



PAIZI

SailWind Layout Command Reference

Release SailWind 3.0
Document Revision 1.0

Copyright and Disclaimer of SailWind Software
Copyright (c) 2023-2024 Chengdu Paizi Interconnect Electronics Technology Co., Ltd.

Copyright Information

All copyrights, patent rights, trademark rights, trade secrets, and other related intellectual property rights of the SailWind software (hereinafter referred to as the "Software"), including but not limited to its source code, object code, user interface design, graphics, images, audio, video, algorithms, data models, documentation, etc., belong to Chengdu Paizi Interconnect Electronics Technology Co. Ltd. (hereinafter referred to as the "Copyright Owner").

Installation and Use License

Users should clearly agree to all terms of this copyright and disclaimer before installing and using this software. By running this installation or software, the user indicates that they have read and agree to be bound by this copyright and disclaimer.

The copyright owner grants users a non exclusive, limited, and revocable installation license, allowing them to install the software on their designated computer devices and use its related features to complete their design tasks under the guidance of the software.

Users are not allowed to copy, distribute, modify, sell, rent, lend, transfer, reverse engineer, decompile, create derivative works, or otherwise use this software in any form, except with the explicit written permission of the copyright owner.

Disclaimer During Installation and Use

This software is provided as is, and the copyright owner does not guarantee that it is error free, defect free, and does not guarantee that the installation and use process will be successfully completed, nor does it make any commitment to the applicability, stability, security, or reliability of the installation and use process.

Users should bear the risk of using this software themselves. The copyright owner shall not be liable for any direct or indirect losses, data loss, business interruption, system damage, or other damages caused by the use or inability to use this software.

Limitations and Reservations of Rights

The use of this software is subject to the limitations and constraints of this copyright and disclaimer. The copyright owner reserves all rights not explicitly granted to users. Users are not allowed to perform any form of reverse engineering, decompilation, disassembly, decryption, modification, creation of derivative works, or use to create similar software on this software.

Other

The copyright owner has the right to modify the terms of this copyright and disclaimer at any time, and the modified terms will be notified to users through appropriate means. If the user continues to use this installation and software, it means that they have accepted the modified terms.

If any part of this copyright and disclaimer is deemed invalid or unenforceable for any reason, that part shall be deemed separate from the whole, but shall not affect the validity of other parts.

Based on the permanently authorized PADS[®] software of Siemens Industry Software Inc.

Contact Information

If you have any questions or suggestions about this installation or software, please contact:

Email: market@pzeda.com
Phone: 0755-86703052
Website: www.pzeda.com

Revision History

Revision	Changes	Date
1.0	Initial release, corresponding to SailWind V3.0	2024-03-15

Table of Contents

Chapter 1

Scripting and the Automation Server.....	25
Scripts.....	25
Automation in a Multiple Release Environment.....	26
Basic Sample Scripts.....	27
Basic Sample Scripts/RGL Reports.....	28
Script Creation.....	30
Creating a New Script.....	30
Inserting an Automation Statement.....	30
Setting the Next Statement.....	31
Showing the Next Statement.....	31
Saving the Script.....	31
Script Implementation.....	32
Running a Script.....	32
Pausing a Running Script.....	32
Stopping a Running Script.....	33
Managing Scripts.....	34
Opening an Existing Script.....	34
Managing Open Scripts.....	34
Editing a Script.....	35
Editing a User Dialog Box.....	35
Finding an Automation Statement.....	35
Printing a Script.....	35
Saving a Script.....	36
Watching a Variable.....	36
Debugging Scripts.....	37
Setting or Removing the Breakpoints.....	37
Debugging the Scripts.....	37
Removing All Breakpoints in the Script.....	38
Correcting Run-time Errors.....	38
Accessing Help on the Basic Language.....	38
The Automation Server.....	39
OLE Background.....	39
The Automation Server Object Hierarchy.....	40
SailWind Layout Automation Samples.....	43
Troubleshooting Samples.....	44
Creating Wrapper Classes for SailWind Layout Objects.....	44
Sample 1 - Automation Client.....	45
Sample 2 - Notifications.....	46
Sample 3 - Database Access.....	46
Sample 4 - Cross-Probing.....	47
Sample 5 - A SailWind Layout Custom Reports Command.....	48
Sample 6 - Database Objects Accessed through the Automation Interface.....	51
SailWind Layout Automation Server Reference	53
Automation Objects.....	54

Table of Contents

The AntiPad Object.....	55
AntiPad.Application Property.....	56
AntiPad.CornerRadius Property.....	57
AntiPad.CornerRadius Property.....	58
AntiPad.Length Property.....	59
AntiPad.Name Property.....	60
AntiPad.ObjectType Property.....	61
AntiPad.Offset Property.....	62
AntiPad.Orientation Property.....	63
AntiPad.PadStackLayer Property.....	64
AntiPad.Parent Property.....	65
AntiPad.Shape Property.....	66
AntiPad.Size Property.....	67
The Application Object.....	68
Application.ActiveDocument Property.....	69
Application.Application Property.....	70
Application.DefaultFilePath Property.....	71
Application.FullName Property.....	72
Application.Libraries Property.....	73
Application.Name Property.....	74
Application.ObjectType Property.....	75
Application.Parent Property.....	76
Application.Preference Property.....	77
Application.ProgressBar Property.....	78
Application.StatusBarText Property.....	79
Application.Version Property.....	80
Application.Visible Property.....	81
Application.CreateLibrary Method.....	82
Application.ExportLibraryItems Method.....	83
Application.GetConfigParamInt Method.....	84
Application.GetConfigParamString Method.....	85
Application.GetLibraryItems Method.....	86
Application.LockServer Method.....	87
Application.Measure Method.....	88
Application.ModelessCmd Method.....	89
Application.OpenDocument Method.....	90
Application.OpenDocumentNoLock Method.....	91
Application.OpenTempDocument Method.....	92
Application.Quit Method.....	93
Application.RunMacro Method.....	94
Application.UnlockServer Method.....	95
Application.OpenDocument Event.....	96
Application.ProgressChange Event.....	97
Application.Quit Event.....	98
The AssemblyOptions Collection Object.....	99
AssemblyOptions.Application Property.....	100
AssemblyOptions.Count Property.....	101
AssemblyOptions.Item Property.....	102
AssemblyOptions.ItemType Property.....	103

AssemblyOptions.Next Property.....	104
AssemblyOptions.ObjectType Property.....	105
AssemblyOptions.Parent Property.....	106
AssemblyOptions.ParentObject Property.....	107
AssemblyOptions.Add Method.....	108
AssemblyOptions.Delete Method.....	109
AssemblyOptions.Merge Method.....	110
AssemblyOptions.Remove Method.....	111
AssemblyOptions.Reset Method.....	112
AssemblyOptions.Select Method.....	113
AssemblyOptions.Sort Method.....	114
The AssociatedNet Object.....	115
AssociatedNet.Name Property.....	116
AssociatedNet.Nets Property.....	117
AssociatedNet.ObjectType Property.....	118
AssociatedNet.Parent Property.....	119
AssociatedNet.Selected Property.....	119
The Attribute Object.....	120
Attribute.Application Property.....	121
Attribute.Name Property.....	122
Attribute.ObjectType Property.....	124
Attribute.Parent Property.....	125
Attribute.Value Property.....	126
Attribute.Measure Method.....	129
The Attributes Collection Object.....	130
Attributes.Application Property.....	131
Attributes.Count Property.....	132
Attributes.Item Property.....	133
Attributes.ItemType Property.....	134
Attributes.Next Property.....	135
Attributes.ObjectType Property.....	136
Attributes.Parent Property.....	137
Attributes.ParentObject Property.....	138
Attributes.Add Method.....	139
Attributes.Delete Method.....	140
Attributes.Merge Method.....	142
Attributes.Remove Method.....	143
Attributes.Reset Method.....	144
Attributes.Select Method.....	145
Attributes.Sort Method.....	146
The CBP Object.....	147
CBP.Application Property.....	147
CBP.Edge Property.....	148
CBP.Function Property.....	149
CBP.Layer Property.....	150
CBP.Length Property.....	151
CBP.Name Property.....	152
CBP.ObjectType Property.....	153
CBP.Parent Property.....	154

Table of Contents

CBP.PositionX Property.....	155
CBP.PositionY Property.....	156
CBP.SBPs Property.....	157
CBP.Shape Property.....	158
CBP.Width Property.....	159
CBP.Wirebonds Property.....	160
CBP.Component Method.....	161
The Circle Object.....	162
Circle.Application Property.....	163
Circle.CenterX Property.....	164
Circle.CenterY Property.....	166
Circle.Geometry Property.....	168
Circle.Layer Property.....	169
Circle.LineStyle Property.....	170
Circle.LineWidth Property.....	171
Circle.ObjectType Property.....	172
Circle.OutlineType Property.....	173
Circle.Parent Property.....	174
Circle.Radius Property.....	175
Circle.ShapeType Property.....	176
The Component Object.....	178
Component.Application Property.....	180
Component.Attributes Property.....	181
Component.CBPs Property.....	181
Component.CenterX Property.....	183
Component.CenterY Property.....	184
Component.Decal Property.....	185
Component.DecalAttributes Property.....	187
Component.DecalCompatibleList Property.....	188
Component.DieHeight Property.....	189
Component.DieLength Property.....	189
Component.DieWidth Property.....	190
Component.Glued Property.....	192
Component.Installed Property.....	193
Component.IsDiePart Property.....	194
Component.IsSMD Property.....	195
Component.Labels Property.....	196
Component.Layer Property.....	197
Component.Name Property.....	198
Component.ObjectType Property.....	200
Component.Orientation Property.....	202
Component.Parent Property.....	203
Component.PartType Property.....	204
Component.PartTypeAttributes Property.....	205
Component.PartTypeECORegistered Property.....	206
Component.PartTypeLogic Property.....	207
Component.PartTypeObject Property.....	208
Component.Pins Property.....	209
Component.Placed Property.....	210

Component.PositionX Property.....	211
Component.PositionY Property.....	212
Component.SBPs Property.....	212
Component.Selected Property.....	214
Component.Substituted Property.....	215
Component.WireBondRulesAngleMaximum Property.....	215
Component.WireBondRulesClearanceWireToPad Property.....	216
Component.WireBondRulesClearanceWireToWire Property.....	217
Component.WireBondRulesLengthMaximum Property.....	218
Component.WireBondRulesLengthMinimum Property.....	219
Component.Wirebonds Property.....	220
Component.AddLabel Method.....	222
Component.Move Method.....	224
Component.MoveCenter Method.....	226
The Connection Object.....	227
Connection.Application Property.....	228
Connection.Length Property.....	229
Connection.Name Property.....	230
Connection.Net Property.....	232
Connection.ObjectType Property.....	233
Connection.Parent Property.....	235
Connection.Pins Property.....	236
Connection.RouteSegments Property.....	237
Connection.Selected Property.....	238
Connection.Vias Property.....	239
The Decal Object.....	240
Decal.Application Property.....	241
Decal.Attributes Property.....	242
Decal.Components Property.....	243
Decal.LibraryTimeStamp Property.....	244
Decal.Name Property.....	245
Decal.ObjectType Property.....	246
Decal.Parent Property.....	247
Decal.Selected Property.....	248
Decal.TimeStamp Property.....	249
The Document Object.....	250
Document.ActiveLayer Property.....	252
Document.ActiveView Property.....	253
Document.Application Property.....	254
Document.AssemblyOptions Property.....	255
Document.AssociatedNets Property.....	257
Document.Attributes Property.....	258
Document.BoardOutlineSurface Property.....	259
Document.Components Property.....	261
Document.Connections Property.....	262
Document.Drawings Property.....	264
Document.ElectricalLayerCount Property.....	265
Document.Errors Property.....	266
Document.FullName Property.....	267

Table of Contents

Document.GridX Property.....	268
Document.GridY Property.....	269
Document.Jumpers Property.....	270
Document.LayerCount Property.....	272
Document.LayerEnabled Property.....	273
Document.LayerName Property.....	274
Document.Layers Property.....	275
Document.LayerType Property.....	277
Document.MaxRealValue Property.....	278
Document.MinRealValue Property.....	279
Document.Name Property.....	280
Document.NetClasses Property.....	281
Document.Nets Property.....	282
Document.ObjectType Property.....	283
Document.OriginX Property.....	284
Document.OriginY Property.....	285
Document.Parent Property.....	286
Document.PartTypes Property.....	287
Document.Path Property.....	288
Document.Pins Property.....	289
Document.Preference Property.....	290
Document.RouteSegments Property.....	291
Document.Saved Property.....	292
Document.Texts Property.....	293
Document.Unit Property.....	294
Document.Vias Property.....	295
Document.Activate Method.....	296
Document.AddText Method.....	297
Document.CheckASCII Method.....	298
Document.ExportASCII Method.....	299
Document.ExportECOFile Method.....	300
Document.ExportNetList Method.....	301
Document.ExportRules Method.....	302
Document.GetColor Method.....	303
Document.GetObjects Method.....	305
Document.GetVisibility Method.....	308
Document.ImportECOFile Method.....	309
Document.ImportNetList Method.....	310
Document.IntegrityTest Method.....	311
Document.Save Method.....	312
Document.SaveAs Method.....	313
Document.SaveAsNoLock Method.....	314
Document.SaveNoLock Method.....	315
Document.SaveAsTemp Method.....	316
Document.SaveTemp Method.....	317
Document.SelectObjects Method.....	318
Document.SetColor Method.....	320
Document.SetVisibility Method.....	322
Document.PositionsChange Event.....	323

Document.Save Event.....	324
Document.SecurityLimit Event.....	325
Document.SelectionChange Event.....	326
The Drawing Object.....	327
Drawing.Application Property.....	328
Drawing.DrawingType Property.....	329
Drawing.Geometry Property.....	331
Drawing.Name Property.....	332
Drawing.Net Property.....	333
Drawing.ObjectType Property.....	334
Drawing.Parent Property.....	335
Drawing.PositionX Property.....	336
Drawing.PositionY Property.....	337
Drawing.Selected Property.....	338
Drawing.Texts Property.....	339
The Error Object.....	340
Error.ActualValue Property.....	341
Error.Application Property.....	343
Error.Conflicts Property.....	344
Error.Description Property.....	345
Error.ErrorClass Property.....	346
Error.ErrorType Property.....	347
Error.HasActualValue Property.....	348
Error.HasRequiredValue Property.....	349
Error.IsClearanceError Property.....	350
Error.IsConnectivityError Property.....	351
Error.IsHighSpeedError Property.....	352
Error.IsIgnoredFlag Property.....	353
Error.IsInvisibleFlag Property.....	354
Error.IsLatiumError Property.....	355
Error.IsMiscError Property.....	356
Error.IsTestPointError Property.....	357
Error.LayerNumber Property.....	358
Error.Name Property.....	359
Error.ObjectType Property.....	360
Error.Parent Property.....	361
Error.PositionX Property.....	362
Error.PositionY Property.....	363
Error.RequiredValueMax Property.....	363
Error.RequiredValueMin Property.....	363
The ErrorConflict Object.....	364
ErrorConflict.Application Property.....	365
ErrorConflict.ConflictObject Property.....	366
ErrorConflict.ConflictObjectDesc Property.....	367
ErrorConflict.ConflictObjectType Property.....	368
ErrorConflict.Name Property.....	369
ErrorConflict.ObjectType Property.....	370
ErrorConflict.Parent Property.....	371
The Jumper Object.....	372

Table of Contents

Jumper.Application Property.....	373
Jumper.Installed Property.....	374
Jumper.Length Property.....	375
Jumper.Name Property.....	376
Jumper.Net Property.....	378
Jumper.ObjectType Property.....	379
Jumper.Orientation Property.....	381
Jumper.Parent Property.....	382
Jumper.Points Property.....	383
Jumper.Selected Property.....	384
The Label Object.....	385
Label.Application Property.....	386
Label.Attribute Property.....	387
Label.Component Property.....	388
Label.Display Property.....	389
Label.Name Property.....	391
Label.ObjectType Property.....	392
Label.Parent Property.....	393
Label.RightReading Property.....	394
Label.Selected Property.....	396
Label.Text Property.....	397
Label.Type Property.....	398
Label.Delete Method.....	400
The Layer Object.....	401
Layer.Application Property.....	402
Layer.CopperThickness Property.....	403
Layer.Enabled Property.....	404
Layer.Name Property.....	405
Layer.Number Property.....	406
Layer.ObjectType Property.....	407
Layer.Parent Property.....	408
Layer.PlaneType Property.....	409
Layer.RoutingDirection Property.....	410
Layer.Type Property.....	411
Layer.Visible Property.....	412
Layer.GetColor Method.....	413
Layer.GetDielectricConstant Method.....	414
Layer.GetDielectricThickness Method.....	415
Layer.GetDielectricType Method.....	416
Layer.SetColor Method.....	417
Layer.SetDielectricConstant Method.....	418
Layer.SetDielectricThickness Method.....	419
Layer.SetDielectricType Method.....	420
The Library Object.....	421
Library.Application Property.....	422
Library.FullName Property.....	423
Library.Name Property.....	424
Library.ObjectType Property.....	425
Library.Parent Property.....	426

Library.Path Property.....	427
Library.GetLibraryItems Method.....	428
Library.ImportLibraryItems Method.....	429
Library.ImportLibraryItems2 Method.....	430
The LibraryItem Object.....	431
LibraryItem.Application Property.....	432
LibraryItem.Library Property.....	433
LibraryItem.Name Property.....	434
LibraryItem.ObjectType Property.....	435
LibraryItem.Parent Property.....	436
LibraryItem.Type Property.....	437
The Measure Object.....	439
Measure.Application Property.....	440
Measure.Name Property.....	441
Measure.Number Property.....	442
Measure.Normalize Property.....	443
Measure.ObjectType Property.....	444
Measure.Parent Property.....	445
Measure.Prefix Property.....	446
Measure.Text Property.....	447
Measure.Unit Property.....	448
Measure.Value Property.....	449
The Net Object.....	452
Net.Application Property.....	453
Net.AssociatedNet Property.....	454
Net.Attributes Property.....	455
Net.Connections Property.....	456
Net.Drawings Property.....	457
Net.Length Property.....	458
Net.Name Property.....	459
Net.NetClass Property.....	461
Net.NetClassAttributes Property.....	462
Net.ObjectType Property.....	463
Net.Parent Property.....	465
Net.Pins Property.....	466
Net.Power Property.....	467
Net.Selected Property.....	468
Net.Vias Property.....	469
The NetClass Object.....	470
NetClass.Application Property.....	471
NetClass.Attributes Property.....	472
NetClass.Name Property.....	473
NetClass.Nets Property.....	474
NetClass.ObjectType Property.....	475
NetClass.Parent Property.....	476
The Objects Collection Object.....	477
Objects.Application Property.....	478
Objects.Count Property.....	479
Objects.Item Property.....	480

Table of Contents

Objects.ItemType Property.....	482
Objects.Next Property.....	483
Objects.ObjectType Property.....	484
Objects.Parent Property.....	485
Objects.ParentObject Property.....	486
Objects.Add Method.....	487
Objects.Merge Method.....	488
Objects.Remove Method.....	489
Objects.Reset Method.....	490
Objects.Select Method.....	491
Objects.Sort Method.....	492
The Pad Object.....	493
Pad.Application Property.....	494
Pad.CornerRadius Property.....	495
Pad.CornerType Property.....	496
Pad.Diameter Property.....	497
Pad.InnerDiameter Property.....	498
Pad.Length Property.....	499
Pad.Name Property.....	500
Pad.ObjectType Property.....	501
Pad.Offset Property.....	502
Pad.Orientation Property.....	503
Pad.PadStackLayer Property.....	504
Pad.Parent Property.....	505
Pad.Shape Property.....	506
Pad.Width Property.....	507
The PadStackLayer Object.....	508
PadStackLayer.AntiPad Property.....	509
PadStackLayer.Application Property.....	510
PadStackLayer.Name Property.....	511
PadStackLayer.Number Property.....	512
PadStackLayer.ObjectType Property.....	513
PadStackLayer.Pad Property.....	514
PadStackLayer.Parent Property.....	515
PadStackLayer.Pin Property.....	516
PadStackLayer.ThermalPad Property.....	517
PadStackLayer.Via Property.....	518
The PartType Object.....	519
PartType.Application Property.....	520
PartType.Attributes Property.....	521
PartType.Components Property.....	522
PartType.ECOWRegistered Property.....	523
PartType.Logic Property.....	524
PartType.Name Property.....	525
PartType.ObjectType Property.....	528
PartType.Parent Property.....	530
PartType.Selected Property.....	531
The Pin Object.....	532
Pin.Application Property.....	533

Pin.Attributes Property.....	534
Pin.Component Property.....	535
Pin.DrillSize Property.....	536
Pin.ElectricalType Property.....	537
Pin.FunctionName Property.....	539
Pin.Glued Property.....	540
Pin.Highlighted Property.....	541
Pin.IsSMD Property.....	542
Pin.Name Property.....	543
Pin.Net Property.....	545
Pin.Number Property.....	546
Pin.ObjectType Property.....	547
Pin.PadStackLayers Property.....	549
Pin.Parent Property.....	552
Pin.PlaneThermal Property.....	553
Pin.Plated Property.....	554
Pin.PositionX Property.....	555
Pin.PositionY Property.....	556
Pin.Selected Property.....	557
Pin.SlotLength Property.....	558
Pin.SlotOffset Property.....	559
Pin.SlotOrientation Property.....	560
Pin.TestPoint Property.....	561
The Polyline Object.....	562
Polyline.Application Property.....	563
Polyline.CenterX Property.....	564
Polyline.CenterY Property.....	566
Polyline.Geometry Property.....	568
Polyline.Layer Property.....	569
Polyline.LineStyle Property.....	570
Polyline.LineWidth Property.....	571
Polyline.ObjectType Property.....	572
Polyline.OutlineType Property.....	573
Polyline.Parent Property.....	575
Polyline.Points Property.....	576
Polyline.Radius Property.....	578
Polyline.ShapeType Property.....	580
The RouteSegment Object.....	582
RouteSegment.Application Property.....	583
RouteSegment.Layer Property.....	584
RouteSegment.Length Property.....	585
RouteSegment.Name Property.....	586
RouteSegment.Net Property.....	588
RouteSegment.ObjectType Property.....	589
RouteSegment.Parent Property.....	591
RouteSegment.Points Property.....	592
RouteSegment.SegmentType Property.....	593
RouteSegment.Selected Property.....	595
RouteSegment.Width Property.....	596

Table of Contents

The SBP Object.....	597
SBP.Application Property.....	597
SBP.CBPs Property.....	598
SBP.Component Property.....	599
SBP.Function Property.....	599
SBP.Layer Property.....	600
SBP.Length Property.....	601
SBP.Name Property.....	602
SBP.ObjectType Property.....	603
SBP.Orientation Property.....	604
SBP.Parent Property.....	605
SBP.Position X Property.....	606
SBP.PositionY Property.....	607
SBP.Shape Property.....	608
SBP.Tier Property.....	609
SBP.Width Property.....	610
SBP.Wirebonds Property.....	611
The Text Object.....	613
Text.Application Property.....	614
Text.Drawing Property.....	615
Text.Height/Label.Height Property.....	616
Text.HorzJustification/Label.HorzJustification Property.....	618
Text.Layer/Label.Layer Property.....	620
Text.LineWidth/Label.LineWidth Property.....	622
Text.Mirror/Label.Mirror Property.....	624
Text.Name Property.....	626
Text.ObjectType Property.....	627
Text.Orientation/Label.Orientation Property.....	628
Text.Parent Property.....	630
Text.PositionX/Label.PositionX Property.....	631
Text.PositionY/Label.PositionY Property.....	633
Text.Selected Property.....	635
Text.Text Property.....	636
Text.VertJustification/Label.VertJustification Property.....	637
Text.Delete Method.....	639
The ThermalPad Object.....	640
ThermalPad.Application Property.....	641
ThermalPad.CornerRadius Property.....	642
ThermalPad.CornerType Property.....	643
ThermalPad.InnerLength Property.....	644
ThermalPad.InnerSize Property.....	645
ThermalPad.Name Property.....	646
ThermalPad.ObjectType Property.....	647
ThermalPad.Offset Property.....	648
ThermalPad.Orientation Property.....	649
ThermalPad.OuterSize Property.....	650
ThermalPad.PadStackLayer Property.....	651
ThermalPad.Parent Property.....	652
ThermalPad.Shape Property.....	653

ThermalPad.SpokeAngle Property.....	654
ThermalPad.Spokes Property.....	655
ThermalPad.SpokeWidth Property.....	656
The Via Object.....	657
Via.Application Property.....	658
Via.Attributes Property.....	659
Via.DrillSize Property.....	660
Via.EndLayer Property.....	661
Via.Glued Property.....	662
Via.Highlighted Property.....	663
Via.Name Property.....	664
Via.Net Property.....	666
Via.ObjectType Property.....	667
Via.PadStackLayers Property.....	669
Via.Parent Property.....	671
Via.PlaneThermal Property.....	672
Via.Plated Property.....	673
Via.PositionX Property.....	674
Via.PositionY Property.....	675
Via.Selected Property.....	676
Via.StartLayer Property.....	677
Via.Stitching Property.....	678
Via.TestPoint Property.....	679
Via.Type Property.....	680
The View Object.....	681
View.Application Property.....	682
View.BottomRightX Property.....	683
View.BottomRightY Property.....	684
View.CenterX Property.....	685
View.CenterY Property.....	686
View.Name Property.....	687
View.ObjectType Property.....	688
View.Parent Property.....	689
View.PointerX Property.....	690
View.PointerY Property.....	691
View.TopLeftX Property.....	692
View.TopLeftY Property.....	693
View.Zoom Property.....	694
View.Pan Method.....	695
View.Refresh Method.....	696
View.SetExtents Method.....	697
View.SetExtentsToAll Method.....	699
View.SetExtentsToBoard Method.....	700
View.SetExtentsToSelection Method.....	701
View.SetScale Method.....	702
View.Change Event.....	703
The Wirebond Object.....	704
Wirebond.Angle Property.....	704
Wirebond.Application Property.....	705

Table of Contents

Wirebond.Component Property.....	706
Wirebond.EndOffsetX Property.....	706
Wirebond.EndOffsetY Property.....	707
Wirebond.EndPad Property.....	708
Wirebond.EndX Property.....	709
Wirebond.EndY Property.....	710
Wirebond.Name Property.....	711
Wirebond.ObjectType Property.....	712
Wirebond.Parent Property.....	713
Wirebond.StartOffsetX Property.....	714
Wirebond.StartOffsetY Property.....	715
Wirebond.StartPad Property.....	716
Wirebond.StartX Property.....	717
Wirebond.StartY Property.....	718
Constants.....	719
Optional Argument.....	732
Variant.....	733
Exception.....	733
Samples.....	734
Code Samples.....	734
Running Code Samples in SailWind Layout.....	734
Running Code Samples Outside of SailWind Layout.....	735
Sample Code Enhancement.....	736
Troubleshooting Code Samples.....	737
RGL Format Files.....	738
Converted RGL Format Files.....	738
Automation for RGL Top Level Keywords.....	739
Automation for RGL SubLevel Keywords.....	740
Automation for RGL Field Keywords.....	744
New Automation Functions for RGL.....	745

Chapter 2

ASCII Format.....	747
ASCII Messages.....	747
ASCII Parts and Connections Lists	749
Units Used in ASCII Parts and Connections Lists.....	749
Parts List Section.....	750
Minimum Parts List Content.....	750
The *Standard* Header.....	752
Notes on Parts List Format.....	752
Sample Parts List.....	753
Notes on Parts List Files.....	754
Parts List Error Messages.....	755
Connection List.....	756
Mandatory Connection List Content.....	756
Reference Designator Format.....	757
Signal Statements and To-From Signal Names.....	757
Discrete Component Pad Numbering.....	758

Tying Nets Together.....	758
Abbreviated Notations.....	758
Width.....	759
Sample Connection List.....	759
Connection List Error Messages.....	760
 Chapter 3	
Macros and the Macro Language.....	761
Macros.....	762
Macro Text Colors.....	762
Macro Editing Shortcuts.....	762
Design Work Recordings in Macro Files.....	764
Recording a Macro of an Entire Software Session.....	764
Recording a Macro of Selective Actions in the Design Area.....	764
Replay Recorded Macros.....	766
Running a Macro When You Start the Program.....	766
Running Macros From Within the Software.....	766
Assigning Shortcut Keys to Macros.....	767
Adding Macros to Toolbars and Menus.....	768
Macro Editing and Debugging.....	769
Toggling Breakpoints.....	769
Macro Debugging Tools.....	769
The Macro Language.....	771
Variables.....	771
Expressions.....	773
Operators.....	774
& Operator.....	775
* Operator.....	776
+ Operator.....	777
/ Operator.....	778
- Operator.....	779
= Operator.....	780
^ Operator.....	781
And Operator.....	782
Comparison Operators.....	783
Mod Operator.....	784
Not Operator.....	785
Or Operator.....	786
Xor Operator.....	787
Statements.....	788
Call.....	789
Close	790
Dim.....	791
Do...Loop.....	792
For-Next.....	793
Function.....	794
If...Then...Else statement.....	797
Input #.....	799

Table of Contents

Modal.....	800
Open.....	801
Print #.....	802
ReDim.....	804
Set.....	806
Sub.....	806
While...Wend.....	809
Width #.....	810
Functions.....	811
Asc.....	812
Atn.....	812
Chr.....	814
Command.....	815
Cos.....	816
CreateObject.....	817
CurDir.....	818
Dir.....	819
DoEvents.....	820
Environ.....	821
Eof.....	821
Exp.....	823
GetObject.....	824
GetTmpFileName.....	825
InStr.....	826
InStrRev.....	827
Left.....	828
Len.....	829
Mid.....	830
MkDir.....	831
MoveFile.....	832
MsgBox.....	833
Right.....	836
Sin.....	836
Spc.....	838
Str.....	839
Tab.....	840
Val.....	841
Automation Support.....	841
Dialog Box Controls.....	842
CheckBox.State.....	844
CheckBox.Value.....	845
CheckListBox.State.....	846
CheckListBox.SetCheck.....	847
CheckListBox.ListCount.....	848
CheckListBox.SelCount.....	849
CheckListBox.Selected.....	850
CheckListBox.Check.....	851
CheckListBox.Text.....	852
ComboBox.Select.....	853

ComboBox.Edit.....	854
ComboBox.Text.....	855
ComboBox.List.....	856
ComboBox.SelStart.....	857
ComboBox.SelLength.....	858
ComboBox.SelText.....	859
EditBox.State.....	860
EditBox.Text.....	861
EditBox.SelStart.....	862
EditBox.SelLength.....	863
EditBox.SelText.....	864
GridControl.....	864
ListBox.State.....	865
ListBox.List.....	866
ListBox.ListCount.....	867
ListBox.SelCount.....	868
ListBox.Selected.....	869
ListBox.Text.....	870
PushButton.....	871
RadioBox.State.....	872
RadioBox.Value.....	873
SliderControl.State.....	874
SliderControl.Value.....	875
SpinButton.State.....	876
TabControl.State.....	877
TabControl.Value.....	878
Treeltem.Select.....	879
Treeltem.Expand.....	880
Treeltem.Focus.....	881
TreeView.Item.....	882
TreeView.BeginDrag.....	883
TreeView.Copy.....	884
TreeView.Drop.....	885
TreeView.Paste.....	886
TreeView.CreateNewItem.....	887
Internal Macro Objects.....	888
Application Object.....	889
CreateNewDocument.....	890
ExecuteCommand.....	891
Help.....	892
HelpContents.....	893
HelpPane.....	894
OpenCustomizeDialog.....	895
OpenDocument.....	896
OpenOptionsDialog.....	897
OpenPropertiesDialog.....	898
Quit.....	899
RunMacro.....	900
Dialog Objects.....	901

Table of Contents

CloseHelpPane.....	902
Control.....	903
Focus.....	904
OpenHelpPane.....	905
ShowHelpFor.....	906
Document Object.....	907
Print.....	908
PrintSetup.....	909
RepeatLastAction.....	910
Save.....	911
SaveAs.....	912
Application.HelpContents.Item Object.....	913
HelpContentsItem Object.....	914
Location.....	915
Name.....	916
Select.....	917
SubItem.....	918
SubItemCount.....	919
HelpPane Object.....	919
Main View Object.....	920
ActiveLayer.....	921
ToggleFullScreen.....	922
MouseDown.....	923
MouseEndDrag.....	924
MouseMove.....	925
MouseStartDrag.....	927
MouseUp.....	928
Print.....	929
PrintPreview.....	930
Chapter 4	
Intermediate Data Format.....	931
IDF File Export.....	932
3D Data Export.....	932
Part Height Information.....	932
Considerations for Exporting Holes.....	933
Part Outline Information.....	933
IDF Export Messages.....	934
IDF Export Status File.....	935
Height Messages.....	935
Outline Messages.....	936
Keepout Messages in the IDF Export Status File.....	936
IDF File Import.....	937
Hole Import.....	937
Component Addition During Import.....	938
Part Name Incompatibility.....	938
Illegal Character Incompatibility.....	939
IDF Import Messages.....	939

IDF Import Status File.....	941
Drilled Hole Messages.....	941
Component Messages.....	942
Keepout Messages.....	943
Cutout Messages.....	943
Protected Trace Messages.....	943
Physical Design Reuse Messages.....	944
IDF File Format.....	945
Panel Files.....	945
Board Outline.....	945
Other Outline.....	945
Routing Outline.....	946
Placement Outline.....	946
Units.....	946
Routing Keepout.....	946
Via Keepout.....	947
Placement Keepout.....	947
Placement Group Area.....	947
Drilled Holes.....	947
Component Placement.....	947
Library File Details.....	948
Part Outlines.....	949
IDF 3.0 Enhancements.....	949

Chapter 5

ECO File Commands.....951

File Formats for Passing Data Between SailWind Layout and the Schematic Tool.....	953
Modification Data Types.....	953
Header and End of File Formats.....	954
Add Pin to Net Format.....	955
Delete Pin from Net Format.....	955
Exclude Nets Format.....	956
Join Two Nets Together Format.....	956
Exclude Connections Format.....	957
Split Net into Two New Nets Format.....	957
Rename Net Format.....	958
Add Part Format.....	958
Delete Part Format.....	958
Rename Part Format.....	959
Change Part Type Format.....	959
Change Decal Format.....	960
Move Part Format.....	960
Change Design Origin Format.....	961
Swap Two Gates Format.....	961
Swap Two Pins Format.....	962
General Attributes Format.....	962
Rule Change Formats.....	965
Add Object Assignments to a Rule Set (General Rules Only).....	965

Table of Contents

Delete Object Assignments from a Rule Set (General Rules Only).....	966
Changes in Values for a Rule.....	966
Add a Rule.....	967
Split a Rule.....	967
Sample Backward Annotation File.....	967
The Format of Rule Commands.....	971
The Formatting for Design Rules in the ECO File.....	971
The Formatting for Conditional Clearance Rules in the ECO File.....	972
The Formatting for Conditional High-Speed Rules in the ECO File.....	973
The Formatting for Differential Pairs Rules in the ECO File.....	974
The Formatting for General Clearance Rules in the ECO File.....	974
The Formatting for General High-Speed Rules in the ECO File.....	975
The Formatting for General Routing Rules in the ECO File.....	976
The Formatting for Net Classes Rules in the ECO File.....	976
The Formatting for Pin Pair Group Rules in the ECO File.....	977
Example ECO File.....	978

Chapter 6

Report Generation Language..... 983

New Reports.....	983
Searches and Loops.....	983
Format File Structure.....	984
Nested Loops.....	985
Keywords.....	987
Top Level Keywords.....	987
Sublevel Keywords.....	988
Field Keywords.....	990

Chapter 7

CAM Plus Supported Machine Formats..... 991

Siemens HS-180.....	991
Siemens MS-72k.....	992
Quad 100.....	993
Daum SMF.....	993
Zevatech PPM-9.....	995
Universal Machine Output Files.....	996
Universal Sequencer File.....	997
Philips CSM Format.....	998
Parts List.....	999
Part Definition File Format.....	1001
Format of the part.def File.....	1001
Machine Codes.....	1001
Bin Assignments.....	1004

Chapter 1

Scripting and the Automation Server

SailWind Layout allows you to create custom scripts for automating certain functions.

[Scripts](#)

[Automation in a Multiple Release Environment](#)

[Basic Sample Scripts](#)

[Basic Sample Scripts/RGL Reports](#)

[Script Creation](#)

[Script Implementation](#)

[Managing Scripts](#)

[Debugging Scripts](#)

[The Automation Server](#)

[SailWind Layout Automation Samples](#)

[SailWind Layout Automation Server Reference](#)

[RGL Format Files](#)

Scripts

SailWind Layout includes an internal scripting capability, the Sax Basic Engine, by Sax Software Corporation. The Sax Basic Engine makes the Automation features within SailWind Layout more accessible to everyone—developer and nondeveloper. This engine includes the entire development environment required to develop Basic scripts, including editor, debugger, interpreter, tracer, variable watch, dialog editor, Automation object browser, and new/open/save/print capabilities. These capabilities ensure that every user can work with Basic scripting without previous programming knowledge.

Basic is a scripting language developed to provide users with a unified language in Windows. More and more Windows applications like SailWind Layout include Basic capabilities, such as Microsoft Word and Microsoft Excel, to let you customize your application with a standard scripting language.

Scripts written with the editor comply with all Microsoft requirements in terms of Basic syntax and, therefore, you can play these scripts in any other Basic interpreter, such as Word or Excel. However, you cannot run Basic scripts created outside of the Sax Basic Engine within the Sax Basic Engine because the Sax Basic Engine is a subset of Basic. You cannot, for example, run the Automation samples within the Sax Basic Engine.

You can create a script that calls another script. For example, ScriptA can call ScriptB. A script can also call a SailWind Layout macro using the RunMacro() Automation call. A SailWind Layout macro, however, cannot call a script.

You can also create a script that runs a series of scripts, or a "master" script. For example:

```
'$Include: "scriptA.bas"
'$Include: "scriptB.bas"
'$Include: "scriptC.bas"
Sub Main
  Call scriptA
```

```
Call scriptB  
Call scriptC  
End Sub
```

As shown in the table in [Table 1](#), the editor displays source code using different colors.

Table 1. Text Colors Representation

Color	Represents
Blue	Basic Keywords
Black	User Variables
Cyan	Basic Functions
Purple	SailWind Layout Automation Objects or Members
Red	Errors
Green	Description

The color is context-sensitive. That is, when you place the cursor on the text and press F1, the correct help file opens to the correct help topic. For example, if the cursor is on a SailWind Layout Automation Object when you press F1, the SailWind Layout Automation Help appears.

For more information on the Basic editor and language, see the Sax Basic Editor Help. In the editor, right-click, point to Help, and then click Editor Help or Language Help.

In addition to the editor, SailWind Layout has a script manager that lets you manage the existing Basic scripts listed in [Basic Sample Scripts](#). Using this script manager, you can load your most frequently used scripts and run them independently of the editor itself.

Automation in a Multiple Release Environment

To use automation in an environment where multiple SailWind Software releases exist on the same machine, you must know and apply the correct COM version for each of the software installs.

- COM automation scripts — These scripts extend the functionality of the main authoring applications (for example, SailWind Layout). These scripts connect to an instance of a tool to access its functionality. That functionality is contained in COM objects within the applications. It is these COM objects that are registered by the Registrator.
- COM objects — These objects encapsulate product functionality and are referenced in user-created COM automation scripts. Because product functionality differs between releases, the COM objects associated with each release have unique versions. For example, (the COM number used in this example are for the example only and do not reflect the true COM numbers of the releases mentioned) any PADS 9.5 release object is version 1, a X-ENTP VX.2.3 object is version 2, a PADS-Pro VX.2.3 object is version 3, a PADS VX.2.3 object is version 4, etc. Versioning the objects enables the product functionality for each release to be available to customer-created scripts. Registering these COM objects is a primary function of the Registrator (PADS VX.2.3) and the Configurator (PADS9.5).

Basic Sample Scripts

SailWind Layout includes some basic sample scripts.

Table 2. Basic Sample Scripts

	Script	Description
00	What is a VB Script.BAS	Empty script demonstrating what a Basic script is and how to define it.
01	Using a Message Box.BAS	Demonstrates how to display an OK dialog box.
02	Using a Variable.BAS	Demonstrates a how to assign a value to a variable and how to get its value.
03	Using a VB Function.BAS	Demonstrates how to use a standard Basic function and display its result in a message box.
04	Using a SailWind Layout Function.BAS	Demonstrates Basic interaction with a SailWind Layout Automation function.
05	Using If and Then Statements.BAS	Demonstrates If, Then statements.
06	Using a Custom Dialog1.BAS	Demonstrates a simple dialog box using the dialog box editor.
07	Using a Custom Dialog2.BAS	Demonstrates a standard dialog box using the dialog box editor.
08	Using a Custom Dialog3.BAS	Demonstrates a complex dialog box using the dialog box editor.
09	Using It All Together.BAS	Provides a "real life" example. Lists all .pcb files in \SailWind Projects. Selecting a file from the list opens that file in SailWind Layout.
10	List of Comps and Nets.BAS	Lists all components and nets.
11	Select by Pin Count.BAS	Selects all parts that have the number of pins you enter.
12	Move by Pin Count.BAS	Moves all parts that have the number of pins you enter.
13	Width Table.BAS	Lists all possible width values. Selecting a width changes the current width.
14	Part List Report.BAS	Creates a Part List report from the open .pcb or .bga file in Microsoft Excel.
15	Pin List Report.BAS	Creates a Pin List report from the open .pcb or .bga file in Excel.

Table 2. Basic Sample Scripts (continued)

	Script	Description
16	Via List Report.BAS	Creates a Via List report from the open .pcb or .bga file in Excel.
17	Select All Test Points.BAS	Selects all test points.
18	Part Web Search.BAS	Searches major semiconductor manufacturers' Web sites for the selected part's part type.
19	Off-Grid Pins.BAS	Lists all pins and vias that are off the current grid.
20	SailWind Layout Script Wizard.BAS	Generates a Wizard dialog box, which you use to create a Basic report.

Basic Sample Scripts/RGL Reports

SailWind Layout contains scripts that are equivalent to existing Report Generation Language (RGL) reports. These files are located in *C:\SailWind Projects\Samples\Scripts\Layout\RGL_Samples*.

Table 3. Sample Scripts Equivalent to RGL Reports

Script	Purpose
RGL.BAS	Contains a library of functions, which is used by the other scripts in this group; the scripts in this group must contain RGL.BAS to function.
Netlist Without Pin Info.BAS	Reports signals by netname without pin information.
Netlist With Pin Info.BAS	Reports signals by netname with pin information.
Part List 1.BAS	Reports parts by reference designator.
Part List 2.BAS	Reports reference designator by part type.
Test Points.BAS	Reports test point locations and netnames.
Jumper List.BAS	Reports jumper locations and netnames.
PowerPCB V2.0 Format netlist.BAS	Reports a PowerPCB V2.0 format netlist.
PowerPCB V3.0 Format netlist.BAS	Reports a PowerPCB V3.0 format netlist.
DFT Extended Test Point.BAS	Reports test points by nets, nets without test points, and the number of test points per net.

Use the Sax Basic Script and Editor features to design, develop, and run scripts that add to, replace, enhance, or customize existing SailWind Layout features.

- [Creating a New Script](#)
- [Managing Open Scripts](#)
- [Debugging the Scripts](#)
- [Accessing Help on the Basic Language](#)

Script Creation

You can create scripts to simplify redundant activities.

- [Creating a New Script](#)
- [Inserting an Automation Statement](#)
- [Setting the Next Statement](#)
- [Showing the Next Statement](#)
- [Saving the Script](#)

Creating a New Script

You can start the script creation process using the built-in script editor.

Procedure

1. Click the **Tools > Basic Scripts > Basic Script Editor** menu item.
In SailWind Layout and SailWind Logic, the SAX Basic Engine dialog box appears.
2. Click the **New** button.

Inserting an Automation Statement

You can add automation statements to the bottom of a script automatically.

Procedure

1. Use the Object and Procedure lists to select and insert a statement. These lists contain the most commonly used statements.
 - a. Click the **Object** list and click an object type. The Object list shows all the objects for the current module. The (General) object groups all of the procedures that are not part of any specific object.
 - b. Click the **Procedure** list and click a non-bold procedure to insert. The Procedure list shows all the procedures for the current object. Selecting a procedure that is not bold inserts the proper procedure definition for that procedure.

The statement appears at the bottom of the script.

2. Alternatively, for a more extensive list of statements, right-click in the Basic Script Editor and click the **Debug > Browse** menu item. Use the ActiveX Automation Members dialog box to select and insert a statement.



Tip

If the pointer is on any line in the script other than the bottom line, the line is overwritten.

Setting the Next Statement

You can force a particular line in a script to run next. You can only select statements in the current subroutine or function.

Procedure

1. In the Basic Script Editor, put your cursor on the line you want to run next.
2. Right-click and click the **Debug > Set Next Statement** popup menu item.
3. An instruction pointer appears next to the selected line. This line, and only this line, will run next. If you go to other parts of the script, you can return to this line by clicking **Show Next Statement**.

Showing the Next Statement

Use the Basic Script Editor's built-in features to advance from one statement to the next.

Procedure

1. In the Basic Script Editor, right-click and click the **Debug > Show Next Statement** popup menu item.
2. An instruction pointer indicates the next statement to run. Pausing a running script or setting a statement to run next sets the next statement. You can locate the set statement from anywhere in the script.

Saving the Script

Save changes to your script from within the Basic Script Editor.

Procedure

1. In the Basic Script Editor, click the **Save** button.
2. Enter a file name, if necessary, and then click **Save**.

Script Implementation

You can run an existing script using Run. The Run command also resumes the playback of a paused script. When you run a script, you cannot use the mouse in the workspace.

[Running a Script](#)

[Pausing a Running Script](#)

[Stopping a Running Script](#)

Running a Script

You can run a script from the Basic Scripts dialog box, or you can run a run a script from the Basic Script Editor. Running it from the Editor gives you options for pausing and stopping the script.

Procedure

1. From the Basic Scripts dialog box:
 - a. Click the **Tools > Basic Scripts > Basic Scripts** menu item.
 - b. Select a script from the list.
 - c. If the script you want to run does not appear in the list, click **Load File** and browse for the script to load.
 - d. Click **Run**. Scripts are compiled when you run them. You cannot run multiple scripts at the same time.

If the selected script has a compiling error, it automatically opens in the Basic editor for correction.

2. Alternatively, to run a script from the Basic Script Editor:
 - a. In the Basic Script Editor, open a script file.
 - b. Right-click and click the **Macro > Run** menu item, or on the Basic Script Editor toolbar, click the **Start/Resume** button.

Pausing a Running Script

When running a long script, you may need to pause it to perform some other design activity.

Procedure

In the Basic Script Editor, right-click and click the **Macro > Pause** menu item, or on the Basic Script Editor toolbar, click the Pause button.



Tip

If you paused the script, you can also use Run, Step Over, or Step to Cursor to resume running the script. Right-click and select **Run** to resume running the script.

Stopping a Running Script

You can stop a running script at any time. However, you cannot resume running a script after you have stopped it. When you click Run, the script starts from the beginning.

Procedure

In the Basic Script Editor, right-click and click the **Macro > End** popup menu item, or on the Basic Script Editor toolbar, click the **Stop** button.



Tip

While a script is running, a third command, "Stop *scriptname*," appears on the Basic Scripting submenu (on the Tools menu) along with Basic Editor and Basic Scripts. Click Stop *scriptname* to halt the execution of the currently running script.

Managing Scripts

The Basic Script Editor provides tools for managing your scripts.

- [Opening an Existing Script](#)
- [Managing Open Scripts](#)
- [Editing a Script](#)
- [Editing a User Dialog Box](#)
- [Finding an Automation Statement](#)
- [Printing a Script](#)
- [Saving a Script](#)
- [Watching a Variable](#)

Opening an Existing Script

Scripts are created in and stored in script files that have a *.bas* extension. The default location for *.bas* files is *C:\SailWind Projects*.

Procedure

1. In the Basic Script Editor, click the **Open** button.
2. Select the script and then click **Open**.
3. You can have up to nine scripts open at the same time.

Managing Open Scripts

The commands on the Sheet submenu provide script management methods. Since you can have up to nine scripts open at the same time, you can open #uses, close sheets, close multiple sheets, and choose scripts to view and edit.

Procedure

1. The #uses modules are Basic scripts that are called from within other scripts. To open these secondary scripts, right-click in the Basic Script Editor and click the **Sheet > Open Uses** menu item.

The #uses modules called in the script appear as script sheets in the Basic Script Editor. They are assigned a numbered tab and you can edit or run them.

2. To close an open script, right-click in the Basic Script Editor and click the **Sheet > Close** menu item.

Alternatively, you can double-click the script's numbered tab in the gutter.

3. To close all open scripts, right-click in the Basic Script Editor and click the **Sheet > Close All** menu item.

4. The Basic Script Editor allows you to have up to nine scripts open at the same time. To view a particular script, right-click and click the **Sheet** popup menu item. Then click the script you want to view from the list of open scripts on the submenu. Alternatively, you can click the script's numbered tab in the gutter.

Editing a Script

You can copy or cut selected text from the Basic Script Editor to the Clipboard. You can also paste a selection from the Clipboard into the text window. You can also paste text from the Clipboard into other applications.

Procedure

1. In the Basic Script Editor, select the text you want to copy or cut.
2. Right-click and click the **Edit > Copy**(or **Cut**) popup menu item.
3. Right-click and click the **Edit > Paste** popup menu item to paste the script text. Your selection is pasted in the Output window at the insertion point.

Alternative: Click the **Copy**, **Cut**, and **Paste** buttons on the Basic Script Editor toolbar.

Editing a User Dialog Box

A UserDialog is defined by a Begin Dialog...End Dialog block.

Procedure

1. In the Basic Script Editor, put your cursor in a UserDialog block of the script.
2. Click the **Edit UserDialog** button.

For more information, see *Sax Basic Editor On Line Help* (C:\<install_folder>\<version>\Programs\lsbe5_000.hlp)

Finding an Automation Statement

If you are working with a long script, you can search for particular statements.

Procedure

1. In the Basic Script Editor, click the **Object** list and select an object type. The Object list shows all the objects for the current module. The (General) object groups all of the procedures that are not part of any specific object.
2. Click the **Procedure** list and select a bold procedure. The Procedure list shows all the procedures for the current object. Selecting a procedure that is bold locates the procedure in the script.
3. The statement appears in the Basic Script Editor.

Printing a Script

You can print a script from within the Basic Script Editor.

Procedure

1. Open the script in the Basic Script Editor.
2. On toolbar, click the **Print** button.

Saving a Script

You can save a script after making your changes in the Basic Script Editor.

Procedure

1. In the Basic Script Editor, click the **Save** button.
2. Enter a file name, if necessary, and then click **Save**.

Watching a Variable

Quick Watch shows the value of the expression under the cursor in the immediate window.

Procedure

Right-click and click **Quick Watch**, or in the Basic Script Editor, click the Quick Watch button.

Debugging Scripts

When running a script, you can run it step-by-step or to a certain location in the script. To perform these debugging tasks, insert breakpoints in the script at the points at which you want the script to stop.

[Setting or Removing the Breakpoints](#)
[Debugging the Scripts](#)
[Removing All Breakpoints in the Script](#)
[Correcting Run-time Errors](#)
[Accessing Help on the Basic Language](#)

Setting or Removing the Breakpoints

The ability to set or remove breakpoints is useful when you debug a script. If the Basic engine encounters a breakpoint when running a script, it pauses the script.

Procedure

1. Place the cursor on the line to which to add a breakpoint.
2. On the Basic Script Editor toolbar, click the **Toggle Breakpoint** button.

Alternative: In the Basic Editor, right-click and click the **Debug > Toggle Break** popup menu item.

Results

A breakpoint inserts at the current cursor location. A breakpoint marker appears in the gutter area.

When the Basic engine encounters a breakpoint while running a script, it pauses the script. The next line in the script is marked with the instruction pointer.

Debugging the Scripts

Debug your scripts after inserting breakpoints. You can run a single line, subroutine, or otherwise run the script to a specified point.

Procedure

1. To perform a single line, click the **Step over** button on the Basic Script Editor toolbar.
2. To perform a subroutine call on the current line, click the **Step into** button.
Alternative: In the Basic Script Editor, right-click and click the popup menu item.
3. To return from the subroutine to the point from which it was called, click the **Step out** button.
4. To run a script to a point, right-click in the Basic Script Editor and click the **Debug > Step to cursor** popup menu item.
5. To continue the execution from the current point, click the **Run** button.

Alternative: In the Basic Script Editor, right-click and click the **Macro > Run** popup menu item.

Removing All Breakpoints in the Script

You can clear breakpoints in your script when you no longer need them.

Procedure

In the Basic Script Editor, right-click and click the **Debug > Clear All Breaks** popup menu item. This removes all breakpoints in the script.

Correcting Run-time Errors

If run-time errors occur, the script debugger switches to step-by-step mode and displays a detailed message on the status bar.

Procedure

The instruction pointer is set on the line that produced the error. After fixing the error, you can resume running the script.

Accessing Help on the Basic Language

While writing or running scripts, you can access Help that provides information and a sample script using the Basic language statements.

Procedure

1. Select or click in an item in color in the edit area of the Basic Script Editor and then press F1.
2. Help appears for the current statement.

The Automation Server

SailWind Layout contains several Automation Server features.

- [OLE Background](#), a short description of OLE and Automation.
- [The Automation Server Object Hierarchy](#)
- [SailWind Layout Automation Samples](#) showing how to get started with writing Automation clients, in Microsoft Visual Basic, Microsoft Visual C++, and Microsoft Excel

Disclaimer: The code samples in SailWind Layout Automation Server Help are freeware. These samples are provided as a courtesy to its users. Freeware is provided as is and no warranties with respect to freeware are made, either express or implied, including any implied warranties of merchantability or fitness for a particular purpose.

- [SailWind Layout Automation Server Reference](#), listing and detailing each Automation object, property, method, and event in the Automation Server.

[OLE Background](#)

[The Automation Server Object Hierarchy](#)

OLE Background

Technically, OLE stands for Object Linking and Embedding, but that is now an insufficient title. Today, OLE is a continuously growing set of features (including Object Linking and Embedding, Automation, Compound Documents, Visual Editing, ActiveX, DCOM, and so on) based on Microsoft's version of a framework allowing heterogeneous applications to communicate with each other.

Automation is a feature of OLE that defines a protocol for applications to share their data and functionality with any other application using that same protocol. Automation involves a *server* and a *client*. A server is the entity that makes available some data and functionality. A client, also called the controller, is the entity that uses the server's data and functionality. Different people at different organizations can author servers and clients at different times, using different development tools and languages. The only required commonality is that they use the same Automation communication protocol. SailWind Layout is an Automation server because it makes some data (the database) and some functionality (such as opening design files and selecting objects) available to other applications. A client, such as Microsoft Excel, can access SailWind Layout data and functionality using the Automation protocol.

The Automation implementation allows third party companies and users to:

- Integrate their products.
- Expand the SailWind Layout set of functionality.
- Customize existing features.
- Automate tasks using standard scripting languages.

In addition, this can be done independently from release cycles.

[Sample 5](#) on page 48 demonstrates exactly these benefits, as it integrates SailWind Layout and Excel and expands the Report command with clean, customizable Excel reports; giving the end-user a tightly integrated and customizable Excel report solution.

The Automation Server Object Hierarchy

Automation server objects in SailWind Layout conform to a specific hierarchy.

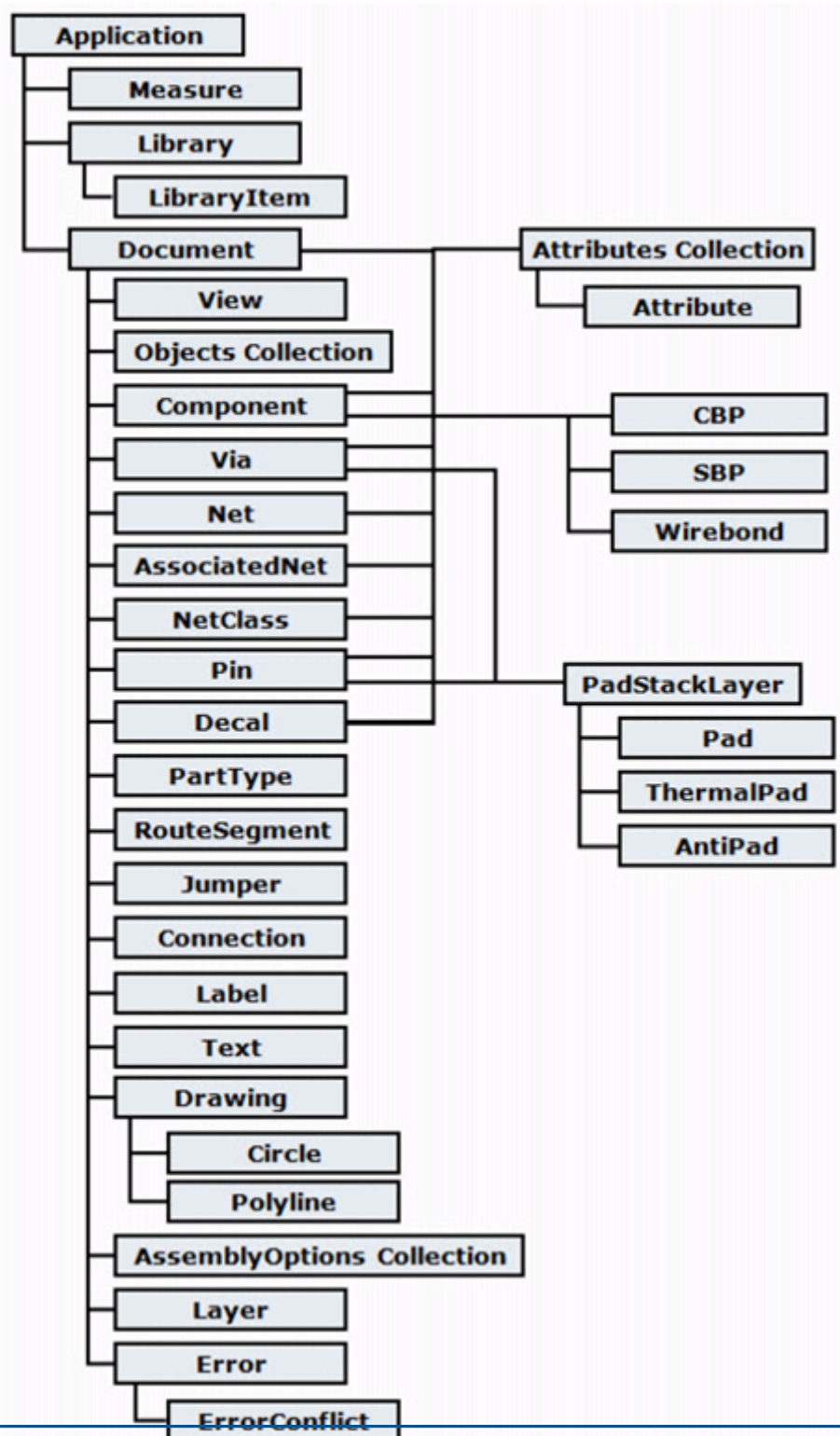


Tip

The CBP, SBP, and Wirebond objects are for the Advanced Packaging Toolkit only and are only applicable if the object is a die part ([Component.IsDiePart Property](#)).

Figure 1. The Automation Server Object Hierarchy

(Click on an item to view its properties, methods and events.)



SailWind Layout Automation Samples

To make it easy for you to develop Automation clients designed for SailWind Layout, or to integrate existing applications, several samples of this technology that you can use and modify are available. These samples are in the C:\SailWind Projects\Samples\Scripts\Layout\samples folder.

- [Sample 1](#) on page 45 details the basics of creating an Automation client for the SailWind Layout server. This sample demonstrates how to connect to an existing or new instance, how to disconnect from it, and how to access simple data.
- [Sample 2](#) on page 46 extends Sample 1 by adding events, which notify clients when the state of SailWind Layout changes.
- [Sample 3](#) on page 46 extends Sample 2 by adding access to the SailWind Layout database.
- [Sample 4](#) on page 47 extends Sample 3 by adding a full-duplex cross-probing feature between the client and SailWind Layout. Sample 4 also adds detailed database information retrieval.
- [Sample 5](#) on page 48 is a full-featured custom plug-in that generates custom “intelligent reports” of the current design in various formats, such as Microsoft Excel spreadsheets, Microsoft Word tables, and HTML pages. Intelligent reports are dynamically linked to the original SailWind Layout data, allowing users to modify component data (such as set position, mounting layer, and glued status) from the generated report.

This sample demonstrates that the Automation Server allows users and third party vendors to develop their own plug-ins, which may dynamically link data with the open design. You can easily extend this sample to include dynamic data links with other applications, such as Microsoft Access and Visual FoxPro.

- [Sample 6](#) on page 51 is a Visual C++ application that displays all SailWind Layout geometric objects, accessible through the Automation interface.



Tip

You need to install the appropriate development tool on your system to run these samples. In other words, to run the Visual Basic version of any of these samples, you need Visual Basic on your system, or to run the Visual C++ version of any of these samples, you need Visual C++ on your system. If you encounter difficulty with one of the samples or with your own client application, see [Troubleshooting Code Samples](#). If you want to create wrapper classes for SailWind Layout objects, you can use your Visual C++ development tool to [create](#) on page 44 them.

The code samples in SailWind Layout Automation Server Help are freeware. These samples are provided as a courtesy to its users. Freeware is provided as is and no warranties with respect to freeware are made, either express or implied, including any implied warranties of merchantability or fitness for a particular purpose.

[Troubleshooting Samples](#)

[Creating Wrapper Classes for SailWind Layout Objects](#)

[Sample 1 - Automation Client](#)

[Sample 2 - Notifications](#)

[Sample 3 - Database Access](#)

[Sample 4 - Cross-Probing](#)

[Sample 5 - A SailWind Layout Custom Reports Command](#)

[Sample 6 - Database Objects Accessed through the Automation Interface](#)

Troubleshooting Samples

If a sample doesn't work as described, you can perform some fundamental checks.

Procedure

1. Ensure sure that only one instance of SailWind Layout is running. By definition, a server is unique. Several instances of the same server may create confusion for clients.
2. Use Task Manager to ensure you do not have another instance of SailWind Layout running in the background. Exiting and restarting SailWind Layout may help in some cases because these programs self-register as Automation servers upon start up.
3. Make sure you disconnect all clients from SailWind Layout before exiting the client application.
4. If the above doesn't help, restart your system.

Creating Wrapper Classes for SailWind Layout Objects

You can use your Visual C++ development tool to automatically create wrapper classes for SailWind Layout objects from the SailWind Layout type library. The SailWind Layout type library is included into SailWind Layout resources, so you can import the type library from the SailWind Layout application file, SailWindPCB.exe.

The instructions in this topic apply to Microsoft Visual C++, version 6.0. If you use another Visual C++ development tool, refer to its documentation for information about importing type libraries.

Procedure

1. In Microsoft Visual C++, open a project.
2. On the View menu, click **ClassWizard**. The MFC ClassWizard dialog box opens.
3. Click **Add Class** and then, from the list that appears, select "From a type library." The Import from Type Library dialog box opens.
4. Set the filter to All Files, browse to *C:\<install_folder>\<version>\Programs\SailWindPCB.exe*, and then click **Open**. The Confirm Classes dialog box opens.
5. Select the classes you want to use and click **OK**.

Results

Microsoft Visual C++ creates the .h and .cpp files containing the wrapper classes.

Sample 1 - Automation Client

The goal for providing Sample 1 is to introduce the basics of writing an Automation client application using the Sax Basic Engine, Microsoft Visual C++ 5.0 (with MFC), Microsoft Visual Basic 5.0, and Microsoft Excel 97. One sample is provided for each of these applications.



Note:

The code samples in SailWind Layout Automation Server Help are freeware. These samples are provided as a courtesy to its users. Freeware is provided as is and no warranties with respect to freeware are made, either express or implied, including any implied warranties of merchantability or fitness for a particular purpose.

Each version of this sample demonstrates, in the respective programming language, how to:

- Prepare the communication between your client and the Automation Server.
- Prepare a pointer to the top level Automation object.
- Connect and disconnect from SailWind Layout, regardless of whether it is running or not.
- Access simple data from SailWind Layout.



Tip

The source code for this sample is in the C:\SailWind Projects\Samples\Scripts\samples\sample1 folder. The code is well commented, and you may use it when developing your own client application.

Specification

Sample 1 is a simple dialog box, which can connect and disconnect at run time from the server. It can connect to an existing running instance or a new instance of SailWind Layout. Once connected, the client retrieves the name of the open design file.

An example of the Visual Basic version of Sample 1 appears below. The Visual C++ version of this sample looks almost identical. The Basic scripting version does not have the Connect and Disconnect buttons because the Sax Basic Engine is already connected to SailWind Layout. The Excel 97 version is different because Excel is primarily an application, not a development environment. The Excel version of this sample outputs information in an Excel worksheet.

To run Sample 1 clients, start SailWind Layout and open a design file. If you don't start SailWind Layout first, Sample 1 automatically starts the server, but it will not be visible and will not load a design file.

Make sure you copy the OLE samples from the distribution CD to your hard drive before running any of the samples.



CAUTION:

Never exit a client application without first disconnecting from the server to which it is connected.

Sample 2 - Notifications

The goal and specification for Sample 2 is to add client notifications, also called client callbacks, to Sample 1 so that Sample 1 can be notified when something changes in SailWind Layout. Sample 2 is implemented in SailWind Layout Sax Basic Engine, Microsoft Visual C++ 5.0 (and MFC), Microsoft Visual Basic 5.0, and Microsoft Excel 97.

The code samples in SailWind Layout Automation Server Help are freeware. These samples are provided as a courtesy to its users. Freeware is provided as is and no warranties with respect to freeware are made, either express or implied, including any implied warranties of merchantability or fitness for a particular purpose.



Note:

The source code for this sample is in the C:\SailWind Projects\Samples\Scripts\Layout\samples\sample2 folder. The code is well commented, and you may use it when developing your own client application.

Specification

Sample 2 is based on Sample 1 source code and functionality. This sample adds the ability to retrieve two more data types from the SailWind Layout server: the total number of components in the current design and the number of selected components. Moreover, Sample 2 automatically refreshes these values (name of the design file, component count, and selected component count) to match values in the SailWind Layout server at all times by responding to notifications about design file and selection change.

To test Sample 2, run SailWind Layout, open a design file, and connect the Sample 2 client to SailWind Layout. If you load another file or change the selection, Sample 2 refreshes its values automatically.

To run Sample 2 clients, start SailWind Layout and open a design file. If you don't start SailWind Layout first, Sample 2 automatically starts the server, but SailWind Layout will not be visible and will not load a design file.

Make sure you copy the OLE samples from the distribution CD to your hard drive before running any of the samples.



CAUTION:

Never exit a client application without first disconnecting from the server to which it is connected.

Sample 3 - Database Access

The goal and specification for Sample 3 is to demonstrate access to the SailWind Layout database.

The code samples in SailWind Layout Automation Server Help are freeware. These samples are provided as a courtesy to its users. Freeware is provided as is and no warranties with respect to freeware are made, either express or implied, including any implied warranties of merchantability or fitness for a particular purpose.



Tip

: The source code for this sample is in the C:\SailWind Projects\Samples\Scripts\Layout\samples\sample3 folder. The code is well commented, and you may use it when developing your own client application.

Specification

Sample 3 is based on Sample 2 source code. Sample 3 adds a list box that contains the list of all components in the current design. This list should automatically update when a new file is opened. This is the Basic version of Sample 3. To test Sample 3, run SailWind Layout, open a design file, and connect the Sample 3 client to SailWind Layout. If you open another file or change the selection, Sample 3 refreshes its values automatically.

To run Sample 3 clients, start SailWind Layout and open a design file. If you don't start SailWind Layout first, Sample 3 automatically starts the server, but SailWind Layout will not be visible and will not load a design file.

Make sure you copy the OLE samples from the distribution CD to your hard drive before running any of the samples.



CAUTION:

Never exit a client application without first disconnecting from the server to which it is connected.

Sample 4 - Cross-Probing

The goal and specification for Sample 4 is to extend the access to the SailWind Layout database components and retrieve detailed information about each component. It is also to add full-duplex selection cross-probing and demonstrate the server locking mechanism for improving Automation performance.

The code samples in SailWind Layout Automation Server Help are freeware. These samples are provided as a courtesy to its users. Freeware is provided as is and no warranties with respect to freeware are made, either express or implied, including any implied warranties of merchantability or fitness for a particular purpose.



Tip

: The source code for this sample is in the C:\SailWind Projects\Samples\Scripts\Layout\samples\sample4 folder. The code is well commented, and you may use it when developing your own client application.

Specification

Sample 4 is based on Sample 3 source code. Sample 4 must allow selection and cross-probing between the client and the server. In other words, the list of selected objects must match the list of selected items in the Sample 4 list box at all times and vice-versa. Selection and cross-probing must support multiple selection. If a user double-clicks an item in the list box, a dialog box must show extended information about that selected object. The performance of the access of SailWind Layout OLE methods will be improved using the server locking mechanism.

To test Sample 4, run SailWind Layout, open a design file, and connect the Sample 4 client to SailWind Layout. If you load another file or change the selection, Sample 4 refreshes its values automatically;

matching the selected components at all times. Similarly, a user selection change in the Sample 4 client changes the selection list.

To run Sample 3 clients, start SailWind Layout and open a design file. If you don't start SailWind Layout first, Sample 4 automatically starts the server, but SailWind Layout will not be visible and will not load a design file.

Make sure you copy the OLE samples from the distribution CD to your hard drive before running any of the samples.



CAUTION:

Never exit a client application without first disconnecting from the server to which it is connected.

Sample 5 - A SailWind Layout Custom Reports Command

The goal, specification, additional features, and implementation for Sample 5 is to provide custom reports with output formats such as text files, Microsoft Excel spreadsheets, Microsoft Word documents, and HTML pages.

The code samples in SailWind Layout Automation Server Help are freeware. These samples are provided as a courtesy to its users. Freeware is provided as is and no warranties with respect to freeware are made, either express or implied, including any implied warranties of merchantability or fitness for a particular purpose.



Tip

: The source code for this sample is in the C:\SailWind Projects\Samples\Scripts\Layout\samples\sample5 folder. The code is well commented, and you may use it when developing your own client application. This sample includes Basic files, main source code, an Excel file, report generation macros, and a SailWind Layout macro file that "plugs" the sample into SailWind Layout.

Specification

This sample generates customizable part list reports of the open design and/or any Assembly Options in that file. You can choose the format of the reports: text files, CSV files, Microsoft Excel spreadsheets, Microsoft Word documents (as tables), or HTML pages (for any Web browser). Each format requires that you install the appropriate application on your system. In other words, to generate Excel reports, you must install Excel on your system.

The data that the report outputs is configurable from the dialog box in the sample. You can easily select the part properties to output one by one, configure the order in which to output the properties, and group the report by any part property. Component properties available include reference designator, part type, decal, part type logic, part type description, number of attributes, SMD flag, number of pins, location, placed flag, orientation, mounting layer, and glued flag.

You can dynamically link the Excel reports with SailWind Layout; you can link each piece of data in the Excel report with its equivalent in SailWind Layout. This allows selection and cross-probing between the Excel report and SailWind Layout, and also allows changing the following part properties from the Excel report:

- Part decal. The list of compatible decals for that part appears in a list within the Excel cell.
- Location of the part.
- Orientation of the part.
- Mounting layer of the part. The list of valid layers appears in a list within the Excel cell.
- Glued flag.
- Installed flag, if the report is from an Assembly Option in the design.

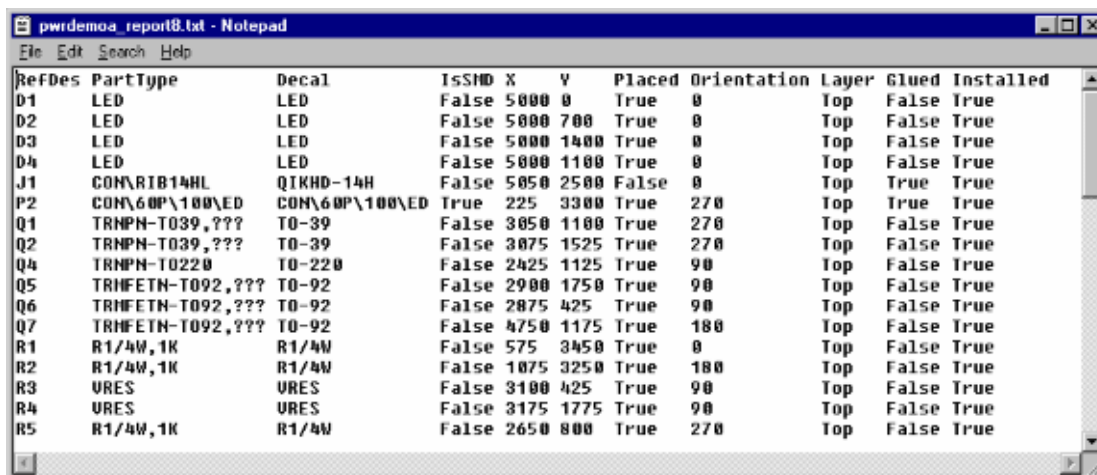
Additional Features

This sample includes many other useful features:

- Real-time refreshing of the dialog box when SailWind Layout opens or saves the design file. This ensures that the dialog box is always in sync with the program.
- The ability to open a design file from the dialog box. This avoids you having to switch back and forth between SailWind Layout and the dialog box when generating reports for many PCBs.
- The ability to save and load the report configurations to and from the system registry.
- The ability to reset default settings with a single button, to return to a “clean” configuration.
- A history of opened designs saved in the registry, up to ten, allowing quick report generation of a previously opened design file.
- Automatic suggestions for report file names.
- The ability to report all or only selected parts. This helps when generating a report on only one part in the PCB design.

