PAIZI

SailWind Router Command Reference

Release SailWind 3.0
Document Revision 1.0

派兹

Copyright and Disclaimer of SailWind Software

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
Router Automation

This section contains the following topics.

Basic Scripting

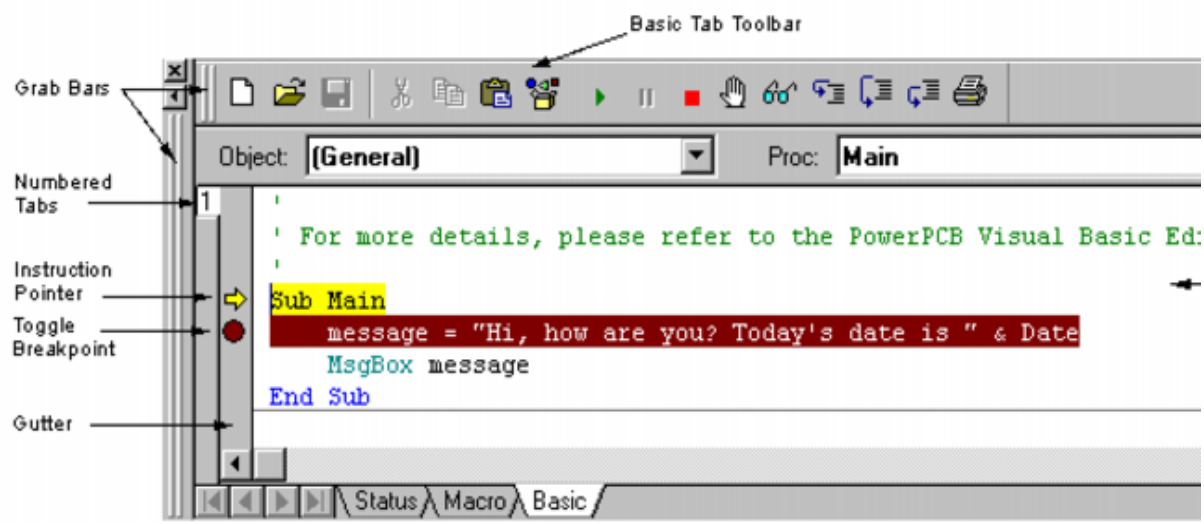
The program has an internal Basic scripting capability that you can use to create Basic scripts with limited programming knowledge.

The Basic editor includes the entire development environment required to develop Basic scripts including an editor, a debugger, an interpreter, a tracer, variable watch, a dialog editor, new/open/save/print capabilities, and an Automation object browser.

The Basic script editor used is the Sax Basic Engine™ by Sax Software Corporation. For more information on the Basic editor and language, see the Sax Basic Editor Help.

The figure below shows the screen elements of the Basic tab when you open a Basic script.

Figure 1. Screen Elements on the Basic Tab



Sample Basic Scripts

Sample Basic Scripts

Sample scripts are located in C:\SailWind Projects\Samples\Scripts\Router.

The following is a list of the sample basic scripts.

Table 1. Sample Basic Scripts

Script Name	Description
00-What is a Basic 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 Basic variable: how to assign a value and how to get its value.

Table 1. Sample Basic Scripts (continued)

Script Name	Description
03-Using a Basic Function.bas	Demonstrates how to use a standard Basic function and display its result in a message box.
04-Using a SailWind Router Function.bas	Demonstrates how to use a simple SailWind Router OLE function.
05-Using If and Then Statements.bas	Demonstrates the If, Then statement.
06-Using a Custom Dialog1.bas	Demonstrates a simple dialog box using the Basic dialog editor.
07-Using a Custom Dialog2.bas	Demonstrates a standard dialog box using the Basic dialog box editor.
08-Using a Custom Dialog3.bas	Demonstrates a complex dialog box using the Basic 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 it.
10-List Of Comps and Nets.bas	Lists all components and nets.
11-Select by Pin Count.bas	Allows you to enter a number of pins. All parts with that number of pins are selected.
12-Select All Test Points.BAS	Selects all test points.
13-Route All.bas	Routes all of the files in a directory that you specify in a dialog box that the script presents.

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).

Object Hierarchy

This section provides information about the Automation object hierarchy important to developers implementing application clients.

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

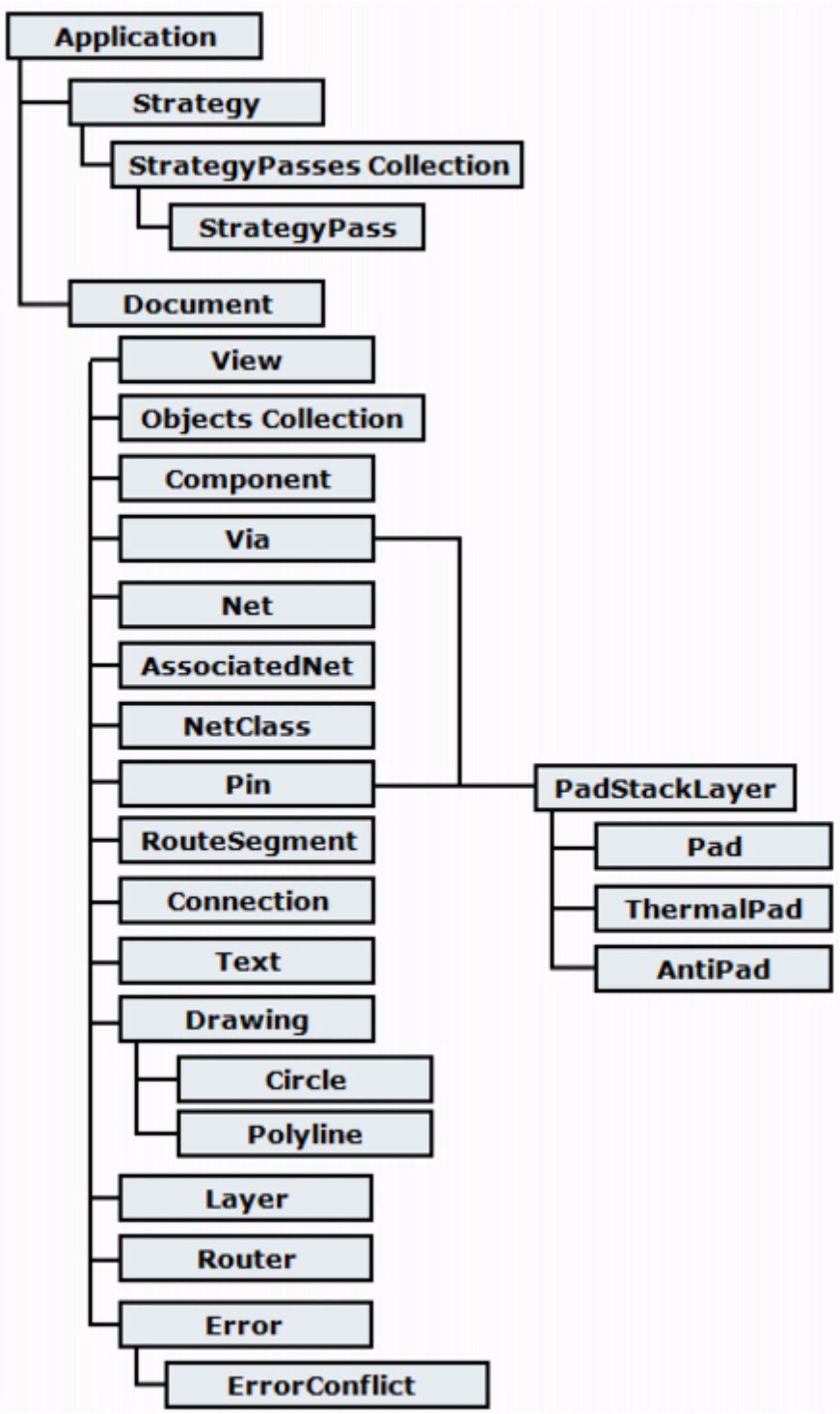
All program data objects, including the Component, Net, NetClass, Pin, and Via objects, are either accessed directly through the Document object or through the Objects Collection object. The Objects Collection object provides a convenient way to work with a set of data objects rather than with each object individually.

Tip
Virtual pins are treated as component pins.

All application Automation Objects implement their interface based on the Microsoft IDispatch interface.

The following diagram shows the application's Automation Server object hierarchy. This hierarchy follows the Microsoft OLE Automation Server guidelines.

Figure 2. Automation Server Object Hierarchy



Types

The following topics list the Router Automation types and their values.

blazeErrorClass

This list displays the Error Class types and their values.

Table 2. Error Class Types and Possible Values

Error Class Type	Possible Values
blazeErrorClassInvalid	-1
blazeErrorClassClearance	0
blazeErrorClassFabrication	1
blazeErrorClassTestability	2
blazeErrorClassConnectivity	3
blazeErrorClassLength	4
blazeErrorClassDiffpair	5
blazeErrorClassPowerPCB	6
blazeErrorClassMaxViaCount	7
blazeErrorClassNumber	8

blazeErrorType

This list displays the Error types and their values.

Table 3. Error Types and Possible Values

Error Type	Possible Values
blazeErrorTypeInvalid	-1
blazeErrorTypeClearance	0
blazeErrorTypeTraceWidth	1
blazeErrorTypeProfileCorner	2
blazeErrorTypeOverlapping	4
blazeErrorTypeBoard	1285
blazeErrorTypeKeepout	1543

Table 3. Error Types and Possible Values (continued)

blazeErrorTypeComponentKeepout	1544
blazeErrorTypeComponentCopper	9
blazeErrorTypeComponentText	10
blazeErrorTypeFabrication	4352
blazeErrorTypeAcid	4353
blazeErrorTypeSliver	4354
blazeErrorTypePadMinSize	4355
blazeErrorTypeTraceMinWidth	4356
blazeErrorTypeViaAtSMDFitInside	4357
blazeErrorTypeViaAtSMDCenter	4358
blazeErrorTypeViaAtSMDEnd	4359
blazeErrorTypeViaAtSMDCenterOut	4360
blazeErrorTypeViaAtSMDTooMany	4361
blazeErrorTypeHeightKeepout	5642
blazeErrorTypeNudgeKeepout	5643
blazeErrorTypeHeightBoard	5388
blazeErrorTypeHeightOverlapping	4365
blazeErrorTypeDrillToDrill	4366
blazeErrorTypeDrillHole	4367
blazeErrorTypeOnLayer	4368
blazeErrorTypeClearanceComponent	4369
blazeErrorTypeCompnentBoard	5394
blazeErrorTypeTestability	8704
blazeErrorTypeProbeNailCount	8705
blazeErrorTypeProbeSize	8706
blazeErrorTypeProbeLayer	8707
blazeErrorTypeProbeHole	8708

Table 3. Error Types and Possible Values (continued)

blazeErrorTypeProbeNotComponent	8709
blazeErrorTypeProbeGrid	8710
blazeErrorTypeProbeOnSMDPin	8711
blazeErrorTypeProbeClearance	8712
blazeErrorTypeProbeNailDiameter	8713
blazeErrorTypeProbeExclude	8714
blazeErrorTypeConnectivity	13056
blazeErrorTypeHighspeed	21504
blazeErrorTypeGapIrregularLength	21505
blazeErrorTypeGapViolation	21507
blazeErrorTypeGapObstacleSize	21508
blazeErrorTypeGapObstacleCount	21509
blazeErrorTypeLength	17408
blazeErrorTypeLengthTolerance	17409
blazeErrorTypeTolerance	17409
blazeErrorTypeMaxViaCount	30720

blazeErrorValueType

This list displays the Error Value types and their values.

Table 4. Error Value Types and Possible Values

Error Value Type	Possible Values
blazeErrorValueTypeMeasure	0
blazeErrorValueTypeInt	1
blazeErrorValueTypeDouble	2

blaze0bjectType

This list displays the object types and their values.

Table 5. Object Types and Possible Values

Object Type	Possible Values

Table 5. Object Types and Possible Values (continued)

blazeObjectTypeUnknown	0 The server returns this value to indicate an invalid object. The client may use this value when working with empty object collections.
blazeObjectTypeComponent	1
blazeObjectTypeNet	2
blazeObjectTypePin	3
blazeObjectTypeVia	4
blazeObjectTypeNetClass	9
blazeObjectTypePadStackLayer	31
blazeObjectTypePad	32
blazeObjectTypeThermalPad	33
blazeObjectTypeAntiPad	34
blazeObjectTypeLayer	35
blazeObjectTypeError	36
blazeObjectTypeErrorConflict	37
blazeObjectTypeAssociatedNet	38
blazeObjectTypeAll	9999 All Automation database object types, including Component, Net, NetClass, Pin, and Via.

blazeRouteComplete

This list displays the completion types and their values.

Table 6. Completion Types and Possible Values

Completion Type	Possible Values
blazeRouteCompleteUnknown	0
blazeRouteCompleteStrategyDone	1
blazeRouteCompleteStrategyPause	2
blazeRouteCompleteUserPause	3
blazeRouteCompleteOLEPause	4

Table 6. Completion Types and Possible Values (continued)

blazeRouteCompleteUserStop	5
blazeRouteCompleteOLEStop	6

Standard Answer Type

This list displays the standard answer types and their values.

Table 7. Standard Answer Types and Possible Values

Standard Answer Type	Possible Values
blazeDefaultAnswer	0 None
blazeOK	1
blazeCancel	2
blazeYes	3
blazeNo	4
blazeRetry	5
blazeAbort	6
blazeIgnore	7

blazeStrategyPassType

This list displays the strategy pass types and their values.

Table 8. Strategy Pass Types and Possible Values

Strategy Pass Type	Possible Values
blazeStrategyPassTypeUnknown	0
blazeStrategyPassTypeFanout	2
blazeStrategyPassTypePatterns	3
blazeStrategyPassTypeRoute	4
blazeStrategyPassTypeOptimize	5
blazeStrategyPassTypeMiters	6
blazeStrategyPassTypeTestPoints	7

Table 8. Strategy Pass Types and Possible Values (continued)

blazeStrategyPassTypeTune	8
blazeStrategyPassTypeCenter	9
blazeStrategyPassTypeAll	9999

blazeStrategyPassIntensity

This list displays the strategy pass intensity types and their values

Table 9. Strategy Pass Intensity Types and Possible Values

Strategy Pass Intensity Type	Possible Values
blazeStrategyPassIntensityUnknown	0
blazeStrategyPassIntensityLow	1
blazeStrategyPassIntensityMedium	2
blazeStrategyPassIntensityHigh	3

.

blazeTestPoint

This list displays the test point types and their values.

Table 10. Test Point Types and Possible Values

Test Point Types	Possible Values
blazeTestPointNone	0 No test point
blazeTestPointTopLayer	1 Test point on top layer
blazeTestPointBottomLayer	2 Test point on bottom layer

blazeUnit

This list displays the unit types and their values

Table 11. Unit Types and Possible Values

Unit Types	Possible Values
blazeUnitCurrent	0 Current unit in use in SailWind Router

Table 11. Unit Types and Possible Values (continued)

blazeUnitDatabase	1 Internal SailWind Router database unit
blazeUnitMils	2 Mils unit (1/1000th inch)
blazeUnitInch	3 Inch unit
blazeUnitMetric	4 Metric unit (1/1000th meter)
blazeUnitMicron	5 Micron unit

.

