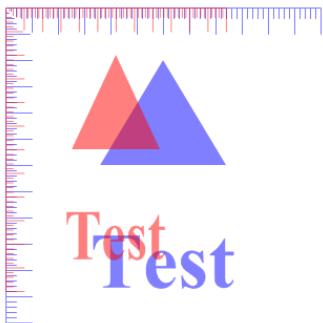
```
#PB_Coordinate_User : Change the coordinate system for points in
the drawing path (default)
#PB_Coordinate_Source: Change the coordinate system for the
vector drawing source
```

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
  EndIf
```

Test

See Also

ResetCoordinates() , TranslateCoordinates() , RotateCoordinates() , SkewCoordinates()
ConvertCoordinateX() , ConvertCoordinateY()

97.16 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
```

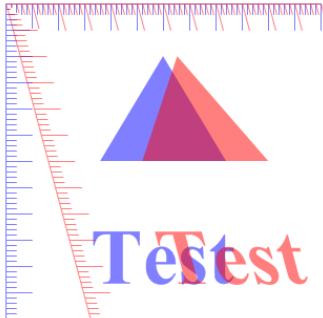
Return value

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  
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     SkewCoordinates(45, 0)
12
13     VectorSourceColor(RGBA(255, 0, 0, 128))
14     MovePathCursor(50, 50)
15     DrawVectorText("Test")
16
17     StopVectorDrawing()
18 EndIf
19 EndIf

```



See Also

[ResetCoordinates\(\)](#) , [TranslateCoordinates\(\)](#) , [ScaleCoordinates\(\)](#) , [RotateCoordinates\(\)](#) ,
[ConvertCoordinateX\(\)](#) , [ConvertCoordinateY\(\)](#)

97.17 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
```

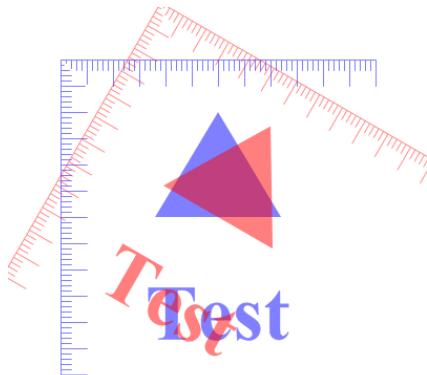
Return value

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 the
13     (50, 50) point
14
15     VectorSourceColor(RGBA(255, 0, 0, 128))
16     MovePathCursor(50, 50)
17     DrawVectorText("Test")
18
19     StopVectorDrawing()
EndIf
EndIf
```



See Also

[ResetCoordinates\(\)](#) , [TranslateCoordinates\(\)](#) , [ScaleCoordinates\(\)](#) , [ConvertCoordinateX\(\)](#) , [ConvertCoordinateY\(\)](#)

97.18 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  
2 ; coordinate system  
3 ; by mapping the coordinates from the device system (pixels) to the  
4 ; user system (points)  
5 ;  
6 Procedure OnCanvas()  
7     If EventType() = #PB_EventType_LButtonDown  
8         If StartVectorDrawing(CanvasVectorOutput(0, #PB_Unit_Point))  
9             RotateCoordinates(0, 0, 30)  
10            CanvasX = GetGadgetAttribute(0, #PB_Canvas_MouseX)  
11            CanvasY = GetGadgetAttribute(0, #PB_Canvas_MouseY)
```

```

13
14     DrawingX = ConvertCoordinateX(CanvasX, CanvasY,
15     #PB_Coordinate_Device, #PB_Coordinate_User)
16     DrawingY = ConvertCoordinateY(CanvasX, CanvasY,
17     #PB_Coordinate_Device, #PB_Coordinate_User)
18
19     AddPathBox(DrawingX, DrawingY, 30, 30)
20     VectorSourceColor(RGBA(Random(255), Random(255), Random(255),
21     255))
22     FillPath()
23
24     StopVectorDrawing()
25 EndIf
26
27 EndIf
28 EndProcedure
29
30 If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
31     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
32     CanvasGadget(0, 0, 0, 400, 200)
33     BindGadgetEvent(0, @OnCanvas())
34 EndIf

```

See Also

[ResetCoordinates\(\)](#) , [TranslateCoordinates\(\)](#) , [ScaleCoordinates\(\)](#) , [RotateCoordinates\(\)](#) , [SkewCoordinates\(\)](#) [ConvertCoordinateY\(\)](#)

97.19 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() ConvertCoordinateX()

97.20 ResetPath

Syntax

```
ResetPath()
```

Description

Resets the vector drawing path to an empty path and moves the cursor to position (0, 0).

Parameters

None.

Return value

None.