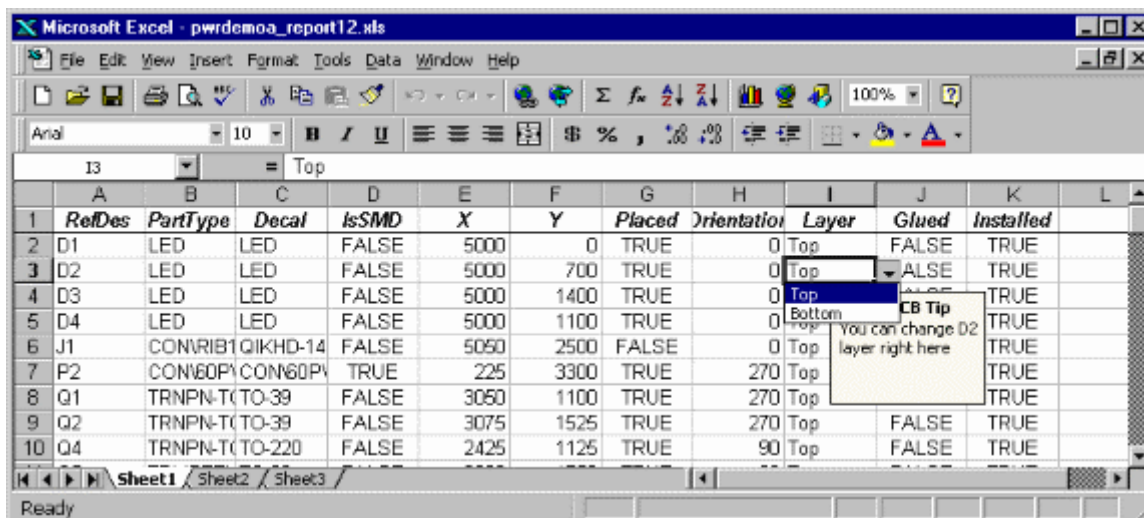
The following examples are screen captures of custom reports, generated by this sample, in Notepad, Microsoft Excel (dynamically linked), Microsoft Word, and Microsoft Internet Explorer 4.0:

Figure 2. Report in Notepad Format



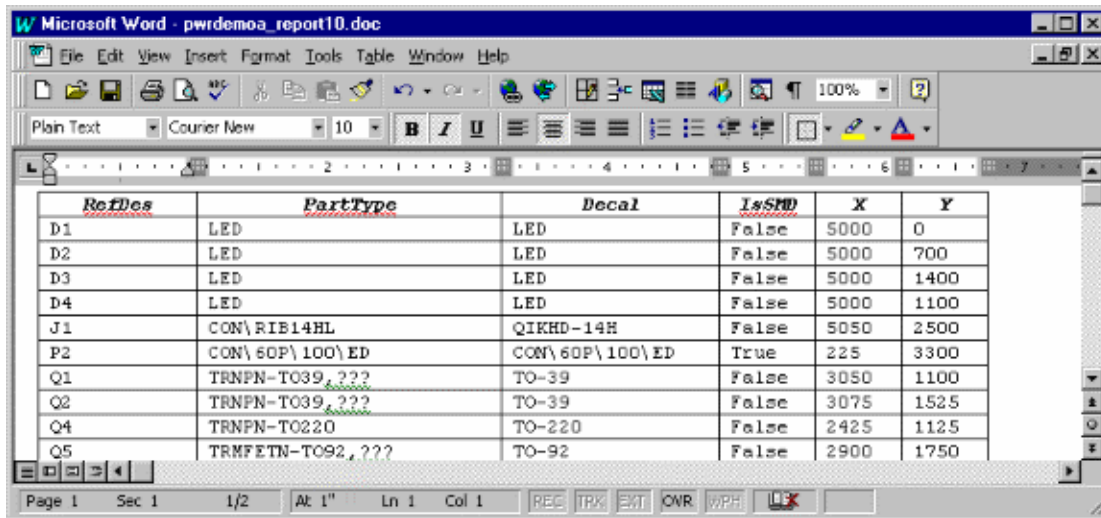
RefDes	PartType	Decal	IsSMD	X	Y	Placed	Orientation	Layer	Glued	Installed
D1	LED	LED	False	5000	0	True	0	Top	False	True
D2	LED	LED	False	5000	700	True	0	Top	False	True
D3	LED	LED	False	5000	1400	True	0	Top	False	True
D4	LED	LED	False	5000	1100	True	0	Top	False	True
J1	CON\60P\100\ED	QIKHD-14H	False	5050	2500	False	0	Top	True	True
P2	CON\60P\100\ED	CON\60P\100\ED	True	225	3300	True	270	Top	True	True
Q1	TRNPN-T039,???	T0-39	False	3050	1100	True	270	Top	False	True
Q2	TRNPN-T039,???	T0-39	False	3075	1525	True	270	Top	False	True
Q4	TRNPN-T0220	T0-220	False	2425	1125	True	90	Top	False	True
Q5	TRNPN-T092,???	T0-92	False	2900	1750	True	90	Top	False	True
Q6	TRNPN-T092,???	T0-92	False	2875	425	True	90	Top	False	True
Q7	TRNPN-T092,???	T0-92	False	4750	1175	True	180	Top	False	True
R1	R1/4W,1K	R1/4W	False	575	3450	True	0	Top	False	True
R2	R1/4W,1K	R1/4W	False	1075	3250	True	180	Top	False	True
R3	URES	URES	False	3100	425	True	90	Top	False	True
R4	URES	URES	False	3175	1775	True	90	Top	False	True
R5	R1/4W,1K	R1/4W	False	2650	800	True	270	Top	False	True

Figure 3. Report in Microsoft Excel Format



	A	B	C	D	E	F	G	H	I	J	K	L
1	RefDes	PartType	Decal	IsSMD	X	Y	Placed	Orientation	Layer	Glued	Installed	
2	D1	LED	LED	FALSE	5000	0	TRUE		0 Top	FALSE	TRUE	
3	D2	LED	LED	FALSE	5000	700	TRUE		0 Top	FALSE	TRUE	
4	D3	LED	LED	FALSE	5000	1400	TRUE		0 Top	FALSE	TRUE	
5	D4	LED	LED	FALSE	5000	1100	TRUE		0 Top	FALSE	TRUE	
6	J1	CON\60P\100\ED	QIKHD-14	FALSE	5050	2500	FALSE		0 Top	TRUE	TRUE	
7	P2	CON\60P\100\ED	CON\60P\100\ED	TRUE	225	3300	TRUE	270	Top	TRUE	TRUE	
8	Q1	TRNPN-T039,???	T0-39	FALSE	3050	1100	TRUE	270	Top	FALSE	TRUE	
9	Q2	TRNPN-T039,???	T0-39	FALSE	3075	1525	TRUE	270	Top	FALSE	TRUE	
10	Q4	TRNPN-T0220	T0-220	FALSE	2425	1125	TRUE	90	Top	FALSE	TRUE	

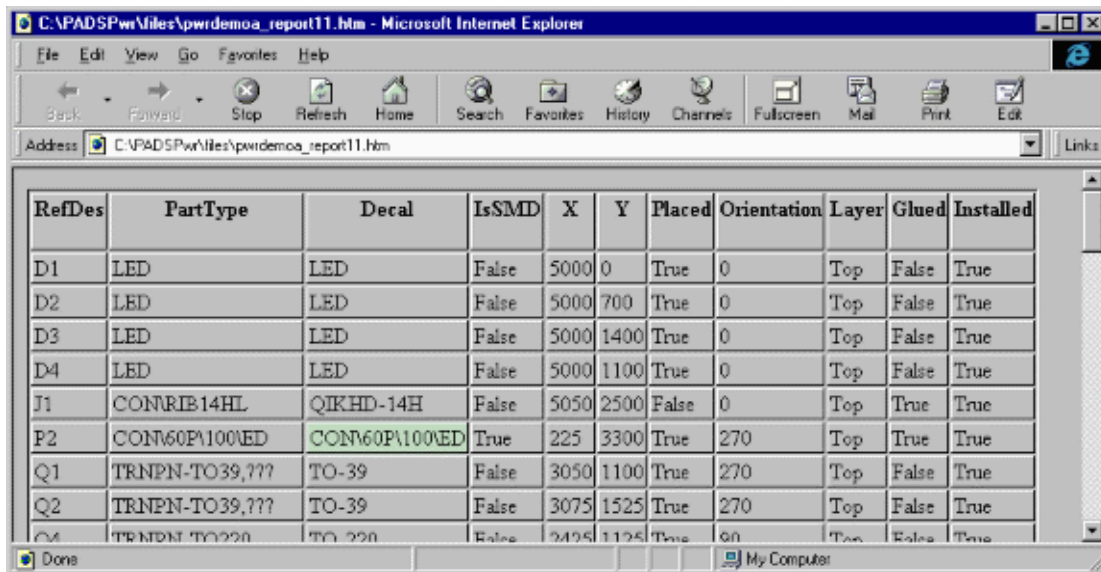
Figure 4. Report in Microsoft Word Format



The screenshot shows a Microsoft Word window titled 'pwrdenoa_report10.doc'. The report is displayed as a table with the following data:

<u>RefDes</u>	<u>PartType</u>	<u>Decal</u>	<u>IsSMD</u>	<u>X</u>	<u>Y</u>
D1	LED	LED	False	5000	0
D2	LED	LED	False	5000	700
D3	LED	LED	False	5000	1400
D4	LED	LED	False	5000	1100
J1	CON\RIB14HL	QIKHD-14H	False	5050	2500
P2	CON\60P\100\ED	CON\60P\100\ED	True	225	3300
Q1	TRNPN-TO39, ???	TO-39	False	3050	1100
Q2	TRNPN-TO39, ???	TO-39	False	3075	1525
Q4	TRNPN-TO220	TO-220	False	2425	1125
Q5	TRMFETN-TO92, ???	TO-92	False	2900	1750

Figure 5. Report in Microsoft Internet Explorer 4.0 Format



The screenshot shows a Microsoft Internet Explorer 4.0 window displaying a report from a file named 'pwrdenoa_report11.htm'. The report is a table with the following data:

RefDes	PartType	Decal	IsSMD	X	Y	Placed	Orientation	Layer	Glued	Installed
D1	LED	LED	False	5000	0	True	0	Top	False	True
D2	LED	LED	False	5000	700	True	0	Top	False	True
D3	LED	LED	False	5000	1400	True	0	Top	False	True
D4	LED	LED	False	5000	1100	True	0	Top	False	True
J1	CON\RIB14HL	QIKHD-14H	False	5050	2500	False	0	Top	True	True
P2	CON\60P\100\ED	CON\60P\100\ED	True	225	3300	True	270	Top	True	True
Q1	TRNPN-TO39, ???	TO-39	False	3050	1100	True	270	Top	False	True
Q2	TRNPN-TO39, ???	TO-39	False	3075	1525	True	270	Top	False	True
Q4	TRNPN-TO220	TO-220	False	2425	1125	True	0	Top	False	True

Implementation

This sample was developed with Microsoft Visual Basic 5.0. Dynamic Excel reports include some Visual Basic for Application (VBA) code in the Excel report file itself. The code is modularized to make it easy to implement other report formats, such as Microsoft Access, Visual FoxPro, and so on.

Sample 6 - Database Objects Accessed through the Automation Interface

The goal and specification for Sample 6 is to demonstrate access to shapes in the SailWind Layout database through the Automation interface.

The code samples in SailWind Layout Automation Server Help are freeware. These samples are provided as a courtesy to its users. Freeware is provided as is and no warranties with respect to freeware are made, either express or implied, including any implied warranties of merchantability or fitness for a particular purpose.



Tip

: The source code for this sample is in the C:\SailWind Projects\Samples\Scripts\Layout\samples\sample6 folder. The code is well commented, and you may use it when developing your own client application. This sample includes source code for Visual C++ v6.0.

Specification

Sample 6 is a Visual C++ application that displays all geometric objects, accessible through the Automation interface.

To test Sample 6, open a design in SailWind Layout, then build and run a sample executable. The sample retrieves data from the design and draws it in its window. To refresh the view, on the View menu, click **Redraw**.

SailWind Layout Automation Server Reference

The Automation Server Object Hierarchy strictly follows Microsoft standards. The Application Object, the root-level object, identifies the application and provides a way for Automation clients to bind to and navigate the application's exposed objects, methods, and properties. The Document Object handles all document-centered operations. The View Object handles all view-centered operations.

All SailWind Layout database objects, including the Drawing, Label, NetClass, Text, Component, PartType, Net, Pin, Via, Jumper, Connection, and RouteSegment objects, are accessed either directly through the Document object or through the Objects collection object. The Objects collection object provides a convenient way of working with a set of data objects rather than on a per object basis.

All Automation objects implement their interface based on the Microsoft IDispatch interface.

[Automation Objects](#)

[Constants](#)

[Optional Argument](#)

[Variant](#)

[Exception](#)

[Samples](#)

Automation Objects

SailWind Layout contains several automation objects.

- The AntiPad Object
- The Application Object
- The AssemblyOptions Collection Object
- The AssociatedNet Object
- The Attribute Object
- The Attributes Collection Object
- The CBP Object
- The Circle Object
- The Component Object
- The Connection Object
- The Decal Object
- The Document Object
- The Drawing Object
- The Error Object
- The ErrorConflict Object
- The Jumper Object
- The Label Object
- The Layer Object
- The Library Object
- The LibraryItem Object
- The Measure Object
- The Net Object
- The NetClass Object
- The Objects Collection Object
- The Pad Object
- The PadStackLayer Object
- The PartType Object
- The Pin Object
- The Polyline Object
- The RouteSegment Object
- The SBP Object
- The Text Object
- The ThermalPad Object
- The Via Object
- The View Object
- The Wirebond Object

The AntiPad Object

The AntiPad object represents an antipad in the padstack definition.

This object contains the following Properties:

- AntiPad.Application Property
- AntiPad.CornerRadius Property
- AntiPad.CornerType Property
- AntiPad.Length Property
- AntiPad.Name Property
- AntiPad.ObjectType Property
- AntiPad.Offset Property
- AntiPad.Orientation Property
- AntiPad.PadStackLayer Property
- AntiPad.Parent Property
- AntiPad.Shape Property
- AntiPad.Size Property

AntiPad.Application Property

Returns the application object.

Usage

Application as [Application](#) on page 68

Arguments

None

AntiPad.CornerRadius Property

Returns the Anti Pad corner radius.

Usage

```
CornerRadius as Double
```

```
CornerRadius (unit as PPcbUnit) as Double
```

Arguments

unit - [Optional]	Unit in which the value is returned.

AntiPad.CornerType Property

Returns the Anti Pad corner type.

Usage

```
CornerType as PPcbPadCornerType
```

Arguments

None

AntiPad.Length Property

Returns the Anti Pad length.

Usage

```
Length as Double
```

```
Length (unit as PPcbUnit) as Double
```

Arguments

<i>unit</i> - [Optional]	Unit in which the value is returned.

AntiPad.Name Property

Returns the name of this anti pad.

Usage

```
Name as String
```

Arguments

None

AntiPad.ObjectType Property

Returns the type of the object - ppcbObjectTypeAntiPad.

Usage

```
ObjectType as PPcbObjectType
```

Arguments

None

AntiPad.Offset Property

Returns the Anti Pad offset.

Usage

```
offset as Double
```

```
offset (unit as PPcbUnit) as Double
```

Arguments

<i>unit</i> - [Optional]	Unit in which the value is returned.

AntiPad.Orientation Property

Returns the Anti Pad orientation.

Usage

```
orientation as Double
```

Arguments

None

AntiPad.PadStackLayer Property

Returns the PadStackLayer Object to which this anti pad belongs to.

Usage

```
PadStackLayer as PadStackLayer
```

Arguments

None

AntiPad.Parent Property

Returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

AntiPad.Shape Property

Returns the anti pad shape.

Usage

```
Shape as PPcbAntiPadShape
```

Arguments

None

AntiPad.Size Property

Returns the anti pad's size. For shape `ppcbAntiPadShapeRound` it returns diameter.

Usage

```
Size (unit as PPcbUnit) as Double
```

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the value is returned. This optional argument is ppcbUnitCurrent on page 719 by default.

The Application Object

The Application object is the root-level object in the Automation Server object hierarchy, and represents the entire application. This object is usually the first object an Automation client connects to, before accessing an object, property, or method.

This object contains the following properties:

- Application.ActiveDocument Property
- Application.Application Property
- Application.DefaultFilePath Property
- Application.FullName Property
- Application.Libraries Property
- Application.Name Property
- Application.ObjectType Property
- Application.Parent Property
- Application.Preference Property
- Application.ProgressBar Property
- Application.StatusBarText Property
- Application.Version Property
- Application.Visible Property
- Application.CreateLibrary Method
- Application.ExportLibraryItems Method
- Application.GetConfigParamInt Method
- Application.GetConfigParamString Method
- Application.GetLibraryItems Method
- Application.LockServer Method
- Application.Measure Method
- Application.ModelessCmd Method
- Application.OpenDocument Method
- Application.OpenDocumentNoLock Method
- Application.OpenTempDocument Method
- Application.Quit Method
- Application.RunMacro Method
- Application.UnlockServer Method
- Application.OpenDocument Event
- Application.ProgressChange Event
- Application.Quit Event

Application.ActiveDocument Property

Returns the active document.

Usage

ActiveDocument As [Document](#) on page 250

Arguments

None

Description

The active document represents the open design.

Examples

The following sample code retrieves the name of the open design using the [Document.Name Property](#) property. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "You are working with " & ActiveDocument.Name
```

```
End Sub
```

Application.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This property identifies the object as an Automation object. All Automation server applications have an Application object and all Automation objects have an application property. This property is usually used in Automation client applications that handle large volumes of objects from different sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

Application.DefaultFilePath Property

Returns the path that the program uses to open design files.

Usage

DefaultFilePath As String

Arguments

None

Description

This property checks the FileDir folder entry in the SailWindpcb.ini file. When you set this property to a new value, the .ini file entry also changes.

For example, "C:\<install_folder>\<version>\Programs" is the default path when you install using the default installation settings.

Examples

The following sample code changes the default file path and notifies the client of this change. See [Code Samples](#) for more information on running this sample.

```
Sub Main

    oldPath = DefaultFilePath

    DefaultFilePath="C:\TEMP"

    MsgBox "The default file path used to be " & oldPath & " and it was just
        changed to " & DefaultFilePath

End Sub
```

Application.FullName Property

Returns the file name of the application, including its path.

Usage

FullName As String

Arguments

None

Description

For example, this function can return the string " C:\<install_folder>\<version>\Programs
\SailWindPCB.exe."

Examples

The following sample code displays the common name and the actual .exe name of the program. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "Hi, my name is " & Name & " and I am located in " & FullName
```

```
End Sub
```

Application.Libraries Property

Returns a collection of available libraries, or a specific library.

Usage

Libraries as Collection

Libraries(*Name* as String) as Library

Arguments

name	Name of the library to retrieve. Should not include wildcards.

Description

None

Examples

The following sample code displays the number of available libraries. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "Number of libraries: " & Libraries.Count
```

```
End Sub
```

Application.Name Property

Returns the name of the application.

Usage

Name As String

Arguments

None

Description

In SailWind Layout this property returns the string "PowerPCB."

This property is the default property for the Application object.

Examples

The following sample code displays the common name, the version, and the actual .exe name for the program, depending on which application you are running. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "Hi, my name is " & Name & " version " & Version & " and I am  
located in " & FullName
```

```
End Sub
```

Application.ObjectType Property

Returns the type of this object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

Application.Parent Property

Returns the parent of the object.

Usage

Parent As [Application](#) on page 68

Arguments

None

Description

None

Application.Preference Property

Sets or returns a preference.

Usage

Preference(*name* As String) As [Variant](#)

Arguments

<i>name</i>	Name of the preference.

Description

The following are possible *name* argument values:

<i>DRC</i>	Sets or returns the DRC mode of the open design. The DRC mode values are of type PPcbDRCMode .
<i>Nudge</i>	Sets or returns the Nudge mode of the open design. The Nudge mode values are of type .
<i>ModifyUnionMember</i>	Sets or returns the ability (True/False) to move an individual component when that component belongs to a union.

This property generates an [Exception](#) if the name argument is not a valid document preference.

Examples

The following sample code demonstrates how to use this property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

' Set the DRC mode to "Prevent"

ActiveDocument.Preference("DRC") = ppcbDRCPrevent

' Set the Nudge mode to "Automatic"

ActiveDocument.Preference("Nudge") = ppcbNudgeAuto

End Sub
```

Application.ProgressBar Property

Sets or returns current value of the status bar, as a percentage.

Usage

ProgressBar As Integer

Arguments

None

Description

This property allows you to retrieve the current status of long batch processes that are running, or to display the status of long Basic scripts.

Set the value greater than 0 or less than 100 to deactivate the status bar.

Use this property with [Application.StatusBarText Property](#).

Examples

The following sample demonstrates this property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

    StatusBarText="My Batch Process ..." 'show progress text

    For i = 0 to 100

        ProgressBar = i

    Next

    ProgressBar = -1 'deactivate progress bar

    StatusBarText="" 'hide progress text

End Sub
```

Application.StatusBarText Property

Sets or returns the text in the status bar.

Usage

StatusBarText As String

Arguments

None

Description

To set the status bar text to nothing, set this property to an empty string ("").

Examples

The following sample displays a message on the status bar. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
    StatusBarText="Wow! I can even print my own messages in here!"
```

```
End Sub
```

Application.Version Property

Returns the version.

Usage

Version As String

Arguments

None

Description

This property returns the application version as a string with the following format: <major>.<minor>, for example "4.0".

Examples

The following sample displays the application name and version. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "Hi, my name is " & Name & " version " & Version
```

```
End Sub
```

Application.Visible Property

Sets or returns whether the program is visible.

Usage

Visible As Boolean

Arguments

None

Description

This property is usually used in the following cases:

- When an Automation client starts the Automation server using an asynchronous OLE Automation call, such as the Visual Basic function `CreateObject`. The Automation server always starts as invisible (this is a client/server rule). You can use this property to make the program visible.
- When a client attempts to shut down the program (see [Application.Quit Method](#)) by making SailWind Layout invisible, disconnecting from it, and letting the server shut down appropriately.
- When a client needs to make the server window the active window, so it appears on top of other application windows.

Examples

The following sample makes the program invisible, waits a second, and then makes it visible again. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Visible = False
```

```
Wait 1
```

```
Visible = True
```

```
End Sub
```

Application.CreateLibrary Method

Creates a new library.

Usage

Creates new Library

CreateLibrary(*Name* as String, overwrite as Boolean) as Library

Arguments

None

Return Values

Created library object or null when failed.

Description

None

Application.ExportLibraryItems Method

Generates a PADS format ASCII file from the library items.

Usage

ExportLibraryItems(*Filename* as String, *Items* as Collection)

Arguments

<i>filename</i>	Name of the file to which to export the library item. Do not specify a file name extension.
<i>items</i>	Optional collection of items to export. If omitted, all items from the collection are exported.

Return Values

None

Description

Up to four files can be generated.

Examples

The following sample code exports library items, beginning with the letter 'R', from a library to a specified file. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set coll = GetLibraryItems(, "R*")
```

```
ExportLibraryItems("C:\sample", coll)
```

```
End Sub
```

Application.GetConfigParamInt Method

Retrieves the integer value from the specified parameter and section in the SailWindpcb.ini file.

Usage

```
GetConfigParamInt(  
    sectionName as String,  
    paramName as String,  
    defaultValue as Integer) as Integer
```

Arguments

<i>sectionName</i>	Name of the section containing the parameter name.
<i>paramName</i>	Name of the parameter whose associated integer value is to be retrieved.
<i>defaultValue</i>	If parameter name cannot be found, the default value is returned

Return Values

The parameter value or default value.

Examples

```
MsgBox Application.GetConfigParamString(  
  
    "directories", " FileDir", " C:\SailWind Projects\  
  
    MsgBox Application.GetConfigParamInt(  
  
    "general", " Display_Start_up_File_Dialog", 0)
```


Application.GetConfigParamString Method

Retrieves the string from the specified parameter and section in the SailWindpcb.ini file.

Usage

```
GetConfigParamString(  
    sectionName as String,  
    paramName as String,  
    defaultValue as String) as String
```

Arguments

<i>sectionName</i>	Name of the section containing the parameter name.
<i>paramName</i>	Name of the parameter whose associated string is to be retrieved.
<i>defaultValue</i>	If parameter name cannot be found, the default value is returned

Return Values

The parameter value or default value.

Examples

```
MsgBox Application.GetConfigParamString(  
  
"directories", " FileDir", " C:\SailWind Projects\  
  
MsgBox Application.GetConfigParamInt(  
  
"general", " Display_Start_up_File_Dialog", 0)
```

Application.GetLibraryItems Method

Returns a collection of library items in all available libraries, a collection of all items of a given type, or a specific item.

Usage

GetLibraryItems (*Type* as [PPcbLibraryItemType](#), *Name* as String) as LibraryItem

Arguments

<i>type</i>	Type of items (optional) to retrieve. Default value is PPcbLibraryItemType .
<i>name</i>	Name of the item to retrieve (optional). May include wildcards, ranges, and lists.

Description

None

Examples

The following sample code displays the number of available library items. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "Number of library items: " & GetLibraryItems().Count
```

```
End Sub
```

Application.LockServer Method

Locks the Automation server.

Usage

LockServer()

Arguments

None

Return Values

None

Description

To speed OLE call processing, use this function when your client makes many OLE calls to the server.



CAUTION:

Never forget to unlock a locked server. Do not keep the server locked for more than a few minutes.

The server locking mechanism speeds up processing by a factor of 2X to 8X the access time of the OLE server method or property calls. The server locking mechanism is dangerous because it disables many internal background tasks in the server, such as memory cleanup and visual updates. These server tasks must be performed regularly, but are disabled so that OLE incoming calls can be processed quickly.

Do not keep the server locked for more than a few minutes.

Examples

The following sample code shows how to use this method. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
LockServer
```

```
' Do something lengthy, which makes many calls to the SailWind Layout  
Automation server
```

```
UnlockServer
```

```
End Sub
```

Application.Measure Method

Creates and returns a measure value object.

Usage

Measure(*Value* As Variant, [*DefaultUnit* As String=""]) As Measure

Arguments

<i>value</i>	String or number representing a measure value including optional prefix and optional physical unit.
<i>defaultunit</i>	Optional string that contains default prefix and/or physical unit.

Description

This property parses string value such as "100pF" and creates a special object from which you can extract additional information such as real value, unit name, quantity name, and so on.

If the *Value* parameter contains unit information then *DefaultUnit* parameter is ignored.

If the *Value* parameter does not contain unit information and *DefaultUnit* is empty then a new Measure of Size/Dimension is created, and *Value* parameter will be interpreted by default as a number in current SailWind Layout design units (mils).

If the parser cannot recognize measure in the *Value* parameter it creates a dummy [Measure](#) on page 439 object with the [Measure.Value Property](#) property equal to 0.0 and [Measure.Text Property](#) property equal to the *Value* parameter.

Notice that there is a difference between Measure ("100", "pF") and Measure ("100pF"). Though both versions represent the same physical measure 100pF, the first one stores exact text representation without units.

Examples

The following sample adds a measure. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set M1 = Measure("500pF")

MsgBox M1.Text 'displays 500pF

End Sub
```

Application.ModelessCmd Method

Executes modeless commands.

Usage

ModelessCmd(command As String) As Void

Arguments

command

Return Values

none

Description

This method allows you to execute all the application's modeless commands available in the modeless command dialog box.

Examples

The following sample code switches layer visibility of the opened design. See "[Running Code Samples](#) on page 734" for more information on running this sample.

```
Sub Main
```

```
ModelessCmd("z 4")
```

```
End Sub
```

Related Topics

[Modeless Commands](#)

Application.OpenDocument Method

Opens a design file.

Usage

OpenDocument(*filename* As String) As [Document](#) on page 250

Arguments

<i>filename</i>	Name of the file to open.

Return Values

If the function succeeds, the return value is the newly opened [Document](#) on page 250.

If the function fails, the return value is current document.

Description

If *filename* does not contain the full path to the file, the program uses the path specified by the [Application.DefaultFilePath Property](#) to locate the file.

If *filename* is an empty string, a new blank design file is created.

If the file specified by *filename* cannot be found or cannot be opened, the return value is current document.

This method does not check if the currently opened file was saved or not. It is the client's responsibility to use the [Document.Saved Property](#) to check if the open design needs to be saved.

This method generates an [Exception](#) if a security failure occurs during processing.

Examples

The following sample code opens PWRDEMOA.PCB, assuming that it exists in the folder specified in the [Application.DefaultFilePath Property](#) property. The sample then displays the name of the newly opened file. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
OpenDocument(DefaultFilePath & "\PWRDEMOA.PCB")
```

```
MsgBox ActiveDocument.FullName & " has just been opened."
```

```
End Sub
```

Application.OpenDocumentNoLock Method

Opens a design file without file locking.

Usage

OpenDocumentNoLock(*filename* As String) As [Document](#) on page 250

Arguments

<i>filename</i>	Name of the file to open.

Return Values

If the function succeeds, the return value is the newly opened [Document](#) on page 250.

If the function fails, the return value is current document.

Application.OpenTempDocument Method

Works in the same way as OpenDocumentNoLock but additionally does not add the filename to the MRU (Most Recently Used) list. The Application.OpenTempDocument method is intended for macro tests.

Usage

OpenTempDocument(*filename* As String) As [Document](#) on page 250

Arguments

<i>filename</i>	Name of the file to open.

Return Values

If the function succeeds, the return value is the newly opened [Document](#) on page 250.

If the function fails, the return value is current document.

Application.Quit Method

Shuts down the program.



CAUTION:

Do not use this method.

Usage

Quit()

Arguments

None

Return Values

None

Description

Microsoft requires that all Automation Application objects implement this method; however, calling this method violates some important client/server rules:

- A server cannot shut down until all clients disconnect from it. Since it is, by definition, not possible for a client to call the Quit method (or any other server method) after it disconnects from the server, it is not possible for a client to shut down a server.
- A client cannot know if other clients are connected to the server. It, therefore, should not shut down a server.
- A server has its own shutdown management process: when the last client disconnects from the server, the server automatically shuts down only if its Graphical User Interface (GUI) is not active (not visible). Otherwise, the server remains active.

To force a shutdown, an Automation client needs to make the program invisible using the [Application.Visible Property](#) property, and then disconnect from it. If no other clients are connected to SailWind Layout at that time, the program automatically shuts down. If an Automation client is a Visual Basic script running in the Sax Basic Engine, it can never successfully shut down the Automation server.

Application.RunMacro Method

Runs a macro.

Usage

RunMacro(*filename* As String, *macroname* As String)

Arguments

<i>filename</i>	Name of the macro file to use.
<i>macroname</i>	Name of the macro to run.

Return Values

None

Description

If *filename* does not contain the full path to the file, the program uses the path specified by the [Application.DefaultFilePath Property](#) property to locate the file.

If *filename* is Empty, or is an empty string, the current default macro file is used.

If *macroname* is Empty, or not a valid macro name, nothing is performed.

This method generates an [Exception](#) if a security failure occurs during processing.

Examples

The following sample code runs the macro MACRO1, recorded in the macro file MACROS.MCR, assuming that the macro exists in the folder specified by the [Application.DefaultFilePath Property](#) property. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
RunMacro(DefaultFilePath & "\MACROS.MCR", "MACRO1")
```

```
End Sub
```

Application.UnlockServer Method

Unlocks the Automation server.

Usage

```
UnlockServer()
```

Arguments

None

Return Values

None

Examples

The following sample code shows how to use this method. See [Code Samples](#) for more information on running this sample.

```
Sub Main

LockServer

' Do something lengthy, which makes many calls to the SailWind Layout
Automation server

UnlockServer

End Sub
```

Application.OpenDocument Event

Occurs after the program opens a new document.

Usage

Application_OpenDocument (doc as Document)

Arguments

None

Return Values

None

Description

None

Application.ProgressChange Event

Occurs after the status bar value is changed.

Usage

Application_ProgressChange

Arguments

None

Return Values

None

Description

None

Application.Quit Event

Occurs before the program exits.

Usage

Application.Quit()

Arguments

None

Return Values

None

Description

This event occurs before the program exits.

The AssemblyOptions Collection Object

The AssemblyOptions collection object is the collection of all the assembly options in the open design.

This object is usually retrieved using the [Document.AssemblyOptions Property](#) property.

The AssemblyOptions collection object contains the following properties and methods:

- [AssemblyOptions.Application Property](#)
- [AssemblyOptions.Count Property](#)
- [AssemblyOptions.Item Property](#)
- [AssemblyOptions.ItemType Property](#)
- [AssemblyOptions.Next Property](#)
- [AssemblyOptions.ObjectType Property](#)
- [AssemblyOptions.Parent Property](#)
- [AssemblyOptions.ParentObject Property](#)
- [AssemblyOptions.Add Method](#)
- [AssemblyOptions.Delete Method](#)
- [AssemblyOptions.Merge Method](#)
- [AssemblyOptions.Remove Method](#)
- [AssemblyOptions.Reset Method](#)
- [AssemblyOptions.Select Method](#)
- [AssemblyOptions.Sort Method](#)

AssemblyOptions.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This is a Microsoft-required property.

This property identifies the object as an Automation object. All Automation server applications have an Application object and all Automation objects have an Application property. This property is usually used in Automation client applications that handle large volumes of objects from different sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

AssemblyOptions.Count Property

Returns the number of Assembly Options.

Usage

Count As Long

Arguments

None

Description

None

AssemblyOptions.Item Property

Returns an Assembly Option, given its index or its name.

Usage

Item(*index* As Long) As [Document](#) on page 250

Item(*name* As String) As [Document](#) on page 250

Arguments

<i>index</i>	Index (in the collection) of the Assembly Option to retrieve.
<i>name</i>	Name of the Assembly Option to retrieve.

Description

This is the default member of the AssemblyOptions collection object.

This property generates an [Exception](#) if the *index* or *name* argument is not valid.

AssemblyOptions.ItemType Property

Returns the type of an object given its index.

Usage

ItemType(*index* As long) As [PPcbObjectType](#)

ItemType(*name* As String) As [PPcbObjectType](#)

Arguments

<i>index</i>	Index of the object in the collection to query.
<i>name</i>	Name of the object to retrieve.

Description

This property generates an [Exception](#) if the *index* argument is not valid.

AssemblyOptions.Next Property

Returns the index of the next object of a specified type after the specified index.

Usage

Next(*index* As long, *type* As [PPcbObjectType](#)) As Long

Arguments

<i>index</i>	Index of the object in the collection to query.
<i>type</i>	Type of the object to query.

Description

This property generates an [Exception](#) if the *index* argument is not valid.

If Index = zero (0), then this function returns the index of the first item of the given type. If an item is not found, then the return value is zero (0).

AssemblyOptions.ObjectType Property

Returns the type of this object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

AssemblyOptions.Parent Property

Returns the parent of the object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

This is a Microsoft-required property.

AssemblyOptions.ParentObject Property

Returns the parent object of the collection.

Usage

ParentObject as Object

Arguments

None

Description

If a collection has ParentObject, it means the collection is “active.” Addition or deletion of an item to or from the collection modifies the parent object as well. Otherwise, only the collection is affected.

AssemblyOptions.Add Method

Adds a new Assembly option.

Usage

Add(*name* As String) As [Document](#) on page 250

Arguments

<i>name</i>	Name of the new assembly option to add.

Return Values

The new Assembly Option packaged as a [Document](#) on page 250 object.

Description

This property generates an [Exception](#) if the name argument is already an existing Assembly Option or if it is not a valid Assembly Option name.

AssemblyOptions.Delete Method

Deletes an Assembly Option.

Usage

Delete(*index* As Long)

Delete(*name* As String)

Arguments

<i>index</i>	Index (in the collection) of the Assembly Option to delete.
<i>name</i>	Name of the Assembly Option to delete.

Return Values

None

Description

This property generates an [Exception](#) if:

- The name argument is not an existing Assembly Option.
- The name argument is not a valid Assembly Option to delete.
- The index argument is greater than the number of existing Assembly Options.

Examples

The following sample code deletes all Assembly Options from the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set assopts = ActiveDocument.AssemblyOptions

For index=assopts.Count To 1 Step -1

assopts.Delete(index)

Next index

End Sub
```

AssemblyOptions.Merge Method

Merges two object collections.

Usage

Merge(*objects* As [Objects](#) on page 477)

Arguments

<i>objects</i>	The collection with objects to merge with the current object collection.

Return Values

None

Description

This method adds all objects in the *objects* collection object to the current object collection.

AssemblyOptions.Remove Method

Removes an object from the collection.

Usage

Remove(*index* As Long)

Remove(*name* As String)

Arguments

<i>index</i>	<i>Index</i> of the object to remove.
<i>name</i>	<i>Name</i> of the object to remove.

Return Values

None

Description

This property generates an [Exception](#) if the index argument is not valid.

AssemblyOptions.Reset Method

Resets the object collection.

Usage

Reset()

Arguments

None

Return Values

None

Description

None

AssemblyOptions.Select Method

Selects or deselects all objects in the collection.

Usage

Select([*bSelect* As Boolean = True])

Arguments

<i>bSelect</i>	[Optional on page 732] True to select. False to deselect.

Return Values

None

Description

None

AssemblyOptions.Sort Method

Sorts objects in the collection by object name.

Usage

Sort()

Arguments

None

Return Values

None

Description

None

The AssociatedNet Object

The AssociatedNet object represents an associated net existing in the PCB design currently open.

The AssociatedNet object contains the following properties:

- [AssociatedNet.Name Property](#)
- [AssociatedNet.Nets Property](#)
- [AssociatedNet.ObjectType Property](#)
- [AssociatedNet.Parent Property](#)
- [AssociatedNet.Selected Property](#)

AssociatedNet.Name Property

Returns the name of the associated net. For example, this property returns the string ?\$\$\$1*\$\$ \$2? for associated net \$\$\$1*\$\$\$2. The AssociatedNet.Name property is the default property for the AssociatedNet object.

Usage

```
Name as String
```

Arguments

None

AssociatedNet.Nets Property

Returns the collection of all nets belonging to the associated net.

Usage

```
Nets as Objects
```

```
Nets (name as String) as Net
```

Arguments

- name
Optional — Name of an existing net

Return Values

When an existing net name is passed to this property, it returns the net object with that name. If the net name is not specified, this property returns the collection of all nets in Objects.

AssociatedNet.ObjectType Property

Returns the type of the object. All application database objects in the Automation server implement this property to compensate for the lack of a Visual Basic equivalent for the Visual C++ QueryInterface function.

Usage

```
ObjectType as BlazeObjectType
```

Arguments

None

Return Values

This property always returns blazeObjectTypeAssociatedNet.

AssociatedNet.Parent Property

Returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

AssociatedNet.Selected Property

Sets or determines whether the associated net is selected or not.

You can also select an application database object using the [Document.SelectObjects Method](#) and [Objects.Select Method](#) methods.

Usage

```
Selected as Boolean
```

Arguments

None

Examples

The following sample code selects associated net \$\$\$1*\$\$\$2 only, assuming it exists in the open design. For more information about running this sample code, see [Code Samples](#).

```
Sub Main  
  
    ActiveDocument.SelectObjects(, , False)  
  
    ActiveDocument.AssociatedNets("$$$1*$$$2").Selected = True  
  
End Sub
```

Related Topics

[Document.SelectionChange Event](#)

[Document.SelectObjects](#)

[Objects.Select](#)

The Attribute Object

The Attribute object represents an attribute of a physical object, such as a Component or Net object.

This object is usually retrieved from the [The Attributes Collection Object](#).

The Attribute object contains the following properties and method:

- [Attribute.Application Property](#)
- [Attribute.Name Property](#)
- [Attribute.ObjectType Property](#)
- [Attribute.Parent Property](#)
- [Attribute.Value Property](#)
- [Attribute.Measure Method](#)

Attribute.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This is a Microsoft-required property.

This property identifies the object as a SailWind Layout Automation object. All Automation server applications have an Application object and all Automation objects have an Application property. This property is usually used in Automation client applications that handle large volumes of objects from different sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

Attribute.Name Property

Returns the name of the attribute.

Usage

Name As String

Arguments

None

Description

None

Examples

The following sample code lists all attributes in the open design and places that list in a custom dialog box. This sample uses the UserDialog Editor in the Sax Basic Engine in SailWind Layout. See the Sax Basic Editor On-line Help for more information. See [Code Samples](#) for more information on running this sample.

```
Dim ListAttrs$(10000)

Sub Main

    index = 0

    For Each nextAttr In ActiveDocument.Attributes

        ListAttrs$(index) = nextAttr.Name

        index = index + 1

    Next nextAttr

    ' This piece of code is automatically generated by the Basic Dialog Editor
    ' in SailWind Layout.

    Begin Dialog UserDialog 180,238,"Attributes" ' %GRID:10,7,1,1

        ListBox 10,7,160,203,ListAttrs(),.ListBox1

        OKButton 10,210,160,21
```

```
End Dialog
```

```
Dim dlg As UserDialog
```

```
Dialog dlg
```

```
End Sub
```

Attribute.ObjectType Property

Returns the type of this object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

Attribute.Parent Property

Returns the parent of the object.

Usage

Parent As Object

Arguments

None

Description

This is a Microsoft-required property.

Attribute.Value Property

Sets or returns the value of the attribute.

Usage

Value As [Variant](#)

Arguments

None

Description

The value of an attribute may be of type Boolean, Byte, Single, Integer, PortInt, Long, Double, String, or Measure object.

This property generates an [Exception](#) if a type mismatch exists between the value type and the type of the attribute.

If a value is not defined, an empty string is returned.

Examples

The following sample code changes the Cost attribute of all LED components to US\$3.99, in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each nextComp In ActiveDocument.Components

If nextComp.PartType="LED" Then

' Ignore exceptions generated when that attribute does not exist

On Error Resume Next

nextComp.Attributes("PRICE").Value = 3.99

End If

Next nextComp

End Sub
```

The following sample code adds attributes of different types to the level of PCB design and then changes their values. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
'Create new file to access some predefined attribute types such as List or  
Measure OpenDocument'''
```

```
'Add attributes of different types to PCB design
```

```
With Active Document
```

```
'Free text type
```

```
.Attributes.Add "Description","Free Text Value"
```

```
'Yes/No type
```

```
.Attributes.Add "PowerGround", "True"
```

```
'Number type
```

```
.Attributes.Add "Some Number", 100
```

```
'Decimal Number type
```

```
.Attributes.Add "Some Decimal Number", 0.45
```

```
'Measure type (in volts)
```

```
.Attributes.Add "Voltage", Measure("5V")
```

```
'list type (only values from predefined set allowed)
```

```
.Attributes.Add "HyperLynx.Signal Type", "Clock"
```

```
End With
```

```
'Modify PCB attributes of different types
```

```
With ActiveDocument
```

```
'Free text type
```

```
.Attributes("Description").Value="Another Free Text Value"
```

```
'Yes/No type
```

```
.Attributes("PowerGround").Value = False
```

```
'Number type
```

```
.Attributes("Some Number").Value = 200
```

```
'Decimal Number type
```

```
.Attributes("Some Decimal Number").Value = 0.25
```

```
'Measure type (in volts)
```

```
.Attributes("Voltage").Value = Measure("2.5V")
```

```
'list type (only values from predefined set allowed)
```

```
.Attributes("HyperLynx.Signal Type").Value="Data"
```

```
End With
```

```
End Sub
```

Attribute.Measure Method

Returns the measure value object.

Usage

Measure As [Measure](#) on page 439

Arguments

None

Description

This property parses an attribute string value such as “100pF” and creates a special object from which you can extract additional information such as real value, unit name, quantity name, and so on.

If the internal parser cannot recognize a measure value object in the attribute it creates a dummy [Measure](#) on page 439 object with the [Measure.Value Property](#) property equal to 0.0 and [Measure.Text Property](#) property equal to [Attribute.Value Property](#) property.

Examples

The following sample shows a quantity name of the capacitor's Value attribute (Capacitance) of part C1 assuming that it exists in the open schematic. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set C1 = ActiveDocument.Components("C1")
```

```
MsgBox C1.Attributes("Value").Measure.Quantity 'shows "Capacitance"
```

```
End Sub
```

The Attributes Collection Object

The Attributes collection object is a collection of physical object attributes, such as attributes of a Document, a Component, or a Pin.

The Attributes Collection object contains the following properties and methods:

- Attributes.Application Property
- Attributes.Count Property
- Attributes.Item Property
- Attributes.ItemType Property
- Attributes.Next Property
- Attributes.ObjectType Property
- Attributes.Parent Property
- Attributes.ParentObject Property
- Attributes.Add Method
- Attributes.Delete Method
- Attributes.Merge Method
- Attributes.Remove Method
- Attributes.Reset Method
- Attributes.Select Method
- Attributes.Sort Method

Attributes.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This is a Microsoft-required property.

This property identifies the object as a SailWind Layout Automation object. All Automation server applications have an Application object and all Automation objects have an Application property. This property is usually used in Automation client applications that handle large volumes of objects from different sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

Attributes.Count Property

Returns the number of attributes.

Usage

Count As Long

Arguments

None

Description

None

Attributes.Item Property

Returns an attribute, given its index or its name.

Usage

Item(*index* As Long) As [Attribute](#) on page 120

Item(*name* As String) As [Attribute](#) on page 120

Arguments

<i>index</i>	Index (in the collection) of the attribute to retrieve.
<i>name</i>	Name of the attribute to retrieve.

Description

This is the default member of the Attributes collection object.

This property generates an [Exception](#) if the *index* or *name* argument is not valid.

Attributes.ItemType Property

Returns the type of an object given its index.

Usage

ItemType(*index* As long) As [PPcbObjectType](#)

ItemType(*name* As String) As [PPcbObjectType](#)

Arguments

<i>index</i>	Index of the object in the collection to query.
<i>name</i>	Name of the object to retrieve.

Description

This property generates an [Exception](#) if the *index* argument is not valid.

Attributes.Next Property

Returns the index of the next object of a specified type after the specified index.

Usage

Next(*index* As long, type As [PPcbObjectType](#)) As Long

Arguments

<i>index</i>	Index of the object in the collection to query.
<i>type</i>	Type of the object to query.

Description

This property generates an [Exception](#) if the *index* argument is not valid.

If Index = zero (0), then this function returns the index of the first item of the given type. If an item is not found, then the return value is zero (0).

Attributes.ObjectType Property

Returns the type of this object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

Attributes.Parent Property

Returns the parent of the object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

This is a Microsoft-required property.

Attributes.ParentObject Property

Returns the parent object of the collection.

Usage

ParentObject as Object

Arguments

None

Description

If a collection has ParentObject, it means the collection is “active.” Addition or deletion of an item to or from the collection modifies the parent object as well. Otherwise, only the collection is affected.

Attributes.Add Method

Adds a new attribute.

Usage

Add(*name* As String, [*value* As [Variant](#)]) As [Attribute](#) on page 120

Arguments

<i>name</i>	Name of the new attribute.
<i>value</i>	Optional value of the new attribute. The attribute may be of type Boolean, Byte, Single, Integer, PortInt, Long, Double, String, and Measure object.

Return Values

The new attribute packaged as an [Attribute](#) on page 120 object.

Description

This property generates an [Exception](#) if the name argument is already an existing attribute or if it is not a valid attribute name.

Attributes.Delete Method

Deletes an attribute.

Usage

Delete(*index* As Long)

Delete(*name* As String)

Arguments

<i>index</i>	Index (in the collection) of the attribute to delete.
<i>name</i>	Name of the attribute to delete.

Return Values

None

Description

This property generates an [Exception](#) in the following cases:

- If the *name* argument is not an existing attribute
- If the *name* argument is not a valid attribute to be deleted
- If the *index* argument is higher than the number of existing attributes

Examples

The following sample code deletes the Cost attribute of all components in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each nextComp In ActiveDocument.Components

' Avoid exceptions generated when that attribute does not exist

If Not nextComp.Attributes("Cost") Is Nothing

    nextComp.Attributes.Delete("Cost")

End If
```



```
Next nextComp
```

```
End Sub
```

Attributes.Merge Method

Merges two object collections.

Usage

Merge(*objects* As [Objects](#) on page 477)

Arguments

<i>objects</i>	The collection with objects to merge with the current object collection.

Return Values

None

Description

This method adds all objects in the objects collection object to the current object collection.

Attributes.Remove Method

Removes an object from the collection.

Usage

Remove(*index* As Long)

Remove(*name* As String)

Arguments

<i>index</i>	Index of the object to remove.
<i>name</i>	Name of the object to remove.

Return Values

None

Description

This property generates an [Exception](#) if the index argument is not valid.

Attributes.Reset Method

Resets the object collection.

Usage

Reset()

Arguments

None

Return Values

None

Description

None

Attributes.Select Method

Selects or deselects all objects in the collection.

Usage

```
Select([bSelect As Boolean = True])
```

Arguments

<i>bSelect</i>	[Optional on page 732] True to select. False to deselect.

Return Values

None

Description

None

Attributes.Sort Method

Sorts objects in the collection by object name.

Usage

Sort()

Arguments

None

Return Values

None

Description

None

The CBP Object

The CBP object represents the physical component bond pad of a die part. You can retrieve the CBP object, or collection of CBP objects, only from the Component object, which represents a die part (Component.IsDiePart returns True), or from other die part constituents (such as SBP or Wirebond).

CBP objects are not selectable.

You cannot retrieve the collection of all CBP objects from a document using Document.GetObjects (ppcbObjectTypeCBP). This method returns an empty collection. Also, Document.GetObjects (ppcbObjectTypeAll) does not return CBP, SBP, or Wirebond objects. You can get only collections of CBPs, SBPs, or Wirebonds using the Component object methods, if the component is a die part ([Component.IsDiePart Property](#)).

The CBP object contains the following properties and method:

- CBP.Application Property
- CBP.Edge Property
- CBP.Function Property
- CBP.Layer Property
- CBP.Length Property
- CBP.Name Property
- CBP.ObjectType Property
- CBP.Parent Property
- CBP.PositionX Property
- CBP.PositionY Property
- CBP.SBPs Property
- CBP.Shape Property
- CBP.Width Property
- CBP.Wirebonds Property
- CBP.Component Method

CBP.Application Property

Returns the Application object.



Restriction:

This information applies only to the BGA toolkit.

Usage

Application As [Application](#) on page 68

Arguments

None

Descriptions

This property identifies the object as an Automation object. All Automation server applications have an Application object and all Automation objects have an application property. This property is usually used in Automation client applications that handle large volumes of objects from different sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

This is a Microsoft-required property.

CBP.Edge Property

Returns the edge of the CBP. This information applies to only the BGA toolkit.



Restriction:

This information applies to only the BGA toolkit.

Usage

Edge As [PPcbBondPadEdge](#)

Arguments

None

Description

None

Examples

The following sample code displays information about the edge of each CBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aCBP in comp.CBPs

Select Case aCBP.Edge

Case ppcbBondPadEdgeUnknown

MsgBox aCBP.Name & " Edge: Unknown"
```



```
Case ppcbBondPadEdgeLeft

MsgBox aCBP.Name & " Edge: Left"

Case ppcbBondPadEdgeTop

MsgBox aCBP.Name & " Edge: Top"

Case ppcbBondPadEdgeRight

MsgBox aCBP.Name & " Edge: Right"

Case ppcbBondPadEdgeBottom

MsgBox aCBP.Name & " Edge: Bottom"

Case Else

MsgBox aCBP.Name & " Edge: Unknown"

End Select

Next

End If

Next

End Sub
```

CBP.Function Property

Returns the signal name of the CBP.



Restriction:

This information applies to only the BGA toolkit.

Usage

Function As String

Arguments

None

Descriptions

None

Examples

The following sample code retrieves the signal name of each CBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aCBP in comp.CBPs

MsgBox aCBP.Name & " Function: " & aCBP.Function

Next

End If

Next

End Sub
```

CBP.Layer Property

Returns the LIQ layer of the CBP.



Restriction:

This information applies to only the BGA toolkit.

Usage

Layer As String

Arguments

None

Descriptions

None

Examples

The following sample code retrieves the layer on which each CBP in each die in the open design is mounted. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aCBP in comp.CBPs

MsgBox aCBP.Name & " Layer: " & aCBP.Layer

Next

End If

Next

End Sub
```

CBP.Length Property

Returns the length of the CBP.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

Length([*unit* As [PPcbUnit](#) = ppcbUnitCurrent]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the length value is returned.

Descriptions

None

Examples

The following sample code retrieves the length of each CBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aCBP in comp.CBPs

MsgBox aCBP.Name & " Length: " & Format (aCBP.Length, "#.###")

Next

End If

Next

End Sub
```

CBP.Name Property

Returns the name of the CBP.



Restriction:
This information applies to only the BGA toolkit.

Usage

Name As String

Arguments

None

Descriptions

This is a Microsoft-required property.

Examples

The following sample code retrieves the name of each CBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aCBP in comp.CBPs

MsgBox aCBP.Name

Next

End If

Next

End Sub
```

CBP.ObjectType Property

Returns the type of the object.



Restriction:

This information applies to only the BGA toolkit.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

This property always returns ppcbObjectTypeCBP.

This property is generally used to identify the kind of database objects in a heterogeneous [Objects](#) on page 477 collection or when implementing a generic routine that depends on the type of the database object passed as the argument.

All database objects in the Automation server implement this property to compensate for the lack of a Visual Basic equivalent for the Visual C++ QueryInterface function.

Examples

See [Code Samples](#) for more information on running this sample.

```
Sub DoSomethingToDieObject(dbObject As Object)

Select Case dbObject.ObjectType

Case ppcbObjectTypeCBP

' Do something specific to CBP objects

Case ppcbObjectTypeSBP

' Do something specific to SBP objects

Case ppcbObjectTypeWirebond

' Do something specific to Wirebond objects

Case Else

MsgBox "Not a Die object"

End Select

End Sub
```

CBP.Parent Property

Returns the parent of the object.



Restriction:

This information applies to only the BGA toolkit.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

This is a Microsoft-required property.

CBP.PositionX Property

Returns the x-coordinate of the CBP.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

PositionX([*unit* As [PPcbUnit](#) = ppcbUnitCurrent]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the x-coordinate is returned.

Description

None

Examples

The following sample code retrieves the location of each CBP in each die in the open design, in current units. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aCBP in comp.CBPs

MsgBox aCBP.Name & ": (" & aCBP.PositionX & ", " & aCBP.PositionY & ")"

Next

End If
```

Next

End Main

CBP.PositionY Property

Returns the y-coordinate of the CBP.



Restriction:

This information applies to only the BGA toolkit.

Usage

PositionY(*[unit As PPcbUnit = ppcbUnitCurrent]*) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the y-coordinate is returned.

Descriptions

None

Examples

The following sample code retrieves the location of each CBP in each die in the open design, in current units. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
For Each comp In ActiveDocument.Components
```

```
If comp.IsDiePart Then
```

```
For Each aCBP in comp.CBPs
```

```
MsgBox aCBP.Name & ": (" & aCBP.PositionX & ", " & aCBP.PositionY & ")"
```

```
Next
```

```
End If
```

```
Next
```


End Sub

CBP.SBPs Property

Returns the collection of SBPs linked to this CBP object.



Restriction:

This information applies to only the BGA toolkit.

Usage

SBPs As [Objects](#) on page 477

Arguments

None

Description

None

Examples

The following sample retrieves the number of SBPs linked to each CBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
For Each comp In ActiveDocument.Components
```

```
  If comp.IsDiePart Then
```

```
    For Each aCBP in comp.CBPs
```

```
      MsgBox "Number of SBPs linked to " & aCBP.Name & ": " & aCBP.SBPs.Count
```

```
    Next
```

```
  End If
```

```
Next
```

```
End Main
```

CBP.Shape Property

Returns the shape of the CBP.



Restriction:

This information applies to only the BGA toolkit.

Usage

Shape As [PPcbShapeType](#)

Arguments

None

Description

None

Examples

The following sample retrieves the shape of each CBP on each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aCBP in comp.CBPs

Select Case aCBP.Shape

Case ppcbBondPadShapeUnknown

MsgBox aCBP.Name & " Shape: Unknown"

Case ppcbBondPadShapeRectangle

MsgBox aCBP.Name & " Shape: Rect"

Case ppcbBondPadShapeOval

MsgBox aCBP.Name & " Shape: Oval"
```

```
Case Else
```

```
MsgBox aCBP.Name & " Shape: Unknown"
```

```
End Select
```

```
Next
```

```
End If
```

```
Next
```

```
End Sub
```

CBP.Width Property

Returns the width of the CBP.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

Width(*[unit As PPcbUnit = ppcbUnitCurrent]*) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the width value is returned.

Description

None

Examples

The following sample retrieves the width of each CBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
For Each comp In ActiveDocument.Components
```

```
If comp.IsDiePart Then
```

```
For Each aCBP in comp.CBPs
```

```
MsgBox aCBP.Name & " Width: " & Format (aCBP.Width, "#.###")
```

```
Next
```

```
End If
```

```
Next
```

```
End Sub
```

CBP.Wirebonds Property

Returns the collection of bond wires attached to the CBP object.



Restriction:

This information applies to only the BGA toolkit.

Usage

Wirebonds As [Objects](#) on page 477

Arguments

None

Description

None

Examples

The following sample retrieves the number of bond wires connected to each CBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
For Each comp In ActiveDocument.Components
```

```
If comp.IsDiePart Then
```

```
For Each aCBP in comp.CBPs
```

```
MsgBox "Number of WBs connected to " & aCBP.Name & ":" &  
aCBP.Wirebonds.Count
```

```
Next
```

```
End If
```

```
Next
```

```
End Sub
```

CBP.Component Method

Returns the component object of the CBP.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

Component As [Component](#) on page 178

Arguments

None

Descriptions

None

Examples

The following sample code represents a subroutine that depends on the CBP object passed as an argument. The code retrieves the name of the component and the name of the decal to which the CBP belongs. See [Code Samples](#) for more information on running this sample.

```
Sub QueryCBP(aCBP As CBP)
```

```
MsgBox "Component Bond Pad" & aCBP.Name &
```

```
"belongs to component" & aCBP.Component.Name &
```

```
"(" & aCBP.Component.Decal & ")"
```

```
End Sub
```

The Circle Object

The Circle object represents a physical circle in the open design.

The Circle object contains the following properties:

- Circle.Application Property
- Circle.CenterX Property
- Circle.CenterY Property
- Circle.Geometry Property
- Circle.Layer Property
- Circle.LineStyle Property
- Circle.LineWidth Property
- Circle.ObjectType Property
- Circle.OutlineType Property
- Circle.Parent Property
- Circle.Radius Property
- Circle.ShapeType Property

Circle.Application Property

Returns the Application object.

Usage

Application as [Application](#) on page 68

Arguments

None

Description

This is a Microsoft-required property.

Circle.CenterX Property

Returns the x-coordinate of the circle's center.

Usage

CenterX (*Unit* as [PPcbUnit](#), *Origin* as [PPcbOriginType](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the center X value is returned. This optional argument is ppcbUnitCurrent on page 719 by default.
<i>origin</i>	[Optional on page 732]Type of reference point from which the result is counted. The default value is ppcbOriginTypeDesign on page 719.

Description

None

Examples

The following sample code shows the position of the selected circle. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,TRUE)

For Each drw In selected

Dim geom as Object

For Each geom In drw.Geometry

If geom.ObjectType = ppcbObjectTypeCircle Then

MsgBox "Position: (" & geom.CenterX & ", " & geom.CenterY & ")"

End If

Next geom

Next drw
```


End Sub

Circle.CenterY Property

Returns the y-coordinate of the circle's center.

Usage

CenterY (*Unit* as [PPcbUnit](#), *Origin* as [PPcbOriginType](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the center Y value is returned. This optional argument is ppcbUnitCurrent on page 719 by default.
<i>origin</i>	[Optional on page 732]Type of reference point from which the result is counted. The default value is ppcbOriginTypeDesign on page 719.

Description

None

Examples

The following sample code shows the position of the selected circle. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,TRUE)

For Each drw In selected

Dim geom as Object

For Each geom In drw.Geometry

If geom.ObjectType = ppcbObjectTypeCircle Then

MsgBox "Position: (" & geom.CenterX & ", " & geom.CenterY & ")"

End If

Next geom

Next drw
```

End Sub

Circle.Geometry Property

Returns a collection of objects, currently polylines, texts, or circles, representing this object's child geometry objects.

Usage

Geometry as Collection

Arguments

None

Description

None

Examples

The following sample code shows the number of child objects. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,TRUE)

For Each drw In selected

Dim geom as Object

For Each geom In drw.Geometry

MsgBox "Child object count: " & geom.Geometry.Count

Next geom

Next drw

End Sub
```

Circle.Layer Property

Returns the layer number of the object.

Usage

Layer as Long

Arguments

None

Description

None

Examples

The following sample code shows the layer number of the selected circle. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,TRUE)

For Each drw In selected

Dim geom as Object

For Each geom In drw.Geometry

MsgBox "Layer number: " & geom.Layer

Next geom

Next drw

End Sub
```

Circle.LineStyle Property

Returns the line style of the circle.

Usage

LineStyle as [PPcbLineStyle](#)

Arguments

None

Description

None

Examples

None

Circle.LineWidth Property

Returns the line width of the circle.

Usage

LineWidth (*Unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the line width value is returned. This optional argument is ppcbUnitCurrent on page 719 by default.

Description

None

Examples

The following sample code shows the line width of the selected circle. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing, , TRUE)

For Each drw In selected

Dim geom as Object

For Each geom In drw.Geometry

MsgBox "LineWidth: " & geom.LineWidth

Next geom

Next drw

End Sub
```

Circle.ObjectType Property

Returns the type of object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

Examples

The following sample code tests the ObjectType property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each drw In ActiveDocument.Drawings

For Each geom In drw.Geometry

type = geom.ObjectType

If type <> ppcbObjectTypePolyline And type <> ppcbObjectTypeCircle Then

MsgBox "Test failed"

End If

Next geom

Next drw

End Sub
```


Circle.OutlineType Property

Returns the outline type of the circle.

Usage

OutlineType as PPcbOutlineType

Arguments

None

Description

None

Examples

The following sample code shows the outline type of the selected circle. See “Running Code Samples” for more information on running this sample.

```
Sub Main
Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,TRUE)
For Each drw In selected
Dim geom as Object
For Each geom In drw.Geometry
Select Case geom.OutlineType
Case ppcbOutLineTypeCenter
s="Center line"
Case ppcbOutLineTypeOuter
s="Outer line"
Case ppcbOutLineTypeInner
s="Inner line"
End Select
MsgBox "Outline type: " & s
Next geom
Next drw
End Sub
```

Circle.Parent Property

Returns the parent of the object.

Usage

Parent as Document

Arguments

None

Description

This is a Microsoft-required property.

Circle.Radius Property

Returns the value of the radius of the circle.

Usage

Radius (*Unit* as [PPcbUnit](#)) as Double

Arguments

<i>Unit</i>	[Optional on page 732] Unit in which the radius value is returned. This optional argument is ppcbUnitCurrent on page 719 by default.

Description

None

Examples

The following sample code shows the radius of the selected circle. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,TRUE)

For Each drw In selected

Dim geom as Object

For Each geom In drw.Geometry

If geom.ObjectType = ppcbObjectTypeCircle Then

MsgBox "Radius: " & geom.Radius

End If

Next geom

Next drw

End Sub
```

Circle.ShapeType Property

Returns the shape type of the circle.

The value of [PPcbShapeOpen](#) on page 719 is not applicable.

Usage

ShapeType as [PPcbShapeType](#)

Arguments

None

Description

None

Examples

The following sample code shows the shape type of the selected circle. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,TRUE)

For Each drw In selected

Dim geom as Object

For Each geom In drw.Geometry

Select Case geom.ShapeType

Case ppcbShapeTypeOpen

s="Open"

Case ppcbShapeTypeHollow

s="Hollow"

Case ppcbShapeTypeFilled

s="Filled"
```

```
Case ppcbShapeTypeVoid
```

```
s="Void"
```

```
End Select
```

```
MsgBox "Shape type: " & s & " shape"
```

```
Next geom
```

```
Next drw
```

```
End Sub
```

The Component Object

The Component object represents a physical component, in the currently open design.

The Component object contains the following properties and methods:

- Component.Application Property
- Component.Attributes Property
- Component.CBPs Property
- Component.CenterX Property
- Component.CenterY Property
- Component.Decal Property
- Component.DecalAttributes Property
- Component.DecalCompatibleList Property
- Component.DieHeight Property
- Component.DieLength Property
- Component.DieWidth Property
- Component.Glued Property
- Component.Installed Property
- Component.IsDiePart Property
- Component.IsSMD Property
- Component.Labels Property
- Component.Layer Property
- Component.Name Property
- Component.ObjectType Property
- Component.Orientation Property
- Component.Parent Property
- Component.PartType Property
- Component.PartTypeAttributes Property
- Component.PartTypeECORegistered Property
- Component.PartTypeLogic Property
- Component.PartTypeObject Property
- Component.Pins Property
- Component.Placed Property
- Component.PositionX Property
- Component.PositionY Property
- Component.SBPs Property
- Component.Selected Property
- Component.Substituted Property
- Component.WireBondRulesAngleMaximum Property
- Component.WireBondRulesClearanceWireToPad Property
- Component.WireBondRulesClearanceWireToWire Property
- Component.WireBondRulesLengthMaximum Property
- Component.WireBondRulesLengthMinimum Property

[Component.Wirebonds Property](#)
[Component.AddLabel Method](#)
[Component.Move Method](#)
[Component.MoveCenter Method](#)

Component.Application Property

Returns the application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This property identifies the object as an automation object. All Automation server applications have an application object and all automation objects have an application property. This property is usually used in automation client applications that handle large volumes of objects from different sources, such as different automation server applications, to quickly identify the application to which the object belongs.

Component.Attributes Property

Returns the collection of all attributes of the component.

Usage

Attributes As [Attributes](#) on page 130

Attributes(*name* As String) As [Attribute](#) on page 120

Arguments

<i>name</i>	Name of an existing component attribute.

Description

When an existing attribute *name* is passed to this property, it returns that component [Attribute](#) on page 120 object. Otherwise, it returns the collection of all component attributes in an [Attributes](#) on page 130 collection object.

Examples

The following sample code retrieves the number of attributes on component U1, assuming it exists in the open design, using the [Attributes.Count Property](#) property. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set attrs = ActiveDocument.Components("U1").Attributes
```

```
MsgBox "There are " & attrs.Count & " attribute(s) in component U1."
```

```
End Sub
```

Component.CBPs Property

Returns the CBP collection of the die.



Restriction:

This information applies to only the BGA toolkit.

Usage

CBPs As [Objects](#) on page 477

Arguments

None

Description

This property generates an [Exception](#) if [Component.IsDiePart Property](#) is False.

Examples

The following sample code iterates through all CBPs in each die in the open design and prints the names of the CBPs. See [Code Samples](#) for more information on running this sample.

```
Sub Main  
  
For Each comp In ActiveDocument.Components  
  
If comp.IsDiePart Then  
  
For Each aCBP In comp.CBPs  
  
Debug.Print aCBP.Name  
  
Next aCBP  
  
End If  
  
Next  
  
End Sub
```

Component.CenterX Property

Returns the x-coordinate of the center of the component.

Usage

CenterX (*Unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the CenterX value is returned. This optional argument is ppcbUnitCurrent on page 719 by default.

Description

None

Examples

The following sample code displays coordinates of the center of a component. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeComponent,,True)

For Each comp In selected

Dim msg as String

Msg="Component center: (" & comp.CenterX & ", " & comp.CenterY & ")"

MsgBox msg

Exit For

Next comp

End Sub
```

Component.CenterY Property

Returns the y-coordinate of the center of the component.

Usage

CenterY (*Unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the CenterY value is returned. This optional argument is ppcbUnitCurrent on page 719 by default.

Description

None

Examples

The following sample code displays coordinates of the center of the selected component. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeComponent,,True)

For Each comp In selected

Dim msg as String

Msg="Component center: (" & comp.CenterX & ", " & comp.CenterY & ")"

MsgBox msg

Exit For

Next comp

End Sub
```

Component.Decal Property

Sets or returns the decal of the component.

Usage

Decal As String

Arguments

None

Description

This property can only set a decal that is in the list of component-compatible decals, which is returned by the [Component.DecalCompatibleList Property](#) property.

This property generates an [Exception](#) if the decal to set is invalid, or if the change cannot occur because of violations to the current DRC setting, the Nudge setting, locked test point status, or existing rules.

Examples

The following sample code retrieves the current decal of component C1, assuming it exists in the open design, along with the list of compatible decals to which it can be changed. See [Code Samples](#) for more information on running this sample.

```
Sub Main

    decalList = ActiveDocument.Components("C1").DecalCompatibleList

    decalListString=""

    For index = LBound(decalList) To UBound(decalList)

        If index = 1 Then

            decalListString = decalList(index)

        Else

            decalListString = decalListString & " or " & decalList(index)

        End If

    Next index
```

```
MsgBox "C1 decal is " & ActiveDocument.Components("C1").Decal & " and can  
be changed to : " & decalListString
```

```
End Sub
```

Component.DecalAttributes Property

Returns the collection of all attributes assigned to the component decal.

Usage

DecalAttributes As [Attributes](#) on page 130

DecalAttributes(*name* As String) As [Attribute](#) on page 120

Arguments

<i>name</i>	Name of an existing decal attribute.

Description

When an existing attribute *name* is passed to this property, it returns that decal [Attribute](#) on page 120 object. Otherwise, it returns the collection of all decal attributes in an [Attributes](#) on page 130 collection object.

Examples

The following sample code retrieves the number of attributes assigned to the decal for component U1, assuming it exists in the open design, using the

Component.DecalCompatibleList Property

Returns the list of compatible decals of the component.

Usage

DecalCompatibleList As [Variant](#)

Arguments

None

Description

Use this property to identify all compatible decals for the component before changing the component decal using the [Component.Decal Property](#) property.

Examples

The following sample code retrieves the current decal for component C1, assuming it exists in the open design, along with the list of compatible decals to which it can be changed. See [Code Samples](#) for more information on running this sample.

```
Sub Main

    decalList = ActiveDocument.Components("C1").DecalCompatibleList

    decalListString=""

    For index = LBound(decalList) To UBound(decalList)

        If index = 1 Then

            decalListString = decalList(index)

        Else

            decalListString = decalListString & " or " & decalList(index)

        End If

    Next index

    MsgBox "C1 decal is " & ActiveDocument.Components("C1").Decal & " and can  
be changed to : " & decalListString
```



```
End Sub
```

Component.DieHeight Property

Returns the height of the die.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

DieHeight([*unit* As [PPcbUnit](#) = ppcbUnitCurrent]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the die height value is returned. This optional argument is ppcbUnitCurrent on page 719 by default.

Description

This property generates an [Exception](#) if [Component.IsDiePart Property](#) is False.

Examples

The following sample code retrieves the height of each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

MsgBox "DieHeight (" & comp.Name & "): " & Format (comp.DieHeight,
"#.###")

End If

Next

End Sub
```

Component.DieLength Property

Returns the length of the die.



Restriction:

This information applies to only the BGA toolkit.

Usage

DieLength([*unit* As [PPcbUnit](#) = pcbUnitCurrent]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the die length value is returned. This optional argument is pcbUnitCurrent on page 719 by default.

Description

This property generates an [Exception](#) if [Component.IsDiePart Property](#) is False.

Examples

The following sample code retrieves the length of each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

MsgBox "DieLength (" & comp.Name & "): " & Format (comp.DieLength,
"#.###")

End If

Next

End Sub
```

Component.DieWidth Property

Returns the width of the die.



Restriction:

This information applies to only the BGA toolkit.

Usage

DieWidth(*unit* As [PPcbUnit](#) = ppcbUnitCurrent) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the die width value is returned. This optional argument is ppcbUnitCurrent on page 719 by default.

Descriptions

This property generates an [Exception](#) if [Component.IsDiePart Property](#) is False.

Examples

The following sample code retrieves the width of each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

MsgBox "DieWidth (" & comp.Name & "): " & Format (comp.DieWidth, "#.###")

End If

Next

End Sub
```

Component.Glued Property

Sets or returns whether the component is glued.

Usage

Glued As Boolean

Arguments

None

Description

None

Examples

The following sample code glues all components in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main  
  
For Each nextComp In ActiveDocument.Components  
  
nextComp.Glued = True  
  
Next nextComp  
  
End Sub
```

Component.Installed Property

Sets or returns whether the component is installed in the current assembly option.

Usage

Installed As Boolean

Arguments

None

Description

This property is useful only when the parent of the component is an Assembly Option. If the parent of the component is not an Assembly Option, this property always returns True and cannot be set to a different value.

Examples

The following sample code creates a new Assembly Option named EmptyAssOpt in the open design, and then uninstalls all components in that newly created Assembly Option. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set newAssOpt = ActiveDocument.AssemblyOptions.Add("EmptyAssOpt")

For Each nextComp In newAssOpt.Components

nextComp.Installed = False

Next nextComp

End Sub
```

Component.IsDiePart Property

Returns whether the component is a die.

Usage

IsDiePart As Boolean

Arguments

None

Description

In SailWind Layout, this property always returns FALSE.

Examples

The following sample code retrieves the name of each die presented in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main  
  
For Each comp In ActiveDocument.Components  
  
If comp.IsDiePart Then  
  
MsgBox comp.Name  
  
End If  
  
Next  
  
End Sub
```

Component.IsSMD Property

Returns whether the component is SMD.

Usage

IsSMD As Boolean

Arguments

None

Description

This function returns whether this component is 100% SMD, meaning that all its pins are SMD pins.

Examples

The following sample code retrieves the number of SMD components in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

nbSMDComponents = 0

For Each nextComp In ActiveDocument.Components

If nextComp.IsSMD = True Then nbSMDComponents = nbSMDComponents + 1

Next nextComp

MsgBox "There are " & nbSMDComponents & " SMD components, out of " &
ActiveDocument.Components.Count

End Sub
```

Component.Labels Property

Returns a collection of all labels associated with the component.

Usage

Labels as Collection

Labels (*Name* as String) as Label

Arguments

<i>name</i>	Name of the label to retrieve (optional).

Description

None

Examples

The following sample code displays a message box showing the name of the first label in the selected component. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(pcbObjectTypeComponent,,True)

For Each comp In selected

For Each label In comp.Labels

MsgBox "The first component label is " & label.Text

Exit For

Next label

Exit For

Next comp

End Sub
```


Component.Layer Property

Sets or returns the mounting layer of the component.

Usage

Layer As Long

Arguments

None

Description

This property can only be set to a component layer([PPcbLayerType](#)). Use the [Document.LayerType Property](#) property to identify all valid component layers in the open design.

Use the [Document.LayerName Property](#) property to identify the layer name that matches the returned layer number.

This property generates an [Exception](#) if the layer to set is invalid or is not a component layer.

Examples

The following sample code mounts component U1, assuming it exists in the open design, on the opposite component layer. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set compU1 = ActiveDocument.Components("U1")

For index=1 To Document.LayerCount

If Document.LayerType(index) = ppcbLayerComponent And index <> compU1.layer
Then

compU1.layer = index

Exit For

End If

Next index

End Sub
```

Component.Name Property

Returns the name of the component.

Usage

Name As String

Arguments

None

Description

For example, this property returns the string "U1" for component U1.

This property is the default property for the Component object.

Examples

The following sample code lists all components in the open design and places that list in a custom dialog box. When a component is selected in the list box, the sample selects that component in SailWind Layout.

This sample uses the UserDialog Editor in the Sax Basic Engine in SailWind Layout. See [Code Samples](#) for more information on running this sample.

For more information, see *Sax Basic Editor On Line Help* (C:\<install_folder>\<version>\Programs\sbe5_000.hlp)

```
Dim ListComps$(10000)

Sub Main

    index = 0

    For Each nextComp In ActiveDocument.Components

        ListComps$(index) = nextComp.Name

        index = index + 1

    Next nextComp

    ' This piece of code is automatically generated by the Basic Dialog Editor
    ' in SailWind Layout.

    Begin Dialog UserDialog 180,238,"Components",.CallbackFunc '
        %GRID:10,7,1,1
```

```
ListBox 10,7,160,203,ListComps(),.ListBox1

OKButton 10,210,160,21

End Dialog

Dim dlg As UserDialog

Dialog dlg

End Sub

' The following function is automatically called by the system when
  something has happened in the dialog; it is used to easily process user
  actions.

Function CallbackFunc%(DlgItem$, Action%, SuppValue%)

Select Case Action%

Case 2 ' Value changing or button pressed

If DlgItem$="ListBox1" Then

ActiveDocument.SelectObjects(ppcbObjectTypeAll, , False)

ActiveDocument.SelectObjects(ppcbObjectTypeComponent,
  ListComps(SuppValue%))

End If

End Select

End Function
```

Component.ObjectType Property

Returns the type of the object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

This property returns [ppcbObjectTypeComponent](#) on page 719.

All SailWind Layout database objects in the SailWind Layout Automation server implement this property to compensate for the lack of a Basic equivalent for the Visual C++ QueryInterface function.

This property is generally used:

- To identify the kind of SailWind Layout database objects in a heterogeneous [Objects](#) on page 477 collection.
- When implementing a generic routine that depends on the type of the SailWind Layout database object passed as argument. For example:

```
Sub DoSomething(dbObject As Object)

Select Case dbObject.ObjectType

Case ppcbObjectTypeComponent

' Do something specific to component objects

Case ppcbObjectTypeNet

' Do something specific to net objects

Case ppcbObjectTypePin

' Do something specific to pin objects

Case ppcbObjectTypeVia

' Do something specific to via objects
```

```
Case ppcbObjectTypeConnection
' Do something specific to connection objects

Case ppcbObjectTypeRouteSegment
' Do something specific to route segment objects

Case ppcbObjectTypeJumper
' Do something specific to jumper objects

Case Else
MsgBox "Not a SailWind Layout database object"

End Select

End Sub
```

Component.Orientation Property

Sets or returns the orientation of the component.

Usage

Orientation As Double

Arguments

None

Description

The proper completion of this method depends on the items listed below. To force a component rotation, you must first disable the following:

- Glued status, which is set using the [Component.Glued Property](#) property.
- DRC mode, which is set using the [Document.Preference Property](#) property.
- Nudge mode, which is set using the [Document.Preference Property](#) property.
- ModifyUnionMember mode, which is set using the [Document.Preference Property](#) property.

This method generates an [Exception](#) if a SailWind Layout security failure occurs during processing.

Examples

The following sample code rotates U1 to 45 degrees, assuming U1 exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main  
  
ActiveDocument.Components("U1").Orientation = 45  
  
End Sub
```

Component.Parent Property

Returns the parent of the object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

None

Component.PartType Property

Sets or returns the part type of the component.