Running Code Samples

Many topics in the Automation Reference of the application provide Basic code samples to illustrate how to use an Automation property, method, or event. You can run code samples either in the application or outside of it.

The code sample is always in the following format:

```
Sub Main

   ' Do something

End Sub
```

For more information about running code samples, see the Troubleshooting the Code Samples and Enhancing Sample Code topics.

Disclaimer:

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

Running Code Samples in the Application

Run a sample using the Sax Basic Engine.

Procedure

1. Select the code sample, including the Sub Main and End Sub statements.
2. From the **Edit** menu, click **Copy** to copy the code to the Clipboard.
3. In the Status window, click the **Basic** tab to display the Sax Basic Engine editor.
4. From the **Edit** menu, click **Paste** to paste the code sample into the Basic editor.
5. In the Sax Basic Editor dialog box, click the **Run** button to run the sample code.

Running Code Samples outside the Application

Use the code sample with other Basic scriptable applications (also called host applications) such as recent versions of Visual Basic, Microsoft Excel, or Microsoft Word.

Procedure

1. Import the Automation server references of this application into the host application.
2. Paste the code sample into the Visual Basic Editor of the host application.
3. Add code to the beginning of the code sample to connect the host application to this application using the Visual Basic GetObject function.
4. Add the object returned by the Visual Basic GetObject function to the beginning of each Automation method and property of this application in the sample code.
5. Add the code to the end of the sample to disconnect the host application from this application.

Troubleshooting the Code Samples

If a code sample does not run correctly on your system, there are things you can check.

Table 12. Troubleshooting Chart

Check This	Because
Make sure a design is open in the application.	Almost all code samples require a design file open in the application when you run the sample.
Check for any assumptions made in the samples about the design files.	For example, some code samples assume that component U1 exists. These assumptions are clearly stated in the text preceding each sample. If these assumptions are not met, the sample code will not run properly. You can adapt the sample to your design file.
If you modified the sample code to run outside the application, make sure that you properly applied all required changes to the code, as described in Running Code Samples.	Common mistakes include forgetting to import application references and forgetting to prefix Automation constants of the application.

Disclaimer for the Code Samples

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

Enhancing Sample Code

The code samples in the application 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 application code.

The following list suggests possible enhancements:

- Add strict type checking, using the Option Explicit declaration at the beginning of the code and declare all variables using the Dim Visual Basic keyword. This ensures that the application interprets your variables properly and generates compiling errors when a problem exists.
- Use the On Error Visual Basic keyword to add error checking, thus improving how code reacts to a run-time error.
- Use the UserDialog Visual Basic keyword instead of the MsgBox keyword to send information to custom dialog boxes.

Disclaimer: Code Samples

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

Related Topics

Running Code Samples

Running Code Samples in the Application

Running Code Samples outside the Application

Troubleshooting the Code Samples

Automation Objects

The following list identifies the Automation objects.

- The AntiPad Object
- The Application Object
- The Associated Net Object
- The Circle Object
- The Component Object
- The Connection Object
- The Document Object
- The Drawing Object
- The Error Object
- The ErrorConflict Object
- The Layer Object
- The NetClass Object
- The Net Object
- The Objects Collection Object
- The Pad Object
- The PadStackLayer Object
- The Pin Object
- The Polyline Object
- The Router Object
- The RouteSegment Object
- The Strategy Object
- The StrategyPasses Collection Object
- The StrategyPass Object
- The Text Object
- The ThermalPad Object
- The Via Object
- The View Object

The AntiPad Object

This object represents an antipad in the padstack definition.

The following list identifies the Antipad properties:

Antipad.Application Property

This property returns the application object.

Usage

```
Application as Application
```

Arguments

None

AntiPad.CornerRadius Property

This property returns the Anti Pad corner radius.

Usage

```
CornerRadius as Double
```

```
      CornerRadius (unit as  blazeUnit on page 22 )  as Double
```

Arguments

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

AntiPad.CornerType Property

This property returns the Anti Pad corner type.

Usage

```
CornerType as blazePadCornerType
```

Arguments

None

AntiPad.Length Property

This property returns the Anti Pad length.

Usage

```
Length as Double
```

```
      Length (unit as blazeUnit on page 22)  as Double
```

Arguments

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

Antipad.Name Property

This property returns the name of the antipad.

Usage

```
Name as String
```

Arguments

None

Antipad.ObjectType Property

This property returns the type of the object - blazeObjectTypeAntiPad.

Usage

```
ObjectType as blazeObjectType on page 22
```

Arguments

None

AntiPad.Offset Property

This property returns the Anti Pad offset.

Usage

```
Offset as Double
```

```
      Offset  (unit as blazeUnit on page 22)  as Double
```

Arguments

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

AntiPad.Orientation Property

This property returns the Anti Pad orientation.

Usage

```
Orientation as Double
```

Arguments

None

Antipad.PadStackLayer Property

This property returns the PadStackLayer Object to which this anti pad belongs to.

Usage

```
PadStackLayer as blazePadStackLayerType
```

Arguments

None

Antipad.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

Antipad.Shape Property

This property returns the antipad shape.

Usage

```
Shape as blazeAntiPadShape
```

Arguments

None.

Antipad.Size Property

This property returns the antipad's size. For shape blazeAntiPadShapeRound, it returns diameter.

Usage

```
Size (unit as blazeUnit on page 22) as Double
```

Arguments

unit	[Optional] Unit in which the value is returned. This optional Argument is blazeUnitCurrent by default.

The Application Object

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

The Application object has the following properties, method, and events.

Application.ActiveDocument Property

This property returns the active document that represents the open design.

Usage

```
ActiveDocument as Document
```

Arguments

None

Examples

The following sample code retrieves the name of the open design using Document.Name Property. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  MsgBox "You are working with " & ActiveDocument.Name
End Sub
```

Related Topics

Application.OpenDocument Method

Document.Name Property

Application.ActiveStrategy Property

This property returns current routing strategy as a Strategy object. The active strategy represents the current routing strategy used by the router.

Usage

```
ActiveStrategy as Strategy
```

Arguments

None

Examples

The following sample code retrieves the number of passes from the routing strategy using Strategy.Passes. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  MsgBox "There are " & ActiveStrategy.Passes.Count & " routing passes in
 the strategy"
End Sub
```

Related Topics

Strategy.Passes Property

Application.Application Property

This property returns the Application object of this application. This property identifies the object as an Automation object of the application.

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 various sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

Usage

```
Application as Application
```

Arguments

None

Application.DefaultFilePath Property

This property sets or returns the path used to open design files. This property checks the default file folder entry in the registry database for the application.

When you set this property to a new value, the Registry setting is also changed. For example, \My Documents\SailWind Projects\Samples is the default path when you install using the default installation settings.

Usage

```
DefaultFilePath as String
```

Arguments

None

Examples

The following sample code changes the default application file path and notifies the client of this change. For more information about running this sample code, see Running Code Samples.

```
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

This property returns the filename of the application, including its path. For example, this function can return the string “C:\<install_folder>\<version>\Programs”.

Usage

```
FullName as String
```

Arguments

None

Examples

The following sample code displays the common name and the executable name. For more information about running this sample code, see Running Code Samples.

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

Related Topics

Application.Name Property

Application.Name Property

This property returns the name of the application. The Application.Name property is the default property for the Application object.

Usage

```
Name as String
```

Arguments

None

Examples

The following sample code displays the common name, the version, and the executable name. For more information about running this sample code, see Running Code Samples.

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

Related Topics

Application.FullName Property

Application.Version Property

Application.Parent Property

This property returns the parent of the object.

Usage

```
Parent As Application
```

Arguments

None

Application.Version Property

This property returns the version as a string with the following format: “major.minor,” for example “5.0”.

Usage

```
Version as String
```

Arguments

None

Examples

The following sample displays the application name and version. For more information about running this sample code, see Running Code Samples.

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

Related Topics

Application.FullName Property

Application.Name Property

Application.Visible Property

This property sets or returns whether the application is visible.

This property is used in the following cases:

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

Usage

```
Visible as Boolean
```

Arguments

None

Examples

The following sample makes the program invisible, waits one second, and then makes it visible again. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  Visible = False
  Wait 1
  Visible = True
End Sub
```

Related Topics

Application.Quit Method

Application.GetConfigParamInt Method

This method retrieves the integer value from the specified parameter and section in the registry.

Note: SailWind Router keeps configuration in the registry while SailWind Logic and SailWind Layout in INI files. However some parameters are also kept in the SailWindRouter.ini file, therefore this method first tries to read parameter from INI and if parameter is not found then it is read from the registry.

Usage

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

Arguments

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

Return Values

Parameter value or default value.

Examples

```
MsgBox Application.GetConfigParamInt("options\placement",
 "angular_increment", 1)
```

Application.GetConfigParamString Method

This method retrieves the string from the specified parameter and section in the registry.

Note: SailWind Router keeps configuration in the registry while SailWind Logic and SailWind Layout in INI files. However some parameters are also kept in the SailWindRouter.ini file, therefore this method first tries to read the parameter from INI and if parameter is not found then it is read from the registry.

Usage

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

Arguments

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

Return Values

Parameter value or default value.

Examples

```
MsgBox Application.GetConfigParamString("general", "editor", "default
 editor")
```

Application.ModelessCmd Method

This 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" for more information on running this sample.

```
Sub Main
```

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

```
End Sub
```

Application.OpenDocument Method

Opens a design file. The Application.OpenDocument method does not check, however, whether the currently opened file has been saved or not. Use the Document.Saved property to verify whether the open design must be saved.

Usage

```
OpenDocument (filename as String) as Document
```

Arguments

filename	Name of the file to open

If *filename* does not contain the full path to the file, the application uses the path specified by the Application.DefaultFilePath Property property to locate the file.

Return Values

If the function succeeds, the return value is the name of the newly opened document.

If the function fails, the return value is Nothing.

If *filename* is Nothing or an empty string, a new blank design file is created.

If the file specified by *filename* cannot be found or opened, the return value is Nothing.

This method generates an exception if an application security failure occurs during processing.

Examples

The following sample code opens PWRDEMOA.PCB, if it exists in the folder specified in the Application.DefaultFilePath Property property. Then the code displays the name of the newly opened file. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  OpenDocument DefaultFilePath & "\PWRDEMOA.PCB"
  MsgBox ActiveDocument.FullName & " has just been opened."
End Sub
```

Related Topics

OpenDocument

Application.DefaultFilePath Property

Application.OpenDocument Event

Document.Name Property

Document.Saved Property

Application.OpenDocument Event

This event occurs after the application opens a document.

Usage

```
OpenDocument (Doc as Document)
```

Arguments

Doc	Open document

Return Values

None

Related Topics

Application.OpenDocument Method

Application.ProgressChange Event

This event occurs after the routing progress changes.

Usage

```
ProgressChange (Percent as Long)
```

Arguments

Percent	Routing completion status in percents

Return Values

None

Application.Quit Method

This method shuts down the application.

CAUTION:
Do not use this method.

Microsoft requires that all Automation Application objects implement the Application.Quit 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 the program at that time, the program automatically shuts down. If an Automation client is a Basic script running in the Sax Basic Engine, it can never successfully shut down the program Automation server.

Usage

```
Quit
```

Arguments

None

Return Values

None

Related Topics

Application.Quit Event

Application.Quit Event

This event occurs before the application exits.

Usage

```
Quit
```

Arguments

None

Return Values

None

Related Topics

Application.Quit Method

Application.StandardDialog Event

This event occurs when additional standard dialog box input is required from the user during an automation API call. If the event handler is not implemented, the response in the dialog box assumes the default entry.

Usage

```
StandardDialog (question as String, answer as Long)
```

Arguments

Table 13. Application.StandardDialogEvent Arguments

Question	Answer
What string shows in the dialog box?	One of those enumerated by the SailWind Router Standard Answer Type on page 22. Initially, *answer* contains a default answer, which can be changed by a client to provide a response.

Return Values

None

The Associated Net Object

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

The following list identifies the Associated Net properties:

AssociatedNet.Name Property

This 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

This 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

This 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 on page 22
```

Arguments

None

Return Values

This property always returns blazeObjectTypeAssociatedNet.

AssociatedNet.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

AssociatedNet.Selected Property

This property sets or determines whether the associated net is selected or not. You can also select an application database object using the Document.SelectObjects and Objects.Select 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 Running Code Samples.

```
Sub Main

   ActiveDocument.SelectObjects(,,False)

   ActiveDocument.AssociatedNets("$$$1*$$$2").Selected = True

End Sub
```

Related Topics

Document.SelectionChange Event

Document.SelectObjects Method

Objects.Select Method

The Circle Object

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

The following list identifies the Circle properties:

Circle.Application Property

This property returns the Application object.

Usage

Application as Application

Arguments

None

Description

This is a Microsoft-required property.

Circle.CenterX Property

This property returns the x-coordinate of the circle's center.

Usage

CenterX (*Unit* as blazeUnit on page 22, *Origin* as blazeOriginType) as Double

Arguments

unit	[Optional] Unit in which the center X value is returned. This optional argument is blazeUnitCurrent by default.
origin	[Optional] Type of reference point from which the result is counted. The default value is blazeOriginTypeDesign.

Description

None

Examples

The following sample code shows the position of the selected circle.

```
Sub Main

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

For Each drw In selected

Dim geom as Object

For Each geom In drw.Geometry

If geom.ObjectType = blazeObjectTypeCircle Then

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

End If

Next geom

Next drw

End Sub
```

Circle.CenterY Property

This property returns the y-coordinate of the circle's center.

Usage

CenterY (*Unit* as blazeUnit on page 22, *Origin* as blazeOriginType) as Double

Arguments

unit	[Optional] Unit in which the center Y value is returned. This optional argument is blazeUnitCurrent by default.
origin	[Optional] Type of reference point from which the result is counted. The default value is blazeOriginTypeDesign.

Description

None

Examples

The following sample code shows the position of the selected circle.

```
Sub Main

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

For Each drw In selected

Dim geom as Object

For Each geom In drw.Geometry

If geom.ObjectType = blazeObjectTypeCircle Then

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

End If

Next geom

Next drw

End Sub
```

Circle.Geometry Property

This 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.

```
Sub Main

Set selected = ActiveDocument.GetObjects(blazeObjectTypeDrawing,,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

This 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.

```
Sub Main

Set selected = ActiveDocument.GetObjects(blazeObjectTypeDrawing,,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

This property returns the line style of the circle.

Usage

LineStyle as blazeLineStyle

Arguments

None

Description

None

Examples

None

Circle.LineWidth Property

This property returns the line width of the circle.

Usage

LineWidth (*Unit* as blazeUnit on page 22) as Double

Arguments

unit	[Optional] Unit in which the line width value is returned. This optional argument is blazeUnitCurrent by default.

Description

None

Examples

The following sample code shows the line width of the selected circle.

```
Sub Main

Set selected = ActiveDocument.GetObjects(blazeObjectTypeDrawing,,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

This property returns the type of object.

Usage

ObjectType As blazeObjectType on page 22

Arguments

None

Description

None

Examples

The following sample code tests the ObjectType property.

```
Sub Main

For Each drw In ActiveDocument.Drawings

For Each geom In drw.Geometry

type = geom.ObjectType

If type <> blazeObjectTypePolyline And type <> blazeObjectTypeCircle Then

MsgBox "Test failed"

End If

Next geom

Next drw

End Sub
```

Circle.OutlineType Property

This property returns the outline type of the circle.

Usage

OutlineType as blazeOutlineType

Arguments

None

Description

None

Examples

The following sample code shows the outline type of the selected circle.