See Also

IsPathEmpty()

97.21 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 EndIf
```



See Also

FillPath() , IsInsidePath() , MovePathCursor() , AddPathLine()

97.22 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:

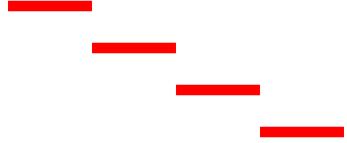
```
#PB_Path_Default : The new position is absolute (default)  
#PB_Path_Relative: The new position is relative to the last  
cursor position.
```

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      MovePathCursor(40, 40)  
7      For i = 1 To 4  
8        AddPathLine(80, 0, #PB_Path_Relative)  
9        MovePathCursor(0, 40, #PB_Path_Relative)  
10     Next i  
11  
12     VectorSourceColor(RGBA(255, 0, 0, 255))  
13     StrokePath(10)  
14  
15     StopVectorDrawing()  
16   EndIf  
17 EndIf
```



See Also

`ClosePath()` , `AddPathLine()` , `FillPath()` , `StrokePath()`

97.23 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.

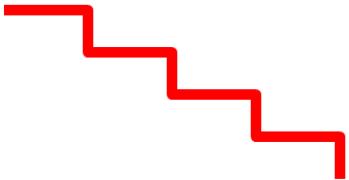
Example

```
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 EndIf

```



See Also

[MovePathCursor\(\)](#) , [ClosePath\(\)](#) , [AddPathArc\(\)](#) , [AddPathCurve\(\)](#) , [AddPathCircle\(\)](#) , [AddPathEllipse\(\)](#) , [AddPathBox\(\)](#)

97.24 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:

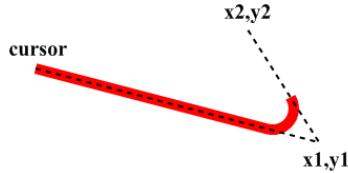
```
#PB_Path_Default : The positions are absolute (default)
#PB_Path_Relative: The positions are relative to the last cursor
position.
```

Return value

None.

Remarks

The following image illustrates the meaning of the two reference points and the 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
5      If StartVectorDrawing(CanvasVectorOutput(0))
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 EndIf
```



See Also

MovePathCursor() , AddPathLine() , AddPathCurve() , AddPathCircle() , AddPathEllipse() , AddPathBox()

97.25 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:

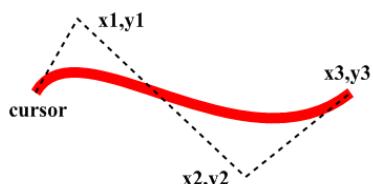
```
#PB_Path_Default : The positions are absolute (default)
#PB_Path_Relative: The positions are relative to the last cursor
position.
```

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          MovePathCursor(50, 100)
7          AddPathCurve(90, 30, 250, 180, 350, 100)
8          VectorSourceColor(RGBA(255, 0, 0, 255))
9          StrokePath(10)
10
11      StopVectorDrawing()
12  EndIf
13 EndIf
```

See Also

MovePathCursor() , AddPathLine() , AddPathArc() , AddPathCircle() , AddPathEllipse() , AddPathBox()

97.26 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 EndIf
```



See Also

MovePathCursor() , AddPathLine() , AddPathArc() , AddPathCircle() , AddPathEllipse() ,
AddPathCurve()

97.27 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 stand-alone 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
15   VectorSourceColor(RGBA(255, 0, 0, 255))
16   StrokePath(10)
17
18   StopVectorDrawing()
19 EndIf
EndIf
```



See Also

MovePathCursor() , AddPathLine() , AddPathArc() , AddPathBox() , AddPathEllipse() , AddPathCurve()

97.28 AddPathEllipse

Syntax

```
AddPathEllipse(x.d, y.d, RadiusX.d, RadiusY.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.

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
7      ; regular ellipse
8      AddPathEllipse(100, 100, 80, 30)
9
10     ; rotated ellipse
11     SaveVectorState()
12     RotateCoordinates(300, 100, 45)
13     AddPathEllipse(300, 100, 80, 30)
14     RestoreVectorState()
15
16     VectorSourceColor(RGBA(255, 0, 0, 255))
17     StrokePath(10)
18
19     StopVectorDrawing()
20 EndIf
EndIf
```



See Also

MovePathCursor() , AddPathLine() , AddPathArc() , AddPathBox() , AddPathCircle() ,
AddPathCurve()

97.29 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	MovePathCursor()
L x y	AddPathLine()
C x1 y1 x2 y2 x3 y3	AddPathCurve()
Z	ClosePath()

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:

#PB_Path_Default	: No special behavior (<i>default value</i>)
#PB_Path_Relative	: Interpret all coordinates as relative to the current path cursor

Return value

None.