Usage

PartType As String

Arguments

Non

Description

This property can be set to a new value only when the parent of the component is an Assembly Option. If the parent is an Assembly Option, this property substitutes the component in the current Assembly Option with a new part type. The new part type must have a decal compatible with the default decal.

This property generates an [Exception](#) if the parent of the component is not an Assembly Option or if the part type is invalid.

Examples

The following sample code retrieves the part type of component U1, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

MsgBox "The part type of U1 is " &
  ActiveDocument.Components("U1").PartType

End Sub
```


Component.PartTypeAttributes Property

Returns the collection of all attributes for the component part type.

Usage

PartTypeAttributes As [Attributes](#) on page 130

PartTypeAttributes(*name* As String) As [Attribute](#) on page 120

Arguments

<i>name</i>	Name of an existing part type attribute.

Description

When an existing attribute *name* is passed to this property, it returns that part type [Attribute](#) on page 120 object. If the attribute *name* does not exist, this property returns the collection of all part type attributes in an [Attributes](#) on page 130 collection object.

Examples

The following sample code retrieves the number of attributes assigned to the part type for component U1, assuming it exists in the open design, using the

Component.PartTypeECORegistered Property

Sets or returns the part type ECO registration status.

Usage

PartTypeECORegistered As Boolean

Arguments

None

Description

Beginning with PowerPCB v3.5 this property becomes obsolete. Use [PartType.ECORegistered Property](#) instead.

Component.PartTypeLogic Property

Returns the logic type of the part type for the component.

Usage

PartTypeLogic As String

Arguments

None

Description

None

Examples

The following sample code retrieves the logic type of the part type for component U1, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

MsgBox "The Logic type of U1 is " &
  ActiveDocument.Components("U1").PartTypeLogic

End Sub
```

Component.PartTypeObject Property

Returns the part type object of this component.

Usage

PartTypeObject As [PartType](#) on page 519

Arguments

None

Description

None

Examples

The following sample code retrieves the part type object for component U1, assuming it exists in the open schematic. See [Code Samples](#) for more information on running this sample.

```
Sub Main

MsgBox "The part type of U1 is " &
    ActiveDocument.Components("U1").PartTypeObject.Name

End Sub
```

Component.Pins Property

Returns the collection of all pins in the component.

Usage

Pins As [Objects](#) on page 477

Pins(*name* As String) As [Pin](#) on page 532

Arguments

<i>name</i>	Name of an existing pin.

Description

When an existing pin *name* is passed to this property, it returns that [Pin](#) on page 532 object. If the pin *name* does not exist, this property returns the collection of all pins of the component in an [Objects](#) on page 477 collection object.

Examples

The following sample code retrieves the number of pins in component U1, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "U1 has " & ActiveDocument.Components("U1").Pins.Count & " pins."
```

```
End Sub
```

Component.Placed Property

Returns whether the component is placed.

Usage

Placed As Boolean

Arguments

None

Description

A component is considered *placed* if all its pins are inside the board outline, but outside a board cut out.

Examples

The following sample code determines whether component U1 is placed or not, assuming U1 exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Select Case ActiveDocument.Components("U1").Placed

Case True

MsgBox "Component U1 is placed"

Case False

MsgBox "Component U1 is not placed"

End Select

End Sub
```

Component.PositionX Property

Returns the x-coordinate of the component.

Usage

PositionX(*unit* As [PPcbUnit](#)) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the x-coordinate is returned.

Description

None

Examples

The following sample code retrieves the location of component U1, assuming it exists in the open design, in current design units. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set compU1 = ActiveDocument.Components("U1")
```

```
MsgBox "U1 position is = (" & compU1.PositionX & ", " & compU1.PositionY &
    ")"
```

```
End Sub
```

Component.PositionY Property

Returns the y-coordinate of the component.

Usage

PositionY(*[unit As PPcbUnit]*) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the y-coordinate is returned.

Description

None

Examples

The following sample code retrieves the location of component U1, assuming it exists in the open design, in current design units. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set compU1 = ActiveDocument.Components("U1")
```

```
MsgBox "U1 position is = (" & compU1.PositionX & ", " & compU1.PositionY &
    ")"
```

```
End Sub
```

Component.SBPs Property

Returns the die's SBP collection.



Restriction:

This information applies to only the BGA toolkit.

Usage

SBPs As Objects

Arguments

None

Description

This property generates an [Exception](#) if [Component.IsDiePart Property](#) is False.

Examples

The following sample code iterates through SBPs for each die component in the open design and prints the names of the SBPs. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aSBP In comp.SBPs

Debug.Print aSBP.Name

Next

End If

Next

End Sub
```

Component.Selected Property

Sets or returns whether the component is selected.

Usage

Selected As Boolean

Arguments

None

Description

You can also select a SailWind Layout database object using the [Document.SelectObjects Method](#) and [Objects.Select Method](#) methods.

Examples

The following sample code selects component U1 only, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.SelectObjects(, , False)
```

```
ActiveDocument.Components("U1").Selected = True
```

```
End Sub
```

Component.Substituted Property

Returns whether the component is substituted in the current Assembly Option.

Usage

Substituted As String

Arguments

None

Description

This property is useful only when the parent of the component is an Assembly Option. If the parent of the component is not an Assembly Option, this property always returns False.

To substitute a component in an Assembly Option, use the [Component.Substituted Property](#) property.

Examples

The following sample code retrieves the number of substituted components in the MyAssOpt Assembly Option, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set myAssOpt = ActiveDocument.AssemblyOptions("MyAssOpt")

nbSubstComp = 0

For Each nextComp In myAssOpt.Components

    If nextComp.Substituted = True Then nbSubstComp = nbSubstComp + 1

Next nextComp

MsgBox "There are " & nbSubstComp & " substituted components in " &
    myAssOpt.Name

End Sub
```

Component.WireBondRulesAngleMaximum Property

Returns the bond wire maximum angle rule of the die, in degrees.



Restriction:

This information applies to only the BGA toolkit.

Usage

WireBondRulesAngleMaximum As Double

Arguments

None

Description

This property generates an [Exception](#) if [Component.IsDiePart Property](#) is False.

Examples

The following sample code retrieves the maximum angle rule for each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

MsgBox "MaxAngleRule: " & Format (comp.WireBondRulesAngleMaximum, "#.###")

End If

Next

End Sub
```

Component.WireBondRulesClearanceWireToPad Property

Returns the bond wire-to-pad clearance rule of the die.



Restriction:

This information applies to only the BGA toolkit.

Usage

WireBondRulesClearanceWireToPad([unit As [PPcbUnit](#) = ppcbUnitCurrent]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the clearance value is returned.

Description

This property generates an [Exception](#) if [Component.IsDiePart Property](#) is False.

Examples

The following sample code retrieves the wire-to-pad clearance rule for each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

MsgBox "WireToPad: " & Format (comp.WireBondRulesClearanceWireToPad,
    "#.###")

End If

Next

End Sub
```

Component.WireBondRulesClearanceWireToWire Property

Returns the bond wire-to-wire clearance rule of the die.



Restriction:

This information applies to only the BGA toolkit.

Usage

WireBondRulesClearanceWireToWire([*unit* As [PPcbUnit](#) = ppcbUnitCurrent]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the clearance value is returned.

Description

This property generates an [Exception](#) if [Component.IsDiePart Property](#) is False.

Examples

The following sample code retrieves the wire-to-wire clearance rule for each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

'For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

MsgBox "WireToWire: " & Format
(comp.WireBondRulesClearanceWireToWire, "#.###")

End If

Next

End Sub
```

Component.WireBondRulesLengthMaximum Property

Returns the bond wire maximum length rule of the die.



Restriction:

This information applies to only the BGA toolkit.

Usage

WireBondRulesLengthMaximum([*unit* As [PPcbUnit](#) = ppcbUnitCurrent]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the length value is returned.

Description

This property generates an [Exception](#) if [Component.IsDiePart Property](#) is False.

Examples

The following sample code retrieves the maximum length rule for each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

MsgBox "MaxLengthRule: " & Format (comp.WireBondRulesLengthMaximum,
    "#.###")

End If

Next

End Sub
```

Component.WireBondRulesLengthMinimum Property

Returns the bond wire minimum length rule of the die.



Restriction:

This information applies to only the BGA toolkit.

Usage

WireBondRulesLengthMinimum(*[unit As PPcbUnit = ppcbUnitCurrent]*) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the length value is returned.

Description

This property generates an [Exception](#) if [Component.IsDiePart Property](#) is False.

Examples

The following sample code retrieves the minimum length rule for each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

MsgBox "MinLengthRule: " & Format (comp.WireBondRulesLengthMinimum,
"#.###")

End If

Next

End Sub
```

Component.Wirebonds Property

Returns the bond wire collection of the die.



Restriction:
This information applies to only the BGA toolkit.

Usage

Wirebonds As Objects

Arguments

None

Description

None

Examples

The following sample code iterates through all bond wires in each die component in the open design and prints the bond wire names. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each nWB In comp.Wirebonds
```



```
Debug.Print nWB.Name
```

```
Next
```

```
End If
```

```
Next
```

```
End Sub
```

Component.AddLabel Method

Adds a new label to the component.

Usage

AddLabel (*Type* as [PPcbLabelType](#), *Attribute* as Object, *Layer* as Long, *PositionX* as Double, *PositionY* as Double, *Height* as Double, *LineWidth* as Double, *Units* as [PPcbUnit](#), *Orientation* as Double, *Mirror* as Boolean) as Label

Arguments

<i>type</i>	Type of new label (optional). Default value is ppcbLabelTypeRefDesignator on page 719.
<i>attribute</i>	Attribute object to which the label is linked (optional).
<i>layer</i>	Number of layer on which to place the new label (optional). The default is the component layer.
<i>positionX</i>	x-coordinate of the new object relative to the component's origin (optional). Default is 0.
<i>positionY</i>	y-coordinate of the new object relative to the component's origin (optional). Default is 0.
<i>height</i>	Height of the label (optional). If omitted, the value specified in the Tools/Options/Drafting tab/Reference Designators frame. The Height edit box is used.
<i>linewidth</i>	Line width of the label. If omitted, the value specified in the Tools/Options/Drafting tab/Reference Designators frame. The Width edit box is used.
<i>units</i>	[Optional on page 732] Unit in which PositionX, PositionY, Height, and LineWidth are returned. Default is ppcbUnitCurrent on page 719.
<i>orientation</i>	Orientation of the label (optional). The default is 0.
<i>mirror</i>	True if the label is mirrored (optional). The default is False.

Return Values

Returns the newly added label object.

Description

None

Examples

The following sample code adds a reference designator to a U1 component at a position (200 mils, 200 mils) relative to the component's origin. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Dim c as Component
```

```
Set c = Document.Components("U1")
```

```
c.AddLabel(ppcbLabelTypeRefDesignator, , , 200, 200, , , ppcbUnitMils)
```

```
End Sub
```

Component.Move Method

Moves the component to a new location.

Usage

Move(*x* As Double, *y* As Double, [*unit* As [PPcbUnit](#)])

Arguments

<i>x</i>	x-coordinate of the new component position.
<i>y</i>	y-coordinate of the new component position.
<i>unit</i>	[Optional on page 732] Unit in which the x and y coordinates are expressed. This optional argument is ppcbUnitCurrent on page 719 by default.

Return Values

None

Description

The proper completion of this method depends on the items listed below. To force a component move, you must first disable the following:

- Glued status, which is set using the [Component.Glued Property](#) property.
- DRC mode, which is set using the [Document.Preference Property](#) property.
- Nudge mode, which is set using the [Document.Preference Property](#) property.
- ModifyUnionMember mode, which is set using the [Document.Preference Property](#) property.

In addition, a component cannot be moved if it belongs to a physical design reuse component or if it is attached to protected traces.

This property generates an [Exception](#) if any of the above constraints are violated or if a security failure occurs during processing.

Examples

The following sample code moves all components to the origin of the open design, after ungluing them and disabling all automatic DRC and Nudge checking options. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.Preference("DRC") = ppcbDRCOff
```

```
ActiveDocument.Preference("Nudge") = ppcbNudgeOff
```

```
For Each nextComp In ActiveDocument.Components
```

```
nextComp.Glued = False
```

```
nextComp.Move(0,0)
```

```
Next nextComp
```

```
End Sub
```

The following sample code places all components, in the open design, in a circle centered on the design origin, with a 5000 mils radius. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.Preference("DRC") = ppcbDRCOff
```

```
ActiveDocument.Preference("Nudge") = ppcbNudgeOff
```

```
angleInc = 2*3.14159/ActiveDocument.Components.Count
```

```
counter = 0
```

```
For Each nextComp In ActiveDocument.Components
```

```
nextComp.Glued = False
```

```
nextComp.Move(5000.0*Cos(counter*angleInc), 5000.0*Sin(counter*angleInc),  
ppcbUnitMils)
```

```
counter = counter+1
```

```
Next nextComp
```

```
End Sub
```

Component.MoveCenter Method

Moves the component so that the specified point becomes the component's center.

Usage

MoveCenter (x as Double, y as Double, *unit* as PPcbUnit, Origin as PPcbOriginType)

Arguments

<i>x</i>	x-coordinate of new center.
<i>y</i>	y-coordinate of new center.
<i>unit</i>	[Optional on page 732] Unit in which the x and y coordinates are expressed. This optional argument is ppcbUnitCurrent on page 719 by default.
<i>origin</i>	[Optional on page 732]Type of reference point from which the result is counted. The default value is ppcbOriginTypeDesign on page 719.

Return Values

None

Description

This method moves the component by x and y coordinates simultaneously, as opposed to assigning values to CenterX and CenterY properties.

Examples

The following sample code moves component U1 to the point with coordinates (3200 mils, 1500 mils). See [Code Samples](#) for more information on running this sample.

```
Sub Main

Dim c as Component

Set c = Document.Components("U1")

c.MoveCenter(3200, 1500, pcbUnitMils)

End Sub
```

The Connection Object

The Connection object represents a physical connection, also called a pin pair, in the open design.

The Connection object contains the following properties:

- Connection.Application Property
- Connection.Length Property
- Connection.Name Property
- Connection.Net Property
- Connection.ObjectType Property
- Connection.Parent Property
- Connection.Pins Property
- Connection.RouteSegments Property
- Connection.Selected Property
- Connection.Vias Property

Connection.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This property identifies the object as an Automation object. All Automation server applications have an Application object and all Automation objects have an Application property. This property is usually used in Automation client applications that handle large volumes of objects from different sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

Connection.Length Property

Returns the length of the connection.

Usage

Length(*[unit As PPcbUnit]*) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the length value is returned.

Description

None

Examples

The following sample code retrieves the length of the first connection found in the open design, assuming at least one connection exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set firstConn = ActiveDocument.Connections(1)
```

```
MsgBox "Connection " & firstConn.Name & " has a length of " &  
firstConn.Net.Length
```

```
End Sub
```

Connection.Name Property

Returns the name of the connection.

Usage

Name As String

Arguments

None

Description

This property is the default property for the Connection object.

Examples

The following sample code lists all connections in the open design by name and then places that list in a custom dialog box. When a connection is selected in the list box, the sample selects that connection.

This sample uses the UserDialog Editor in the Sax Basic Engine. See [Code Samples](#) for more information on running this sample.

For more information, see *Sax Basic Editor On Line Help* (C:\<install_folder>\<version>\Programs\sbe5_000.hlp)

```
Dim ListConns$(10000)

Sub Main

    index = 0

    For Each nextConn In ActiveDocument.Connections

        ListConns$(index) = nextConn.Name

        index = index + 1

    Next nextConn

    ' This piece of code is automatically generated by the Basic Dialog Editor
    ' in SailWind Layout.

    Begin Dialog UserDialog 180,238,"Connections",.CallbackFunc '
        %GRID:10,7,1,1

        ListBox 10,7,160,203,ListConns(),.ListBox1
```

```
OKButton 10,210,160,21
```

```
End Dialog
```

```
Dim dlg As UserDialog
```

```
Dialog dlg
```

```
End Sub
```

```
' The following function is automatically called by the system when  
something has happened
```

```
' in the dialog; it is used to easily process user actions.
```

```
Function CallbackFunc%(DlgItem$, Action%, SuppValue%)
```

```
Select Case Action%
```

```
Case 2 ' Value changing or button pressed
```

```
If DlgItem$="ListBox1" Then
```

```
ActiveDocument.SelectObjects(ppcbObjectTypeAll, , False)
```

```
ActiveDocument.SelectObjects(ppcbObjectTypeConnection,  
ListConns(SuppValue%))
```

```
End If
```

```
End Select
```

```
End Function
```

Connection.Net Property

Returns the net connected to the connection.

Usage

Net As [Net](#) on page 452

Arguments

None

Description

None

Examples

The following sample code retrieves the net connected to the first connection found in the open design, assuming at least one connection exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set firstConn = ActiveDocument.Connections(1)
```

```
MsgBox "Connection " & firstConn.Name & " is connected to net " &  
firstConn.Net.Name
```

```
End Sub
```

Connection.ObjectType Property

Returns the type of the object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

This property returns [ppcbObjectTypeConnection](#) on page 719.

All database objects in the SailWind Layout Automation server implement this property to compensate for the lack of a Visual Basic equivalent for the Visual C++ QueryInterface function.

This property is generally used:

- To identify the kind of database objects in a heterogeneous [Objects](#) on page 477 collection.
- When implementing a generic routine that depends on the type of the database object passed as argument. For example:

```
Sub DoSomething(dbObject As Object)

Select Case dbObject.ObjectType

Case ppcbObjectTypeComponent

' Do something specific to component objects

Case ppcbObjectTypeNet

' Do something specific to net objects

Case ppcbObjectTypePin

' Do something specific to pin objects

Case ppcbObjectTypeVia

' Do something specific to via objects
```

```
Case ppcbObjectTypeConnection  
  
    ' Do something specific to connection objects  
  
Case ppcbObjectTypeRouteSegment  
  
    ' Do something specific to route segment objects  
  
Case ppcbObjectTypeJumper  
  
    ' Do something specific to jumper objects  
  
Case Else  
  
    MsgBox "Not a SailWind Layout database object"  
  
End Select  
  
End Sub
```

Connection.Parent Property

Returns the parent of the object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

None

Connection.Pins Property

Returns the collection of all pins in the connection.

Usage

Pins As [Objects](#) on page 477

Pins(*name* As String) As [Pin](#) on page 532

Arguments

<i>name</i>	Name of an existing pin.

Description

When an existing pin name is passed to this property, it returns that [Pin](#) on page 532 object. If the pin name does not exist, this property returns the collection of all pins of the connection in an [Objects](#) on page 477 collection object.

Examples

The following sample code retrieves the number of pins in the first connection found in the open design, assuming at least one connection exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set firstConn = ActiveDocument.Connections(1)
```

```
MsgBox "Connection " & firstConn.Name & " connects " & firstConn.Pins.Count  
& " pins."
```

```
End Sub
```


Connection.RouteSegments Property

Returns the collection of all trace segments in the connection.

Usage

RouteSegments As [Objects](#) on page 477

RouteSegments(*name* As String) As [RouteSegment](#) on page 582

Arguments

<i>name</i>	Name of an existing trace segment.

Description

When an existing trace segment *name* is passed to this property, it returns that [RouteSegment](#) on page 582 object. If the trace segment *name* does not exist, this property returns the collection of all route segments in the connection in an [Objects](#) on page 477 collection object.

Examples

The following sample code retrieves the number of trace segments in the first connection found in the open design, assuming at least one connection exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set firstConn = ActiveDocument.Connections(1)
```

```
MsgBox "Connection " & firstConn.Name & " includes " &  
firstConn.RouteSegments.Count & " route segments."
```

```
End Sub
```

Connection.Selected Property

Sets or returns whether the connection is selected.

Usage

Selected As Boolean

Arguments

None

Description

None

Examples

The following sample code selects the first connection found in the open design, assuming at least one connection exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main  
  
ActiveDocument.SelectObjects(, , False)  
  
ActiveDocument.Connections(1).Selected = True  
  
End Sub
```

Connection.Vias Property

Returns the collection of all vias in the connection.

Usage

Vias As [Objects](#) on page 477

Vias(*name* As String) As [Via](#) on page 657

Arguments

<i>name</i>	Name of an existing via.

Description

When an existing via *name* is passed to this property, it returns that [Via](#) on page 657 object. If the via *name* does not exist, this property returns the collection of all vias in the collection in an [Objects](#) on page 477 collection object.

Examples

The following sample code retrieves the number of vias in the first connection found in the open design, assuming at least one connection exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set firstConn = ActiveDocument.Connections(1)
```

```
MsgBox "Connection " & firstConn.Name & " includes " & firstConn.Vias.Count  
& " vias."
```

```
End Sub
```

The Decal Object

The Decal object represents a decal, in the currently open design.

The Decal object contains the following properties:

- [Decal.Application Property](#)
- [Decal.Attributes Property](#)
- [Decal.Components Property](#)
- [Decal.LibraryTimeStamp Property](#)
- [Decal.Name Property](#)
- [Decal.ObjectType Property](#)
- [Decal.Parent Property](#)
- [Decal.Selected Property](#)
- [Decal.TimeStamp Property](#)

Decal.Application Property

Returns the application object.

Usage

Application as [Application](#) on page 68

Arguments

None

Decal.Attributes Property

Returns the Decals's collection of attributes.

Usage

Attributes (*name* As String) As Attribute

Arguments

None

Decal.Components Property

Returns the collection of Components objects of this decal.

Usage

Components (*name* As String) As Component

Arguments

None

Decal.LibraryTimeStamp Property

Returns the Decal's library timestamp.

Usage

TimeStamp as Date

Arguments

None

Decal.Name Property

Returns the Decal's name.

Usage

Name as String

Arguments

None

Decal.ObjectType Property

Returns the type of this database object: a decal.

Usage

ObjectType as [PPcbObjectType](#)

Arguments

None

Decal.Parent Property

Returns the parent of the object.

Usage

Parent as Document

Arguments

None

Decal.Selected Property

Sets or returns the Decal's selection status.

Usage

Selected as Boolean

Arguments

None

Decal.TimeStamp Property

Returns the Decal's timestamp.

Usage

TimeStamp as Date

Arguments

None

The Document Object

The Document object represents a currently open PCB design file.

This object is usually retrieved using the [Application.ActiveDocument Property](#) property.

The Document object contains the following properties, methods, and events:

- [Document.ActiveLayer Property](#)
- [Document.ActiveView Property](#)
- [Document.Application Property](#)
- [Document.AssemblyOptions Property](#)
- [Document.AssociatedNets Property](#)
- [Document.Attributes Property](#)
- [Document.BoardOutlineSurface Property](#)
- [Document.Components Property](#)
- [Document.Connections Property](#)
- [Document.Drawings Property](#)
- [Document.ElectricalLayerCount Property](#)
- [Document.Errors Property](#)
- [Document.FullName Property](#)
- [Document.GridX Property](#)
- [Document.GridY Property](#)
- [Document.Jumpers Property](#)
- [Document.LayerCount Property](#)
- [Document.LayerEnabled Property](#)
- [Document.LayerName Property](#)
- [Document.Layers Property](#)
- [Document.LayerType Property](#)
- [Document.MaxRealValue Property](#)
- [Document.MinRealValue Property](#)
- [Document.Name Property](#)
- [Document.NetClasses Property](#)
- [Document.Nets Property](#)
- [Document.ObjectType Property](#)
- [Document.OriginX Property](#)
- [Document.OriginY Property](#)
- [Document.Parent Property](#)
- [Document.PartTypes Property](#)
- [Document.Path Property](#)
- [Document.Pins Property](#)
- [Document.Preference Property](#)
- [Document.RouteSegments Property](#)
- [Document.Saved Property](#)
- [Document.Texts Property](#)

Document.Unit Property
Document.Vias Property
Document.Activate Method
Document.AddText Method
Document.CheckASCII Method
Document.ExportASCII Method
Document.ExportECOFile Method
Document.ExportNetList Method
Document.ExportRules Method
Document.GetColor Method
Document.GetObjects Method
Document.GetVisibility Method
Document.ImportECOFile Method
Document.ImportNetList Method
Document.IntegrityTest Method
Document.Save Method
Document.SaveAs Method
Document.SaveAsNoLock Method
Document.SaveNoLock Method
Document.SaveAsTemp Method
Document.SaveTemp Method
Document.SelectObjects Method
Document.SetColor Method
Document.SetVisibility Method
Document.PositionsChange Event
Document.Save Event
Document.SecurityLimit Event
Document.SelectionChange Event

Document.ActiveLayer Property

Sets or returns the active layer.

Usage

ActiveLayer As Long

Arguments

None

Description

When the layer with the given number exists in the currently opened design and it is enabled then it's set as active layer.

When we set the property to 0, <All layers> is set as active layer.

In other cases nothing happens.

Examples

The following sample code retrieves the current active layer and then sets the second layer as the active one. See "[Running Code Samples](#) on page 734" for more information on running this sample.

```
Sub Main
```

```
MsgBox "Current active layer " & ActiveDocument.ActiveLayer
```

```
ActiveDocument.ActiveLayer = 2
```

```
End Sub
```


Document.ActiveView Property

Returns the active view.

Usage

ActiveView As [View](#) on page 681

Arguments

None

Description

The active view represents the main view of the open design.

Examples

The following sample code pans the current view to the origin of the open design using the [View.Pan Method](#) method. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.ActiveView.Pan(0,0)
```

```
End Sub
```

Document.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This property identifies the object as a SailWind Layout Automation object. All Automation server applications have an Application object and all Automation objects have an Application property. This property is usually used in Automation client applications that handle large volumes of objects from different sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

Document.AssemblyOptions Property

Returns the collection of all assembly options.

Usage

AssemblyOptions As [AssemblyOptions](#) on page 99

AssemblyOptions(*name* As String) As [Document](#) on page 250

Arguments

<i>name</i>	Name of an existing assembly option.

Description

When an existing assembly option *name* is passed to this property, it returns that assembly option object, packaged as a [Document](#) on page 250 object. If the Assembly Option *name* does not exist, this property returns the collection of all assembly options existing in the open design, packaged as an [AssemblyOptions](#) on page 99 collection object.

The [Document.Name Property](#) property of an assembly option document is in the following format: Assembly Option Name:PCB Design File Name, for example WithoutU1:PWRDEMOA.PCB.



Tip

Assembly options are always packaged as [Document](#) on page 250 objects.

Examples

The following sample code retrieves the number of assembly options in the open design using the [AssemblyOptions.Count Property](#) property. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set assopts = ActiveDocument.AssemblyOptions
```

```
MsgBox "There are " & assopts.Count & " assembly option(s) in " &  
ActiveDocument.Name
```

```
End Sub
```

The following sample code retrieves the number of uninstalled and substituted components in the assembly option named MyAssOpt, assuming it exists in the open design, using the [Component.Installed Property](#) and [Component.Substituted Property](#) properties. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
nbUninstalled = 0
```

```
nbSubstituted = 0
```

```
For Each nextComp In ActiveDocument.AssemblyOptions("MyAssOpt")
```

```
If nextComp.Installed = False Then nbUninstalled = nbUninstalled + 1
```

```
If nextComp.Substituted = True Then nbSubstituted = nbSubstituted + 1
```

```
Next nextComp
```

```
MsgBox "There are " & nbUninstalled & " uninstalled and " & nbSubstituted &  
" substituted components in assembly option MyAssOpt."
```

```
End Sub
```

Document.AssociatedNets Property

Returns the collection of all associated nets.

Usage

```
AssociatedNets as Objects
```

```
AssociatedNets (name as String) as AssociatedNet
```

Arguments

Name Optional

Name of an existing associated net

Return Values

When an existing associated net name is passed to this property, it returns the associated net object with that name. If the associated net name is not specified, this property returns the collection of all associated nets in Objects.

Document.Attributes Property

Returns the collection of all attributes of the document.

Usage

Attributes As [Attributes](#) on page 130

Attributes(*name* As String) As [Attribute](#) on page 120

Arguments

<i>name</i>	Name of an existing document attribute.

Description

When an existing attribute *name* is passed to this property, it returns that document [Attribute](#) on page 120 object. If the attribute *name* does not exist, this property returns the collection of all document attributes in an [Attributes](#) on page 130 collection object.

Examples

The following sample code retrieves the number of document attributes in the open design using the

Document.BoardOutlineSurface Property

Returns the surface of the document board outline.

Usage

BoardOutlineSurface([*unit* As [PPcbUnit](#)]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the board outline surface value is returned.

Description

The board outline surface is calculated disregarding board outline cut outs or holes.

Examples

The following sample code retrieves the board outline surface of the open design. See [Code Samples](#) for more information on running this sample.

```
Function UnitName(unit As Long) As String
```

```
    Select Case unit
```

```
        Case ppcbUnitMils
```

```
            UnitName="mils"
```

```
        Case ppcbUnitInch
```

```
            UnitName="inches"
```

```
        Case ppcbUnitMetric
```

```
            UnitName="mm"
```

```
        Case Else
```

```
            UnitName="unknown"
```

```
    End Select
```

```
End Function
```

```
Sub Main
```

```
MsgBox "The board outline surface is " & ActiveDocument.BoardOutlineSurface  
& " square " & UnitName(ActiveDocument.unit)
```

```
End Sub
```


Document.Components Property

Returns the collection of all components.

Usage

Components As [Objects](#) on page 477

Components(*name* As String) As [Component](#) on page 178

Arguments

<i>name</i>	Name of an existing component.

Description

When an existing component *name* is passed to this property, it returns that [Component](#) on page 178 object. If the component *name* does not exist, this property returns the collection of all components in an [Objects](#) on page 477 collection object.

Examples

The following sample code retrieves the number of components in the open design using the [Objects.Count Property](#) property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set comps = ActiveDocument.Components

MsgBox "There are " & comps.Count & " component(s) in " &
ActiveDocument.Name

End Sub
```

The following sample code retrieves the number of pins in component U1, assuming it exists in the open design, using the [Component.Pins Property](#) and [Objects.Count Property](#) properties. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set compU1 = ActiveDocument.Components("U1")

MsgBox "Component " & compU1.Name & " has " & compU1.Pins.Count & "
pin(s)."
```

```
End Sub
```

Document.Connections Property

Returns the collection of all connections.

Usage

Connections As [Objects](#) on page 477

Connections(*name* As String) As [Connection](#) on page 227

Arguments

<i>name</i>	Name of an existing connection.

Description

When an existing connection name is passed to this property, it returns that [Connection](#) on page 227 object. If the connection name does not exist, this property returns the collection of all connections in an [Objects](#) on page 477 collection object.

Examples

The following sample code retrieves the number of connections in the open design using the [Objects.Count Property](#) property. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set conns = ActiveDocument.Connections
```

```
MsgBox "There are " & conns.Count & " connection(s) in " &  
ActiveDocument.Name
```

```
End Sub
```

The following sample code retrieves the sum of the length (routed and unrouted) of all connections in the open design using the [Connection.Length Property](#) property. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
totalLength = 0.0
```

```
For Each nextConn In ActiveDocument.Connections
```

```
totalLength = totalLength + nextConn.Length
```

```
Next nextConn
```

```
MsgBox "The sum of all connection length of " & ActiveDocument.Name & " is  
" & totalLength
```

```
End Sub
```

Document.Drawings Property

Returns the collection of drawings in the document or a specific drawing.

Usage

Drawings as Collection

Drawings (*Name* as String) as Drawing

Arguments

<i>name</i>	Name of the drawing to retrieve (optional).

Description

None

Examples

See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "Number of Drawings: " & ActiveDocument.Drawings.Count
```

```
End Sub
```

Document.ElectricalLayerCount Property

Returns the number of electrical layers in the design.

Usage

ElectricalLayerCount as Boolean

Arguments

None

Description

None

Examples

This sample shows the number of electrical layers in the design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "El. layer count: " & ActiveDocument.ElectricalLayerCount
```

```
End Sub
```

Document.Errors Property

Returns the number of electrical layers in the design.

Usage

Errors as Objects

Errors (errorNumber as Integer) as Error

Arguments

<i>errorNumber</i>	error number

Description

When the error number is passed to this property, it returns that Error Object. If the number is not specified, this property returns the collection of all errors in an Objects collection object.

Document.FullName Property

Returns the file name of the document, including its path.

Usage

FullName As String

Arguments

None

Description

None

Examples

The following sample code retrieves the name and location of the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "Hi, you are using " & ActiveDocument.FullName & " located in " &  
ActiveDocument.Path
```

```
End Sub
```

Document.GridX Property

Sets or returns the X grid of the document.

Usage

GridX([*type* As [PPcbGridType](#) = ppcbGridDesign], [*unit* As [PPcbUnit](#)]) As Double

Arguments

<i>type</i>	[Optional on page 732] Type of grid to set or return.
<i>unit</i>	[Optional on page 732] Unit in which the grid value is set or returned.

Description

None

Examples

The following sample code sets the display grid of the open design to the same value as the routing grid. See [Code Samples](#) for more information on running this sample.

```
Sub Main

curGridX = ActiveDocument.GridX

curGridY = ActiveDocument.GridY

ActiveDocument.GridX(ppcbGridDisplay) = curGridX

ActiveDocument.GridY(ppcbGridDisplay) = curGridY

End Sub
```


Document.GridY Property

Sets or returns the Y grid of the document.

Usage

GridY([*type* As [PPcbGridType](#) = ppcbGridDesign], [*unit* As [PPcbUnit](#)]) As Double

Arguments

<i>type</i>	[Optional on page 732] Type of grid to set or return.
<i>unit</i>	[Optional on page 732] Unit in which the grid value is set or returned.

Description

None

Examples

The following sample code sets the display grid of the open design to the same value as the routing grid. See [Code Samples](#) for more information on running this sample.

```
Sub Main

curGridX = ActiveDocument.GridX

curGridY = ActiveDocument.GridY

ActiveDocument.GridX(ppcbGridDisplay) = curGridX

ActiveDocument.GridY(ppcbGridDisplay) = curGridY

End Sub
```

Document.Jumpers Property

Returns the collection of all jumpers.

Usage

Jumpers As [Objects](#) on page 477

Jumpers(*name* As String) As [Jumper](#) on page 372

Arguments

<i>name</i>	Name of an existing jumper.

Description

When an existing jumper *name* is passed to this property, it returns that [Jumper](#) on page 372 object. If the jumper *name* does not exist, this property returns the collection of all jumpers in an [Objects](#) on page 477 collection object.

Examples

The following sample code retrieves the number of jumpers in the open design using the [Objects.Count Property](#) property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set jmps = ActiveDocument.Jumpers

MsgBox "There are " & jmps.Count & " jumpers(s) in " & ActiveDocument.Name

End Sub
```

The following sample code retrieves the sum of the length of all jumpers in the open design using the [Jumper.Length Property](#) property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

totalLength = 0.0

For Each nextJmp In ActiveDocument.Jumpers

totalLength = totalLength + nextJmp.Length

Next nextJmp
```

```
MsgBox "The sum of all jumper length of " & ActiveDocument.Name & " is " &  
totalLength
```

```
End Sub
```

Document.LayerCount Property

Returns the total number of layers in the document.

Usage

LayerCount As Long

Arguments

None

Description

Layer 0 is not considered a valid layer in the Automation server.

Examples

The following sample code retrieves the total number of layers in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox ActiveDocument.LayerCount & " layers in this PCB design."
```

```
End Sub
```

Document.LayerEnabled Property

Returns the enabled status of a layer.

Usage

LayerEnabled(*layer* as Long) as Boolean

Arguments

<i>name</i>	Number of layer (required).

Description

None

Examples

This sample shows the enabled status of layer 17 in the design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "Is layer 17 enabled? " & ActiveDocument.LayerEnabled(17)
```

```
End Sub
```

Document.LayerName Property

Returns the layer name of a layer number.

Usage

LayerName(*layer* As Long) As String

Arguments

<i>layer</i>	Layer

Description

Layer 0 is not considered a valid layer in the Automation server.

This property generates an [Exception](#) if the layer argument is not a valid layer number.

Examples

The following sample code retrieves the name of layer 2 in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox ActiveDocument.LayerName(2) & " is layer 2."
```

```
End Sub
```

Document.Layers Property

Returns the collection of all layers in the design.

Usage

Layers as Objects

Layers(*layerNumber* as Integer) as Layer

Layers(*layerName* as String) as Layer

Arguments

<i>layerNumber</i>	Number of the layer
<i>layerName</i>	Name of the layer

Return Values

When a layer number/name is passed to this property, it returns that Layer Object. If the number/name is not specified, this property returns the collection of all layers in an Objects collection object.

Examples

```
doc = Application.ActiveDocument
```

```
msg=""
```

```
For Each layer In doc.Layers
```

```
    msg = msg & layer.Number & ", " & layer.Name & ", "
```

```
        & layer.Type & ", " & layer.PlaneType & ", "
```

```
        & layer.RoutingDirection & ", " & layer.Visible & ", "
```

```
        & layer.Enabled & ", "
```

```
        & layer.GetColor(ppcbLayerColorPad) & ", "
```

```
        & layer.CopperThickness & ", "
```

```
& layer.GetDielectricThickness(ppcbDielectricLayerAbove)

& ", "

& layer.GetDielectricConstant(ppcbDielectricLayerBelow)

msg = msg & chr(13)


Next layer


MsgBox msg


layer = doc.Layers(1)

curr_pad_color = layer.GetColor(ppcbLayerColorPad)

layer.SetColor(ppcbLayerColorPad, 10)


MsgBox "Press any key"


layer.Visible = false


MsgBox "Press any key"


layer.Visible = true


MsgBox "Press any key"


layer.SetColor(ppcbLayerColorPad, curr_pad_color)
```


Document.LayerType Property

Returns the type of a given layer number.

Usage

LayerType(*layer* As Long) As [PPcbLayerType](#)

Arguments

<i>layer</i>	Layer

Description

Layer 0 is not considered a valid layer in the Automation server.

This property generates an [Exception](#) if the layer argument is not a valid layer number.

Examples

The following sample code retrieves the type of layer 2 in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub MainDim strType As StringSelect Case ActiveDocument.LayerType(2)Case ppcbLayerComponentstrType="Component layer"Case ppcbLayerRoutingstrType="Routing layer"Case ppcbLayerDrillstrType="Drill layer"Case ppcbLayerSilkscreenstrType="Silkscreen layer"Case ppcbLayerPasteMaskstrType="Paste mask layer"Case ppcbLayerSolderMaskstrType="Solder mask layer"Case ppcbLayerAssemblystrType="Assembly layer"Case ppcbLayerGeneralstrType="General layer"Case ElsestrType="Unknown layer type"End SelectMsgBox "Layer 2 is a " & strTypeEnd Sub
```

Document.MaxRealValue Property

Returns the maximum real value in the system.

Usage

MaxRealValue as double

Arguments

None

Document.MinRealValue Property

Returns the minimum real value in the system.

Usage

MinRealValue as double

Arguments

None

Document.Name Property

Returns the name of the document. For example, if the current design file is \SailWind Projects\Samples\pwrdemoa.pcb, this function returns the string "pwrdemoa.pcb".

Usage

Name As String

Arguments

None

Description

This property is the default property for the [Document](#) on page 250 object.

Examples

The following sample code retrieves the name and location of the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "Hi, you are using " & ActiveDocument.Name & " located in " &  
ActiveDocument.Path
```

```
End Sub
```

Document.NetClasses Property

Returns the collection of net classes in the document or in a specific net class.

Usage

NetClasses as Collection

NetClasses (*Name* as String) as NetClass

Arguments

<i>name</i>	Name of the net class to retrieve.

Description

None

Examples

This sample shows the total number of net classes in the design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "Number of Net Classes: " & ActiveDocument.NetClasses.Count
```

```
End Sub
```

Document.Nets Property

Returns the collection of all nets.

Usage

Nets As [Objects](#) on page 477

Nets(*name* As String) As [Net](#) on page 452

Arguments

<i>name</i>	Name of an existing net.

Description

When an existing net *name* is passed to this property, it returns that [Net](#) on page 452 object. If the net *name* does not exist, this property returns the collection of all nets in an [Objects](#) on page 477 collection object.

Examples

The following sample code retrieves the number of nets in the open design using the [Objects.Count Property](#) property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set nets = ActiveDocument.Nets

MsgBox "There are " & nets.Count & " net(s) in " & ActiveDocument.Name

End Sub
```

The following sample code retrieves the number of pins connected to net VCC, assuming it exists in the open design, using the [Net.Pins Property](#) and [Objects.Count Property](#) properties. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set netVCC = ActiveDocument.Nets("VCC")

MsgBox "Net " & netVCC.Name & " connects " & netVCC.Pins.Count & "
pin(s)."
```

```
End Sub
```

Document.ObjectType Property

Returns the type of this object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

Document.OriginX Property

Returns the x origin of the document.

Usage

OriginX(*unit* As [PPcbUnit](#)) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the x origin is returned.

Description

If the unit passed as an argument is different from the server database unit type, the return value is 0.0. The origin of the document in mils, millimeters, or inches is 0.0 by definition.

Examples

The following sample code retrieves the origin of the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
Set doc = ActiveDocument
MsgBox "Origin = (" & doc.OriginX(ppcbUnitDatabase) & ", " &
doc.OriginY(ppcbUnitDatabase) & ")"
End Sub
```


Document.OriginY Property

Returns the y origin of the document.

Usage

OriginY(*unit* As [PPcbUnit](#)) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the y origin is returned.

Description

If the unit passed as an argument is different from the server database unit type, the return value is 0.0. The origin of the document in mils, millimeters, or inches is 0.0 by definition.

Examples

The following sample code retrieves the origin of the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
Set doc = ActiveDocument
MsgBox "Origin = (" & doc.OriginX(ppcbUnitDatabase) & ", " &
doc.OriginY(ppcbUnitDatabase) & ")"
End Sub
```

Document.Parent Property

Returns the parent of the object.

Usage

Parent As [Application](#) on page 68

Arguments

None

Description

None

Document.PartTypes Property

Returns the collection of all part types.

Usage

PartTypes As [Objects](#) on page 477

PartTypes(*name* As String) As [PartType](#) on page 519

Arguments

<i>name</i>	Name of an existing part type.

Description

When an existing part type name is passed to this property, it returns that [PartType](#) on page 519 object. Otherwise, this property returns the collection of all part types in an [Objects](#) on page 477 collection object.

Examples

The following sample code retrieves the number of part types in the open schematic using the [Objects.Count Property](#) property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set pkgs = ActiveDocument.PartTypes

MsgBox "There are " & pkgs.Count & " part type(s) in " &
ActiveDocument.Name

End Sub
```

Document.Path Property

Returns the path of the document.

Usage

Path As String

Arguments

None

Description

None

Examples

The following sample code retrieves the name and path of the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "Hi, you are using " & ActiveDocument.Name & " located in " &  
ActiveDocument.Path
```

```
End Sub
```

Document.Pins Property

Returns the collection of all pins.

Usage

Pins As [Objects](#) on page 477

Pins(*name* As String) As [Pin](#) on page 532

Arguments

<i>name</i>	Name of an existing pin.

Description

When an existing pin *name* is passed to this property, it returns that [Pin](#) on page 532 object. If the pin *name* does not exist, this property returns the collection of all pins in an [Objects](#) on page 477 collection object.

Examples

The following sample code retrieves the number of pins in the open design using the [Objects.Count Property](#). See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set pins = ActiveDocument.Pins

MsgBox "There are " & pins.Count & " pin(s) in " & ActiveDocument.Name

End Sub
```

The following sample code retrieves the drill size of pin U1.1, assuming it exists in the open design, using the [Pin.DrillSize Property](#). See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set pinU1_1= ActiveDocument.Pins("U1.1")

MsgBox "Pin " & pinU1_1.Name & "'s drill size is " & pinU1_1.DrillSize

End Sub
```

Document.Preference Property

Sets or returns a document preference.

Usage

Preference(*name* As String) As [Variant](#)

Arguments

<i>name</i>	Name of the preference

Table

Description

The following are possible *name* argument values:

<i>DRC</i>	Sets or returns the DRC mode of the open design. The DRC mode values are of type PPcbDRCMode .
<i>Nudge</i>	Sets or returns the Nudge mode of the open design. The Nudge mode values are of type PPcbNudgeMode .
<i>ModifyUnionMember</i>	Sets or returns the ability (True/False) to move an individual component when that component belongs to a union.

This property generates an [Exception](#) if the name argument is not a valid document preference.

Examples

The following sample code demonstrates how to use this property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

' Set the DRC mode to "Prevent"

ActiveDocument.Preference("DRC") = ppcbDRCPrevent

' Set the Nudge mode to "Automatic"

ActiveDocument.Preference("Nudge") = ppcbNudgeAuto

End Sub
```

Document.RouteSegments Property

Returns the collection of all trace segments.

Usage

RouteSegments As [Objects](#) on page 477

RouteSegments(*name* As String) As [RouteSegment](#) on page 582

Arguments

<i>name</i>	Name of an existing trace segment.

When an existing trace segment *name* is passed to this property, it returns that [RouteSegment](#) on page 582 or [Net](#) on page 452. If the trace segment *name* does not exist, this property returns the collection of all trace segments in an [Objects](#) on page 477 collection object.

Description

None

Examples

The following sample code retrieves the number of trace segments in the open design using the [Objects.Count Property](#) property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set rtesegs = ActiveDocument.RouteSegments

MsgBox "There are " & rtesegs.Count & " route segment(s) in " &
ActiveDocument.Name

End Sub
```

Document.Saved Property

Sets or returns whether the document is saved.

Usage

Saved As Boolean

Arguments

None

Description

This property is usually used before opening a new design, using the [Application.OpenDocument Method](#) method, to ensure that the message Save old file before reloading? does not appear.

Examples

The following sample code sets the saved status of the open design to True and then opens PWRDEMOA.PCB, assuming that it exists in the folder specified by the [Application.DefaultFilePath Property](#) property. It then retrieves the name of the opened design file. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.Saved = True
```

```
OpenDocument(DefaultFilePath & "\PWRDEMOA.PCB")
```

```
MsgBox ActiveDocument.FullName & " has just been opened."
```

```
End Sub
```


Document.Texts Property

Returns the collection of free texts in the document.

Usage

Texts as Collection

Texts (*Name* as String) as Text

Arguments

<i>name</i>	Name of the text to retrieve (optional).

Description

None

Examples

This sample shows the total number of texts in the design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "Number of Texts: " & ActiveDocument.Texts.Count
```

```
End Sub
```

Document.Unit Property

Sets or returns the system unit used in the document.

Usage

Unit As [PPcbUnit](#)

Arguments

None

Examples

The following sample code sets the unit in the open design to metric. See [Code Samples](#) for more information on running this sample.

```
Sub Main  
  
ActiveDocument.Unit = ppcbUnitMetric  
  
End Sub
```

Document.Vias Property

Returns the collection of all vias.

Usage

Vias As [Objects](#) on page 477

Vias(*name* As String) As [Via](#) on page 657

Arguments

<i>name</i>	Name of an existing via.

Description

When an existing via name is passed to this property, it returns that [Via](#) on page 657 object. If the via name does not exist, this property returns the collection of all vias in an [Objects](#) on page 477 collection object.

Examples

The following sample code retrieves the number of vias in the open design using the [Objects.Count Property](#) property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set vias = ActiveDocument.Vias

MsgBox "There are " & vias.Count & " via(s) in " & ActiveDocument.Name

End Sub
```

Document.Activate Method

Activates the window associated with a document.

Usage

Activate()

Arguments

None

Return Values

None

Description

This is a Microsoft requirement. However, because SailWind Layout is an SDI (Single Document Interface) server application, this function has no effect.

Document.AddText Method

Adds new text to the document.

Usage

AddText (*Text* as String, *Layer* as Long, *PositionX* as Double, *PositionY* as Double, *Height* as Double, *LineWidth* as Double, *Units* as [PPcbUnit](#), *Orientation* as Double, *Mirror* as Boolean) as Text

Arguments

<i>text</i>	Contents of the new text object (required).
<i>layer</i>	Number of layer on which to place the new text (optional). The default is 1.
<i>positionX</i>	x-coordinate of the new object (optional). The default is 0.
<i>positionY</i>	y-coordinate of the new object (optional). The default is 0.
<i>height</i>	Height of the text (optional). If omitted, the value specified in the Tools/Options/Drafting tab/Text frame/Height option is used.
<i>linewidth</i>	Line width of the text. If omitted, the value specified in the Tools/Options/Drafting tab/Text frame/Width option is used.
<i>units</i>	[Optional on page 732] Unit in which the unit of measurement is returned. This optional argument is ppcbUnitCurrent on page 719 by default.
<i>orientation</i>	Orientation of the text (optional). The default is 0.
<i>mirror</i>	True if the text is mirrored (optional). The default is False.

Return Values

Returns the newly added text object.

Description

None

Examples

This sample adds the text "Hello!" to the design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.AddText("Hello !", 1, 200, 200, , , pcbUnitMils)
```

```
End Sub
```

Document.CheckASCII Method

Compares the given SailWind Layout format netlist with current SailWind Layout format netlist.

Usage

CheckASCII(*name* As String, [*ignorenet* As String]) As Long

Arguments

<i>name</i> :	Name of an existing netlist file to compare against.
<i>ignorenet</i> :	[Optional on page 732] Ignore the specified net.

Return Values

If the function succeeds, the return value is nonzero. If the function fails, the return value is zero (0).

Description

The function fails if the specified file *name* does not exist or its format is incorrect.

This method generates an [Exception](#) if a SailWind Layout security failure occurs during processing.

Examples

The following sample code compares the specified netlist (padsnet.asc) with the current netlist in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.CheckASCII(DefaultFilePath & "\padsnet.asc")
```

```
End Sub
```

Document.ExportASCII Method

Generates a SailWind Layout-format ASCII file from the current design.

Usage

ExportASCII(*name* As String, [*sections* As [PPcbASCIISections](#) = [ppcbASCIISectionAll](#) on page 719], [*ver* As [PPcbASCIIVersion](#)], [*expandAttrs* As [PPcbAttrFlags](#) = [ppcbAttrNone](#)]) As Long

Arguments

<i>name</i>	Name of the file to which to export the netlist.
<i>sections</i>	[Optional on page 732] Sections of ASCII file to output (all by default).
<i>ver</i>	[Optional on page 732] Version in which to export (for backward compatibility).
<i>expandAttrs</i>	[Optional on page 732] Attribute expansion options (none by default).

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero (0).

Description

This method is an advanced version of [Document.ExportNetList Method](#), extended by the *sections* argument.

If the specified file *name* does not exist the function creates a new file. If the file *name* does exist, this method overwrites the existing file.

This property generates an [Exception](#) if the *ver* argument is not a valid SailWind Layout ASCII version number.

Examples

The following sample code creates an ASCII file with the specified name padsnet.asc, containing only part and connection sections. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.ExportASCII(DefaultFilePath & "\padsnet.asc",  
    ppcbASCIISectionParts Or ppcbASCIISectionConnections)
```

```
End Sub
```

Document.ExportECOFile Method

Generates an .eco file with the specified file name.

Usage

ExportECOFile(*name* As String) As Long

Arguments

<i>name</i>	Name of the new file to which to copy.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero (0).

Description

The function fails if the .eco file does not currently exist. To create an ECO file, the function has to exit .eco mode and clear the undo buffer .

Examples

The following sample code creates an ECO file with the specified name pcb2eco.eco. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.ExportECOFile(DefaultFilePath & "\pcb2eco.eco")
```

```
End Sub
```


Document.ExportNetList Method

Generates a SailWind Layout format netlist from the current design.

Usage

ExportNetList(*name* As String, [*ver* As [PPcbASCIIVersion](#)]) As Long

Arguments

<i>name</i>	Name of the file to which to export the netlist.
<i>ver</i>	[Optional on page 732] Version in which to export (for backward compatibility).

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero (0).

Description

If the specified file *name* does not exist the function creates new file. If the file *name* does exist, this method overwrites the existing file.

This property generates an [Exception](#) if the *ver* argument is not a valid SailWind Layout ASCII version number.

Examples

The following sample code creates a netlist file with the specified name padsnet.asc. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.ExportNetList(DefaultFilePath & "\padsnet.asc")
```

```
End Sub
```

Document.ExportRules Method

Generates an ASCII file with rules from the current design.

Usage

ExportRules(*name* As String, [*ver* As [PPcbASCIIVersion](#)]) As Long

Arguments

<i>name</i>	Name of the file to which to export rules.
<i>ver</i>	[Optional on page 732] Version in which to export (for backward compatibility).

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero (0).

Description

If the specified file name does not exist, the function creates a new file. If the file name does exist, this method overwrites the existing file.

This property generates an [Exception](#) if the *ver* argument is not a valid SailWind Layout ASCII version number.

Examples

The following sample code creates a rules file with the specified name padsnet.rul. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.ExportRules(DefaultFilePath & "\padsnet.rul")
```

```
End Sub
```

Document.GetColor Method

Returns the color of the specified document element.

Usage

GetColor(*colorType* as PPcbDocumentColor) as Integer

Arguments

<i>colorType</i>	Specifies the document element

Return Values

The index of the color in the palette.

Examples

```
doc = Application.ActiveDocument
```

```
msg=""
```

```
msg = msg & doc.GetColor(ppcbDocumentColorBackground) & ", "
```

```
msg = msg & doc.GetColor(ppcbDocumentColorSelection) & ", "
```

```
msg = msg & doc.GetColor(ppcbDocumentColorHighlight) & ", "
```

```
msg = msg & doc.GetColor(ppcbDocumentColorBoardOutline) & ", "
```

```
msg = msg & doc.GetColor(ppcbDocumentColorConnection)
```

```
MsgBox msg
```

```
curr_bkg_color = doc.GetColor(ppcbDocumentColorBackground)
```

```
doc.SetColor(ppcbDocumentColorBackground, 2)
```

```
MsgBox "Press any key"
```

```
doc.SetColor(ppcbDocumentColorBackground, curr_bkg_color)
```

Document.GetObjects Method

Returns a collection of SailWind Layout database objects.

Usage

GetObjects([*type* As [PPcbObjectType](#) = ppcbObjectTypeAll], [*value* As String], [*selected* As Boolean = False]) As [Objects](#) on page 477

Arguments

<i>type</i>	[Optional on page 732] Type of SailWind Layout database object to get.
<i>value</i>	[Optional on page 732] Value or name of the object(s) to get.
<i>selected</i>	[Optional on page 732] True to get selected objects only. False to get all objects.

All arguments to this method are optional, which means that it can be called with no argument at all, or with any combination of arguments. See the samples below for more information.

Name argument supports wildcarding ("U*"), lists of items delimited by comma ("U1, U2, R1"), ranges specified by two object names and the dash character ("U1 - U10, U12, R1 - R20"). Dash must be surrounded by spaces since the dash is a legal symbol in an object name. Only one wildcard per name is allowed and you cannot specify wildcards in a range. You can pass name such as "U*, R*, C1 - C100" but you cannot pass name such as "U*1*" or "C1* - C10*".

Return Values

The returned object is an [Objects](#) on page 477 collection object. If no objects satisfy the request, the returned collection is empty.

Description

To get all objects of the same type, use the corresponding object Document property instead of this method. For example, to get all connections in the open design, use [Document.Connections Property](#) instead of Document.GetObjects (ppcbObjectTypeConnection).

This method does not support all of the object types defined in [PPcbObjectType](#). The method will generate an [Exception](#) if the type argument is not one of the following values:

ppcbObjectTypeAssociatedNet

ppcbObjectTypeComponent

ppcbObjectTypeNet

ppcbObjectTypePin

ppcbObjectTypeVia

ppcbObjectTypeConnection

ppcbObjectTypeRouteSegment

ppcbObjectTypeJumper

ppcbObjectTypePartType

ppcbObjectTypeCBP
ppcbObjectTypeSBP
ppcbObjectTypeWirebond
ppcbObjectTypeNetClass
ppcbObjectTypeDrawing
ppcbObjectTypeText
ppcbObjectTypeLabel
ppcbObjectTypeAll

Examples

The following sample code shows different ways to use this method, displaying the number of objects retrieved for each way, using the [Objects.Count Property](#) property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Dim objs As Object

' Ex1: Get all objects of all types

Set objs = ActiveDocument.GetObjects

MsgBox "Ex1: " & objs.Count & " objects."

' Ex2: Get all selected objects of all types

Set objs = ActiveDocument.GetObjects(, , True)

MsgBox "Ex2: " & objs.Count & " selected objects."

' Ex3: Get all net objects

Set objs = ActiveDocument.GetObjects(ppcbObjectTypeNet)

MsgBox "Ex3: " & objs.Count & " net objects."

' Ex4: Get all net objects of name "VCC" (there is at least 1 of course)

Set objs = ActiveDocument.GetObjects(ppcbObjectTypeNet, "VCC")
```

```
MsgBox "Ex4: " & objs.Count & " VCC net objects."
```

```
' Ex5: Get all part objects which names begin with U
```

```
Set objs = ActiveDocument.GetObjects(ppcbObjectTypeComponent, "U*")
```

```
MsgBox "Ex3: " & objs.Count & " U* part objects."
```

```
End Sub
```

Document.GetVisibility Method

Returns the visibility of the specified design object type.

Usage

GetVisibility(*objectType* as PPcbDesignObject) as Boolean

Arguments

<i>objectType</i>	Specifies design object type.

Return Values

TRUE if visible, FALSE if hidden.

Examples

```
doc = Application.ActiveDocument

If doc.GetVisibility(ppcbDesignObjectPad) = true Then
    MsgBox "Pads are visible"
Else
    MsgBox "Pads are invisible"
End If

doc.SetVisibility(ppcbDesignObjectTrace, false)

MsgBox "Press any key"

doc.SetVisibility(ppcbDesignObjectTrace, true)
```


Document.ImportECOFile Method

Imports an .eco file.

Usage

ImportECOFile(*name* As String) As Long

Arguments

<i>name</i>	Name of an existing .eco file to import.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero (0).

Description

The function fails if the specified file *name* does not exist or its format is incorrect.

This method generates an [Exception](#) if a SailWind Layout security failure occurs during processing.

Examples

The following sample code imports the specified .eco file, pcb2eco.eco, into the current design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.ImportECOFile(DefaultFilePath & "\pcb2eco.eco")
```

```
End Sub
```

Document.ImportNetList Method

Imports a SailWind Layout format netlist file.

Usage

ImportNetList(*name* As String) As Long

Arguments

<i>name</i>	Name of an existing netlist file to import.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero (0).

Description

The function fails if the specified file *name* does not exist or its format is incorrect.

This method generates an [Exception](#) if a SailWind Layout security failure occurs during processing.

Examples

The following sample code imports the specified netlist file, padsnet.asc, into the current design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.ImportNetList(DefaultFilePath & "\padsnet.asc")
```

```
End Sub
```

Document.IntegrityTest Method

Runs the integrity test.

Usage

IntegrityTest() as Boolean

Arguments

None

Return Values

TRUE if the test passed without errors, FALSE in other cases.

Document.Save Method

Saves the document.

Usage

Save

Arguments

None

Return Values

None

Description

This method generates an [Exception](#) if a SailWind Layout security failure occurs during processing.

Examples

The following sample code saves, if needed, the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
  If ActiveDocument.Saved = False Then ActiveDocument.Save
```

```
End Sub
```

Document.SaveAs Method

Saves the document with a new name.

Usage

Save(*name* As String)

Arguments

name	New name for the file.

Return Values

None

Description

This method generates an [Exception](#) if a SailWind Layout security failure occurs during processing.

Examples

The following sample code creates a custom backup of the open design. If the PCB design file name is XXX.PCB, the backup file is saved in the same folder with the name XXX (Backup on 12-28-2001 4h40).PCB. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
curDesignName = ActiveDocument.FullName
```

```
theDate = Month(Date) & "-" & Day(Date) & "-" & Year(Date) & " at " &  
Hour(Now) & "h" & Minute(Now)
```

```
bakDesignName = Left$(curDesignName, Len(curDesignName)-4) & " (Backup on "  
& theDate & ").pcb"
```

```
ActiveDocument.SaveAs(bakDesignName)
```

```
OpenDocument(curDesignName)
```

```
End Sub
```

Document.SaveAsNoLock Method

Saves the document with a new name and releases the file lock.

Usage

SaveAsNoLock(*filename* as String)

Arguments

<i>filename</i>	New name for the file

Return Values

This method generates an exception if a SailWind Layout security failure occurs during processing.

Document.SaveNoLock Method

Saves the document and releases the file lock.

Usage

SaveNoLock

Arguments

None

Return Values

This method generates an exception if a SailWind Layout security failure occurs during processing.

Document.SaveAsTemp Method

Works in the same way as SaveAsNoLock but additionally it doesn't add the filename to the MRU (Most Recently Used) list. The Document.SaveAsTemp method is intended mainly for macro tests.

Usage

SaveAsTemp(*filename* as String)

Arguments

<i>filename</i>	New name for the file

Return Values

This method generates an exception if a SailWind Layout security failure occurs during processing.

Document.SaveTemp Method

Works in the same way as SaveNoLock but additionally it doesn't add the filename to the MRU (Most Recently Used) list. This method is mainly intended for macro tests.

Usage

SaveTemp

Arguments

None

Return Values

This method generates an exception if a SailWind Layout security failure occurs during processing.

Document.SelectObjects Method

Selects or deselects SailWind Layout database objects.

Usage

SelectObjects([*type* As [PPcbObjectType](#) = ppcbObjectTypeAll], [*value* As String], [*select* As Boolean = True])

Arguments

<i>type</i>	[Optional on page 732] Type of database object to select/deselect.
<i>value</i>	[Optional on page 732] Value or name of the objects to select/deselect.
<i>select</i>	[Optional on page 732] True to select. False to deselect.

All arguments for this method are optional, which means that it can be called with no argument at all, or with any combination of arguments. See the Examples below for more information.

Name argument supports wildcarding ("U*"), lists of items delimited by comma ("U1, U2, R1"), ranges specified by two object names and the dash character ("U1 - U10, U12, R1 - R20"). Dash must be surrounded by spaces since the dash is a legal symbol in an object name. Only one wildcard per name is allowed and you cannot specify wildcards in a range. You can pass *name* such as "U*, R*, C1 - C100" but you cannot pass *name* such as "U*1*" or "C1* - C10*".

Return Values

None

Description

This property generates an [Exception](#) if the type argument is not a valid SailWind Layout database object type.

Examples

The following sample code shows different ways to use this method, displaying the number of objects selected for each way, using the [Document.GetObjects Method](#) method and the [Objects.Count Property](#) property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Dim objs As Object

' Ex1: Select all objects of all types

ActiveDocument.SelectObjects
```

```
Set objs = ActiveDocument.GetObjects(,,True)

MsgBox "Ex1: " & objs.Count & " selected objects (all)."
```

' Ex2: Unselect all objects of all types

```
ActiveDocument.SelectObjects(,,False)

Set objs = ActiveDocument.GetObjects(,,True)

MsgBox "Ex2: " & objs.Count & " selected objects (none)."
```

' Ex3: Select all net objects

```
ActiveDocument.SelectObjects(ppcbObjectTypeNet)

Set objs = ActiveDocument.GetObjects(,,True)

MsgBox "Ex3: " & objs.Count & " selected objects (all nets)."
```

' Ex4: Unselect net VCC

```
ActiveDocument.SelectObjects(ppcbObjectTypeNet, "VCC", False)

Set objs = ActiveDocument.GetObjects(,,True)

MsgBox "Ex4: " & objs.Count & " selected objects (all nets except VCC)."
```

' Ex5: Select only part objects which names begin with U

```
ActiveDocument.SelectObjects(,,False)

ActiveDocument.SelectObjects(ppcbObjectTypeComponent, "U*")

Set objs = ActiveDocument.GetObjects(,,True)

MsgBox "Ex5: " & objs.Count & " selected U* part objects."
```

End Sub

Document.SetColor Method

Sets the color for the specified document element.

Usage

SetColor(*colorType* as PPcbDocumentColor, *colorIndex* as Integer)

Arguments

<i>colorType</i>	Specifies document element
<i>colorIndex</i>	Index of the color in the palette. Must be between 0 and 31

Examples

```
doc = Application.ActiveDocument
```

```
msg=""
```

```
msg = msg & doc.GetColor(ppcbDocumentColorBackground) & ", "
```

```
msg = msg & doc.GetColor(ppcbDocumentColorSelection) & ", "
```

```
msg = msg & doc.GetColor(ppcbDocumentColorHighlight) & ", "
```

```
msg = msg & doc.GetColor(ppcbDocumentColorBoardOutline) & ", "
```

```
msg = msg & doc.GetColor(ppcbDocumentColorConnection)
```

```
MsgBox msg
```

```
curr_bkg_color = doc.GetColor(ppcbDocumentColorBackground)
```

```
doc.SetColor(ppcbDocumentColorBackground, 2)
```

```
MsgBox "Press any key"
```

```
doc.SetColor(ppcbDocumentColorBackground, curr_bkg_color)
```

Document.SetVisibility Method

Sets visibility for the specified design object type.

Usage

`SetVisibility(objectType as PPcbDesignObject, objectVisibility as Boolean)`

Arguments

<i>objectType</i>	Specifies design object type.
<i>objectVisibility</i>	TRUE if visible, FALSE if hidden.

Examples

```
doc = Application.ActiveDocument

If doc.GetVisibility(ppcbDesignObjectPad) = true Then
    MsgBox "Pads are visible"
Else
    MsgBox "Pads are invisible"
End If

doc.SetVisibility(ppcbDesignObjectTrace, false)

MsgBox "Press any key"

doc.SetVisibility(ppcbDesignObjectTrace, true)
```

Document.PositionsChange Event

Occurs after the position of a component is changed.

Usage

Document_PositionsChange()

Arguments

None

Description

None

Document.Save Event

Occurs after the document is saved.

Usage

Document_Save()

Arguments

None

Return Values

None

Description

None

Document.SecurityLimit Event

Occurs when the database security limits are reached.

Usage

Document_SecurityLimit (Reached as Boolean)

Arguments

None

Return Values

None

Description

None

Document.SelectionChange Event

Occurs when the current selection changes.

Usage

Document_SelectionChange()

Arguments

None

Description

None

The Drawing Object

The Drawing object represents a physical drawing object or a copper in the open design.

The Drawing object contains the following properties:

- Drawing.Application Property
- Drawing.DrawingType Property
- Drawing.Geometry Property
- Drawing.Name Property
- Drawing.Net Property
- Drawing.ObjectType Property
- Drawing.Parent Property
- Drawing.PositionX Property
- Drawing.PositionY Property
- Drawing.Selected Property
- Drawing.Texts Property

Drawing.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This is a Microsoft-required property.

Examples

Not required.

Drawing.DrawingType Property

Returns the type of the open drawing.

Usage

DrawingType as [PPcbDrawingType](#)

Arguments

None

Description

None

Examples

The following sample function displays the name and type of the drawing. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,, TRUE)

For Each drw In selected

Select Case drw.DrawingType

Case ppcbDrw2Dline

Msgbox "Type: 2D Line"

Case ppcbDrwBoard

Msgbox "Type: Board Outline"

Case ppcbDrwCopper

Msgbox "Type: Copper"

Case ppcbDrwCopperPour

Msgbox "Type: CopperPour"
```

Case ppcbDrwCopperHatch

Msgbox "Type: CopperHatch"

Case ppcbDrwCopperThermal

Msgbox "Type: CopperThermal"

Case ppcbDrwKeepout

Msgbox "Type: Keepout"

End Select

Next drw

End Sub

Drawing.Geometry Property

Returns a collection of objects, currently polylines, text, or circles, representing the geometry of the drawing.

Usage

Geometry as Collection

Arguments

None

Description

None

Examples

The following sample shows the number of polylines in the selected drawing. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,TRUE)

For Each drw In selected

I = 0

Dim geom as Object

For Each geom in drw.Geometry

If geom.ObjectType = ppcbObjectTypePolyline Then I = I + 1

Next geom

MsgBox "Drawing contains " & I & " polylines"

Exit For

Next drw

End Sub
```

Drawing.Name Property

Returns the name of the drawing.

Usage

Name as String

Arguments

None

Description

None

Examples

The following example displays a message box showing the name of the first drawing in the active document. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each drw In ActiveDocument.Drawings

MsgBox "The first drawing's name is " & drw.Name

Exit For

Next drw

End Sub
```


Drawing.Net Property

Returns a net with which the drawing is associated.

Usage

Net as Net

Arguments

None

Description

None

Examples

The following sample function displays the name of the parent object in the selected drawing. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,TRUE)

For Each drw In selected

If drw.Net Is Nothing Then

Msgbox "No Net"

Else

Msgbox "Net: " & drw.Net.Name

End If

Next drw

End Sub
```

Drawing.ObjectType Property

Returns the type of object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

Examples

The following example shows the number of selected drawings. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set sel = ActiveDocument.GetObjects(, , TRUE)

n = 0

For Each obj In sel

If obj.ObjectType = ppcbObjectTypeDrawing Then n = n + 1

Next obj

MsgBox n & " drawing(s) selected"

End Sub
```

Drawing.Parent Property

Returns the parent of the object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

This is a Microsoft-required property.

Drawing.PositionX Property

This property returns the X coordinate of the origin of the drawing.

Usage

PositionX(*Unit* as [PPcbUnit](#), *Origin* as [PPcbOriginType](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the result is to be expressed. This optional argument is ppcbUnitCurrent on page 719 by default.
<i>origin</i>	[Optional on page 732]Type of reference point from which the result is counted. The default value is ppcbOriginTypeDesign on page 719.

Description

None

Examples

The following sample function displays selected drawing positions. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing, ,TRUE)

For Each drw In selected

Msgbox "Position: (" & drw.PositionX & ", " & drw.PositionY & ") "

Next drw

End Sub
```

Drawing.PositionY Property

Returns the Y coordinate of the origin of the drawing.

Usage

PositionY(*Unit* as [PPcbUnit](#), *Origin* as [PPcbOriginType](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the result is to be expressed. This optional argument is ppcbUnitCurrent on page 719 by default.
<i>origin</i>	[Optional on page 732]Type of reference point from which the result is counted. The default value is ppcbOriginTypeDesign on page 719.

Description

None

Examples

The following sample function displays the positions of the selected drawing objects. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing, ,TRUE)

For Each drw In selected

Msgbox "Position: (" & drw.PositionX & ", " & drw.Position.Y & ") "

Next drw

End Sub
```

Drawing.Selected Property

Sets or returns whether the component is selected.

Usage

Selected as Boolean

Arguments

None

Description

None

Examples

The following sample function displays a message showing whether or not the first drawing is selected. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each drw In ActiveDocument.Drawings

Msgbox "Is Drawing " & drw.Name & " selected? " & drw.Selected

Exit For

Next drw

End Sub
```

Drawing.Texts Property

Returns a collection of text objects associated with the drawing, or a specific text object, referenced by its name.

Usage

Texts as Collection

Texts (*Name* as String) as Text

Arguments

<i>name</i>	Text object name.

Description

None

Examples

The following sample function returns the number of texts associated with the selected drawing. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,TRUE)

For Each drw In selected

MsgBox "Number of associated texts: " & drw.Texts.Count

Next drw

End Sub
```

The Error Object

The Error object represents a design error.

The Error object contains the following properties:

- Error.ActualValue Property
- Error.Application Property
- Error.Conflicts Property
- Error.Description Property
- Error.ErrorClass Property
- Error.ErrorType Property
- Error.HasActualValue Property
- Error.HasRequiredValue Property
- Error.IsClearanceError Property
- Error.IsConnectivityError Property
- Error.IsHighSpeedError Property
- Error.IsIgnoredFlag Property
- Error.IsInvisibleFlag Property
- Error.IsLatiumError Property
- Error.IsMiscError Property
- Error.IsTestPointError Property
- Error.LayerNumber Property
- Error.Name Property
- Error.ObjectType Property
- Error.Parent Property
- Error.PositionX Property
- Error.PositionY Property
- Error.RequiredValueMax Property
- Error.RequiredValueMin Property

Error.ActualValue Property

Returns the actual value of the Error Object.

Usage

Conflicts as Objects

Conflicts(conflictNumber as Integer) as ErrorConflict

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the actual value is returned.

Examples

The following sample code displays information about all of the errors' actual value in SailWind Layout. See [Code Samples](#) for more information on running this sample.

```
' Display information about all errors' actual value in Layout
```

```
doc = Application.ActiveDocument
```

```
MsgBox "Total number of errors: " & doc.Errors.Count
```

```
For Each theErr In doc.Errors
```

```
    ' Error actual value information text
```

```
    info="Actual value N/A"
```

```
    If theErr.HasActualValue Then
```

```
        info=" Actual value: " & theErr.ActualValue
```

```
        info = info & " [default unit]; "
```

```
        If (theErr.ValueType = ppcbErrorValueTypeMeasure) Then
```

```
            info = info & theErr.ActualValue(ppcbUnitCurrent)
```

```
            info = info & " [current unit]; "
```

```
    info = info & theErr.ActualValue(ppcbUnitDatabase)

    info = info & " [database unit]; "

    info = info & theErr.ActualValue(ppcbUnitMils)

    info = info & " [mils]; "

    info = info & theErr.ActualValue(ppcbUnitInch)

    info = info & " [inch]; "

    info = info & theErr.ActualValue(ppcbUnitMetric)

    info = info & " [millimeter]"

End If

End If

MsgBox info

' Next error

Next
```

Error.Application Property

Returns the Application object.

Usage

Application as [Application](#) on page 68

Arguments

None

Error.Conflicts Property

Returns the collection of all error conflicts in the Error Object.

Usage

Conflicts as Objects

Conflicts(conflictNumber as Integer) as ErrorConflict

Arguments

<i>conflictNumber</i>	Conflict number

Description

When the conflict number is passed to this property, it returns that ErrorConflict Object. If the number is not specified, this property returns the collection of all error conflicts in an Objects collection object.

Error.Description Property

Returns the description of the Error Object.

Usage

Description as String

Arguments

None

Error.ErrorClass Property

Returns the error class of the Error Object.

Usage

```
ErrorClass as PPcbErrorClass
```

Arguments

None

Error.ErrorType Property

Returns the error type of the Error Object.

Usage

ErrorType as PPcbErrorType

Arguments

None

Error.HasActualValue Property

Returns if the actual value of the Error Object exists.

Usage

HasActualValue as Boolean

Arguments

None

Error.HasRequiredValue Property

Returns if the required value of the Error Object exists.

Usage

HasRequiredValue as Boolean

Arguments

None

Error.IsClearanceError Property

This property returns if the error is a clearance error.

Usage

IsClearanceError as Boolean

Arguments

None

Error.IsConnectivityError Property

Returns if the error is a connectivity error.

Usage

IsConnectivityError as Boolean

Arguments

None

Error.IsHighSpeedError Property

Returns if the error is a high speed error.

Usage

IsHighSpeedError as Boolean

Arguments

None

Error.IsIgnoredFlag Property

Returns if the error is flagged as ignored.

Usage

IsIgnoredFlag as Boolean

Arguments

None

Error.IsInvisibleFlag Property

Returns if the error is flagged as invisible.

Usage

IsInvisibleFlag as Boolean

Arguments

None

Error.IsLatiumError Property

Returns if the error is a latium error.