```
Sub Main
Set selected = ActiveDocument.GetObjects(blazeObjectTypeDrawing,,TRUE)
For Each drw In selected
Dim geom as Object
For Each geom In drw.Geometry
Select Case geom.OutlineType
Case blazeOutLineTypeCenter
s="Center line"
Case blazeOutLineTypeOuter
s="Outer line"
Case blazeOutLineTypeInner
s="Inner line"
```

```
End Select

MsgBox "Outline type: " & s

Next geom

Next drw

End Sub
```

Circle.Parent Property

This property returns the parent of the object.

Usage

Parent as Document

Arguments

None

Description

This is a Microsoft-required property.

Circle.Radius Property

This property returns the value of the radius of the circle.

Usage

Radius (*Unit* as blazeUnit on page 22) as Double

Arguments

Unit	[Optional] Unit in which the radius value is returned. This optional argument is blazeUnitCurrent by default.

Description

None

Examples

The following sample code shows the radius of the selected circle.

```
Sub Main

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

For Each drw In selected

Dim geom as Object

For Each geom In drw.Geometry

If geom.ObjectType = blazeObjectTypeCircle Then

MsgBox "Radius: " & geom.Radius

End If

Next geom

Next drw

End Sub
```

Circle.ShapeType Property

This property returns the shape type of the circle. The value of blazeShapeOpen is not applicable.

Usage

ShapeType as blazeShapeType

Arguments

None

Description

None

Examples

The following sample code shows the shape type of the selected circle.

```
Sub Main

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

For Each drw In selected

Dim geom as Object

For Each geom In drw.Geometry

Select Case geom.ShapeType

Case blazeShapeTypeOpen

s="Open"

Case blazeShapeTypeHollow

s="Hollow"

Case blazeShapeTypeFilled

s="Filled"
```

```
Case blazeShapeTypeVoid

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, which exists in the currently open design.

The following list identifies the Component Object properties:

Component.Application Property
Component.Glued Property
Component.IsSMD Property
Component.Layer Property
Component.Name Property
Component.ObjectType Property
Component.Orientation Property
Component.Parent Property
Component.Pins Property
Component.PositionX Property
Component.PositionY Property
Component.Selected Property

Component.Application Property

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

Usage

```
Application as Application
```

Arguments

None

Component.Glued Property

This property returns if the component is glued.

Usage

```
Glued as Boolean
```

Arguments

None

Component.IsSMD Property

This property determines whether or not the component is 100%SMD. When a component is 100% SMD, it means that all of its pins are Surface Mounted Device (SMD) pins.

Usage

```
IsSMD as Boolean
```

Arguments

None

Examples

The following sample code retrieves the number of SMD components in the open design. For more information about running this sample code, see Running Code Samples.

```
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.Layer Property

This property returns the mounting layer of the component.

Usage

```
Layer as Long
```

Arguments

layer	Layer to set

This property generates an exception if *layer* is invalid or is not a component layer.

Component.Name Property

This property returns the name of the component. For example, this property returns the string “U1” for component U1. The Component.Name property is the default property for the Component object.

Usage

```
Name as String
```

Arguments

None

Component.ObjectType Property

This property returns the type of the object. This property always returns blazeObjectTypeComponent. All application database objects in the application 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 on page 22
```

Arguments

None

Component.Orientation Property

This property returns the orientation of the component.

Usage

```
Orientation as Double
```

Arguments

None

Return Values

This method generates an exception if an application security failure occurs during processing.

Related Topics

Component.PositionX Property

Component.PositionY Property

Component.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

Component.Pins Property

This property returns the collection of all pins in the component.

Usage

```
Pins as Objects
```

```
Pins (name as String) as Pin
```

Arguments

name	[Optional] Name of an existing pin

Return Values

When an existing pin name is passed to this property, it returns that Pin object. If the pin name does not exist, this property returns the collection of all pins of the component in an Objects collection object.

Examples

The following sample code retrieves the number of pins in component U1, assuming the component exists in the open design. For more information about running this sample code, see Running Code Samples.

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

Component.PositionX Property

This property returns the x-coordinate of the component.

Usage

```
PositionX ([unit as blazeUnit on page 22]) as Double
```

Arguments

unit	[Optional] Unit in which the x-coordinate is returned

Examples

The following sample code retrieves the location of component U1, assuming the component exists in the open design, in current design units. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  Set compU1 = ActiveDocument.Components("U1")
  MsgBox "U1 position is = (" & compU1.PositionX & ", " & compU1.PositionY
 & ")"
End Sub
```

Related Topics

Component.PositionY Property

Component.PositionY Property

This property returns the y-coordinate of the component.

Usage

```
PositionY ([unit as blazeUnit on page 22]) as Double
```

Arguments

unit	[Optional] Unit in which the y-coordinate is returned

Examples

The following sample code retrieves the location of component U1, assuming it exists in the open design, in current design units. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  Set compU1 = ActiveDocument.Components("U1")
  MsgBox "U1 position is = (" & compU1.PositionX & ", " & compU1.PositionY
 & ")"
End Sub
```

Related Topics

Component.PositionX Property

Component.Selected Property

This property sets or determines whether the component is selected or not. You can also select an application database object using the Document.SelectObjects and Objects.Select methods.

Usage

```
Selected as Boolean
```

Arguments

None

Examples

The following sample code selects component U1 only, assuming it exists in the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  ActiveDocument.SelectObjects(,,False)
  ActiveDocument.Components("U1").Selected = True
End Sub
```

Related Topics

Document.SelectionChange Event

The Connection Object

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

The following list identifies the Connection 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

This property returns the Application object.

Usage

Application As Application

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

This property returns the length of the connection.

Usage

Length([*unit* As blazeUnit on page 22]) As Double

Arguments

unit	[Optional] 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.

```
Sub Main
```

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

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

```
End Sub
```

Connection.Name Property

This 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.

```
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 PADS 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(blazeObjectTypeAll, , False)

ActiveDocument.SelectObjects(blazeObjectTypeConnection,
 ListConns(SuppValue%))

End If

End Select

End Function
```

Connection.Net Property

This property returns the net connected to the connection.

Usage

Net As Net

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.

```
Sub Main
```

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

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

```
End Sub
```

Connection.ObjectType Property

This property returns the type of the object.

Usage

ObjectType as blazeObjectType on page 22

Arguments

None

Description

This property returns blazeObjectTypeConnection.

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 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 blazeObjectTypeComponent

' Do something specific to component objects

Case blazeObjectTypeNet

' Do something specific to net objects

Case blazeObjectTypePin

' Do something specific to pin objects

Case blazeObjectTypeVia

' Do something specific to via objects
```

```
Case blazeObjectTypeConnection

' Do something specific to connection objects

Case blazeObjectTypeRouteSegment

' Do something specific to route segment objects

Case blazeObjectTypeJumper

' Do something specific to jumper objects

Case Else

MsgBox "Not a PADS Layout database object"

End Select

End Sub
```

Connection.Parent Property

This property returns the parent of the object.

Usage

Parent as Document

Arguments

None

Description

None

Connection.Pins Property

This property returns the collection of all pins and virtual pins in the connection.

Usage

Pins As Objects

Pins(*name* As String) As Pin

Arguments

name	Name of an existing pin.

Description

When an existing pin name is passed to this property, it returns that Pin on page 281 object. If the pin name does not exist, this property returns the collection of all pins of the connection in an Objects on page 241 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.

```
Sub Main
```

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

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

```
End Sub
```

Connection.RouteSegments Property

This property returns the collection of all trace segments in the connection.

Usage

RouteSegments As Objects

RouteSegments(*name* As String) As RouteSegment

Arguments

name	Name of an existing trace segment.

Description

When an existing trace segment *name* is passed to this property, it returns that RouteSegment object. If the trace segment *name* does not exist, this property returns the collection of all route segments in the connection in an Objects 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.

```
Sub Main
```

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

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

```
End Sub
```

Connection.Selected Property

This 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.

```
Sub Main
```

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

```
ActiveDocument.Connections(1).Selected = True
```

```
End Sub
```

Related Topics

Document.SelectionChange Event

Connection.Vias Property

This property returns the collection of all vias in the connection.

Usage

Vias As Objects

Vias(*name* As String) As Via

Arguments

name	Name of an existing via.

Description

When an existing via *name* is passed to this property, it returns that Via object. If the via *name* does not exist, this property returns the collection of all vias in the collection in an Objects 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.

```
Sub Main
```

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

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

```
End Sub
```

The Document Object

The Document object represents a PCB design file that is currently open.

This object is usually retrieved using the Application.OpenDocument property. The following list identifies the Document Object properties, methods, and events:

Document.Activate Method

This method activates the window associated with a document. This is a Microsoft requirement, but because the application is a Single Document Interface (SDI) server application, this function has no effect.

Usage

```
Activate
```

Arguments

None

Return Values

None

Document.ActiveLayer Property

This 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" for more information on running this sample.

```
Sub Main
```

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

```
ActiveDocument.ActiveLayer = 2
```

```
End Sub
```

Document.Application Property

This property returns an Application object of the application. This property identifies the object as an Automation object of the application. 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 various sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

Usage

```
Application as Application
```

Arguments

None

Document.AssociatedNets Property

This 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.Components Property

This property returns the collection of all components.

Usage

```
Components as Objects
```

```
Components (name as String) as Component
```

Arguments

name	Name of an existing component

Return Values

When an existing component name is passed to this property, it returns the Component object. If the component name does not exist, this property returns the collection of all components in an Objects collection object.

Examples

The following sample code retrieves the number of components in the open design using the Objects.Count Property property. For more information about running this sample code, see Running Code Samples.

```
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 (if it exists in the open design) using the Component.Pins Property and Objects.Count Property properties. For more information on running this sample, see Running Code Samples.

```
Sub Main
  Set compU1 = ActiveDocument.Components("U1")
  MsgBox "Component " & compU1.Name & " has " & compU1.Pins.Count & "
 pin(s)."
End Sub
```

Related Topics

Component.Pins Property

Document.GetObjects Method

Objects.Count Property

Document.Errors Property

This property returns the number of electrical layers in the design.

Usage

Errors as Objects

Errors (errorNumber as Integer) as Error

Arguments

errorNumber	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

This property returns the filename (including path) of the document.

Usage

```
FullName as string
```

Arguments

None

Examples

The following sample code retrieves the name and location of the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  MsgBox "Hi, you are using " & ActiveDocument.FullName & " located in " &
 ActiveDocument.Path
End Sub
```

Related Topics

Document.Name Property

Document.Path Property

Document.GetColor Method

This method returns the color of the specified document element.

Usage

```
GetColor(colorType as blazeDocumentColor) as Integer
```

Arguments

colorType	Specifies document element.

Return Values

Index of the color in the palette.

Examples

```
doc = Application.ActiveDocument

msg=""
msg = msg & doc.GetColor(blazeDocumentColorBackground) & ", "
msg = msg & doc.GetColor(blazeDocumentColorSelection) & ", "
msg = msg & doc.GetColor(blazeDocumentColorConnection) & ", "
msg = msg & doc.GetColor(blazeDocumentColorBoardOutline) & ", "
msg = msg & doc.GetColor(blazeDocumentColorTestPoint) & ", "
msg = msg & doc.GetColor(blazeDocumentColorThermal) & ", "
msg = msg & doc.GetColor(blazeDocumentColorGuardBand)

MsgBox msg
```

```
curr_bkg_color = doc.GetColor(blazeDocumentColorBackground)

doc.SetColor(blazeDocumentColorBackground, 2)


MsgBox "Press any key"


doc.SetColor(blazeDocumentColorBackground, curr_bkg_color)
```

Document.GetObjects Method

This method returns a collection of database objects of the application.

Usage

```
GetObjects ([type as blazeObjectType = blazeObjectTypeAll], [value as
 String], [selected as Boolean = False]) as Objects
```

Arguments

type	[Optional] Type of application database object to return
value	[Optional] Value or name of the objects to return
selected	[Optional] True to get selected objects only or 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. For more information, see the samples below.

This property generates an exception if the *type* argument is not a valid application database object type.

Name supports the use of Wildcards.

To get all objects of the same *type*, use the corresponding object document property instead of this method. For example, to get all nets in the open design, use Document.Nets Property instead of DocumentGetObjects.

Return Values

The returned object is an Objects collection object. If no objects satisfy the request, the returned collection is empty.

Examples

The following sample code shows different ways to use this method, displaying the number of objects retrieved each way using the Objects.Count Property property. For more information about running this sample code, see Running Code Samples.

```
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 (blazeObjectTypeNet)
  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 (blazeObjectTypeNet, "VCC")
  MsgBox "Ex4: " & objs.Count & " VCC net objects."
```

```
' Ex5: Get all part objects which names begin with U
  Set objs = ActiveDocument.GetObjects (blazeObjectTypeComponent, "U*")
  MsgBox "Ex3: " & objs.Count & " U* part objects."
End Sub
```

Related Topics

Document.Nets Property

Objects.Count Property

Document.GetVisibility Method

This method returns the visibility of the specified design object type.

Usage

```
GetVisibility(objectType as blazeDesignObject) as Boolean
```

Arguments

objectType	Specifies design object type.

Return Values

TRUE if visible, FALSE if hidden.

Examples

```
doc = Application.ActiveDocument

If doc.GetVisibility(blazeDesignObjectPad) = true Then
MsgBox "Pads are visible"
Else
MsgBox "Pads are invisible"
End If

doc.SetVisibility(blazeDesignObjectTrace, false)

MsgBox "Press any key"

doc.SetVisibility(blazeDesignObjectTrace, true)
```

Document.Layers Property

This 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

layerNumber	Number of layer.
layerName	Name of 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(blazeLayerColorPad) & ", "
```

```
            & layer.CopperThickness & ", "
            & layer.GetDielectricThickness(blazeDielectricLayerAbove)
            & ", "
            & layer.GetDielectricConstant(blazeDielectricLayerBelow)
  msg = msg & chr(13)

Next layer

MsgBox msg

layer = doc.Layers(1)
curr_pad_color = layer.GetColor(blazeLayerColorPad)
layer.SetColor(blazeLayerColorPad, 10)

MsgBox "Press any key"

layer.Visible = false

MsgBox "Press any key"

layer.Visible = true

MsgBox "Press any key"
```

```
layer.SetColor(blazeLayerColorPad, curr_pad_color)
```

Wildcards

Arguments such as *Name* support the use of wildcards.

(“U*”), referring to lists of items delimited by comma (“U1, U2, R1”) and ranges specified by two object names and the dash character (“U1 - U10, U12, R1 - R20”). The dash symbol 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 a name with “U*, R*, C1 - C100” but you cannot pass a name with “U*1*” or “C1* - C10*”

```

```

Document.MaxRealValue Property

This property Returns the maximum real value in the system.

Usage

MaxRealValue as double

Arguments

None

Document.MinRealValue Property

This property Returns the minimum real value in the system.

Usage

MinRealValue as double

Arguments

None

Document.Name Property

This property returns the name of the document. For example, if the current design file path is “\My Documents\SailWind Projects\Samples\pwrdemoa.pcb,” this function returns the string “pwrdemoa.pcb”. The Document.Name property is the default property for the Document object.

Usage

```
Name as String
```

Arguments

None

Examples

The following sample code retrieves the name and location of the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  MsgBox "Hi, you are using " & ActiveDocument.Name & " located in " &
 ActiveDocument.Path
End Sub
```

Related Topics

Document.FullName Property

Document.Path Property

Document.Nets Property

This property returns the collection of all nets.