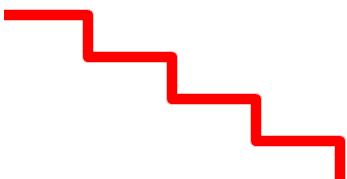
Example

```
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
7     280 100 L 280 140 L 360 140 L 360 180")
8   VectorSourceColor(RGBA(255, 0, 0, 255))
9   StrokePath(10, #PB_Path_RoundCorner)
10
11  StopVectorDrawing()
12 EndIf
EndIf

```



See Also

[PathSegments\(\)](#)

97.30 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 ; This example uses the IsInsidePath() function to color the figure
2     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
11        content
12        FillVectorOutput()
13
14        AddPathEllipse(200, 100, 150, 75)                  ; prepare path
15
16        If IsInsidePath(x, y, #PB_Coordinate_Device)    ; check if the
17            mouse is inside
18            VectorSourceColor(RGBA(0, 255, 0, 255))
19        Else
20            VectorSourceColor(RGBA(0, 0, 255, 255))
21        EndIf
22
23        FillPath()                                     ; fill path
24        StopVectorDrawing()
25        EndIf
26    EndProcedure
27
28
29 Procedure OnMouseMove()
30     Draw()
31 EndProcedure
32
33 If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
34     #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
35     CanvasGadget(0, 0, 0, 400, 200)
36     LoadFont(0, "Times New Roman", 20, #PB_Font_Bold)
37     Draw()
38
39     BindGadgetEvent(0, @OnMouseMove(), #PB_EventType_MouseMove)
40 EndIf
```

See Also

FillPath() , ClosePath() , ResetPath()

97.31 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()

97.32 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:

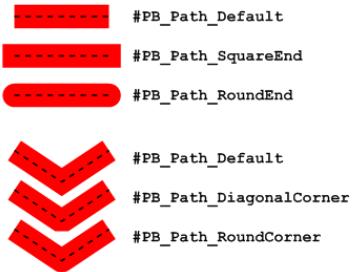
#PB_Path_Default	: No special behavior (default value)
#PB_Path_Preserve	: Don't reset the path after this function
#PB_Path_RoundEnd	: Draw the line(s) with a rounded ends
#PB_Path_SquareEnd	: Draw the line(s) with a square box at the ends
#PB_Path_RoundCorner	: Draw the line(s) with rounded corners
#PB_Path_DiagonalCorner	: Draw the line(s) with diagonally cut corners

Return value

None.

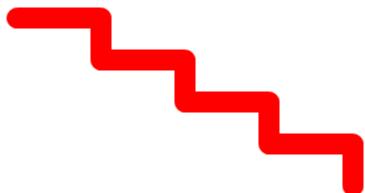
Remarks

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
13     VectorSourceColor(RGBA(255, 0, 0, 255))
14     StrokePath(20, #PB_Path_RoundCorner|#PB_Path_RoundEnd)
15
16     StopVectorDrawing()
17 EndIf
EndIf
```



See Also

[FillPath\(\)](#) , [DotPath\(\)](#) , [DashPath\(\)](#) , [CustomDashPath\(\)](#) , [ResetPath\(\)](#)

97.33 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	:	Don't reset the path after this function
#PB_Path_RoundEnd	:	Draw the dots round
#PB_Path_SquareEnd	:	Draw the dots as squares

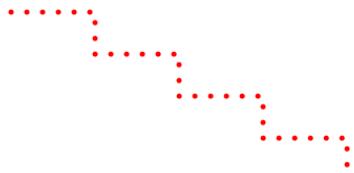
StartOffset.d (optional) Specifies the distance to skip within the dot pattern before starting to draw the path. The default value is 0.

Return value

None.

Example

```
1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
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     DotPath(5, 10, #PB_Path_RoundEnd)
15
16     StopVectorDrawing()
17 EndIf
EndIf
```



See Also

`FillPath()` , `StrokePath()` , `DashPath()` , `CustomDashPath()` , `ResetPath()`

97.34 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:

<code>#PB_Path_Default</code>	: No special behavior (<code>default</code> 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

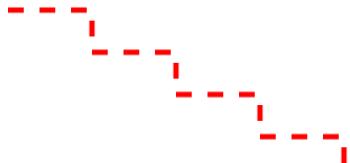
None.

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             DashPath(5, 15)
15
16             StopVectorDrawing()
17             EndIf
EndIf

```



See Also

[FillPath\(\)](#) , [StrokePath\(\)](#) , [DotPath\(\)](#) , [CustomDashPath\(\)](#) , [ResetPath\(\)](#)

97.35 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:

```

#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

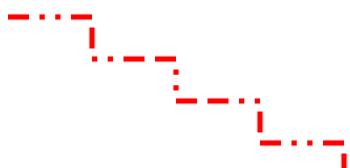
None.

Example

```

1  If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
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
17  dashes(1) = 10
18  dashes(2) = 0 ; draw a dot
19  dashes(3) = 10
20  dashes(4) = 0
21  dashes(5) = 10
22  dashes(6) = 20
23  dashes(7) = 10
24  CustomDashPath(5, dashes())
25
26  StopVectorDrawing()
27  EndIf
EndIf

```



See Also

FillPath() , StrokePath() , DotPath() , DashPath() , ResetPath()

97.36 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 one of the following values:

```
#PB_Path_Default      : No special behavior (default value)
#PB_Path_Preserve     : Don't reset the path after this function
```

Return value

None.