Usage

IsLatiumError as Boolean

Arguments

None

Error.IsMiscError Property

Returns if the error is a miscellaneous error.

Usage

IsMiscError as Boolean

Arguments

None

Error.IsTestPointError Property

Returns if the error is a Test Point error.

Usage

IsTestPointError as Boolean

Arguments

None

Error.LayerNumber Property

Returns the layer number of this Error Object.

Usage

LayerNumber as int

Arguments

None

Error.Name Property

Returns the name of this error.

Usage

Name As String

Arguments

None

Error.ObjectType Property

Returns the type of this object - ppcbObjectTypeError.

Usage

ObjectType as PPcbObjectType

Arguments

None

Error.Parent Property

Returns the parent of the object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Error.PositionX Property

Returns the X-coordinate of the Error Object.

Usage

PositionX as Double

PositionX(unit as PPcbUnit) as Double

Arguments

<i>unit</i>	[Optional Argument] Unit in which the X-coordinate is returned.

Error.PositionY Property

Returns the Y-coordinate of the Error Object.

Usage

PositionY as Double

PositionY(unit as PPcbUnit) as Double

Arguments

<i>unit</i>	[Optional Argument] Unit in which the Y-coordinate is returned.

Error.RequiredValueMax Property

Returns the required maximum value of the Error Object.

If this value equals the maximum real value (see [Document.MaxRealValue Property](#)) it means infinity.

Usage

RequiredValueMax as Double

RequiredValueMax (unit as PPcbUnit) as Double

Arguments

<i>unit</i>	[Optional Argument] Unit in which the required maximum value is returned.

Error.RequiredValueMin Property

Returns the required minimum value of the Error Object.

If this value equals the minimum real value (see [Document.MinRealValue Property](#)) it means minus infinity.

Usage

RequiredValueMin as Double

RequiredValueMin (unit as PPcbUnit) as Double

Arguments

<i>unit</i>	[Optional Argument] Unit in which the required minimum value is returned.

The ErrorConflict Object

The Error object represents an error conflict in the Error Object.

The ErrorConflict object contains the following properties:

- ErrorConflict.Application Property
- ErrorConflict.ConflictObject Property
- ErrorConflict.ConflictObjectDesc Property
- ErrorConflict.ConflictObjectType Property
- ErrorConflict.Name Property
- ErrorConflict.ObjectType Property
- ErrorConflict.Parent Property

ErrorConflict.Application Property

Returns the Application object.

Usage

Application as [Application](#) on page 68

Arguments

None

ErrorConflict.ConflictObject Property

Returns the conflicting object.

Usage

ConflictObject as Object

Arguments

None

ErrorConflict.ConflictObjectDesc Property

Returns the description of the conflicting object.

Usage

ConflictObjectDesc as String

Arguments

None

ErrorConflict.ConflictObjectType Property

Returns the type of the conflicting object.

Usage

ConflictObjectType as PPcbObjectType

Arguments

None

ErrorConflict.Name Property

Returns the name of this error conflict.

Usage

Name as String

Arguments

None

ErrorConflict.ObjectType Property

Returns the type of the object - ppcbObjectTypeErrorConflict.

Usage

ObjectType as PPcbObjectType

Arguments

None

ErrorConflict.Parent Property

Returns the parent of the object.

Usage

Parent as [Document](#) on page 250

Arguments

None

The Jumper Object

The Jumper object represents a physical jumper that exists in the open design.

The Jumper object contains the following properties:

[Jumper.Application Property](#)

[Jumper.Installed Property](#)

[Jumper.Length Property](#)

[Jumper.Name Property](#)

[Jumper.Net Property](#)

[Jumper.ObjectType Property](#)

[Jumper.Orientation Property](#)

[Jumper.Parent Property](#)

[Jumper.Points Property](#)

[Jumper.Selected Property](#)

Jumper.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This property identifies the object as an automation object. All automation server applications have an application object and all automation objects have an application property. This property is usually used in automation client applications that handle large volumes of objects from different sources, such as different automation server applications, to quickly identify the application to which the object belongs.

Jumper.Installed Property

Sets or returns whether the component is installed in the current assembly option.

Usage

Installed As Boolean

Arguments

None

Description

This property is useful only when the parent of the component is an Assembly Option. If the parent of the component is not an Assembly Option, this property always returns True and cannot be set to a different value.

Examples

The following sample code creates a new Assembly Option named EmptyAssOpt in the open design, and then uninstalls all components in that newly created Assembly Option. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set newAssOpt = ActiveDocument.AssemblyOptions.Add("EmptyAssOpt")

For Each nextComp In newAssOpt.Components

nextComp.Installed = False

Next nextComp

End Sub
```

Jumper.Length Property

Returns the length of the jumper.

Usage

Length([unit As [PPcbUnit](#)]) As Double

Arguments

unit	[Optional] on page 732] Unit in which the length value is returned.

Description

None

Examples

The following sample code retrieves the length of jumper JMP1, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set jmp1 = ActiveDocument.Jumpers("JMP1")
```

```
MsgBox "Jumper " & jmp1.name & " has a length of " & jmp1.Length
```

```
End Sub
```

Jumper.Name Property

Returns the name of the jumper.

Usage

Name As String

Arguments

None

Description

This property is the default property for the Jumper object.

Examples

The following sample code lists all jumpers in the open design and then places that list in a custom dialog box. When a jumper is selected in the list box, the sample selects that jumper.

This sample uses the User Dialog Editor in the Sax Basic Engine in SailWind Layout. See [Code Samples](#) for more information on running this sample.

For more information, see *Sax Basic Editor On Line Help* (C:\<install_folder>\<version>\Programs\sbe5_000.hlp)

```
Dim ListJmps$(10000)

Sub Main

    index = 0

    For Each nextJmp In ActiveDocument.Jumpers

        ListJmps$(index) = nextJmp.Name

        index = index + 1

    Next nextJmp

    ' This piece of code is automatically generated by the Basic Dialog Editor
    ' in SailWind Layout.

    Begin Dialog UserDialog 180,238,"Jumpers",.CallbackFunc ' %GRID:10,7,1,1

        ListBox 10,7,160,203,ListJmps(),.ListBox1
```

```
OKButton 10,210,160,21
```

```
End Dialog
```

```
Dim dlg As UserDialog
```

```
Dialog dlg
```

```
End Sub
```

```
' The following function is automatically called by the system when  
something has happened
```

```
' in the dialog; it is used to easily process user actions.
```

```
Function CallbackFunc%(DlgItem$, Action%, SuppValue%)
```

```
Select Case Action%
```

```
Case 2 ' Value changing or button pressed
```

```
If DlgItem$="ListBox1" Then
```

```
ActiveDocument.SelectObjects(ppcbObjectTypeAll, , False)
```

```
ActiveDocument.SelectObjects(ppcbObjectTypeJumper, ListJmps(SuppValue%))
```

```
End If
```

```
End Select
```

```
End Function
```

Jumper.Net Property

Returns the net connected to the jumper.

Usage

Net As [Net](#) on page 452

Arguments

None

Description

None

Examples

The following sample code identifies the net connected to jumper JMP1, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set jmp1 = ActiveDocument.Jumpers("JMP1")

MsgBox "Jumper " & jmp1.name & " is connected to net " & jmp1.Net.Name

End Sub
```

Jumper.ObjectType Property

Returns the type of the object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

This property returns [ppcbObjectTypeJumper](#) on page 719.

All SailWind Layout database objects in the SailWind Layout Automation server implement this property to compensate for the lack of a Visual Basic equivalent for the Visual C++ QueryInterface function.

This property is generally used:

- To identify the kind of SailWind Layout database objects in a heterogeneous [Objects](#) on page 477 collection.
- When implementing a generic routine that depends on the type of the SailWind Layout database object passed as argument. For example:

```
Sub DoSomething(dbObject As Object)

Select Case dbObject.ObjectType

Case ppcbObjectTypeComponent

' Do something specific to component objects

Case ppcbObjectTypeNet

' Do something specific to net objects

Case ppcbObjectTypePin

' Do something specific to pin objects

Case ppcbObjectTypeVia

' Do something specific to via objects
```

```
Case ppcbObjectTypeConnection
' Do something specific to connection objects

Case ppcbObjectTypeRouteSegment
' Do something specific to route segment objects

Case ppcbObjectTypeJumper
' Do something specific to jumper objects

Case Else
MsgBox "Not a SailWind Layout database object"

End Select

End Sub
```


Jumper.Orientation Property

Returns the orientation of the jumper.

Usage

Orientation As Double

Arguments

None

Description

None

Examples

The following sample code retrieves the orientation of jumper JMP1, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set jmp1 = ActiveDocument.Jumpers("JMP1")
```

```
MsgBox "Jumper " & jmp1.name & " has an orientation of " & jmp1.Orientation  
& " degrees."
```

```
End Sub
```

Jumper.Parent Property

Returns the parent of the object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

None

Jumper.Points Property

Returns the array of points defining the jumper.

Usage

Points([*unit* As [PPcbUnit](#)]) As [Variant](#)

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the point coordinate values are returned.

Description

None

Jumper.Selected Property

Sets or returns whether the jumper is selected.

Usage

Selected As Boolean

Arguments

None

Description

You can also select a SailWind Layout database object using the [Document.SelectObjects Method](#) and [Objects.Select Method](#) methods.

Examples

The following sample code selects jumper JMP1, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

ActiveDocument.SelectObjects(, , False)

ActiveDocument.Jumpers("JMP1").Selected = True

End Sub
```

The Label Object

The Label object represents a physical label object in the open design.

The Label object contains the following properties and method:

- Label.Application Property
- Label.Attribute Property
- Label.Component Property
- Label.Display Property
- Label.Name Property
- Label.ObjectType Property
- Label.Parent Property
- Label.RightReading Property
- Label.Selected Property
- Label.Text Property
- Label.Type Property
- Label.Delete Method

Label.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This is a Microsoft-required property.

Examples

Not required.

Label.Attribute Property

Returns an attribute with which the label is associated.

Usage

Attribute as [Attributes](#) on page 130

Arguments

None

Description

None

Examples

The following sample routine displays the name of an attribute with which the label is associated, if any. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,,TRUE)

For Each lab In selected

If Not lab.Attribute Is Nothing Then

MsgBox "The label is associated with attribute " & lab.Attribute.Name

Else

MsgBox "The label is not associated with any attribute"

End If

Next lab

End Sub
```

Label.Component Property

Returns a component with which the label is associated.

Usage

Component as [Component](#) on page 178

Arguments

None

Description

None

Examples

The following sample routine displays the name of a component with which the selected label is associated, if any. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,,TRUE)

For Each lab In selected

If Not lab.Component Is Nothing Then

MsgBox "The label is associated with component " & lab.Component.Name

Else

MsgBox "The label is not associated with any component"

End If

Next lab

End Sub
```


Label.Display Property

Returns or sets the label display mode.

Usage

Display as [PPcbLabelDisplayMode](#)

Arguments

None

Description

None

Examples

The following sample shows the display type of the selected label. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,,TRUE)

For Each label In selected

Select Case label.Display

Case ppcbLabelDisplayNone

MsgBox "Label shows nothing"

Case ppcbLabelDisplayValue

MsgBox "Label shows attribute value"

Case ppcbLabelDisplayNameAndValue

MsgBox "Label shows attribute name and value"

Case ppcbLabelDisplayFullNameAndValue

MsgBox "Label shows attribute full name and value"
```

End Select

Next label

End Sub

Label.Name Property

Returns the name of the Label object.

Usage

Name as String

Arguments

None

Description

None

Examples

The following example displays a message box showing the name of the selected label in the active document. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,,TRUE)

For Each label In selected

MsgBox "Selected label name: " & label.Name

Next label

End Sub
```

Label.ObjectType Property

Returns the type of object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

Examples

The following example shows the number of selected labels. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(, , TRUE)

n = 0

For Each obj In selected

If obj.ObjectType = ppcbObjectTypeLabel Then n = n + 1

Next obj

MsgBox n & " label(s) selected"

End Sub
```

Label.Parent Property

Returns the parent of the object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

This is a Microsoft-required property.

Label.RightReading Property

Returns or sets the right-to-left reading status of the label.

Usage

RightReading as [PPcbRightReadingStatus](#)

Arguments

None

Description

None

Examples

This sample switches the selected label to Orthogonal Right-Reading mode. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,,TRUE)

For Each lab In selected

Select Case lab.RightReading

Case ppcbRightReadingNone

s="None"

Case ppcbRightReadingOrthogonal

s="Orthogonal"

Case ppcbRightReadingAngled

s="Angled"

End Select

MsgBox "Selected label Right Reading mode: " & s
```

```
lab.RightReading = ppcbRightReadingOrthogonal
```

```
Next lab
```

```
End Sub
```

Label.Selected Property

Sets or returns whether the Label object is selected.

Usage

Selected as Boolean

Arguments

None

Description

None

Examples

The following sample displays messages indicating whether the label is selected for all labels of the first component, then selects the labels. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

For Each label In comp.Labels

Msgbox "Is Label " & label.Text & " selected ?" & label.Selected

label.Selected = True

Next label

Exit For

Next comp

End Sub
```


Label.Text Property

Returns the displayed string for the Label object.

Usage

Text as String

Arguments

None

Description

None

Examples

This sample shows the text of the selected labels. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,, TRUE)
```

```
For Each label In selected
```

```
MsgBox "Selected label: " & label.Text
```

```
Next label
```

```
End Sub
```

Label.Type Property

Returns the type of label.

Usage

LabelType as [PPcbLabelType](#)

Arguments

None

Description

None

Examples

The following sample displays the type of the selected label. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,,TRUE)

For Each label In selected

Select Case label.LabelType

Case ppcbLabelTypeRefDesignator

MsgBox "Label type: Ref Designator"

Case ppcbLabelTypePartType

MsgBox "Label type: Part Type"

Case ppcbLabelTypeAttribute

MsgBox "Label type: Attribute"

End Select

Next label
```

End Sub

Label.Delete Method

Deletes the Label object.

Usage

Delete as Boolean

Arguments

None

Return Values

Returns True if the object was successfully deleted. Returns False if the object cannot be deleted.

Description

None

Examples

The following sample deletes selected labels. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,,TRUE)

For Each label In selected

If label.Delete() = False Then

MsgBox "Cannot delete the label!"

Else

MsgBox "The label was deleted successfully!"

End If

Next label

End Sub
```

The Layer Object

The Layer object represents a design layer.

This object contains the following properties and methods:

- Layer.Application Property
- Layer.CopperThickness Property
- Layer.Enabled Property
- Layer.Name Property
- Layer.Number Property
- Layer.ObjectType Property
- Layer.Parent Property
- Layer.PlaneType Property
- Layer.RoutingDirection Property
- Layer.Type Property
- Layer.Visible Property
- Layer.GetColor Method
- Layer.GetDielectricConstant Method
- Layer.GetDielectricThickness Method
- Layer.GetDielectricType Method
- Layer.SetColor Method
- Layer.SetDielectricConstant Method
- Layer.SetDielectricThickness Method
- Layer.SetDielectricType Method

Layer.Application Property

Returns the application object.

Usage

Application as Application

Arguments

None

Layer.CopperThickness Property

Sets or returns the layer copper thickness.

Usage

CopperThickness (*unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i>	[Optional] Unit in which the value is returned. This optional argument is ppcbUnitCurrent by default.

Layer.Enabled Property

Sets or returns the layer enabled property.

Usage

Enabled as Boolean

Arguments

None

Layer.Name Property

Sets or returns the name of this layer.

Usage

Name as String

Arguments

None

Layer.Number Property

Returns the layer number.

Usage

Number as Integer

Arguments

None

Layer.ObjectType Property

Returns the type of the object - ppcbObjectTypeLayer.

Usage

ObjectType as [PPcbObjectType](#)

Arguments

None

Layer.Parent Property

Returns the parent of the object.

Usage

Parent as Document

Arguments

None

Layer.PlaneType Property

Sets or returns the layer plane type.

Usage

PlaneType as [PPcbPlaneType](#)

Arguments

None

Layer.RoutingDirection Property

Sets or returns the layer routing direction.

Usage

RoutingDirection as [PPcbRoutingdirection](#)

Arguments

None

Layer.Type Property

Sets or returns the layer type.

Usage

Type as [PPcbLayerType](#)

Arguments

None

Layer.Visible Property

Sets or returns the layer visibility.

Usage

Visible as Boolean

Arguments

None

Layer.GetColor Method

Returns color of the specified layer element.

Usage

GetColor(*colorType* as PPcbLayerColor) as Integer

Arguments

<i>colorType</i>	Specifies layer element.

Return Values

Index of the color in the palette.

Layer.GetDielectricConstant Method

Returns the layer dielectric constant.

Usage

GetDielectricConstant(*dielectricLayer* as [PPcbDielectricLayer](#)) as Double

Arguments

<i>dielectricLayer</i>	Specifies dielectric layer relevant to this electrical layer.

Return Values

The dielectric constant.

Layer.GetDielectricThickness Method

Gets the layer dielectric thickness.

Usage

```
GetDielectricThickness (  
  
    dielectricLayer as PPcbDielectricLayer,  
  
    unit as PPcbUnit) as Double
```

Arguments

<i>dielectricLayer</i>	Specifies dielectric layer relevant to this electrical layer.
<i>unit</i>	[Optional] Unit in which value is returned. This optional argument is ppcbUnitCurrent by default.

Return Values

The dielectric thickness.

Layer.GetDielectricType Method

Returns the layer dielectric type.

Usage

GetDielectricType(*dielectricLayer* as [PPcbDielectricLayer](#)) as [PPcbDielectricType](#)

Arguments

<i>dielectricLayer</i>	Specifies dielectric layer relevant to this electrical layer.

Return Values

The dielectric type.

Layer.SetColor Method

Sets color for the specified layer element.

Usage

SetColor(*colorType* as PPcbLayerColor, *colorIndex* as Integer)

Arguments

<i>colorType</i>	Specifies layer element.
<i>colorIndex</i>	Index of the color in the palette. Must be between 0 and 31.

Layer.SetDielectricConstant Method

Sets the layer dielectric constant.

Usage

```
SetDielectricConstant(  
  
    dielectricLayer as PPcbDielectricLayer,  
  
    dielectricConstant as Double)
```

Arguments

<i>dielectricLayer</i>	Specifies dielectric layer relevant to this electrical layer.
<i>dielectricConstant</i>	Specifies dielectric constant.

Layer.SetDielectricThickness Method

Sets the layer dielectric thickness.

Usage

```
SetDielectricThickness (
    dielectricLayer as PPcbDielectricLayer,
    dielectricThickness as Double,
    unit as PPcbUnit)
```

Arguments

<i>dielectricLayer</i>	Specifies dielectric layer relevant to this electrical layer.
<i>dielectricThickness</i>	Specifies dielectric thickness.
<i>unit</i>	[Optional] Unit in which value is passed to this method. This optional argument is ppcbUnitCurrent by default.

Layer.SetDielectricType Method

Sets the layer dielectric type.

Usage

```
SetDielectricType (  
  
    dielectricLayer as PPcbDielectricLayer,  
  
    dielectricType as PPcbDielectricType)
```

Arguments

<i>dielectricLayer</i>	Specifies dielectric layer relevant to this electrical layer.
<i>dielectricType</i>	Specifies dielectric type.

The Library Object

The Library object represents a library included in the SailWind Layout library list. The library corresponds to the set of four library files, with extensions .pt07, .pd07, .ln07, and .ld07.

The Library object contains the following properties:

- Library.Application Property
- Library.FullName Property
- Library.Name Property
- Library.ObjectType Property
- Library.Parent Property
- Library.Path Property
- Library.GetLibraryItems Method
- Library.ImportLibraryItems Method
- Library.ImportLibraryItems2 Method

Library.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This is a Microsoft-required property.

Examples

Not required.

Library.FullName Property

Returns the full name of library files, including path and name.

Usage

FullName As String

Arguments

None

Description

None

Examples

The following example displays a message box showing the name of the first available library. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each lib In Libraries

MsgBox "The full name of the first available library is " & lib.FullName

Exit For

Next lib

End Sub
```

Library.Name Property

Returns the name of the library.

Usage

Name As String

Arguments

None

Description

This is a Microsoft-required property.

Examples

The following example displays a message box showing the name of the first available library. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each lib In Libraries

MsgBox "The name of the first available library is " & lib.Name

Exit For

Next lib

End Sub
```

Library.ObjectType Property

Returns the type of object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

Examples

The following example tests the ObjectType property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each lib In Libraries

If lib.ObjectType <> ppcbObjectTypeLibrary Then

MsgBox "Test failed"

End If

Next lib

End Sub
```

Library.Parent Property

Returns the parent of this object.

Usage

Parent As [Application](#) on page 68

Arguments

None

Description

This is a Microsoft-required property.

Library.Path Property

Returns the path to the library.

Usage

Path As String

Arguments

None

Description

None

Examples

The following example displays a message box showing the name of the first available library. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each lib In Libraries

MsgBox "The path to the first available library is " & lib.Path

Exit For

Next lib

End Sub
```

Library.GetLibraryItems Method

Returns either the object collection of all items in this library or a specified item.

Usage

GetLibraryItems (*Type* as [PPcbLibraryItemType](#), *Name* as String) As Collection

Arguments

<i>name</i>	Name of LibraryItem objects to retrieve (optional). May contain wildcards, lists, and ranges. If omitted, matches any name.
<i>type</i>	Parameter specifying the type of the objects to retrieve (optional). It is ppcbLibraryItemTypeAll on page 719 by default.

Description

None

Examples

This sample displays the number of items in a library. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each lib In Libraries

count = lib.GetLibraryItems(ppcbLibraryItemTypeDecal, "DIP*").Count

MsgBox "Library " & lib.Name & " has " & count & " DIP decals"

Exit For

Next lib

End Sub
```


Library.ImportLibraryItems Method

Reads library items from PowerPCB-format or SailWind Layout-format ASCII files.

Usage

ImportLibraryItems (*Filename* as String) as Collection

Arguments

<i>name</i>	Name of the file from which to import library items.
<i>return value</i>	Returns the collection of imported items.

Return Values

Returns the collection of imported items.

Description

None

Examples

This sample imports library items from a file, and displays the number of imported items. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each lib In Libraries

Set coll = lib.ImportLibraryItems("C:\sample")

MsgBox coll.Count & " items are successfully imported"

Exit For

Next lib

End Sub
```

Library.ImportLibraryItems2 Method

Reads library items from PowerPCB-format or SailWind Layout-format ASCII files.

Usage

ImportLibraryItems2 (*Filename* as String, ImportOption as PcbImportLibMode) as Collection

Arguments

<i>Filename</i>	Name of the file from which to import library items.
<i>ImportOption</i>	specifies the preference for overwriting existing items.

Return Values

Returns the collection of just imported items.

Description

If ImportOption is pcbImportLibModePrompt, then the Application object fires the OverwriteLibraryItemPrompt event. The client application must process this event to specify whether the existing library item should be overwritten.

The LibraryItem Object

The LibraryItem object represents a physical item in a particular part library.

The LibraryItem object contains the following properties:

- [LibraryItem.Application Property](#)
- [LibraryItem.Library Property](#)
- [LibraryItem.Name Property](#)
- [LibraryItem.ObjectType Property](#)
- [LibraryItem.Parent Property](#)
- [LibraryItem.Type Property](#)

LibraryItem.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This is a Microsoft-required property.

LibraryItem.Library Property

Returns the library the library item belongs to.

Usage

Library As Object

Arguments

None

Description

None

Examples

The following sample code displays the name of the library an item belongs to. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set items = GetLibraryItems()

For Each item In items

MsgBox "The item " & item & " belongs to " & item.Library & " library"

Exit For

Next item

End Sub
```

LibraryItem.Name Property

Returns the name of the library item.

Usage

Name As String

Arguments

None

Description

This is a Microsoft-required property.

Examples

The following example displays a message box showing the name of the first item in the available libraries. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set items = GetLibraryItems()

For Each item In items

MsgBox "The first item name is " & item.Name

Exit For

Next item

End Sub
```

LibraryItem.ObjectType Property

Returns the type of object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

Examples

The following example tests the ObjectType property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set items = GetLibraryItems()

For Each item In items

If item.ObjectType <> ppcbObjectTypeLibraryItem Then

MsgBox "Test failed"

End If

Next item

End Sub
```

LibraryItem.Parent Property

Returns the parent of this object.

Usage

Parent As [Application](#) on page 68

Arguments

None

Description

This is a Microsoft-required property. The parent property of a LibraryItem object always returns the application.

LibraryItem.Type Property

Returns the type of the library item object.

Usage

Type as [PPcbLibraryItemType](#)

Arguments

None

Description

None

Examples

This sample displays the type of a library item. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set items = GetLibraryItems()

For Each item In items

Select Case item.Type

Case ppcbLibraryItemTypePartType

MsgBox "Item type: PartType"

Case ppcbLibraryItemType

MsgBox "Item type: Decal"

Case ppcbLibraryItemType

MsgBox "Item type: LogicDrawing"

Case ppcbLibraryItemType

MsgBox "Item type: Drawing"
```

End Select

Exit For

Next item

End Sub

The Measure Object

The Measure object provides access to the internal SailWind Layout unit parser. The Measure object can be either constructed from Application object (using the Application.Measure method) or obtained from the object (using the Attribute.Measure property). From the measure object, you can extract information about real value and unit.

The Measure object contains the following properties:

- Measure.Application Property
- Measure.Name Property
- Measure.Number Property
- Measure.Normalize Property
- Measure.ObjectType Property
- Measure.Parent Property
- Measure.Prefix Property
- Measure.Text Property
- Measure.Unit Property
- Measure.Value Property

Measure.Application Property

Returns the Measure object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This property identifies the object as an automation object. All automation server applications have an application object and all automation objects have an application property. This property is usually used in large automation client applications that handle large volumes of objects from different sources, such as different automation server applications, to quickly identify the application to which the object belongs.

Measure.Name Property

Returns the name of the quantity represented by the measure. For example “Capacitance” for measure “10pF”, “Voltage” for “5V”.

Usage

Name As String

Arguments

None

Examples

The following sample code retrieves the quantity name of “10pF” measure (the Capacitance). See [Code Samples](#) for more information on running this sample.

```
Sub Main

MsgBox Measure("10pF").Quantity ' displays Capacitance

End Sub
```

The following sample code retrieves the quantity name of “10k” measure (the Resistance).

```
Sub Main

MsgBox Measure("10k").Name ' displays Resistance

End Sub
```

The following sample code retrieves the quantity name of “12V” measure (the Voltage).

```
Sub Main

MsgBox Measure("12V").Name ' displays Voltage

End Sub
```

Measure.Number Property

Returns the number to combine with a prefix and unit.

Usage

Number As Double

Arguments

None

Description

This property returns the left numeric part of the measurement. You can format this number before output using standard methods available in the programming language you use for creating a client. Use this property with [Measure.Prefix Property](#) and [Measure.Unit Property](#).

Examples

The following sample code retrieves a formatted measurement (to a hundredth of a decimal point) using standard Basic formatting. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set Cap = Measure (1.12345e-12, "F")
```

```
MsgBox Format(Cap.Number, "#.###") & Cap.Prefix & Cap.Unit
```

```
End Sub
```

Measure.Normalize Property

Normalizes the text value of the measure and returns a new text.

Usage

Normalize As String

Arguments

None

Description

This property selects the appropriate unit prefix and appends a unit if it is missing. For example, it converts a time measurement from "5e-9" to "5ns."

Examples

The following sample code normalizes a capacitance measure. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set Cap = Measure(2e-10, "F")

MsgBox Cap.Normalize ' shows 200pF

End Sub
```

Measure.ObjectType Property

Returns the type of this object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

Measure.Parent Property

Returns the SailWind Layout Application object.

Usage

Parent As [Application](#) on page 68

Arguments

None

Description

None

Measure.Prefix Property

Returns the unit prefix to combine with a number and unit.

Usage

Prefix([*format* As [PPcbMeasureFormat](#) = ppcbMeasureFormatStandard]) As String

Arguments

<i>format</i>	[Optional on page 732] Indicates the format of the prefix.

Description

This property returns the prefix currently used in the measurement. Use this property with [Measure.Number Property](#) and [Measure.Unit Property](#).

Possible format values:

<i>ppcbMeasureFormatStandard</i>	To return a standard prefix representation (for example, p for Pico or k for Kilo).
<i>ppcbMeasureFormatCurrent</i>	To return a prefix in the format currently used in the Measure value.
<i>ppcbMeasureFormatShort</i>	To return a short prefix (for example, p for Pico or k for Kilo).
<i>ppcbMeasureFormatLong</i>	To return a long unit name (for example, Pico, Kilo, or Mega).

Examples

See the example for [Measure.Number Property](#)

Measure.Text Property

Sets or returns the exact text value of the measure.

Usage

Text As String

Arguments

None

Description

This property defines a custom format of a measurement. The text value consists of a number followed by an optional prefix and unit. The text value always matches the `Attribute.Value` property if an attribute represents a measure.

Examples

The following sample code demonstrates the difference between `Measure.Text`, [Measure.Value Property](#), and [Attribute.Value Property](#) properties. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set C1 = ActiveDocument.Components("C1")

Set Cap = Measure("500pF")

C1.Attributes.Add "Capacitance", Cap

MsgBox Cap 'displays 0.0000000005 (the default Value property)

MsgBox Cap.Value 'displays 0.0000000005

MsgBox Cap.Text 'displays 500pF

MsgBox C1.Attributes("Capacitance") 'displays 500pF

MsgBox C1.Attributes("Capacitance").Value 'displays 500pF (long form)

End Sub
```

Measure.Unit Property

Returns the unit name of the measure.

Usage

Unit([*format* As [PPcbMeasureFormat](#) = ppcbMeasureFormatStandard]) As String

Arguments

<i>format</i>	[Optional on page 732] Indicates the format of the unit representation.

Description

This property returns the name of the physical unit without a prefix.

Possible Format values:

<i>ppcbMeasureFormatStandard</i>	To return a standard unit representation (for example, F for Capacitance).
<i>ppcbMeasureFormatCurrent</i>	To return the unit in the format currently used in this Measure value.
<i>ppcbMeasureFormatShort</i>	To return a short unit name (for example F for Capacitance).
<i>ppcbMeasureFormatLong</i>	To return a long unit name (for example Farad for Capacitance).

Examples

See the example for [Measure.Number Property](#).

Measure.Value Property

Sets or returns the value of the measure.

Usage

Value As Double

Arguments

None

Description

This property returns a floating-point number for a measure, taking into account the unit prefix. For example, 10000 for 10K or 2e-10 for 200pF.

This is a default property, so it can be omitted in Basic scripts.

If you set a new Value, then Measure is automatically normalized. See also: [Measure.Normalize Property](#)

Examples

The following sample code demonstrates how to work with measures. See [Code Samples](#) for more information on running this sample.

```
Sub Main

'1 - Compare two existing measure attributes (.Value call may be omitted)

Set U1_Val =
  ActiveDocument.Components("U1").Attributes("Value").Measure.Value

Set U2_Val =
  ActiveDocument.Components("U2").Attributes("Value").Measure.Value

Set U1_Quantity =
  ActiveDocument.Components("U1").Attributes("Value").Measure.Name

Set U2_Quantity =
  ActiveDocument.Components("U2").Attributes("Value").Measure.Name

If U1_Quantity <> U2_Quantity then

MsgBox "Cannot compare values with different physical units"

ElseIf U1_Val < U2_Val then
```

```
MsgBox "U1 value is less than U2 value"

ElseIf U1_Val > U2_Val then

MsgBox "U1 value is greater than U2 value"

Else

MsgBox "U1 value is equal to U2 value"

End If

'2 - Check that the Resistor's Value is in range between 100K and 10M

Set R1_Val = ActiveDocument.Components("R1").Attributes("Value").Measure

If R1_Val >= Measure("100k") And R1_Val <= Measure("10M") And
  R1_Val.Name="Resistance" Then

MsgBox "Resistor Is In Range [100k, 10M]

End If

'3 - Calculate total thermal dissipation for all parts

'make sure attribute exists for all parts

For Each part In ActiveDocument.Components

If part.Attributes("Thermal.Dissipation") Is Nothing Then

Part.Attributes.Add "Thermal.Dissipation", Measure("10mW")

End If

Next

Dim Total As Measure 'declare Total as Measure object explicitly!

Set Total = Measure("0mW") 'create Measure to accamulate the total
```

```
For Each part In ActiveDocument.Components
```

```
Total = Total + part.Attributes("Thermal.Dissipation").Measure
```

```
Next
```

```
MsgBox "Total Thermal Dissipation=" & Total.Text
```

```
End Sub
```

The Net Object

The Net object represents a physical net in the open design.

This object contains the following properties:

- Net.Application Property
- Net.AssociatedNet Property
- Net.Attributes Property
- Net.Connections Property
- Net.Drawings Property
- Net.Length Property
- Net.Name Property
- Net.NetClass Property
- Net.NetClassAttributes Property
- Net.ObjectType Property
- Net.Parent Property
- Net.Pins Property
- Net.Power Property
- Net.Selected Property
- Net.Vias Property

Net.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This property identifies the object as an Automation object. All automation server applications have an application object and all automation objects have an application property. This property is usually used in automation client applications that handle large volumes of objects from different sources, such as different automation server applications, to quickly identify the application to which the object belongs.

Net.AssociatedNet Property

Returns the associated net which the net belongs to.

Usage

```
AssociatedNet as AssociatedNet
```

Arguments

None

Return Values

This property returns Nothing if the net doesn't belong to any associated net.

Net.Attributes Property

Returns the collection of all attributes assigned to the net.

Usage

Attributes As [Attributes](#) on page 130

Attributes(*name* As String) As [Attribute](#) on page 120

Arguments

<i>name</i>	Name of an existing net attribute.

Description

When an existing attribute *name* is passed to this property, it returns that net [Attribute](#) on page 120 object. If the attribute *name* does not exist, this property returns the collection of all net attributes in an [Attributes](#) on page 130 collection object.

Examples

The following sample code retrieves the number of attributes assigned to net VCC, assuming it exists in the open design, using the

Net.Connections Property

Returns the collection of all connections on the net.

Usage

Connections As [Objects](#) on page 477

Connections(*name* As String) As [Connection](#) on page 227

Arguments

<i>name</i>	Name of an existing connection.

Description

When an existing connection *name* is passed to this property, it returns that [Connection](#) on page 227 object. If the connection *name* does not exist, this property returns the collection of all connections of the net in an [Objects](#) on page 477 collection object.

Examples

The following sample code retrieves the number of connections in net VCC, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "VCC has " & ActiveDocument.Nets("VCC").Connections.Count & "  
connections."
```

```
End Sub
```

Net.Drawings Property

Returns the collection of coppers and copper planes belonging to the net object, or a specific copper or copper plane.

Usage

Drawings as Collection

Drawings (*Name* as String) as Drawing

Arguments

<i>name</i>	Name of the drawing.

Description

None

Examples

The following sample code displays the number of drawings associated with a net. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each net In ActiveDocument.Nets

If net.Drawings.Count > 0 Then

MsgBox "Net " & net.Name & " has " & net.Drawings.Count & " drawings"

Exit For

End If

Next net

End Sub
```

Net.Length Property

Returns the length of the net.

Usage

Length(*[bRouted* As Boolean = False], *[unit* As [PPcbUnit](#)]) As [Double](#) on page 477

Arguments

<i>bRouted</i>	[Optional on page 732] True to retrieve the routed length only. False to retrieve the total length, including unrouted parts approximated using the Manhattan method.
<i>unit</i>	[Optional on page 732] Unit in which the length is returned.

Description

None

Examples

The following sample code retrieves the routed, unrouted, and total length of net VCC, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
lenTotal = Format$(ActiveDocument.Nets("VCC").Length, "Fixed")
```

```
lenRouted = Format$(ActiveDocument.Nets("VCC").Length(True), "Fixed")
```

```
MsgBox "VCC's total length is (in current unit) " & lenTotal & " including  
" & lenRouted & " routed and " & lenTotal-lenRouted & " unrouted."
```

```
End Sub
```

Net.Name Property

Returns the name of the net. For example, this property returns the string “GND” for net GND. The Net.Name property is the default property for the Net object.

Usage

Name As String

Arguments

None

Examples

The following sample code lists all nets in the open design and then places that list in a custom dialog box. When a net is selected in the list box, the sample selects that net.

This sample uses the UserDialog Editor in the Sax Basic Engine in SailWind Layout. See [Code Samples](#) for more information on running this sample.

For more information, see *Sax Basic Editor On Line Help* (C:\<install_folder>\<version>\Programs\sbe5_000.hlp)

```
Dim ListNets$(10000)

Sub Main

    index = 0

    For Each nextNet In ActiveDocument.Nets

        ListNets$(index) = nextNet.Name

        index = index + 1

    Next nextNet

    ' This piece of code is automatically generated by the Basic Dialog Editor
    ' in SailWind Layout.

    Begin Dialog UserDialog 180,238,"Nets",.CallbackFunc ' %GRID:10,7,1,1

        ListBox 10,7,160,203,ListNets(),.ListBox1

        OKButton 10,210,160,21
```

```
End Dialog

Dim dlg As UserDialog

Dialog dlg

End Sub

' The following function is automatically called by the system when
' something has happened

' in the dialog; it is used to easily process user actions.

Function CallbackFunc%(DlgItem$, Action%, SuppValue%)

Select Case Action%

Case 2 ' Value changing or button pressed

If DlgItem$="ListBox1" Then

ActiveDocument.SelectObjects(ppcbObjectTypeAll, , False)

ActiveDocument.SelectObjects(ppcbObjectTypeNet, ListNets(SuppValue%))

End If

End Select

End Function
```


Net.NetClass Property

Returns the net object's net class.

Usage

NetClass as NetClass

Arguments

None

Description

None

Examples

The following sample code displays the name of the net class a net belongs to, if any. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each net In ActiveDocument.Nets

If Not net.NetClass Is Nothing Then

MsgBox "Net " & net.Name & " belongs to class " & net.NetClass.Name

Exit For

End If

Next net

End Sub
```

Net.NetClassAttributes Property

Returns the collection of all attributes assigned to the net class.

Usage

NetClassAttributes As [Attributes](#) on page 130

NetClassAttributes(*name* As String) As [Attribute](#) on page 120

Arguments

<i>name</i>	Name of an existing net class attribute.

Description

When an existing attribute *name* is passed to this property, it returns that net class [Attribute](#) on page 120 object. If the attribute *name* does not exist, this property returns the collection of all net class attributes in an [Attributes](#) on page 130 collection object.

Examples

The following sample code retrieves the number of attributes assigned to the net class of net VCC, assuming it exists in the open design, using the

Net.ObjectType Property

Returns the type of the object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

This property returns [ppcbObjectTypeNet](#) on page 719.

All database objects in the Automation server implement this property to compensate for the lack of a Visual Basic equivalent for the Visual C++ QueryInterface function.

This property is generally used:

- To identify the kind of database objects in a heterogeneous [Objects](#) on page 477 collection.
- When implementing a generic routine that depends on the type of the database object passed as argument. For example:

```
Sub DoSomething(dbObject As Object)

Select Case dbObject.ObjectType

Case ppcbObjectTypeComponent

' Do something specific to component objects

Case ppcbObjectTypeNet

' Do something specific to net objects

Case ppcbObjectTypePin

' Do something specific to pin objects

Case ppcbObjectTypeVia

' Do something specific to via objects
```

```
Case ppcbObjectTypeConnection
' Do something specific to connection objects

Case ppcbObjectTypeRouteSegment
' Do something specific to route segment objects

Case ppcbObjectTypeJumper
' Do something specific to jumper objects

Case Else
MsgBox "Not a SailWind Layout database object"

End Select

End Sub
```

Net.Parent Property

Returns the parent of the object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

None

Net.Pins Property

Returns the collection of all pins connected to the net.

Usage

Pins As [Objects](#) on page 477

Pins(*name* As String) As [Pin](#) on page 532

Arguments

<i>name</i>	Name of an existing pin.

Description

When an existing pin *name* is passed to this property, it returns that [Pin](#) on page 532 object. If the pin *name* does not exist, this property returns the collection of all pins connected to the net in an [Objects](#) on page 477 collection object.

Examples

The following sample code retrieves the number of pins connected to net VCC, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "VCC connects " & ActiveDocument.Nets("VCC").Pins.Count & " pins."
```

```
End Sub
```

Net.Power Property

Returns whether the net is a power net.

Usage

Power As Boolean

Arguments

None

Description

None

Examples

The following sample code retrieves the number of power nets in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

nbPowerNets = 0

For Each nextNet In ActiveDocument.Nets

If nextNet.Power = True Then nbPowerNets = nbPowerNets +1

Next nextNet

MsgBox "There are " & nbPowerNets & " power nets (out of " &
ActiveDocument.Nets.Count & ") in " & ActiveDocument.Name

End Sub
```

Net.Selected Property

Sets or returns whether the net is selected.

Usage

Selected As Boolean

Arguments

None

Description

You can also select a database object using the [Document.SelectObjects Method](#) and [Objects.Select Method](#) methods.

Examples

The following sample code selects net VCC only, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

ActiveDocument.SelectObjects(, , False)

ActiveDocument.Nets("VCC").Selected = True

End Sub
```


Net.Vias Property

Returns the collection of all vias connected to the net.

Usage

Vias As [Objects](#) on page 477

Vias(*name* As String) As [Via](#) on page 657

Arguments

<i>name</i>	Name of an existing via.

Description

When an existing via name is passed to this property, it returns that [Via](#) on page 657 object. If the via name does not exist, this property returns the collection of all vias connected to the net in an [Objects](#) on page 477 collection object.

Examples

The following sample code retrieves the number of vias connected to net VCC, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "VCC connects " & ActiveDocument.Nets("VCC").Vias.Count & " vias."
```

```
End Sub
```

The NetClass Object

The NetClass object represents a netclass object in the open design.

This object contains the following properties:

- NetClass.Application Property
- NetClass.Attributes Property
- NetClass.Name Property
- NetClass.Nets Property
- NetClass.ObjectType Property
- NetClass.Parent Property

NetClass.Application Property

Returns the Application object.

Usage

Application as [Application](#) on page 68

Arguments

None

Description

This is a Microsoft-required property.

NetClass.Attributes Property

Returns a collection of attributes of the net class, or a specific attribute, referenced by its name.

Usage

Attributes as [Attributes](#) on page 130

Attributes (*Name* as String) as Attribute

Arguments

<i>name</i>	Name of the attribute to retrieve.

Description

None

Examples

The following sample code displays a message box showing the number of the attribute for a net class. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each class In ActiveDocument.NetClasses

Set attrs = class.Attributes

MsgBox "There are " & attrs.Count & " attribute(s) for net class " &
class.Name

Exit For

Next class

End Sub
```

NetClass.Name Property

Returns the name of the net class.

Usage

Name as String

Arguments

None

Description

This is a Microsoft-required property.

Examples

The following sample code displays a message box showing the number of nets in the first net class in the active document. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each class In ActiveDocument.NetClasses

MsgBox "Net Class " & class.Name & " includes " & class.Nets.Count & "
nets"

Exit For

Next class

End Sub
```

NetClass.Nets Property

Returns the collection of all nets of the net class, or a specific net of the net class, referenced by its name.

Usage

Nets as Collection

Nets (*Name* as String) as [Net](#) on page 452

Arguments

<i>name</i>	Name of the net to retrieve.

Description

None

Examples

The following sample code retrieves the number of nets in a net class, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
For Each class In ActiveDocument.NetClasses
```

```
MsgBox "Net Class " & class.Name & " includes " & class.Nets.Count & "  
nets"
```

```
Exit For
```

```
Next class
```

```
End Sub
```

NetClass.ObjectType Property

Returns the type of object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

Examples

The following sample code finds a net class starting with "A." See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set objs = ActiveDocument.GetObjects(, "A*")

For Each obj In objs

If obj.ObjectType = ppcbObjectTypeNetClass Then

MsgBox obj.Name & " is a net class"

Exit For

End If

Next obj

End Sub
```

NetClass.Parent Property

Returns the parent of the object.

Usage

Parent as Document

Arguments

None

Description

This is a Microsoft-required property.

The Objects Collection Object

The Objects collection object is a collection of homogeneous or heterogeneous database objects, such as Component objects, Jumper objects, Net objects, Pin objects, Via objects, Connection objects, and RouteSegment objects, in the open design.

This object is usually retrieved using the [Document.GetObjects Method](#) method or using database object specific properties such as [Document.Components Property](#), [Document.Nets Property](#), [Document.Vias Property](#), [Document.Jumpers Property](#), [Document.Connections Property](#), and [Document.RouteSegments Property](#).

The Objects collection object contains the following properties and methods:

- [Objects.Application Property](#)
- [Objects.Count Property](#)
- [Objects.Item Property](#)
- [Objects.ItemType Property](#)
- [Objects.Next Property](#)
- [Objects.ObjectType Property](#)
- [Objects.Parent Property](#)
- [Objects.ParentObject Property](#)
- [Objects.Add Method](#)
- [Objects.Merge Method](#)
- [Objects.Remove Method](#)
- [Objects.Reset Method](#)
- [Objects.Select Method](#)
- [Objects.Sort Method](#)

Objects.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This is a Microsoft-required property.

This property identifies the object as an Automation object. All Automation server applications have an Application object and all Automation objects have an Application property. This property is usually used in Automation client applications that handle large volumes of objects from different sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

Objects.Count Property

Returns the number of objects in the collection.

Usage

Count As Long

Arguments

None

Description

None

Objects.Item Property

Returns the object given its index or its name.

Usage

Item(*index* As Long) As Object

Item(*name* As String) As Object

Arguments

<i>index</i>	Index (in the collection) of the object to retrieve.
<i>name</i>	Name of the object to retrieve.

Description

This is the default member of the Objects collection object.

This property generates an [Exception](#) if the *index* passed to this function is negative or out of range, or if the *index* or *name* argument is not valid.

Examples

The following sample code shows two different methods for iterating through all database objects in the open design, using [Component](#) on page 178 objects here as an example. The second method is preferred because it is cleaner and faster. See [Code Samples](#) for more information on running this sample.

```
' Method 1: Use the Object.Item property
```

```
Sub Main
```

```
Set comps = ActiveDocument.Components
```

```
For I=1 To comps.Count
```

```
Set thisComp = comps.Item(I)
```

```
' Do something with the component thisComp
```

```
Next I
```

```
End Sub
```

```
' Method 2: Do not use the Object.Item property (preferred method)
```

```
Sub Main
```

```
For Each nextComp in ActiveDocument.Components
```

```
    ' Do something with the component nextComp
```

```
Next nextComp
```

```
End Sub
```

Objects.ItemType Property

Returns the type of an object given its index.

Usage

ItemType(*index* As long) As [PPcbObjectType](#)

ItemType(*name* As String) As [PPcbObjectType](#)

Arguments

<i>index</i>	Index of the object in the collection to query.
<i>name</i>	Name of the object to retrieve.

Description

This property generates an [Exception](#) if the index argument is not valid.

Objects.Next Property

Returns the index of the next object of a specified type after the specified index.

Usage

Next(*index* As long, *type* As [PPcbObjectType](#)) As Long

Arguments

<i>index</i>	Index of the object in the collection to query.
<i>type</i>	Type of the object to query.

Description

This property generates an [Exception](#) if the *index* argument is not valid.

If Index = zero (0), then this function returns the index of the first item of the given type. If an item is not found, then the return value is zero (0).

Objects.ObjectType Property

Returns the type of this object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

Objects.Parent Property

Returns the parent of the object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

This is a Microsoft-required property.

Objects.ParentObject Property

Returns the parent object of the collection.

Usage

ParentObject as Object

Arguments

None

Description

If a collection has ParentObject, it means the collection is “active.” Addition or deletion of an item to or from the collection modifies the parent object as well. Otherwise, only the collection is affected.

Objects.Add Method

Adds an object to the collection.

Usage

Add(*obj* As object)

Arguments

<i>obj</i>	Object to add to the collection. It must be a database object, such as Component on page 178, Net on page 452, Jumper on page 372, Pin on page 532, Via on page 657, Connection on page 227, or RouteSegment on page 582.

Return Values

None

Description

This property generates an [Exception](#) if the object argument is not a database object.

Objects.Merge Method

Merges two object collections.

Usage

Merge(*objects* As [Objects](#) on page 477)

Arguments

<i>objects</i>	The collection with objects to merge with the current object collection.

Return Values

None

Description

This method adds all objects in the objects collection object to the current object collection.

Objects.Remove Method

Removes an object from the collection.

Usage

Remove(*index* As Long)

Remove(*name* As String)

Arguments

<i>index</i>	Index of the object to remove.
<i>name</i>	Name of the object to remove.

Return Values

None

Description

This property generates an [Exception](#) if the *index* argument is not valid.

Objects.Reset Method

Resets the object collection.

Usage

Reset()

Arguments

None

Return Values

None

Description

None

Objects.Select Method

Selects or deselects all objects in the collection.

Usage

```
Select([bSelect As Boolean = True])
```

Arguments

<i>bSelect</i>	[Optional on page 732] True to select. False to deselect.

Return Values

None

Description

None

Objects.Sort Method

Sorts objects in the collection by object name.

Usage

Sort()

Arguments

None

Return Values

None

Description

None

The Pad Object

The Pad object represents a pad in the padstack definition.

This object contains the following properties:

- [Pad.Application Property](#)
- [Pad.CnerRadius Property](#)
- [Pad.CornerRadius Property](#)
- [Pad.Diameter Property](#)
- [Pad.InnerDiameter Property](#)
- [Pad.Length Property](#)
- [Pad.Name Property](#)
- [Pad.ObjectType Property](#)
- [Pad.Offset Property](#)
- [Pad.Orientation Property](#)
- [Pad.PadStackLayer Property](#)
- [Pad.Parent Property](#)
- [Pad.Shape Property](#)
- [Pad.Width Property](#)

Pad.Application Property

Returns the application object.

Usage

Application as Application

Arguments

None

Pad.CornerRadius Property

Returns the pad's corner radius. This property should be used for the following pad shapes: `ppcbPadShapeSquare`, `ppcbPadShapeRectangularFinger`.

Usage

CornerRadius (*unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i> - [Optional]	Unit in which the value is returned. This optional argument is <code>ppcbUnitCurrent</code> by default

Pad.CornerType Property

Returns the pad's corner type. This property should be used for the following pad shapes: `ppcbPadShapeSquare`, `ppcbPadShapeRectangularFinger`.

Usage

CornerType as [PPcbPadCornerType](#)

Arguments

None

Pad.Diameter Property

Returns the pad's diameter. This property should be used for the following pad shapes:
ppcbPadShapeRound, ppcbPadShapeAnnular, ppcbPadShapeOdd.

Usage

Diameter (*unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i> - [Optional]	Unit in which the value is returned. This optional argument is ppcbUnitCurrent by default.

Pad.InnerDiameter Property

Returns the pad's inner diameter. This property should be used for the following pad shapes:
ppcbPadShapeAnnular.

Usage

InnerDiameter (*unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i> - [Optional]	

Pad.Length Property

Returns the pad's length. This property should be used for the following pad shapes:
ppcbPadShapeOvalFinger, ppcbPadShapeRectangularFinger.

Usage

Length (*unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i> - [Optional]	Unit in which the value is returned. This optional argument is ppcbUnitCurrent by default.

Pad.Name Property

Returns the name of this pad.

Usage

Name as String

Arguments

None

Pad.ObjectType Property

Returns the type of the object - ppcbObjectTypePad.

Usage

ObjectType as [PPcbObjectType](#)

Arguments

None

Pad.Offset Property

Returns the pad's offset. This property should be used for the following pad shapes:
ppcbPadShapeOvalFinger, ppcbPadShapeRectangularFinger.

Usage

Offset (unit as [PPcbUnit](#)) as Double

Arguments

unit - [Optional]	Unit in which the value is returned. This optional argument is ppcbUnitCurrent by default.

Pad.Orientation Property

Returns the pad's orientation. This property should be used for the following pad shapes:
ppcbPadShapeOvalFinger, ppcbPadShapeRectangularFinger.

Usage

Orientation as Double

Arguments

None

Pad.PadStackLayer Property

Returns the PadStackLayer Object to which this pad belongs to.

Usage

```
PadStackLayer as PadStackLayer
```

Arguments

None

Pad.Parent Property

Returns the parent of the object.

Usage

Parent as Document

Arguments

None

Pad.Shape Property

Returns the pad shape.

Usage

Shape as [PPcbPadShape](#)

Arguments

None

Pad.Width Property

Returns the pad's width. This property should be used for the following pad shapes: `ppcbPadShapeOvalFinger`, `ppcbPadShapeRectangularFinger`, `ppcbPadShapeSquare`.

Usage

Width (*unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i> - [Optional]	Unit in which the value is returned. This optional argument is <code>ppcbUnitCurrent</code> by default.

The PadStackLayer Object

The PadStackLayer object represents a layer in the padstack definition.

This object contains the following properties:

- [PadStackLayer.AntiPad Property](#)
- [PadStackLayer.Application Property](#)
- [PadStackLayer.Name Property](#)
- [PadStackLayer.Number Property](#)
- [PadStackLayer.ObjectType Property](#)
- [PadStackLayer.Pad Property](#)
- [PadStackLayer.Parent Property](#)
- [PadStackLayer.Pin Property](#)
- [PadStackLayer.ThermalPad Property](#)
- [PadStackLayer.Via Property](#)

PadStackLayer.AntiPad Property

Returns the AntiPad Object assigned to this padstack layer or Nothing if the antipad is not defined.

Usage

AntiPad as AntiPad

Arguments

None

PadStackLayer.Application Property

Returns the application object.

Usage

Application as Application

Arguments

None

PadStackLayer.Name Property

Returns the name of this padstack layer.

Usage

Name as String

Arguments

None

PadStackLayer.Number Property

Returns the number of this padstack layer. Values -2, -1, 0 correspond to PPcbPadStackLayerType.

Usage

Number as Integer

Arguments

None

PadStackLayer.ObjectType Property

Returns the type of the object - ppcbObjectTypePadStackLayer.

Usage

ObjectType as [PPcbObjectType](#)

Arguments

None

PadStackLayer.Pad Property

Returns the Pad Object assigned to this padstack layer.

Usage

Pad as Pad

Arguments

None

PadStackLayer.Parent Property

Returns the parent of the object.

Usage

Parent as Document

Arguments

None

PadStackLayer.Pin Property

Returns the Pin Object to which this padstack layer belongs to. If padstack layer belongs to via then Nothing is returned.

Usage

Pin as Pin

Arguments

None

PadStackLayer.ThermalPad Property

Returns the ThermalPad Object assigned to this padstack layer or Nothing if thermal pad is not defined.

Usage

ThermalPad as ThermalPad

Arguments

None

PadStackLayer.Via Property

Returns the Via Object to which this padstack layer belongs to. If padstack layer belongs to pin then Nothing is returned.

Usage

Via as Via

Arguments

None

The PartType Object

The PartType object represents a part type (package) of a physical component in the PCB design currently open in SailWind Layout.

The PartType object contains the following properties:

- PartType.Application Property
- PartType.Attributes Property
- PartType.Components Property
- PartType.ECORegistered Property
- PartType.Logic Property
- PartType.Name Property
- PartType.ObjectType Property
- PartType.Parent Property
- PartType.Selected Property

PartType.Application Property

Returns the SailWind Layout Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This property identifies the object as a SailWind Layout automation object. All automation server applications have an application object and all automation objects have an application property. This property is usually used in large automation client applications that handle large volumes of objects from different sources, such as different automation server applications, to quickly identify the application to which the object belongs.

PartType.Attributes Property

Returns the collection of all attributes of the part type.

Usage

Attributes As [Attributes](#) on page 130

Attributes(*name* As String) As [Attribute](#) on page 120

Arguments

<i>name</i>	Name of an existing part type attribute.

Description

When an existing attribute *name* is passed to this property, it returns that part type [Attribute](#) on page 120 object. Otherwise, it returns the collection of all part type attributes in an [Attributes](#) on page 130 collection object.

Examples

The following sample code retrieves the number of attributes on part type 7400, assuming it exists in the open design, using the

PartType.Components Property

Returns object collection of all components of this part type.

Usage

Components As [Objects](#) on page 477

Components(*name* As String) As [Component](#) on page 178

Arguments

<i>name</i>	Name of an existing component.

Description

When an existing pin *name* is passed to this property, it returns that [Component](#) on page 178 object. Otherwise, it returns the collection of all components of the part type as an [Objects](#) on page 477 collection object.

Examples

The following sample displays total number of 7400 parts.

```
Sub Main
```

```
MsgBox Str(ActiveDocument.PartTypes("7400").Components.Count)
```

```
End Sub
```

PartType.ECORegistered Property

Sets or returns the part type ECO registration status.

Usage

ECORegistered As Boolean

Arguments

None

Description

This property replaces [Component.PartTypeECORegistered Property](#)

PartType.Logic Property

Returns the logic family of the part type.

Usage

Logic As String

Arguments

None

Description

None

Examples

The following sample code retrieves the logic family of the part type 7400, assuming it exists in the open schematic. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "The Logic family of 7400 is " &  
ActiveDocument.PartTypes("7400").Logic
```

```
End Sub
```


PartType.Name Property

Returns the name of the part type. For example, this property returns the string "7402" for part type 7402.

Usage

Name As String

Arguments

None

Description

This property is the default property for the Component object.

Examples

The following sample code retrieves all part types in the open schematic and places that list in a custom dialog list box. When a part type is selected in the list box, the sample selects all parts of that part type in SailWind Layout. This sample uses the UserDialog Editor in the Sax Basic Engine in SailWind Layout. See [Code Samples](#) for more information on running this sample.

For more information, see *Sax Basic Editor On Line Help* (C:\<install_folder>\<version>\Programs\lsbe5_000.hlp)

```
Dim ListPkgs$(10000)

Sub Main

    index = 0

    For Each nextPkg In ActiveDocument.PartTypes

        ListPkgs$(index) = nextPkg.Name

        index = index + 1

    Next nextPkg

    ' This piece of code is automatically generated by the Basic Dialog Editor
    ' in SailWind Layout.

    Begin Dialog UserDialog 180,238,"Part Types",.CallbackFunc '
        %GRID:10,7,1,1

        ListBox 10,7,160,203,ListPkgs(),.ListBox1
```

```
OKButton 10,210,160,21
```

```
End Dialog
```

```
Dim dlg As UserDialog
```

```
Dialog dlg
```

```
End Sub
```

```
' The following function is automatically called by the system when  
something has happened
```

```
' in the dialog; it is used to easily process user actions.
```

```
Function CallbackFunc%(DlgItem$, Action%, SuppValue%)
```

```
Select Case Action%
```

```
Case 2 ' Value changing or button pressed
```

```
If DlgItem$="ListBox1" Then
```

```
ActiveDocument.SelectObjects(ppcbObjectTypeAll, , False)
```

```
'get part by name
```

```
Set pkg = ActiveDocument.PartTypes(ListPkgs(SuppValue%))
```

```
'select part
```

```
pkg.Selected = True
```

```
'activate sheet where first gate of the part is located
```

```
pkg.Components(1).Gates(1).Sheet.Activate
```

```
End If
```

```
End Select
```

End Function

PartType.ObjectType Property

Returns the type of the object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

This property returns [ppcbObjectTypePartType](#) on page 719.

All SailWind Layout database objects in the SailWind Layout Automation server implement this property to compensate for the lack of Visual Basic equivalent for the Visual C++® QueryInterface function.

This property is generally used:

To identify the kind of SailWind Layout database objects in a heterogeneous [Objects](#) on page 477 collection.

When implementing a generic routine that depends on the type of the SailWind Layout database object passed as argument. For example:

```
Sub DoSomething(dbObject As Object)

Select Case dbObject.ObjectType

Case ppcbObjectTypeComponent

' Do something specific to component objects

Case ppcbObjectTypeNet

' Do something specific to net objects

Case ppcbObjectTypePin

' Do something specific to pin objects

Case ppcbObjectTypePartType

' Do something specific to part type objects

Case Else
```

```
' Do something about other SailWind Layout objects
```

```
End Select
```

```
End Sub
```

PartType.Parent Property

Returns the parent of the object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

None

PartType.Selected Property

Sets or returns whether the components of the part type are selected.

Usage

Selected As Boolean

Arguments

None

Description

The part type is considered selected when one or more components of this part type are selected. You can also select a SailWind Layout database object using the [Document.SelectObjects Method](#) or [Objects.Select Method](#) methods.

Examples

The following sample code selects part type 7400 only, assuming it exists in the open schematic, and activates the sheet on which it resides. See [Code Samples](#) for more information on running this sample.

```
Sub Main

ActiveDocument.SelectObjects(, , False)

ActiveDocument.PartTypes("7400").Selected = True

End Sub
```

The Pin Object

The Pin object represents a physical pin in the open design.

This object contains the following properties:

- [Pin.Application Property](#)
- [Pin.Attributes Property](#)
- [Pin.Component Property](#)
- [Pin.DrillSize Property](#)
- [Pin.ElectricalType Property](#)
- [Pin.FunctionName Property](#)
- [Pin.Glued Property](#)
- [Pin.Highlighted Property](#)
- [Pin.IsSMD Property](#)
- [Pin.Name Property](#)
- [Pin.Net Property](#)
- [Pin.Number Property](#)
- [Pin.ObjectType Property](#)
- [Pin.PadStackLayers Property](#)
- [Pin.Parent Property](#)
- [Pin.PlaneThermal Property](#)
- [Pin.Plated Property](#)
- [Pin.PositionX Property](#)
- [Pin.PositionY Property](#)
- [Pin.Selected Property](#)
- [Pin.SlotLength Property](#)
- [Pin.SlotOffset Property](#)
- [Pin.SlotOrientation Property](#)
- [Pin.TestPoint Property](#)

Pin.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This property identifies the object as an automation object. All automation server applications have an application object and all automation objects have an application property. This property is usually used in automation client applications that handle large volumes of objects from different sources, such as different automation server applications, to quickly identify the application to which the object belongs.

Pin.Attributes Property

Returns the collection of all attributes of the pin.

Usage

Attributes As [Attributes](#) on page 130

Attributes(*name* As String) As [Attribute](#) on page 120

Arguments

<i>name</i>	Name of an existing pin attribute.

Description

When an existing attribute name is passed to this property, it returns that pin [Attribute](#) on page 120 object. If the attribute name does not exist, this property returns the collection of all pin attributes in an [Attributes](#) on page 130 collection object.

Examples

The following sample code retrieves the number of attributes assigned to pin U1.1, assuming it exists in the open design, using the

Pin.Component Property

Returns the component owning the pin.

Usage

Component As [Component](#) on page 178

Arguments

None

Description

None

Examples

The following sample code retrieves the component to which pin U1.1 belongs, assuming U1.1 exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

MsgBox "Pin U1.1 belongs to component " &
    ActiveDocument.Pins("U1.1").Component.Name

End Sub
```

Pin.DrillSize Property

Returns the drill size of the pin.

Usage

DrillSize(*unit* As [PPcbUnit](#)) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the drill size value is returned.

Description

This property returns 0.0 for SMD pins.

Examples

The following sample code retrieves the drill size of pin U1.1, assuming U1.1 exists in the open design. See [Code Samples](#) or more information on running this sample.

```
Sub Main
```

```
MsgBox "Pin U1.1 drill size is " & ActiveDocument.Pins("U1.1").DrillSize
```

```
End Sub
```

Pin.ElectricalType Property

Returns the gate type of the pin.

Usage

ElectricalType As [PPcbPinElectricalType](#)

Arguments

None

Description

None

Examples

The following sample code retrieves the electrical type of pin U1, assuming U1 exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Function ElectricalTypeName(theType As Long) As String
```

```
    Select Case theType
```

```
        Case ppcbElectricalTypeUnknown
```

```
            ElectricalTypeName="unknown"
```

```
        Case ppcbElectricalTypeSource
```

```
            ElectricalTypeName="source"
```

```
        Case ppcbElectricalTypeBidirectional
```

```
            ElectricalTypeName="bidirectional"
```

```
        Case ppcbElectricalTypeOpenCollector
```

```
            ElectricalTypeName="open collector"
```

```
        Case ppcbElectricalTypeOrTieableSource
```

```
            ElectricalTypeName="tieable source"
```

```
Case ppcbElectricalTypeTristate
```

```
ElectricalTypeName="tristate"
```

```
Case ppcbElectricalTypeLoad
```

```
ElectricalTypeName="load"
```

```
Case ppcbElectricalTypeTerminator
```

```
ElectricalTypeName="terminator"
```

```
Case ppcbElectricalTypePower
```

```
ElectricalTypeName="power"
```

```
Case ppcbElectricalTypeGround
```

```
ElectricalTypeName="ground"
```

```
Case Else
```

```
ElectricalTypeName="unknown"
```

```
End Select
```

```
End Function
```

```
Sub Main
```

```
MsgBox "Pin U1.1 electrical type is " &  
    ElectricalTypeName(ActiveDocument.Pins("U1.1").ElectricalType)
```

```
End Sub
```

Pin.FunctionName Property

Returns the function name of the pin.

Usage

FunctionName As String

Arguments

None

Description

None

Examples

The following sample code retrieves the function name of pin U1, assuming U1 exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

MsgBox "Pin U1.1 function name is " &
  ActiveDocument.Pins("U1.1").FunctionName

End Sub
```

Pin.Glued Property

Sets or returns whether the pin is glued.

Usage

Glued As Boolean

Arguments

None

Description

None

Examples

The following sample code glues all pins in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main  
  
For Each nextPin In ActiveDocument.Pins  
  
nextPin.Glued = True  
  
Next nextPin  
  
End Sub
```


Pin.Highlighted Property

Returns whether the pin is highlighted.

Usage

Highlighted as Boolean

Arguments

None

Pin.IsSMD Property

Returns whether the pin is SMD.

Usage

IsSMD As Boolean

Arguments

None

Description

None

Examples

The following sample code retrieves the number of SMD pins in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

nbSMDPins = 0

For Each nextPin In ActiveDocument.Pins

If nextPin.IsSMD = True Then nbSMDPins = nbSMDPins +1

Next nextPin

MsgBox "There are " & nbSMDPins & " SMD pins (out of " &
ActiveDocument.Pins.Count & ") in " & ActiveDocument.Name

End Sub
```

Pin.Name Property

Returns the name of the pin.

Usage

Name As String

Arguments

None

Description

For example, this property returns the string "U1.1" for pin U1.1.

This property is the default property for the Pin object.

Examples

The following sample code lists all pins in the open design by name and then places that list in a custom dialog box. When a pin is selected in the list box, the sample selects that pin.

This sample uses the UserDialog Editor in the Sax Basic Engine in SailWind Layout. See [Code Samples](#) for more information on running this sample.

For more information, see *Sax Basic Editor On Line Help* (C:\<install_folder>\<version>\Programs\sbe5_000.hlp)

```
Dim ListPins$(10000)

Sub Main

    index = 0

    For Each nextPin In ActiveDocument.Pins

        ListPins$(index) = nextPin.Name

        index = index + 1

    Next nextPin

    ' This piece of code is automatically generated by the Basic Dialog Editor
    ' in SailWind Layout.

    Begin Dialog UserDialog 180,238,"Pins",.CallbackFunc ' %GRID:10,7,1,1
```

```
ListBox 10,7,160,203,ListPins(),.ListBox1

OKButton 10,210,160,21

End Dialog

Dim dlg As UserDialog

Dialog dlg

End Sub

' The following function is automatically called by the system when
' something has happened

' in the dialog; it is used to easily process user actions.

Function CallbackFunc%(DlgItem$, Action%, SuppValue%)

Select Case Action%

Case 2 ' Value changing or button pressed

If DlgItem$="ListBox1" Then

ActiveDocument.SelectObjects(ppcbObjectTypeAll, , False)

ActiveDocument.SelectObjects(ppcbObjectTypePin, ListPins(SuppValue%))

End If

End Select

End Function
```

Pin.Net Property

Returns the net connected to the pin.

Usage

Net As [Net](#) on page 452

Arguments

None

Description

This property returns nothing if the pin is not connected.

Examples

The following sample code identifies the net connected to pin U1.1, assuming U1.1 exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

MsgBox "Pin U1.1 is connected to net " &
ActiveDocument.Pins("U1.1").Net.Name

End Sub
```

Pin.Number Property

Returns the number of the pin.

Usage

Number As Long

Arguments

None

Description

None

Examples

The following sample code retrieves the pin number of pin U1.1, assuming U1.1 exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox "Pin U1.1 is pin number " & ActiveDocument.Pins("U1.1").Number & "  
in component " & ActiveDocument.Pins("U1.1").Component.Name
```

```
End Sub
```

Pin.ObjectType Property

Returns the type of the object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

This property returns [ppcbObjectTypePin](#) on page 719.

All database objects in the automation server implement this property to compensate for the lack of a Visual Basic equivalent for the Visual C++ QueryInterface function.

This property is generally used:

- To identify the kind of database objects in a heterogeneous [Objects](#) on page 477 collection.
- When implementing a generic routine that depends on the type of the database object passed as argument. For example:

```
Sub DoSomething(dbObject As Object)

Select Case dbObject.ObjectType

Case ppcbObjectTypeComponent

' Do something specific to component objects

Case ppcbObjectTypeNet

' Do something specific to net objects

Case ppcbObjectTypePin

' Do something specific to pin objects

Case ppcbObjectTypeVia

' Do something specific to via objects
```

```
Case ppcbObjectTypeConnection
' Do something specific to connection objects

Case ppcbObjectTypeRouteSegment
' Do something specific to route segment objects

Case ppcbObjectTypeJumper
' Do something specific to jumper objects

Case Else
MsgBox "Not a SailWind Layout database object"

End Select

End Sub
```


Pin.PadStackLayers Property

Returns the collection of all padstack layers for this pin.

Usage

PadStackLayers as Objects

PadStackLayers(*layerName* as String) as PadStackLayer

Arguments

<i>layerName</i>	Name of padstack layer

Return Values

When a layer name is passed to this property, it returns that PadStackLayer Object. If the name is not specified, this property returns the collection of all padstack layers in an Objects collection object.

Examples

```
For Each comp In Application.ActiveDocument.Components
```

```
For Each pin In comp.Pins
```

```
For Each layer in pin.PadStackLayers
```

```
MsgBox layer.Number & ", " & layer.Name
```

```
pad = layer.Pad
```

```
MsgBox pad.Name & ", " & pad.Shape & ", " & pad.Diameter & ", "
```

```
& pad.InnerDiameter & ", " & pad.Width & ", "
```

```
& pad.Length & ", " & pad.offset & ", "
```

```
& pad.Orientation & ", " & pad.CornerType & ", "
```

```
& pad.CornerRadius
```

```
thermalpad = layer.ThermalPad
```

```
If thermalpad Is Nothing Then
```

```
    MsgBox "Thermal pad not defined on this layer"
```

```
Else
```

```
    MsgBox thermalpad.Name & ", " & thermalpad.Shape & ", "
```

```
        & thermalpad.InnerSize & ", "
```

```
        & thermalpad.OuterSize & ", " & thermalpad.Spokes & ", "
```

```
        & thermalpad.SpokeAngle & ", " & thermalpad.SpokeWidth
```

```
End If
```

```
antipad = layer.AntiPad
```

```
If antipad Is Nothing Then
```

```
    MsgBox "Anti pad not defined on this layer"
```

```
Else
```

```
    MsgBox antipad.Name & ", " & antipad.Shape & ", "
```

```
        & antipad.Size
```

```
End If
```

```
Next layer
```

```
Next pin
```

Next comp

Pin.Parent Property

Returns the parent of the object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

None

Pin.PlaneThermal Property

Returns whether the pin has a plane thermal.

Usage

PlaneThermal As Boolean

Arguments

None

Description

None

Examples

The following sample code retrieves the number of plane thermal pins in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

nbPTPins = 0

For Each nextPin In ActiveDocument.Pins

If nextPin.PlaneThermal = True Then nbPTPins = nbPTPins +1

Next nextPin

MsgBox "There are " & nbPTPins & " plane thermal pins (out of " &
ActiveDocument.Pins.Count & ") in " & ActiveDocument.Name

End Sub
```

Pin.Plated Property

Returns whether the pin is plated.

Usage

Plated As Boolean

Arguments

None

Description

None

Examples

The following sample code retrieves the number of non-SMD, nonplated pins in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
nbPlatedPins = 0
```

```
For Each nextPin In ActiveDocument.Pins
```

```
  If nextPin.Plated = False And nextPin.IsSMD = False Then nbPlatedPins =  
    nbPlatedPins +1
```

```
Next nextPin
```

```
MsgBox "There are " & nbPlatedPins & " non-SMD non-plated pins (out of " &  
  ActiveDocument.Pins.Count & ") in " & ActiveDocument.Name
```

```
End Sub
```

Pin.PositionX Property

Returns the x-coordinate of the pin.

Usage

PositionX(*unit* As [PPcbUnit](#)) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the x-coordinate is returned.

Description

None

Examples

The following sample code retrieves the location of pin U1.1, assuming U1.1 exists in the open design, in current design units. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set pinU1_1 = ActiveDocument.Pins("U1.1")
```

```
MsgBox "U1.1 position is = (" & pinU1_1.PositionX & ", " &  
pinU1_1.PositionY & ")"
```

```
End Sub
```

Pin.PositionY Property

Returns the y-coordinate of the pin.

Usage

PositionY(*[unit As PPcbUnit]*) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the y-coordinate is returned.

Description

None

Examples

The following sample code retrieves the location of the pin U1.1, assuming U1.1 exists in the open design, in current design units. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set pinU1_1 = ActiveDocument.Pins("U1.1")
```

```
MsgBox "U1.1 position is = (" & pinU1_1.PositionX & ", " &  
pinU1_1.PositionY & ")"
```

```
End Sub
```


Pin.Selected Property

Sets or returns whether the pin is selected.

Usage

Selected As Boolean

Arguments

None

Description

You can also select a SailWind Layout database object using the [Document.SelectObjects Method](#) and [Objects.Select Method](#) methods.

Examples

The following sample code selects pin U1.1 only, assuming U1.1 exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

ActiveDocument.SelectObjects(, False)

ActiveDocument.Pins("U1.1").Selected = True

End Sub
```

Pin.SlotLength Property

Returns the slot length of the pin.

Usage

SlotLength (*Unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the result is to be represented. This optional argument is ppcbUnitCurrent on page 719 by default.

Description

If a pin is not slotted, it returns zero. The property could be used to check if a pin has a slot or round drill.

Examples

The following sample code displays the slot length of the selected pin. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypePin,,True)

For Each pin In selected

MsgBox "Slot length:" & pin.SlotLength

Exit For

Next pin

End Sub
```

Pin.SlotOffset Property

Returns the slot offset of the pin.

Usage

SlotOffset (*Unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the result is to be represented. This optional argument is ppcbUnitCurrent on page 719 by default.

Description

If pin is not slotted, it returns zero.

Examples

The following sample code displays the slot length of the selected pin. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypePin,,True)

For Each pin In selected

MsgBox "Slot offset:" & pin.SlotOffset

Exit For

Next pin

End Sub
```

Pin.SlotOrientation Property

Returns the slot orientation of the pin, in degrees.

Usage

SlotOffset as Double

Arguments

None

Description

If pin is not slotted, it returns zero.

Examples

The following sample code displays the slot orientation of the selected pin. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypePin,,True)

For Each pin In selected

MsgBox "Slot orientation:" & pin.SlotOrientation

Exit For

Next pin

End Sub
```

Pin.TestPoint Property

Sets or returns whether the pin is a test point.

Usage

TestPoint As [PPcbTestPointType](#)

Arguments

None

Description

None

Examples

The following sample code removes all test points from the open design and adds one top layer test point per net, arbitrarily choosing a pin on the net for the test point. See [Code Samples](#) for more information on running this sample.

```
Sub Main

' Remove all test points

For Each nextPin In ActiveDocument.Pins

nextPin.TestPoint = ppcbTestPointNone

Next nextPin

' Add one top layer test point per net

For Each nextNet In ActiveDocument.Nets

Set arbitPin = nextNet.Pins.Item(0)

arbitPin.TestPoint = ppcbTestPointTopLayer

Next nextNet

End Sub
```

The Polyline Object

The Polyline object represents a physical polyline object in the open design.