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 associated with that name. If the net name is not specified, this property returns the collection of all nets in Objects.

Examples

The following sample code retrieves the number of nets in the open design using the Objects.Count Property property. For more information about running this sample code, see Running Code Samples.

```
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 and virtual pins connected to Net VCC, assuming it exists in the open design, using the Net.Pins Property and Objects.Count Property properties. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  Set netVCC = ActiveDocument.Nets ("VCC")
  MsgBox "Net " & netVCC.Name & " connects " & netVCC.Pins.Count & "
 in(s)."
End Sub
```

Related Topics

Net.Pins Property

Objects.Count Property

Document.NetClasses Property

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

Usage

```
NetClasses as Objects
```

```
NetClass (name as String) as NetClass
```

Arguments

name	[Optional] Name of an existing net class

Examples

The following sample code retrieves the number of net classes in the open design using the Objects.Count Property property. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  Set classes = ActiveDocument.NetClasses
  MsgBox "There are " & classes.Count & " net class(es) in " &
 ActiveDocument.Name
End Sub
```

Related Topics

Document.GetObjects Method

Objects.Count Property

Document.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Application
```

Arguments

None

Document.Path Property

This property returns the path of the document.

Usage

```
Path as String
```

Arguments

None

Examples

The following sample code retrieves the name and path of the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  MsgBox "Hi, you are using " & ActiveDocument.Name & " located in " &
 ActiveDocument.Path
End Sub
```

Related Topics

Document.FullName Property

Document.Name Property

Document.Pins Property

This property returns a pin or virtual pin object or the collection of all pins and virtual pins.

Usage

```
Pins as Objects
```

```
Pins (name as String) as Pin
```

Arguments

name	[Optional] Name of an existing pin

Return Values

When an existing pin *name* is passed to this property, it returns that pin object. If the pin *name* does not exist, this property returns the collection of all pins in an Objects collection object.

Examples

The following sample code retrieves the number of pins in the open design using the Objects.Count Propertyproperty. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  Set pins = ActiveDocument.Pins
  MsgBox "There are " & pins.Count & " pin(s) in " & ActiveDocument.Name
End Sub
```

Related Topics

Document.GetObjects Method

Objects.Count Property

Document.Router Property

This property returns the Router object, which is a special object that provides access to routing-specific functions.

Usage

```
Router as Router
```

Arguments

None

Document.SaveAs Method

This method saves the document.

Usage

```
SaveAs
```

Arguments

None

Return Values

This method generates an exception if an application security failure occurs during processing.

Related Topics

Document.Save Method

Document.Save Event

Document.Saved Property

Document.Saved Property

This property determines whether the document is saved or not. The Document.Saved property is most commonly used before opening a new design using the Application.OpenDocument method to ensure that the message, “Save old file before reloading?” does not appear.

Usage

```
Saved as Boolean
```

Arguments

None

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. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  ActiveDocument.Saved = True
  OpenDocument (DefaultFilePath & "\PWRDEMOA.PCB")
  MsgBox ActiveDocument.FullName & " has just been opened."
End Sub
```

Related Topics

Application.DefaultFilePath Property

Application.OpenDocument Method

Document.Save Method

Document.SaveAs Method

Document.Save Event

Document.Save Method

This method saves the document.

Usage

Save

Arguments

None

Return Values

This method generates an exception if an application security failure occurs during processing.

Examples

The following sample code saves the open design if changes have been made. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  If ActiveDocument.Saved = False Then ActiveDocument.Save
End Sub
```

Related Topics

Document.SaveAs Method

Document.Saved Property

Document.Save Event

Document.Save Event

This event occurs after the document is saved.

Usage

```
Document.Save
```

Arguments

None

Return Values

None

Related Topics

Document.Save Method

Document.SaveAs Method

Document.Saved Property

Document.SecurityLimit Event

This event occurs when the database security limits are reached.

Usage

```
Document.SecurityLimit (Reached as Boolean)
```

Arguments

reached	Indicates whether security limit is reached or not

Return Values

None

Document.SelectionChange Event

This event occurs when the current selection changes.

Usage

Document.SelectionChange

Arguments

None

Related Topics

Document.SelectObjects Method

Objects.Select Method

Document.SelectObjects Method

This method selects or clears application database objects.

Usage

```
SelectObjects ([type as blazeObjectType = blazeObjectTypeAll], [value as
 String], [select as Boolean = True])
```

Arguments

type	[Optional] Type of application database object to select/clear
value	[Optional] Value of the object(s) to select/clear
name	[Optional] Name of the object(s) to select/clear
select	[Optional] True to select or False to clear

All arguments for this method are optional, which means that the method can be called with no argument at all, or with any combination of arguments. See the samples below for more information.

Name supports the use of:

Table 14. Document.SelectObjects Object Specifiers

Supported Item	Example
Lists of items delimited by comma	U1, U2, R1
Ranges specified by two object names and the dash character	U1 - U10, U12, R1 - R20 The dash symbol must be surrounded by spaces since the dash is a legal symbol in an object name.
Wildcards	U* Only one wildcard per name is allowed and you cannot specify wildcards in a range.

You can pass *name* with **U***, **R***, or **C1 - C100** but you cannot pass name with **U*1*** or **C1* - C10***.

This property generates an exception if *type* is not a valid application 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. For more information about running this sample code, see Running Code Samples.

```
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 (blazeObjectTypeNet)
  Set objs = ActiveDocument.GetObjects (,,True)
  MsgBox "Ex3: " & objs.Count & " selected objects
  (all nets)."
```

```
  ' Ex4: Unselect net VCC
  ActiveDocument.SelectObjects (blazeObjectTypeNet,
  "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)
  &#9;ActiveDocument.SelectObjects
  (blazeObjectTypeComponent, "U*")
  Set objs = ActiveDocument.GetObjects (,,True)
  MsgBox "Ex5: " & objs.Count & " selected U* part
  objects."
```

```
End Sub
```

Related Topics

Document.GetObjects Method

Document.SelectionChange Event

Objects.Count Property

Document.SetColor Method

This method sets color for the specified document element.

Usage

```
SetColor(colorType as blazeDocumentColor, colorIndex as Integer)
```

Arguments

colorType	Specifies document element.
colorIndex	Index of the color in the palette. Must be between 0 and 31.

Examples

```
doc = Application.ActiveDocument

msg=""
msg = msg & doc.GetColor(blazeDocumentColorBackground) & ", "
msg = msg & doc.GetColor(blazeDocumentColorSelection) & ", "
msg = msg & doc.GetColor(blazeDocumentColorConnection) & ", "
msg = msg & doc.GetColor(blazeDocumentColorBoardOutline) & ", "
msg = msg & doc.GetColor(blazeDocumentColorTestPoint) & ", "
msg = msg & doc.GetColor(blazeDocumentColorThermal) & ", "
msg = msg & doc.GetColor(blazeDocumentColorGuardBand)

MsgBox msg

curr_bkg_color = doc.GetColor(blazeDocumentColorBackground)
```

```
doc.SetColor(blazeDocumentColorBackground, 2)

MsgBox "Press any key"

doc.SetColor(blazeDocumentColorBackground, curr_bkg_color)
```

Document.SetVisibility Method

This method sets visibility for the specified design object type.

Usage

```
SetVisibility(objectType as blazeDesignObject, objectVisibility as
 Boolean)
```

Arguments

objectType	Specifies design object type.
objectVisibility	TRUE if visible, FALSE if hidden.

Examples

```
doc = Application.ActiveDocument

If doc.GetVisibility(blazeDesignObjectPad) = true Then
MsgBox "Pads are visible"
Else
MsgBox "Pads are invisible"
End If

doc.SetVisibility(blazeDesignObjectTrace, false)

MsgBox "Press any key"

doc.SetVisibility(blazeDesignObjectTrace, true)
```

Document.Unit Property

This property returns the system unit used in the document.

Usage

```
Unit as blazeUnit on page 22
```

Arguments

None

Document.Unroute Method

This method unroutes selected objects.

Usage

```
Unroute
```

Arguments

None

Document.UnrouteCount Property

This property returns the number of unroutes in the design.

Usage

```
UnrouteCount as Long
```

Arguments

None

Examples

The following sample displays a message box showing the number of unroutes.

```
Sub Main
```

```
  MsgBox "Number of Unroutes: " & ActiveDocument.UnrouteCount
```

```
End Sub
```

Document.Vias Property

This property returns the collection of all vias.

Usage

```
Vias as Objects
```

```
Vias (name as String) as Via
```

Arguments

name	[Optional] Name of an existing via

Return Values

When an existing via *name* is passed to this property, it returns that Via object. If the via *name* does not exist, this property returns the collection of all vias in an Objects collection object.

Examples

The following sample code retrieves the number of vias in the open design using the Objects.Count Property property. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  Set vias = ActiveDocument.Vias
  MsgBox "There are " & vias.Count & " via(s) in " & ActiveDocument.Name
End Sub
```

Related Topics

Document.GetObjects Method

Objects.Count Property

The Drawing Object

The Drawing object represents a physical drawing object or a copper in the open design.

The following list identifies the Drawing Object 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

This property returns the Application object.

Usage

Application as Application

Arguments

None

Description

This is a Microsoft-required property.

Examples

Not required.

Drawing.DrawingType Property

This property returns the type of the open drawing.

Usage

DrawingType as blazeDrawingType

Arguments

None

Description

None

Examples

The following sample function displays the name and type of the drawing.

```
Sub Main

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

For Each drw In selected

Select Case drw.DrawingType

Case blazeDrw2Dline

Msgbox "Type: 2D Line"

Case blazeDrwBoard

Msgbox "Type: Board Outline"

Case blazeDrwCopper

Msgbox "Type: Copper"

Case blazeDrwCopperPour

Msgbox "Type: CopperPour"
```

```
Case blazeDrwCopperHatch

Msgbox "Type: CopperHatch"

Case blazeDrwCopperThermal

Msgbox "Type: CopperThermal"

Case blazeDrwKeepout

Msgbox "Type: Keepout"

End Select

Next drw

End Sub
```

Drawing.Geometry Property

This 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.

```
Sub Main

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

For Each drw In selected

I = 0

Dim geom as Object

For Each geom in drw.Geometry

If geom.ObjectType = blazeObjectTypePolyline Then I = I + 1

Next geom

MsgBox "Drawing contains " & I & " polylines"

Exit For

Next drw

End Sub
```

Drawing.Name Property

This default 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.

```
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

This 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.

```
Sub Main

Set selected = ActiveDocument.GetObjects(blazeObjectTypeDrawing,,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

This property returns the type of object.

Usage

ObjectType as blazeObjectType on page 22

Arguments

None

Description

None

Examples

The following example shows the number of selected drawings.

```
Sub Main

Set sel = ActiveDocument.GetObjects(,,TRUE)

n = 0

For Each obj In sel

If obj.ObjectType = blazeObjectTypeDrawing Then n = n + 1

Next obj

MsgBox n & " drawing(s) selected"

End Sub
```

Drawing.Parent Property

This property returns the parent of the object.

Usage

Parent as Document

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 blazeUnit on page 22, *Origin* as blazeOriginType) as Double

Arguments

unit	[Optional] Unit in which the result is to be expressed. This optional argument is blazeUnitCurrent by default.
origin	[Optional] Type of reference point from which the result is counted. The default value is blazeOriginTypeDesign.

Description

None

Examples

The following sample function displays selected drawing positions.

```
Sub Main

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

For Each drw In selected

Msgbox "Position: (" & drw.PositionX & ", " & drw.PositionY & ") "

Next drw

End Sub
```

Drawing.PositionY Property

This property returns the Y coordinate of the origin of the drawing.

Usage

PositionY(*Unit* as blazeUnit on page 22, *Origin* as blazeOriginType) as Double

Arguments

unit	[Optional] Unit in which the result is to be expressed. This optional argument is blazeUnitCurrent by default.
origin	[Optional] Type of reference point from which the result is counted. The default value is blazeOriginTypeDesign.

Description

None

Examples

The following sample function displays the positions of the selected drawing objects.

```
Sub Main

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

For Each drw In selected

Msgbox "Position: (" & drw.PositionX & ", " & drw.Position.Y & ") "

Next drw

End Sub
```

Drawing.Selected Property

This 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.

```
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

This 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

name	Text object name.

Description

None

Examples

The following sample function returns the number of texts associated with the selected drawing.

```
Sub Main

Set selected = ActiveDocument.GetObjects(blazeObjectTypeDrawing,,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 following list identifies the Error Object properties:

Error.ActualValue Property

This property returns the actual value of the Error Object.

Usage

Conflicts as Objects

Conflicts(conflictNumber as Integer) as ErrorConflict

Arguments

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

Examples

The following sample code displays information about all of the errors' actual value in SailWind Router. See "Running Code Samples" for more information on running this sample.