Remarks

If the path has overlapping figures, it is filled in an odd/even fashion. 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 EndIf
```



See Also

[StrokePath\(\)](#) , [DotPath\(\)](#) , [DashPath\(\)](#) , [CustomDashPath\(\)](#) , [ResetPath\(\)](#)

97.37 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\(\)](#)

97.38 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()

97.39 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      ; construct path
7      MovePathCursor(150, 125)
8      AddPathCurve(0, 270, 0, -150, 350, 180)
9
10     ; get location & angle of point on the path
11     x = PathPointX(200)
12     y = PathPointY(200)
13     a = PathPointAngle(200)
```

```

14      ; stroke the path
15      VectorSourceColor(RGB(255, 0, 0))
16      StrokePath(5)
17
18      ; draw a marker at the path point
19      AddPathCircle(x, y, 10)
20      VectorSourceColor(RGB(0, 0, 255))
21      FillPath()
22
23      MovePathCursor(x, y)
24      AddPathLine(30*Cos(Radian(a)), 30*Sin(Radian(a)),
#PB_Path_Relative)
25      StrokePath(5)
26
27
28      StopVectorDrawing()
29      EndIf
30      EndIf

```



See Also

[PathPointY\(\)](#) , [PathPointAngle\(\)](#) , [PathLength\(\)](#)

97.40 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()

97.41 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()

97.42 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
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
17     StopVectorDrawing()
18 EndIf
19 EndIf
```

See Also

PathPointX() , PathPointY() , PathPointAngle()

97.43 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

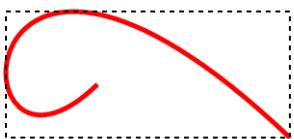
None.

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($FF0000FF)
19     StrokePath(5)
20
21     ; draw bounding box
22     AddPathBox(x, y, w, h)
23     VectorSourceColor($FF000000)
24     DashPath(2, 5)
25
26     StopVectorDrawing()
27     EndIf
EndIf
```



See Also

PathBoundsY() , PathBoundsWidth() , PathBoundsHeight()

97.44 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()

97.45 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()

97.46 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()

97.47 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
EndIf
```

See Also

AddPathSegments()

97.48 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() , FillPath() , FillVectorOutput() , StrokePath() , DotPath() , DashPath() , CustomDashPath()

97.49 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

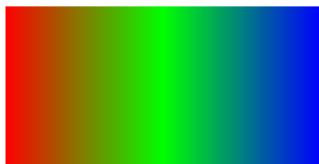
None.

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",
2      #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3      CanvasGadget(0, 0, 0, 400, 200)
4
5  If StartVectorDrawing(CanvasVectorOutput(0))
6
7      VectorSourceLinearGradient(50, 0, 350, 0)
8      VectorSourceGradientColor(RGBA(255, 0, 0, 255), 0.0)
9      VectorSourceGradientColor(RGBA(0, 255, 0, 255), 0.5)
10     VectorSourceGradientColor(RGBA(0, 0, 255, 255), 1.0)
11
12     AddPathBox(50, 25, 300, 150)
13     FillPath()
14
15     StopVectorDrawing()
16 EndIf
EndIf
```



See Also

[VectorSourceGradientColor\(\)](#) , [VectorSourceCircularGradient\(\)](#) , [VectorSourceColor\(\)](#)

97.50 VectorSourceCircularGradient

Syntax

```
VectorSourceCircularGradient(x.d, y.d, Radius.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.

Return value

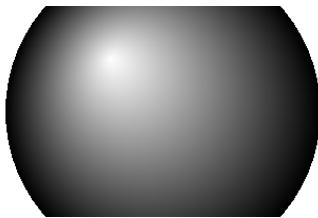
None.

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
7      VectorSourceCircularGradient(200, 100, 150, -50, -50)
8      VectorSourceGradientColor(RGBA(255, 255, 255, 255), 0.0)
9      VectorSourceGradientColor(RGBA(0, 0, 0, 255), 1.0)
10
11     FillVectorOutput()
12
13   EndIf
14 EndIf
```



See Also

[VectorSourceGradientColor\(\)](#) , [VectorSourceLinearGradient\(\)](#) , [VectorSourceColor\(\)](#)

97.51 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()

97.52 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 width 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 Procedure Loaded(Type, Filename$, ObjectId)
2     If OpenWindow(0, 0, 0, 400, 200, "VectorDrawing",
3         #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
4         CanvasGadget(0, 0, 0, 400, 200)
5
6     If StartVectorDrawing(CanvasVectorOutput(0))
7
8         MovePathCursor(120, 0)
9         RotateCoordinates(120, 0, 35)
10        DrawVectorImage(ImageID(0), 127)
11
12    StopVectorDrawing()
13 EndIf
14 EndProcedure
15
16 Procedure LoadingError(Type, Filename$, ObjectId)
17     Debug Filename$ + ": loading error"
18 EndProcedure
19
20 ; Register the loading event before calling any resource load command
21 BindEvent(#PB_Event>Loading, @Loaded())
22 BindEvent(#PB_Event>LoadingError, @LoadingError())
23
24 LoadImage(0, "Data/Spider.png")
```

See Also

MovePathCursor() , PathCursorX() , PathCursorY()

97.53 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

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.

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
8          VectorFont(FontID(0), 25)
9          VectorSourceColor(RGBA(0, 0, 0, 80))
10         Text$ = "The quick brown fox jumped over the lazy doc"
11
12         For i = 1 To 6
13             MovePathCursor(200 - VectorTextWidth(Text$)/2, 100 -
14                 VectorTextHeight(Text$)/2)
15             DrawVectorText(Text$)
16             RotateCoordinates(200, 100, 30)
17             Next i
18
19         StopVectorDrawing()
20     EndIf
21 EndIf
```