This object contains the following properties:

- [Polyline.Application Property](#)
- [Polyline.CenterX Property](#)
- [Polyline.CenterY Property](#)
- [Polyline.Geometry Property](#)
- [Polyline.Layer Property](#)
- [Polyline.LineStyle Property](#)
- [Polyline.LineWidth Property](#)
- [Polyline.ObjectType Property](#)
- [Polyline.OutlineType Property](#)
- [Polyline.Parent Property](#)
- [Polyline.Points Property](#)
- [Polyline.Radius Property](#)
- [Polyline.ShapeType Property](#)

Polyline.Application Property

Returns the Application object.

Usage

Application as [Application](#) on page 68

Arguments

None

Description

This is a Microsoft-required property.

Polyline.CenterX Property

Returns a center x-coordinate for an arc in the polyline.

Usage

CenterX (*Corner* as Long, *Unit* as [PPcbUnit](#), *Origin* as [PPcbOriginType](#)) as Double

Arguments

<i>corner</i>	Starting corner for the arc.
<i>unit</i>	[Optional on page 732] Unit in which the result is to be represented. This optional argument is ppcbUnitCurrent on page 719 by default.
<i>Origin</i>	[Optional on page 732]Type of reference point from which the result is counted. The default value is ppcbOriginTypeDesign on page 719.

Description

None

Examples

The following sample code displays center coordinates of the selected polyline. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,True)

For Each drw In selected

Dim geom As Object

For Each geom In drw.Geometry

If geom.ObjectType = ppcbObjectTypePolyline Then

p = geom.Points

n = UBound(p, 1)

For i = 1 To n

If p(i, 3) <> 0 Then
```



```
m="Arc " & i & " : (" & p(i, 1) & ", " & p(i, 2) & ") "
```

```
m = m & "Center: (" & geom.CenterX(i) & ", " & geom.CenterY(i) & ")"
```

```
MsgBox m
```

```
End If
```

```
Next i
```

```
Exit For
```

```
End If
```

```
Next geom
```

```
Exit For
```

```
Next drw
```

```
End Sub
```

Polyline.CenterY Property

Returns a center y-coordinate for an arc in the polyline.

Usage

CenterY (*Corner* as Long, *Unit* as [PPcbUnit](#), *Origin* as [PPcbOriginType](#)) as Double

Arguments

<i>corner</i>	Starting corner for the arc.
<i>unit</i>	[Optional on page 732] Unit in which the result is to be represented. This optional argument is ppcbUnitCurrent on page 719 by default.
<i>origin</i>	[Optional on page 732]Type of reference point from which the result is counted. The default value is ppcbOriginTypeDesign on page 719.

Description

None

Examples

The following sample code displays center coordinates of the selected polyline. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,True)

For Each drw In selected

Dim geom As Object

For Each geom In drw.Geometry

If geom.ObjectType = ppcbObjectTypePolyline Then

p = geom.Points

n = UBound(p, 1)

For i = 1 To n

If p(i, 3) <> 0 Then
```

```
m="Arc " & i & " : (" & p(i, 1) & ", " & p(i, 2) & ") "
```

```
m = m & "Center: (" & geom.CenterX(i) & ", " & geom.CenterY(i) & ")"
```

```
MsgBox m
```

```
End If
```

```
Next i
```

```
Exit For
```

```
End If
```

```
Next geom
```

```
Exit For
```

```
Next drw
```

```
End Sub
```

Polyline.Geometry Property

Returns a collection of objects, currently, polylines or circles, representing this object's child geometry objects.

Usage

Geometry as Collection

Arguments

None

Description

None

Examples

The following sample code shows the number of child objects. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,TRUE)

For Each drw In selected

For Each geom In drw.Geometry

MsgBox "Child object count: " & geom.Geometry.Count

Next geom

Next drw

End Sub
```

Polyline.Layer Property

Returns the layer number of the object

Usage

Layer as Long

Arguments

None

Description

None

Examples

The following sample code shows the layer number of the selected polyline. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,TRUE)

For Each drw In selected

Dim geom as Object

For Each geom In drw.Geometry

MsgBox "Layer number: " & geom.Layer

Next geom

Next drw

End Sub
```

Polyline.LineStyle Property

Returns the line style of the polyline.

Usage

LineStyle as [PPcbLineStyle](#)

Arguments

None

Description

None

Examples

None

Polyline.LineWidth Property

Returns the width of the polyline.

Usage

LineWidth (*Unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the result is to be represented. This optional argument is ppcbUnitCurrent on page 719 by default.

Description

None

Examples

The following sample code shows the line width of the selected polyline. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing, , TRUE)

For Each drw In selected

For Each geom In drw.Geometry

MsgBox "Line width: " & geom.LineWidth

Next geom

Next drw

End Sub
```

Polyline.ObjectType Property

Returns the type of object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

Examples

The following sample code tests the ObjectType property. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each drw In ActiveDocument.Drawings

For Each geom In drw.Geometry

t = geom.ObjectType

If t <> pcbObjectTypePolyline And t <> ppcbObjectTypeCircle Then

MsgBox "Test failed"

End If

Next geom

Next drw

End Sub
```


Polyline.OutlineType Property

Returns the outline type of the polyline.

Usage

OutlineType as [PPcbOutlineType](#)

Arguments

None

Description

None

Examples

The following sample code shows the outline type of the selected polyline. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,TRUE)

For Each drw In selected

Dim geom as Object

For Each geom In drw.Geometry

Select Case geom.OutlineType

Case ppcbOutLineTypeCenter

s="Center line"

Case ppcbOutLineTypeOuter

s="Outer line"

Case ppcbOutLineTypeInner

s="Inner line"
```

```
End Select
```

```
MsgBox "Outline type: " & s
```

```
Next geom
```

```
Next drw
```

```
End Sub
```

Polyline.Parent Property

Returns the parent of the object.

Usage

Parent as Document

Arguments

None

Description

This is a Microsoft-required property.

Polyline.Points Property

Returns coordinates of the polyline points.

Usage

Points (*Unit* as [PPcbUnit](#), *Origin* as [PPcbOriginType](#)) as Variant

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the result is to be represented. This optional argument is ppcbUnitCurrent on page 719 by default.
<i>origin</i>	[Optional on page 732]Type of reference point from which the result is counted. The default value is ppcbOriginTypeDesign on page 719.

Return Values

A two-dimensional array of coordinates representing ends of the polyline's segments. Points (n, 1) are the X coordinate and points (n, 2) are the Y coordinate. Points (n, 3) contain the arc angle, if the point n and point n + 1 are connected with an arc, or zero. The angle is positive if the arc has counterclockwise direction, and negative when the arc has clockwise direction. The angle is measured in degrees.

Description

For shape types Hollow, Filled, and Void, the last point duplicates the first one. For example, points array dimensions of a filled rectangle object (5, 3). The angle for the last point is always 0.

Examples

The following sample code displays the corner coordinates of the selected polyline. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing, , True)

For Each drw In selected

    For Each geom In drw.Geometry

        If geom.ObjectType = ppcbObjectTypePolyline Then

            p = geom.Points

            n = UBound(p, 1)

            For i = 1 To n - 1
```

```
If p(i, 3) = 0 Then

m="Segment " & i

m = m & " From (" & p(i, 1) & ", " & p(i, 2) & ")"

m = m & " To (" & p(i + 1, 1) & ", " & p(i + 1, 2) & ")"

Else

m="Arc " & i

m = m & " From (" & p(i, 1) & ", " & p(i, 2) & ")"

m = m & " To (" & p(i + 1, 1) & ", " & p(i + 1, 2) & ")"

m = m & " Angle: " & p(i, 3)

End If

MsgBox m

Next i

Exit For

End If

Next geom

Exit For

Next drw

End Sub
```

Polyline.Radius Property

Returns a radius for an arc in the polyline.

Usage

Radius (*Corner* as Long, *Unit* as [PPcbUnit](#)) as Double

Arguments

<i>corner</i>	Starting corner for the arc.
<i>unit</i>	[Optional on page 732] Unit in which the result is to be represented. This optional argument is ppcbUnitCurrent on page 719 by default.

Description

None

Examples

The following sample code displays the arc radii of the selected polyline. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,True)

For Each drw In selected

Dim geom As Object

For Each geom In drw.Geometry

If geom.ObjectType = ppcbObjectTypePolyline Then

p = geom.Points

n = UBound(p, 1)

For i = 1 To n

If p(i, 3) <> 0 Then

m="Arc " & i & " : (" & p(i, 1) & ", " & p(i, 2) & ") "
```

```
m = m & "Radius: " & geom.Radius(i)
```

```
MsgBox m
```

```
End If
```

```
Next i
```

```
Exit For
```

```
End If
```

```
Next geom
```

```
Exit For
```

```
Next drw
```

```
End Sub
```

Polyline.ShapeType Property

Returns the shape type of the polyline.

Usage

ShapeType as [PPcbShapeType](#)

Arguments

None

Description

None

Examples

The following sample code shows the shape type of the selected polyline. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeDrawing,,TRUE)

For Each drw In selected

Dim geom as Object

For Each geom In drw.Geometry

Select Case geom.ShapeType

Case ppcbShapeTypeOpen

s="Open"

Case ppcbShapeTypeHollow

s="Hollow"

Case ppcbShapeTypeFilled

s="Filled"
```



```
Case ppcbShapeTypeVoid
```

```
s="Void"
```

```
End Select
```

```
MsgBox "Polyline type: " & s & " shape"
```

```
Next geom
```

```
Next drw
```

```
End Sub
```

The RouteSegment Object

The RouteSegment object represents a physical route segment, also called a route, in the open design.

The RouteSegment object contains the following properties:

- [RouteSegment.Application Property](#)
- [RouteSegment.Layer Property](#)
- [RouteSegment.Length Property](#)
- [RouteSegment.Name Property](#)
- [RouteSegment.Net Property](#)
- [RouteSegment.ObjectType Property](#)
- [RouteSegment.Parent Property](#)
- [RouteSegment.Points Property](#)
- [RouteSegment.SegmentType Property](#)
- [RouteSegment.Selected Property](#)
- [RouteSegment.Width Property](#)

RouteSegment.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This property identifies the object as an Automation object. All Automation server applications have an Application object and all Automation objects have an Application property. This property is usually used in Automation client applications that handle large volumes of objects from different sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

RouteSegment.Layer Property

Returns the mounting layer of the trace segment.

Usage

Layer As Long

Arguments

None

Description

None

Examples

The following sample code retrieves the layer on which the first routed trace segment found in the open design exists, assuming at least one trace segment exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set firstRteSeg = ActiveDocument.RouteSegments(1)
```

```
MsgBox "Route segment " & firstRteSeg.Name & " is on layer " &  
firstRteSeg.Layer
```

```
End Sub
```

RouteSegment.Length Property

Returns the length of the trace segment.

Usage

Length(*unit* As [PPcbUnit](#)) As Double

Arguments

<i>unit</i>	[Optional] on page 732] Unit in which the length value is returned.

Description

None

Examples

The following sample code retrieves the length of the first trace segment found in the open design, assuming at least one trace segment exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set firstRteSeg = ActiveDocument.RouteSegments(1)
```

```
MsgBox "Route segment " & firstRteSeg.Name & " length is " &  
firstRteSeg.Length
```

```
End Sub
```

RouteSegment.Name Property

Returns the name of the trace segment.

Usage

Name As String

Arguments

None

Description

This property is the default property for the RouteSegment object.

Examples

The following sample code lists all trace segments in the open design by name and then places that list in a custom dialog box. When a trace segment is selected in the list box, the sample selects that trace segment.

This sample uses the UserDialog Editor in the Sax Basic Engine. See [Code Samples](#) for more information on running this sample.

For more information, see *Sax Basic Editor On Line Help* (C:\<install_folder>\<version>\Programs\sbe5_000.hlp)

```
Dim ListRteSegs$(10000)

Sub Main

index = 0

For Each nextRteSeg In ActiveDocument.RouteSegments

ListRteSegs$(index) = nextRteSeg.Name

index = index + 1

Next nextRteSeg

' This piece of code is automatically generated by the Basic Dialog Editor
in SailWind Layout.

Begin Dialog UserDialog 180,238,"Route Segments",.CallbackFunc '
%GRID:10,7,1,1
```

```

ListBox 10,7,160,203,ListRteSegs(),.ListBox1

OKButton 10,210,160,21

End Dialog

Dim dlg As UserDialog

Dialog dlg

End Sub

' The following function is automatically called by the system when
  something has happened

' in the dialog; it is used to easily process user actions.

Function CallbackFunc%(DlgItem$, Action%, SuppValue%)

Select Case Action%

Case 2 ' Value changing or button pressed

If DlgItem$="ListBox1" Then

ActiveDocument.SelectObjects(ppcbObjectTypeAll, , False)

ActiveDocument.SelectObjects(ppcbObjectTypeRouteSegment,
  ListRteSegs(SuppValue%))

End If

End Select

End Function

```

RouteSegment.Net Property

Returns the net connected to the trace segment.

Usage

Net As [Net](#) on page 452

Arguments

None

Description

None

Examples

The following sample code retrieves the net of the first trace segment found in the open design, assuming at least one trace segment exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set firstRteSeg = ActiveDocument.RouteSegments(1)
```

```
MsgBox "Route segment " & firstRteSeg.Name & " is connected on net " &  
firstRteSeg.Net.Name
```

```
End Sub
```


RouteSegment.ObjectType Property

Returns the type of the object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

This property returns [ppcbObjectTypeRouteSegment](#) on page 719.

All database objects in the Automation server implement this property to compensate for the lack of a Visual Basic equivalent for the Visual C++ QueryInterface function.

This property is generally used:

- To identify the kind of SailWind Layout database objects in a heterogeneous [Objects](#) on page 477 collection.
- When implementing a generic routine that behavior depends on the type of database object passed as argument. For example:

```
Sub DoSomething(dbObject As Object)

Select Case dbObject.ObjectType

Case ppcbObjectTypeComponent

' Do something specific to component objects

Case ppcbObjectTypeNet

' Do something specific to net objects

Case ppcbObjectTypePin

' Do something specific to pin objects

Case ppcbObjectTypeVia

' Do something specific to via objects
```

```
Case ppcbObjectTypeConnection
' Do something specific to connection objects

Case ppcbObjectTypeRouteSegment
' Do something specific to route segment objects

Case ppcbObjectTypeJumper
' Do something specific to jumper objects

Case Else
MsgBox "Not a SailWind Layout database object"

End Select

End Sub
```

RouteSegment.Parent Property

Returns the parent of the object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

None

RouteSegment.Points Property

Returns the array of points defining the trace segment.

Usage

Points(*[unit As PPcbUnit]*) AsVariant

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the point coordinate values are returned.

Description

None

RouteSegment.SegmentType Property

Returns the type of the trace segment.

Usage

SegmentType As [PPcbSegmentType](#)

Arguments

None

Description

None

Examples

The following sample code retrieves the segment type of the first trace segment found in the open design, assuming at least one trace segment exists. See [Code Samples](#) for more information on running this sample.

```
Function SegmentTypeName(theType As Long) As String

Select Case theType

Case ppcbSegmentUnknown

SegmentTypeName="unknown"

Case ppcbSegmentLine

SegmentTypeName="line"

Case ppcbSegmentArc

SegmentTypeName="arc"

Case Else

SegmentTypeName="unknown"

End Select

End Function
```

```
Sub Main
```

```
Set firstRteSeg = ActiveDocument.RouteSegments(1)
```

```
MsgBox "Route segment " & firstRteSeg.Name & " is of type " &  
SegmentTypeName(firstRteSeg.SegmentType)
```

```
End Sub
```

RouteSegment.Selected Property

Sets or returns whether the trace segment is selected.

Usage

Selected As Boolean

Arguments

None

Description

None

Examples

The following sample code selects the first trace segment found in the open design, assuming at least one trace segment exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.SelectObjects(, , False)
```

```
ActiveDocument.RouteSegments(1).Selected = True
```

```
End Sub
```

RouteSegment.Width Property

Returns the width of the trace segment or the width of draw lines used to fill copper planes.

Usage

Width(*[unit As PPcbUnit]*) As Double

Arguments

Argument	Description
<i>unit</i>	[Optional on page 732] Unit in which the width value is returned.

Description

None

Examples

The following sample code retrieves the width of the first trace segment found in the open design, assuming at least one trace segment exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set firstRteSeg = ActiveDocument.RouteSegments(1)

MsgBox "Route segment " & firstRteSeg.Name & " width is " &
firstRteSeg.Width

End Sub
```


The SBP Object

The SBP object represents the physical substrate bond pad of a die part.

You can retrieve the SBP object, or collection of SBP objects, only from the [Component](#) on page 178 object, which represents a die part ([Component.IsDiePart Property](#) returns True), or from other die part constituents (such as [CBP](#) on page 147 or [Wirebond](#) on page 704).

SBP objects are not selectable.

You cannot retrieve the collection of all SBP objects from a document using Document.GetObjects (ppcbObjectTypeSBP). This method returns an empty collection. Also, Document.GetObjects (ppcbObjectTypeAll) does not return CBP, SBP, or Wirebond objects. You can get only collections of CBPs, SBPs, or Wirebonds using the Component object methods, if the component is a die part ([Component.IsDiePart Property](#)).

The SBP object contains the following properties:

- [SBP.Application Property](#)
- [SBP.CBPs Property](#)
- [SBP.Component Property](#)
- [SBP.Function Property](#)
- [SBP.Layer Property](#)
- [SBP.Length Property](#)
- [SBP.Name Property](#)
- [SBP.ObjectType Property](#)
- [SBP.Orientation Property](#)
- [SBP.Parent Property](#)
- [SBP.Position X Property](#)
- [SBP.PositionY Property](#)
- [SBP.Shape Property](#)
- [SBP.Tier Property](#)
- [SBP.Width Property](#)
- [SBP.Wirebonds Property](#)

SBP.Application Property

Returns the Application object.



Restriction:

This information applies to only the BGA toolkit.

Usage

Application As [Application](#) on page 68

Arguments

None

Descriptions

This property identifies the object as an Automation object. All Automation server applications have an application object and all Automation objects have an application property. This property is usually used in Automation client applications that handle large volumes of objects from different sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

This is a Microsoft-required property.

SBP.CBPs Property

Returns the collection of CBPs linked to the SBP object.



Restriction:

This information applies to only the BGA toolkit.

Usage

CBPs As Objects

Arguments

None

Description

None

Examples

The following sample code retrieves the number of CBPs connected to each SBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aSBP in comp.SBPs

MsgBox "Number of CBPs linked to " & aSBP.Name & ": " & aSBP.CBPs.Count

Next

End If
```

Next

End Sub

SBP.Component Property

Returns the Component object of the SBP.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

Component As Component

Arguments

None

Description

None

Examples

The following sample code represents a subroutine that depends on the SBP object passed as an argument. The code retrieves the name of the component and the name of the decal to which the SBP belongs. See [Code Samples](#) for more information on running this sample.

```
Sub QuerySBP(aSBP As SBP)
```

```
MsgBox "Substrate Bond Pad" & aSBP.Name &
```

```
"belongs to component" & aSBP.Component.Name &
```

```
"(" & aSBP.Component.Decal &")"
```

```
End Sub
```

SBP.Function Property

Returns the function of the SBP.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

Function As String

Arguments

None

Description

None

Examples

The following sample code retrieves the function of each SBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aSBP in comp.SBPs

MsgBox aSBP.Name & " Function: " & aSBP.Function

Next

End If

Next

End Sub
```

SBP.Layer Property

Returns the LIQ layer of the SBP.



Restriction:

This information applies to only the BGA toolkit.

Usage

Layer As String

Arguments

None

Description

None

Examples

The following sample code retrieves the layer of each SBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aSBP in comp.SBPs

MsgBox aSBP.Name & " Layer: " & aSBP.Layer

Next

End If

Next

End Sub
```

SBP.Length Property

Returns the length of the SBP.



Restriction:

This information applies to only the BGA toolkit.

Usage

Length([unit As [PPcbUnit](#) = ppcbUnitCurrent]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the length value is returned.

Description

None

Examples

The following sample code retrieves the length of each SBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aSBP in comp.SBPs

MsgBox aSBP.Name & " Length: " & Format (aSBP.Length, "#.###")

Next

End If

Next

End Sub
```

SBP.Name Property

Default property that returns the name of the SBP.



Restriction:

This information applies to only the BGA toolkit.

Usage

Name As String

Arguments

None

Description

This is a Microsoft-required property.

Examples

The following sample code retrieves the name of each SBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aSBP in comp.SBPs

MsgBox aSBP.Name

Next

End If

Next

End Sub
```

SBP.ObjectType Property

Returns the type of the object.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

This property always returns `ppcbObjectTypeSBP`.

This property is generally used to identify the kind of database objects in a heterogeneous [Objects](#) on page 477 collection or when implementing a generic routine that depends on the type of the database object passed as argument.

All database objects in the Automation server implement this property to compensate for the lack of an equivalent for the Visual C++ `QueryInterface` function.

Examples

See [Code Samples](#) for more information on running this sample.

```
Sub DoSomethingToDieObject(dbObject As Object)

Select Case dbObject.ObjectType

Case ppcbObjectTypeCBP

' Do something specific to CBP objects

Case ppcbObjectTypeSBP

' Do something specific to SBP objects

Case ppcbObjectTypeWirebond

' Do something specific to Wirebond objects

Case Else

MsgBox "Not a Die object"

End Select

End Sub
```

SBP.Orientation Property

Returns SBP orientation, in degrees.



Restriction:

This information applies to only the BGA toolkit.

Usage

Orientation As Double

Arguments

None

Description

None

Examples

The following sample code retrieves the orientation of each SBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aSBP in comp.SBPs

MsgBox aSBP.Name & " Orientation: " & Format (aSBP.Orientation, "#.###")

Next

End If

Next

End Sub
```

SBP.Parent Property

Returns the parent of the object.



Restriction:

This information applies to only the BGA toolkit.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

This is a Microsoft-required property.

SBP.Position X Property

Returns the x-coordinate of the SBP.



Restriction:

This information applies to only the BGA toolkit.

Usage

PositionX(*[unit* As [PPcbUnit](#) = ppcbUnitCurrent]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the x-coordinate is returned.

Description

None

Examples

The following sample code retrieves the position of each SBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aSBP in comp.SBPs

MsgBox aSBP.Name & ": (" & aSBP.PositionX & ", " & aSBP.PositionY & ")"

Next

End If
```

Next

End Sub

SBP.PositionY Property

Returns the y-coordinate of the SBP.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

PositionY(*unit* As [PPcbUnit](#) = ppcbUnitCurrent) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the y-coordinate is returned.

Description

None

Examples

The following sample code retrieves the position of each SBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
For Each comp In ActiveDocument.Components
```

```
  If comp.IsDiePart Then
```

```
    For Each aSBP in comp.SBPs
```

```
      MsgBox aSBP.Name & ": (" & aSBP.PositionX & ", " & aSBP.PositionY & ")"
```

```
    Next
```

```
  End If
```

```
Next
```

End Sub

SBP.Shape Property

Returns the shape of the SBP.



Restriction:

This information applies to only the BGA toolkit.

Usage

Shape As [PPcbBondPadShape](#)

Arguments

None

Descriptions

None

Examples

The following sample code retrieves the shape of each SBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aSBP in comp.SBPs

Select Case aSBP.Shape

Case ppcbBondPadShapeUnknown

MsgBox aSBP.Name & " Shape: Unknown"

Case ppcbBondPadShapeRectangle

MsgBox aSBP.Name & " Shape: Rect"

Case ppcbBondPadShapeOval
```

```
MsgBox aSBP.Name & " Shape: Oval"
```

```
Case Else
```

```
MsgBox aSBP.Name & " Shape: Unknown"
```

```
End Select
```

```
Next
```

```
End If
```

```
Next
```

```
End Sub
```

SBP.Tier Property

Returns the LIQ tier in which the SBP is placed.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

Tier As String

Arguments

None

Description

None

Examples

The following sample code retrieves the tier for each SBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
For Each comp In ActiveDocument.Components
```

```
If comp.IsDiePart Then
```

```
For Each aSBP in comp.SBPs
```

```
MsgBox aSBP.Name & " Tier: " & aSBP.Tier
```

```
Next
```

```
End If
```

```
Next
```

```
End Sub
```

SBP.Width Property

Returns the width of the SBP.



Restriction:

This information applies to only the BGA toolkit.

Usage

Width([*unit* As [PPcbUnit](#) = ppcbUnitCurrent]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the width value is returned.

Description

None

Examples

The following sample code retrieves the width of each SBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
For Each comp In ActiveDocument.Components
```

```
If comp.IsDiePart Then
```

```
For Each aSBP in comp.SBPs
```

```
MsgBox aSBP.Name & " Width: " & Format (aSBP.Width (ppcbUnitInch),  
"#.###")
```

```
Next
```

```
End If
```

```
Next
```

```
End Sub
```

SBP.Wirebonds Property

Returns the collection of bond wires attached to the SBP object.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

Wirebonds As Objects

Arguments

None

Description

None

Examples

The following sample code retrieves the number of bond wires connected to each SBP in each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
For Each comp In ActiveDocument.Components
```

```
If comp.IsDiePart Then
```

```
For Each aSBP in comp.SBPs
```

```
MsgBox "Number of WBs connected to " & aSBP.Name & ":" &  
aSBP.Wirebonds.Count
```

Next

End If

Next

End Sub

The Text Object

The Text object represents free text, or text associated with a 2D line in the open design.

This object contains the following properties:

- Text.Application Property
- Text.Drawing Property
- Text.Height/Label.Height Property
- Text.HorzJustification/Label.HorzJustification Property
- Text.Layer/Label.Layer Property
- Text.LineWidth/Label.LineWidth Property
- Text.Mirror/Label.Mirror Property
- Text.Name Property
- Text.ObjectType Property
- Text.Orientation/Label.Orientation Property
- Text.Parent Property
- Text.PositionX/Label.PositionX Property
- Text.PositionY/Label.PositionY Property
- Text.Selected Property
- Text.Text Property
- Text.VertJustification/Label.VertJustification Property
- Text.Delete Method

Text.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This is a Microsoft-required property.

Text.Drawing Property

Returns a drawing with which the text is associated.

Usage

Drawing as Drawing

Arguments

None

Description

None

Examples

The following sample function displays the name of a drawing with which the text is associated. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeText, , TRUE)

For Each txt In selected

Dim drw as Drawing

Set drw = txt.Drawing

If Not drw Is Nothing Then

s="The " & txt.Name & " text"

MsgBox s & " is associated with " & drw.Name & " drawing"

End If

Next txt

End Sub
```

Text.Height/Label.Height Property

Returns or sets text or label height.

Usage

Height (*Unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the result is to be represented. This optional argument is ppcbUnitCurrent on page 719 by default.

Description

None

Examples

The following example shows the height of a selected text object, and then sets the height to 100 mils. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeText,, TRUE)

For Each text In selected

MsgBox "Selected text height: " & text.Height

text.Height(ppcbUnitMils) = 100

Next text

End Sub
```

The following example shows the height of selected label object, and then sets height to 100 mils. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,, TRUE)

For Each label In selected
```

```
MsgBox "Selected label height: " & label.Height
```

```
label.Height(ppcbUnitMils) = 100
```

```
Next label
```

```
End Sub
```

Text.HorzJustification/Label.HorzJustification Property

Returns or sets the horizontal justification type of the text or label object.

Usage

HorzJustification as [PPcbHorizontalJustification](#)

Arguments

None

Description

None

Examples

The following example shows the horizontal justification setting of the selected text object, and then sets the justification to center. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeText,, TRUE)

For Each text In selected

Select Case text.HorzJustification

Case ppcbJustifyLeft

MsgBox "Horizontal justification: Left"

Case pcbJustifyHCenter

MsgBox "Horizontal justification: Center"

Case pcbJustifyRight

MsgBox "Horizontal justification: Right"

End Select

text.HorzJustification = ppcbJustifyHCenter
```

```
Next text
```

```
End Sub
```

The following example shows the horizontal justification setting of the selected label object, and then sets the justification to center. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,, TRUE)
```

```
For Each label In selected
```

```
Select Case label.HorzJustification
```

```
Case ppcbJustifyLeft
```

```
MsgBox "Horizontal justification: Left"
```

```
Case ppcbJustifyHCenter
```

```
MsgBox "Horizontal justification: Center"
```

```
Case ppcbJustifyRight
```

```
MsgBox "Horizontal justification: Right"
```

```
End Select
```

```
label.HorzJustification = ppcbJustifyHCenter
```

```
Next label
```

```
End Sub
```

Text.Layer/Label.Layer Property

Returns or sets the layer number of the text or label.

Usage

Layer as Long

Arguments

None

Description

None

Examples

The following example shows the layer number of the selected text object, and then sets the layer to 1. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeText,, TRUE)

For Each text In selected

MsgBox "Selected text layer number: " & text.Layer

text.Layer = 1

Next text

End Sub
```

The following example shows the layer number of the selected label object, and then sets the layer to 1. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,, TRUE)

For Each label In selected

MsgBox "Selected label layer number: " & label.Layer
```



```
label.Layer = 1
```

```
Next label
```

```
End Sub
```

Text.LineWidth/Label.LineWidth Property

Returns or sets the line width of the text or label.

Usage

LineWidth (*Unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the result is to be represented. This optional argument is Constants ppcbUnitCurrent on page 719 by default.

Description

None

Examples

The following example shows the line width of the selected text objects, and then sets the width to 10 mils. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeText,, TRUE)

For Each text In selected

MsgBox "Selected text line width: " & text.LineWidth

text.LineWidth(ppcbUnitMils) = 10

Next text

End Sub
```

The following example shows the line width of the selected label objects, and then sets the width to 10 mils. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,, TRUE)

For Each label In selected
```

```
MsgBox "Selected label line width: " & label.LineWidth
```

```
label.LineWidth(ppcbUnitMils) = 10
```

```
Next label
```

```
End Sub
```

Text.Mirror/Label.Mirror Property

Returns or sets the mirrored state of the text or label.

Usage

Mirror(*Origin* as [PPcbOriginType](#)) as Boolean

Arguments

<i>origin</i>	Shows whether the mirror status is relative to the object's parent or to the design (optional). The default value is ppcbOriginTypeDesign on page 719.

Description

None

Examples

The following example shows whether the selected text object is mirrored, and then mirrors the text. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeText,, TRUE)

For Each text In selected

MsgBox "Is selected text mirrored? " & text.Mirror

text.Mirror = True

Next text

End Sub
```

The following example shows whether the selected label object is mirrored, and then mirrors the label. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,, TRUE)

For Each label In selected
```

```
MsgBox "Is selected label mirrored? " & label.Mirror
```

```
label.Mirror = True
```

```
Next label
```

```
End Sub
```

Text.Name Property

Default property that returns the name of the Text object.

Usage

Name as String

Arguments

None

Description

None

Examples

The following example displays a message box showing the name of the first free text in the active document. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set text = ActiveDocument.Texts(1)

MsgBox "The first free text's name is " & text.Name

End Sub
```

Text.ObjectType Property

Returns the type of object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

Examples

The following sample shows the number of selected texts. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set sel = ActiveDocument.GetObjects(, , TRUE)

n = 0

For Each obj In sel

If obj.ObjectType = ppcbObjectTypeText Then n = n + 1

Next obj

MsgBox n & " text(s) selected"

End Sub
```

Text.Orientation/Label.Orientation Property

Returns or sets the rotation angle of the text or label.

Usage

Orientation(*Origin* as [PPcbOriginType](#)) as Double

Arguments

<i>origin</i>	Shows whether the orientation value is relative to the object's parent or to the design (optional). The default value is ppcbOriginTypeDesign on page 719.

Description

None

Examples

The following example shows the orientation of the selected text object, and then sets the orientation to 90 degrees. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeText,, TRUE)

For Each text In selected

MsgBox "Selected text orientation: " & text.Orientation

text.Orientation = 90

Next text

End Sub
```

The following example shows the orientation of the selected label object, and then sets the orientation to 90 degrees. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,, TRUE)

For Each label In selected
```



```
MsgBox "Selected label orientation: " & label.Orientation
```

```
label.Orientation = 90
```

```
Next label
```

```
End Sub
```

Text.Parent Property

Returns the parent of this object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

This is a Microsoft-required property.

Examples

Not required.

Text.PositionX/Label.PositionX Property

Returns or sets the x-coordinate of the origin of the Text or Label object.

Usage

PositionX (*Unit* as [PPcbUnit](#), *Origin* as [PPcbOriginType](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the result is to be represented. This optional argument is ppcbUnitCurrent on page 719 by default.
<i>origin</i>	[Optional on page 732]Type of reference point from which the result is counted. The default value is ppcbOriginTypeDesign on page 719.

Description

None

Examples

The following example shows the position of a selected text object, and then sets the position to (200, 200) mil. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeText,, TRUE)

For Each text In selected

MsgBox "Selected text position: (" & text.PositionX & ", " & text.PositionX
& ")"

text.PositionX(ppcbUnitMils) = 200

text.PositionY(ppcbUnitMils) = 200

Next text

End Sub
```

The following example shows the position of a selected label object, and then sets the position to (200, 200) mil. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,, TRUE)
```

```
For Each label In selected
```

```
MsgBox "Selected label position: (" & label.PositionX & ", " &  
label.PositionX & ")"
```

```
label.PositionX(ppcbUnitMils) = 200
```

```
label.PositionY(ppcbUnitMils) = 200
```

```
Next label
```

```
End Sub
```

Text.PositionY/Label.PositionY Property

Returns or sets the y-coordinate of the origin of the text or label object.

Usage

PositionY(*Unit* as [PPcbUnit](#), *Origin* as [PPcbOriginType](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the result is to be represented. This optional argument is ppcbUnitCurrent on page 719 by default.
<i>origin</i>	[Optional on page 732]Type of reference point from which the result is counted. The default value is ppcbOriginTypeDesign on page 719.

Description

None

Examples

The following example shows the position of the selected text object, and then sets position to (200, 200) mil. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeText,, TRUE)

For Each text In selected

MsgBox "Selected text position: (" & text.PositionX & ", " & text.PositionX
& ")"

text.PositionX(ppcbUnitMils) = 200

text.PositionY(ppcbUnitMils) = 200

Next text

End Sub
```

The following example shows the position of the selected label object, and then sets position to (200, 200) mil. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,, TRUE)
```

```
For Each label In selected
```

```
MsgBox "Selected label position: (" & label.PositionX & ", " &  
label.PositionY & ")"
```

```
label.PositionX(ppcbUnitMils) = 200
```

```
label.PositionY(ppcbUnitMils) = 200
```

```
Next label
```

```
End Sub
```

Text.Selected Property

Sets or returns whether the text object is selected.

Usage

Selected as Boolean

Arguments

None

Description

None

Examples

The following sample displays a message indicating whether the first text is selected, then selects the text. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each txt In ActiveDocument.Texts

Msgbox "Is Text " & txt.Text & " selected ? " & txt.Selected

txt.Selected = True

Exit For

Next txt

End Sub
```

Text.Text Property

Returns or sets contents of the text object.

Usage

Text as String

Arguments

None

Description

None

Examples

This sample shows the text string in the selected text object and then sets the string to "Hello!" See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeText,, TRUE)

For Each text In selected

MsgBox "Selected text: " & text.Text

text.Text="Hello !"

Next text

End Sub
```


Text.VertJustification/Label.VertJustification Property

Returns or sets the vertical justification type of the text or label object.

Usage

VertJustification as [PPcbVerticalJustification](#)

Arguments

None

Description

None

Examples

The following example shows the vertical justification setting of the selected text object, then sets the justification to center. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeText,, TRUE)

For Each text In selected

Select Case text.VertJustification

Case ppcbJustifyBottom

MsgBox "Vertical justification: Bottom"

Case ppcbJustifyVCenter

MsgBox "Vertical justification: Center"

Case ppcbJustifyTop

MsgBox "Vertical justification: Top"

End Select

text.VertJustification = ppcbJustifyVCenter
```

```
Next Text
```

```
End Sub
```

The following example shows the vertical justification setting of the selected label object, then sets the justification to center. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set selected = ActiveDocument.GetObjects(ppcbObjectTypeLabel,, TRUE)
```

```
For Each label In selected
```

```
Select Case label.VertJustification
```

```
Case ppcbJustifyBottom
```

```
MsgBox "Vertical justification: Bottom"
```

```
Case ppcbJustifyVCenter
```

```
MsgBox "Vertical justification: Center"
```

```
Case ppcbJustifyTop
```

```
MsgBox "Vertical justification: Top"
```

```
End Select
```

```
label.VertJustification = ppcbJustifyVCenter
```

```
Next Label
```

```
End Sub
```

Text.Delete Method

Deletes the Text object.

Usage

Delete as Boolean

Arguments

None

Description

None

Examples

The following sample deletes the selected text. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set selected = ActiveDocument.GetObjects(ppcbObjectTypeText,, TRUE)

For Each text In selected

If text.Delete() = False Then

MsgBox "Cannot delete the text!"

Else

MsgBox "The text was deleted successfully!"

End If

Next text

End Sub
```

The ThermalPad Object

The ThermalPad object represents a thermal pad in the padstack definition.

This object contains the following properties:

- ThermalPad.Application Property
- ThermalPad.CornerRadius Property
- ThermalPad.CornerType Property
- ThermalPad.InnerLength Property
- ThermalPad.InnerSize Property
- ThermalPad.Name Property
- ThermalPad.ObjectType Property
- ThermalPad.Offset Property
- ThermalPad.Orientation Property
- ThermalPad.OuterSize Property
- ThermalPad.PadStackLayer Property
- ThermalPad.Parent Property
- ThermalPad.Shape Property
- ThermalPad.SpokeAngle Property
- ThermalPad.Spokes Property
- ThermalPad.SpokeWidth Property

ThermalPad.Application Property

Returns the application object.

Usage

Application as Application

Arguments

None

ThermalPad.CornerRadius Property

Returns the Thermal Pad corner radius.

Usage

```
CornerRadius as Double
```

```
CornerRadius (unit as PPcbUnit) as Double
```

Arguments

<i>unit</i> - [Optional]	Unit in which the value is returned.

ThermalPad.CornerType Property

Returns the Thermal Pad corner type.

Usage

```
CornerType as PPcbPadCornerType
```

Arguments

None

ThermalPad.InnerLength Property

Returns the Thermal Pad inner length.

Usage

```
InnerLength as Double
```

```
InnerLength (unit as PPcbUnit) as Double
```

Arguments

<i>unit</i> - [Optional]	Unit in which the value is returned.

ThermalPad.InnerSize Property

Returns the thermal pad's inner size. For shape `ppcbThermalPadShapeRound` it returns inner diameter.

Usage

```
InnerSize (unit as PPcbUnit) as Double
```

Arguments

<i>unit</i> - [Optional]	Unit in which the value is returned. This optional argument is <code>ppcbUnitCurrent</code> by default.

ThermalPad.Name Property

Returns the name of this thermal pad.

Usage

```
Name as String
```

Arguments

None

ThermalPad.ObjectType Property

Returns the type of the object - ppcbObjectTypeThermalPad.

Usage

```
ObjectType as PPcbObjectType
```

Arguments

None

ThermalPad.Offset Property

Returns the Thermal Pad offset.

Usage

```
offset as Double
```

```
offset (unit as PPcbUnit) as Double
```

Arguments

<i>unit</i> - [Optional]	Unit in which the value is returned.

ThermalPad.Orientation Property

Returns the Thermal Pad orientation.

Usage

```
orientation as Double
```

Arguments

None

ThermalPad.OuterSize Property

Returns the thermal pad's outer size. For shape `ppcbThermalPadShapeRound` it returns outer diameter.

Usage

```
outerSize (unit as PPcbUnit) as Double
```

Arguments

<i>unit</i> - [Optional]	Unit in which the value is returned. This optional argument is <code>ppcbUnitCurrent</code> by default.

ThermalPad.PadStackLayer Property

Returns the PadStackLayer Object to which this thermal pad belongs to.

Usage

```
PadStackLayer as PadStackLayer
```

Arguments

None

ThermalPad.Parent Property

Returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

ThermalPad.Shape Property

Returns the thermal pad shape.

Usage

```
Shape as PPcbThermalPadShape
```

Arguments

None

ThermalPad.SpokeAngle Property

Returns the thermal pad's spoke angle.

Usage

```
SpokeAngle as Double
```

Arguments

None

ThermalPad.Spokes Property

Returns the thermal pad's number of spokes.

Usage

```
Spokes as Integer
```

Arguments

None

ThermalPad.SpokeWidth Property

Returns the thermal pad's spoke width.

Usage

```
SpokeWitdth (unit as PPcbUnit) as Double
```

Arguments

<i>unit</i> - [Optional]	Unit in which the value is returned. This optional argument is ppcbUnitCurrent by default.

The Via Object

The Via object represents a physical via in the open design.

This object contains the following properties:

- Via.Application Property
- Via.Attributes Property
- Via.DrillSize Property
- Via.EndLayer Property
- Via.Glued Property
- Via.Highlighted Property
- Via.Name Property
- Via.Net Property
- Via.ObjectType Property
- Via.PadStackLayers Property
- Via.Parent Property
- Via.PlaneThermal Property
- Via.Plated Property
- Via.PositionX Property
- Via.PositionY Property
- Via.Selected Property
- Via.StartLayer Property
- Via.Stitching Property
- Via.TestPoint Property
- Via.Type Property

Via.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This property identifies the object as an Automation object. All Automation server applications have an application object and all Automation objects have an application property. This property is usually used in Automation client applications that handle large volumes of objects from different sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

Via.Attributes Property

Returns the collection of all attributes of the via.

Usage

Attributes As [Attributes](#) on page 130

Attributes(*name* As String) As [Attribute](#) on page 120

Arguments

<i>name</i>	Name of an existing via attribute.

Description

When an existing attribute name is passed to this property, it returns that via [Attribute](#) on page 120 object. If the attribute name does not exist, this property returns the collection of all via attributes in an [Attributes](#) on page 130 collection object.

Examples

The following sample code retrieves the number of attributes assigned to the first via found in the open design, assuming at least one via exists, using the

Via.DrillSize Property

Returns the drill size of the via.

Usage

DrillSize(*unit* As [PPcbUnit](#)) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the drill size value is returned.

Description

None

Examples

The following sample code retrieves the drill size of the first via found in the open design, assuming at least one via exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set firstVia = ActiveDocument.Vias(1)
```

```
MsgBox firstVia.Name & " has a drill of " & firstVia.DrillSize
```

```
End Sub
```


Via.EndLayer Property

Returns the end layer of the via.

Usage

EndLayer As Long

Arguments

None

Description

None

Examples

The following sample code retrieves the first/end layer of the first via found in the open design, assuming at least one via exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set firstVia = ActiveDocument.Vias(1)
```

```
MsgBox firstVia.Name & " starts on layer " & firstVia.StartLayer & " and  
ends on layer " & firstVia.EndLayer
```

```
End Sub
```

Via.Glued Property

Sets or returns whether the via is glued.

Usage

Glued As Boolean

Arguments

None

Description

None

Examples

The following sample code glues all vias in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main  
  
For Each nextVia In ActiveDocument.Vias  
  
nextVia.Glued = True  
  
Next nextVia  
  
End Sub
```

Via.Highlighted Property

Returns whether the via is highlighted.

Usage

Highlighted as Boolean

Arguments

None

Via.Name Property

Returns the name of the via.

Usage

Name As String

Arguments

None

Description

This property is the default property for the Via object.

Examples

The following sample code lists all vias in the open design by name and then places that list in a custom dialog box. When a via is selected in the list box, the sample selects that via in SailWind Layout.

This sample uses the User Dialog Editor in the Sax Basic Engine in SailWind Layout. See [Code Samples](#) for more information on running this sample.

For more information, see *Sax Basic Editor On Line Help* (C:\<install_folder>\<version>\Programs\sbe5_000.hlp)

```
Dim ListVias$(10000)

Sub Main

    index = 0

    For Each nextVia In ActiveDocument.Vias

        ListVias$(index) = nextVia.Name

        index = index + 1

    Next nextVia

    ' This piece of code is automatically generated by the Basic Dialog Editor
    ' in SailWind Layout.

    Begin Dialog UserDialog 180,238,"Vias",.CallbackFunc ' %GRID:10,7,1,1

        ListBox 10,7,160,203,ListVias(),.ListBox1
```

```
OKButton 10,210,160,21
```

```
End Dialog
```

```
Dim dlg As UserDialog
```

```
Dialog dlg
```

```
End Sub
```

```
' The following function is automatically called by the system when  
something has happened
```

```
' in the dialog; it is used to easily process user actions.
```

```
Function CallbackFunc%(DlgItem$, Action%, SuppValue%)
```

```
Select Case Action%
```

```
Case 2 ' Value changing or button pressed
```

```
If DlgItem$="ListBox1" Then
```

```
ActiveDocument.SelectObjects(ppcbObjectTypeAll, , False)
```

```
ActiveDocument.SelectObjects(ppcbObjectTypeVia, ListVias(SuppValue%))
```

```
End If
```

```
End Select
```

```
End Function
```

Via.Net Property

Returns the net attached to the via.

Usage

Net As [Net](#) on page 452

Arguments

None

Description

None

Examples

The following sample code retrieves the net connected to the first via found in the open design, assuming at least one via exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set firstVia = ActiveDocument.Vias(1)

MsgBox firstVia.Name & " is connected to net " & firstVia.Net.Name

End Sub
```

Via.ObjectType Property

Returns the type of the object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

This property returns [ppcbObjectTypeVia](#) on page 719.

All SailWind Layout database objects in the SailWind Layout Automation server implement this property to compensate for the lack of a Visual Basic equivalent for the Visual C++ QueryInterface function.

This property is generally used:

- To identify the kind of SailWind Layout database objects in a heterogeneous [Objects](#) on page 477 collection.
- When implementing a generic routine that depends on the type of the SailWind Layout database object passed as argument. For example:

```
Sub DoSomething(dbObject As Object)

Select Case dbObject.ObjectType

Case ppcbObjectTypeComponent

' Do something specific to component objects

Case ppcbObjectTypeNet

' Do something specific to net objects

Case ppcbObjectTypePin

' Do something specific to pin objects

Case ppcbObjectTypeVia

' Do something specific to via objects
```

```
Case ppcbObjectTypeConnection  
  
    ' Do something specific to connection objects  
  
Case ppcbObjectTypeRouteSegment  
  
    ' Do something specific to route segment objects  
  
Case ppcbObjectTypeJumper  
  
    ' Do something specific to jumper objects  
  
Case Else  
  
    MsgBox "Not a SailWind Layout database object"  
  
End Select  
  
End Sub
```


Via.PadStackLayers Property

Returns the collection of all padstack layers for this via.

Usage

PadStackLayers as Objects

PadStackLayers(*layerName* as String) as PadStackLayer

Arguments

<i>layerName</i>	Name of padstack layer

Return Values

When a layer name is passed to this property, it returns that PadStackLayer Object. If the name is not specified, this property returns the collection of all padstack layers in an Objects collection object.

Examples

```
For Each via In Application.ActiveDocument.Vias
```

```
    For Each layer in via.PadStackLayers
```

```
        MsgBox layer.Number & ", " & layer.Name
```

```
        pad = layer.Pad
```

```
        MsgBox pad.Name & ", " & pad.Shape & ", " & pad.Diameter & ", "
```

```
            & pad.InnerDiameter & ", " & pad.Width & ", "
```

```
            & pad.Length & ", " & pad.offset & ", "
```

```
            & pad.Orientation & ", " & pad.CornerType & ", "
```

```
            & pad.CornerRadius
```

```
thermalpad = layer.ThermalPad
```

```
If thermalpad Is Nothing Then
```

```
    MsgBox "Thermal pad not defined on this layer"
```

```
Else
```

```
    MsgBox thermalpad.Name & ", " & thermalpad.Shape & ", "
```

```
        & thermalpad.InnerSize & ", "
```

```
        & thermalpad.OuterSize & ", " & thermalpad.Spokes & ", "
```

```
        & thermalpad.SpokeAngle & ", " & thermalpad.SpokeWidth
```

```
End If
```

```
antipad = layer.AntiPad
```

```
If antipad Is Nothing Then
```

```
    MsgBox "Anti pad not defined on this layer"
```

```
Else
```

```
    MsgBox antipad.Name & ", " & antipad.Shape & ", "
```

```
        & antipad.Size
```

```
End If
```

```
Next layer
```

```
Next via
```

Via.Parent Property

Returns the parent of the object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

None

Via.PlaneThermal Property

Returns whether the via has a plane thermal.

Usage

PlaneThermal As Boolean

Arguments

None

Description

None

Examples

The following sample code retrieves the number of plane thermal vias in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

nbPTVias = 0

For Each nextVia In ActiveDocument.Vias

If nextVia.PlaneThermal = True Then nbPTVias = nbPTVias +1

Next nextVia

MsgBox "There are " & nbPTVias & " plane thermal vias (out of " &
ActiveDocument.Vias.Count & ") in " & ActiveDocument.Name

End Sub
```

Via.Plated Property

Returns whether the via is plated.

Usage

Plated As Boolean

Arguments

None

Description

None

Examples

The following sample code retrieves the number of nonplated vias in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

nbPlatedVias = 0

For Each nextVia In ActiveDocument.Vias

If nextVia.Plated = False Then nbPlatedVias = nbPlatedVias +1

Next nextVia

MsgBox "There are " & nbPlatedVias & " non-plated pins (out of " &
ActiveDocument.Vias.Count & ") in " & ActiveDocument.Name

End Sub
```

Via.PositionX Property

Returns the x-coordinate of the via.

Usage

PositionX(*[unit As PPcbUnit]*) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the x-coordinate is returned.

Description

None

Examples

The following sample code retrieves the location of the first via found in the open design, assuming at least one via exists, in current design units. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set firstVia = ActiveDocument.Vias(1)
```

```
MsgBox firstVia.Name & " position is (" & firstVia.PositionX & ", " &  
firstVia.PositionY & ")"
```

```
End Sub
```

Via.PositionY Property

Returns the y-coordinate of the via.

Usage

PositionY(*[unit As PPcbUnit]*) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the y-coordinate is returned.

Description

None

Examples

The following sample code retrieves the location of the first via found in the open design, assuming at least one via exists, in current design units. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set firstVia = ActiveDocument.Vias(1)
```

```
MsgBox firstVia.Name & " position is (" & firstVia.PositionX & ", " &  
firstVia.PositionY & ")"
```

```
End Sub
```

Via.Selected Property

Sets or returns whether the via is selected.

Usage

Selected As Boolean

Arguments

None

Description

None

Examples

The following sample code selects the first via found in the open design, assuming at least one via exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main

ActiveDocument.SelectObjects(,False)

ActiveDocument.Vias(1).Selected = True

End Sub
```


Via.StartLayer Property

Returns the start layer of the via.

Usage

StartLayer As Long

Arguments

None

Description

None

Examples

The following sample code retrieves the first/end layer of the first via found in the open design, assuming at least one via exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set firstVia = ActiveDocument.Vias(1)
```

```
MsgBox firstVia.Name & " starts on layer " & firstVia.StartLayer & " and  
ends on layer " & firstVia.EndLayer
```

```
End Sub
```

Via.Stitching Property

Sets or returns the via's stitching status.

Usage

Stitching As Boolean

Arguments

None

Description

None

Examples

The following sample code selects all stitching vias in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
  For Each nextVia In ActiveDocument.Vias
    If nextVia.Stitching Then
      nextVia.Selected = True
    End If
  Next nextVia
End Sub
```

Via.TestPoint Property

Sets or returns whether the via is a test point.

Usage

TestPoint As [PPcbTestPointType](#)

Arguments

None

Description

None

Examples

The following sample code removes all via test points from the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main  
  
For Each nextVia In ActiveDocument.Vias  
  
nextVia.TestPoint = ppcbTestPointNone  
  
Next nextVia  
  
End Sub
```

Via.Type Property

Returns the type of the via.

Usage

Type As String

Arguments

None

Description

None

Examples

The following sample code retrieves the type of the first via found in the open design, assuming at least one via exists. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set firstVia = ActiveDocument.Vias(1)

MsgBox firstVia.Name & " type is " & firstVia.Type

End Sub
```

The View Object

The View object represents the current view window, showing the open design.

This object is usually retrieved using the [Document.ActiveView Property](#) property.

The View object contains the following properties:

- [View.Application Property](#)
- [View.BottomRightX Property](#)
- [View.BottomRightY Property](#)
- [View.CenterX Property](#)
- [View.CenterY Property](#)
- [View.Name Property](#)
- [View.ObjectType Property](#)
- [View.Parent Property](#)
- [View.PointerX Property](#)
- [View.PointerY Property](#)
- [View.TopLeftX Property](#)
- [View.TopLeftY Property](#)
- [View.Zoom Property](#)
- [View.Pan Method](#)
- [View.Refresh Method](#)
- [View.SetExtents Method](#)
- [View.SetExtentsToAll Method](#)
- [View.SetExtentsToBoard Method](#)
- [View.SetExtentsToSelection Method](#)
- [View.SetScale Method](#)
- [View.Change Event](#)

View.Application Property

Returns the Application object.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This property identifies the object as an Automation object. All Automation server applications have an application object and all Automation objects have an application property. This property is usually used in Automation client applications that handle large volumes of objects from different sources, such as different Automation server applications), to quickly identify the application to which the object belongs.

View.BottomRightX Property

Returns the x-coordinate of the lower right corner of the view.

Usage

BottomRightX(*[unit As PPcbUnit]*) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the lower right x-coordinate is returned.

Description

None

Examples

The following sample code retrieves the coordinates of the current view in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

x0 = Format$(ActiveDocument.ActiveView.TopLeftX, "Fixed")

y0 = Format$(ActiveDocument.ActiveView.TopLeftY, "Fixed")

x1 = Format$(ActiveDocument.ActiveView.BottomRightX, "Fixed")

y1 = Format$(ActiveDocument.ActiveView.BottomRightY, "Fixed")

MsgBox "View is (" & x0 & ", " & y0 & ") - (" & x1 & ", " & y1 & ")

End Sub
```

View.BottomRightY Property

Returns the y-coordinate of the lower right corner of the view.

Usage

BottomRightY(*[unit As PPcbUnit]*) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the lower right y-coordinate is returned.

Description

None

Examples

The following sample code retrieves the coordinates of the current view in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

x0 = Format$(ActiveDocument.ActiveView.TopLeftX, "Fixed")

y0 = Format$(ActiveDocument.ActiveView.TopLeftY, "Fixed")

x1 = Format$(ActiveDocument.ActiveView.BottomRightX, "Fixed")

y1 = Format$(ActiveDocument.ActiveView.BottomRightY, "Fixed")

MsgBox "View is (" & x0 & ", " & y0 & ") - (" & x1 & ", " & y1 & ")

End Sub
```


View.CenterX Property

Returns the x-coordinate of center of the view.

Usage

CenterX (*Unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the result is to be represented. This optional argument is ppcbUnitCurrent on page 719 by default.

Description

None

Examples

The following sample code displays the current view's center and zoom level. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set v = ActiveDocument.ActiveView
```

```
Dim msg as String
```

```
msg="View parameters:" & Chr(13) & Chr(10)
```

```
msg = msg & "Center: (" & v.CenterX & ", " & v.CenterY & ")" & Chr(13) &  
Chr(10)
```

```
MsgBox msg & "Zoom: " & v.Zoom
```

```
End Sub
```

View.CenterY Property

Returns the y-coordinate of center of the view.

Usage

CenterY(*Unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the result is to be represented. This optional argument is ppcbUnitCurrent on page 719 by default.

Description

None

Examples

The following sample code displays the current view's center and zoom level. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set v = ActiveDocument.ActiveView
```

```
Msg="View parameters:" & Chr(13) & Chr(10)
```

```
Msg = Msg & "Center: (" & v.CenterX & ", " & v.CenterY & ")" & Chr(13) &  
Chr(10)
```

```
MsgBox msg & "Zoom: " & v.Zoom
```

```
End Sub
```

View.Name Property

Returns the name of the view. For example, in SailWind Layout, this function returns the string “Current View.”

Usage

Name As String

Arguments

None

Description

This property is the default property for the view object.

Examples

The following sample code retrieves the name of the current view. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
MsgBox ActiveDocument.ActiveView.Name
```

```
End Sub
```

View.ObjectType Property

Returns the type of this object.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

None

View.Parent Property

Returns the parent of the object.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

None

View.PointerX Property

Returns the pointer's X position.

Usage

PointerX (*unit* as [PPcbUnit](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the value is returned. This optional argument is <code>ppcbUnitCurrent</code> by default.

Examples

```
' load preview.pcb
```

```
Application.ModelessCommand("s")
```

```
DlgModelessCmd.Command="s C8"
```

```
DlgModelessCmd.OnOk()
```

```
doc = Application.ActiveDocument
```

```
view = doc.ActiveView
```

```
MsgBox view.PointerX & ", " & view.PointerY
```

View.PointerY Property

Returns the pointer's Y position.

Usage

PointerY (unit as [PPcbUnit](#)) as Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the value is returned. This optional argument is <code>ppcbUnitCurrent</code> by default.

Examples

```
' load preview.pcb
```

```
Application.ModelessCommand("s")
```

```
DlgModelessCmd.Command="s C8"
```

```
DlgModelessCmd.OnOk()
```

```
doc = Application.ActiveDocument
```

```
view = doc.ActiveView
```

```
MsgBox view.PointerX & ", " & view.PointerY
```

View.TopLeftX Property

Returns the x-coordinate of the upper left corner of the view.

Usage

TopLeftX(*[unit* As [PPcbUnit](#)]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the upper left x-coordinate is returned.

Description

None

Examples

The following sample code retrieves the coordinates of the current view in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

x0 = Format$(ActiveDocument.ActiveView.TopLeftX, "Fixed")

y0 = Format$(ActiveDocument.ActiveView.TopLeftY, "Fixed")

x1 = Format$(ActiveDocument.ActiveView.BottomRightX, "Fixed")

y1 = Format$(ActiveDocument.ActiveView.BottomRightY, "Fixed")

MsgBox "View is (" & x0 & ", " & y0 & ") - (" & x1 & ", " & y1 & ")

End Sub
```


View.TopLeftY Property

Returns the y-coordinate of the upper left corner of the view.

Usage

TopLeftY(*unit* As [PPcbUnit](#)) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the upper left y-coordinate is returned.

Description

None

Examples

The following sample code retrieves the coordinates of the current view in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

x0 = Format$(ActiveDocument.ActiveView.TopLeftX, "Fixed")

y0 = Format$(ActiveDocument.ActiveView.TopLeftY, "Fixed")

x1 = Format$(ActiveDocument.ActiveView.BottomRightX, "Fixed")

y1 = Format$(ActiveDocument.ActiveView.BottomRightY, "Fixed")

MsgBox "View is (" & x0 & ", " & y0 & ") - (" & x1 & ", " & y1 & ")

End Sub
```

View.Zoom Property

Returns the current zoom factor.

Usage

Zoom as Double

Arguments

None

Description

The zoom factor is 1 when the view is zoomed out as far as possible. When zooming in, the zoom factor grows in proportion to the size of object.

Examples

The following sample code displays the current view's center and zoom factor. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set v = ActiveDocument.ActiveView

msg="View parameters:" & Chr(13) & Chr(10)

msg = msg & "Center: (" & v.CenterX & ", " & v.CenterY & ")" & Chr(13) &
Chr(10)

MsgBox msg & "Zoom: " & v.Zoom

End Sub
```

View.Pan Method

Pans the view to a specified location.

Usage

Pan(*x* As Double, *y* As Double, [*unit* As [PPcbUnit](#)])

Arguments

<i>x</i>	x-coordinate of the point to which to pan.
<i>y</i>	y-coordinate of the point to which to pan.
<i>unit</i>	[Optional on page 732] Unit in which the X and Y values are given.

Return Values

None

Description

None

Examples

The following sample code centers the view to the location of component U1, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

x = ActiveDocument.Components("U1").PositionX

y = ActiveDocument.Components("U1").PositionY

ActiveDocument.ActiveView.Pan(x,y)

End Sub
```

View.Refresh Method

Refreshes the view.

Usage

Refresh

Arguments

None

Return Values

None

Description

None

Examples

The following sample code redraws the view. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
    ActiveDocument.ActiveView.Refresh
```

```
End Sub
```

View.SetExtents Method

Sets the view extents.

Usage

SetExtents(*tlx* As Double, *tly* As Double, *brx* As Double, *bry* As Double, [*unit* As [PPcbUnit](#)])

Arguments

<i>tlx</i>	x-coordinate of the top left corner of the new view.
<i>tly</i>	y-coordinate of the top left corner of the new view.
<i>brx</i>	x-coordinate of the bottom right corner of the new view.
<i>bry</i>	y-coordinate of the bottom right corner of the new view.
<i>unit</i>	[Optional on page 732] Unit in which the <i>tlx</i> , <i>tly</i> , <i>brx</i> , and <i>bry</i> values are given.

Return Values

None

Description

None

Examples

The following sample code sets the view to the extents of all pins connected to net VCC, assuming it exists in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
xMin = 1000000000.0
```

```
yMin = 1000000000.0
```

```
xMax = -1000000000.0
```

```
yMax = -1000000000.0
```

```
For Each nextPin In ActiveDocument.Nets("VCC").Pins
```

```
If nextPin.PositionX < xMin Then xMin = nextPin.PositionX
```

```
If nextPin.PositionX > xMax Then xMax = nextPin.PositionX
```

```
If nextPin.PositionY < yMin Then yMin = nextPin.PositionY
```

```
If nextPin.PositionY > yMax Then yMax = nextPin.PositionY
```

```
Next nextPin
```

```
ActiveDocument.ActiveView.SetExtents(xMin, yMin, xMax, yMax)
```

```
End Sub
```

View.SetExtentsToAll Method

Sets the view extents to all objects in the design.

Usage

```
SetExtentsToAll()
```

Arguments

None

Return Values

None

Description

None

Examples

The following sample code sets the view extents to all objects in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main  
  
ActiveDocument.ActiveView.SetExtentsToAll  
  
End Sub
```

View.SetExtentsToBoard Method

Sets the view extents to the design board extents.

Usage

```
SetExtentsToBoard()
```

Arguments

None

Return Values

None

Description

None

Examples

The following sample code sets the view extents to the board of the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.ActiveView.SetExtentsToBoard
```

```
End Sub
```


View.SetExtentsToSelection Method

Sets the current zoom level and view center to show all currently selected objects.

Usage

SetExtentsToSelection

Arguments

None

Description

None

Examples

The following sample code sets view parameters so that C1 and C2 components are shown. See [Code Samples](#) for more information on running this sample.

```
Sub Main

Set v = ActiveDocument.ActiveView

ActiveDocument.SelectObjects ppcbObjectTypeAll, "*", False

ActiveDocument.Components("C1").Selected = True

ActiveDocument.Components("C2").Selected = True

v.SetExtentsToSelection

End Sub
```

View.SetScale Method

Sets the current zoom level and view center.

Usage

SetScale (*Zoom* as Double, *CenterX* as Double, *CenterY* as Double, *Units* as [PPcbUnit](#))

Arguments

<i>zoom</i>	A new zoom level. Required.
<i>centerX</i>	x-coordinate of new view center. Optional, if omitted, current Center x-coordinate is used.
<i>centerY</i>	y-coordinate of new view center. Optional, if omitted, current Center y-coordinate is used.
<i>units</i>	[Optional on page 732] Unit in which the result is to be represented. This optional argument is ppcbUnitCurrent on page 719 by default.

Return Values

None

Description

None

Examples

The following sample code zooms in the view. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
Set v = ActiveDocument.ActiveView
```

```
v.SetScale( v.Zoom * 2 )
```

```
End Sub
```

View.Change Event

Occurs when the current view changes.

Usage

View_Change()

Arguments

None

Description

None

The Wirebond Object

The Wirebond object represents the physical bond wire of a die part.

You can retrieve the Wirebond object, or collection of Wirebond objects, only from the [Component](#) on page 178 object, which represents a die part ([Component.IsDiePart Property](#)), or from other die part constituents (such as [CBP](#) on page 147 or [SBP](#) on page 597).

Wirebond objects are not selectable.

You cannot retrieve the collection of all Wirebond objects from a document using Document.GetObjects (ppcbObjectTypeWirebond). This method returns an empty collection. Also, Document.GetObjects (ppcbObjectTypeAll) does not return CBP, SBP, or Wirebond objects. You can get only collections of CBPs, SBPs, or Wirebonds using the Component object methods, if the component is a die part ([Component.IsDiePart Property](#)).

The Wirebond object contains the following properties:

- [Wirebond.Angle Property](#)
- [Wirebond.Application Property](#)
- [Wirebond.Component Property](#)
- [Wirebond.EndOffsetX Property](#)
- [Wirebond.EndOffsetY Property](#)
- [Wirebond.EndPad Property](#)
- [Wirebond.EndX Property](#)
- [Wirebond.EndY Property](#)
- [Wirebond.Name Property](#)
- [Wirebond.ObjectType Property](#)
- [Wirebond.Parent Property](#)
- [Wirebond.StartOffsetX Property](#)
- [Wirebond.StartOffsetY Property](#)
- [Wirebond.StartPad Property](#)
- [Wirebond.StartX Property](#)
- [Wirebond.StartY Property](#)

Wirebond.Angle Property

Returns the angle between the wire bond and the edge of the die, in degrees.



Restriction:

This information applies to only the BGA toolkit.

Usage

Angle As Double

Arguments

None

Description

None

Examples

The following sample code retrieves the angle between the bond wire and the edge of the die for each wire bond in the die, in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aWB in comp.Wirebonds

MsgBox aWB.Name & ": Angle: " & Format (aWB.Angle, "#.###")

Next

End If

Next

End Sub
```

Wirebond.Application Property

Returns the Application object.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

Application As [Application](#) on page 68

Arguments

None

Description

This property identifies the object as an Automation object. All Automation server applications have an Application object and all Automation objects have an Application property. This property is usually used

in Automation client applications that handle large volumes of objects from different sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

This is a Microsoft-required property.

Wirebond.Component Property

Returns the component object of the wire bond.



Restriction:

This information applies to only the BGA toolkit.

Usage

Component As Component

Arguments

None

Description

None

Examples

The following sample code represents a subroutine that depends on the Wirebond object passed as an argument. The code retrieves the name of the component and the name of the decal to which the wire bond belongs. See [Code Samples](#) for more information on running this sample.

```
Sub QueryWB(aWB As Wirebond)

MsgBox "Bond Wire" & aWB.Name &

"belongs to component" & aWP.Component.Name &

 "(" & aWP.Component.Decal & ")"

End Sub
```

Wirebond.EndOffsetX Property

Returns the x offset of the endpoint of the bond wire from the center of the end pad.



Restriction:

This information applies to only the BGA toolkit.

Usage

EndOffsetX([*unit* As [PPcbUnit](#) = ppcbUnitCurrent]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the X offset value is returned.

Description

None

Examples

The following sample code retrieves the end offset of each bond wire for each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aWB in comp.Wirebonds

MsgBox aWB.Name & ": (" & aWB.EndOffsetX & ", " & aWB.EndOffsetY & ")"

Next

End If

Next

End Sub
```

Wirebond.EndOffsetY Property

Returns the y offset of the endpoint of the bond wire from the center of the end pad.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

EndOffsetY([*unit* As [PPcbUnit](#) = ppcbUnitCurrent]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the Y offset value is returned.

Description

None

Examples

The following sample code retrieves the end offset of each bond wire for each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aWB in comp.Wirebonds

MsgBox aWB.Name & ": (" & aWB.EndOffsetX & ", " & aWB.EndOffsetY & ")"

Next

End If

Next

End Sub
```

Wirebond.EndPad Property

Returns the end pad to which the bond wire is attached.



Restriction:

This information applies to only the BGA toolkit.

Usage

EndPad As Object

Arguments

None

Description

This property can return either the [CBP](#) on page 147 or [SBP](#) on page 597 object.

Examples

The following sample code retrieves the names of the end pads of each bond wire for each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aWB in comp.Wirebonds

MsgBox aWB.StartPad.Name & " - " & aWB.EndPad.Name

Next

End If

Next

End Sub
```

Wirebond.EndX Property

Returns the x-coordinate for the end of the bond wire.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

EndX([unit As [PPcbUnit](#) = ppcbUnitCurrent]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the x-coordinate value is returned.

Description

None

Examples

The following sample code retrieves the end position of each bond wire for each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aWB in comp.Wirebonds

MsgBox aWB.Name & " End: (" & aWB.EndX & ", " & aWB.EndY & ")"

Next

End If

Next

End Sub
```

Wirebond.EndY Property

Returns the y-coordinate for the end of the bond wire.



Restriction:
This information applies to only the BGA toolkit.

Usage

EndY(*[unit As PPcbUnit = ppcbUnitCurrent]*) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the y-coordinate value is returned.

Description

None

Examples

The following sample code retrieves the end position of each bond wire for each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aWB in comp.Wirebonds

MsgBox aWB.Name & " End: (" & aWB.EndX & ", " & aWB.EndY & ")"

Next

End If

Next

End Sub
```

Wirebond.Name Property

Default property that returns the name of the bond wire.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

Name As String

Arguments

None

Description

This is a Microsoft-required property.

Examples

The following sample code retrieves the name of each SBP and CBP to which the bond wire is attached. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aWB in comp.Wirebonds

MsgBox aWB.Name

Next

End If

Next

End Sub
```

Wirebond.ObjectType Property

Returns the object type of the bond wire.



Restriction:

This information applies to only the BGA toolkit.

Usage

ObjectType As [PPcbObjectType](#)

Arguments

None

Description

This property returns [PPcbObjectType](#)Wirebond.

This property is generally used to identify the kind of database objects in a heterogeneous [Objects](#) on page 477 collection or when implementing a generic routine that depends on the type of the database object passed as argument.

All database objects in the Automation server implement this property to compensate for the lack of an equivalent for the Visual C++ QueryInterface function.

Examples

See [Code Samples](#) for more information on running this sample.

```
Sub DoSomethingToDieObject(dbObject As Object)

Select Case dbObject.ObjectType

Case ppcbObjectTypeCBP

' Do something specific to CBP objects

Case ppcbObjectTypeSBP

' Do something specific to SBP objects

Case ppcbObjectTypeWirebond

' Do something specific to Wirebond objects

Case Else

MsgBox "Not a Die object"

End Select

End Sub
```

Wirebond.Parent Property

Returns the parent object of the bond wire.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

Parent As [Document](#) on page 250

Arguments

None

Description

This is a Microsoft-required property.

Wirebond.StartOffsetX Property

Returns the x offset of the start point of the bond wire from the center of the start pad.



Restriction:

This information applies to only the BGA toolkit.

Usage

StartOffsetX(*[unit As [PPcbUnit](#) = ppcbUnitCurrent]*) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the x offset value is returned.

Description

None

Examples

The following sample code retrieves the start offset of each bond wire for each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aWB in comp.Wirebonds

MsgBox aWB.Name & ": (" & aWB.StartOffsetX & ", " & aWB.StartOffsetY & ")"
```

```
Next
```

```
End If
```

```
Next
```

```
End Sub
```

Wirebond.StartOffsetY Property

Returns the y offset of the start point of the bond wire from the center of the start pad.



Restriction:

This information applies to only the BGA toolkit.

Usage

StartOffsetY(*[unit As PPcbUnit = ppcbUnitCurrent]*) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the y offset value is returned.

Description

None

Examples

The following sample code retrieves the start offset of each bond wire for each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
For Each comp In ActiveDocument.Components
```

```
    If comp.IsDiePart Then
```

```
        For Each aWB in comp.Wirebonds
```

```
            MsgBox aWB.Name & ": (" & aWB.StartOffsetX & ", " & aWB.StartOffsetY & ")"
```

```
        Next
```

```
End If
```

```
Next
```

```
End Sub
```

Wirebond.StartPad Property

Returns the start pad to which the bond wire is attached.



Restriction:

This information applies to only the BGA toolkit.

Usage

StartPad As Object

Arguments

None

Description

This property can return either the [CBP](#) on page 147 or the [SBP](#) on page 597 object.

Examples

The following sample code retrieves the names of the end pads of each bond wire for each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main
```

```
For Each comp In ActiveDocument.Components
```

```
If comp.IsDiePart Then
```

```
For Each aWB in comp.Wirebonds
```

```
MsgBox aWB.StartPad.Name & " - " & aWB.EndPad.Name
```

```
Next
```

```
End If
```

```
Next
```



```
End Sub
```

Wirebond.StartX Property

Returns the x-coordinate for the start of the bond wire.

**Restriction:**

This information applies to only the BGA toolkit.

Usage

StartX([*unit* As [PPcbUnit](#) = ppcbUnitCurrent]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the x-coordinate value is returned.

Description

None

Examples

The following sample code retrieves the start position of each bond wire for each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aWB in comp.Wirebonds

MsgBox aWB.Name & " Start: (" & aWB.StartX & ", " & aWB.StartY & ")"

Next

End If

Next

End Sub
```

Wirebond.StartY Property

Returns the y-coordinate for the start of the bond wire.



Restriction:

This information applies to only the BGA toolkit.

Usage

StartY([*unit* As [PPcbUnit](#) = ppcbUnitCurrent]) As Double

Arguments

<i>unit</i>	[Optional on page 732] Unit in which the y-coordinate value is returned.

Usage

None

Examples

The following sample code retrieves the start position of each bond wire for each die in the open design. See [Code Samples](#) for more information on running this sample.