```
' Display information about all errors' actual value in Router

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 = blazeErrorValueTypeMeasure) Then
   info = info & theErr.ActualValue(blazeUnitCurrent)
   info = info & " [current unit]; "
```

```
   info = info & theErr.ActualValue(blazeUnitDatabase)

   info = info & " [database unit]; "

   info = info & theErr.ActualValue(blazeUnitMils)

   info = info & " [mils]; "

   info = info & theErr.ActualValue(blazeUnitInch)

   info = info & " [inch]; "

   info = info & theErr.ActualValue(blazeUnitMetric)

   info = info & " [millimeter]"

  End If

 End If

 MsgBox info

 ' Next error

Next
```

Error.Application Property

This property returns the Application object.

Usage

Application as Application on page 46

Arguments

None

Error.Conflicts Property

This property returns the collection of all error conflicts in the Error Object.

Usage

Conflicts as Objects

Conflicts(conflictNumber as Integer) as ErrorConflict

Arguments

conflictNumber	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

This property returns the description of the Error Object.

Usage

Description as String

Arguments

None

Error.ErrorClass Property

This property returns the error class of the Error Object.

Usage

ErrorClass as blazeErrorClass

Arguments

None

Error.ErrorType Property

This property returns the error type of the Error Object.

Usage

ErrorType as blazeErrorType

Arguments

None

Error.HasActualValue Property

This property returns if the actual value of the Error Object exists.

Usage

HasActualValue as Boolean

Arguments

None

Error.HasRequiredValue Property

This 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

This property returns if the error is a connectivity error.

Usage

IsConnectivityError as Boolean

Arguments

None

Error.IsDiffpairError Property

This property returns if the error is a Differential Pair error.

Usage

IsDiffpairError as Boolean

Arguments

None

Error.IsFabricationError Property

This property returns if the error is a fabrication error.

Usage

IsfabricationError as Boolean

Arguments

None

Error.IsIgnoredFlag Property

This property returns if the error is flagged as ignored.

Usage

IsIgnoredFlag as Boolean

Arguments

None

Error.IsInvisibleFlag Property

This property returns if the error is flagged as invisible.

Usage

IsInvisibleFlag as Boolean

Arguments

None

Error.IsLayoutError Property

This property returns if the error is a SailWind Layout error.

Usage

IsLayoutError as Boolean

Arguments

None

Error.IsLengthError Property

This property returns if the error is a length error.

Usage

IsLengthError as Boolean

Arguments

None

Error.IsMaxViaCountError Property

This property returns if the error is a maximum via count error.

Usage

IsMaxViaCountError as Boolean

Arguments

None

Error.IsTestabilityError Property

This property returns if the error is a testability error.

Usage

IsTestabilityError as Boolean

Arguments

None

Error.LayerNumber Property

This property returns the layer number of this Error Object.

Usage

LayerNumber as integer

Arguments

None

Error.Name Property

This property returns the name of this error.

Usage

Name As String

Arguments

None

Error.ObjectType Property

This property returns the type of this object - ppcbObjectTypeError.

Usage

ObjectType as blazeObjectType

Arguments

None

Error.Parent Property

This property returns the parent of the object.

Usage

Parent As Document on page 112

Arguments

None

Error.PositionX Property

This property returns the X-coordinate of the Error Object.

Usage

PositionX as Double

PositionX(unit as blazeUnit) as Double

Arguments

unit	[Optional Argument] Unit in which the X-coordinate is returned.

Error.PositionY Property

This property returns the Y-coordinate of the Error Object.

Usage

PositionY as Double

PositionY(unit as blazeUnit) as Double

Arguments

unit	[Optional Argument] Unit in which the Y-coordinate is returned.

Error.RequiredValueMax Property

Returns the required maximum value of the Error Object. If the value equals the maximum real value, it indicates infinity.

Usage

RequiredValueMax as Double

RequiredValueMax (unit as blazeUnit) as Double

Arguments

unit	[Optional Argument] Unit in which the required maximum value is returned.

Error.RequiredValueMin Property

Returns the required minimum value of the Error Object. If the value equals the minimum real value, it indicates minus infinity.

Usage

RequiredValueMin as Double

RequiredValueMin (unit as blazeUnit) as Double

Arguments

unit	[Optional Argument] Unit in which the required minimum value is returned.

Error.ValueType Property

This property returns the type of the error values (the actual value and the require values).

Usage

ValueType as BlzazeErrorValueType

Arguments

None

The ErrorConflict Object

The Error object represents an error conflict in the Error Object.

The following list identifies the ErrorConflict properties:

ErrorConflict.Application Property

This property returns the Application object.

Usage

Application as Application on page 46

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 blazeObjectType

Arguments

None

ErrorConflict.Name Property

This 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 blazeObjectType

Arguments

None

ErrorConflict.Parent Property

This property returns the parent of the object.

Usage

Parent as Document on page 112

Arguments

None

The Layer Object

This object represents a design layer.

The following list identifies the Layer Object properties and methods:

Layer.Application Property

This property returns the application object.

Usage

```
Application as Application
```

Arguments

None

Layer.CopperThickness Property

This property returns the layer copper thickness.

Usage

```
CopperThickness (unit as blazeUnit on page 22) as Double
```

Arguments

unit	[Optional] Unit in which the value is returned. This argument is blazeUnitCurrent by default.

Layer.Enabled Property

This property returns the layer enabled property.

Usage

```
Enabled as Boolean
```

Arguments

None

Layer.GetColor Method

This method returns color of the specified layer element.

Usage

```
GetColor(colorType as blazeLayerColor) as Integer
```

Arguments

colorType	Specifies layer element.

Return Values

Index of the color in the palette.

Layer.GetDielectricConstant Method

This method returns the layer dielectric constant.

Usage

```
GetDielectricConstant(dielectricLayer as blazeDielectricLayer) as Double
```

Arguments

dielectricLayer	Specifies dielectric layer relevant to this electrical layer.

Return Values

The dielectric constant.

Layer.GetDielectricThickness Method

This method gets the layer dielectric thickness.

Usage

```
GetDielectricThickness (
dielectricLayer as blazeDielectricLayer,
unit as blazeUnit on page 22) as Double
```

Arguments

dielectricLayer	Specifies dielectric layer relevant to this electrical layer.
unit	[Optional] Unit in which value is returned. This optional argument is blazeUnitCurrent by default.

Return Values

The dielectric thickness.

Layer.GetDielectricType Method

This method returns the layer dielectric type.

Usage

```
GetDielectricType(dielectricLayer as blazeDielectricLayer)
 as blazeDielectricType
```

Arguments

dielectricLayer	Specifies dielectric layer relevant to this electrical layer.

Return Values

The dielectric type.

Layer.Name Property

This property returns the name of this layer.

Usage

```
Name as String
```

Arguments

None

Layer.Number Property

This property returns the layer number.

Usage

```
Number as Integer
```

Arguments

None

Layer.ObjectType Property

This property returns the type of the object - blazeObjectTypeLayer.

Usage

```
ObjectType as blazeObjectType on page 22
```

Arguments

None

Layer.PlaneType Property

This property returns the layer plane type.

Usage

```
PlaneType as blazePlaneType
```

Arguments

None

Layer.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

Layer.Routable Property

This property sets or returns the layer routable flag.

Usage

```
Routable as Boolean
```

Arguments

None

Layer.RoutingAngle Property

This property sets or returns the layer routing angle.

Usage

```
RoutingAngle as Integer
```

Arguments

None

Layer.RoutingCost Property

This property sets or returns the layer routing cost.

Usage

```
RoutingCost as Integer
```

Arguments

None

Layer.RoutingDirection Property

This property sets or returns the layer routing direction.

Usage

```
RoutingDirection as blazeRoutingDirection
```

Arguments

None

Layer.SetColor Method

This method sets color for the specified layer element.

Usage

```
SetColor(colorType as blazeLayerColor, colorIndex as Integer)
```

Arguments

colorType	Specifies layer element.
colorIndex	Index of the color in the palette. Must be between 0 and 31.

Layer.Type Property

This property returns the layer type. This property uses BlazeLayerType enumeration which was already defined in previous software versions (see documentation).

Usage

```
Type as BlazeLayerType
```

Arguments

None

Layer.Visible Property

This property sets or returns the layer visibility.

Usage

```
Visible as Boolean
```

Arguments

None

The NetClass Object

The NetClass object represents a physical net class that exists in the PCB design currently open in the application.

The NetClass object has the following properties:

- NetClass.Application Property
- NetClass.Name Property
- NetClass.Nets Property
- NetClass.ObjectType Property
- NetClass.Parent Property
- NetClass.Selected Property
- NetClass.Vias Property

NetClass.Application Property

This property returns the Application object of the application. This property identifies the object as an Automation object of the application. 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 various sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

Usage

```
Application as Application
```

Arguments

None

NetClass.Name Property

This property returns the name of the net class. For example, this property returns the string “POWER” for the net GND if this net belongs to class “POWER”. The NetClass.Name property is the default property for the Net Class object.

Usage

```
Name as String
```

Arguments

None

NetClass.Nets Property

This property returns the collection of all pins connected to the net.

Usage

```
Nets as Objects
```

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

Arguments

name	Name of an existing net

Return Values

When an existing net *name* is passed to this property, it returns that net object. If the net name does not exist, this property returns the collection of all nets that belong to this NetClass.

Examples

The following sample code retrieves the number of nets that belong to net class POWER, assuming it exists in the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  MsgBox "POWER includes " & ActiveDocument.NetClasses("POWER").Nets.Count
 & " pins."
End Sub
```

NetClass.ObjectType Property

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

Usage

```
ObjectType as blazeObjectType on page 22
```

Arguments

None

Return Values

This property always returns blazeObjectTypeNetClass.

NetClass.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

NetClass.Selected Property

This property sets or returns whether the net class is selected. You can also select an application database object using the Document.SelectObjects and Objects.Select methods.

Usage

```
Selected as Boolean
```

Arguments

None

Examples

The following sample code selects net class POWER only, assuming it exists in the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  ActiveDocument.SelectObjects(,,False)
  ActiveDocument.NetClasses("POWER").Selected = True
End Sub
```

Related Topics

Document.SelectObjects Method

Document.SelectionChange Event

Objects.Select Method

NetClass.Vias Property

This property returns the collection of all vias connected to the netclass.

Usage

```
Vias as Objects
```

```
Vias (name as String) as Via
```

Arguments

name	Name of an existing via

Return Values

When an existing via *name* is passed to this property, it returns that Via object. If the via *name* does not exist, this property returns the collection of all vias connected to the net in an Objects collection object.

Examples

The following sample code retrieves the number of vias connected to netclass VCC, assuming it exists in the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  MsgBox "VCC connects " & ActiveDocument.Nets("VCC").Vias.Count & " vias."
End Sub
```

The Net Object

The Net object represents a physical net, which exists in the PCB design currently open in the application.

The following list identifies the Net Object properties:

Net.Application Property

This property returns the Application object of the application. This property identifies the object as an Automation object of the application. 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 various sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

Usage

```
Application as Application
```

Arguments

None

Net.AssociatedNet Property

This 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.Name Property

This 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

Net.NetClass Property

This property returns the class of the net.

Usage

```
NetClass as NetClass
```

Arguments

None

Return Values

This property returns Nothing if the net does not belong to any class.

Net.ObjectType Property

This 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

```
ObjectTypeas blazeObjectType on page 22
```

Arguments

None

Return Values

This property always returns blazeObjectTypeNet.

Net.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

Net.Pins Property

This property returns the collection of all pins and virtual pins connected to the net.

Usage

```
Pins as Objects
```

```
Pins (name as String) as Pin
```

Arguments

name	Name of an existing pin

Return Values

When an existing pin *name* is passed to this property, it returns that Pin object. If the pin *name* does not exist, this property returns the collection of all pins connected to the net in an Objects collection object.

Examples

The following sample code retrieves the number of pins connected to net VCC, assuming it exists in the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  MsgBox "VCC connects " & ActiveDocument.Nets("VCC").Pins.Count & " pins."
End Sub
```

Net.Selected Property

This property sets or determines whether the net is selected or not. You can also select an application database object using the Document.SelectObjects and Objects.Select methods.

Usage

```
Selected as Boolean
```

Arguments

None

Examples

The following sample code selects net VCC only, assuming it exists in the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  ActiveDocument.SelectObjects(,,False)
  ActiveDocument.Nets("VCC").Selected = True
End Sub
```

Related Topics

Document.SelectionChange Event

Document.SelectObjects Method

Objects.Select Method

Net.Vias Property

This property returns the collection of all vias connected to the net.

Usage

```
Vias as Objects
```

```
Vias (name as String) as Via
```

Arguments

name	Name of an existing via

Return Values

When an existing via *name* is passed to this property, it returns that Via object. If the via *name* does not exist, this property returns the collection of all vias connected to the net in an Objects collection object.

Examples

The following sample code retrieves the number of vias connected to net VCC, assuming it exists in the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  MsgBox "VCC connects " & ActiveDocument.Nets("VCC").Vias.Count & " vias."
End Sub
```

The Objects Collection Object

The Objects collection object is a collection of database objects, such as Component objects, Jumper objects, Net objects, NetClass objects, Pin objects, and Via objects, which exist in the open design.

This object is usually retrieved using the Document.GetObjects method or using database object specific properties such as Document.Components, Document.Nets, Document.NetClasses, Document.Pins, and Document.Vias.

The following list identifies the Objects Collection Object properties and methods:

See also: Document.Components, Document.GetObjects, Document.NetClasses, Document.Nets, Document.Pins, Document.Vias

Objects.Add Method

This method adds an object to the collection. If the object argument is not a database object of the application, this method generates an exception.

Usage

```
Add (object as object)
```

Arguments

object	Object to add to the collection. It must be a database object of the application, such as Component, Net, NetClass, Pin, or Via.

Return Values

None

Examples

The following sample code creates a collection of all U* components in the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  Set UComps = ActiveDocument.GetObjects (0)
  For Each nextComp In ActiveDocument.Components
  If Left (nextComp.Name, 1)="U" Then UComps.Add (nextComp)
  Next nextComp
  MsgBox UComps.Count
End Sub
```

Related Topics

Objects.Remove Method

Objects.Application Property

This property returns the Application object of the application. This property identifies the object as an Automation object of the application. 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 various sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

Usage

```
Application as Application
```

Arguments

None

Objects.Count Property

This property returns the number of objects in the collection.

Usage

```
Count as Long
```

Arguments

None

Examples

The following sample code retrieves the number of selected items in the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  Set selectedObjects = ActiveDocument.GetObjects (,,True)
  MsgBox "There are " & selectedObjects.Count & " selected object(s)."
End Sub
```

Objects.Item Property

This property returns the object given its index or its name. This is the default member of the Objects collection object.

Usage

```
Item (index as Long) as Object
```

```
Item (name as String) as Object
```

Arguments

index	Index (in the collection) of the object to retrieve
name	Name of the object to retrieve

Return Values

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 ways to repeat through all database objects of the application in the open design. The second method is preferred because it is cleaner and faster. For more information about running this sample code, see Running Code Samples.

```
' 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

This property returns the type of an object given its index.

Usage

```
ItemType (index as Long) as blazeObjectType on page 22
```

```
ItemType (name as String) as blazeObjectType on page 22
```

Arguments

index	Required Index of the object in the collection to query
name	Name of the object to retrieve

This property generates an exception if the *index* argument is not valid.

Related Topics

Objects.Item Property

Objects.Merge Method

This method merges two object collections. This method adds all objects in the objects collection object to the current Object Collection.

Usage

```
Merge (objects as Objects)
```

Arguments

objects	The collection with objects to merge with the current object collection

Return Values

None

Objects.Next Property

This property returns the index of the next object of the given type after the given index.

Usage

```
Next (index as long, type as blazeObjectType on page 22) as Long
```

Arguments

index	Index of the object in the collection to query
type	Type of the object to query

This property generates an exception if the *index* argument is not valid. If *index* = zero (0), this property 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.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

Objects.Remove Method

This method removes an object from the collection.

Usage

```
Remove (index as Long)
```

```
Remove (name as String)
```

Arguments

index	Index of the object to remove
name	Name of the object to remove

This property generates an exception if the *index* argument is not valid.

Return Values

None

Related Topics

Objects.Add Method

Objects.Reset Method

This method resets the object collection.

Usage

```
Reset
```

Arguments

None

Return Values

None

Objects.Select Method

This method selects or deselects all objects in the collection.

Usage

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

Arguments

bSelect	[Optional] True to select or False to clear

Return Values

None

Examples

The following sample code selects all vias in the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  ActiveDocument.Vias.Select
End Sub
```

Related Topics

Document.SelectionChange Event

Objects.Sort Method

This method sorts objects in the collection by object name.

Usage

```
Sort
```

Arguments

None

Return Values

None

Examples

The following sample code creates a collection of pins, sorts them by name, and displays them in a list box. For more information about running this sample code, see Running Code Samples.

```
Dim ListPins$(10000)
Sub Main
  Set Objs = ActiveDocument.Pins
  Objs.Sort
  index = 0
  For Each nextPin In Objs
  ListPins$(index) = nextPin.Name
  index = index + 1
  Next nextPin
  ' This piece of code is automatically generated by the PADS Layout Basic
 Dialog Editor.
  Begin Dialog UserDialog 180,238,"All Pins Sorted " ' %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 Pad Object

This object represents a pad in padstack definition.

The following list identifies the Pad Object properties:

Pad.Application Property

This property returns the application object.