See Also

VectorTextWidth() , VectorTextHeight() , VectorFont()

97.54 VectorFont

Syntax

```
VectorFont(FontID [, Size.d])
```

Description

Specifies the font to use for vector drawing.

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.

See Also

DrawVectorText() , VectorTextWidth() , VectorTextHeight()

97.55 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:

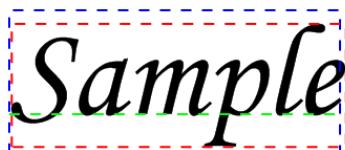
The "logical bounding box" of a character or text defines the space that the cursor must move to properly draw text next to each other. However, the actual drawn characters may extend beyond that box (for example in case of cursive or serif fonts). When determining where to draw text, the logical bounding 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",
2   #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
3   CanvasGadget(0, 0, 0, 500, 250)
4   LoadFont(0, "Monotype Corsiva", 50, #PB_Font_Bold)
5
6   If StartVectorDrawing(CanvasVectorOutput(0))
7
8     VectorFont(FontID(0), 125)
9     Text$ = "Sample"
10
11    ; draw text
12    MovePathCursor(25, 25)
13    DrawVectorText(Text$)
14
15    ; draw logical bounding box
16    AddPathBox(25, 25, VectorTextWidth(Text$),
17      VectorTextHeight(Text$))
18    VectorSourceColor(RGBA(0, 0, 255, 255))
19    DashPath(2, 10)
20
21    ; draw visible bounding box
22    AddPathBox(25 + VectorTextWidth(Text$,
23      #PB_VectorText_Visible | #PB_VectorText_Offset),
24      25 + VectorTextHeight(Text$,
25        #PB_VectorText_Visible | #PB_VectorText_Offset),
26        VectorTextWidth(Text$, #PB_VectorText_Visible),
27        VectorTextHeight(Text$, #PB_VectorText_Visible))
28    VectorSourceColor(RGBA(255, 0, 0, 255))
29    DotPath(2, 10)
30
31    ; draw baseline
32    MovePathCursor(25, 25 + VectorTextHeight(Text$,
33      #PB_VectorText_Baseline))
34    AddPathLine(VectorTextWidth(Text$), 0, #PB_Path_Relative)
35    VectorSourceColor(RGBA(0, 255, 0, 255))
    DashPath(2, 10)
36
37    StopVectorDrawing()
38  EndIf
39 EndIf
```



See Also

[VectorTextHeight\(\)](#) , [DrawVectorText\(\)](#) , [VectorFont\(\)](#)

97.56 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", 50, #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$)
13|
14|         ; draw logical bounding box
15|         AddPathBox(25, 25, VectorTextWidth(Text$),
16|                     VectorTextHeight(Text$))
17|         VectorSourceColor(RGBA(0, 0, 255, 255))
18|         DashPath(2, 10)
19|
20|         ; draw visible bounding box
21|         AddPathBox(25 + VectorTextWidth(Text$,
22|                                         #PB_VectorText_Visible|#PB_VectorText_Offset),
23|                     25 + VectorTextHeight(Text$,
24|                                         #PB_VectorText_Visible|#PB_VectorText_Offset),
25|                     VectorTextWidth(Text$, #PB_VectorText_Visible),
26|                     VectorTextHeight(Text$, #PB_VectorText_Visible))
27|         VectorSourceColor(RGBA(255, 0, 0, 255))
28|         DotPath(2, 10)
29|
30|         ; draw baseline
31|         MovePathCursor(25, 25 + VectorTextHeight(Text$,
32|                                         #PB_VectorText_Baseline))
33|         AddPathLine(VectorTextWidth(Text$), 0, #PB_Path_Relative)
34|         VectorSourceColor(RGBA(0, 255, 0, 255))
35|         DashPath(2, 10)

36|         StopVectorDrawing()
37|     EndIf
38| EndIf

```



See Also

[VectorTextWidth\(\)](#) , [DrawVectorText\(\)](#) , [VectorFont\(\)](#)

Chapter 98

Window

Overview

Windows are essential components of modern interfaces. SpiderBasic provides full access to them.