```
Sub Main

For Each comp In ActiveDocument.Components

If comp.IsDiePart Then

For Each aWB in comp.Wirebonds

MsgBox aWB.Name & " Start: (" & aWB.StartX & ", " & aWB.StartY & ")"

Next

End If

Next

End Sub
```

Constants

As part of its hierarchy, SailWind Layout also contains constants.

SailWind Layout contains the following constants:

• PPcbAntiPadShape • PPcbASCIISections • PPcbASCIIVersion • PPcbAttrFlags • PPcbBondPadEdge • PPcbBondPadShape • PPcbDesignObject • PPcbDielectricLayer • PPcbDielectricType • PPcbDocumentColor • PPcbDrawingType • PPcbDRCMode • PPcbGridType	• PPcbHorizontalJustification • PPcbLabelType • PPcbLabelDisplayMode • PPcbLayerColor • PPcbLayerType • PPcbLibraryItemType • PPcbMeasureFormat • PPcbNudgeMode • PPcbObjectType • PPcbOriginType • PPcbOutlineType • PPcbPadCornerType • PPcbPadShape	• PPcbPadStackLayerType • PPcbPinElectricalType • PPcbPlaneType • PPcbRightReadingStatus • PPcbSegmentType • PPcbShapeType • PPcbTestPointType • PPcbThermalPadShape • PPcbUnit • PPcbVerticalJustification

PPcbAntiPadShape

ppcbAntiPadShapeRound= 8

ppcbAntiPadShapeSquare= 9

PPcbASCIISections

Possible ASCII section values are:

ppcbASCIISectionPCB = &H00000001

ppcbASCIISectionReuse = &H00000002

ppcbASCIISectionText = &H00000004

ppcbASCIISectionLines = &H00000008

ppcbASCIISectionClusters = &H00000010

ppcbASCIISectionVias = &H00000020

ppcbASCIISectionDecals = &H00000040

ppcbASCIISectionPartTypes = &H00000080

ppcbASCIISectionParts = &H00000100

ppcbASCIISectionJumpers = &H00000200

ppcbASCIISectionConnections = &H00000400

ppcbASCIISectionRoutes = &H00000800
ppcbASCIISectionTeardrops = &H00001000
ppcbASCIISectionMisc = &H00002000
ppcbASCIISectionRules = &H00004000
ppcbASCIISectionCAM = &H00008000
ppcbASCIISectionPour = &H00010000
ppcbASCIISectionAssemblyOptions= &H00020000
ppcbASCIISectionTestPoints = &H00040000
ppcbASCIISectionAttributes = &H00080000
ppcbASCIISectionAll = &HFFFFFFF

PPcbASCIIVersion

Possible ASCII version values are:

ppcbASCIIVerCurrent = 0
ppcbASCIIVer1_1 = 2
ppcbASCIIVer1_5 = 3
ppcbASCIIVer2_0 = 4
ppcbASCIIVer2_1 = 5
ppcbASCIIVer2_5 = 6
ppcbASCIIVer3_0 = 7
ppcbASCIIVer4_0 = 8

PPcbAttrFlags

Possible attribute flags values are:

ppcbAttrNone = &H00000001
ppcbAttrPart = &H00000002
ppcbAttrNet = &H00000004
ppcbAttrPin = &H00000008
ppcbAttrVia = &H00000010
ppcbAttrPCB = &H00000020
ppcbAttrPartType = &H00000040
ppcbAttrDecal = &H00000080
ppcbAttrNetClass = &H00000100
ppcbAttrAll = &HFFFFFFF

PPcbBondPadEdge



Restriction:

This information applies to only the BGA toolkit.

Possible bond pad edge values are:

ppcbBondPadEdgeUnknown = 0

ppcbBondPadEdgeLeft = 1

ppcbBondPadShapeTop = 2

ppcbBondPadShapeRight = 3

ppcbBondPadShapeBottom = 4

PPcbBondPadShape



Restriction:

This information applies to only the BGA toolkit.

Possible bond pad shape values are:

ppcbBondPadShapeUnknown = 0

ppcbBondPadShapeRectangle = 1

ppcbBondPadShapeOval = 2

PPcbDesignObject

ppcbDesignObjectTrace = 0

ppcbDesignObjectVia = 1

ppcbDesignObjectPad = 2

ppcbDesignObjectCopper = 3

ppcbDesignObjectLine = 4

ppcbDesignObjectText = 5

ppcbDesignObjectError = 6

ppcbDesignObjectOutlineTop = 7

ppcbDesignObjectOutlineBottom = 8

ppcbDesignObjectRefDes = 9

ppcbDesignObjectPartType = 10

ppcbDesignObjectAttribute = 11

ppcbDesignObjectKeepout = 12

ppcbDesignObjectPinNumber = 13

PPcbDielectricLayer

ppcbDielectricLayerAbove = 0

ppcbDielectricLayerBelow = 1

PPcbDielectricType

ppcbDielectricTypeCoating = 0

ppcbDielectricTypeSubstrate = 1

ppcbDielectricTypePrepreg = 2

PPcbDocumentColor

ppcbDocumentColorBackground = 0

ppcbDocumentColorSelection = 1

ppcbDocumentColorHighlight = 2

ppcbDocumentColorBoardOutline = 3

ppcbDocumentColorConnection = 4

PPcbDrawingType

Possible drawing type values are:

ppcbDrw2Dline = 0

ppcbDrwBoard = 1

ppcbDrwCopper = 3

ppcbDrwCopperPour = 6

ppcbDrwCopperHatch = 7

ppcbDrwCopperThermal = 8

ppcbDrwKeepout = 9

PPcbDRCMode

Possible Design Rules Checking (DRC) mode values are:

ppcbDRCTNone = 0

ppcbDRCOff = 1

ppcbDRCWarn = 2

ppcbDRCIgnoreClearance = 3

ppcbDRCPrevent = 4

PPcbErrorClass

Possible Error Class values are:

ppcbErrorClassClearanceError = 0
ppcbErrorClassConnectivityError = 0x20
ppcbErrorClassHighSpeedError = 0x40
ppcbErrorClassMiscError = 0x60
ppcbErrorClassTestPointError = 0x80
ppcbErrorClassDfmError = 0xa0
ppcbErrorClassBgaError = 0xc0
ppcbErrorClassViaCountError = 0xe0

PPcbErrorType

Possible Error Type values are:

ppcbErrorTypePadToPadError = 0 | 0
ppcbErrorTypePadToTrackError = 0 | 0x1
ppcbErrorTypeTrackToTrackError = 0 | 0x2
ppcbErrorTypeCopperPourError = 0 | 0x3
ppcbErrorTypeDrillHoleError = 0 | 0x4
ppcbErrorTypeDrillToDrillError = 0 | 0x5
ppcbErrorTypeTraceWidthError = 0 | 0x6
ppcbErrorTypePlacementKeepoutError = 0 | 0x7
ppcbErrorTypeHeightKeepoutError = 0 | 0x8
ppcbErrorTypeDrillKeepoutError = 0 | 0x9
ppcbErrorTypeTraceKeepoutError = 0 | 0xa
ppcbErrorTypePourKeepoutError = 0 | 0xb
ppcbErrorTypeViaKeepoutError = 0 | 0xc
ppcbErrorTypeTestPointKeepoutError = 0 | 0xd
ppcbErrorTypeBoardOutlineError = 0 | 0xe
ppcbErrorTypeSameNetPadToPadError = 0 | 0xf
ppcbErrorTypeSameNetPadToTrackError = 0 | 0x10
ppcbErrorTypeBodyToBodyError = 0 | 0x11
ppcbErrorTypeConnectivityError = 0x20 | 0
ppcbErrorTypeConnectivityDrillError = 0x20 | 0x1
ppcbErrorTypeConnectivityPlaneError = 0x20 | 0x2

ppcbErrorTypeDrillLayerPairingError = 0x20 | 0x3
ppcbErrorTypeDrillToPlaneShortError = 0x20 | 0x4
ppcbErrorTypeEDC_CapacitanceError = 0x40 | 0
ppcbErrorTypeEDC_LengthError = 0x40 | 0x1
ppcbErrorTypeEDC_DelayError = 0x40 | 0x2
ppcbErrorTypeEDC_MinImpedanceError = 0x40 | 0x3
ppcbErrorTypeEDC_MaxImpedanceError = 0x40 | 0x4
ppcbErrorTypeEDC_LoopError = 0x40 | 0x5
ppcbErrorTypeEDC_StubError = 0x40 | 0x6
ppcbErrorTypeEDC_ParallelismError = 0x40 | 0x7
ppcbErrorTypeTiePlaneError = 0x60 | 0
ppcbErrorTypeTearDropGenError = 0x60 | 0x1
ppcbErrorTypeLatiumMarkerError = 0x60 | 0x2
ppcbErrorTypeBLZ_ViaAtSMDFitInsideError = 0x60 | 0x3
ppcbErrorTypeBLZ_ViaAtSMDCenterError = 0x60 | 0x4
ppcbErrorTypeBLZ_ViaAtSMDEndError = 0x60 | 0x5
ppcbErrorTypeBLZ_ViaAtSMDCenterOutError = 0x60 | 0x6
ppcbErrorTypeBLZ_ViaAtSMDTooManyError = 0x60 | 0x7
ppcbErrorTypeBLZ_GapBadError = 0x60 | 0x8
ppcbErrorTypeBLZ_GapViolationError = 0x60 | 0x9
ppcbErrorTypeBLZ_GapObstacleSizeError = 0x60 | 0xa
ppcbErrorTypeBLZ_GapObstacleCountError = 0x60 | 0xb
ppcbErrorTypeBLZ_GapIrregularLengthError = 0x60 | 0xc
ppcbErrorTypeBLZ_GapControlledPercentError = 0x60 | 0xd
ppcbErrorTypeBLZ_LengthToleranceError = 0x60 | 0xe
ppcbErrorTypeBLZ_ProfileCornerError = 0x60 | 0xf
ppcbErrorTypeTestPointGenError = 0x80 | 0
ppcbErrorTypeTestPointProbeToProbeError = 0x80 | 0x1
ppcbErrorTypeTestPointProbeToComponentError = 0x80 | 0x2
ppcbErrorTypeTestPointProbeToBoardError = 0x80 | 0x3
ppcbErrorTypeTestPointProbeToKeepoutError = 0x80 | 0x4
ppcbErrorTypeTestPointNailCountError = 0x80 | 0x5
ppcbErrorTypeBLZ_ProbeNotComponentError = 0x80 | 0x6
ppcbErrorTypeBLZ_ProbeOnSMDPinError = 0x80 | 0x7

```
ppcbErrorTypeBLZ_ProbeGridError = 0x80 | 0x8
ppcbErrorTypeBLZ_ProbeNailDiameterError = 0x80 | 0x9
ppcbErrorTypeBLZ_ProbeToPadError = 0x80 | 0xa
ppcbErrorTypeBLZ_ProbeToTraceError = 0x80 | 0xb
ppcbErrorTypeDFMGenError = 0xa0
ppcbErrorTypeDFMAcidTrapError = 0xa0 | 0x1
ppcbErrorTypeDFMSliverError = 0xa0 | 0x2
ppcbErrorTypeDFMSolderBridgeError = 0xa0 | 0x3
ppcbErrorTypeDFMStarvedThermalError = 0xa0 | 0x4
ppcbErrorTypeDFMLayerCompareError = 0xa0 | 0x5
ppcbErrorTypeDFMDrillError = 0xa0 | 0x6
ppcbErrorTypeDFFGenError = 0xa0 | 0x7
ppcbErrorTypeDFFAcidTrapError = 0xa0 | 0x8
ppcbErrorTypeDFFSliverError = 0xa0 | 0x9
ppcbErrorTypeDFFSolderBridgeError = 0xa0 | 0xa
ppcbErrorTypeDFFStarvedThermalError = 0xa0 | 0xb
ppcbErrorTypeDFFLayerCompareError = 0xa0 | 0xc
ppcbErrorTypeDFFDrillError = 0xa0 | 0xd
ppcbErrorTypeDFFDrillToMaskError = 0xa0 | 0xe
ppcbErrorTypeDFFPadToMaskError = 0xa0 | 0xf
ppcbErrorTypeDFFDrillToPadError = 0xa0 | 0x10
ppcbErrorTypeDFFTraceWidthError = 0xa0 | 0x11
ppcbErrorTypeDFFPadSizeError = 0xa0 | 0x12
ppcbErrorTypeBGA_GenError = 0xc0
ppcbErrorTypeBGA_WBRMinLengthError = 0xc0 | 0x1
ppcbErrorTypeBGA_WBRMaxLengthError = 0xc0 | 0x2
ppcbErrorTypeBGA_WBRMaxAngleError = 0xc0 | 0x3
ppcbErrorTypeBGA_WBRWBToWBEError = 0xc0 | 0x4
ppcbErrorTypeBGA_WBRWBToSBPEError = 0xc0 | 0x5
ppcbErrorTypeViaCountError = 0xe0
```

PPcbErrorValueType

Possible Error Value Type values are:

```
ppcbErrorValueTypeMeasure = 0
```

ppcbErrorValueTypeInt = 1

ppcbErrorValueTypeDouble = 2

PPcbGridType

Possible grid type values are:

ppcbGridNone = 0 None.

ppcbGridDesign = 1 SailWind Layout design grid.

ppcbGridVia = 2 SailWind Layout via grid.

ppcbGridDisplay = 3 SailWind Layout display grid.

ppcbGridAll = 9999 All SailWind Layout grids.

PPcbHorizontalJustification

Possible horizontal justification values are:

ppcbJustifyLeft = 0

ppcbJustifyHCenter = 1

ppcbJustifyRight = 2

PPcbLabelType

Possible label type values are:

ppcbLabelTypeRefDesignator = 0

ppcbLabelTypePartType = 1

ppcbLabelTypeAttribute = 2

PPcbLabelDisplayMode

Possible label display mode values are:

ppcbLabelDisplayNone = 0

ppcbLabelDisplayValue = 1

ppcbLabelDisplayNameAndValue = 2

ppcbLabelDisplayFullNameAndValue = 3

PPcbLayerColor

ppcbLayerColorTrace = 0

ppcbLayerColorVia = 1

ppcbLayerColorPad = 2

ppcbLayerColorCopper = 3

ppcbLayerColorLine = 4

ppcbLayerColorText = 5
ppcbLayerColorError = 6
ppcbLayerColorOutlineTop = 7
ppcbLayerColorOutlineBottom = 8
ppcbLayerColorRefDes = 9
ppcbLayerColorPartType = 10
ppcbLayerColorAttribute = 11
ppcbLayerColorKeepout = 12,
ppcbLayerColorPinNumber = 13

PPcbLayerType

Possible layer type values are:

ppcbLayerUnknown = 0
ppcbLayerComponent = 1
ppcbLayerRouting = 2
ppcbLayerDrill = 3
ppcbLayerSilkscreen = 4
ppcbLayerPasteMask = 5
ppcbLayerSolderMask = 6
ppcbLayerAssembly = 7
ppcbLayerGeneral = 8
ppcbLayerAll = 9999

PPcbLibraryItemType

Possible library item type values are:

ppcbLibraryItemTypePartType = 0
ppcbLibraryItemTypeDecal = 1
ppcbLibraryItemTypeLogicDrawing = 2
ppcbLibraryItemTypeDrawing = 3
ppcbLibraryItemTypeAll = 9999

PPcbLineStyle

Possible line style values are:

ppcbLineStyleSolid = 0
ppcbLineStyleDashed = 1

ppcbLineStyleDotted = 2
ppcbLineStyleDashDotted = 3
ppcbLineStyleDashDoubleDotted = 4
ppcbLineStyleMultiple = 5

PPcbMeasureFormat

Possible SailWind Layout measure format values are:

ppcbMeasureFormatStandard = 0
ppcbMeasureFormatCurrent = 1
ppcbMeasureFormatShort = 2
ppcbMeasureFormatLong = 3

PPcbNudgeMode

Possible nudge mode values are:

ppcbNudgeNone = 0
ppcbNudgeOff = 1
ppcbNudgeWarn = 2
ppcbNudgeAuto = 3

PPcbObjectType

Possible database object type values are:



Tip

ppcbObjectTypeCBP, ppcbObjectTypeSBP, and ppcbObjectTypeWirebond are available in the Advanced Packaging Toolkit only.

ppcbObjectTypeUnknown = 0

The server may return this value to indicate an invalid object. The client may use this value when trying to work with empty object collections.

ppcbObjectTypeComponent = 1
ppcbObjectTypeNet = 2
ppcbObjectTypePin = 3
ppcbObjectTypeVia = 4
ppcbObjectTypeConnection = 5
ppcbObjectTypeRouteSegment = 6
ppcbObjectTypeJumper = 7
ppcbObjectTypePartType = 8

ppcbObjectTypeCBP = 9
ppcbObjectTypeSBP = 10
ppcbObjectTypeWirebond = 11
ppcbObjectTypeNetClass = 12
ppcbObjectTypeDrawing = 13
ppcbObjectTypeText = 14
ppcbObjectTypeLabel = 15
ppcbObjectTypePolyline = 16
ppcbObjectTypeCircle = 17
ppcbObjectTypeLibrary = 18
ppcbObjectTypeLibraryItem = 19
ppcbObjectTypeApplication = 20
ppcbObjectTypeAttribute = 21
ppcbObjectTypeAttributeType = 22
ppcbObjectTypeDocument = 23
ppcbObjectTypeMeasure = 24
ppcbObjectTypeView = 25
ppcbObjectTypeAssemblyOptions = 26
ppcbObjectTypeAttributes = 27
ppcbObjectTypeAttributeTypes = 28
ppcbObjectTypeObjects = 29
ppcbObjectTypePadStackLayer = 31
ppcbObjectTypePad = 32
ppcbObjectTypeThermalPad = 33
ppcbObjectTypeAntiPad = 34
ppcbObjectTypeLayer = 35
ppcbObjectTypeError = 36
ppcbObjectTypeErrorConflict = 37
ppcbObjectTypeAssociatedNet = 38
ppcbObjectTypeAll = 9999

All SailWind Layout Automation database object types, including the Component, Net, Pin, Via, Connections, and RouteSegment objects, but not the CBP, SBP, and Wirebond objects.

This specifically means that if you use the Document.GetObjects (ppcbObjectTypeAll) method, the collection of objects returned does not include any CBP, SBP, or Wirebond objects.

PPcbOriginType

Possible origin type values are:

ppcbOriginTypeDesign = 0

ppcbOriginTypeParent = 1

PPcbOutlineType

Possible outline type values are:

ppcbOutlineTypeCenter = 0

ppcbOutlineTypeOuter = 1

ppcbOutlineTypeInner = 2

PPcbPadCornerType

ppcbPadCornerType90Degree = 0

ppcbPadCornerTypeChamfered = 1

ppcbPadCornerTypeRounded = 2

PPcbPadShape

ppcbPadShapeOvalFinger = 0

ppcbPadShapeRectangularFinger = 1

ppcbPadShapeRound = 2

ppcbPadShapeSquare = 3

ppcbPadShapeAnnular = 4

ppcbPadShapeOdd = 5

PPcbPadStackLayerType

ppcbPadStackLayerTypeMounted = -2

ppcbPadStackLayerTypeInner = -1

ppcbPadStackLayerTypeOpposite = 0

PPcbPinElectricalType

Possible gate electrical type values are:

ppcbElectricalTypeUnknown = 0

ppcbElectricalTypeSource = 1

ppcbElectricalTypeBidirectional = 2

ppcbElectricalTypeOpenCollector = 3

ppcbElectricalTypeOrTieableSource = 4

ppcbElectricalTypeTristate = 5
ppcbElectricalTypeLoad = 6
ppcbElectricalTypeTerminator = 7
ppcbElectricalTypePower = 8
ppcbElectricalTypeGround = 9

PPcbPlaneType

ppcbPlaneTypeNoPlane = 0
ppcbPlaneTypeCAMPlane = 1
ppcbPlaneTypeSplitMixedPlane = 2

PPcbRightReadingStatus

Possible right-reading status values are:

ppcbRightReadingNone = 0
ppcbRightReadingOrthogonal = 1
ppcbRightReadingAngled = 2

PPcbRoutingdirection

ppcbRoutingDirectionHorizontal = 0
ppcbRoutingDirectionVertical = 1
ppcbRoutingDirectionAny = 2
ppcbRoutingDirectionDiagonal45 = 3
ppcbRoutingDirectionDiagonal135 = 4

PPcbSegmentType

Possible segment type values are:

ppcbSegmentUnknown = 0
ppcbSegmentLine = 1
ppcbSegmentArc = 2

PPcbShapeType

Possible shape type values are:

ppcbShapeTypeOpen = 0
ppcbShapeTypeHollow = 1
ppcbShapeTypeFilled = 2
ppcbShapeTypeVoid = 3

PPcbTestPointType

Possible test point type values are:

ppcbTestPointNone = 0 No test point.

ppcbTestPointTopLayer = 1 Test point on top layer.

ppcbTestPointBottomLayer = 2 Test point on bottom layer.

PPcbThermalPadShape

ppcbThermalPadShapeRound = 6

ppcbThermalPadShapeSquare = 7

PPcbUnit

Possible unit type values are:

ppcbUnitCurrent = 0 Current unit in use in SailWind Layout.

ppcbUnitDatabase = 1 Internal SailWind Layout database unit.

ppcbUnitMils = 2 Mils unit (1/1000 of an inch).

ppcbUnitInch = 3 Inch unit.

ppcbUnitMetric = 4 Metric unit (1/1000 of a meter).

PPcbVerticalJustification

Possible vertical justification values are:

ppcbJustifyTop = 0

ppcbJustifyVCenter = 1

ppcbJustifyBottom = 2

Optional Argument

An *optional argument* is an argument to a property or a method that can be omitted because a default value is used when the optional argument is not specified. The default value is the value that is statistically used most, or one that is most representative for the argument.

For example if the method `M([arg1], [arg2])` has both `arg1` and `arg2` arguments set as *optional*, you can call the method `M` in four different ways:

M() The default values of both `arg1` and `arg2` are passed.

M(<value1>) `Arg1 <value1>` and `arg2` default value are passed.

M(, <value2>) `Arg1` default value and `arg2 <value2>` are passed.

M(<value1>, <value2>) Both `arg1 <value1>` and `arg2 <value2>` are passed.

Variant

A *Variant* is a data type which can contain or represent any kind of data, such as a Boolean value, an Integer value, a Long value, a Double value, a String value, or an Array.

The Variant data type is used in Automation in two different situations:

- When the argument of a property or method is data that can be represented as different values and value types. For example, a specific object in a collection of objects is referenced by its index in the collection (a Long value) or its name (a String value).
- When the argument or the return value of a property or a method is a complex type that is not explicitly defined. For example, an array of points is represented as a Variant.

Exception

An Automation exception is a special notification from the server that signals the Automation client of an error. For example, if a Basic script is trying to delete an attribute that doesn't exist, the program generates an exception. When the script receives the exception, the following occurs:

- If an exception handler is implemented using the On Error Basic statement, then the flow of execution of the client code is rerouted to the exception handler.
- If an exception handler is not implemented, then the client's default handler is called. In all Basic interpreters, the default handler is a break point at the line that generated the exception.

Samples

Sample code in this document is provided for getting started with your automation scripts.

[Code Samples](#)
[Running Code Samples in SailWind Layout](#)
[Running Code Samples Outside of SailWind Layout](#)
[Sample Code Enhancement](#)
[Troubleshooting Code Samples](#)

Code Samples

Code samples illustrate how to use an Automation property, method, or event.

You can run code samples either [in SailWind Layout](#) on page 734 or [outside of SailWind Layout](#) on page 735.

The code sample is always in the following form:

```
Sub Main
```

```
' Do something
```

```
End Sub
```

For more information on running code samples, see [Troubleshooting Code Samples](#) and [Sample Code Enhancement](#).



Note:

The code samples in SailWind Layout Automation Server Help are *freeware*. These samples are provided as a courtesy to its users. Freeware is provided as is and no warranties with respect to freeware are made, either express or implied, including any implied warranties of merchantability or fitness for a particular purpose.

Running Code Samples in SailWind Layout

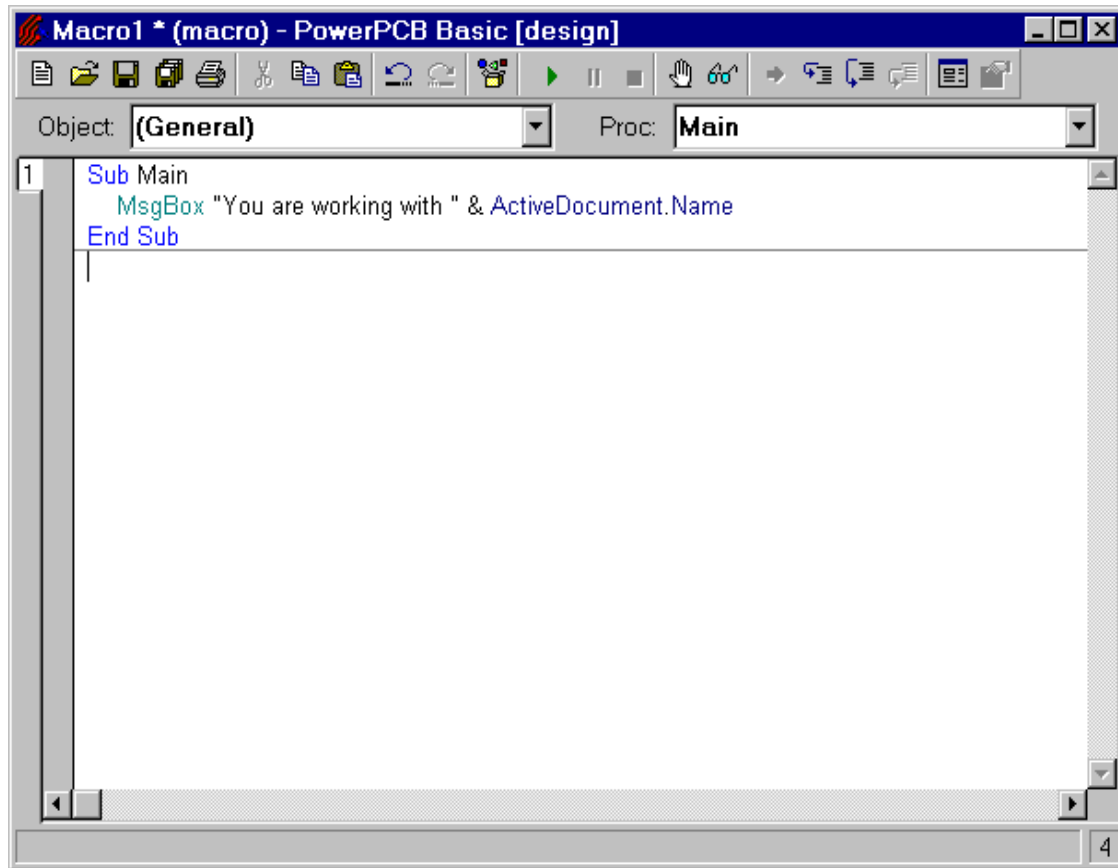
Use the Basic Script Editor to create code samples from within the software.

Procedure

1. Select the sample, including the Sub Main and End Sub statements.
2. Copy the sample to the Clipboard using Edit/Copy.
3. Choose Tools/Basic Scripting/Script Editor. The Sax Basic Engine dialog box appears.
4. Paste the code sample into the Sax Basic editor using Edit/Paste.
5. Click the Start button on the toolbar in the Sax Basic Editor dialog box to run the sample.

- Figure 6 shows the code sample for the [Application.ActiveDocument Property](#) property pasted into the Basic Engine dialog box.

Figure 6. Code Sample in Basic Engine Dialog Box



Running Code Samples Outside of SailWind Layout

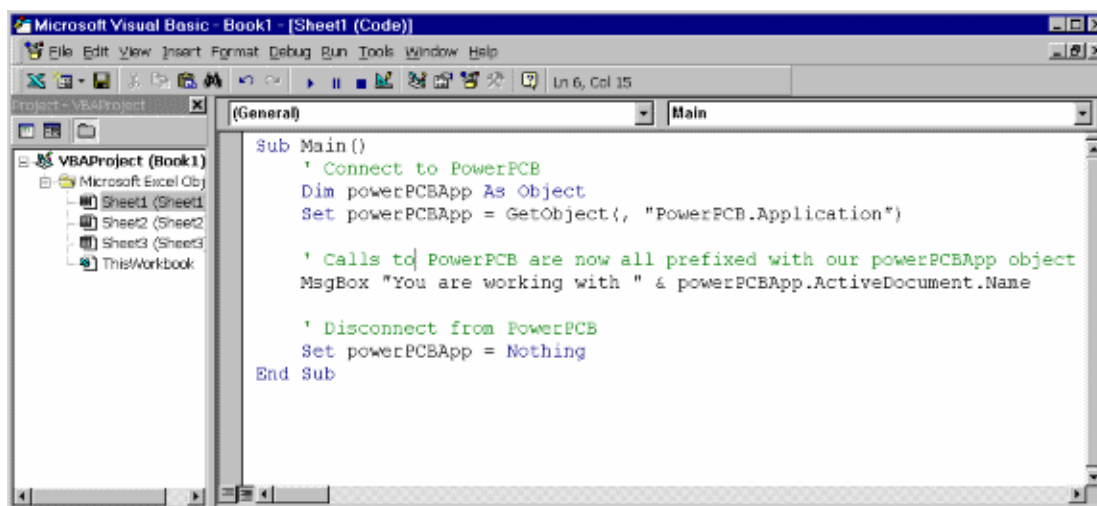
You can use sample code with Basic scriptable applications (also called host applications) such as Visual Basic, Microsoft Excel, Microsoft Word.

Procedure

- Import the Automation server references into the host application. For example, to use Excel as the host application:
- Choose Tools/Macro/Visual Basic Editor.
- Choose Tools/References. The References dialog box appears.
- In the list box, select SailWind Layout <latest_version> Type Library.
- Paste the code sample in the editor of the host application. To do this in Excel:
- Choose View/Code.
- Choose Edit/Paste.

8. Add code to the beginning of the code sample to connect the host application to SailWind Layout using the Basic GetObject function.
9. Add the object returned by the Basic GetObject function to the beginning of each Automation method and property in the sample code.
10. Add code to the end of the sample to disconnect the application from the program.
11. **Figure 7** shows the code sample for the [Application.ActiveDocument Property](#) property pasted and modified in Microsoft Excel's Visual Basic Editor. See the host application help for more information.

Figure 7. Code Sample in Excel Visual Basic Editor



Sample Code Enhancement

The code samples in the Automation Server Help are reduced to the minimum number of lines necessary to quickly illustrate how to use Automation properties and methods. You can use these code samples as a base for developing your own SailWind Layout features.

The following list provides ideas for possible enhancements:

- Add strict type checking, using the Option Explicit declaration at the beginning of the code and declare all variables using the Dim Basic keyword. This ensures that the program can interpret your variables properly and can generate compiling errors when a problem exists.
- Add error checking using the On Error Basic keyword. Error checking improves how code reacts to a run-time error.
- Display information in custom dialog boxes using the UserDialog Basic keyword, instead of displaying in simple dialog boxes created using the MsgBox keyword.



The code samples in the Automation Server On-line Help are freeware. These samples are provided as a courtesy to its users. Freeware is provided as is and no warranties with respect to freeware are made, either express or implied, including any implied warranties of merchantability or fitness for a particular purpose.

Troubleshooting Code Samples

You can perform a series of basic checks if a code sample does not run correctly.

Procedure

1. Make sure a design is open. Most code samples require a design file to be open when you run the sample.
2. Check for any assumptions the samples make about the design files.

For example, some code samples assume that component U1 exists, others assume that the Assembly Option MyAssOpt exists, and others assume that at least one Assembly Option exists.

These assumptions are clearly stated in the text preceding each sample. If these assumptions are not true, the sample code will not run properly. You can adapt the sample to your design file.

3. If you modified the sample code to run outside of the program, ensure that you properly applied all required changes to the code, as described in [Code Samples](#).

Common mistakes include forgetting to import references and forgetting to prefix Automation constants.



Note:

The code samples in the Automation Server Help are *freeware*. These samples are provided as a courtesy to its users. Freeware is provided as is and no warranties with respect to freeware are made, either express or implied, including any implied warranties of merchantability or fitness for a particular purpose.

RGL Format Files

SailWind Layout includes Automation Server support for all Report Generation Language (RGL) keywords.

- [Converted RGL Format Files](#)
- [Automation for RGL Top Level Keywords](#)
- [Automation for RGL SubLevel Keywords](#)
- [Automation for RGL Field Keywords](#)
- [New Automation Functions for RGL](#)

Converted RGL Format Files

The OLE samples include examples for each RGL format file (.fmt). You can customize these new report format files to fit your output requirements. The files, located in the *C:\SailWind Projects\Samples\Scripts\Layout\RGL_Samples* folder, were created using the Sax Basic Engine. The file name extension used for these files is .bas. You can load these scripts into the Tools/Basic Scripts menu using the Basic Scripts dialog box.

The RGL replacement scripts described below (*.bas) include references to rgl.bas. This file is a library file that contains functions the .bas scripts need to operate. The .bas scripts *must* refer to *rgl.bas* (within the script) to run.

RGL keywords are replaced with Basic functions and subroutines.

Script Names and Report Types

To help you determine the type of output each program provides, [Table 4](#) identifies the Sax Basic script name and type of report it generates.

Table 4. Sax Basic Script Names and Report Type

RGL Report	Sax Basic File (.bas)	Description
Netlist without pin info	Net List without pin info.bas	Reports signals by netname without pin information.
Netlist with pin info	Net List with pin info.bas	Reports signals by netname with pin information.
Parts List 1	Part List 1.bas	Reports parts by reference designator.
Parts List 2	Part List 2.bas	Reports reference designators by part type.
Test points report	Test points report.bas	Reports test point locations and netname.
Jumper List	Jumper List.bas	Reports jumper locations and netnames.

Table 4. Sax Basic Script Names and Report Type (continued)

PowerPCB Format Netlist	PowerPCB V2.0 Format Netlist.bas, PowerPCB V3.0 Format Netlist.bas	Reports a PowerPCB format netlist.
DFT Extended Test Point	DFT Extended test point report.bas	Reports test points by nets, nets without test points, and number of test points per net.

Related Topics

[Automation for RGL Field Keywords](#)

[Automation for RGL SubLevel Keywords](#)

[Automation for RGL Top Level Keywords](#)

Automation for RGL Top Level Keywords

Certain automation methods and functions replace RGL top level keywords.

[Table 5](#) and [Table 6](#) list the methods and functions.

Table 5. Top Level Keywords Replaced by Direct Automation Methods

RGL Keywords	Method	Description
JOBNM	ActiveDocument.Name	Returns the JOB NAME.
COMPCNT	ActiveDocument.Components.Count	Returns the total number of components.
SIGCNT	ActiveDocument.Nets.Count	Returns the total signal count.
BOARDSZ	ActiveDocument.BoardOutline Surface	Returns the board size in current units.

Table 6. Top Level Keywords Replaced by Functions and Subroutines in RGL.BAS

RGL Keywords	Automation Function	Description
TIME	Function GetTime() As String	Returns the current date and time in the format “ddd mmm dd hh:mm:ss yyyy.”
LAYERCNT	Function LayerCnt(doc As Object)	Returns the total number of component and routing layers.
PKG CNT	Function PkgCnt(doc As Object) As Integer	Returns the total number of packages in use on the PCB.
SYMCNT	Function SymCnt(doc As Object) As Integer	Returns the total number of decals in use on the PCB.

Table 6. Top Level Keywords Replaced by Functions and Subroutines in RGL.BAS (continued)

EQUIV_IC	Function EquivalentIC(doc As Object) As Double	Uses the total number of pins on all components on the board divided by 14.
BD_DENSITY	Function BdDensity(doc As Object) As Double	Calculates the board density by using the surface area of the PCB divided by the Equivalent IC value.
PSIGCNT	Function PSigCnt(doc As Object) As Integer	Returns the total number of Power Nets on the PCB.
SSIGCNT	Function SSigCnt(doc As Object) As Integer	Returns the total number of signal nets on the PCB, excluding power nets.
TOPCOMP CNT	Function TopCompCnt(doc As Object) As Integer	Returns the total number of components on the top side of the PCB.
BOTCOMP CNT	Function BottomCompCnt(doc As Object) As Integer	Returns the total number of components on the bottom side of the PCB.
PADCNT	Function PadCnt(doc As Object) As Integer	Returns the total number of pads (drilled and undrilled) on the PCB.
DRPADCNT	Function DrPadCnt(doc As Object) As Integer	Returns the total number of drilled pads on the PCB (Drill size > 0).
NDPADCNT	Function NDPadCnt(doc As Object) As Integer	Returns the total number of surface mount pads (drill size = 0).

Related Topics

[Converted RGL Format Files](#)

[Automation for RGL SubLevel Keywords](#)

[Automation for RGL Field Keywords](#)

[New Automation Functions for RGL](#)

Automation for RGL SubLevel Keywords

Certain methods and functions replace RGL sublevel keywords.

[Table 7](#) and [Table 8](#) list the methods and functions. Lower level keywords are indented.

Table 7. Sublevel Keywords Replaced by Direct Automation Methods

RGL Keywords	Description
SIGNALS	Search within signals. <i>' Iterate through all net of document doc like thisFor Each nextNet In doc.Nets... ' Do Something with that Net nextNetNext nextNet</i>
SIGNAME	<i>Net.Name</i> Name of signals.
PINS	Search within pins. <i>' Iterate through all pins of net nextNet like thisFor Each nextPin In nextNet.Pins... ' Do Something with that Pin nextPinNext nextPin</i>
COMPNAME	<i>Pin.Component.Name</i> Component name
PINNUM	<i>Pin.Number</i> Pin number
CONNECTIONS	Search connections. <i>' Iterate through all connections of document doc like thisFor Each nextConn In doc.Connections... ' Do Something with that Connection nextConnNext nextConn</i>
SIGNAME	<i>Connection.Net.Name</i> Name of signal(s)
COMP1	<i>Connection.Pins(1).Component.Name</i> Reference designator of connection end one
PIN1	<i>Connection.Pins(1).Number</i> Pin number of connection end one
COMP2	<i>Connection.Pins(2).Component.Name</i> Reference designator of connection end two
PIN2	<i>Connection.Pins(2).Number</i> Pin number of connection end two
ROUTSEGS	Search within routes. <i>' Iterate through all route segs of conn. nextConn like thisFor Each nextSeg In nextConn.RouteSegments... ' Do Something with that route segment nextSegNext nextSeg</i>
END1	First end point of segment <i>Points = RouteSegment.Points '! First, assign to variant !Points(1)</i>

Table 7. Sublevel Keywords Replaced by Direct Automation Methods (continued)

END2	Second end point of segment <i>Points = RouteSegment.Points '! First, assign to variant !Points(2)</i>
WIDTH	<i>RouteSegment.Width</i> Width of a trace segment
LAYER	<i>RouteSegment.Layer</i> Layer number of a trace segment
VIAS	Search within Vias <i>' Iterate through all vias of connection nextConn like thisFor Each nextVia In nextConn.Vias... ' Do Something with that via nextViaNext nextVia</i>
LOCX	<i>Via.PositionX</i> X Coordinate of VIA
LOCY	<i>Via.PositionY</i> Y Coordinate of via.
VIANAME	<i>Via.Type</i> Via name
PACKAGE	(in RGL.BAS: see PKGCNT)
PKGNAME	Package name <i>' Use the Component.PartType property</i>
PKGDSCR	<i>Component.Attributes("DESCRIPTION").Value</i> Package description
COMPONENTS	Search within packages. <i>' Iterate through all components of document doc like thisFor Each nextComp In doc.Components... ' Do Something with that Component nextCompNext nextComp</i>
COMPNAME	<i>Component.Name</i> Component name
SYMNAME	<i>Component.Decal</i> Symbol name
PKGTYPE	<i>Component.PartType</i> Package name
ANGLE	<i>Component.Orientation</i> Component placement angle

Table 7. Sublevel Keywords Replaced by Direct Automation Methods (continued)

LOCX	X	<i>Component.PositionX</i> Coordinate of placement
LOCY	Y	<i>Component.PositionY</i> Coordinate of placement
TESTPOINTS		(in RGL.BAS)
TPNAME		<i>Pin.Name</i> ' for pin test points <i>Via.Type</i> ' for via test points Test point name. The test point name for component pins is the standard pin name, such as U1.2. The test point name for vias is the via type name, such as STANDARDVIA.
SIGNAME		<i>Pin.Net</i> ' for pin test points <i>Via.Net</i> ' for via test points Signal (net) name. If the test point is on an unused component pin, the name *NONE* is returned.
LOCX	X	<i>Pin.PositionX</i> ' for pin test points <i>Via.PositionX</i> ' for via test points Coordinate for test point
LOCY	Y	<i>Pin.PositionY</i> ' for pin test points <i>Via.PositionY</i> ' for via test points Coordinate for test point
TESTSIDE		<i>Pin.TestPoint</i> ' for pin test points <i>Via.TestPoint</i> ' for via test points Testing side for the test point: TOP or BOTTOM

Table 8. Sublevel Keywords Replaced by Functions and Subroutines in RGL.BAS

RGL Keywords	Description
SIGNALS	Search within signals. (Direct automation method)
SIGNAME	Name of signals. (Direct automation method)
PINS	Search within pins. (Direct automation method)
PINTYP	Function PinType(Pin As Object) As String Returns a string value that identifies the type on pin, if it has been assigned by the user: Source="S", Bidirectional="B", OpenCollector="C", OrTieableSource="O", Tristate="T", Load="L", Terminator="Z", Power="P", Ground="G", Undefined="U"

Table 8. Sublevel Keywords Replaced by Functions and Subroutines in RGL.BAS (continued)

TPASSIGNED	Function TPAssigned(Net As Object) As Boolean Returns a true/false value if a net has a test point on it.
TESTPOINTCNT	Function TPCnt(Net As Object) As Integer Returns the total number of test points on all nets.
TESTPINCNT	Function TPPinCnt(Net As Object) As Integer Returns the total number of pins assigned as test points.
TESTVIACNT	Function TPViaCnt(Net As Object) As Integer Returns the total number of vias as test points.
PACKAGE	Sub GetPackageList(doc As Object, ByRef pkgList() As Package, Optional bSort As Boolean = False) Returns a list of all the packages on the PCB. Optional parameter to specify whether to sort the list.
TESTPOINTS	Function TestPoints(doc As Object) As objects Returns a list of test points on component pins, vias, and unused pins.

Automation for RGL Field Keywords

Certain functions and subroutines in *RGL.bas* replace RGL field keywords.

[Table 9](#) lists the replaced RGL field keywords.

Table 9. Field Keywords Replaced by Functions and Subroutines in RGL.BAS

RGL Keywords	Automation Function	Description
MAXCOLS	Sub MaxCols(cols As Integer)	Calculates the maximum number of columns in a report.
BETWEEN	Sub Between(Optional betweenCol As Integer = 0)	Calculates the space required between different columns of data.
BETWEEN	Sub End_Between()	Calculates the space required between different columns of data.
COLUMNS ,LEADING	Sub Columns(ParamArray formatParam())	Outputs the columns of a report in a table format.
COLUMNS, LEADING	Sub End_Columns()	Outputs the last column of data in a report.
DELIMITER	Not needed	

New Automation Functions for RGL

SailWind Layout makes available certain new functions and subroutines in *RGL.bas* to support RGL replacement.

[Table 10](#) lists the new functions and subroutines.

Table 10. New Functions and Subroutines in RGL.BAS to Support RGL Replacement

Automation Function	Description
Function Format2(value As Double) As String	Returns floating point numbers as a string to 5 decimal places.
Function NetPinsSortedByConnection(aNet As Object) As objects	Returns a list of pin pairs sorted alphabetically.
Function OpenReport(file As String) As Integer	Opens an output stream to a filename that you supply.
Sub CloseReport()	Closes the currently open output stream.
Sub Out(ParamArray formatParam())	Outputs the formatted data to the open stream.

Related Topics

[Converted RGL Format Files](#)

[Automation for RGL SubLevel Keywords](#)

[Automation for RGL Field Keywords](#)

[New Automation Functions for RGL](#)

Chapter 2

ASCII Format

The **File > Export** command can generate PADS-format ASCII files that carry all design-related information associated with a *.pcb* file.

Read the sections that follow for a high level overview of the ASCII format. For detailed information, please see the SailWind Layout ASCII File Format Specification.

You cannot always export all data to earlier ASCII formats. Earlier versions do not support several database structures. If your design exceeds any earlier database limits, either export is canceled or the unsupported data is not exported.

Layer-specific reference designator and part type references are converted to generic reference designator and part type settings when converting version 4.0 and higher CAM documents to earlier versions using ASCII Export. In addition, attributes and keepout settings are lost when exporting.

All information about part type and attribute labels is ignored. If a component has more than one reference designator label, one is selected and converted. Visibility and size status for the label is ignored; but the position is maintained.

Because jumpers, although not part of the part list, are considered vias, you can export them to ASCII.

PADS-format ASCII files created with previous versions are compatible with this version. However, importing an ASCII file that was not created with SailWind Layout resets any rules or layer assignments that are not part of the ASCII file to the SailWind Layout default. For example, if you set the directions for the layer of an empty database to vertical on top and horizontal on the bottom, the layers revert to horizontal on top and vertical on bottom when you read a netlist and save the file in *.pcb* format.

To avoid overwriting SailWind Layout rules when you read an ASCII file, use the Export command to output the rules as a *rules.asc* file. Read your ASCII file and then import the *rules.asc* file.

[ASCII Messages](#)

[ASCII Parts and Connections Lists](#)

ASCII Messages

You may receive one or more ASCII messages while running scripts in SailWind Layout.

Possible ASCII messages are detailed in the following table.

ASCII Message	Description
Part <Name> has more than 100 gates. The design cannot be exported to the selected format.	The design has a decal, part type, or a component exceeding the limit for the specified format.
The design has more than 30 electrical layers and cannot be exported to the selected format.	You have tried to export an increased layer mode design to a previous version of SailWind Layout.
The selected format does not support more than 30 layers. Objects on layers <layer range> will be deleted.	You have tried to export an increased layer mode design to a previous version of SailWind Layout in the ASCII Output dialog box. SailWind Layout

ASCII Format
ASCII Messages

ASCII Message	Description
	versions prior to 4.0 support default layer mode, but not increased layer mode. If you press OK, the file is exported to PADS-format ASCII. Layers 1 - n are unchanged in the ASCII file; layers in the indicated layer range are deleted in the ASCII file and all objects on these layers are deleted as well. Press Cancel to return to the ASCII Export dialog box

ASCII Parts and Connections Lists

The Export command generates PADS-format ASCII files that carry all design-related information associated with a .pcb file: positions for routing and vias, parts, copper, rules, GUI setups, and router strategy files.

Each information category appears after a unique control statement and with its own data format. This section covers formats necessary for creating the most commonly used PADS-format ASCII files: parts and connections lists. You can use any text editor to prepare these files.

[Units Used in ASCII Parts and Connections Lists](#)

[Parts List Section](#)

[Connection List](#)

Units Used in ASCII Parts and Connections Lists

You can replace mils with metric or inches.

Table 11. Units in ASCII Parts and Connections Lists

Unit	Description
Mils	Data is expressed to the nearest whole milli-inch (THOU). Degrees are expressed to the nearest tenth of a degree. Allowed ranges are from 56000 to +56000, not to exceed a total of 56000. Allowed ranges in degrees are from 0.0 to 359.9 degrees.
Inches	Data is expressed to the nearest thousandth of an inch. Degrees are expressed to the nearest tenth of a degree. Allowed ranges are 56.000 to +56.000, not to exceed a total of 56.000. Allowed ranges in degrees are from 0.0 to 359.9 degrees.
Metric	Data is expressed to the nearest tenth of a millimeter in millimeters. Degrees are expressed to the nearest tenth. Allowed ranges are from 1422.4 to +1422.4, not to exceed a total of 1422.4
Basic	One database unit = $2/3 \times 10^9$ meters. Allowed ranges are from 213,360,000,000 to +213,360,000,000.

Parts List Section

A V4.0 parts list contains a mandatory minimum content.

- [Minimum Parts List Content](#)
- [The *Standard* Header](#)
- [Notes on Parts List Format](#)
- [Sample Parts List](#)
- [Notes on Parts List Files](#)
- [Parts List Error Messages](#)

Minimum Parts List Content

The parts list must conform to a specific format.

The following displays the mandatory content. The parenthetical statements are for information only and not included.

!POWERPCB-V4.0-MILS!	(Initialization Header)
Part	(Part Section Header)
U1 7401	(Ref Des and Part Type)
U2 7401	(Ref Des and Part Type)
etc.	
etc.	
End	(Terminator of the parts list)

If you output an ASCII file parts list from a placed board, you can see the reference line summarizing the optional information you can include:

REMARK REFNM PTYPENM X Y ORI GLUE MIRROR ALT CLSTID CLSTATTR BROTHERID LABELS
REMARK .REUSE. INSTANCE RPART
REMARK VISIBLE LX LY LORI LHEIGHT LWIDTH LLEVEL MIRRORRED HJUST VJUST RTREAD
REMARK

Table 12 shows the fields in the reference line.

Table 12. Reference Line Fields

Field	Description
REFNM	Unique reference designator name, up to 15 alphanumeric characters.
PTYPENM	Part type name and optional PCB decal information. The part type can consist of 40 characters. The PCB decal will be appended at the end of the part type separated by an at (@) sign. The PCB decal will be appended to the part type in the ASCII file only if a part decal has been modified using the Pad Stacks Properties dialog box. Value and tolerance are not specified in this field.
X and Y	Coordinates of the part placement origin in the design.
ORI	Orientation in degrees of the part in the design. Allowed values range from 0.0 to 359.999, in increments of 0.001 of a degree.
GLUE	Flag indicating whether the symbol is glued or not. Allowed values are G (glue) or U (unglued).
MIRROR	Flag indicating whether the symbol has been mirrored to the opposite side of the board. Allowed values are M (mirrored on bottom layer) or N (not mirrored on top layer).
ALT	Alternate decal number. This is a sequential number and follows the sequence as defined in the part type file; for example, DIP14:DIP14\SO:DIP14\SOL are 0, 1, 2, respectively. Allowed values are from 0 to 15.
LABELS	Number of part labels. Each label is a two-line entry

When the part is a member of a physical design reuse, the following line is added after the part header line:

```
*REMARK* .REUSE. INSTANCE RPART
```

Table 13 shows the fields in the part header line.

Table 13. Fields in the Physical Design Reuse Part Header Line

Field	Description
.REUSE.	Keyword preceded and followed by periods (.).
INSTANCE	Physical design reuse name.
RPART	Part reference designator within a physical design reuse.

Labels define the locations for reference designator, part type, and attributes.

```
* REMARK * VISIBLE LX LY LORI LHEIGHT LWIDTH LLEVEL MIRRORED HJUST VJUST  
RTREAD
```

Table 14 shows the fields in the labels

Table 14. Fields in the Labels Line

Field	Description
VISIBLE	Label visibility type. Possible values are: VALUE, FULL_NAME, NAME, FULL_BOTH, BOTH, and NONE.
LX and LY	Coordinates of the label origin.
LORI	Relative label orientation. Precision is three digits after the decimal point.
LHEIGHT	Height of the label text.
LWIDTH	Line width of the label text.
LLEVEL	Layer on which the label is located. Possible values are from 0 to 30. 0 means all layers.
MIRRORED	Flag. Possible values are 0 and 1.
HJUST	Horizontal justification. Possible values are LEFT, CENTER, and RIGHT.
VJUST	Vertical justification. Possible values are UP, CENTER, and DOWN.
RTREAD	Right reading status. Possible values are ORTHO (orthogonal), ANGLED, or NONE.

The second line is the attribute name, for example, reference designator or part type.

SailWind Layout uses this format to record part position and placement status. You can use it before you load a part list to prelocate and glue connectors or mounting holes or to orient or rotate a part away from the orientation as stored in the parts library.



Tip

: The following characters are illegal for part names: carets (^), brackets ([]), ampersands (&), asterisks (*), braces ({}), parentheses (), at signs (@), question marks (?), equal signs (=), backward slashes (\), periods, commas, colons, and spaces.

The *Standard* Header

Adding the *Standard* header after the *Part* header automatically creates a power (VCC) and ground (GND) connection netlist, generated from the Signal Pin (SIGPIN) data contained in the part type information for each part in the part list. This eliminates your inputting the power and ground nets. However, if the part type data does not contain the power and ground data, these nodes are not generated automatically.

Notes on Parts List Format

Six columns is the maximum of mandatory and optional data.

To accept the data in a column, all previous columns must have entries. If no data exists for the previous column, enter zero (0).

For example, if all ICs are glued in the above parts list, G must exist in the GLU column for each IC. In addition, the X, Y, and Orientation columns require zero (0) entry for each IC in the parts list.

Sample Parts List

You can mix usage of the mandatory and optional features in your parts list.

An example of a short V4.0 parts list, where some parts use both the mandatory and optional features and other parts use only the mandatory features, appears as follows:

```
!POWERPCB-V4.0-MILS! DESIGN DATABASE ASCII FILE 1.0
```

```
*Part*
```

```
P1    CON\26P\ED    15200    8500    0.0    G
```

```
U1    74LS00
```

```
U2    74LS04
```

```
U3    74LS00
```

```
U4    74LS00
```

```
U{5-20}    74LS04
```

```
C1    CL25
```

```
C2    CL25
```

```
C3    CL25
```

```
C{4-20}    CK05
```

```
R1    R1/4W
```

```
ATTRIBUTE VALUES
```

```
{
```

```
PART R1
```

```
{
```

```
Value 1K
```

```
Tolerance 5%
```

```
}
```

```
}
```

```
*End*
```

From the above parts list connector P1 uses library part CON\26P\ED, is located at X=15.2" Y=8.5" from the origin, is oriented the same as originally created, and is glued at the indicated position. U1-U20 are the ICs. They are not prelocated, are at their original orientation, and are free to move.

The entries for U{5-20} and C{4-20} demonstrate a shorthand way to enter a long series of part reference designators using the identical part, without using an individual line for each part. The format is as follows:

- The first letter is the reference designator stem or family, for example U.
- The range of part reference designations, for example 5-20, enclosed with brackets.
- The part type preceded by a space.

In the following example, U5 through U20 use a 74LS04 part type.

```
U{5-20} 74LS04
```

The following example uses the family and tag approach for identifying the resistor R1. R1/4W is the family. R1/4W is the part type for the entire family of quarter watt resistors. This part type name calls the part type from the library. Attributes value 1K and Tolerance 5% are associated with the part after it is imported, and they uniquely identify the resistors.

```
R1 R1/4W
```

Notes on Parts List Files

The X and Y distance between the system board origin and the part origin is in 0.001" units (mils). Because it is assumed that the system origin and the board origin are the same, the parts are positioned in the proper location on the board relative to the board origin. If the system origin is not the same as the board origin, the X and Y placement of the components is not on the board.

Always read the parts list before the connection list. When the connection list is read via Import ASCII, it is checked, by reference designator, for the availability of each part in the connection list. If the part reference designator is in the connection list but not in the parts list, a part not found message appears. Correct the error to accept that line of the connection list.

Use Compare Netlists after the parts and connections are imported to ensure that all items are correctly transferred to the database.

Parts List Error Messages

As the parts list is read, it is checked for data inconsistencies. An error file is created of any errors. In addition, the line of data is not accepted in the database.

The following are possible error messages:

Error Message	Corrective Action
No *PowerPCB* found at beginning of file.	Add the correct header.
Expecting a valid *XXX* command statement.	Examine the list. Command statements starting with * are either in error or missing.
No *End* command found at end of file.	Add the *End* statement.
Bad *Parts* ASCII Data Line Format U11.1	Examine the line containing U11.1 for errors.

Connection List

A connection list must conform to a mandatory format.

- Mandatory Connection List Content
- Reference Designator Format
- *Signal* Statements and To-From Signal Names
- Discrete Component Pad Numbering
- Tying Nets Together
- Abbreviated Notations
- Width
- Sample Connection List
- Connection List Error Messages

Mandatory Connection List Content

A connection list must meet certain format requirements.

The mandatory format is outlined as follows:

!POWERPCB-V4.0-MILS! DESIGN DATABASE ASCII FILE 1.0			
Net			
Signal	Name 1		
Node 1	Node 2	Node 3	
Signal	Name 2		
Node 1	Node 2	Node 3	
"			
"			
"			
Signal	Name 3		
Node 1	Node 2	Node 3	

End

Table 15 lists the fields in the connection list format.

Table 15. Connection List Format Fields

Field	Description
Net	Section command for the connection list.
Signal	Heading for each net. Starts the line containing the name of each individual net.
Name	Name of the net. This is required for each net. The maximum netname length is 47 characters. You can use any alphanumeric characters except brackets ({ }), asterisks (*), and spaces.
Node	Part reference designator and pad/pin number.

Notes

- A To-From connection represents the electrical connection between two or more physical nodes (part pin/pads).
- A node consists of two items of data.
- Use a space to separate two nodes. Within a node, use a period to separate the reference designator and the pad number.

Reference Designator Format

The reference designator entry names a part. You can use up to fifteen alphanumeric characters; however, the first character must be alphabetical.

Signal Statements and To-From Signal Names

A *Signal* statement and name are used to assign a signal name for a To-From tree and to identify electrically different To-From trees. A *Signal* statement must exist between each successive To-From tree/net. If a *Signal* statement is not present, the two trees are tied.

The signal name is from the logic diagram, such as SIGA, CLK, +5V, and GND. If the signal is unnamed, use the *Signal* statement to identify the connection and supply the name. The maximum signal name length is 47 characters. You can use any alphanumeric characters except space, tab, comma (,), period (.), braces ({}), asterisks (*), and ampersands (&).

For example:

```
*Signal* D0
```

```
U1.3 R1.1 U2.1      A 3-node To-From connection tree named D0.
```

Discrete Component Pad Numbering

Discrete components do not normally have pin numbers on the logic diagram. They must, however, have a pin number in the database.

Tying Nets Together

In logic diagram documentation, parts of a multiple-node connection tree can be depicted anywhere, on all sheets, and at various locations on a sheet. In preparing your To-From list, you do not need to enter the entire tree as one continuous entry. If you use a common signal name for each partial connection, you can enter the different parts of a common connection at different locations of the To-From list. SailWind Layout will automatically tie the partial nets together.

To tie all entries in the To-From list give them the same signal name. For example:

Signal	D0
U1.3	R2.1 U1.6
Signal	D0
U5.2	R6.1
Signal	D0
U5.8	

results in:

Signal D0
U1.3 R2.1 U1.6 U5.2 R6.1 U5.8

Use the common signal name technique (above) to enter a long multiple node connection tree that requires several lines of data in the To-From list. Use a signal name that is common on all lines of data.

Abbreviated Notations

Certain type connection trees that have a common pin, such as power trees, ground trees, and memory trees, can use an abbreviated or shorthand format for entering the tree.

This abbreviated format uses the format X{F-L}.P and is interpreted as shown in [Table 16](#).

Table 16. Abbreviated Notations

Shorthand	Description
X	Reference designator letter

Table 16. Abbreviated Notations (continued)

Shorthand	Description
F	First reference designator number of a continuous series of reference designations
L	Last reference designator number in the series
{ }	Abbreviated notation symbol
P	Common pin/pad number

A dash (-) separates F and L, a period (.) precedes P. For example, the following results in a common tree for GND, which is read U1 through U50, pin 7. SailWind Layout expands the abbreviated format into the full GND connection tree:

```
*Signal*    GND 50
```

```
U{1-50}.7
```

Width

The Width entry is in thousandths of an inch. The first entry of a net establishes the width of all succeeding nets until the width changes. Therefore, arrange the Connection lists by width.

Sample Connection List

The automatically generated power and ground connection list, generated from the SIGPIN data contained in the part type information will not contain power and ground nodes not identified in the part type data. Examples of this are power and ground pins of an I/O connector, the grounding of spare gates of an IC, and so on. The designer must enter this data in the power and ground net.

```
!PADS-POWERPCB-V4.0-MILS! DESIGN DATABASE ASCII FILE 1.0
```

```
*Net*
```

```
*Signal*    D0
```

```
P1.1 U1.1 U2.3 U4.5 U6.1 U7.2
```

```
*Signal     D1
```

```
U1.2 U2.4 U3.3
```

```
*Signal*    D2
```

U1.6 U2.5 U3.4

Signal GND

U1.7 U2.7 U3.7 U4.7 P1.7

U8.7 U9.7 U10.7

Signal PWR

U1.14 U2.14 U3.14 U4.14 P1.14

U8.14 U9.14 U10.14

End

Connection List Error Messages

As the connection list ASCII file is read and merged with the parts list, it is checked for inconsistencies. If errors are found, the line of data is not accepted.

You may encounter one or more of the following error messages:

Error Message	Description
The following line of connection data contains a node using P1; however, the parts list did not include P1.	The following line of connection data contains a node using P1; however, the parts list did not include P1.
Bad *Signal data line	The line does not contain a signal name.
Mixing Nets: Node 1 Node 2 Node 3 Node 3 Node 4 Node 5	Two different nets have a common node (node 3).
Bad *Connections ASCII Data Line Format U20.A U20.5	The line of data is not in the required format. Alphanumeric pin/pad identifications are not allowed unless they are predefined in the part type file.

Chapter 3

Macros and the Macro Language

Macros simplify redundant activities. Record any set of procedural steps for replay as a single action - such as setting preferences and settings. You can nest macros inside other macros.

[Macros](#)

[Design Work Recordings in Macro Files](#)

[Replay Recorded Macros](#)

[Assigning Shortcut Keys to Macros](#)

[Adding Macros to Toolbars and Menus](#)

[Macro Editing and Debugging](#)

[The Macro Language](#)

Macros

You can create one or more macros as a means of performing redundant activities by recording any set of procedural steps for replay. You can replay a macro as a single action - such as setting preferences and settings. You can also nest macros inside other macros.

Dialog box actions are recorded as results rather than actions, so when you replay, you do not see the dialog boxes in the replay process. Because of this you cannot create a macro that stops on an open dialog box; it must follow through to some result or action.

The Macro editor supports standard editing features. It includes unlimited Undo/Redo operations, syntax highlighting, and provides context-sensitive help on the Macro language. You can also open multiple macros in the macro editor. You can move between open macros and edit them.

The embedded Macro debugger lets you:

- Start, pause, and stop macro execution.
- Insert breakpoints into the macro for step-by-step execution.
- Report run-time errors.

[Macro Text Colors](#)

[Macro Editing Shortcuts](#)

Macro Text Colors

Text colors in the macro editor represent certain features or functions.

Table 17. Text Color Representations

Color	Meaning
Blue	Basic Keywords
Black	User Variables
Cyan	Basic Functions
Purple	Automation Objects or Members
Red	Errors
Green	Comments

Macro Editing Shortcuts

Editing macros requires the use of standard Windows editing functions. Use the mouse to select areas of text.

[Table 18](#) shows the default shortcut keys for macro editing functions.

Table 18. Default Macro Editing Shortcuts

Command	Description
Up, Down arrows	Move the insertion point one line up or down.
Left, Right arrows	Move the insertion point one character left or right.
Home, End	Move the insertion point to the beginning or the end of the line.
Page Up, Page Down	Moves the insertion point one page up or one page down.
Ctrl+Home	Moves the insertion point to the top of a file.
Ctrl+End	Moves the insertion point to the end of the file.
Ctrl+Left, Ctrl+Right	Moves the insertion point one word left or right.
Shift+pointer move command	Moves the text between the start and end insertion points.
Ctrl+C, Ctrl+Ins	Copies text selected in the Edit area to the Clipboard.
Ctrl+V, Shift+Ins	Pastes the contents of the Clipboard to the Edit area.
Ctrl+X, Delete	Cuts selected text from the Edit area to the Clipboard.
Alt+Backspace, Ctrl+Z	Undoes the last edit. This is a multilevel undo. For more information, see “Undo and redo” in SailWind Router Help
Ctrl+Shift+Z	Restores the edit rescinded by the Undo command. For more information, see “Undo and redo” in SailWind Router Help
Ctrl+A	Selects all.
Ctrl+O	Opens the macro.
Ctrl+S	Saves the current macro.
Ctrl+P	Prints the current macro.

Design Work Recordings in Macro Files

You can record your work to replay and reuse by generating macros in one of two ways.

[Recording a Macro of an Entire Software Session](#)

[Recording a Macro of Selective Actions in the Design Area](#)

Recording a Macro of an Entire Software Session

You can add a command line switch to the shortcut used to launch the software to activate a feature that records your entire software session in a macro file.

Procedure

1. Click the **Start menu > PADS [version] > SailWind Layout** menu item.
2. In the SailWind Layout Properties dialog box, click the **Shortcut** tab and click in the Target box.
3. Press **End**.
4. At the end of the existing shortcut, type the following: **-log:[path and file name]**. Make sure to type a space between “*SailWindPCB.exe*” and the “-log” command. Paths or filenames with a space need double quotes. For example, -log:“C:\SailWind Projects\mylog.txt”.
5. Click **OK**.

Results

When you run the program, a log is created in the specified location using the specified name.

A log file records all of your actions during a session. This file is useful when trying to reproduce problems during debugging. Events are logged to the log file and to the macro editor. Each time you create a log, the previous .log file is saved to a .bak file, so you can retrieve a record of your actions up to the last recorded session. The macro .log file is written to the folder specified in the General tab of the Options dialog box. The default folder is \SailWind Projects.



Tip

: If you use the SailWind Router Link from SailWind Layout, the .log file is not written.

Recording a Macro of Selective Actions in the Design Area

You can record one or more macros of your actions in the design area.

Procedure

1. In the Output Window, on the **Macro** tab, click the **New** button. New macros receive a name of Macro#, where # is a numeric sequence such as Macro1 or Macro2.
2. Set the mouse move preferences. Mouse movements are recorded in macros. You can set mouse movements to be compressed or uncompressed, and choose either relative or absolute movements.

- **Compressed and Uncompressed Mouse Mode**—With the “Compress mouse moves” option selected, only the start point and end point of a mouse movement are recorded. Compression is recommended under most circumstances because it significantly reduces the size of your macro file.

When the “Compress mouse moves” option is not selected, it records the intermediate coordinates between the start and end points. Recording intermediate mouse movements increases the file size, but records coordinate information if required for a special application.

- **Relative and Absolute Mouse Mode**—With the “Relative mouse moves” option selected, coordinates are recorded incrementally. In other words, each starting point is recorded as x=0, y=0.

When the “Relative mouse moves” option is not selected, the absolute grid coordinates are recorded. In other words, the exact coordinates of the design are recorded.

3. On the Macro tab toolbar, click the **Record** button.
4. Perform the keystrokes, commands, and mouse clicks to include in the macro.
5. When finished recording, on the Macro tab toolbar, click the **Stop** button.
6. Click the **Save** button.
7. In the standard Windows Save As dialog box, enter a filename, if desired, and click **Save**.

Replay Recorded Macros

You can play back an existing macro using Run. Run also resumes the playback of a paused macro. When you play a macro, you cannot use the mouse in the workspace.

You can play back a macro using either of the following methods:

[Running a Macro When You Start the Program](#)

[Running Macros From Within the Software](#)

Running a Macro When You Start the Program

You can run a macro at the software launch when you start SailWind Layout.

Procedure

1. Click the **Start menu > PADS [version] > SailWind Layout** menu item.
2. In the SailWind Layout Properties dialog box, click the **Shortcut** tab and click in the Target box.
3. Press **End**.
4. At the end of the existing shortcut, type the following: **-run=[path and file name]**. Ensure you type a space between "SailWindPCB.exe" and the "-run" command. You can use a slash instead of the hyphen (for example, /run=mymacro.mcr). When there are spaces in paths or file names you must use double quotes around the full path. For example,

```
-run="C:\SailWind Projects\Samples\mymacro.mcr"
```

5. Click **OK**.

Results

When you launch the program, the specified macro runs as soon as the program starts.

Running Macros From Within the Software

You can load one or more macros into the Macro tab of the Output Window and run them when desired.

Procedure

1. In the **Output Window**, in the **Macro** tab, click the **Open** button.
2. In the Open File dialog box, select the macro file (.mcr) to open and click **Open**. The macro opens in the Macro Editor. Recent macros can be found on the Tools menu. Click Macros and then choose a macro.
3. Repeat these steps to open additional macros if needed. The macro editor supports nested macros. To switch between multiple macros loaded into the Macro tab, click the macro you want to view and the macro recording appears in the editor area.
4. On the Macro tab toolbar, click the **Run** button.

- To pause the macro at any time, on the Macro tab toolbar, click the **Pause** button. Click the Run button to resume the macro.
- To stop the macro at any time, click **Stop**. You cannot resume the macro after you have stopped it. If you click Run, the macro starts from the beginning.

Results

If run-time errors occur, the macro debugger switches to step-by-step mode and displays a detailed message on the status bar. The instruction pointer is set in the gutter on the line that produced the error. After fixing the error, you can resume playback of the macro.

Assigning Shortcut Keys to Macros

You can create a shortcut key that executes a macro.

Restrictions and Limitations

In order to assign a macro to a shortcut key, the macro command file (.mcr) must already exist. You can create a macro by recording it or scripting it in Macro language. For more information, see [Design Work Recordings in Macro Files](#).

Procedure

1. On the **Tools** menu, click **Customize**.
2. In the Customize dialog box, click the **Macro Files** tab.
3. In the Macro Files tab, in the Macro Command Files area, click the **New** button.
4. In the Open macro file dialog box, select the macro file you want. Then click **Open**. The PADS tool loads the macro and makes it available for use as a command (the check box to the left of the macro name is selected).

To close the macro file or make it unavailable in the Customize dialog box, clear the check box next to the macro name.
5. To assign the macro to a shortcut key, click the **Keyboard and Mouse** tab.
6. In the Keyboard and Mouse tab, in the Mode list, select "All modes."
7. In the Commands area, double-click "Macros" to expand the list of available macros. Then select the macro you want.
8. In the Current Shortcuts area, click the **New** button.
9. In the Assign shortcut dialog box, create one of the following types of shortcut:

- To assign shortcut keys, select the “Press new shortcut key(s)” option, and then press the keys that you want to use. For detailed information about rules and restrictions for creating shortcut keys, see Rules and Restrictions for Key Sequences in the SailWind Layout User’s Guide.

As you enter the new shortcut, similar shortcuts appear in the “Similar shortcuts assigned to other commands” box. This helps you to avoid creating a new shortcut that conflicts with an existing shortcut.

- To create a mouse action, click the “or select a pointer event” option, and then select from a combination of mouse actions, mouse button events, and modifier keys.
10. Click **OK** in the Assign shortcut dialog box. The new shortcut appears in the Current shortcuts box of the Keyboard and Mouse tab.
 11. When finished, close the Customize dialog box.

Adding Macros to Toolbars and Menus

You can create commands from macro files and add them to toolbars and menus.

Restrictions and Limitations

To create a command from a macro command file, the macro command file (.mcr) must already exist. Create a macro by recording it in a PADS tool or scripting it in Macro language. For more information, see [Design Work Recordings in Macro Files](#).

Procedure

1. On the Tools menu, click **Customize**.
2. In the Customize dialog box, click the **Macro Files** tab.
3. In the Macro Files tab, click the **New** button.
4. In the Open macro file dialog box, select the macro file you want to use as a command. Then click **Open**. The macro is loaded and available for use as a command (the check box to the left of the macro name is selected).

To close the macro file, or make it unavailable in the Customize dialog box, clear the check box next to the macro name.

5. In the Commands tab, in the Categories list, select “Macros.”
6. In the Commands list, select the macro you want and drag it to the toolbar or menu.

To remove an item from a toolbar or menu (while the Customize dialog box is open), click the item and drag it outside the toolbar or menu.
7. When you have finished adding commands, click **Close**.

Related Topics

[Assigning Shortcut Keys to Macros](#)

Macro Editing and Debugging

You can edit a macro using the debugging tools within the Status tab. You can also copy the macro and paste it into an external text editor where you can view or edit more of the file at a time without needing to continually scroll in the Status tab.

You can copy or cut selected text to the Clipboard. You can also paste the selection from the Clipboard into the text window. You can paste text from the Clipboard into other applications. You can also switch between open macros to edit multiple macros.

When playing back a macro, you can run it step-by-step, or to a certain location in the script. To perform these debugging tasks, insert breakpoints in the macro at the points at which you want the macro to pause.

[Toggling Breakpoints](#)

[Macro Debugging Tools](#)

Toggling Breakpoints

The ability to set or remove breakpoints is useful when you debug a macro.

Procedure

1. Place the cursor on the line in which to add a breakpoint.
2. On the Macro tab toolbar, click the **Toggle Breakpoint** button. You might find it easier to simply click in the gutter on the line to toggle a breakpoint on or off. A breakpoint marker appears in the gutter area.

Results

When the macro engine encounters a breakpoint while playing back a macro, it pauses the macro. The next line in the macro is marked with the instruction pointer.

Macro Debugging Tools

You can debug macros after inserting breakpoints.

Use the following toolbar buttons for debugging:

- **Step Over**—To play a single line of the macro.
- **Step Into**—To perform a subroutine call on the current line.
- **Step Out**—To return from the subroutine to the point from which it was called.
- **Step to Cursor**—To play back a macro to a point.
- **Run**—To continue the execution from the current point.

Related Topics

[The Macro Language](#)

The Macro Language

The macro language of this program is similar to standard Visual Basic Script (VBScript) language.

It supports most of the VBScript features including the following:

- [Variables](#) and arrays of variables
- [Expressions](#) — The full set of standard arithmetic and Boolean Expressions
- [Operators](#)
- [Statements](#)
- [Functions](#)
- [Automation Support](#) for internal and external automation objects
- [Objects](#), properties, and methods
- [Dialog Box Controls](#)
- [Internal Macro Objects](#)

[Variables](#)

[Expressions](#)

[Operators](#)

[Statements](#)

[Functions](#)

[Automation Support](#)

[Dialog Box Controls](#)

[Internal Macro Objects](#)

Variables

The macro engine of this program supports variables, which can be Null.

Variables can also contain values of the following types:

- [Numeric](#)
- [Logical](#)
- [String](#)

- [Double](#)
- [Objects](#)

Value types are converted to each other according to these rules, see [Table 19](#).

Table 19. & Operator Arguments

From...To...	Logical	Numeric	String
Null	False	0	Empty string
Logical	None	0 if False, 1 if True	0 if False, 1 if True
Numeric	False if 0, True if not 0	None	String representation of the numeric value
String	Converted to Numeric first	If the beginning of the string can be interpreted as a number, then the value of the number is used. Otherwise it is 0.	None

Numeric

The Numeric value represents a floating-point number.



Note:

The Numeric and [String](#) value types are interchangeable; they automatically convert into one another upon assignment.

Logical

The Logical value can be True or False.

String

The String value represents a character string.

Double

Represents numeric value.

The double and string types are interchangeable; that is, they automatically convert into one another upon assignment.

Objects

Objects represent complex entities that are handled through an interface consisting of methods and properties. Objects are different from [Numeric](#) and [String](#) value types.

Objects may be of two types:

- **Macro objects:** Internal objects that are handled using the macro engine vocabularies, and may or may not have the Automation interface.
- **Automation objects:** Internal or external objects that are handled using Automation.

Syntax

The syntax for both object types is the same:

```
Object.Method arg1, ..., argn
```

```
var = Object.Method( arg1, ..., argn )
```

Related Topics

[Str](#)

Expressions

The macro engine of this program uses either numeric or string expressions.

- **Numeric:** Any expression that can be evaluated as a number. Elements of a numeric expression can include any combination of keywords, variables, constants, and operators that result in a number.
- **String:** Any expression that evaluates a sequence of adjacent characters. Elements of a string expression can include a string, a string literal, or a string variable.

Operators

The Macro engine of this program allows use of certain operators.

- & Operator
- * Operator
- + Operator
- / Operator
- Operator
- = Operator
- ^ Operator
- And Operator
- Comparison Operators
- Mod Operator
- Not Operator
- Or Operator
- Xor Operator

& Operator

Forces string concatenation of two expressions.

Syntax

```
result = expression1 & expression2
```

Arguments

result	Required Any numeric variable
expression1	Required Any expression When the expression is not a string, it is converted to a string.
expression2	Required Any expression When the expression is not a string, it is converted to a string.

The & operator has these arguments:

Examples

```
S="abc" & "123"
```

* Operator

Multiplies two numbers.

Syntax

```
result = number1 * number2
```

Arguments

The * operator has these arguments::

result	Required Any numeric variable
number1	Required Any expression. If the expression value is not numeric, it is converted to a numeric value.
number2	Required Any expression. If the expression value is not numeric, it is converted to a numeric value.

Examples

```
X = y * z
```

+ Operator

Sums two numbers.

Syntax

```
result = expression1 + expression2
```

Arguments

The + operator syntax has these arguments:

result	Required Any numeric variable
expression1	Required Any expression
expression2	Required Any expression

**Note:**

When you use the + operator, you may not be able to determine whether addition or string concatenation will occur. To force string concatenation, use the & operator instead. This will eliminate ambiguity and provide self-documenting code.

The following table describes the behavior of the + operator for the three combinations of types.

Arguments	Action
Both expressions are numeric	Add
Both expressions are string	Concatenate
One expression is numeric and the other is a string	Add

Examples

```
x = y + z
```

/ Operator

Divides one number by a second number and returns a floating-point result.

Syntax

```
result = number1 / number2
```

Arguments

The / operator has these arguments:

result	Required Any numeric variable
number1	Required Any expression. If the expression value is not numeric, it is converted to a numeric value.
number2	Required Any expression. If the expression value is not numeric, it is converted to a numeric value.

Examples

```
x = y/z
```

- Operator

Finds the difference between two numbers or indicates the negative value of a numeric expression.

Syntax

```
result = number1 - number2
```

The - operator is the arithmetic subtraction operator, which finds the difference between two numbers. Alternatively, you can use the operator as follows:

```
- number
```

The - operator is the unary negation operator which indicates the negative value of an expression.

Arguments

The - operator has these arguments:

result	Required Any numeric variable
number1	Required Any expression. If the expression value is not numeric, it is converted to a numeric value.
number2	Required Any expression. If the expression value is not numeric, it is converted to a numeric value.

Examples

```
x = y - z
```

```
-x
```

= Operator

Assigns a value to a variable or property.

Syntax

variable = value

Arguments

The = operator has these arguments:

variable	Can only be a variable or a writable property. Can be a simple scalar variable or an element of an array.
Value	Any numeric expression, string expression, literal, or constant

Examples

```
a = 1
```


^ Operator

Raises a number to the power of an exponent.

Syntax

```
result = number ^ exponent
```

Arguments

The ^ operator has these arguments:

result	Required Any numeric variable
number	Required Any expression
exponent	Required Any numeric expression A number can be negative only if the exponent is an integer.

When more than one exponentiation is performed in a single expression, the ^ operator is evaluated as it is encountered from left to right.

Examples

```
x = y ^ z
```

And Operator

Performs a logical conjunction of two expressions.

Syntax

result = expression1 **And** expression2

Arguments

The And operator has these arguments:

result	Required Any numeric variable
expression1	Required Any expression. If the expression value is not logical, it is converted to a logical value.
expression2	Required Any expression. If the expression value is not logical, it is converted to a logical value.

The following table illustrates how the result is determine.

If expression1 is	And expression2 is	The result is
True	True	True
True	False	False
False	True	False
False	False	False

Examples

```
a = b And c
```

Comparison Operators

Compares expressions.

Syntax

```
result = expression1 comparisonoperator expression2
```

Arguments

Comparison operators have these arguments:

result	Required Any variable
expression1	Required Any expression
expression2	Required Any expression
comparisonoperator	Required Can be any comparison operator. For more information, see the table below.

The following table lists the comparison operators and the conditions that determine whether result is True, False, or Null.

Comparison Operator	True if	False if
< (Less than)	<i>expression1 < expression2</i>	<i>expression1 >= expression2</i>
<= (Less than or equal to)	<i>expression1 <= expression2</i>	<i>expression1 > expression2</i>
> (Greater than)	<i>expression1 > expression2</i>	<i>expression1 <= expression2</i>
>= (Greater than or equal to)	<i>expression1 >= expression2</i>	<i>expression1 < expression2</i>
= (Equal to)	<i>expression1 = expression2</i>	<i>expression1 <> expression2</i>
<> (Not equal to)	<i>expression1 <> expression2</i>	<i>expression1 = expression2</i>

Examples

```
b = 1 > 2
```

Mod Operator

Divides one number by a second number and returns only the remainder. The modulus (remainder) operator rounds floating-point numbers to integers.

Syntax

```
result = number1 Mod number2
```

Arguments

The Mod operator has these arguments:

result	Required Any numeric variable
number1	Required Any numeric expression
number2	Required Any numeric expression

Examples

```
x = y Mod z
```

Not Operator

Performs a logical negation on an expression.

Syntax

```
result = Not expression
```

Arguments

The Not operator has these arguments:

result	Required Any numeric variable
expression	Required Any expression. If the expression value is not logical, it is converted to a logical value.

The following table illustrates how result is determined:

If expression is	Then result is
True	False
False	True

Examples

```
x = Not y
```

Or Operator

Performs a logical disjunction on two expressions.

Syntax

```
result = expression1 or expression2
```

Arguments

The Or operator has these arguments:

result	Required Any numeric variable
expression1	Required Any expression. If the expression value is not logical, it is converted to a logical value.
expression2	Required Any expression. If the expression value is not logical, it is converted to a logical value.

The following table illustrates how result is determined:

If expression1 is	And expression2 is	Then result is
True	True	True
True	False	True
False	True	True
False	False	False

Examples

```
x = y or z
```

Xor Operator

Performs a logical exclusion on two expressions.

Syntax

```
[result =] expression1 Xor expression2
```

Arguments

The Xor operator has these arguments:

result	Optional Any numeric variable
expression1	Required Any expression. If the expression value is not logical, it is converted to a logical value.
expression2	Required Any expression. If the expression value is not logical, it is converted to a logical value.

The following table illustrates how result is determined:

If expression1 is	And expression2 is	Then result is
True	True	False
True	False	True
False	True	True
False	False	False

Examples

```
x = y Xor z
```

Statements

The macro engine of this program supports the certain VBScript and other statements.

- Call
- Close
- Dim
- Do...Loop
- For-Next
- Function
- If...Then...Else statement
- Input #
- Modal
- Open
- Print #
- ReDim
- Set
- Sub
- While...Wend
- Width #

Call

Transfers control to a sub procedure or function procedure.

Syntax

```
[Call] name [argumentlist]
```

When you specify the Call keyword to call a procedure that requires arguments, you must enclose argumentlist in parentheses. See example below.

Arguments

The Call statement has these arguments:

Call	Optional keyword You are not required to use the Call keyword when calling a procedure. If you omit the Call keyword, you must also omit the parentheses around argumentlist. If you use either the Call syntax to call any intrinsic or user-defined function, the function's return value is discarded.
name	Required Name of the procedure to call
argumentlist	Optional Comma-delimited list of variables, arrays, or expressions to pass to the procedure.

Examples

```
Call MyProc(0)
```

Related Topics

[Statements](#)

Close

Concludes input/output (I/O) to a file opened using the Open statement. When you close files that were opened for Output or Append, the final buffer of output is written to the operating system buffer for that file; All buffer space associated with the closed file is released; and the association of a file with its file number ends.

Syntax

```
close [filenumberlist]
```

Arguments

The Close statement has this argument:

filenumberlist	Optional Can be one or more file numbers using the following syntax, where <i>filenumber</i> is any valid file number: [[#]filenumber] [, [#]filenumber] . . .

If you omit filenumberlist, all active files opened by the Open statement are closed.

Examples

```
close #1
```

Related Topics

[Statements](#)

Dim

Declares variables and allocates storage space.

Syntax

```
Dim varname([[subscripts]]) [,varname([[subscripts]])] . . .
```

Arguments

The Dim statement has these arguments:

varname	Required The variable name follows standard variable-naming conventions
subscripts	Optional Dimensions of an array variable The subscripts argument uses the following syntax: [lower To] upper [, [lower To] upper] . . . When not explicitly stated, the lower bound is zero.

You can also use the Dim statement, with empty parentheses, to declare a dynamic array. After declaring a dynamic array, use the [ReDim](#) statement to define the number of dimensions and elements in the array.

Examples

```
Dim x(10), y(20)
```

Related Topics

[Statements](#)

Do...Loop

Repeats a block of statements while a condition is True or until a condition becomes True.

Syntax

```
Do [{While | Until} condition]  
[statements]  
Exit Do  
[statements]  
Loop
```

Alternatively, you can use the following syntax:

```
Do  
[statements]  
Exit Do  
[statements]  
Loop [{While | Until} condition]
```

Arguments

The Do Loop statement has these arguments:

condition	Optional One or more of the following two types of expressions: • A numeric expression that evaluates to True or False • A string expression that evaluates to True or False When <i>condition</i> is Null, <i>condition</i> is treated as False.
statements	One or more statements that are repeated while, or until, <i>condition</i> is True

You may place any number of Exit Do statements anywhere in the Do...Loop statement as an alternative way to exit a Do Loop statement. Exit Do is often used after evaluating some condition, in which case the Exit Do statement transfers control to the statement immediately following the Loop.

When used within nested Do Loop statements, Exit Do transfers control to the loop that is nested one level above the loop where Exit Do occurs.

Examples

```
Do while i < 10  
i = i + 1  
loop
```

Related Topics

[Statements](#)

For-Next

Repeats a group of statements a specified number of times.

Syntax

```
For counter = start To end [Step step]
[statements]
Exit For
[statements]
Next [counter]
```

Arguments

The For-Next statement has these arguments:

counter	Required Numeric variable used as a loop counter. <i>Counter</i> cannot be a Boolean element or an array element.
start	Required Initial value of <i>counter</i>
end	Required Final value of <i>counter</i>
step	Optional The number by which <i>counter</i> is incremented each time control passes through the loop. If not specified, step defaults to one.
statements	Optional One or more statements between For and Next that are executed the <i>counter</i> -specified number of times

The *step* argument can be either positive or negative. The value of step determines loop processing as described in the following table:

Value	Loop executes if
Positive or zero	counter <= end
Negative	counter >= end

After all statements in the loop execute, *step* is added to *counter*. At this point, either the statements in the loop execute again (based on the same test that caused the loop to execute initially), or the loop is exited and execution continues with the statement following the Next statement.

You can place any number of Exit For statements anywhere in the loop as alternative ways to exit. Exit For is often used after evaluating a condition to transfer control to the statement immediately following Next.

You can nest For-Next statements by placing one For-Next statement within another. Give each statement a unique variable name as its *counter*. The following construct is correct:

```
For A = 1 To 10
  For B = 2 To 20
    For C = 3 To 30
      ...
    Next C
  Next B
Next A
```

Examples

```
for i = 0 to 10 step 2
s = sti
next i
```

Related Topics

[Statements](#)

Function

Declares the name, arguments, and code that form the body of a Function procedure.

Like a sub procedure, a function procedure is a separate procedure that can take arguments, perform a series of statements, and change the values of its arguments. However, unlike a sub procedure, a function procedure can be used on the right side of an expression in the same way you use any intrinsic function, such as Sqr, Cos, or Chr, when you want to use the value returned by the function.

There are two categories of variables you can use in function procedures:

- **Explicitly declared variables within the procedure:** These variables are always local to the procedure and use the Dim statement or the equivalent. The values of local variables in a function are not preserved between calls to the procedure.
- **Not explicitly declared variables within the procedure:** These variables are also local, unless they are explicitly declared at some higher level outside the procedure.

Usage

```
Function name [(arglist)]
```

```
[statements]
```

```
[name = expression]
```

[Exit Function]

[statements]

[name = expression]

End Function

The Exit Function statement causes an immediate exit from a function procedure. Program execution continues with the statement that follows the statement that called the function procedure. You can add any number of Exit Function statements in a function procedure.

Arguments

The Function statement has these arguments:

name	Required Name of the function procedure Follows standard variable-naming conventions. To return a value from a function, assign a value to the function name. You can assign values to function names anywhere in the procedure. If no value is assigned to name, the procedure returns an empty value.
arglist	Optional List of variables representing arguments that are passed to the function procedure when the procedure is called. Use commas to separate multiple variables.
statements	Optional Any group of statements to execute within the body of the function procedure.
expression	Return value of the function procedure.

The arglist argument has the following syntax:

[ByVal | ByRef] varname[()]

The following table describes the arglist syntax elements:

Part	Description
<i>ByVal</i>	Indicates that the argument is passed by value.
<i>ByRef</i>	Indicates that the argument is passed by reference.
<i>Varname</i>	Name of the variable representing the argument.

	Follows standard variable naming conventions.	

You cannot define a function procedure inside any other procedure such as a sub procedure or another function procedure.

For specific information about calling a function procedure, see the [Call](#) statement.

Examples

The following example shows how to assign a return value to a function named Example. In this case, False is assigned to the name to indicate that a certain condition is not met.