Usage

```
Application as Application
```

Arguments

None

Pad.CornerRadius Property

This property returns the pad's corner radius. This property should be used for the following pad shapes: blazePadShapeSquare, blazePadShapeRectangularFinger.

Usage

```
CornerRadius (unit as blazeUnit on page 22) as Double
```

Arguments

unit	[Optional] Unit in which the value is returned. This optional argument is blazeUnitCurrent by default.

Pad.CornerType Property

This property returns the pad's corner type. This property should be used for the following pad shapes: blazePadShapeSquare, blazePadShapeRectangularFinger.

Usage

```
CornerType as blazePadCornerType
```

Arguments

None

Pad.Diameter Property

This property returns the pad's diameter. This property should be used for the following pad shapes: blazePadShapeRound, blazePadShapeAnnular, blazePadShapeOdd.

Usage

```
Diameter (unit as blazeUnit on page 22) as Double
```

Arguments

unit	[Optional] Unit in which the value is returned. This optional argument is blazeUnitCurrent by default.

Pad.InnerDiameter Property

This property returns the pad's inner diameter. This property should be used for the following pad shapes: blazePadShapeAnnular.

Usage

```
InnerDiameter (unit as blazeUnit on page 22) as Double
```

Arguments

unit	[Optional] Unit in which the value is returned. This argument is blazeUnitCurrent by default.

Pad.Length Property

This property returns the pad's length. This property should be used for the following pad shapes: blazePadShapeOvalFinger, blazePadShapeRectangularFinger.

Usage

```
Length (unit as blazeUnit on page 22) as Double
```

Arguments

unit	[Optional] Unit in which the value is returned. This argument is blazeUnitCurrent by default.

Pad.Name Property

This property returns the name of this pad.

Usage

```
Name as String
```

Arguments

None

Pad.ObjectType Property

This property returns the type of the object - blazeObjectTypePad.

Usage

```
ObjectType as blazeObjectType on page 22
```

Arguments

None

Pad.Offset Property

This property returns the pad's offset. This property should be used for the following pad shapes: blazePadShapeOvalFinger, blazePadShapeRectangularFinger.

Usage

```
Offset (unit as blazeUnit on page 22) as Double
```

Arguments

unit	[Optional] Unit in which the value is returned. This argument is blazeUnitCurrent by default.

Pad.Orientation Property

This property returns the pad's orientation. This property should be used for the following pad shapes: blazePadShapeOvalFinger, blazePadShapeRectangularFinger.

Usage

```
Orientation as Double
```

Arguments

None

Pad.PadStackLayer Property

This property returns the PadStackLayer Object to which this pad belongs to.

Usage

```
PadStackLayer as blazePadStackLayerType
```

Arguments

None

Pad.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

Pad.Shape Property

This property returns the pad shape.

Usage

```
Shape as blazePadShape
```

Arguments

None

Pad.Width Property

This property returns the pad's width. This property should be used for the following pad shapes: blazePadShapeOvalFinger, blazePadShapeRectangularFinger, blazePadShapeSquare.

Usage

```
Width (unit as blazeUnit on page 22) as Double
```

Arguments

unit	[Optional] Unit in which the value is returned. This optional argument is blazeUnitCurrent by default.

The PadStackLayer Object

This object represents a layer in padstack definition.

The following list identifies the PadStackLayer 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 Method Property

PadStackLayer.AntiPad Property

This 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

This property returns the application object.

Usage

```
Application as Application
```

Arguments

None

PadStackLayer.Name Property

This property returns the name of this padstack layer.

Usage

```
Name as String
```

Arguments

None

PadStackLayer.Number Property

This property returns the number of this padstack layer. Values -2, -1, 0 correspond to BlazePadStackLayerType.

Usage

```
Number as Integer
```

Arguments

None

PadStackLayer.ObjectType Property

This property returns the type of the object - blazeObjectTypePadStackLayer.

Usage

```
ObjectType as blazeObjectType on page 22
```

Arguments

None

PadStackLayer.Pad Property

This property returns the Pad Object assigned to this padstack layer.

Usage

```
Pad as Pad
```

Arguments

None

PadStackLayer.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

PadStackLayer.Pin Property

This 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

This 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 Method Property

This 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 Pin Object

The Pin object represents a physical component pin, which exists in the PCB design currently open in the application.

The following list identifies the Pin Object properties:

Pin.Application Property

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

Usage

```
Application as Application
```

Arguments

None

Pin.Component Property

This property returns the component that owns the pin.

Usage

```
Component as Component
```

Arguments

None

Examples

The following sample code retrieves the component to which pin U1.1 belongs, assuming U1.1 exists in the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  MsgBox "Pin U1.1 belongs to component " &
 ActiveDocument.Pins("U1.1").Component.Name
End Sub
```

Pin.Name Property

This property returns the name of the pin, for example, this property returns the string “U1.1” for pin U1.1. The Pin.Name property is the default property for the Pin object.

Usage

```
Name as String
```

Arguments

None

Pin.Net Property

This property returns the net connected to the pin.

Usage

```
Net as Net
```

Arguments

None

Return Values

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. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  MsgBox "Pin U1.1 is connected to net " &
 ActiveDocument.Pins("U1.1").Net.Name
End Sub
```

Pin.ObjectType Property

This property returns the type of the object. All database objects of the application's 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 on page 22
```

Arguments

None

Return Values

This property always returns blazeObjectTypePin.

Pin.PadStackLayers Property

This property returns the collection of all padstack layers for this pin.

Usage

```
PadStackLayers as Objects
```

```
PadStackLayers(layerName as String) as blazePadStackLayerType
```

Arguments

layerName	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

This property returns the parent of the object.

Usage

Parent as Document

Arguments

None

Pin.PositionX Property

This property returns the X-coordinate of the pin.

Usage

```
PositionX([unit as blazeUnit on page 22]) as Double
```

Arguments

unit	[Optional] Unit in which the X-coordinate is returned

Examples

The following sample code retrieves the location of pin U1.1, assuming U1.1 exists in the open design, in current design units. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  Set pinU1_1 = ActiveDocument.Pins("U1.1")
  MsgBox "U1.1 position is = (" & pinU1_1.PositionX & ", " &
 pinU1_1.PositionY & ")"
End Sub
```

Related Topics

Pin.PositionY Property

Pin.PositionY Property

This property returns the y-coordinate of the pin.

Usage

```
PositionY([unit as blazeUnit on page 22]) as Double
```

Arguments

unit	[Optional] Unit in which the y-coordinate is returned

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. For more information about running this sample code, see Running Code Samples.

```
Sub Main

  Set pinU1_1 = ActiveDocument.Pins("U1.1")

  MsgBox "U1.1 position is = (" & pinU1_1.PositionX & ", " &
pinU1_1.PositionY & ")"

End Sub
```

Related Topics

Pin.PositionX Property

Pin.Selected Property

This property sets or determines whether the pin is selected or not. You can also select a database object of the application using the Document.SelectObjects and Objects.Select methods.

Usage

```
Selected as Boolean
```

Arguments

None

Examples

The following sample code selects pin U1.1 only, assuming U1.1 exists in the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  ActiveDocument.SelectObjects(,,False)
  ActiveDocument.Pins("U1.1").Selected = True
End Sub
```

Related Topics

Document.SelectionChange Event

Document.SelectObjects Method

Objects.Select Method

Pin.TestPoint Property

This property determines whether the pin is a test point or not.

Usage

```
TestPoint as blazeTestPoint on page 22
```

Arguments

None

The Polyline Object

The Polyline object represents a physical polyline object in the open design.

The following list identifies the Polyline 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

This property returns the Application object.

Usage

Application as Application

Arguments

None

Description

This is a Microsoft-required property.

Polyline.CenterX Property

This property returns a center x-coordinate for an arc in the polyline.

Usage

CenterX (*Corner* as Long, *Unit* as blazeUnit on page 22, *Origin* as blazeOriginType) as Double

Arguments

corner	Starting corner for the arc.
unit	[Optional] Unit in which the result is to be represented. This optional argument is blazeUnitCurrent by default.
Origin	[Optional] Type of reference point from which the result is counted. The default value is blazeOriginTypeDesign.

Description

None

Examples

The following sample code displays center coordinates of the selected polyline.

```
Sub Main
Set selected = ActiveDocument.GetObjects(blazeObjectTypeDrawing,,True)
For Each drw In selected
Dim geom As Object
For Each geom In drw.Geometry
If geom.ObjectType = blazeObjectTypePolyline 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

This property returns a center y-coordinate for an arc in the polyline.

Usage

CenterY (*Corner* as Long, *Unit* as blazeUnit on page 22, *Origin* as blazeOriginType) as Double

Arguments

corner	Starting corner for the arc.
unit	[Optional] Unit in which the result is to be represented. This optional argument is blazeUnitCurrent by default.
origin	[Optional] Type of reference point from which the result is counted. The default value is blazeOriginTypeDesign.

Description

None

Examples

The following sample code displays center coordinates of the selected polyline.

```
Sub Main

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

For Each drw In selected

Dim geom As Object

For Each geom In drw.Geometry

If geom.ObjectType = blazeObjectTypePolyline 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

This 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.

```
Sub Main

Set selected = ActiveDocument.GetObjects(blazeObjectTypeDrawing,,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

This 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.

```
Sub Main

Set selected = ActiveDocument.GetObjects(blazeObjectTypeDrawing,,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

This property returns the line style of the polyline.

Usage

LineStyle as blazeLineStyle

Arguments

None

Description

None

Examples

None

Polyline.LineWidth Property

This property returns the width of the polyline.

Usage

LineWidth (*Unit* as blazeUnit on page 22) as Double

Arguments

unit	[Optional] Unit in which the result is to be represented. This optional argument is blazeUnitCurrent by default.

Description

None

Examples

The following sample code shows the line width of the selected polyline.

```
Sub Main

Set selected = ActiveDocument.GetObjects(blazeObjectTypeDrawing,,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

This property returns the type of object.

Usage

ObjectType as blazeObjectType on page 22

Arguments

None

Description

None

Examples

The following sample code tests the ObjectType property.

```
Sub Main

For Each drw In ActiveDocument.Drawings

For Each geom In drw.Geometry

t = geom.ObjectType

If t <> pcbObjectTypePolyline And t <> blazeObjectTypeCircle Then

MsgBox "Test failed"

End If

Next geom

Next drw

End Sub
```

Polyline.OutlineType Property

This property returns the outline type of the polyline.

Usage

OutlineType as blazeOutlineType

Arguments

None

Description

None

Examples

The following sample code shows the outline type of the selected polyline.

```
Sub Main
Set selected = ActiveDocument.GetObjects(blazeObjectTypeDrawing,,TRUE)
For Each drw In selected
Dim geom as Object
For Each geom In drw.Geometry
Select Case geom.OutlineType
Case blazeOutLineTypeCenter
s="Center line"
Case blazeOutLineTypeOuter
s="Outer line"
Case blazeOutLineTypeInner
s="Inner line"
```

```
End Select

MsgBox "Outline type: " & s

Next geom

Next drw

End Sub
```

Polyline.Parent Property

This property returns the parent of the object.

Usage

Parent as Document

Arguments

None

Description

This is a Microsoft-required property.

Polyline.Points Property

This property returns coordinates of the polyline points.

Usage

Points (*Unit* as blazeUnit on page 22, *Origin* as blazeOriginType) as Variant

Arguments

unit	[Optional] Unit in which the result is to be represented. This optional argument is blazeUnitCurrent by default.
origin	[Optional] Type of reference point from which the result is counted. The default value is blazeOriginTypeDesign.

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.

```
Sub Main

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

For Each drw In selected

  For Each geom In drw.Geometry

  If geom.ObjectType = blazeObjectTypePolyline 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

This property returns a radius for an arc in the polyline.

Usage

Radius (*Corner* as Long, *Unit* as blazeUnit on page 22) as Double

Arguments

corner	Starting corner for the arc.
unit	[Optional] Unit in which the result is to be represented. This argument is blazeUnitCurrent by default.

Description

None

Examples

The following sample code displays the arc radii of the selected polyline.

```
Sub Main
Set selected = ActiveDocument.GetObjects(blazeObjectTypeDrawing,,True)
For Each drw In selected
Dim geom As Object
For Each geom In drw.Geometry
If geom.ObjectType = blazeObjectTypePolyline 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

This property returns the shape type of the polyline.

Usage

ShapeType as blazeShapeType

Arguments

None

Description

None

Examples

The following sample code shows the shape type of the selected polyline.

```
Sub Main

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

For Each drw In selected

Dim geom as Object

For Each geom In drw.Geometry

Select Case geom.ShapeType

Case blazeShapeTypeOpen

s="Open"

Case blazeShapeTypeHollow

s="Hollow"

Case blazeShapeTypeFilled

s="Filled"
```

```
Case blazeShapeTypeVoid

s="Void"

End Select

MsgBox "Polyline type: " & s & " shape"

Next geom

Next drw

End Sub
```

The Router Object

The Router object represents batch routing and pass events in the application.

The following list identifies the Router Object properties, methods, and events:

Router.ActivePass Property

This property returns the current pass represented by the StrategyPass object.

Usage

```
ActivePass as blazeStrategyPassType on page 22
```

Arguments

None

Return Values

Router.ActivePass returns nothing if the routing process is not active.