98.1 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_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_Insert  
#PB_Shortcut_Delete
```

```
#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_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
```

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. 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 allows to use the 'Apple' key (left or right) to define shortcuts. This constant is also supported on other OS (to ease portability), but will act like #PB_Shortcut_Control.

Example

```
1 Procedure MenuEvents()  
2   Debug "Menu event: " + EventMenu()  
3 EndProcedure  
4  
5 If OpenWindow(0, 0, 0, 295, 260, "CTRL+F Shortcut",  
6   #PB_Window_SystemMenu | #PB_Window_SizeGadget |  
7   #PB_Window_ScreenCentered)  
8  
9   AddKeyboardShortcut(0, #PB_Shortcut_Control | #PB_Shortcut_F, 15)  
10  BindEvent(#PB_Event_Menu, @MenuEvents())  
11 EndIf
```

See Also

RemoveKeyboardShortcut()

98.2 AddWindowTimer

Syntax

```
AddWindowTimer(#Window, Timer, Timeout)
```

Description

Adds a new timer to the specified window. BindEvent() can be used with #PB_Event_Timer to get the event. RemoveWindowTimer() function can be used to remove the timer.

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 Procedure TimerEvents()
2     Debug "Timer event: " + EventTimer()
3 EndProcedure
4
5 If OpenWindow(0, 0, 0, 295, 260, "Timer example",
6     #PB_Window_SystemMenu | #PB_Window_SizeGadget |
7     #PB_Window_ScreenCentered)
8
9     AddWindowTimer(0, 0, 1000) ; First timer every seconds
10    AddWindowTimer(0, 1, 3000) ; Second timer every 3 seconds
11
12    BindEvent(#PB_Event_Timer, @TimerEvents())
13 EndIf
```

See Also

RemoveWindowTimer() , EventTimer()

98.3 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()

98.4 EventTimer

Syntax

```
Timer = EventTimer()
```

Description

After an event of type #PB_Event_Timer 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()

98.5 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.

See Also

OpenWindow()

98.6 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.

98.7 EventGadget

Syntax

```
GadgetNumber = EventGadget()
```

Description

After an event of type #PB_Event_Gadget, use this function to determine which gadget has been triggered.

Parameters

None.

Return value

Returns the #Gadget number associated to the event.

See Also

EventMenu() , EventTimer() , EventType()

98.8 EventMenu

Syntax

```
MenuItem = EventMenu()
```

Description

After an event of type #PB_Event_Menu 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

[EventGadget\(\)](#) , [EventWindow\(\)](#)

98.9 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\(\)](#) , [EventGadget\(\)](#) , [EventWindow\(\)](#)

98.10 EventType

Syntax

```
EventType = EventType()
```

Description

After a [#PB_Event_Gadget](#), 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\(\)](#)

- EditorGadget()
- ImageGadget()
- ListViewGadget()
- ListIconGadget()
- SpinGadget()
- StringGadget()
- 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) occurs:

```
#PB_EventType_LeftClick      : Left mouse button click
#PB_EventType_RightClick    : right mouse button click
#PB_EventType_LeftDoubleClick: Left mouse button double click
#PB_EventType_RightDoubleClick: Right mouse button double click
#PB_EventType_Focus         : Get the focus.
#PB_EventType_LostFocus     : Lose the focus.
#PB_EventType_Change        : Content change.
```

Example

```

1  Procedure GadgetEvents()
2      Select EventGadget()
3          Case 1
4              Select EventType()
5                  Case #PB_EventType_LeftClick      : Debug "Click with left
mouse button"
6                  Case #PB_EventType_RightClick    : Debug "Click with right
mouse button"
7                  Case #PB_EventType_LeftDoubleClick: Debug "Double-click
with left mouse button"
8                  Case #PB_EventType_RightDoubleClick: Debug "Double-click
with right mouse button"
9              EndSelect
10             EndSelect
11         EndProcedure
12
13     If OpenWindow(0, 0, 0, 330, 320, "EventType example...", 
#PB_Window_SystemMenu | #PB_Window_ScreenCentered)
14         ListIconGadget(1, 10, 10, 310, 300, "ListIcon", 280)
15         For k = 1 To 4
16             AddGadgetItem(1, -1, "Line " + k)
17         Next
18
19         BindEvent(#PB_Event_Gadget, @GadgetEvents())
20     EndIf

```

See Also

EventGadget() , EventType() , EventTimer()

98.11 EventWindow

Syntax

```
WindowNumber = EventWindow()
```

Description

After an event, 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

EventGadget() , EventMenu() , EventTimer()

98.12 GetActiveWindow

Syntax

```
WindowNumber = GetActiveWindow()
```

Description

Returns the number of the window which currently has the keyboard focus or -1 if no window 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

A window can be activated (set the focus on it) with the SetActiveWindow() function.

See Also

[SetActiveWindow\(\)](#)

98.13 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\(\)](#)

98.14 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\(\)](#)

98.15 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

[SetTitle\(\)](#)

98.16 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.

See Also

OpenWindow()

98.17 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()

98.18 OpenWindow

Syntax

```
Result = OpenWindow(#Window, x, y, InnerWidth, InnerHeight, Title$, [,
Flags [, ParentWindowID]])
```

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.