```
Function Example()
```

```
...
```

```
' Value not found. Return False.
```

```
  If ConditionNotMet Then
```

```
    Example = False
```

```
      Exit Function
```

```
    End If
```

```
...
```

```
    Example = True
```

```
End Function
```

Related Topics

[Statements](#)

If...Then...Else statement

May execute a group of statements, based on the value of an expression.

Syntax

```
If condition Then [statements] [Else [elsestatements]]
```

You can write the statement in block Syntax as follows:

```
If condition Then  
[statements]  
[ElseIf condition-n Then  
[elseifstatements]] ...  
[Else  
[elsestatements]]  
End If
```

Arguments

The If...Then...Else statement has these arguments::

condition	Required Any expression. If the expression is not Logical, it is converted to Logical.
statements	Optional in block form; required in single-line form that has no Else clause One or more statements separated by colons; executed when condition is True.
condition-n	Optional Any logical expression. If the expression is not Logical, it is converted into a Logical expression.
elseifstatements	Optional One or more <i>statements</i> executed when associated <i>condition-n</i> is True
elsestatements	Optional One or more <i>statements</i> executed when no previous condition or when <i>condition-n</i> is True

You can use the single-line syntax for short, simple tests. For more structure and flexibility use the block syntax. The block syntax can also be easier to read, maintain, and debug.

With the single-line syntax, you can execute multiple statements as the result of an If...Then decision. All statements must be on the same line and separated by colons, as in the following statement:

```
If A > 10 Then A = A + 1 : B = B + A : C = C + B
```

A block If statement must be the first statement on a line. The Else, Elself, and End If parts of the statement can be preceded by only a line number or a line label. The block If statement must end with an End If statement.

To determine whether or not a statement is a block If statement, examine what follows the Then keyword. If anything other than a comment appears after Then on the same line, the statement is treated as a single-line If statement.

The Else and Elself clauses are both optional. You can have as many Elself clauses as you want in a block If statement, but none can appear after an Else clause. Block If statements can be nested.

During the execution of a block If statement, *condition* is tested. When *condition* is True, the statements following Then are executed. When condition is False, any Elself condition is evaluated in turn. When a True condition is found, the statements immediately following the associated Then are executed. If none of the Elself conditions are True (or if there are no Elself clauses), the statements following Else are executed. After executing the statements following Then or Else, execution continues with the statement following End If.

Examples

```
If x < y then x = y
```

and

```
If x < y then  
x = y  
End if
```

Related Topics

[Statements](#)

Input

Reads data from an open text file and assigns the data to variables. Use this statement only with files opened in Input mode. When read, string or numeric data is assigned to variables without modification.

Syntax

```
Input #filename, varlist
```

Arguments

The Input # statement has these arguments:

filename	Required Can be any valid file number
varlist	Required Can be a comma-delimited list of variables that are assigned values read from the file. This cannot be an array or object variable. However, you may use variables that describe an element of an array.

Related Topics

[Statements](#)

Modal

Opens access to a variable. While a modal dialog is open in a PADS macro, access to all variables outside the open dialog is blocked—only controls within the open dialog are accessible. To make a variable accessible in the context of any open modal dialog, declare it “modal” at the beginning of the macro file.

Syntax

```
modal variablename
```

Arguments

The modal keyword has these arguments:

variablename	Required The name of the variable you want to make accessible.

Examples

modal docname

Open

Enables input and output to a file. You must open a file before performing any I/O operation on it. Open allocates an I/O buffer for the file and determines the mode of access to use with the buffer. If the file is already opened by another process and the specified type of access is not allowed, the Open operation fails and an error occurs.

Syntax

```
Open pathname For mode [Access access] [lock] As [#]filename  
[Len=reclength]
```

Arguments

The Open statement has these arguments:

pathname	Required String expression that specifies a filename. May also include folder and drive names.
mode	Required Keyword specifying the file access mode: Append, Binary, Input, Output, or Random. If mode is unspecified, the file is opened for Random access.
access	Optional Keyword specifying the operations permitted on the open file: Read, Write, or Read Write
lock	Optional Keyword specifying the operations restricted on the open file by other processes: Shared, Lock Read, Lock Write, and Lock Read Write

If the file specified by pathname does not exist, it is created when a file is opened for Append, Binary, Output, or Random access modes.

Examples

```
Open "C:\data.txt" for read as #1
```

Related Topics

[Statements](#)

Print

Writes formatted data to a sequential file.

Syntax

```
Print # filename, [outputlist]
```

Arguments

The Print # statement has these arguments:

filename	Required Any valid file number
outputlist	Optional An expression or list of expressions to print. Nothing is written to the file when <i>outputlist</i> is empty. However, when <i>outputlist</i> is Null, Null is written to the file.

If you omit *outputlist* and include only a list separator after *filename*, a blank line prints to the file.

You can separate multiple expressions with either a space or a semicolon. A space has the same effect as a semicolon.

Examples

```
print #1, a, b, c
```

outputlist has the following syntax:

```
[{Spc(n) | Tab[(n)]}] [expression] [charpos]
```

Table 20 describes the outputlist syntax elements:

Table 20. Print # Statement outputlist Syntax

Setting	Description
Spc (n)	Inserts space characters in the output, where n is the number of spaces to insert.
Tab (n)	Positions the insertion point at an absolute column number, where n is the column number. Use Tab with no argument to position the insertion point at the beginning of the next print zone. Because Print # writes an image of the data to the file, you must delimit the data so it prints correctly. If you use Tab with no arguments to move the print position to the next print zone, Print # also writes the spaces between print fields to the file.

Table 20. Print # Statement outputlist Syntax (continued)

expression	Numeric or string expressions to print. You can separate multiple expressions with either a space or a semicolon.
charpos	Specifies the insertion point for the next character. If you omit charpos, the next character prints on the next line. Use a semicolon to position the insertion point immediately after the last displayed character.

Data written with Print # is usually read from a file with Input #.

```
Print #1, a, Spc(3), b
```

Related Topics

[Statements](#)

ReDim

Reallocates storage space for dynamic array variables. Use ReDim to size a dynamic array that has already been declared using the Dim statement with empty parentheses (without dimension subscripts). You can use ReDim repeatedly to change the number of elements and dimensions in an array.



Note:

If you re-declare a dimension for an array variable whose size was explicitly specified in a Dim statement, an error occurs.

Syntax

```
ReDim [Preserve] varname(subscripts) [, varname(subscripts)] . . .
```

Arguments

The ReDim statement has these arguments:

Preserve	Optional Keyword used to preserve the data in an existing array when you change the size of the last dimension
varname	Required Name of the variable; follows standard variable-naming conventions
subscripts	Required Dimensions of an array variable The subscripts argument uses the following syntax: <pre>[lower To] upper [, [lower To] upper] . . .</pre> When not explicitly stated, lower bound is zero.



Note:

If you make an array smaller than it was, any data in the eliminated elements is lost.

When using *Preserve* you can resize only the last array dimension and you cannot change the number of dimensions. For example, if your array has only one dimension, you can resize that dimension because it is the last and only dimension.

However, if your array has two or more dimensions, you can change the size of only the last dimension while preserving the contents of the array. The following example shows how you can increase the size of the last dimension of a dynamic array without erasing existing data contained in the array:

```
ReDim X(10, 10, 10)
. . .
ReDim Preserve X(10, 10, 15)
```


When you use *Preserve* you can change the size of the array only by changing the upper bound. Changing the lower bound causes an error.

Examples

```
ReDim x(150)
```

Related Topics

[Statements](#)

Set

Assigns an object reference to a variable or property.

Syntax

```
Set objectvar = {objectexpression | Nothing}
```

Arguments

The Set statement has these arguments:

<i>objectvar</i>	Required Name of the variable, or property; follows standard variable-naming conventions
<i>objectexpression</i>	Required Expression, which consists of the name of an object, another declared variable of the same object type, or a function or method that returns an object of the same object type
Nothing	Optional Discontinues association of <i>objectvar</i> with any specific object. Assigning Nothing to <i>objectvar</i> releases all the system and memory resources associated with the previously referenced object when no other variable refers to it.

To be valid, ***objectvar*** must be of the same object type as the object being assigned to it.

The [Dim](#) and [ReDim](#) statements declare only the variable name, which refers to an object. No actual object is referred to unless the Set statement assigns a specific object.

When you use Set to assign an object reference to a variable, a reference to the object is created - not a copy of the object. More than one object variable can refer to the same object. Because such variables are references to the object rather than copies of the object, any change in the object is reflected in all variables that refer to it.

Examples

```
Set obj = application
```

Related Topics

[Statements](#)

Sub

Declares the name, arguments, and code that form the body of a sub procedure.

Like a function procedure, a sub procedure is a separate procedure that can take arguments, perform a series of statements, and change the value of its arguments. However, unlike a function procedure, which returns a value, a sub procedure cannot be used in an expression.

There are two categories of variables you can use in sub statements:

- **Explicitly declared variables within the procedure:** These variables are always local to the procedure and use the Dim statement or the equivalent. The value of local variables in a Sub procedure is not preserved between calls to the procedure.
- **Not explicitly declared variables within the procedure:** These variables are also local, unless they are explicitly declared at some higher level outside the procedure.

Usage

```
Sub name [(arglist)]
```

```
[statements]
```

```
[Exit Sub]
```

```
[statements]
```

```
End Sub
```

Arguments

The Sub statement has these arguments:

name	Required The Sub procedure name Follows standard variable-naming conventions.
arglist	Optional List of variables representing arguments that are passed to the sub procedure when the sub procedure is called. Use commas to separate multiple variables.
statements	Any group of statements to execute within the body of the sub procedure. The Exit Sub statement causes an immediate exit from a sub procedure. Program execution continues with the statement following the statement that called the sub procedure. Any number of Exit Sub statements can appear anywhere in a sub procedure.

Examples

The *arglist* argument has the following syntax:

```
[ByVal | ByRef] varname[( )]
```

[Table 21](#) describes the arglist syntax elements::

Table 21. Sub Statement arglist Syntax

Part	Description
<i>ByVal</i>	Indicates that the argument is passed by value.
<i>ByRef</i>	Indicates that the argument is passed by reference.
<i>varname</i>	Name of the variable representing the argument. Follows standard variable-naming conventions.



Note:

You cannot define a sub procedure inside any other procedure such as a function or another sub procedure.

For specific information about calling sub procedures, see the [Call](#) statement.

Related Topics

[Statements](#)

While...Wend

Executes a series of statements as long as a given condition is True.

Syntax

```
While condition
[statements]
Wend
```

Arguments

The While Wend statement has these arguments:

<i>condition</i>	Required Any expression. If the expression is not Logical, it is converted to a Logical type.
<i>statements</i>	Optional One or more statements executed while <i>condition</i> is True

If ***condition*** is True, all ***statements*** are executed until the Wend statement is encountered. Control then returns to the While statement and ***condition*** is again checked. If ***condition*** is still True, the process is repeated. If ***condition*** is not True, execution resumes with the statement following the Wend statement.

You can nest While...Wend statements to any level. Each Wend matches the most recent While statement.

Examples

```
While i < 10
i = i + 1
Wend
```

Related Topics

[Statements](#)

Width

Assigns an output line width to a file opened using the Open statement.

Syntax

```
width #filenumber, width
```

Arguments

The Width # statement has these arguments:

filenumber	Required Any valid file number
width	Required Numeric expression in the range 0 to 255, inclusive. Indicates how many characters appear on a line before a new line starts. If width equals 0, there is no limit to the length of a line. The default value for width is 0.

Examples

```
Width #2, 100
```

Related Topics

[Statements](#)

Functions

The macro engine of this program currently supports certain embedded functions.

Asc
Atn
Chr
Command
Cos
CreateObject
CurDir
Dir
DoEvents
Environ
Eof
Exp
GetObject
GetTmpFileName
InStr
InStrRev
Left
Len
Mid
MkDir
MoveFile
MsgBox
Right
Sin
Spc
Str
Tab
Val

Asc

This function returns an integer representing the character code that corresponds to the first letter in a string.

Syntax

```
Asc(string)
```

Arguments

The Atn function has this argument:

string	Required Any valid string expression If the string contains no characters, a run-time error occurs.

Examples

```
i = Asc("abc")
```

Atn

This function returns a Double specifying the arctangent of a number.

The range of the result is - pi/2 to pi/2 radians. To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

The Atn function takes the ratio of two sides of a right triangle and returns the corresponding angle in radians. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

Atn is the inverse trigonometric function of Tan, which takes an angle as its argument and returns the ratio of two sides of a right triangle. Do not confuse Atn with cotangent, which is the inverse of a tangent (1/tangent).

Usage

```
Atn(number)
```

Arguments

The Atn function has this argument:

number	Required Any expression. If the expression is not Logical, it is converted to a Logical type.

Examples

```
f = Atn(2)
```

Chr

This function returns a string containing the character associated with the specified character code.

Syntax

```
chr ( charcode )
```

Arguments

The Chr function has this argument:

charcode	Required A long that identifies a character. Values from 0 to 31 are standard, nonprintable ASCII codes. For example, Chr(10) returns a linefeed character. The normal range for charcode is 0 to 255, inclusive.

Examples

```
c = chr(64)
```

Command

This function returns the command line used to start the program, including the path to the executable file and any subsequent arguments.

Syntax

Command

When you launch the program from the command line, the command line is available to Macro scripts.

Arguments

None

Examples

Assuming that the program is launched by the following command:

```
BlazeRouter log:logfile.log preview.pcb
```

The Command function returns:

```
"C:\<install_folder>\<version>\Programs\BlazeRouter.exe log:logfile.log  
preview.pcb"
```

Cos

This function returns a Double specifying the cosine of an angle. The Cos function takes an angle and returns the ratio of the length of the side adjacent to the angle divided by the length of the hypotenuse. The result lies in the range -1 to 1. To convert degrees to radians, multiply degrees by $\pi/180$. To convert radians to degrees, multiply radians by $180/\pi$.

Syntax

```
Cos(number )
```

Arguments

The Cos function has this argument

number	Required A double or any valid numeric expression expressing an angle in radians

Examples

```
x = Cos(1.57)
```

CreateObject

This function creates and returns a reference to an ActiveX object.

Syntax

```
CreateObject(class, [servername])
```

Arguments

The CreateObject function syntax has these arguments:

<i>class</i>	Required String The application name and the class of the object to create.
<i>servername</i>	Optional String The name of the network server where the object will be created. If the remote server does not exist or is unavailable, a run-time error occurs.

The ***class*** argument uses the syntax “***appname.objecttype***” and has these elements:

<i>appname</i>	Required String The name of the application providing the object.
<i>objecttype</i>	Required String The type or class of object to create.

To create an ActiveX object, assign the object returned by CreateObject to an object variable.

Use CreateObject when there is no current instance of the object. If an instance of the object is already running, a new instance is started, and an object of the specified type is created. To use the current instance, or to start the application and have it load a file, use the GetObject function.

If an object has registered itself as a single-instance object, only one instance of the object is created, no matter how many times CreateObject is executed.

Examples

```
set obj = CreateObject("PowerPCB.Application")
```

CurDir

This function returns a variant string representing the current path.

Syntax

```
CurDir [(drive)]
```

Arguments

The CurDir function has this argument:

<i>drive</i>	Optional Any string expression that specifies an existing drive. If no drive is specified or if <i>drive</i> is a zero-length string (" "), CurDir returns the path for the current drive.

Examples

```
s = CurDir("d: ")
```

Dir

This function returns a string representing the name of a file or folder that matches a specified pattern, file attribute, or drive volume label.

Syntax

```
Dir(pathname)
```

Arguments

The Dir function has this argument:

pathname	Optional String expression that specifies a file name. May also include folder and drive names. A zero-length string ("") is returned if pathname is not found.

Dir supports the use of multiple character (*) and single character (?) wildcards to specify multiple files.

Examples

To get the first file in the c: drive root:

```
Dir("C:\*. *")
```

To get the next file in the same path:

```
Dir
```

To start another search in C:\SailWind Projects\Samples:

```
Dir("C:\SailWind Projects\Samples\*. *")
```

DoEvents

This function passes control to the operating system. The operating system returns control after it finishes processing the events in its queue.

Syntax

```
DoEvents ( )
```

DoEvents may be useful if the macro is performing long calculations. Inserting DoEvents calls every second or more may prevent the accumulation of unprocessed events in the queue.

Arguments

None

Environ

This function returns the string associated with an operating system environment variable.

Syntax

```
Environ [(envstring)]
```

Arguments

The Environ function has this argument:

Envstring	Optional String expression containing the name of an environment variable

If envstring can't be found in the environment-string table, a zero-length string ("") is returned.

Environ returns the text assigned to the specified envstring; that is, the text which follows the equal sign (=) in the environment-string table for that environment variable.

Examples

```
s = Environ ("path")
```

Eof

This function returns an integer, which represents the Boolean value True when the end of a file opened for random or sequential input has been reached.

Use the Eof function to avoid the error generated by attempting to get input after the end of a file.

[Table 22](#) lists the Eof function returned values:

Table 22. Eof Function Returned Values

Condition	Eof function returns
Before end of file is reached	False
After end of file is reached	True
File opened for random access and Get statement is able to read an entire record	False
File opened for random access and Get statement is not able to read an entire record	True
File opened for binary access and Get statement is able to read an entire record	False

Table 22. Eof Function Returned Values (continued)

File opened for binary access and Get statement is not able to read an entire record	True
File opened for output	True

Usage

```
Eof [(filenumber)]
```

Arguments

The Eof function has this argument:

filenumber	Optional An integer containing any valid file number

Examples

```
If Eof (1) then ....
```

Exp

This function returns a Double specifying e (the base of natural logarithms) raised to a power. The constant e is approximately 2.718282.

 **Note:**

The Exp function complements the Log function and may be referred to as the antilogarithm.

Syntax

Exp(*number*)

Arguments

The Exp function has this argument:

<i>number</i>	Required A double or any valid numeric expression. When the value of <i>number</i> exceeds 709.782712893, an error occurs.

Examples

x = exp(y)

GetObject

This function returns a reference to an object provided by an ActiveX component.

Syntax

```
GetObject([pathname] [, class])
```

Arguments

The GetObject function has these arguments:

<i>pathname</i>	Optional Variant string The full path (folder, and drive) and name of the file containing the object to retrieve. If <i>pathname</i> is omitted, class is required.
<i>class</i>	Optional Variant string A string representing the class of the object.

The *class* argument uses the syntax “***appname.objecttype***”, which has these elements:

<i>appname</i>	Required Variant string The name of the application providing the object.
<i>objecttype</i>	Required Variant string Type or class of object to create.

Use the GetObject function to access an ActiveX object from a file and assign the object to an object variable. Use the [Set](#) statement to assign the object returned by the GetObject function to the object variable.

Examples

```
obj = GetObject(, "PowerPCB.Application")
```

GetTmpFileName

This function returns a string specifying a new file name guaranteed to be unique in the folder identified by the string argument.

Syntax

```
GetTmpFileName(string)
```

Arguments

The GetTmpFileName function has this argument:

string	Required A string expression

Examples

```
s = GetTmpFileName("d:\tmp")
```

InStr

This function returns the position of the first occurrence of one string within another string.

Syntax

```
InStr([start, ]string1, string2)
```

Arguments

The InStr function has these arguments:

<i>start</i>	Optional Numeric expression that sets the starting position for each search. When <i>start</i> is omitted, the search begins at the first character position.
<i>string1</i>	Required String expression to search.
<i>string2</i>	Required String expression to find.

Return Values

Table 23. InStr Function Return Values

If	InStr returns
string1 is zero-length	0
string2 is zero-length	start
string2 is not found	0
string2 is found within string1	position at which a match is found
start > string	0

Examples

```
i = InStr(1, "cbc", "c")
```

InStrRev

This function returns the position of an occurrence of one string within another, from the end of the string.

Syntax

```
InStrRev(string1, string2, [start])
```

Arguments

The InStrRev function has these arguments:

<i>string1</i>	Required String expression being searched.
<i>string2</i>	Required String expression to find.
<i>start</i>	Optional Numeric expression that sets the starting position for each search. If omitted, the search begins at the last character position.

Return Values

Table 24. InStrRev Function Return Values

If	InStrRev returns
<i>string1 is zero-length</i>	0
<i>string2 is zero-length</i>	Start
<i>string2 is not found</i>	0
<i>string2 is found within string1</i>	Position at which match is found
<i>start > Len(string2)</i>	0

Examples

The following sample code returns "4."

```
InStrRev("abcdbc", "bc")
```

Left

This function returns a specified number of characters from the left side of a string.

Syntax

```
Left(string, length)
```

Arguments

The Left function has these arguments:

<i>string</i>	Required A string expression from which the characters from the left side are returned
<i>length</i>	A numeric expression indicating the number of characters to return If <i>length</i> is 0, Left returns a zero-length string (" "). If <i>length</i> is greater than or equal to the number of characters in the string, Left returns the entire string.

Examples

The following sample code returns "ab."

```
Left("abcd", 2)
```


Len

This function returns the number of characters in a string (the length of the string).

Syntax

```
Len(string)
```

Arguments

The Len function has this argument:

string	Required Any valid string expression

Examples

The following sample code returns 4.

```
Len("abcd")
```

Mid

This function returns a specified number of characters from a string.

Syntax

```
Mid (string, start, [length])
```

Arguments

The Mid function has these arguments:

<i>string</i>	Required A string expression from which the characters are returned
<i>start</i>	Required The character position in <i>string</i> at which the part to return begins. If start is greater than the number of characters in <i>string</i> , Mid returns a zero-length string ("").
<i>length</i>	Optional The number of characters to return. If omitted, when there are less characters in the string (including the character at start), than <i>length</i> , Mid returns all of the characters from the start position to the end of the string.

Examples

The following sample code returns "cd."

```
Mid("abcdbc", 3, 2)
```

MkDir

This function creates a new folder.

Syntax

```
MkDir (path)
```

Arguments

The MkDir function has this argument:

path	Required A string expression that identifies the folder to create. The path may include the drive. When no drive is specified, MkDir creates the new folder on the current drive.

Examples

```
MkDir ("D:\newdir")
```

MoveFile

This function moves the file identified by path1 argument to the location specified in path2.

Syntax

```
MoveFile(path1, path2)
```

Arguments

The MoveFile function has these arguments:

path1	Required A string expression
path2	Required A string expression

Examples

```
MoveFile("C:\data.bin", "D:\")
```

MsgBox

This function displays a message in a dialog box, waits for the user to click a button, and returns an integer indicating which button the user clicked.

Syntax

```
MsgBox(prompt [, buttons] [, title])
```

Arguments

The MsgBox function has these arguments:

<i>prompt</i>	Required String expression displays as the message in the dialog box. The maximum length of prompt is 1024 characters, depending on the width of the characters used. If <i>prompt</i> consists of more than one line, you can separate the lines using a carriage return character (Chr(13)), a linefeed character (Chr(10)), or carriage return - linefeed character combination (Chr(13) & Chr(10)) between each line.
<i>buttons</i>	Optional Numeric expression which is the sum of values specifying the number and type of buttons to display, the icon style to use, and the identity of the default button. The default value for <i>buttons</i> is 0.
<i>title</i>	Optional String expression that displays in the title bar of the dialog box. The default value for title is the application name.

- Settings

The ***buttons*** argument settings are as follows:

Table 25. MsgBox buttons Settings

Constant	Value	Description
mbOKOnly	0	Display OK button only.
mbOKCancel	1	Display OK and Cancel* buttons.
mbAbortRetryIgnore	2	Display Abort, Retry, and Ignore buttons.
mbYesNoCancel	3	Display Yes, No, and Cancel* buttons.

Table 25. MsgBox buttons Settings (continued)

mbYesNo	4	Display Yes and No buttons.
mbRetryCancel	5	Display Retry and Cancel* buttons.
mbCritical	16	Display Critical Message icon.
mbQuestion	32	Display Warning Query icon.
mbExclamation	48	Display Warning Message icon.
mbInformation	64	Display Information Message icon.
mbDefaultButton1	0	First button is default.
mbDefaultButton2	256	Second button is default.
mbDefaultButton3	512	Third button is default.

- The first group of values (0-5) describes the number and type of buttons to display in the dialog box
- The second group of values (16, 32, 48, 64) describes the icon style
- The third group of values (0, 256, 512) shows which button is the default.

When adding numbers to create a final value for the buttons argument, use only one number from each group.

Return Values

Table 26. MsgBox Return Values

Constant	Value	Description
mbOK	1	OK
mbCancel	2	Cancel*
mbAbort	3	Abort
mbRetry	4	Retry
mbIgnore	5	Ignore
mbYes	6	Yes
mbNo	7	No

*If the dialog box displays a Cancel button, pressing Esc has the same effect as clicking Cancel.

Examples

```
MsgBox("Hello",mbOK, "This is a message box")
```

Right

This function returns a specified number of characters from the right side of a string.

Syntax

```
Right(string, length)
```

Arguments

The Right function has these arguments:

<i>string</i>	Required A string expression from which the characters on the right side are returned
<i>length</i>	A numeric expression indicating how many characters to return If <i>length</i> is 0, Right returns a zero-length string (" "). If <i>length</i> is greater than or equal to the number of characters in the string, Right returns the entire string.

Examples

The following sample code returns "dbc."

```
Right("abcdbc", 3)
```

Sin

This function returns a Double specifying the sine of an angle.

The Sin function takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse.

The result lies in the range -1 to 1.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

Usage

```
Sin(number)
```

Arguments

The Sin function has this argument:

number	Required

	Any expression that expresses an angle in radians. If the expression is not numeric, it is converted to a Numeric type.

Examples

```
x = sin(y)
```

Spc

This function is used with the Print # statement or the Print method to position output.

Syntax

```
Spc(n)
```

Arguments

The Spc function has this argument:

<i>n</i>	Required The number of spaces to insert before displaying or printing the next expression in a list

If *n* is less than the output line width, the next print position immediately follows the number of spaces printed. If *n* is greater than the output line width, Spc calculates the next print position using the formula:

```
currentprintposition + (n Mod width)
```

For example, if the current print position is 24, the output line width is 80, and you specify Spc(90), the next print will start at position 34 (current print position + the remainder of 90/80). If the difference between the current print position and the output line width is less than *n* (or *n* Mod width), the Spc function skips to the beginning of the next line and generates spaces equal to *n* - (width - currentprintposition).

Examples

```
Spc(3)
```

Str

This function returns a string representation of a number.

Syntax

```
Str(number)
```

Arguments

The Str function has the following argument:

<i>number</i>	Required Any expression. If the expression is not numeric, it is converted to a Numeric type.

When numbers are converted to strings, a leading space is always reserved for the sign of *number*. If *number* is positive, the returned string contains a leading space and the plus sign is implied.

Examples

```
x = Str(324)
```

Tab

This function is used with the Print # statement or the Print method to position output.

Syntax

```
Tab[ ( n ) ]
```

Arguments

The Tab function has this argument:

<i>n</i>	Optional The column number to move to before displaying or printing the next expression in a list

If *n* is omitted, Tab moves the insertion point to the beginning of the next print zone. This allows you to use Tab instead of a comma whenever you use the comma as a decimal separator.

If the current print position on the current line is greater than *n*, Tab skips to the *n*th column on the next output line. If *n* is less than 1, Tab moves the print position to column 1. If *n* is greater than the output line width, Tab calculates the next print position using the formula:

```
n Mod width
```

For example, if *width* is 80 and you specify *Tab(90)*, the next print will start at column 10 (the remainder of 90/80). If *n* is less than the current print position, printing begins on the next line at the calculated print position. If the calculated print position is greater than the current print position, printing begins at the calculated print position on the same line.

The leftmost print position on an output line is always 1. When you use the b statement to print to files, the rightmost print position is the current width of the output file, which you can set using the [Width #](#) statement.

Examples

```
Tab(2)
```

Val

This function returns the numbers contained in a string as a numeric value of the appropriate type.

Syntax

```
val(string)
```

Arguments

The Val function has this argument:

string	Required Any valid string expression

The Val function stops reading the string at the first character it can't recognize as part of a number. However, the function recognizes the radix prefixes &O (for octal) and &H (for hexadecimal). Blanks, tabs, and linefeed characters are stripped from the argument. The Val function recognizes only the period (.) as a valid decimal separator.

Examples

```
i=Val("123")
```

Automation Support

The macro engine in the program supports Automation through its automation objects. After an object is created using CreateObject() or connected to using GetObject(), the object's methods may be called using the usual syntax.

```
Object.Method arg1, ..., argn
var = Object.Method( arg1, ..., argn )
var = Object.Property
Object.Property = expression
```

Related Topics

- [CreateObject](#)
- [GetObject](#)

Dialog Box Controls

The macro language of the program uses certain dialog box controls.

- CheckBox.State
- CheckBox.Value
- CheckListBox.State
- CheckListBox.SetCheck
- CheckListBox.ListCount
- CheckListBox.SelCount
- CheckListBox.Selected
- CheckListBox.Check
- CheckListBox.Text
- ComboBox.Select
- ComboBox.Edit
- ComboBox.Text
- ComboBox.List
- ComboBox.SelStart
- ComboBox.SelLength
- ComboBox.SelText
- EditBox.State
- EditBox.Text
- EditBox.SelStart
- EditBox.SelLength
- EditBox.SelText
- GridControl
- ListBox.State
- ListBox.List
- ListBox.ListCount
- ListBox.SelCount
- ListBox.Selected
- ListBox.Text
- PushButton
- RadioBox.State
- RadioBox.Value
- SliderControl.State
- SliderControl.Value
- SpinButton.State
- TabControl.State
- TabControl.Value
- Treeltem.Select
- Treeltem.Expand
- Treeltem.Focus

TreeView.Item
TreeView.BeginDrag
TreeView.Copy
TreeView.Drop
TreeView.Paste
TreeView.CreateNewItem

CheckBox.State

Represents a check box on a dialog box. You can reference a particular check box using the Control method of a dialog box. The CheckBox object uses the State and Property methods. This method sets the state of the check box.

Syntax

```
checkbox.State(iState)
```

Arguments

State has this argument:

<i>iState</i>	Required A numeric expression, representing the check box state to set

iState may have one of the following values:

Value	Description
0, or False	Unchecked
1, or True	Checked
2	Indeterminate

CheckBox.Value

This method returns or sets the state of the check box control.

Syntax

```
checkbox.Value[=iState]
```

Arguments

The Value Property has this argument:

<i>iState</i>	Required A numeric expression, representing the check box state to set

iState may have one of the following values::

Value	Description
0, or False	Unchecked
1, or True	Checked
2	Indeterminate

CheckListBox.State

This method sets the selection state of the check list box on a dialog box.

Syntax

```
CheckListBox.State(string)
```

Arguments

The State method has this argument:

string	Required Any valid string expression, containing the list of items to select

CheckListBox.SetCheck

Sets the check state of the check list box.

Syntax

```
CheckListBox.SetCheck(string)
```

Arguments

The SetCheck method has this argument:

string	Required A string expression, containing the list of the items to check. All the items that do not appear in the list are not checked.

CheckListBox.ListCount

This method returns the number of items in the check list box.

Syntax

```
CheckListBox.ListCount
```

Arguments

None

CheckListBox.SelCount

This method returns the number of selected items in the check list box.

Syntax

CheckListBox.SelCount

Arguments

None

CheckBox.Selected

This method returns or sets the selection status of an item in the check list box. This property is an array of Boolean values with the same number of items as the List property.

Syntax

```
CheckBox.Selected(index) [=boolean]
```

Arguments

None

CheckListBox.Check

This method returns or sets the checked status of an item in a check list box. This property is an array of Boolean values with the same number of items as the list property.

Syntax

```
CheckListBox.Check(index) [=boolean]
```

Arguments

None

CheckListBox.Text

This method returns the text of the item currently selected in the check list box.

Syntax

```
CheckListBox.Text
```

Arguments

None

ComboBox.Select

This method sets the selection state of the combo box on a dialog box.

Syntax

```
ComboBox.Select(string)
```

Arguments

The Select method has this argument:

string	Required A string expression, containing a string to select in the combo box

Examples

```
ActiveLayer.Select("Top")
```

ComboBox.Edit

This method sets the edit state of the combo box.

Syntax

```
ComboBox.Edit(string)
```

Arguments

The Edit method has this argument:

<i>string</i>	Required A string expression, containing a string to insert into the edit box of the combo box

ComboBox.Text

This method returns or sets the combo box text.

Syntax

```
ComboBox.Text[=string]
```

Arguments

The Text method has this argument:

<i>string</i>	Required A string expression, containing the text to set into the combo box

ComboBox.List

This method returns or sets the items contained in the combo box list. This list is a string array in which each element is a list item.

Syntax

```
ComboBox.List(index)
```

Arguments

None

ComboBox.SelStart

This method returns or sets the starting point of the selected text. If no text is selected, this method indicates the position of the insertion point.

Syntax

```
ComboBox.SelStart[=index]
```

Arguments

None

ComboBox.SelLength

This method returns or sets the number of characters selected.

Syntax

```
ComboBox.SelLength[=number]
```

Arguments

None

ComboBox.SelText

This method returns or sets the string containing the currently selected text. If no characters are selected, this method returns a zero length string ("").

Syntax

```
ComboBox.SelText[=string]
```

Arguments

The argument is a string expression, containing the text to set into the edit box.

EditBox.State

Represents an edit box on a dialog box. You can use the Control method of a dialog box to reference a particular edit box. This method sets the state of the edit box.

Syntax

```
EditBox.State(string)
```

Arguments

State has this argument:

string	Required A string expression of the text to display in the edit box

EditBox.Text

This method returns or sets the edit box text.

Syntax

```
EditBox.Text[=string]
```

Arguments

The string argument is a string expression, containing the text to set into the edit box.

EditBox.SelStart

This method returns or sets the starting point of selected text. If no text is selected this method indicates the position of the insertion point.

Syntax

```
EditBox.SelStart[=index]
```

Arguments

None

EditBox.SelLength

This method returns or sets the number of characters selected.

Syntax

```
EditBox.SelLength[=number]
```

Arguments

None

EditBox.SelText

This method returns or sets the string containing the currently selected text. If no characters are selected this method returns a zero length string ("").

Syntax

```
EditBox.SelText[=string]
```

Arguments

The SelText method has this argument:

string	A string expression, containing the text to set into the edit box

GridControl

Represents a grid control on a dialog box. You can use the control method to reference a particular grid control.

ListBox.State

This method sets the selection state of the dialog box.

Syntax

```
ListBox.State(string)
```

Arguments

The State method has this argument:

string	Required A string expression, containing the item numbers to select. All the items that do not appear in the list are not selected.

ListBox.List

This method returns the items contained in a dialog box list portion. The list is a string array in which each element is a list item.

Syntax

```
ListBox.List
```

Arguments

None

ListBox.ListCount

This method returns the number of items in the list box.

Syntax

```
ListBox.ListCount
```

Arguments

None

ListBox.SelCount

This method returns the number of selected items in a list box.

Syntax

```
ListBox.SelCount
```

Arguments

None

ListBox.Selected

This method returns or sets the selection status of an item in a list box. This property is an array of Boolean values with the same number of items as the list property.

Syntax

```
ListBox.Selected(index) [=boolean]
```

Arguments

None

ListBox.Text

This method returns the currently selected (focused) item's text in a list box.

Syntax

```
ListBox.Text
```

Arguments

None

PushButton

Represents a push button (also called a command button) on a dialog box. This method emulates pressing a push button.

Syntax

```
button.Click()
```

Arguments

None

Examples

```
dlg.control("OK").click()
```

RadioBox.State

This method checks the state of an option button on a dialog box.

Syntax

```
RadioBox.State(iState)
```

Arguments

The State method has this argument:

iState	Required A numeric expression, representing the position of an option button to check. -1 means that no button is selected.

RadioBox.Value

This method returns or checks the state of the option button.

Syntax

```
RadioBox.Value[=iState]
```

Arguments

Value Property has this argument:

iState	Required A numeric expression, representing the position of an option button to check. -1 means that no button is selected.

SliderControl.State

This method sets the state of a slider control on a dialog box.

Syntax

```
SliderControl.State(iState)
```

Arguments

State has this argument:

iState	Required A numeric expression

SliderControl.Value

This method returns or sets the current slider position.

Syntax

```
Slider.Value[=val]
```

Arguments

Value Property has this argument:

val	Required A numeric expression

SpinButton.State

This method sets the state of a spin button on a dialog box.

Syntax

```
SpinButton.State(iState)
```

Arguments

State has this argument:

iState	Required A numeric expression

TabControl.State

This method sets the selection state of a tab on a dialog box.

Syntax

```
TabControl.State(iState)
```

Arguments

State has this argument:

iState	Required A numeric expression, which represents the position of a tab.

Examples

```
dlg.Control("Tab").State(3)
```

TabControl.Value

This method returns or sets the current tab position.

Syntax

```
TabControl.Value[=tab]
```

Arguments

Value Property has this argument:

tab	Required Either a numeric expression representing the tab position or a string expression representing the tab caption

Treeltem.Select

This method sets the selection state of a tree item on a dialog box.

Syntax

```
Treeltem.Select(flag)
```

Arguments

The Select method has this argument:

flag	Required A numeric expression

flag may have one of the following values:

Value	Description
0, or False	Unselect
1, or True	Select

Examples

```
item.Select(true)
```

TreeItem.Expand

This method sets the expand state of the tree item.

Syntax

```
TreeItem.Expand(flag)
```

Arguments

Expand has this argument:

flag	Required A numeric expression

flag may have one of the following values:

Value	Description
0, or False	Collapse
1, or True	Expand

Examples

```
item.Expand(true)
```

TreeItem.Focus

This method sets the tree item focus to the item.

Syntax

```
TreeItem.Focus()
```

Arguments

None

Examples

```
item.Focus(1)
```

TreeView.Item

This method returns a Treeltem object from a tree view on a dialog box.

Syntax

```
TreeView.Item(itemname)
```

Arguments

Item has this argument:

itemname	Required A string expression representing the name of the item

Examples

```
item = tree.Item("Net Objects\Nets\end")
```

TreeView.BeginDrag

This method emulates dragging selected items off the tree.

Syntax

```
TreeView.BeginDrag(itemname)
```

Arguments

BeginDrag has this argument:

itemname	Required A string expression representing the name of the item to drag

TreeView.Copy

This method copies the selected item to the Clipboard.

Syntax

```
TreeView.Copy( itemname )
```

Arguments

Copy has this argument:

itemname	Required A string expression representing the name of the item to copy

TreeView.Drop

This method emulates dropping the dragged items onto an item.

Syntax

```
TreeView.Drop(itemname)
```

Arguments

The Drop method has this argument:

itemname	Required A string expression representing the name of the item onto which to drop the dragged items

Examples

```
tree.Drop("Net Objects\Net classes")
```

TreeView.Paste

This method pastes the contents of the Clipboard into the selected branch.

Syntax

```
TreeView.Paste(itemname)
```

Arguments

Paste has this argument:

itemname	Required A string expression representing the name of the item to paste

TreeView.CreateNewItem

This method creates a new item in the selected branch. This method returns a TreeItem object that corresponds to the created item.

Syntax

```
TreeView.CreateNewItem(itemname)
```

Arguments

CreateNewItem has this argument:

itemname	Required A string expression representing the name of the item to create

Internal Macro Objects

The SailWind Layout program includes internal Macro Objects.

- [Application Object](#)
- [Dialog Objects](#)
- [Document Object](#)
- [Application.HelpContents.Item Object](#)
- [HelpContentsItem Object](#)
- [HelpPane Object](#)
- [Main View Object](#)

[Application Object](#)

[Dialog Objects](#)

[Document Object](#)

[Application.HelpContents.Item Object](#)

[HelpContentsItem Object](#)

[HelpPane Object](#)

[Main View Object](#)

Application Object

The Application object represents applications of the program.

The object has the following methods:

- CreateNewDocument
- ExecuteCommand
- Help
- HelpContents
- HelpPane
- OpenCustomizeDialog
- OpenDocument
- OpenOptionsDialog
- OpenPropertiesDialog
- Quit
- RunMacro

CreateNewDocument

Creates an empty document.

Syntax

```
Application.CreateNewDocument
```

Arguments

None

ExecuteCommand

Executes one of the commands of the program.

Syntax

```
Application.ExecuteCommand(command, [arg1, ...])
```

Arguments

The ExecuteCommand method has these arguments:

<i>command</i>	Required A string expression representing a PADS product command
<i>arg1</i> ,...	Optional Represents optional arguments that are passed to <i>command</i>

Examples

```
Application.ExecuteCommand("View Board")
```

```
Application.ExecuteCommand("Open", "C:\SailWind Projects\preview.pcb")
```

Help

Invokes the product Help.

Syntax

```
Application.Help()
```

Arguments

None

HelpContents

Returns the Help contents in the Help Contents window.

Syntax

```
Application.HelpContents
```

Arguments

None

Examples

```
Set var = Application.HelpContents
```

HelpPane

Returns the Help window.

Syntax

```
Application.HelpPane
```

Arguments

None

Examples

```
Set var = Application.HelpPane
```

OpenCustomizeDialog

Opens the Customize modal dialog box.

Syntax

```
Application.OpenCustomizeDialog()
```

Arguments

None

OpenDocument

Opens an existing document identified by the path argument.

Syntax

```
Application.OpenDocument(path)
```

Arguments

The OpenDocument method has this argument:

path	Required A string expression containing the path of the document to open

Examples

```
Application.OpenDocument("C:\SailWind Projects\preview.pcb")
```

OpenOptionsDialog

Opens the Options modeless dialog box.

Syntax

```
Application.OpenOptionsDialog()
```

Arguments

None

OpenPropertiesDialog

Opens the Properties modeless dialog box.

Syntax

```
Application.OpenPropertiesDialog()
```

Arguments

None

Quit

Exits the application.

Syntax

```
Application.Quit()
```

Arguments

None

RunMacro

Executes one of the program commands.

Syntax

```
Application.RunMacro(path[, function [, arg1, ...]])
```

Arguments

The RunMacro method has these arguments:

<i>path</i>	Required A string expression containing the file path of the macro to run
<i>function</i>	Optional Name of a function or a sub in the macro file to be called. If the <i>function</i> is specified, RunMacro returns what the function returns. If the <i>function</i> is not specified, or it is a sub, which returns nothing, RunMacro returns nothing.
<i>arg1</i>,...	Optional Arguments that are passed onto <i>function</i>

Examples

```
Application.RunMacro("C:\SailWind Projects\mymacro.mcr")
```

```
Var = Application.RunMacro("C:\SailWind Projects\mymacro.mcr", myfunction",  
1, 2, 3)
```


Dialog Objects

A dialog object represents a dialog box.

This object has the following methods:

- [CloseHelpPane](#)
- [Control](#)
- [Focus](#)
- [OpenHelpPane](#)
- [ShowHelpFor](#)

CloseHelpPane

Closes the open help pane in a dialog box.

Syntax

```
Dialog.CloseHelpPane
```

Arguments

None

Examples

```
Dialog.CloseHelpPane
```

Control

Returns a dialog box control.

Syntax

```
Dialog.Control(controlname)
```

Arguments

Control has this argument:

controlname	Required A string expression representing the name of the control

Examples

This example returns the OK button.

```
set obj = dlg.Control("OK")
```

Related Topics

[Focus](#)

Focus

Sets focus to a dialog box control.

Syntax

```
Dialog.Focus(controlname)
```

Arguments

The Focus method has this argument:

controlname	Required A string expression representing the name of the control

Related Topics

[Control](#)

OpenHelpPane

Displays the help pane for the dialog box.

Syntax

```
Dialog.OpenHelpPane
```

Arguments

None

Examples

```
Dialog.OpenHelpPane
```

ShowHelpFor

Displays help for the specified control.

Syntax

```
Dialog.ShowHelpFor(controlname)
```

Arguments

The ShowHelpFor method has this argument:

controlname	Required The name of the control.

Examples

This example displays Help for the Apply button.

```
Dialog.ShowHelpFor("Apply")
```

Document Object

The Document object represents any currently loaded design.

This object has the following methods:

- [Print](#)
- [PrintSetup](#)
- [RepeatLastAction](#)
- [Save](#)
- [SaveAs](#)

Print

Prints the document.

Syntax

```
Document.Print()
```

Arguments

None

PrintSetup

Opens the Print Setup dialog box.

Syntax

```
Document.PrintSetup()
```

Arguments

None

RepeatLastAction

Repeats the last action performed during the current session.

Syntax

```
Document.RepeatLastAction()
```

Arguments

None

Save

Saves the document, if it has been modified.

Syntax

```
Document . Save ( )
```

Arguments

None

SaveAs

Saves the document to a user-defined name or path location.

Syntax

```
Document . SaveAs (path
```

Arguments

<i>path</i>	Required A string expression representing the path to which to save the document

Application.HelpContents.Item Object

The Item property finds the location of the Help contents item in the contents tree of the Help Contents window.

Syntax

```
Application.HelpContents.Item (path)
```

Arguments

None

Examples

```
Set item = Application.HelpContents.Item("File Operations\To Restore  
Files")
```

Related Topics

[Internal Macro Objects](#)

HelpContentsItem Object

Finds the name of the Help Contents item.

The HelpContentsItem object has the following properties and one method (Select):

- [Location](#)
- [Name](#)
- [Select](#)
- [SubItem](#)
- [SubItemCount](#)

Location

Returns the location of the item.

Syntax

```
Item.Location
```

Arguments

None

Examples

In this example, the `item_loc` variable is assigned a value of "its:C:\<install_folder>\<version>\Documentation\Router\BlazeRouter.chm::fileops/To_Restore_Files.htm".

```
Set item = Application.HelpContents.Item("File Operations\To Restore  
Files")
```

```
item_loc = item.Location
```

Name

Returns the name of the item.

Syntax

```
Item.Name
```

Arguments

None

Examples

In this example, the item_name variable is assigned the value of "To Restore Files."

```
Set item = Application.HelpContents.Item("File Operations\To Restore  
Files")
```

```
item_name = item.Name
```


Select

Selects the item.

Syntax

```
Item.Select
```

Arguments

None

Examples

```
Set item = Application.HelpContents.Item("File Operations").SubItem(3)
```

```
item.Select
```

SubItem

Returns a sub item of the item by its position. The required integer pos argument is the zero-based serial number of a sub item in the tree branch that the item represents.

Syntax

```
Item.SubItem(pos)
```

Arguments

None

Examples

In this example, the item_name variable is assigned value of "To Restore Files".

```
Set item = Application.HelpContents.Item("File Operations").SubItem(3)
```

```
item_name = item.Name
```

SubItemCount

Returns the number of sub items within the item.

Syntax

```
Item.SubItemCount
```

Arguments

None

Examples

In this example, the *count* variable is assigned a value of 10.

```
Set item = Application.HelpContents.Item("File Operations")
```

```
count = item.SubItemCount
```

HelpPane Object

Represents the Help window.

Document

The Document property displays the HTML document in the Help window. This property uses the Document Object Model (DOM). For the full description of the HTMLDocument interface, see the Microsoft Software Developer's Network (MSDN) documentation.

The following are the most useful properties that the Macro engine uses:

title	Sets or retrieves the title of the document. This property displays the document title in the title bar of the document window. Also, it identifies the contents of the document.
URL	Sets or retrieves the Uniform Resource Locator (URL) for the current document.

Related Topics

[Internal Macro Objects](#)

Main View Object

The MainView object represents the main view of the program.

This object uses the following methods:

- ActiveLayer
- ToggleFullScreen
- MouseDown
- MouseEndDrag
- MouseMove
- MouseStartDrag
- MouseUp
- Print
- PrintPreview

ActiveLayer

Displays the Active Layer combo box.

Syntax

```
MapView.ActiveLayer
```

Arguments

None

Examples

```
set layerCombo = MapView.ActiveLayer
```

ToggleFullScreen

Turns the Full Screen mode on.

Syntax

```
MainView.ToggleFullScreen()
```

Arguments

None

MouseDown

Emulates pressing a mouse button.

Syntax

```
MainView.MouseDown(x, y, button)
```

Arguments

The MouseDown method has these arguments:

<i>x</i>	Required Any numeric expression X-coordinate of pointer
<i>y</i>	Required Any numeric expression Y-coordinate of pointer
<i>button</i>	Required String expression Pressed button, if any

The ***button*** argument may contain one or more of the following values and modifiers:

Value	Description
L	Left mouse button pressed
M	Middle mouse button pressed
R	Right mouse button pressed
C	Ctrl button pressed (modifier)
S	Shift button pressed (modifier)
A	Alt button pressed (modifier)

MouseEndDrag

Emulates ending a mouse drag operation.

Syntax

```
MainView.MouseEndDrag(x, y, button)
```

Arguments

The Mouse End Drag method has these arguments:

<i>x</i>	Required Any numeric expression X-coordinate of pointer
<i>y</i>	Required Any numeric expression Y-coordinate of pointer
<i>button</i>	Required String expression Pressed button, if any

The ***button*** argument may contain one or more of the following values and modifiers:

Value	Description
L	Left mouse button pressed
M	Middle mouse button pressed
R	Right mouse button pressed
C	Ctrl button pressed (modifier)
S	Shift button pressed (modifier)
A	Alt button pressed (modifier)

MouseMove

Emulates moving the mouse.

Syntax

```
MainView.MouseMove(x, y, button)
```

Arguments

The Mouse Move method has these arguments:

<i>x</i>	Required Any numeric expression X-coordinate of pointer
<i>y</i>	Required Any numeric expression Y-coordinate of pointer
<i>button</i>	Required String expression * indicates relative mode + indicates origin plus relative mode Pressed button, if any

The ***button*** argument may contain one or more of the following values and modifiers:

Value	Description
L	Left mouse button pressed
M	Middle mouse button pressed
R	Right mouse button pressed
C	Ctrl button pressed (modifier)
S	Shift button pressed (modifier)
A	Alt button pressed (modifier)

Examples

```
MainView.MouseMove(300, 350, "+L")
```

or

```
MainView.MouseMove(300,350,"*L")
```

MouseDownDrag

Emulates starting a mouse drag operation.

Syntax

```
MainView.MouseDownDrag(x, y, button)
```

Arguments

TheMouseDownDrag method has these arguments:

<i>x</i>	Required Any numeric expression X-coordinate of pointer
<i>y</i>	Required Any numeric expression Y-coordinate of pointer
<i>button</i>	Required String expression Pressed button, if any

The ***button*** argument may contain one or more of the following values and modifiers:

Value	Description
L	Left mouse button pressed
M	Middle mouse button pressed
R	Right mouse button pressed
C	Ctrl button pressed (modifier)
S	Shift button pressed (modifier)
A	Alt button pressed (modifier)

MouseUp

Emulates releasing a mouse button.

Syntax

```
MainView.MouseUp(x, y, button)
```

Arguments

<i>x</i>	Required Any numeric expression X-coordinate of pointer
<i>y</i>	Required Any numeric expression Y-coordinate of pointer
<i>button</i>	Required String expression Pressed button, if any

The ***button*** argument may contain one or more of the following values and modifiers:

Value	Description
L	Left mouse button pressed
M	Middle mouse button pressed
R	Right mouse button pressed
C	Ctrl button pressed (modifier)
S	Shift button pressed (modifier)
A	Alt button pressed (modifier)

Print

Prints the current view.

Syntax

```
MapView.Print()
```

Arguments

None

PrintPreview

Turns the Print Preview mode on.

Syntax

```
MainView.PrintPreview()
```

Arguments

None

Chapter 4

Intermediate Data Format

If you have the IDF Interface option, you can import and export Intermediate Data Format (IDF) files. IDF is a format that allows data exchange between SailWind Layout and a mechanical design system. SailWind Layout supports IDF 2.0 and IDF 3.0.

For information about the IDF file format, see the [Intermediate Data Format Specification](#)..



Tip

SailWind Layout can exchange IDF files with mechanical design systems that support IDF. While the following sections were originally written for Pro/ENGINEER, the information in those sections may also apply to some of the other many mechanical design systems on the market today.

[IDF File Export](#)

[IDF File Import](#)

[IDF File Format](#)

[IDF 3.0 Enhancements](#)

IDF File Export

You can, at any stage of your design, export the design data to IDF format, mechanical design software can import to edit and analyze the mechanical data.

When exporting IDF files, you create two files:

- A .emn board file
- A .emp library file

When exporting to IDF, you can export the following information about parts: part height, drilled holes, and part outlines. In addition, settings in another file, a .map file, allow mechanical design software to model parts from SailWind Layout in 3D.

[3D Data Export](#)
[Part Height Information](#)
[Considerations for Exporting Holes](#)
[Part Outline Information](#)
[IDF Export Messages](#)
[IDF Export Status File](#)
[Height Messages](#)
[Outline Messages](#)
[Keepout Messages in the IDF Export Status File](#)

3D Data Export

When you import IDF files into mechanical design software, it uses the 2D data exported from SailWind Layout to create the appropriate part outline. The exported part outline and the height data are used to construct a protrusion that is the 3D component outline.

If you want to use geometrically accurate 3D models instead, you can replace simplified SailWind Layout models with mechanical design software models using a mechanical design software library and the `ecad_hint.map` file. To replace SailWind Layout models, you must create a mechanical design software part library containing all your parts and then set up the `ecad_hint.map` file to map the 3D replacements.

The `ecad_hint.map` file is a text file that you create, edit, and maintain. This file replaces approximated parts from SailWind Layout with geometrically accurate components, previously modeled in the mechanical design software. This file must exist in either the current working directory or in the mechanical design software's loadpoint/text directory.

Part Height Information

To export part heights to IDF, first assign the `Geometry.Height` attribute at the Decal or Part Type hierarchy level. The IDF library file format requires height information for each part type and decal pair in the design. As a general rule, add height information to decals because of their geometric nature.

If the height of a part is less than the minimum height specified on the IDF Export dialog box, then the part is not exported.

During export, the file is searched for part height information in the following order:

1. Search for Geometry.Height attribute on part type or decal
2. Search for the text string ZHEIGHT=height in the decal
3. Search for the text string \$... in the decal on layer 30 (in default layer mode) or 130 (in increased layer mode)

If the search does not find part heights, the Missing Height dialog box appears where you can enter part height information. For more information, see [Setting IDF-Specific Part Height Information for IDF Export](#)

Considerations for Exporting Holes

To model holes in mechanical design software, export from SailWind Layout a single pin component with a nonzero drill diameter. For mechanical design software to recognize a part as a drilled hole, one of the following conditions must be true:

- Its part type has a HOLE attribute (for PowerPCB 3.0 and later)
- Its decal has a HOLE attribute
- Its decal contains the text string HOLE

For more information, see the [Adding IDF-Specific Drill Hole Information for IDF Export](#) topic.

These specially-marked components are exported into the Drilled Holes section of the IDF file, and the mechanical design system interprets these as actual holes in the board. Drill oversize information is not included for plated holes, and oversize information does not apply to nonplated holes.



Tip

If you use the same decal for holes and nonholes, but use different part types, set the HOLE attribute in the part type.

Part Outline Information

In SailWind Layout, a part outline is a closed line element, with perhaps some arcs drawn along the way, starting with a polygon or a completed polygon with segments stretched into arcs. Disjointed elements are not accepted.

The IDF library file format requires that SailWind Layout exports outline information for each part type and decal pair in the design. Add this information to the SailWind Layout library decal using the PCB Decal Editor; otherwise, the outline is approximated by a bounding rectangle, capturing the extents in a box.

Use the Shape Layer list to select the decal layer where you entered outline information. You must have only one, single, closed outline on the selected layer. Additional segments or more than one closed outline are not recognized as an outline; geometry is approximated by a bounding rectangle.

SailWind Layout searches for the outline in the following order:

1. Selected layer
2. Layer 0
3. Layer 1
4. Bounding rectangle. If SailWind Layout cannot find a valid outline on any of the above layers, SailWind Layout use a bounding rectangle. The bounding rectangle is the smallest rectangle that encloses all nontext graphics on all layers.



Tip

: Even though layer 0 is treated specially in SailWind Layout because graphics on layer 0 appear on all layers, this does not affect how SailWind Layout finds an outline for IDF export. For example, if you have outlines on layer 0 and layer 5 and you click layer 5 in the Shape Layer list, the layer 5 outline is exported.

Part outlines are not imported to SailWind Layout.

IDF Export Messages

During IDF export, you may receive one or more messages.

- **Wrong Layer Specified for Cluster Keepout.**

Inner was specified as the layer for a cluster keepout. IDF does not support the Inner attribute for layers. Keepouts assigned to Inner in SailWind Layout are not exported to IDF.

- **Can't write the component outline for <reference designator>. The outline is self-intersecting. Continue with export?**

The outline for this component is self-intersecting. Some mechanical design systems do not support outlines with this geometry. An outline is not passed for this part, and as a result, the mechanical design software cannot import the IDF file properly.

- **A part <part type> has a HOLE attribute, but has more than 1 pin. Continue with export?**

A part type was found with the attribute HOLE. Because this part type has more than one pin, the part can't be passed to the mechanical design software as a drilled hole. Click OK to ignore the warning and save the part type as a normal electrical part. Click Cancel to stop exporting and then change the part type to one pin in SailWind Layout. Export can save the one pin part type to the IDF file as a drilled hole.

- **A decal <decal> containing the text string HOLE has more than 1 pin. Continue with export?**

A decal was found with the text string HOLE. Because this decal has more than one pin, the decal can't be passed to the mechanical design software as a drilled hole. Click OK to ignore this warning and save the decal as a normal electrical part. Click Cancel to stop exporting and then change the decal to one pin in SailWind Layout. Export can save the one pin decal to the IDF file as a drilled hole.

- **A decal <decal> containing the text string HOLE has drill diameter = 0. Continue with export?**

A decal was found with the text string HOLE. Because this decal has a drill diameter of 0, the decal can't be passed to the mechanical design software as a drilled hole. Click OK to ignore this warning and save the decal as a normal electrical part. Click Cancel to stop exporting and then change the decal to a drill diameter greater than 0. Export can save the revised decal as a drilled hole.

- **Via keepout containing point (x,y) on layer <layer> is not written to IDF file. IDF supports only through via keepouts.**

IDF recognizes only via keepouts that are defined for all layers. Click OK to continue exporting. Via keepouts that are not defined as through via keepouts are ignored.

- **Trace keepout containing point (x,y) on layer <layer> is not written to IDF file. IDF does not support trace keepouts for individual inner layers.**

A trace keepout was found for an individual inner layer. IDF supports trace keepouts only on TOP, BOTTOM, BOTH, INNER, and ALL. Click OK to continue exporting. Trace keepouts defined for individual inner layers are not exported.

- **Warning: Keepouts of the following types are found in design, and are not written to IDF file. IDF does not support these keepout types. See the report file for a list of the keepouts found.**

Keepouts of a type that IDF does not support were found in the design. IDF supports keepouts for placement, routing, and vias only. Click OK to continue exporting. Keepouts that are not supported are not exported.

IDF Export Status File

When you export to IDF, SailWind Layout creates a status log file named *idfexport.sts* in *\SailWind Projects*. This file contains a log of the locations of height and outline information used during exporting for each part type and decal combination, as well as any name changes made to avoid illegal characters in the mechanical design software.

For example, for part type CAP\CK05 using decal CAP\MR05, you might see the following four lines in the *idfexport.sts* file:

```
Warning: Decal illegal character, CAP\MR05 is now CAP_MR05
```

```
Warning: Part Type illegal character, CAP\CK05 is now CAP_CK05
```

```
Note: Part type CAP\CK05 using decal CAP\MR05: Layer 26 height 300 used.
```

```
Note: Part type CAP\CK05 using decal CAP\MR05: Shape found on layer 0.
```

Height Messages

You may receive one or more messages when specifying height for a component or decal.

The following are possible height messages:

Message	Description
Note: Part type MTHOLE6 using decal MTHOLE6: Attribute height 1 2 used.	The height was found in the value of a ZHEIGHT attribute found on the part type.
Note: Part type MTHOLE6 using decal MTHOLE6: Dialog height 250.0 used.	You entered a height in the Missing Height dialog box.
Note: Part type MTHOLE6 using decal MTHOLE6: Layer 26 height 100 used.	A ZHEIGHT text string was found on layer 26 of the decal.
Warning: Part type MTHOLE6 using decal MTHOLE6: Dialog height 0.0 reused.	You entered a height for a previous part type and decal pair in the Missing Height dialog box and selected the For All Parts check box.
Warning: Part type MTHOLE6 using decal MTHOLE6: Dxf height \$35.1 used.	The DXF convention was used.
Warning: Part type <part type> using decal <decal>: Not exported, has height <xxx> less than minimum <xxx> <units>.	A part was not exported because it is less than the minimum height set in the IDF Export dialog box.

Outline Messages

You may receive one or more messages regarding component outline.

The following are possible outline messages:

Message	Description
Note: Part type TO-92 using decal TO-92: Shape found on layer 1.	A valid outline was found on layer 1 and layer 1 was selected in the Shape Layer list in the IDF Export dialog box.
Warning: Part type TO-92 using decal TO-92: Pick box shape used.	A valid outline was not found on the layer selected in the Shape Layer list in the IDF Export dialog box.

Keepout Messages in the IDF Export Status File

You may receive one or more message regarding keepouts.

The following are possible keepout messages:

Message	Description
Warning: <Test Point Copper Plane Component Drill Decal > keepout containing point <x,y> found in design, not written to IDF file.	Keepouts were found in the design that IDF does not support. They are not written to the IDF file.

IDF File Import

When you import IDF files created in mechanical design software, SailWind Layout imports the board outline and part placement that you changed in the mechanical design software. SailWind Layout also imports drilled holes, allowing a mechanical engineer to send hole placement to the layout designer.

You do not need library parts for these holes in your SailWind Layout library, as SailWind Layout creates them as needed. For more information see [“Hole Import”](#).

You can also import components, allowing a mechanical engineer to send some part placement to the layout designer. These parts must have corresponding part types and decals in the SailWind Layout library. Route data remains unchanged, except for unroutes which move with new pin locations. For more information see [“Component Addition During Import”](#).

If you import an IDF file that contains a scaled arc with a radius that is too big for the database, a dialog box appears asking for an arc approximation error. The arc approximation error determines how the arc will be broken into line segments. For more information, see the Drafting Properties Dialog Box topic.

Because importing an IDF file can move and delete parts and keepouts, the translator checks to see whether route and reuse protection is violated, and reports any violations. For more information, see [“IDF Import Status File”](#).

[Hole Import](#)

[Component Addition During Import](#)

[Part Name Incompatibility](#)

[Illegal Character Incompatibility](#)

[IDF Import Messages](#)

[IDF Import Status File](#)

Hole Import

You can import drilled holes in the IDF file. Part type and decal names are created for all holes added to the design in the form DH<drill diameter><units><plated status>.

The following table shows the field definitions.

Table 27. Field Definitions for Imported Holes

Field	Description
DH	Drill hole notation
<drill diameter>	Size of the diameter, less than or equal to 11 characters, including the decimal point
<units>	Units: MM (millimeters) or ML (Mils)
<plated status>	Plating status: P (plated) or N (nonplated)

The entire drill hole part type and decal name must be less than or equal to 16 characters. For example, a 123.456 mil plated hole has a part type and decal name of DH123_456MLP, where DH = Drill Hole, 123_456 = drill diameter, ML = units, and P = plated status.

Parts are added to the design in the following order:

1. If the part type or decal with the assigned name already exists in the design, the existing part is used to represent the hole.
2. If the part type or decal with the assigned name does not exist, the library is searched.
3. If a match is found in the library, the part in the library is used.
4. If a match is not found in the library, a new part type and decal are created in the design using the logic family UND.

The new decal has a HOLE attribute (if importing IDF 3.0 format) or the text string HOLE (if importing IDF 2.0 format), one pin, and the proper drill diameter. These components receive the reference designators assigned to the logic family to which they belong. Holes are returned to SailWind Layout with ECO registration turned off for the part and attribute.

Holes that existed in the same locations prior to the import keep their original reference designators. Holes that do not exist in the imported IDF file are deleted.

Component Addition During Import

You can add components to your design when importing an IDF file. This lets mechanical designers place mechanical components, such as connectors or components with restrictions on placement (for instance, a processor with a large heat sink).

SailWind Layout adds components to the design as follows:

1. If the correct part type and decal already exist in the design, the existing component is used.
2. If the part type or decal does not exist, the library is searched.
3. If a match is found in the library, the component in the library is used.
4. If a match is not found in the library, an error message appears.

Added components use the reference designator specified in the IDF file.

You usually need to edit the IDF board file when you add components in mechanical design software to resolve two incompatibilities that exist between mechanical design software and SailWind Layout.

Part Name Incompatibility

When exporting the IDF board file, the mechanical design software exports only the decal name. SailWind Layout, however, exports both the part type and the decal.

While mechanical design software imports this information correctly, SailWind Layout cannot import the information from the mechanical design software. For example, SailWind Layout exports the following:

<decal name>	<part name>	<refdes>
DIP14	7400	U1

But the mechanical design software exports:

DIP14	DIP14	U1

Typically in SailWind Layout, decal and part names are different. To import the IDF board file into SailWind Layout, you must restore the part name field by editing the .emn file:

DIP14	7400	U1

Illegal Character Incompatibility

Many characters that are legal in SailWind Layout names are not legal in mechanical design systems.

When SailWind Layout exports part or decal names to the IDF files, it substitutes an underscore (_) for the following characters:

Forward slash (/), backward slash (\), period (.), exclamation point (!), at sign (@), pound sign (#), dollar sign (\$), percent sign (%), circumflex (^), ampersand (&), asterisk (*), parentheses (), plus sign (+), equal sign (=), pipe (|), comma (,), question mark (?), colon (:), semicolon (;), apostrophe ('), and quotation mark (")

For example, SailWind Layout writes the following to the IDF board (.emn) file for part type +5VREG with decal TO-220-UP:

<decal name>	<part name>	<refdes>
TO-220-UP _	5VREG	U5

SailWind Layout, however, will not find this part in the library when it tries to add it because of the name change. You must edit the IDF board (.emn) file to match the following to correctly add the part.

TO-220-UP	+5VREG	U5

IDF Import Messages

You may receive one or more messages during IDF file import.

The following are possible IDF import messages:

- **Component not found.**

The IDF data does not match the SailWind Layout data for part type, decal name, or reference designator. The data is ignored. Choose whether to continue importing.

- **Fatal error - Not a valid coordinate.**

The coordinates of one of the parts are outside of the work area. Return to the mechanical design software and correct the part placement. Then export the IDF file from the mechanical design software.

- **Fatal error - Self-intersecting polygon.**

The polygon that defines either a cluster keepout or the board outline intersects itself. Return to the mechanical design software and correct the self-intersecting polygon. Then export the IDF file from the mechanical design software.

- **Mechanical components found and ignored.**

SailWind Layout found mechanical components in the IDF file which are not imported into SailWind Layout.

- **This is an IDF library file. You can only import IDF board files.**

To ensure the integrity of your SailWind Layout library you cannot import .emp library files. Import the .emn board file instead.

- **This is not an IDF File.**

You are trying to import a file that is not in IDF. You can only import files with an .emn file extension using the IDF option.

- **Warning: Board cutout containing point <x,y> intersects existing board cutout containing point <x,y>. Original board outline restored.**

Click OK to continue importing and restore the original board outline.

- **Warning: Board cutout containing point <x,y> intersects the board outline. Original board outline restored.**

Click OK to continue importing and restore the original board outline.

- **Warning: Keepouts are imported. Existing keepouts are found of the following types, and will be deleted.**

When you import an IDF file into SailWind Layout, keepouts (placement, routing, or via) on the design are deleted and replaced with the same keepouts from the imported IDF file.

IDF Import Status File

As you import IDF data, SailWind Layout performs several checks to ensure the validity of the data and that current SailWind Layout database limits are not exceeded. SailWind Layout creates an import status file named *idfimport.sts* in *\SailWind Projects*. This file contains a log of the imported hole and component locations, including drill diameter, plated status, part type, and decal. This file also identifies whether the board outline was modified.

Since importing an IDF file can move and delete parts and keepouts, the translator checks to see whether route and reuse protection is violated, and reports the violations.

When the import status file is created, the message “IDF import finished with xx errors and xx warnings. Show report file?” appears.

[Drilled Hole Messages](#)

[Component Messages](#)

[Keepout Messages](#)

[Cutout Messages](#)

[Protected Trace Messages](#)

[Physical Design Reuse Messages](#)

Drilled Hole Messages

You may receive one or more messages regarding drilled holes during IDF file import.

The following are possible drilled hole messages:

- **Warning: (xxx,yyy) <drill diameter> mil <plated/nonplated> hole changed to <drill diameter> mil <plated | nonplated> hole.**

A hole exists at this location in both the design and the imported IDF file. Either the drill size or the plating status of the hole was changed. Drilled hole size and plating are not usually changed in the mechanical design software. Ignore the warning if this was your intention.

- **Warning: (xxx,yyy) <drill diameter> mil <plated/nonplated> hole added.**

A drilled hole exists at this location only in the imported IDF file, no corresponding hole exists in the design. This hole was added to the design. Drilled holes are not usually added in the mechanical design software. Ignore the warning if this was your intention.

- **Warning: (xxx,yyy) <drill diameter> mil <plated/nonplated> hole deleted.**

A hole exists at this location only in the design, no corresponding hole exists in the imported IDF file. This hole was deleted from the design. Drilled holes are not usually deleted in the mechanical design software. Ignore the warning if this was your intention.

- **Note: (xxx,yyy) <drill diameter> mil <plated/nonplated> hole unchanged.**

A hole exists at this location in both the design and the imported IDF file. The drill size and plating status of the hole are unchanged.

Component Messages

You may receive one or more messages regarding components in your design during IDF file import.

The following are possible component messages:

- **Error: <Ref Des> <parttype> <decal> (xxx,yyy) <Top/Bottom> xxx Imported part type is <parttype>.**

A component with this reference designator exists in both the design and the imported IDF file, and the part type is different. You cannot change the part type with IDF import. Correct the design or the IDF file.

- **Error: <Ref Des> <parttype> <decal> (xxx,yyy) <Top/Bottom> xxx Imported decal is <decal>.**

A component with this reference designator exists in both the design and the imported IDF file, and the decal is different. You cannot change the decal with IDF import. Correct the design or the IDF file.

- **Error: <Ref Des> <parttype> <decal> (xxx,yyy) <Top/Bottom> xxx Attempted to add this component, but part type <part type> is not found in design or library.**

SailWind Layout could not find the part type from the IDF file in either the design or the SailWind Layout library. The component cannot be added. Correct the IDF import file or add the part type to the library.

- **Error: <Ref Des> <parttype> <decal> (xxx,yyy) <Top/Bottom> xxx Attempted to add this component, but decal <decal> is not found in design or library.**

SailWind Layout could not find the decal from the IDF file in either the design or the SailWind Layout library. The component cannot be added. Correct the IDF import file or add the decal to the library.

- **Warning: <Ref Des> <parttype> <decal> (xxx,yyy) <Top/Bottom> <rotation> Added by IDF import.**

A component with this reference designator exists in the IDF file, but not in the design. The component was added to the design. Components are not usually added in the mechanical design software. If this was your intention, ignore the warning.

- **Warning: <Ref Des> <parttype> <decal> (xxx,yyy) <Top/Bottom> <rotation> Missing.**

A component with this reference designator exists in the design, but not in the IDF file. No change is made to the design. The component is not deleted. Components are not usually deleted in the mechanical design software. If this was your intention, ignore the warning.

- **Note: <Ref Des> <parttype> <decal> (xxx,yyy) <Top/Bottom> <rotation> Moved to (xxx,yyy) on layer <Top/Bottom> with rotation <rotation>.**

A component with this reference designator exists in both the design and the IDF file. The location, side, or rotation of the component is different. The component in the design was updated with the information from the IDF file.

- **Note:** <Ref Des> <parttype> <decals> (xxx,yyy) <Top/Bottom> <rotation> **Unchanged.**

A component with this reference designator exists in both the design and the imported IDF file. The location, side, and rotation of the component are all the same.

Keepout Messages

You may receive one or more messages regarding keepouts during IDF file import.

The following are possible keepout messages:

Message	Description
Warning: <Placement><Trace and copper><Via and jumper> keepout containing point <x,y> deleted from design.	Keepouts in SailWind Layout have been deleted and replaced with keepouts of the same type.

Cutout Messages

You may receive one or more messages regarding cutouts during IDF file import.

The following are possible cutout messages:

- **Warning: Board cutout containing point <x,y> intersects existing board cutout containing point <x,y>. Original board outline restored.**

Cutouts in the mechanical design software have been deleted and replaced with the original SailWind Layout cutouts.

- **Warning: Board cutout containing point <x,y> intersects the board outline. Original board outline restored.**

The board outline in the mechanical design software was replaced with the original SailWind Layout board outline.

Protected Trace Messages

You may receive one or more messages regarding protected traces during IDF file import.

The following are possible protected trace messages:

Message	Description
Warning: Component <name> is attached to protected traces. Protection is ignored, and component is moved.	The imported IDF file specifies moving a component regardless of it having protected traces attached to it.
Warning: Component <name> which represents a drilled hole has a signal attached, so it can't be deleted.	The imported IDF file specifies deleting a component. Importing cannot delete the component. Delete the component under ECO in SailWind Layout.

Physical Design Reuse Messages

You may encounter one or more messages regarding the reuse of physical designs.

The following are possible physical design reuse messages:

- **Warning: Component <name> is member of reuse <reuse name>. Reuse is dissolved, and component is moved.**

The imported IDF file specifies moving a component regardless of whether the component is part of a physical design reuse.

- **Warning: Hole <name> is member of reuse <reuse Name>. Reuse is dissolved, and hole is deleted.**

The imported IDF file specifies deleting a component regardless of whether the component is part of a physical design reuse. This only occurs for a part representing a drilled hole; no other parts are deleted by IDF import.

- **Warning: Keepout containing point (x,y) on layer <layer> is a member of reuse <reuse name>. Reuse is dissolved, and keepout is dissolved.**

The imported IDF file specifies deleting keepouts of certain types regardless of whether they are part of a physical design reuse.

IDF File Format

Both the IDF file and board file (.emn) must meet certain format requirements.

Details in the sections that follow apply to both the IDF 2.0 and IDF 3.0 formats, unless otherwise stated. The IDF 3.0 file format is described in detail in the [Intermediate Data Format Specification](#).

- [Panel Files](#)
- [Board Outline](#)
- [Other Outline](#)
- [Routing Outline](#)
- [Placement Outline](#)
- [Units](#)
- [Routing Keepout](#)
- [Via Keepout](#)
- [Placement Keepout](#)
- [Placement Group Area](#)
- [Drilled Holes](#)
- [Component Placement](#)
- [Library File Details](#)
- [Part Outlines](#)

Panel Files

Format IDF 3.0 supports an optional panel file. SailWind Layout does not support the panel file.

Board Outline

The IDF board outline consists of a single board outline. All objects are closed polygons made of arcs and lines.

The board outline describes the X and Y dimensions of the board. The Z dimension refers to the thickness of the board, not of 2D lines. To determine the thickness of the board, add the thickness of all substrate layers. View the thickness of layers by clicking the substrate layer in the Layer List on the Layer Thickness dialog box, and then viewing the Thickness text box in the Substrate/Prepreg area.

SailWind Layout supports cutouts in boards. Board cutouts are imported from and exported to IDF. The board outline and all dimensions are imported from and exported to IDF.

Board outlines and keepouts can be owned by the mechanical design software, SailWind Layout, or neither. IDF 3.0 ignores this information on import and always sets it to UNOWNED on export.

Other Outline

SailWind Layout does not support other outlines such as heatsinks or board cores; this information is not imported from or exported to IDF.

Routing Outline

SailWind Layout does not support routing outlines (outlines that designate areas for routing). This information is not imported from or exported to IDF.

Placement Outline

Each placement outline designates an area for placing components, in other words, a keepin. SailWind Layout does not support placement outlines; this information is not imported from or exported to IDF.

Units

SailWind Layout imports and exports design units to and from IDF.

The software converts design units to IDF units as shown in the following tables.

Table 28. Export to IDF Unit Conversion

SailWind Layout Unit	Conversion Multiplier	IDF Unit
Mils	1.0 THOU/mil	THOU
Inches	1.0e3 THOU/inch	THOU
Metric (mm)	1.0 MM/mm	MM

Table 29. Import from IDF Unit Conversion

IDF Unit	Conversion Multiplier	SailWind Layout Unit
THOU	1.0 mils/THOU	Mils
TNM (Ten nanometers) 10e-8 meters	10e-5 mm/TNM	Metric (mm)
MM	1.0 mm/MM	Metric (mm)

When importing THOU units, set the Design Units option in the Global tab of the Options dialog box to Mils. When importing TNM or MM units, set the Design Units option in the to Metric.

Routing Keepout

SailWind Layout supports polygon routing keepouts; this information is imported from and exported to IDF.

Via Keepout

Board level via keepouts specify regions on the board where you cannot place vias, but can route. Through vias are supported; blind and buried vias are not supported.

SailWind Layout supports via keepouts; this information is imported from and exported to IDF.

Placement Keepout

In IDF, board-level placement keepouts specify regions on the board where you cannot place components. A keepout can apply to all components or to only those components above or below a height you specify. Placement keepouts can exist on the top layer, bottom layer, or both layers. Each keepout consists of a simple closed curve made of arcs and lines along with minimum and maximum height restrictions.

SailWind Layout supports importing and exporting placement keepouts. Restrictions on exporting and importing placement keepouts include:

- You cannot export multiple, overlapping keepouts.
- You cannot export keepouts with the Inner setting.
- SailWind Layout converts keepouts that have the Both setting (in the mechanical design software) to All during importing.
- SailWind Layout ignores the minimum and maximum height settings during importing.

Placement Group Area

SailWind Layout does not support placement group areas (polygon areas intended for placement of a group of similar components). This information is not imported from or exported to IDF.

Drilled Holes

Drilled holes are specific mounting or tooling holes distinguishable from circular cutouts in the board outline because they are always drilled, instead of punched or routed, and you can plate them.

You can model holes using a single pin component with a nonzero drill diameter and either a HOLE text string on the decal or a HOLE attribute on the part type. These specially marked components are exported into the Drilled Holes Section of the IDF file, and the mechanical design software sees these as actual holes in the board. Drill oversize information is not included for plated holes, and oversize information does not apply to nonplated holes.

Component Placement

For component placement considerations, a location consists of an X,Y coordinate relative to the board origin, a rotation about the component origin, and a side of the board. Components are either placed or unplaced. You can fix the location of placed components to prevent changes in the mechanical design software. If you glue a component in SailWind Layout, it is considered *fixed*. If you do not glue a

component in SailWind Layout, it is considered *placed*. If an IDF part has the Unplaced attribute, SailWind Layout considers the part unglued and places it at 0,0.

All component information is exported to IDF. Only partial component information is imported from IDF. Refer to the following table for information on IDF attributes and their SailWind Layout equivalents.

SailWind Layout does not import changes to part type, reference designator, or decal name. The data is ignored.

IDF 3.0 includes a field in the placement section that represents the mounting offset for a component. SailWind Layout does not use the offset field. The height passed for the component should include any mounting offset as shown in the following table.

Table 30. Component Placement

IDF Property	SailWind Layout Equivalent	SailWind Layout Export	Modify	SailWind Layout Import
Package Name	Decal Name	Yes	No	No
Part Number	Part Type	Yes	No	No
Reference Designator	Ref Des	Yes	No	Check
X location	X location	Yes	Yes	Update
Y location	Y location	Yes	Yes	Update
Rotation Angle	Rotation Angle	Yes	Yes	Update
Side (Top or Bottom)	Side (Top or Bottom)	Yes	Yes	Update
Fixed	Glued	Yes	Yes	Update
Unplaced	No equivalent	No	Yes	Convert
Placed	Any unglued component	Yes	Yes	Update

Library File Details

SailWind Layout exports the IDF library file (.emp), which describes the geometric information for each part type and decal combination on the board. You cannot import into SailWind Layout any changes you make to a library file in the mechanical design software.

Refer to the following table for information about IDF terminology describing elements of the library file and their SailWind Layout equivalents.

Table 31. IDF Terminology for Library File Elements

IDF Terminology	SailWind Layout Equivalent
Geometry Name	Decal Name
Part Number	Part Type

Table 31. IDF Terminology for Library File Elements (continued)

IDF Terminology	SailWind Layout Equivalent
Units MM THOU TNM	Units Metric THOU No equivalent. IDF 3.0 does not support units of TNM (ten nanometers).
Component Height	The attribute Geometry.Height or the Decal text string ZHEIGHT=height is used if present. Otherwise, the value of Part Type attribute ZHeight is used. If there is still no height, the Missing Height dialog box appears.
Mechanical Component	No equivalent, mechanical components are modeled with single pin electrical components.
Component Outline	Level 20 Outline. If the outline is not present, the bounding box is used.

Part Outlines

Either SailWind Layout or the mechanical design software can “own” part outlines and keepouts, or else neither can. IDF 3.0 ignores this information during importing and always sets it to UNOWNED during exporting. Part outlines are never imported to SailWind Layout.

IDF 3.0 Enhancements

SailWind Layout has certain enhancements to the IDF 3.0 format related to importing and exporting.

- IDF 3.0 recognizes 360-degree circular cutouts.
- Multiple-pin parts and via drills are supported.
- A field representing the mounting offset for a component was added to the placement section. SailWind Layout does not use the offset field. The height passed for the component should include any mounting offset.
- A new optional panel file was added. SailWind Layout does not support panels.
- Board and part outlines and keepouts can be owned by the mechanical design software, SailWind Layout, or neither. IDF 3.0 ignores this information on import and always sets it to UNOWNED on export. Part outlines are never imported to SailWind Layout.
- Units of TNM (ten nanometers) are not supported.

- Since importing an IDF file can move and delete parts and keepouts, violations to route and reuse protection are checked and reported.
- Routing keepouts and via keepouts are supported.

Chapter 5

ECO File Commands

You can use ECO files to pass information between SailWind Layout and other tools.

You can use PADS-format ASCII files to pass data between SailWind Layout and the schematic tool. Some design modification data, such as design rules, are written to the PADS-format ASCII netlist file. For information about the PADS-format ASCII file format, see the SailWind Layout ASCII Format Specification.

**Note:**

This content does not apply to the integrated PADS Designer workflow.

When updating a design with changes from a schematic, you can pass the following data: connectivity changes, part type changes, placement changes, and design rule changes.

When updating a schematic with changes from SailWind Layout, you can pass the following data: reference designator renames, gate swaps, pin swaps, net splits/joins/renames, decal assignments, and design rule changes.

**Tip**

Some schematic capture tools may not support back annotation of all design changes.

[File Formats for Passing Data Between SailWind Layout and the Schematic Tool](#)

[Modification Data Types](#)

[Header and End of File Formats](#)

[Add Pin to Net Format](#)

[Delete Pin from Net Format](#)

[Exclude Nets Format](#)

[Join Two Nets Together Format](#)

[Exclude Connections Format](#)

[Split Net into Two New Nets Format](#)

[Rename Net Format](#)

[Add Part Format](#)

[Delete Part Format](#)

[Rename Part Format](#)

[Change Part Type Format](#)

[Change Decal Format](#)

[Move Part Format](#)

[Change Design Origin Format](#)

[Swap Two Gates Format](#)

[Swap Two Pins Format](#)

[General Attributes Format](#)

[Rule Change Formats](#)

[Example ECO File](#)

File Formats for Passing Data Between SailWind Layout and the Schematic Tool

You can use PADS-format ASCII files to pass data between SailWind Layout and the schematic tool. Some design modification data, such as design rules, are written to the PADS-format ASCII netlist file. For information about the PADS-format ASCII file format, see the SailWind Layout ASCII Format Specification.



Note:

This content does not apply to the integrated PADS Designer workflow.

When updating a design with changes from a schematic, you can pass the following data: connectivity changes, part type changes, placement changes, and design rule changes.

When updating a schematic with changes from SailWind Layout, you can pass the following data: reference designator renames, gate swaps, pin swaps, net splits/joins/renames, decal assignments, and design rule changes.



Tip

Some schematic capture tools may not support back annotation of all design changes.

Modification Data Types

Several types of modification information can be passed between a design and a schematic.

Table 32 summarizes the availability of each modification data type when transferring data between SailWind Layout and the schematic tool.

Table 32. Modification Data Types

Category	Data Type	Data Flow Direction Forward (to layout) or Backward (from layout)
Net Modification Commands	Add pin to net	Forward for all/Backward for SailWind Logic only
	Join two nets together	Forward/Backward
	Delete Pin from Net	Forward for all/Backward for SailWind Logic only
	Split net into two nets	Forward/Backward
	Rename net	Forward for all/Backward for SailWind Logic only
	Net scheduling (ePD only)	Forward

Table 32. Modification Data Types (continued)

Category	Data Type	Data Flow Direction Forward (to layout) or Backward (from layout)
Part Operations	Add part	Forward for all/Backward for SailWind Logic only
	Delete part	Forward for all/Backward for SailWind Logic only
	Change part type	Forward for all/Backward for SailWind Logic only
	Change decal	Forward for all/Backward for SailWind Logic only
	Rename part	Forward/Backward
	Move part	Forward for all/Backward for PADS Designer only
Gate and Pin Swapping	Swap two gates	Forward/Backward
	Swap two pins	Forward/Backward
Attributes	Add/Modify/Delete attributes	Forward/Backward
Design Rules	Add/Modify/Delete design rules (contained in PADS-format ASCII netlist file)	Forward/Backward

SailWind Layout can process all items and apply corresponding changes to the design when importing the ECO file (forward annotation).

Header and End of File Formats

The ECO file commands are similar to the PADS ASCII database format. Each type of modified data begins with a line of header information. The key information is located in a header line between asterisks (*).

The starting header, which appears at the beginning of the file, identifies the file as Forward/Backward Annotation file:

```
*PADS-ECO-V3.0[ -<units>] *
```

where <units> may be MILS, INCHES, or METRIC.

Once the header is found in an annotation file, the modification information following the header is processed.

The end of the file (EOF) information marker is:

```
*END*
```

An annotation file may have one or more entries from the ECO process, or it may contain more than one section of modification information. The file is read until the EOF marker is reached.

Add Pin to Net Format

When adding a pin to a net, the command must meet a specific format.

The header line is:

```
*NET*
```

The header line for the net to which you are adding the pin always follows this.

```
*SIGNAL* <netname> [<width>]
```

where:

<netname>	Net to which to add pins
<width>	Trace width with which to associate the connections. <width> is optional and maintained only for compatibility with older software versions. Note: In the case of adding a new net for forward annotation, if the trace width described in the *SIGNAL* line does not match the default value for the design, a special Clearance Rule containing the width value is created and attached to the net. Other rule settings match rule settings from the default SailWind Layout design.

Example signal name header:

```
*SIGNAL* VCC
```

The format for the pin information is just a list of pins separated by spaces, on one or more lines; for example:

```
Y4.3 U7.3 U6.1 U6.8 U9.1
```

To add pins to a new net, list the pins under a new *SIGNAL* header.

```
*SIGNAL* GND
```

```
U4.7 U6.8 U8.7 U7.8 U9.7
```

If the netname does not currently exist in the design, it is added.

Delete Pin from Net Format

When deleting a pin from a net, the command must follow a specific format.

The header line is:

```
*DELPIN* DELETE CONNECTIONS TO PIN
```

```
U17.1 CLK
```

where:

- “U17” is the part reference name.
- “1” is the pin number to disconnect.
- “CLK” is the net to which pin is connected (optional)

To delete multiple pins from the net, list the pins on separate lines under the same *DELPIN* header.

Exclude Nets Format

Commands for excluding nets must follow a specific format.

The header line is:

```
*EXCNET* MyClass
```

The format for the exclude information is:

```
A02  
A03  
A04
```

where A02, A03, and A04 are the names of the nets to exclude from the class named MyClass.

List the nets to remove from the class named in the header; only those nets listed are removed, all other nets remain with the class.

This applies to Net Class rules only.

Join Two Nets Together Format

The command for joining nets must follow a specific format.

The header line is:

```
*JOINNET*
```

The format for the join information is:

```
OLDNET0 OLDNET1
```

where OLDNET0 and OLDNET1 are the names of the nets to combine. The new combined net uses the OLDNET1 netname.

A connection is added between the nets using two random pins in the selected nets. The trace width of the added connection is the same as that of a connection in the first net (OLDNET0).

Exclude Connections Format

The command for excluding net connections must follow a specific format.

The header line is:

```
*EXCCON* GROUP_0
```

The format for the rename net information is:

```
J1.4 U1.9  
U6.2 U7.2
```

where J1.4 and U1.9, and U6.2 and U7.2 are the names of the pin pairs to exclude from the pin pair group named GROUP_0.

List the pin pairs to remove from the group named in the header; only those pin pairs listed are removed, all other pin pairs remain with the group.

Split Net into Two New Nets Format

The command for splitting a net must follow a specific format.

The header line is:

```
*SPLITNET*
```

The format for the split net information is:

```
*SIGNAL* CLK
```

where CLK is the old netname of the net that was split into two sections.

This is followed by the list of pins remaining in the old net:

```
U15.3 U14.3 U16.3 U5.3 U11.2
```

```
*SIGNAL* CLK-A
```

where CLK-A is the name of the new net followed by a list of pins that were separated from the old net.

```
U13.3 U12.3 U18.3 U19.3 U20.3 U21.3
```

To split multiple nets, list the old and new net pin lists under a new *SPLITNET* header.

Rename Net Format

The command for renaming a net must follow a specific format.

The header line is:

```
*RENNET*
```

The format for the rename net information is:

```
VCC +3.3V
```

where VCC is the old net name and +3.3V is the new net name.

To rename multiple nets, list the old and new net name pairs on separate lines under the same *RENNET* header.

Add Part Format

The command for adding a part to the design must follow a specific format.

The header line is:

```
*PART*
```

The format for the part information is:

```
U1 74LS00
```

where U1 is the part reference name and 74LS00 is the part type name.

Added parts are placed at the lower left corner of the board outline. If the board outline does not exist, parts are placed at the user-defined origin.

To add multiple parts, list them on separate lines under the same *PART* header.

Delete Part Format

The command for deleting or removing a part from the design must follow a specific format.

The header line is:

```
*DELPART*
```

The format for the part information is:

```
U4 74LS02
```

where U4 is the part reference designator to delete and 74LS02 is the part type name (not required).

To delete multiple parts, list them on separate lines under the same *DELPART* header.

**Tip**

Before deleting pins, disconnect them from nets in the design.

Rename Part Format

The command for renaming a part must follow a specific format.

The header line is:

```
*RENPART*
```

The format for the rename part information is:

```
U7 U1
```

where U7 is the old name and U1 is the new name.

To rename multiple parts, list them on separate lines under the same *RENPART* header.

All parts under the same *RENPART* header are renamed simultaneously; therefore, to swap the reference designators between two parts, the following input is accepted:

```
U1 U2
```

```
U2 U1
```

To facilitate renaming parts, duplicate names are not checked until all renaming is complete. This lets the example above run without a conflict over U2. If any error is encountered, parts in the list are not renamed.

Change Part Type Format

The command for changing a component's part type must follow a specific format.

The header line is:

```
*CHGPART*
```

The format for the change part information is:

```
U2 7400 7404
```

where:

- U2 is the reference name of the part.
- 7400 is the old part type.
- 7404 is the new part type.

To change multiple parts, list them on separate lines under the same *CHGPART* header.

Change Decal Format

The command for changing a decal format must follow a specific format.

The header line is:

```
*CHGPART*
```

The format for the change part information is:

```
U2 7400@DIP14 7400@SOIC14
```

where:

- U2 is the reference name of the part.
- 7400 is the old part type.
- 7400 is the new part type.
- DIP14 is the old decal.
- SOIC14 is the new decal.

To change multiple parts, list them on separate lines under the same *CHGPART* header.

Move Part Format

The command for moving a component must follow a specific format.

The header line is:

```
*MOVEPART*
```

The format for the move part information is:

```
U2 [LOC: -2300, -16700, 90] [FLP:ON] [GLU:OFF]
```

where:

- U2 is the reference name of the part.
- “LOC:-2300,-16700,90” is the new part location information: x coordinate, y coordinate, and orientation
- “FLP:ON” is the Flip side flag (ON to flip, OFF to not flip)
- “GLU:OFF” is the Glued flag (ON to glue, OfF to not glue)

**Tip**

: While each move part information parameter is optional, at least one parameter must be specified.

The part location coordinates are relative to the design origin.

The part orientation is specified in degrees. The orientation range is 0 degrees to 359.999 degrees, with a precision of 0.001 degree (values are rounded to the closest 0.001 degree).

To move multiple parts, list them on separate lines under the same *MOVEPART* header.

Change Design Origin Format

The command for changing the location of the design origin must follow a specific format.

The header line is:

```
*MOVEPART*
```

The format for the change design origin information is:

```
ORIGIN: -18000, -16250[, PERM]
```

where:

- “ORIGIN:-18000,-16250” is New design origin information: x coordinate and y coordinate.
- “,PERM” denotes whether to permanently change the design origin (PERM to change permanently, omit PERM to change only until the header of the next ECO command).

The ECO file units apply to the coordinates.

Swap Two Gates Format

The command for swapping gates must follow a specific format.

The header line is:

```
*SWPGATES*
```

The format for gate swapping information is:

```
U1 . A   U4 . B
```

where

- U1 and U4 are part reference designators.
- A and B are gates.

To swap multiple gates, list them on separate lines under the same ***SWPGATES*** header.

Gates are swapped sequentially, in the order listed.

Make sure that the part types of the parts are the same and that the gates are swappable. If they are not, they will still be swapped, but a warning message appears.

Swap Two Pins Format

The command for swapping pins must follow a specific format.

The header line is:

```
*SWPPINS*
```

The format for pin swapping is:

```
U5 1.2
```

```
U5 4.5
```

where:

- U5 is the reference designator for the part.
- 1.2 represents the pin pair to swap.
- 4.5 represents the other pin pair to swap.

You can swap pins only within a gate of the same part.

Make sure that the pins are swappable in the part type description. If they are not, they will still be swapped, but a warning message appears.

To swap pins on other parts, list the pin pairs under a new ***SWPPINS*** header.

General Attributes Format

The command for setting or adding object attributes must follow a specific format.

The following command adds or modifies a list of attributes for a single object:

```
*SET_ATTRIBUTE* <object type> <object name>
```

```
"<attr name>" <value>
```

where:

<object type>	Can be PIN#, PART, PARTTYPE, DECAL, NET, NETCLASS, or PCB
<object name>	Name for the corresponding object, DEFAULT for PCB type
<attr name>	Full, structured name of the attribute, enclosed by quotation marks
<value>	Attribute value written as a text string

For example:

```
*SET_ATTRIBUTE* DECAL DIP14
```

```
"Geometry.Height" 300
```

```
"CAM.AutoInsertable" NO
```

```
*SET_ATTRIBUTE* NETCLASS DATABUS
```

```
"CAE.Frequency" 66 MHz
```

You can list multiple attributes on separate lines under the same `*SET_ATTRIBUTE*` header. A space or spaces separate the attribute name and value strings. Because attribute names can contain spaces, quotation marks are required around the attribute name. The end of the line terminates the attribute value string, so quotation marks are not required around the attribute value string.

Because attribute names themselves can contain quotation marks, repeated quotation marks are used to represent a quotation mark within the attribute name. For example, a yes/no type decal attribute named Double"Quotes can be set in the ECO file with the following statements.

```
*SET_ATTRIBUTE* DECAL DIP14
```

```
"Double""Quotes" NO
```

Attribute types are not tracked in the ECO file. If, during the ECO process, a new attribute name is encountered, Free Text type is assigned and the default attribute hierarchy is applied depending on the object type.

[Table 33](#) represents default rules for adding new attributes to the Attribute Dictionary.

**Table 33. Default Rules for Adding New
Attributes to the Attribute Dictionary**

Object Type	Hierarchy Level
Pin	None
Net	Net Class
Net Class	None
PCB	None
Component	Part type Decal
Decal	None
Part Type	None

To delete a list of attributes for a single object:

```
*DEL_ATTRIBUTE* <object type> <object name>
```

```
<attr name>
```

Example:

```
*DEL_ATTRIBUTE* NET POWER
```

```
CAE.Voltage
```

Only the attribute name is required, therefore, the attribute name does not need quotation marks around it.

Rule Change Formats

The ECO file shows changes when you change design rules.

The following changes appear in the ECO file:

- Add an object to or delete it from a rule set (General Rules only)
- Change values for rules
- Add a rule to or delete it from a design
- Split a rule (General rules only). You might change a single rule with multiple object assignments in the original design to multiple rules with assignment subsets in the new design.

[Add Object Assignments to a Rule Set \(General Rules Only\)](#)

[Delete Object Assignments from a Rule Set \(General Rules Only\)](#)

[Changes in Values for a Rule](#)

[Add a Rule](#)

[Split a Rule](#)

[Sample Backward Annotation File](#)

[The Format of Rule Commands](#)

Add Object Assignments to a Rule Set (General Rules Only)

When you add object assignments to a general rule, the ECO file includes the *CREATE_GENERAL_RULES* statement and adds the name of the object to the list of all objects assigned to the rule.

For example:

```
*ADD_GENERAL_RULES* CLEARANCE
HIERARCHY_OBJECT NET:OLDNET1
HIERARCHY_OBJECT NET:OLDNET2
HIERARCHY_OBJECT NET:NEWNET
```

This example shows the output in the ECO file for a change in the new design that added a net (NEWNET) to the CLEARANCE rule set. In the original design, this rule set included only two members (OLDNET1 and OLDNET2).

Delete Object Assignments from a Rule Set (General Rules Only)

When you delete object assignments from a general rule, the ECO file shows the *DELETE_GENERAL_RULES* statement and lists a HIERARCHY_OBJECT statement for each object that was deleted.

For example:

```
*DELETE_GENERAL_RULES* CLEARANCE  
HIERARCHY_OBJECT NET:OLDNET2
```

This example shows the output in the ECO file for a change in the new design that deleted OLDNET2 from the CLEARANCE rule set.

Changes in Values for a Rule

When you modify values for a general or conditional rule, the comparison operation detects those changes. Values changes can occur when you assign objects to or delete object assignments from general rules.

When the comparison operation detects a values change between two matched rules, the ECO file includes the following:

- A *MODIFY_GENERAL_RULES* statement or a *MODIFY_CONDITIONAL_RULES* statement
- A HIERARCHY_OBJECT statement for each object assigned to the rule
- The entire set of values for the new rule

For example:

```
*MODIFY_GENERAL_RULES* CLEARANCE  
HIERARCHY_OBJECT NET:NET1  
  
MIN_TRACK_WIDTH 6  
REC_TRACK_WIDTH 8  
MAX_TRACK_WIDTH 12  
TRACK_TO_TRACK 12  
VIA_TO_TRACK 12  
VIA_TO_VIA 12  
PAD_TO_TRACK 12  
PAD_TO_VIA 12  
PAD_TO_PAD 12  
SMD_TO_TRACK 12  
SMD_TO_VIA 12  
SMD_TO_PAD 10  
SMD_TO_SMD 12  
...
```

The example shows a change in values for NET1. The output lists all of the values for NET1, including a changed value for SMD_TO_PAD in the new design (10 instead of 12).

Add a Rule

When you add a rule (where no rule of the same type and object assignments already exists), the rule addition appears in the ECO file.

ECO file includes the following:

- A *CREATE_GENERAL_RULES* statement or a *CREATE_CONDITIONAL_RULES* statement
- A HIERARCHY_OBJECT statement for each object assigned to the rule
- The entire set of values for the new rule

For example:

```
*CREATE_GENERAL_RULES*  CLEARANCE
HIERARCHY_OBJECT NET:NET3
MIN_TRACK_WIDTH 6
REC_TRACK_WIDTH 8
MAX_TRACK_WIDTH 12
TRACK_TO_TRACK 12
VIA_TO_TRACK 12
VIA_TO_VIA 12
PAD_TO_TRACK 12
PAD_TO_VIA 12
PAD_TO_PAD 12
SMD_TO_TRACK 12
SMD_TO_VIA 12
SMD_TO_PAD 12
SMD_TO_SMD 12
...
```

This example shows a change that added a new clearance rule and assigned NET3 to it.

Split a Rule

During comparison and update, a change can “split” a single general rule with multiple object assignments into multiple rules with assignment subsets. The comparison operation matches the “old” single rule with only one of the corresponding “new” rules. In addition, the operation modifies the matched rule and adds the unmatched new rules.

Sample Backward Annotation File

A sample backward annotation file is included for reference purposes.

```
*PADS-ECO-V3.0-mils*
```

REMARK

SIGNAL I 12

U17.16 U12.16

JOINNET

DATA0 DATA5

RENNET

SIGH SIGHB

RENNET

STROBEL CLK0

DATAA DATA0

DELPIN

P1.6 ENABLOW

U1.3 ENABLOW

P1.19 CLK

U11.A1 CLK

PART

C30 DCAP1

C31 DCAP1

DELPIN

U2.1 ADD0

U2.2 ADD1

U4.2 ADD1

U2.3 DATA0

U2.4 STROBB

U4.4 STROBB

U2.5 GND

U2.6 SIGH

P1.4 SIGH

U2.7 GND

U2.8 CLONE

U4.12 CLONE

U2.9 ZED011

P1.5 ZED011

U2.10 INT3

U2.13 ENABL0

U2.11 VCO

U2.12 GND

U2.14 VCC

DELPART

U5 7404

RENPART

U7 U4

U4 U7

CHGPART

U11 74007410

END

The Format of Rule Commands

This section covers the syntax used in the ECO file when design rules are created or modified during ECO mode operations. You can pass these rule entries from schematic capture systems that have rules capabilities or to schematic capture systems that have rules capabilities as part of the back annotation process. You can also import the rules into other PCB designs.

**Tip**

: When importing the ECO file into an existing design, rules entries are only applied to nets with the same name. Any nets that aren't found create an error report. Before importing, open the design and rename the target connections.

Heading information for the Rules section of the ECO file are illustrated in the Example ECO file.

[The Formatting for Design Rules in the ECO File](#)

[The Formatting for Conditional Clearance Rules in the ECO File](#)

[The Formatting for Conditional High-Speed Rules in the ECO File](#)

[The Formatting for Differential Pairs Rules in the ECO File](#)

[The Formatting for General Clearance Rules in the ECO File](#)

[The Formatting for General High-Speed Rules in the ECO File](#)

[The Formatting for General Routing Rules in the ECO File](#)

[The Formatting for Net Classes Rules in the ECO File](#)

[The Formatting for Pin Pair Group Rules in the ECO File](#)

The Formatting for Design Rules in the ECO File

Apply design rules at five hierarchical levels. The lowest level, called the default level, applies rules to all objects on the PCB not otherwise specified.

[Table 34](#) lists the design rule hierarchical levels.

Table 34. Design Rules Hierarchical Levels

Hierarchical Level	Design Rule Object
Default	HIERARCHY_OBJECT PCB:PCB
Net	HIERARCHY_OBJECT NET:<net_name>
Class	HIERARCHY_OBJECT CLS:<class_name>
Pin Pair	HIERARCHY_OBJECT CON:<pin_pair_name>
Group	HIERARCHY_OBJECT GRP:<group_name>

General Design Rules

Design rules often apply to a list of objects. For each object, the ECO file contains the notation:

```
HIERARCHY_OBJECT <type>:<name>
```

For example, if a rule applies to nets DATA0 through DATA3, the ECO file lists the nets this way:

```
HIERARCHY_OBJECT NET:DATA0
```

```
HIERARCHY_OBJECT NET:DATA1
```

```
HIERARCHY_OBJECT NET:DATA2
```

```
HIERARCHY_OBJECT NET:DATA3
```

Conditional Design Rules

Conditional rules apply from one object against another object. The ECO file uses the format:

```
<from_type>:<from_name> <against_type>:<against_name> [<layer_name>]
```

to refer to the from and against objects and the layer name for Conditional Rules. The brackets around the layer name indicate it is an optional argument.

For example:

```
NET:GND CLS:Clock_Nets
```

In the above example, the rule applies throughout all layers of the design. When the rule applies only to a single layer named TOP, the ECO file lists the objects this way:

```
NET:GND CLS:Clock_Nets TOP
```

[Conditional clearance](#) on page 972 and [conditional high-speed](#) on page 973 rules are supported. Conditional routing rules do not exist.

The Formatting for Conditional Clearance Rules in the ECO File

When you can create or modify a conditional clearance rule, the ECO file lists the change.

Such changes appear in the ECO file in the following format:

```
*CREATE_CONDITIONAL_RULES* CLEARANCE  
(or MODIFY_CONDITIONAL_RULES* CLEARANCE)
```

```
<from_type>:<from_name> <against_type>:<against_name> [<layer_name>]
```



```
TRACK_TO_TRACK <value>
VIA_TO_TRACK <value>
VIA_TO_VIA <value>
PAD_TO_TRACK <value>
PAD_TO_VIA <value>
PAD_TO_PAD <value>
SMD_TO_TRACK <value>
SMD_TO_VIA <value>
SMD_TO_PAD <value>
SMD_TO_SMD <value>
COPPER_TO_TRACK <value>
COPPER_TO_VIA <value>
COPPER_TO_PAD <value>
COPPER_TO_SMD <value>
TEXT_TO_TRACK <value>
TEXT_TO_VIA <value>
TEXT_TO_PAD <value>
TEXT_TO_SMD <value>
```

When you delete a conditional clearance rule, the ECO file lists the change using the following format.

```
*DELETE_CONDITIONAL_RULES* CLEARANCE
```

```
<from_type>:<from_name> <against_type>:<against_name> [<layer_name>]
```

The Formatting for Conditional High-Speed Rules in the ECO File

When you create or modify a conditional high-speed rule, the ECO file lists the change.

Such changes appear in the following format:

```
*CREATE_CONDITIONAL_RULES* HIGH_SPEED
(or MODIFY_CONDITIONAL_RULES* HIGH_SPEED)
```

```
<from_type>:<from_name> <against_type>:<against_name> [<layer_name>]
PARALLEL_LENGTH <value>
PARALLEL_GAP <value>
TANDEM_LENGTH <value>
TANDEM_GAP <value>
```

When you delete a conditional high speed rule, the ECO file lists the change using the following format:

```
*DELETE_CONDITIONAL_RULES* HIGH_SPEED
```

```
from_type>:<from_name> <against_type>:<against_name> [<layer_name>]
```

The Formatting for Differential Pairs Rules in the ECO File

When you create a Differential Pair or modify an existing one, the ECO file lists the change.

Such changes appear in the following format:

```
*CREATE_DIFFERENTIAL_PAIR* <object1> <object2>
```

```
GAP <value>  
MIN_LENGTH <value>  
MAX_LENGTH <value>
```

```
*MODIFY_DIFFERENTIAL_PAIR* <object1> <object2>
```

```
GAP <value>  
MIN_LENGTH <value>  
MAX_LENGTH <value>
```

```
*DELETE_DIFFERENTIAL_PAIR* <object1> <object2>
```

The Formatting for General Clearance Rules in the ECO File

When you create or modify a clearance rule, the ECO file lists the change.

Such changes appear in the ECO file using this format:

```
*CREATE_GENERAL_RULES* CLEARANCE  
(or *MODIFY_GENERAL_RULES* CLEARANCE)
```

```
HIERARCHY_OBJECT <type>:<name>  
MIN_TRACK_WIDTH <value>  
REC_TRACK_WIDTH <value>  
MAX_TRACK_WIDTH <value>  
TRACK_TO_TRACK <value>  
VIA_TO_TRACK <value>  
VIA_TO_VIA <value>  
PAD_TO_TRACK <value>  
PAD_TO_VIA <value>  
PAD_TO_PAD <value>  
SMD_TO_TRACK <value>  
SMD_TO_VIA <value>  
SMD_TO_PAD <value>  
SMD_TO_SMD <value>  
COPPER_TO_TRACK <value>  
COPPER_TO_VIA <value>  
COPPER_TO_PAD <value>  
COPPER_TO_SMD <value>
```

```
TEXT_TO_TRACK <value>
TEXT_TO_VIA <value>
TEXT_TO_PAD <value>
TEXT_TO_SMD <value>
OUTLINE_TO_TRACK <value>
OUTLINE_TO_VIA <value>
OUTLINE_TO_PAD <value>
OUTLINE_TO_SMD <value>
DRILL_TO_TRACK <value>
DRILL_TO_VIA <value>
DRILL_TO_PAD <value>
DRILL_TO_SMD <value>
SAME_NET_SMD_TO_VIA <value>
SAME_NET_SMD_TO_CRN <value>
SAME_NET_VIA_TO_VIA <value>
SAME_NET_PAD_TO_CRN <value>
DRILL_TO_DRILL <value>
BODY_TO_BODY <value>
```

When you delete a general clearance rule, the ECO file lists the change using the following format:

```
*DELETE_GENERAL_RULES* CLEARANCE
```

```
HIERARCHY_OBJECT <type>:<name>
```

The Formatting for General High-Speed Rules in the ECO File

When you create or modify a high-speed rule, the ECO file lists the change.

The change appears using the following format:

```
*CREATE_GENERAL_RULES* HIGH_SPEED
(or *MODIFY_GENERAL_RULES* HIGH_SPEED)
```

```
HIERARCHY_OBJECT <type>:<name>
MIN_LENGTH <value>
MAX_LENGTH <value>
STUB_LENGTH <value>
PARALLEL_LENGTH <value>
PARALLEL_GAP <value>
TANDEM_LENGTH <value>
TANDEM_GAP <value>
MIN_DELAY <value>
MAX_DELAY <value>
MIN_CAPACITANCE <value>
MAX_CAPACITANCE <value>
MIN_IMPEDANCE <value>
MAX_IMPEDANCE <value>
```

```
AGGRESSOR < ON | OFF>
SHIELD < ON | OFF > <gap_value> <net_name>
MATCH_LENGTH < ON | OFF > <tolerance_value>
```

When you delete a general high-speed rule, the ECO file list the change using the following format:

```
*DELETE_GENERAL_RULES* HIGH_SPEED
```

```
HIERARCHY_OBJECT <type>:<name>
```

The Formatting for General Routing Rules in the ECO File

When you create or modify a routing rule, the ECO file lists the change.

The change appears using the following format:

```
*CREATE_GENERAL_RULES* ROUTING
(or *MODIFY_GENERAL_RULES* ROUTING)
```

```
HIERARCHY_OBJECT <type>:<name>
LENGTH_MINIMIZATION_TYPE <NONE | TOTAL | HORIZONTAL | VERTICAL
SERIAL | PARALLEL | MID_DRIVEN >
DISPLAY_RATSNEST < ON | OFF >
COPPER_SHARE < ON | OFF >
AUTO_ROUTE < ON | OFF >
ALLOW_RIPUP < ON | OFF >
ALLOW_SHOVE < ON | OFF >
ROUTE_PRIORITY < ON | OFF >
VALID_LAYERS <layerName>...
VALID_VIA_TYPES <via_name>...
```

To delete general routing rules in the ECO file, use the following format. Do not supply parameters when deleting General Routing Rules.

```
*DELETE_GENERAL_RULES* ROUTING
```

```
HIERARCHY_OBJECT <type>:<name>
```

The Formatting for Net Classes Rules in the ECO File

When you create, rename, or delete net classes, the ECO file lists the change.

Such changes appear using the following formats:

```
*CLASS* <classname>
```

```
<net_name>. . .<net_name>
```

```
*RENCCLASS* <old_class_name> <new_class_name>
```

```
*DELCLASS* <class_name>
```

When you add nets to an existing class, the ECO file lists the change using the same format as the one for creating new class:

```
*CLASS* <existing_classname>
```

```
<net_name_to_add>
```

```
. . .
```

```
<net_name_to_add>
```

When you remove nets from an existing class, the ECO file lists the change using the following format:

```
*EXCNET* <classname>
```

```
<net_name_to_exclude>
```

```
. . .
```

```
<net_name_to_exclude>
```

The Formatting for Pin Pair Group Rules in the ECO File

When you create, rename, or delete a pin pair group, the ECO file lists the change.

Such changes appear in the following format:

```
*GROUP* <group_name>
```

```
<pin_pair_name>. . .<pin_pair_name>
```

```
*RENGROUP* <old_group_name> <new_group_name>
```

```
*DELGROUP* <group_name>
```

When you add pin pairs to an existing group, the ECO file lists the change using same format as the one for creating new group:

```
*GROUP* <existing_group_name>
```

```
<pin-pair_to_add>
```

```
. . .
```

```
<pin-pair_to_add>
```

When you remove pin pairs from an existing group, the ECO file lists the change using the following format:

```
*EXCCON* <group_name>
```

```
<pin-pair_to_exclude>
```

```
. . .
```

```
<pin-pair_to_exclude>
```

Example ECO File

An example ECO file is included for reference purposes.

```
*PADS-ECO-MILS*
*PADS-ECO* ENGINEERING CHANGE ORDER REPORT
*REMARK* -- DEMOA.PCB -- Mon Jan 03 12:00:00 2000
*REMARK* Create a net class containing 3 nets
*CLASS* CLASS1$$$1879$$$1906$$$1928
*REMARK* Rename it to CLASS2
*RENCCLASS*CLASS1 CLASS2
*REMARK* And delete it
*DELCLASS* CLASS2
*REMARK* Create a pin pair group containing 5 pin pairs
*GROUP* GROUP1U2.2 U1.2 U3.2 U2.2 U3.2 U4.2 U1.5 U2.5 U4.5 U1.5
*REMARK* Rename it to GROUP2
*RENGROUP*GROUP1 GROUP2
*REMARK* And delete it
*DELGROUP* GROUP1
*REMARK* Create default clearance rules for the board
*MODIFY_GENERAL_RULES* CLEARANCE
HIERARCHY_OBJECT PCB:PCB
MIN_TRACK_WIDTH 10
REC_TRACK_WIDTH 10
MAX_TRACK_WIDTH 12
TRACK_TO_TRACK 12
VIA_TO_TRACK 12
VIA_TO_VIA 12
PAD_TO_TRACK 12
PAD_TO_VIA 12
```

```
PAD_TO_PAD 12
SMD_TO_TRACK 12
SMD_TO_VIA 12
SMD_TO_PAD 12
SMD_TO_SMD 12
COPPER_TO_TRACK 12
COPPER_TO_VIA 12
COPPER_TO_PAD 12
COPPER_TO_SMD 12
TEXT_TO_TRACK 12
TEXT_TO_VIA 12
TEXT_TO_PAD 12
TEXT_TO_SMD 12
OUTLINE_TO_TRACK 12
OUTLINE_TO_VIA 12
OUTLINE_TO_PAD 12
OUTLINE_TO_SMD 12
DRILL_TO_TRACK 12
DRILL_TO_VIA 12
DRILL_TO_PAD 12
DRILL_TO_SMD 12
SAME_NET_SMD_TO_VIA 12
SAME_NET_SMD_TO_CRN 12
SAME_NET_VIA_TO_VIA 12
SAME_NET_PAD_TO_CRN 12
DRILL_TO_DRILL 12
BODY_TO_BODY 12
*REMARK* Create routing rules for net class CLASS1
*CREATE_GENERAL_RULES* ROUTINGHIERARCHY_OBJECT CLS:CLASS1
LENGTH_MINIMIZATION_TYPE VERTICAL
DISPLAY_RATSNEST OFF
COPPER_SHARE OFF
AUTO_ROUTE OFF
ALLOW_RIPUP OFF
ALLOW_SHOVE OFF
ROUTE_PRIORITY 3
VALID_LAYERS Top Bottom
VALID_VIA_TYPE STANDARDVIA
*REMARK* Create high speed rules for net $$$1963.Assign GND as a
*REMARK* shield net. (GND must first be assigned to a plane layer)
*CREATE_GENERAL_RULES* HIGH_SPEED
HIERARCHY_OBJECT NET:$$$1963
MIN_LENGTH 0
MAX_LENGTH 50000
STUB_LENGTH 1000
MIN_DELAY 0.000000
MAX_DELAY 10.000000
MIN_IMPEDANCE 50.000000
MAX_IMPEDANCE 150.000000
MIN_CAPACITANCE 0.000000
MAX_CAPACITANCE 10.000000
PARALLEL_LENGTH 1000
PARALLEL_GAP 200
```

ECO File Commands
Example ECO File

```
TANDEM_LENGTH 1000
TANDEM_GAP 200
AGGRESSOR OFF
SHIELD OFF 200 GND
MATCH_LENGTH OFF 200
*REMARK* Create high speed rules for pin pair group GROUP1
*CREATE_GENERAL_RULES* HIGH_SPEEDHIERARCHY_OBJECT GRP:GROUP1
MIN_LENGTH 0
MAX_LENGTH 50000
STUB_LENGTH 1000
MIN_DELAY 0.000000
MAX_DELAY 10.000000
MIN_IMPEDANCE 50.000000
MAX_IMPEDANCE 150.000000
PARALLEL_LENGTH 1000
PARALLEL_GAP 200
TANDEM_LENGTH 1000
TANDEM_GAP 200
AGGRESSOR OFF
*REMARK* Create routing rules for pin pair CON:U4.6-U1.6
*CREATE_GENERAL_RULES* ROUTINGHIERARCHY_OBJECT CON:U4.6-U1.6
AUTO_ROUTE OFF
```

```
ALLOW_RIPUP OFF
```

```
ALLOW_SHOVE OFF
```

```
ROUTE_PRIORITY 3
*REMARK* Create conditional clearance rules for net $$$1928
*REMARK* against net $$$1951 on all layers.
*CREATE_CONDITIONAL_RULES* CLEARANCENET:$$$1928 NET:$$$1951
TRACK_TO_TRACK 12
VIA_TO_TRACK 12
VIA_TO_VIA 12
PAD_TO_TRACK 12
PAD_TO_VIA 12
PAD_TO_PAD 12
SMD_TO_TRACK 12
SMD_TO_VIA 12
SMD_TO_PAD 12
SMD_TO_SMD 12
COPPER_TO_TRACK 12
COPPER_TO_VIA 12
COPPER_TO_PAD 12
COPPER_TO_SMD 12
TEXT_TO_TRACK 12
TEXT_TO_VIA 12
TEXT_TO_PAD 12
TEXT_TO_SMD 12
*REMARK* Create conditional high speed rules for net $$$2008
*REMARK* against net $$$2016 on all layers.
```



```
*CREATE_CONDITIONAL_RULES* HIGH_SPEEDNET:$$$2008 NET:$$$2016
PARALLEL_LENGTH 1000
PARALLEL_GAP 200
TANDEM_LENGTH 1000
TANDEM_GAP 200
*REMARK* Create conditional clearance rules for pin pair net U4.6-U1.6
*REMARK* against pin pair U2.6-U1.6. Applies only on layer BOTTOM.
*CREATE_CONDITIONAL_RULES* CLEARANCECON:U4.6-U1.6 CON:U2.6-U1.6 BOTTOM
TRACK_TO_TRACK 12
VIA_TO_TRACK 12
VIA_TO_VIA 12
PAD_TO_TRACK 12
PAD_TO_VIA 12
PAD_TO_PAD 12
SMD_TO_TRACK 12
SMD_TO_VIA 12
SMD_TO_PAD 12
SMD_TO_SMD 12
COPPER_TO_TRACK 12
COPPER_TO_VIA 12
COPPER_TO_PAD 12
COPPER_TO_SMD 12
TEXT_TO_TRACK 12
TEXT_TO_VIA 12
TEXT_TO_PAD 12
TEXT_TO_SMD 12
*REMARK* Delete the conditional clearance rules for pin pair net U4.6-U1.6
*REMARK* against pin pair U2.6-U1.6 on layer BOTTOM.
*DELETE_CONDITIONAL_RULES* CLEARANCECON:U4.6-U1.6 CON:U2.6-U1.6 Bottom
*REMARK* Create a differential pair of nets $$$1906 and $$$2008
*CREATE_DIFFERENTIAL_PAIR* NET:$$$1906 NET:$$$2008
GAP 10
MIN_LENGTH 10
MAX_LENGTH 100
*REMARK* Create differential pair of pin pairs U2.8-U1.8 and U4.6-U1.6
*CREATE_DIFFERENTIAL_PAIR* CON:U2.8-U1.8 CON:U4.6-U1.6
GAP 10
MIN_LENGTH 10
MAX_LENGTH 100
*END*
```


Chapter 6

Report Generation Language

You can create report formats using Report Generation Language (RGL). This language uses keywords for the information to extract, such as parts and pin numbers. Additional keywords control the appearance of the report file.

- [New Reports](#)
- [Searches and Loops](#)
- [Format File Structure](#)
- [Nested Loops](#)
- [Keywords](#)

New Reports

Use a text editor to create or modify format files. You must use the default *.fmt* extension and save the file to *C:\<install_folder>\<version>\Settings*. Before you can use a report format you created or modified, add the format file to the list in the Reports dialog box.

Searches and Loops

The FOR statement searches for information in a design. The FOR statement acts as a loop to continue searching until all of the specified items are found.

The FOR statement can search for specific information, such as components attached to VCC, or FOR can combine with the ALL, find all, statement. You can also embed wild card characters in the text string.

The following examples use FOR statements with a SIGNAL keyword:

```
FOR VCC SIGNALS
```

```
FOR ALL SIGNALS
```

```
FOR V* SIGNALS
```

The keyword that specifies what information to search for immediately follows the FOR statement. The first example searches for all components attached to signal VCC. This information is called the body of the statement and is enclosed by opening and closing braces ({}). The format to use is explained below.

You can classify searches as either a single-level search or a multi-level search. Use a single-level search for a simple component count. You can only access certain types of information by first searching for the parent, or top level information. For example, you cannot obtain the pin numbers attached to a specific signal unless you first search the signal name. The FOR statement uses nesting to accomplish this. For more information, see [“Nested Loops”](#) on page 985.

Format File Structure

A single-level search follows a basic file structure.

The basic file is as follows:

```
Menu name Keyword
# comments (This report file lists all components)
Header information to include in report
FOR statement for loop
{
body of statement
}
```

A pound sign (#) preceding any text denotes a comment string. The pound sign does not appear in the report but the comment will.

Place keywords in uppercase letters for recognition. Unrecognized text appears in the report file as is.

All text and database item fields placed in braces are considered part of the loop.

Spaces between text strings and data items in the format file appear in the report file.

[Table 35](#) shows an example of format file structure that creates a report containing a list of all components along with the package type. Keywords appear in bold.

Table 35. Example Format File Structure

Format File Structure	Description (not part of format file)
MENUTXT Component List	Text for list
# Component report	User comment
Parts List Report	Header information
Ref Des(no space)Part Type	Header information
FOR ALL COMPONENTS	Search all components
{	Open statement
COLUMNS 0 10	Format in columns
COMPNAME PKGNAME	List component and package names

Table 35. Example Format File Structure (continued)

Format File Structure	Description (not part of format file)
}	Close statement

The above format generates the following report:

Parts List Report

Ref Des Part Type

C1 CL25,10UF

C10 CK05, .01UF

U1 74139

Nested Loops

The nested loop structure is the same as the single-level search loop structure, but nested loops are contained completely within the loop. This includes the FOR statement, as well as the body of the statement which is enclosed by its own braces to define the inner level search.

The following format is an example of nested loop:

MENUTXT Net List Output File

Company XYZ required output format

Net List Report

FOR ALL SIGNALS

{

SIGNAME

FOR ALL PINS

{

MAXCOLS 5

```
BETWEEN 8
```

```
DELIMIT ,
```

```
COMPNAME, PINNAME
```

```
}
```

```
}
```



Tip

: The inner loop and its associated braces are indented here only for ease of reading. For information, see [“Field Keywords”](#) on page 990.

Keywords

You can use keyword types to handle single-level and multiple-level searches.

Keywords can be divided into top level, sublevel, and field types.

Table 36. Keyword Types for Single-Level and Multiple-Level Searching

Keyword	Description
Top level keywords	Access data items that exist as single entities. You can use top level keywords at any time.
Sublevel keywords	Search for items which exist as multiple entries and may need a nested FOR loop.
Field keywords	Produce a column-style format when used within the body of the loop. Each field keyword is followed by a space and the appropriate values.

Top Level Keywords

You can use top-level keywords when searching.

Table 37. Top-Level Search Keywords

Keyword	Returns
JOBNM	Job name
TIME	Current date and time
LAYERCNT	Total number of layers
PKGCNT	Total number of packages
SYMCNT	Total number of symbols
COMP CNT	Total number of components
EQUIV_IC	Equivalent IC count
BD_DENSITY	Board density in current units
SIGCNT	Total signal count
PSIGCNT	Power net signal count
SSIGCNT	Signal net count
BOARDSZ	Board size in current units

Table 37. Top-Level Search Keywords (continued)

Keyword	Returns
TOPCOMPCNT	Total number of components on top layer
BOTCOMPCNT	Total number of components on bottom layer
PADCNT	Total number of pads
DRPADCNT	Number of drilled pads
NDPADCNT	Number of undrilled pads

Sublevel Keywords

You can use sublevel keywords when performing a search.

Table 38. Sublevel Search Keywords

Keyword	Returns
SIGNALS	Search within signals.
SIGNAME	Name of signals.
PINS	Search within pins.
COMPNAME	Component name.
PINNUM	Pin number.
PINTYP	Pin type.
TPASSIGNED*	Whether test points exist in the signal: YES or NO.
TESTPOINTCNT*	Number of all test points in this signal.
TESTPINCNT*	Number of test point pins in this signal.
TESTVIACNT*	Number of test point vias in this signal.
SIGWITHTP*	Search for signals with test points.
SIGNOTP*	Search for signals without test points.
SIGNAME	Name of signals.
PINS	Search within pins.
COMPNAME	Component name.
PINNUM	Pin number.

Table 38. Sublevel Search Keywords (continued)

Keyword	Returns
PINTYP	Pin type.
CONNECTIONS	Search connections.
SIGNAME	Name of signals.
COMP1	Reference designator of connection end 1.
PIN1	Pin number of connection end 1.
COMP2	Reference designator of connection end 2.
PIN2	Pin number of connection end 2.
ROUTSEGS	Search within routes.
END1	First endpoint of segment.
END2	Second endpoint of segment.
WIDTH	Width of route segment.
LAYER	Layer number of route segment.
VIAS	Search within VIAs.
LOCX	X-coordinate of VIA.
LOCY	Y-coordinate of VIA.
VIANAME	VIA name.
PACKAGES	Search within packages.
PKGNAME	Package name.
PKGDESCR	Package description.
COMPONENTS	Search within packages.
COMPNAME	Component name.
SYMNAME	Symbol name.
PKGTYPE	Package name.
ANGLE	Component placement angle.
LOCX	X-coordinate of placement.
LOCY	Y-coordinate of placement.

Table 38. Sublevel Search Keywords (continued)

Keyword	Returns
TESTPOINTS	Search within test points.
TPNAME	Test point name. The test point name for component pins is the standard pin name, such as U1.2. The test point name for vias is the via type name, such as STANDARDVIA.
SIGNAME	Signal (net) name. When the test point is on an unused component pin, *NONE* is returned.
LOCX	X coordinate for test point.
LOCY	Y coordinate for test point.
TESTSIDE	Testing side for the test point: TOP or BOTTOM.

* You must have DFT Audit to create a report using keywords marked with asterisks.

Field Keywords

You can produce columns by using certain keywords within the body of the loop.

[Table 39](#) lists the keywords.

Table 39. Field Keywords

Keyword	Action
MAXCOLS	Determines the number of columns to use in the current loop.
LEADING	Specifies the starting character position for the first field.
BETWEEN	Specifies the number of characters to use as a separator between the first text character of each column.
COLUMNS	Specifies the starting print position for each column of data.
DELIMIT	Distinguishes keywords as entities separate from other fields, by delimiters which are usually spaces, but can be a character other than a space, such as a period.

Chapter 7

CAM Plus Supported Machine Formats

SailWind Layout supports specific machine formats.

- Siemens HS-180
- Siemens MS-72k
- Quad 100
- Daum SMF
- Zevatech PPM-9
- Universal Machine Output Files
- Universal Sequencer File
- Philips CSM Format
- Parts List
- Part Definition File Format

Siemens HS-180

CAM Plus supports the Siemens HS-180 machine format.

Output File Name = SIHS18xx.BP

As shown in [Table 40](#), the header information of the Siemens HS-180 output contains six lines of data:

Table 40. Siemens HS-180 Output File Header Information

File Name	Input ASCII File Name
U CAD-ASCII	The type of data
1.0	Version number
comment	General comments for example, PADSCIM OUTPUT
date	File creation date
num_parts	Number of parts inserted by the HS-180 machines, followed by the number of parts inserted by each machine in the line (not supported)

A single line entry, in the following format, exists for each part:

```
Line_number, Part_Type, X_loc, Y_loc, Rotation, Code, Ref_Name
```

[Table 41](#) describes the fields in each line entry.

Table 41. Siemens HS-180 Output File Line Entry Fields

Field	Description
Line_number	The order of the parts, starting with 1
Part_Type	The part type information; discrete parts include part values
X_IOC	The X insertion coordinate of the part (in mils)
Y_IOC	The Y insertion coordinate of the part (in mils)
Rotation	The rotation of the part in degrees
Code	(blank)
Ref_Name	The reference designator

Siemens MS-72k

CAM Plus supports the Siemens MS-72k machine format.

Output File Name = SIMS72xx.BP

As shown in [Table 42](#), the header information of the Siemens MS-72 output file contains ten lines of data.

Table 42. Siemens MS-72 Output File Header Information

Data	Description
BP/MS72	The type of data
V1.0	The version number
file name	Input ASCII file name, that is test.ast
comment	General comment, for example, PADSCIM OUTPUT
date	File creation date
component file	(left blank)
cluster file	(left blank)
ink spot	(always set to "0")

A single line entry, in the following format, exists for each part:

```
Line_number, Part_Type, X_loc, Y_loc, Rotation, Ref_Name, M_Code
```

[Table 43](#) describes the fields in each line entry.

Table 43. Siemens MS-72 Output File Line Entry Fields

Field	Description
Line_number	The order of the parts, starting with 1
Part_Type	The part type information. Discrete parts include part values
X_IOC	The X insertions coordinate of the part (in mils)
Y_IOC	The Y insertion coordinate of the part (in mils)
Rotation	The rotation of the part in degrees
Ref_Name	The reference designator
M_Code	The part is to be inserted, always set to asterisk (*)

Quad 100

CAM Plus supports the Quad 100 machine format.

Output File Name = QUADxx.100

There is no header information in the file. A single line entry, in the following format, exists for each part:

```
Ref_Name, X_loc, Y_loc, Rotation, Part_Num
```

Table 44 describes the fields in each line entry.

Table 44. Quad 100 Output File Line Entry Fields

Field	Description
Ref_Name	The reference designator
X_IOC	The X insertion coordinate of the part (in mils)
Y_IOC	The Y insertion coordinate of the part (in mils)
Rotation	The rotation of the part in degrees
Part_Num	Part number (from the Part Definition File)

Daum SMF

CAM Plus supports the Daum SMF machine format.

Output File Name = DAUMxx.SMF

Table 45 shows the header information of the DAUM.SMF output file, which contains 24 lines of data.

Table 45. Daum SMF Output File Header Information

Data	Description
Machine directory	Directory where part information is held in the Daum machine
;	
;	
PLACER;	Standard entry
Innoveda;	Standard entry
;	
;	
N;	Optimize output (Y or N)
N;	Optimize feeder info (Y or N)
2.54;	Scale factor of output
rotation;	Degrees of rotation
axis;	Axis of rotation (N = none)
height;	Height of boards
Board_Length;	Length of board (not used)
Board_Width;	Width of board (not used)
X_Offset;	X offset from board origin
Y_Offset;	Y offset from board origin
0;	Step & repeat (not used)
0;	Step & repeat (not used)
0;	Step & repeat (not used)
0;	Step & repeat (not used)
Number_Parts;	Number of parts to insert
Number_Feeder;	Number of feeders to use
Version 1.0;	Version number

A single line entry, in the following format, exists for each part:

```
Ref_Name, part_type, value, case_type, tolerance, trash_box, part_number,  
X_loc, Y_loc, rotation, feeder_num, feeder_type, current_component,  
feeder_num, stop-value;
```

Table 46 describes the fields in each line entry.

Table 46. Daum SMF Output File Line Entry Fields

Field	Description
Ref_Name	The reference designator
part_type	The part type information
value	Discrete parts include part values
case_type	The type of case
trash_box	Waste bin for bad parts
part_number	Part number (from the Part Definition File)
X_IOC	The X insertion coordinate of the part (in mils)
Y_IOC	The Y insertion coordinate of the part (in mils)
Rotation	The rotation of the part in degrees
Feeder_num	Feeder number as defined in PART.DEF.
feeder_type	Feeder type as defined in PART.DEF.
current_component	
Sequence number	
stop-value	Set to 0

The next section is the bin or feeder assignments. It contains the following entry for each unique part type:

```
feeder_number, feeder_type, component_type, value, case, tolerance,  
trash_box, tool, parts, 0, feeder_number;
```

Each entry is the same as described above.

Zevatech PPM-9

CAM Plus supports the Zevatech PPM-9 machine format.

Output File Name = ZEVAxx.PPM

The header information of the Zevatech output is one line of data, containing the version number and the date the report was generated. A single line entry, in the following format, exists for each part:

```
PLACE Ref_Name X_loc Y_loc Rotation 0 0 00 Part_Type
```

Table 47 describes the fields in each line entry.

Table 47. Zevatech PPM Output File Line Entry Fields

Field	Description
PLACE	The standard starting entry
Ref_Name	The part reference designator
X_loc	The X insertion coordinate of the part (in mils)
Y_loc	The Y insertion coordinate of the part (in mils)
Rotation	The rotation of the part in degrees.
0 0 00	Fixed entries
Part_Type	Part type from .ast file. Discrete parts include part values

Universal Machine Output Files

CAM Plus supports the Universal machine formats.

Table 48. Universal Machine Output Files

Machine	Output file
Universal 6241B Axial Inserter/Sequencer	UN6241xx.PUT
Universal 6348A Radial Inserter/Sequencer	UN6348xx.PUT
Universal 6772A Multi-Module Dip Inserter	UN6772xx.PUT
Universal 6287A Axial Inserter	UN6287xx.PUT

There is no header information in the machine program format. A single line entry, in the following format, exists for each part:

```
line_number> X_loc Y_loc z_span depth_stop anvil good bad MC CC
```

Table 49 describes the fields in each line entry.

Table 49. Universal Machine Output File Line Entry Fields

Field	Description
line_number	The line number in the file
X_loc	The X insertion coordinate of the part (in mils)
Y_loc	The Y insertion coordinate of the part (in mils)
z_span	The programmed z span entry
depth_stop	The insertion depth for the part
anvil	The anvil number and dispensing head number
good	The good count, initially set to 0
bad	The bad count, set to 0
MC	The alphabetic modifier code for the Z-span
CC	The insertion command code

The bin use report is a text file, with the name UN6241xx.BIN, also stored in *\SailWind Projects\cam*. This file is designed for use by the machine operator.

A single line entry, in the following format, exists for each part:

```
Part_name Bin_number Pre_allocation Check
```

Table 50 describes the fields in each line entry.

Table 50. Bin Use Report File Line Entry Fields

Field	Description
Part_name	Part to load in the bin, taken from the SailWind Layout part type
Bin_number	The bin assignment
Pre_allocation	Indicates if the bin was pre-assigned in the part.def file or was assigned by the output program
Check	Enables the machine operator to mark if the part is loaded. There is no field to determine whether the machine operator is loaded

Universal Sequencer File

The sequencer file is a Universal program with the name UN6287.SEQ, used to program a sequencer machine that is Sequencer 1 compatible. There is no header information in the file.

A single line entry, in the following format, exists for each part:

```
line> +000000 +000000 +000000 +000000 Bin_number +000000 +000000 B C
```

The following table describes the fields in each line entry.

Field	Description
line	The line number in the file
Bin_number	The bin number to load from

The sequence list is used to program other sequencing machines. It is an ASCII file containing the following information for each part:

```
Sequence: number Part Name: part_type
```

The following table describes the fields in each line entry

Field	Description
number	The insertion sequence number
part_type	The type of part to insert

Philips CSM Format

CAM Plus supports the Philips CSM machine format.

Output File Name = PHILIPxx.CSM

[Table 51](#) shows the header information.

Table 51. Phillips CSM Output File Header Information

Data	Description
*DESIGN	Standard entry for start of header information
Formatvs = 1	
Board = Name	
Twelvenc = 1234 567 89012	
Maskcntr =	
Issuedat =	
Revisdat =	
Userid =	

Table 51. Phillips CSM Output File Header Information (continued)

Data	Description
Unit = M (M for metric, or 1 for inches)	
*COMPONENT	Standard entry for end of header information

A single line entry, in the following format, exists for each part:

```
Ref_Name= (Part_Type{value},Part_Number,,X,Y,Rotation,Layer
```

Table 52 describes the fields in each line entry.

Table 52. Phillips CSM Output File Line Entry Fields

Field	Description
Ref_Name	The reference designator
Part_Type	The part type from the .ast; discrete parts include part values
Part_Number	Part number (from the Part Definition File)
X	The X insertion coordinate of the part (in mils)
Y	The Y insertion coordinate of the part (in mils)
Rotation	The rotation of the part in degrees
Layer	Top or Bottom layer

Parts List

This command outputs a general purpose parts list containing all parts on the board.

The parts list contains the following format:

```
*** CIM Status Report ***
```

```
Ref Part Name Level Inserted by:
```

Table 53 describes the fields in each line entry.

Table 53. General Purpose Parts List Line Entry Fields

Field	Description
Ref	The component reference name

Table 53. General Purpose Parts List Line Entry Fields (continued)

Field	Description
Part Name	The SailWind Layout library part type
Level	The layer, either Top or Bottom
Inserted by	The name of the machine for inserting the part

Part Definition File Format

The Part Definition File, located by default in *C:\<install_folder>\<version>\Libraries*, points particular part shapes to the specific machines that specialize in inserting them. For each part type that you insert automatically, an entry should exist in the part.def file. The entry is a single line, and the order in the file is not important. If an entry is not found, a warning is sent to the error file cimrep.err.

Usually the Read Part Definitions option is selected. The only time part.def would not be necessary is if you output to a selected machine.

[Format of the part.def File](#)

[Machine Codes](#)

[Bin Assignments](#)

Format of the part.def File

The entries in the part definition file are case sensitive. Also, no specific header data is required. Additional information depends on the machine inserting the part.

A single line entry, in the following format, exists for each part:

```
part_type, ins=machine_name,
```

[Table 54](#) describes the fields in each line entry.

Table 54. Part Definition File Line Entry Fields

Field	Description
part_type	The part type from SailWind Layout.
machine_name	The name of the insertion machine that will insert the part.

Machine Codes

Every part requires a specific machine code entry.

[Table 55](#) shows the six-character codes for the supported machines.

Table 55. Machine Codes

Machine	Machine Code
Dynapert 318	dynpro
Philips CMS	philip
Daum	daum
Siemens_HS_180	sims18

Table 55. Machine Codes (continued)

Machine	Machine Code
Siemens MS 72	sims72
Zevatech PPM-9	zeva
Quad 00	quad
Universal 6241	un6241
Universal 6348	un6348
Universal 6772	un6772
Universal 6287	un6287
uParts	(no insert)

Some insertion machines require additional data for each part. The order in which each data item is added is not important. Each entry must be separated by a comma (,).

Dynapert

```
number= part_number
```

In the above line, `part_number` represents the company part or stock number.

Siemens

Parts inserted by Siemens MS-72 and HS-180 require no additional data.

Daum

```
type=type_name, number=part_number, val=value, tol=tolerance, feedtype=
feeder, pip= nozzle_type.
```

Table 56 shows the fields for each line entry.

Table 56. Part Definition File Daum Line Entry Fields

Field	Description
type_name	Brief description of the part: resistor, capacitor, and so on.
part_number	The company part or stock number.
value	The field used to describe discrete part values.
tolerance	The field used to describe discrete part tolerances.
feeder	The type of feeder to use to insert the part; options are 1, 2, or 3.

Table 56. Part Definition File Daum Line Entry Fields (continued)

Field	Description
nozzle_type	The nozzle number used to insert the part; options are 1 through 6.

Quad

```
number= part_number
```

In the above line, part_number represents the company part or stock number.

Philips

```
number=part_number, val=value, tol=tolerance,
```

Table 57 shows the fields for each line entry.

Table 57. Part Definition File Phillips Line Entry Fields

Field	Description
part_number	The company part or stock number
value	The field used to describe discrete part values
tolerance	The field used to describe discrete part tolerances

Universal

```
bodydiam=body, leaddiam=lead, anvil=anvil, feednum= bin_number
```

Table 58 shows the fields for each line entry.

Table 58. Part Definition File Universal Line Entry Fields

Field	Description
body	The diameter size in mils
lead	The lead diameter in mils
anvil	The bend style
bin_number	The number of any pre-assigned bins

Zevatech

```
part_type, ins=zeva
```

Bin Assignments

Assigning parts to bins is an important function in CAM Plus. Parts must be loaded into the bins before the insertion process starts. This can take several hours for complex machines.

Commonly used parts are often kept permanently in some bins. You can define this in CAM Plus for Universal and Daum insertion machines when preassigned bins are used. Other parts are assigned to arbitrary bins.