Examples

The following sample code retrieves the name of the active pass using the StrategyPass.Name Property. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  If ActiveDocument.Router.Run Then
  MsgBox "Current routing pass is " & ActiveDocument.ActivePass.Name
  End If
End Sub
```

Related Topics

StrategyPass.Name Property

StrategyPass.Number Property

Router.Application Property

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

Usage

```
Application as Application
```

Arguments

None

Router.CompletionPercent Property

This property returns the routing completion progress in percentage. This property provides a dynamic status of the routing process.

Usage

```
CompletionPercent as Long
```

Arguments

None

Related Topics

Application.ProgressChange Event

Router.Fanout Method

This method starts fanout of selected objects. This method returns immediately and does not wait until the end of the fanout process.

Usage

```
Fanout
```

Arguments

None

Return Values

None

Router.Name Property

This property returns the name of the Router object.

Usage

```
Name as String
```

Arguments

None

Router.Optimize Method

This method starts optimizing selected objects. This method returns immediately and does not wait until the end of the Optimization process.

Usage

```
Optimize
```

Arguments

None

Return Values

None

Router.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

Router.PassComplete Property

This property determines whether a routing pass has completed or not. This property provides a dynamic status of the routing process.

Usage

```
PassComplete as Boolean
```

Arguments

None

Related Topics

Router.PassComplete Event

Router.PassStart Event

Router.PassStarted Property

Router.PassComplete Event

This event occurs when the autorouter completes a routing pass (in the active strategy).

Usage

```
PassComplete (Pass as blazeStrategyPassType on page 22)
```

Arguments

Pass	Completed pass represented by StrategyPass object

Return Values

None

Related Topics

Router.PassStart Event

Router.PassStarted Property

Router.PassComplete Property

Router.PassStart Event

This event occurs when the autorouter starts a new pass (in the active strategy).

Usage

```
PassStart (Pass as blazeStrategyPassType on page 22)
```

Arguments

Pass	Started pass represented by the StrategyPass object

Return Values

None

Related Topics

Router.PassComplete Event

Router.PassDuration Property

This property returns the duration of the current routing pass. This property provides a dynamic status of the routing process.

Usage

```
PassDuration as Long
```

Arguments

None

Related Topics

Router.TotalDuration Property

Router.PassFanoutCount Property

This property returns the number of fanouts created in the current routing pass. This property provides a dynamic status of the routing process.

Usage

```
PassFanoutCount as Long
```

Arguments

None

Related Topics

Router.TotalFanoutCount Property

Router.PassMiterCount Property

This property returns the number of miters created in the current routing pass. The Router.PassMiterCount property provides a dynamic status of the routing process.

Usage

```
PassMiterCount as Long
```

Arguments

None

Related Topics

Router.TotalMiterCount Property

Router.PassStarted Property

This property determines whether a routing pass has started or not. This property provides a dynamic status of the routing process.

Usage

```
PassStarted as Boolean
```

Arguments

None

Related Topics

Router.PassComplete Property

Router.PassStart Event

Router.PassRoutedCount Property

This property returns the number of routed links in the current routing pass. This property provides a dynamic status of the routing process.

Usage

```
PassRoutedCount as Long
```

Arguments

None

Related Topics

Router.TotalRoutedCount Property

Router.PassRoutedLength Property

This property returns the length of routed links in the current routing pass. The Router.PassRoutedLength property provides a dynamic status of the routing process.

Usage

```
PassRoutedLength as Long
```

Arguments

None

Related Topics

Router.TotalRoutedLength Property

Router.UnroutedCount Property

Router.PassTestPointCount Property

This property returns the number of test points created in the current routing pass. The Router.PassTestPointCount provides a dynamic status of the routing process.

Usage

```
PassTestPointCount as Long
```

Arguments

None

Related Topics

Router.TotalTestPointCount Property

Router.PassViaCount Property

This property returns the number of vias created in the current routing pass. The Router.PassViaCount property provides a dynamic status of the routing process.

Usage

```
PassViaCount as Long
```

Arguments

None

Related Topics

Router.TotalViaCount Property

Router.Pause Property

This property pauses or resumes autorouting. This property returns True if autorouting is paused or False if autorouting is not running. This property provides a dynamic status of the routing process. Set this property to True to pause autorouting or False to resume autorouting after a pause.

Usage

```
Pause as Boolean
```

Arguments

None

Related Topics

Router.Pause Event

Router.Run Property

Router.Resume Event

Router.Pause Event

This event occurs when autorouting is paused.

Usage

```
Pause
```

Arguments

None

Related Topics

Router.Pause Property

Router.Resume Event

This event occurs when autorouting is resumed after pausing.

Usage

```
Resume
```

Arguments

None

Related Topics

Router.Pause Property

Router.Pause Event

Router.Route Method

This method starts routing the selected objects. This method returns immediately and does not wait until the end of the routing process. The Router.Run property returns True automatically until routing is finished.

Usage

```
Route
```

Arguments

None

Return Values

None

Related Topics

Router.Fanout Method

Router.Run Property

Router.Optimize Method

Router.RouteComplete Property

This property determines whether routing has completed or not. This property provides a dynamic status of the routing process.

Usage

```
RouteComplete as Boolean
```

Arguments

None

Related Topics

Router.PassComplete Property

Router.RouteComplete Event

Router.PassStarted Property

Router.RouteComplete Event

This event occurs when autorouter completes its work and the document is routed.

Usage

```
RouteComplete
```

Arguments

None

Related Topics

Router.PassComplete Property

Router.PassStarted Property

Router.RouteCompleteType Property

This property returns the routing completion status. This property provides a dynamic status of the routing process.

Usage

```
RouteCompleteType as blazeRouteComplete on page 22
```

Arguments

None

Related Topics

Router.PassComplete Property

Router.RouteComplete Property

Router.PassStarted Property

Router.RouteComplete Event

Router.Run Property

This property starts or stops autorouting. The property returns True automatically if autorouting is in progress. Set this property to True to start autorouting or False to stop. If autorouting is paused, this property still returns True.

Usage

```
Run as Boolean
```

Arguments

None

Related Topics

Router.Pause Property

Router.RouteComplete Event

Router.RouteComplete Property

Router.TotalDuration Property

This property returns the total duration of the routing. The Router.TotalDuration property provides a dynamic status of the routing process.

Usage

```
TotalDuration as Long
```

Arguments

None

Related Topics

Router.PassDuration Property

Router.TotalFanoutCount Property

This property returns the total number of fanouts created. The Router.TotalFanoutCount property provides a dynamic status of the routing process.

Usage

```
TotalFanoutCount as Long
```

Arguments

None

Related Topics

Router.PassFanoutCount Property

Router.TotalMiterCount Property

This property returns the total number of miters created. The Router.TotalMiterCount property provides a dynamic status of the routing process.

Usage

```
TotalMiterCount as Long
```

Arguments

None

Related Topics

Router.PassMiterCount Property

Router.TotalRoutedCount Property

This property returns the total number of routed links. The Router.TotalRoutedCount provides a dynamic status of the routing process.

Usage

```
PassRoutedCount as Long
```

Arguments

None

Related Topics

Router.PassRoutedCount Property

Router.UnroutedCount Property

Router.TotalRoutedLength Property

This property returns the total length of routed links. The Router.TotalRoutedLength property provides a dynamic status of the routing process.

Usage

```
TotalRoutedLength as Long
```

Arguments

None

Related Topics

Router.PassRoutedLength Property

Router.UnroutedCount Property

Router.TotalTestPointCount Property

The property returns the total number of test points created. The Router.TotalTestPointCount property provides a dynamic status of the routing process.

Usage

```
TotalTestPointCount as Long
```

Arguments

None

Related Topics

Router.PassTestPointCount Property

Router.TotalViaCount Property

This property returns the total number of vias created. The Router.TotalViaCount property provides a dynamic status of the routing process.

Usage

```
TotalViaCount as Long
```

Arguments

None

Related Topics

Router.PassViaCount Property

Router.UnroutedCount Property

This property returns the number of remaining links to route. The Router.UnroutedCount property provides a dynamic status of the routing process.

Usage

```
UnroutedCount as Long
```

Arguments

None

Related Topics

Router.PassRoutedCount Property

Router.TotalRoutedCount Property

The RouteSegment Object

The RouteSegment object represents a physical route segment in the open design.

The following list identifies the RouteSegment Object properties:

RouteSegment.Application Property

This Property returns the application object.

Usage

```
Application as Application
```

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

This property returns the layer on which this segment is placed.

Usage

```
Layer as Integer
```

Arguments

None

RouteSegment.Length Property

This property returns the total length of this route segment.

Usage

```
Length (unit as blazeUnit on page 22) as Double
```

Arguments

unit	[Optional] Unit in which the value is returned. This argument is blazeUnitCurrent by default.

RouteSegment.Name Property

This property returns the name of this route segment.

Usage

```
Name as String
```

Arguments

None

RouteSegment.Net Property

This property returns the net this segment is connected to.

Usage

```
Net as Net
```

Arguments

None

RouteSegment.ObjectType Property

This property returns the type of the object - blazeObjectTypeRouteSegment.

Usage

```
ObjectType as blazeObjectType on page 22
```

Arguments

None

RouteSegment.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

RouteSegment.Points Property

This property returns the array of points defining this segment.

Usage

```
Points (unit as blazeUnit on page 22) as Variant
```

Arguments

unit	[Optional] Unit in which the value is returned. This argument is blazeUnitCurrent by default.

RouteSegment.SegmentType Property

This property returns the type of this segment.

Usage

```
SegmentType as blazeSegmentType
```

Arguments

None

RouteSegment.Selected Property

This property sets or returns the selection status of this object.

Usage

```
Selected as Boolean
```

Arguments

None

RouteSegment.Width Property

This property returns the width of this segment.

Usage

```
Width (unit as blazeUnit on page 22) as Double
```

Arguments

unit	[Optional] Unit in which the value is returned. This argument is blazeUnitCurrent by default.

The Strategy Object

The Strategy object represents a routing strategy defined in the application.

The following list identifies the Strategy Object properties:

Strategy.Application Property

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

Usage

```
Application as Application
```

Arguments

None

Strategy.Name Property

This property returns the name of the strategy. In the current version of the application, only one strategy is named “Default Routing Strategy.”

Usage

```
Name as String
```

Arguments

None

Strategy.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Application
```

Arguments

None

Strategy.Passes Property

This property returns the collection of all passes for this strategy.

Usage

```
Passes as Objects
```

```
Passes (name as String) as StrategyPasses
```

Arguments

name	Name of an existing pass

Return Values

When an existing pass name is passed to this property, it returns that StrategyPass object. If the pass name does not exist, this property returns the collection of all passes for the strategy in the StrategyPasses collection object.

Examples

The following sample code retrieves the number of passes for the active strategy. For more information about running this sample code, see Running Code Samples.

```
Sub Main
```

```
  MsgBox "Current Strategy has " & ActiveStrategy.Passes.Count & "
passes."
```

```
End Sub
```

Related Topics

The StrategyPasses Collection Object

The StrategyPasses Collection Object

The StrategyPasses Collection object represents a collection of strategy passes in a routing strategy.

The following list identifies the StrategyPasses Collection Object properties:

StrategyPasses.Application Property

This property returns the Application object of the application. The StrategyPasses.Application property identifies the object as an Automation object of this application. 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 various sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

Usage

```
Application as Application
```

Arguments

None

StrategyPasses.Count Property

This property returns the number of objects in the collection.

Usage

```
Count as Long
```

Arguments

None

StrategyPasses.Item Property

This property returns the object given its index or its name. The StrategyPasses.Item property is the default member of the StrategyPasses collection.

Usage

```
Item (index as Long) as Object
```

```
Item (name as String) as Object
```

Arguments

index	Index (in the collection) of the object to retrieve
name	Name of the object to retrieve

Return Values

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 StrategyPasses in the active Strategy. The second method is preferred because it is cleaner and faster. For more information about running this sample code, see Running Code Samples.

```
' Method 1: Use the StrategyPasses.Item property
Sub Main
  Set passes = ActiveStrategy.Passes
  For I=1 To passes.Count
  Set thisPass = passes.Item(I)
  ' Do something with the component thisComp
  Next I
End Sub
```

```
' Method 2: Do not use the StrategyPasses.Item property (preferred method)
Sub Main
  For Each nextPass in ActiveStrategy.Passes
  ' Do something with the component nextPass
  Next nextPass
End Sub
```

Related Topics

Application.ActiveStrategy Property

StrategyPasses.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Strategy
```

Arguments

None

The StrategyPass Object

The StrategyPass object represents a strategy pass defined in a routing strategy.

The following list identifies the StrategyPass Object properties:

StrategyPass.Application Property

This property returns the Application object of the application. The StrategyPass.Application property identifies the object as an Automation object of this application. 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.

Usage

```
Application as Application
```

Arguments

None

StrategyPass.Enabled Property

This property sets or returns True to enable the strategy pass or False to disable the strategy pass.

Usage

```
Enabled as Boolean
```

Arguments

None

StrategyPass.Intensity Property

This property returns or sets the intensity of the strategy pass.

Usage

```
Intensity as blazeStrategyPassIntensity on page 22
```

Arguments

None

Examples

The following sample code displays a message box showing the intensity of the strategy pass. For more information about running this sample code, see Running Code Samples.

```
Sub Main

  Dim status As String

  Select Case ActiveStrategy.Passes(1).Intensity

    Case blazeStrategyPassIntensityLow

    status="'Low'."

    Case blazeStrategyPassIntensityMedium

    status="'Medium'."

    Case blazeStrategyPassIntensityHigh

    status="'High'."

    Case blazeStrategyPassIntensityUnknown

    status="'Unknown'."

  End Select

  MsgBox "Strategy pass intensity is " & status
```

```
End Sub
```

StrategyPass.Name Property

This property returns the name of the strategy pass.

Usage

```
Name as String
```

Arguments

None

Related Topics

StrategyPass.Type Property

StrategyPass.Number Property

This property returns the number of the strategy pass.

Usage

```
Name as Long
```

Arguments

None

Related Topics

StrategyPass.Type Property

StrategyPass.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Application
```

Arguments

None

StrategyPass.Type Property

This property returns the type of the strategy pass.

Usage

```
Name as blazeStrategyPassType on page 22
```

Arguments

None

Related Topics

Strategy.Passes Property

The Text Object

The Text object represents free text, or text associated with a 2D line in the open design.

The following list identifies the Text Object properties and method:

Text.Application Property

This property returns the Application object.

Usage

Application As Application

Arguments

None

Description

This is a Microsoft-required property.

Text.Delete Method

This method deletes the Text object.

Usage

Delete as Boolean

Arguments

None

Description

None

Examples

The following sample deletes the selected text.

```
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
```

Text.Drawing Property

This 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.

```
Sub Main

Set selected = ActiveDocument.GetObjects(blazeObjectTypeText, , 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

This property returns or sets text or label height.

Usage

Height (*Unit* as blazeUnit on page 22) as Double

Arguments

unit	[Optional] Unit in which the result is to be represented. This optional argument is blazeUnitCurrent by default.

Description

None

Examples

The following example shows the height of a selected text object, and then sets the height to 100 mils.

```
Sub Main

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

For Each text In selected

MsgBox "Selected text height: " & text.Height

text.Height(blazeUnitMils) = 100

Next text

End Sub
```

The following example shows the height of selected label object, and then sets height to 100 mils.

```
Sub Main

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

For Each label In selected

MsgBox "Selected label height: " & label.Height
```

```
label.Height(blazeUnitMils) = 100
```

```
Next label
```

```
End Sub
```

Text.HorzJustification/Label.HorzJustification Property

This property returns or sets the horizontal justification type of the text or label object.

Usage

HorzJustification as blazeHorizontalJustification

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.

```
Sub Main
Set selected = ActiveDocument.GetObjects(blazeObjectTypeText,, TRUE)
For Each text In selected
Select Case text.HorzJustification
Case blazeJustifyLeft
MsgBox "Horizontal justification: Left"
Case pcbJustifyHCenter
MsgBox "Horizontal justification: Center"
Case pcbJustifyRight
MsgBox "Horizontal justification: Right"
End Select
text.HorzJustification = blazeJustifyHCenter
```

```
Next text

End Sub
```

The following example shows the horizontal justification setting of the selected label object, and then sets the justification to center.

```
Sub Main

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

For Each label In selected

Select Case label.HorzJustification

Case blazeJustifyLeft

MsgBox "Horizontal justification: Left"

Case blazeJustifyHCenter

MsgBox "Horizontal justification: Center"

Case blazeJustifyRight

MsgBox "Horizontal justification: Right"

End Select

label.HorzJustification = blazeJustifyHCenter

Next label

End Sub
```

Text.Layer/Label.Layer Property

This 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.

```
Sub Main

Set selected = ActiveDocument.GetObjects(blazeObjectTypeText,, 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.

```
Sub Main

Set selected = ActiveDocument.GetObjects(blazeObjectTypeLabel,, 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

This property returns or sets the line width of the text or label.

Usage

LineWidth (*Unit* as blazeUnit on page 22) as Double

Arguments

unit	[Optional] Unit in which the result is to be represented. This optional argument is blazeUnitCurrent 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.

```
Sub Main

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

For Each text In selected

MsgBox "Selected text line width: " & text.LineWidth

text.LineWidth(blazeUnitMils) = 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.

```
Sub Main

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

For Each label In selected
```

```
MsgBox "Selected label line width: " & label.LineWidth

label.LineWidth(blazeUnitMils) = 10

Next label

End Sub
```

Text.Mirror/Label.Mirror Property

This property returns or sets the mirrored state of the text or label.