ParentWindowID (optional) The WindowID the new window belongs to. 'ParentWindowID' value can be easily obtained with WindowID().

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_Background	: The window doesn't have a frame and is put in the web browser background. It can only have one window with the background flag opened at the same time. This window will be automatically resized when the web browser is resized.
#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_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 ('ParentWindowID' must be specified).
                                x,y parameters are
                                ignored.
#PB_Window_NoActivate     : Don't activate the window after
    opening.

```

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.

See Also

[CloseWindow\(\)](#)

98.19 PostEvent

Syntax

```
PostEvent(Event [, Window, Object [, Type [, Data]]])
```

Description

Posts an event at the end of the internal event queue.

Parameters

Event The event to post. For a list of SpiderBasic events, see [BindEvent\(\)](#). 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 integer number. 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

None.

See Also

EventWindow() , EventGadget() , EventType() , EventData()

98.20 RemoveKeyboardShortcut

Syntax

```
RemoveKeyboardShortcut(#Window, Shortcut)
```

Description

Removes a keyboard shortcut previously defined with AddKeyboardShortcut() .

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.

Return value

None.

See Also

AddKeyboardShortcut()

98.21 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 The new window position, in pixels. If 'x' or 'y' is set to **#PB_Ignore**, the current value of 'x' or 'y' won't be changed.

Width, Height The new window size, in pixels. If 'Width' or 'Height' is set to **#PB_Ignore**, the current value of 'Width' or 'Height' won't be changed.

Return value

None.

98.22 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()

98.23 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. @remarkGetWindowColor() can be used to get the current background color of the window.

See Also

GetWindowColor()

98.24 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()

98.25 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 EndIf
```

See Also

GetWindowTitle()

98.26 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.

98.27 BindEvent

Syntax

```
BindEvent(Event, @Callback() [, Window [, Object [, EventType]]])
```

Description

Bind an event to a callback. It's the way to handle events in SpiderBasic. An event can be unbound with UnbindEvent() .

Parameters

Event The event to bind. Custom events are also supported, when using PostEvent() . Possible Events are:

```
#PB_Event_Menu : a menu
has been selected
#PB_Event_Gadget : a gadget
has been pushed
#PB_Event_Timer : a timer
has reached its timeout
#PB_Event_CloseWindow : the window close gadget has been
pushed
#PB_Event_SizeWindow : the window has been resized
#PB_Event_MoveWindow : the window has been moved
#PB_Event_ActivateWindow : the window has been activated (got
the focus)
#PB_Event_DeactivateWindow: the window has been deactivated (lost
the focus)
#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
#PB_Event>Loading : a new item has been loaded. It needs
a specific event callback, see below for more information.
#PB_Event>LoadingError : an error occurred when loading an
item. It needs a specific event callback, see below for more
information.
#PB_Event_RenderFrame : a new frame is ready to be rendered.
FlipBuffers()
has to be called to trigger this event.
#PB_Event_SizeDesktop : the desktop has been resized. For
example, it can be the browser window which has been resized,
or the device orientation which has
been changed.
```

@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.

If the event is `#PB_Event>Loading` or `#PB_Event>LoadingError`, the callback has to be declared like this:

```
1 Procedure EventLoading(Type, Filename$, ObjectId)
2     ; Code
3 EndProcedure
```

The 'type' argument will be one of the following value:

```
#PB_Loading_Image : The item is an image loaded with @loadimage()
#PB_Loading_Sound : The item is a sound loaded with @loadsound()
#PB_Loading_Sprite: The item is a sprite loaded with @loadsprite()
#PB_Loading_Xml   : The item is a XML object loaded with
@loadxml()
#PB_Loading_JSON  : The item is a JSON object loaded with
@loadjson()
```

The 'Filename\$' argument will be the original filename as specified in the load command. The 'ObjectId' argument is the object number, as specified in the load command.

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.

Object (optional) The object number to bind the event to. It can be a gadget or menuitem number. `#PB_All` can be used to bind the event to any objects.

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 EndProcedure
4
5 OpenWindow(0, 100, 100, 200, 200, "Resize test",
#PB_Window_SizeGadget | #PB_Window_SystemMenu)
6
7 BindEvent(#PB_Event_SizeWindow, @SizeWindowHandler())
```

See Also

`BindGadgetEvent()` , `BindMenuEvent()` , `UnbindEvent()`

98.28 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 `BindEvent()`. 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 or menuitem 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 it
immediately
```

See Also

`BindEvent()` , `BindGadgetEvent()` , `BindMenuEvent()`

98.29 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.

Example

```
1 If OpenWindow(0, 0, 0, 300, 300, "Resize me !", #PB_Window_SystemMenu  
2 | #PB_Window_ScreenCentered | #PB_Window_SizeGadget)  
3   WindowBounds(0, 200, 200, 400, 400)  
EndIf
```

98.30 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()`

98.31 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.

98.32 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()

98.33 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()

98.34 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()

98.35 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 Procedure TimerEvent()
2     SetGadgetText(0, "Window mouse position: " + WindowMouseX(0) + ", "
+ WindowMouseY(0))
3 EndProcedure
4
5 If OpenWindow(0, 0, 0, 300, 30, "Window mouse monitor",
#PB_Window_SystemMenu | #PB_Window_ScreenCentered)
6     TextGadget(0, 10, 6, 200, 20, "")
7
8     AddWindowTimer(0, 0, 50) ; every 50 ms
9     BindEvent(#PB_Event_Timer, @TimerEvent())
10    EndIf

```

98.36 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 Procedure TimerEvent()
2     SetGadgetText(0, "Window mouse position: " + WindowMouseX(0) + ", "
+ WindowMouseY(0))
3 EndProcedure
4
5 If OpenWindow(0, 0, 0, 300, 30, "Window mouse monitor",
#PB_Window_SystemMenu | #PB_Window_ScreenCentered)
6     TextGadget(0, 10, 6, 200, 20, "")
7
8     AddWindowTimer(0, 0, 50) ; every 50 ms
9     BindEvent(#PB_Event_Timer, @TimerEvent())
10    EndIf

```

Chapter 99

XML

Overview

The XML library provides set of functions to easily add XML parsing and creating capability to applications.

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.

99.1 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()

99.2 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.

99.3 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 ; Display the xml
16 Debug ComposeXML(xml)
```

See Also

FreeXML() , CreateXMLNode() , RootXMLNode()

99.4 LoadXML

Syntax

```
Result = LoadXML(#XML, Filename$, [, Encoding [, Flags]])
```

Description

Loads a XML tree from an URL or a local 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) Ignored.

Flags (optional) It can be one of the following value:

```
#PB_LocalFile: the filename is a local file, OpenFileRequester()
() needs to be called before
          to have access to local files. SelectedFileID()
() is used to get the
          local file identifier.
```

Return value

Nonzero if the xml object has been created. The xml data is still not loaded, the callbacks binded to #PB_Event_Loading and #PB_Event>LoadingError will be called once the loading is done. If #PB_Any was used for the #XML parameter then the generated number is returned on success.

See Also

CreateXML() , FreeXML() , ParseXML()

99.5 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).

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() , ParseXML()

99.6 XMLStatus

Syntax

```
Result = XMLStatus(#XML)
```

Description

Returns the status of the last parsing operation done on this XML tree (using LoadXML() or ParseXML()). This function should be called after every LoadXML() or ParseXML() 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_Error	: an error has occurred

99.7 XMLError

Syntax

```
Result\$ = XMLError(#XML)
```

Description

In case of an error while parsing XML data this function returns an error-message describing the error. XMLStatus() 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, XMLErrorLine() or XMLErrorPosition() can be used.

See Also

XMLErrorLine() , XMLErrorPosition() , XMLStatus()

99.8 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()

99.9 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

XMLError() , XMLErrorLine() , XMLStatus()

99.10 ExportXML

Syntax

```
ExportXML (#XML , $Filename$)
```

Description

Exports the specified xml to the user through a download.

Parameters

#XML The xml to export.

\$Filename\$ The filename to use. This is the name which will be shown to the user when the download starts.

Return value

None.

See Also

LoadXML() , CreateXML() , ComposeXML()

99.11 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) Not used.

Return value

Returns the markup as a string.

See Also

ExportXML()

99.12 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() , SetXMLEncoding()

99.13 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_Uncode (UTF16)  
#PB_UTF8
```

This only affects the exporting of the tree. The data in the #XML object is always stored in the PB string format (Ascii or Unicode depending on the compiler option). So a unicode executable can safely change the encoding to #PB_Ascii for saving and then back to something else without loosing any information in the tree in memory.

Return value

None.

See Also

ExportXML() , GetXMLEncoding()

99.14 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_StandaloneYes : The document mode is standalone  
#PB_XML_StandaloneNo : The document mode is not standalone
```

See Also

SetXMLStandalone()

99.15 SetXMLStandalone

Syntax

```
SetXMLStandalone(#XML, Standalone)
```

Description

Changes the "standalone" attribute of the XML declaration when exporting/saving the document.

Parameters

#XML The XML to use.

Standalone It can be one of these values:

```
#PB_XML_StandaloneYes : The document mode is standalone  
#PB_XML_StandaloneNo : The document mode is not standalone
```

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

GetXMLStandalone()

99.16 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()`

99.17 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()`

99.18 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()

99.19 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()

99.20 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.

99.21 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\(\)](#)

99.22 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\(\)](#)

99.23 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 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
```

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\(\)](#)

99.24 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.

99.25 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 trees 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.)

99.26 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()`

99.27 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()`

99.28 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\(\)](#)

99.29 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\(\)](#)

99.30 GetXMLNodeName

Syntax

```
Result\$ = 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

`GetXMLNodeText()`

99.31 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()`

99.32 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()`

99.33 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()

99.34 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()

99.35 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()

99.36 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()

99.37 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()

99.38 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()

99.39 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()

99.40 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()

99.41 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()

99.42 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\(\)](#)