Usage

Mirror(*Origin* as blazeOriginType) as Boolean

Arguments

origin	Shows whether the mirror status is relative to the object's parent or to the design (optional). The default value is blazeOriginTypeDesign.

Description

None

Examples

The following example shows whether the selected text object is mirrored, and then mirrors the text.

```
Sub Main

Set selected = ActiveDocument.GetObjects(blazeObjectTypeText,, 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.

```
Sub Main

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

For Each label In selected

MsgBox "Is selected label mirrored? " & label.Mirror
```

```
label.Mirror = True
```

```
Next label
```

```
End Sub
```

Text.Name Property

This default property 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.

```
Sub Main

Set text = ActiveDocument.Texts(1)

MsgBox "The first free text's name is " & text.Name

End Sub
```

Text.ObjectType Property

This property returns the type of object.

Usage

ObjectType as blazeObjectType on page 22

Arguments

None

Description

None

Examples

The following sample shows the number of selected texts.

```
Sub Main

Set sel = ActiveDocument.GetObjects(,,TRUE)

n = 0

For Each obj In sel

If obj.ObjectType = blazeObjectTypeText Then n = n + 1

Next obj

MsgBox n & " text(s) selected"

End Sub
```

Text.Orientation/Label.Orientation Property

This property returns or sets the rotation angle of the text or label.

Usage

Orientation(*Origin* as blazeOriginType) as Double

Arguments

origin	Shows whether the orientation value is relative to the object's parent or to the design (optional). The default value is blazeOriginTypeDesign.

Description

None

Examples

The following example shows the orientation of the selected text object, and then sets the orientation to 90 degrees.

```
Sub Main

Set selected = ActiveDocument.GetObjects(blazeObjectTypeText,, 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.

```
Sub Main

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

For Each label In selected
```

```
MsgBox "Selected label orientation: " & label.Orientation

label.Orientation = 90

Next label

End Sub
```

Text.Parent Property

This property returns the parent of this object.

Usage

Parent As Document

Arguments

None

Description

This is a Microsoft-required property.

Examples

Not required.

Text.PositionX/Label.PositionX Property

This property returns or sets the x-coordinate of the origin of the Text or Label object.

Usage

PositionX (*Unit* as blazeUnit on page 22, Origin as blazeOriginType) as Double

Arguments

unit	[Optional] Unit in which the result is to be represented. This optional argument is blazeUnitCurrent by default.
origin	[Optional] Type of reference point from which the result is counted. The default value is blazeOriginTypeDesign.

Description

None

Examples

The following example shows the position of a selected text object, and then sets the position to (200, 200) mil.

```
Sub Main

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

For Each text In selected

MsgBox "Selected text position: (" & text.PositionX & ", " & text.PositionX
 & ")"

text.PositionX(blazeUnitMils) = 200

text.PositionY(blazeUnitMils) = 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.

```
Sub Main
```

```
Set selected = ActiveDocument.GetObjects(blazeObjectTypeLabel,, TRUE)

For Each label In selected

MsgBox "Selected label position: (" & label.PositionX & ", " &
 label.PositionX & ")"

label.PositionX(blazeUnitMils) = 200

label.PositionY(blazeUnitMils) = 200

Next label

End Sub
```

Text.PositionY/Label.PositionY Property

This property returns or sets the y-coordinate of the origin of the text or label object.

Usage

PositionY(*Unit* as blazeUnit on page 22, Origin as blazeOriginType) as Double

Arguments

unit	[Optional] Unit in which the result is to be represented. This optional argument is blazeUnitCurrent by default.
origin	[Optional] Type of reference point from which the result is counted. The default value is blazeOriginTypeDesign.

Description

None

Examples

The following example shows the position of the selected text object, and then sets position to (200, 200) mil.

```
Sub Main

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

For Each text In selected

MsgBox "Selected text position: (" & text.PositionX & ", " & text.PositionX
 & ")"

text.PositionX(blazeUnitMils) = 200

text.PositionY(blazeUnitMils) = 200

Next text

End Sub
```

The following example shows the position of the selected label object, and then sets position to (200, 200) mil.

```
Sub Main
```

```
Set selected = ActiveDocument.GetObjects(blazeObjectTypeLabel,, TRUE)

For Each label In selected

MsgBox "Selected label position: (" & label.PositionX & ", " &
 label.PositionX & ")"

label.PositionX(blazeUnitMils) = 200

label.PositionY(blazeUnitMils) = 200

Next label

End Sub
```

Text.Selected Property

This 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.

```
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

This 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!”

```
Sub Main

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

For Each text In selected

MsgBox "Selected text: " & text.Text

text.Text="Hello !"

Next text

End Sub
```

Text.VertJustification/Label.VertJustification Property

This property returns or sets the vertical justification type of the text or label object.

Usage

VertJustification as blazeVerticalJustification

Arguments

None

Description

None

Examples

The following example shows the vertical justification setting of the selected text object, then sets the justification to center.

```
Sub Main

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

For Each text In selected

Select Case text.VertJustification

Case blazeJustifyBottom

MsgBox "Vertical justification: Bottom"

Case blazeJustifyVCenter

MsgBox "Vertical justification: Center"

Case blazeJustifyTop

MsgBox "Vertical justification: Top"

End Select

text.VertJustification = blazeJustifyVCenter
```

```
Next Text

End Sub
```

The following example shows the vertical justification setting of the selected label object, then sets the justification to center.

```
Sub Main

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

For Each label In selected

Select Case label.VertJustification

Case blazeJustifyBottom

MsgBox "Vertical justification: Bottom"

Case blazeJustifyVCenter

MsgBox "Vertical justification: Center"

Case blazeJustifyTop

MsgBox "Vertical justification: Top"

End Select

label.VertJustification = blazeJustifyVCenter

Next Label

End Sub
```

The ThermalPad Object

This object represents a thermal pad in the padstack definition.

The following list identifies the ThermalPad Object properties:

ThermalPad.Application Property

This property returns the application object.

Usage

```
Application as Application
```

Arguments

None

ThermalPad.CornerRadius Property

This property returns the Thermal Pad corner radius.

Usage

```
CornerRadius as Double
```

```
      CornerRadius (unit as blazeUnit on page 22)  as Double
```

Arguments

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

ThermalPad.CornerType Property

This property returns the Thermal Pad corner type.

Usage

```
CornerType as blazePadCornerType
```

Arguments

None

ThermalPad.InnerLength Property

This property returns the Thermal Pad inner length.

Usage

```
InnerLength as Double
```

```
      InnerLength (Unit as blazeUnit on page 22)  as Double
```

Arguments

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

ThermalPad.InnerSize Property

This property returns the thermal pad's inner size. For shape blazeThermalPadShapeRound it returns inner diameter.

Usage

```
InnerSize (Unit as blazeUnit on page 22) as Double
```

Arguments

unit	[Optional] Unit in which the value is returned. This optional argument is blazeUnitCurrent by default.

ThermalPad.Name Property

This property returns the name of this thermal pad.

Usage

```
Name as String
```

Arguments

None

ThermalPad.ObjectType Property

This property returns the type of the object - blazeObjectTypeThermalPad.

Usage

```
ObjectType as blazeObjectType on page 22
```

Arguments

None

ThermalPad.Offset Property

This property returns the Thermal Pad offset.

Usage

```
Offset as Double
```

```
      Offset  (unit as blazeUnit on page 22)  as Double
```

Arguments

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

ThermalPad.Orientation Property

This property returns the Thermal Pad orientation.

Usage

```
Orientation as Double
```

Arguments

None

ThermalPad.OuterSize Property

This property returns the thermal pad's outer size. For shape blazeThermalPadShapeRound it returns outer diameter.

Usage

```
OuterSize (unit as blazeUnit on page 22) as Double
```

Arguments

unit	[Optional] Unit in which the value is returned. This optional argument is blazeUnitCurrent by default.

ThermalPad.PadStackLayer Property

This property returns the PadStackLayer Object to which this thermal pad belongs to.

Usage

```
PadStackLayer as blazePadStackLayerType
```

Arguments

None

ThermalPad.Parent Property

This property returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

ThermalPad.Shape Property

This property returns the thermal pad shape.

Usage

```
Shape as blazeThermalPadShape
```

Arguments

None

ThermalPad.SpokeAngle Property

This property returns the thermal pad's spoke angle.

Usage

```
SpokeAngle as Double
```

Arguments

None

ThermalPad.Spokes Property

This property returns the thermal pad's number of spokes.

Usage

```
Spokes as Integer
```

Arguments

None

ThermalPad.SpokeWidth Property

This property returns the thermal pad's spoke width.

Usage

```
SpokeWitdth (unit as blazeUnit on page 22) as Double
```

Arguments

unit	[Optional] Unit in which the value is returned. This optional argument is blazeUnitCurrent by default.

The Via Object Property

The Via object represents a physical via, which exists in the PCB design currently open in the application.

The following list identifies the Via Object properties:

Via.Application Property

This property returns the Application object of the application. The Via.Application property identifies the object as an Automation object of this application. 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 various sources, such as different Automation server applications, to quickly identify the application to which the object belongs.

Usage

```
Application as Application
```

Arguments

None

Via.Name Property

This property returns the name of the via. The Via.Name property is the default property for the Via object.

Usage

```
Name as String
```

Arguments

None

Via.Net Property

This property returns the net connected to the via.

Usage

```
Net as Net
```

Arguments

None

Examples

The following sample code retrieves the net connected to the first via found in the open design, assuming that at least one via exists. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  Set firstVia = ActiveDocument.Vias(1)
  MsgBox firstVia.Name & " is connected to net " & firstVia.Net.Name
End Sub
```

Via.ObjectType Property

This property returns the type of the object. All database objects of the application's Automation server implement the Via.ObjectType property to compensate for the lack of a Visual Basic equivalent for the Visual C++ QueryInterface function.

Usage

```
ObjectType as blazeObjectType on page 22
```

Arguments

None

Return Values

This property always returns blazeObjectTypeVia

Via.PadStackLayers Property

This property returns the collection of all padstack layers for this via.

Usage

```
PadStackLayers as Objects
```

```
PadStackLayers(layerName as String) as blazePadStackLayerType on page 22
```

Arguments

layerName	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

This property returns the parent of the object.

Usage

```
Parent as Document
```

Arguments

None

Via.PositionX Property

This property returns the x-coordinate of the via.

Usage

```
PositionX([unit as blazeUnit on page 22]) as Double
```

Arguments

unit	[Optional] Unit in which the x-coordinate is returned

Examples

The following sample code retrieves the location of the first via, assuming that at least one via exists in the open design, in current design units. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  Set firstVia = ActiveDocument.Vias(1)
  MsgBox "First Via position is = (" & firstVia.PositionX & ", " &
 firstVia.PositionY & ")"
End Sub
```

Related Topics

Via.PositionY Property

Via.PositionY Property

This property returns the y-coordinate of the via.

Usage

```
PositionY([unit as blazeUnit on page 22]) as Double
```

Arguments

unit	[Optional] Unit in which the y-coordinate is returned

Examples

The following sample code retrieves the location of the first via, assuming that at least one via exists in the open design, in current design units. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  Set firstVia = ActiveDocument.Vias(1)
  MsgBox "First Via position is = (" & firstVia.PositionX & ", " &
 firstVia.PositionY & ")"
End Sub
```

Related Topics

Via.PositionX Property

Via.Selected Property

This property sets or returns whether the via is selected. You can also select a database object of the application using the Document.SelectObjects and Objects.Select methods.

Usage

```
Selected as Boolean
```

Arguments

None

Examples

The following sample code selects via U1.1 only, assuming U1.1 exists in the open design. For more information about running this sample code, see Running Code Samples.

```
Sub Main
  ActiveDocument.SelectObjects(,,False)
  ActiveDocument.Vias("U1.1").Selected = True
End Sub
```

Related Topics

Document.SelectionChange Event

Document.SelectObjects Method

Objects.Select Method

Via.TestPoint Property

This property determines whether the via is a test point or not.

Usage

```
TestPoint as blazeTestPoint on page 22
```

Arguments

None

The View Object

The following is a list of properties for the View Object.

The following list identifies the View Object properties:

View.PointerX Property
View.PointerY Property

View.PointerX Property

This property returns the pointer's X position.

Usage

```
PointerX (unit as blazeUnit on page 22) as Double
```

Arguments

unit	[Optional] Unit in which the value is returned. This optional argument is blazeUnitCurrent by default.

Examples

```
' load preview.pcb
```

```
Application.ExecuteCommand("Search Absolute", "C8")
```

```

```

```
doc = Application.ActiveDocument
```

```
view = doc.ActiveView
```

```
MsgBox view.PointerX & ", " & view.PointerY
```

View.PointerY Property

This property returns the pointer's Y position.

Usage

```
PointerY (unit as blazeUnit on page 22) as Double
```

Arguments

unit	[Optional] Unit in which the value is returned. This optional argument is blazeUnitCurrent by default.

Examples

```
' load preview.pcb
```

```
Application.ExecuteCommand("Search Absolute", "C8")
```

```
doc = Application.ActiveDocument
```

```
view = doc.ActiveView
```

```
MsgBox view.PointerX & ", " & view.PointerY
```

Constants

The following list identifies the constants.

- blazeAntiPadShape
- blazeDesignObject
- blazeDocumentColor
- blazeDielectricLayer
- blazeDielectricType
- blazeLayerColor
- blazeLineStyle

- blazeObjectType
- blazePadCornerType
- blazePadShape
- blazePadStackLayerType
- blazePlaneType
- blazeRoutingDirection
- blazeSegmentType
- blazeThermalPadShape

blazeAntiPadShape

blazeAntiPadShapeRound	= 8
blazeAntiPadShapeSquare	= 9

blazeDesignObject

blazeDesignObjectPad	= 0
blazeDesignObjectTrace	= 1
blazeDesignObjectVia	= 2
blazeDesignObjectCopper	= 3
blazeDesignObjectText	= 4
blazeDesignObjectLine	= 5
blazeDesignObjectError	= 6
blazeDesignObjectRefDes	= 7
blazeDesignObjectPinNumber	= 8
blazeDesignObjectKeepout	= 9
blazeDesignObjectOutlineTop	= 10
blazeDesignObjectOutlineBottom	= 11
blazeDesignObjectPlacementTop	= 12

blazeDesignObjectPlacementBottom	= 13

blazeDocumentColor

blazeDocumentColorBackground	= 0
blazeDocumentColorSelection	= 1
blazeDocumentColorConnection	= 2
blazeDocumentColorBoardOutline	= 3
blazeDocumentColorTestPoint	= 4
blazeDocumentColorThermal	= 5
blazeDocumentColorGuardBand	= 6

blazeDielectricLayer

blazeDielectricLayerAbove	= 0
blazeDielectricLayerBelow	= 1

blazeDielectricType

blazeDielectricTypeCoating	= 0
blazeDielectricTypeSubstrate	= 1
blazeDielectricTypePrepreg	= 2

blazeLayerColor

blazeLayerColorPad	= 0
blazeLayerColorTrace	= 1
blazeLayerColorVia	= 2
blazeLayerColorCopper	= 3
blazeLayerColorText	= 4
blazeLayerColorLine	= 5
blazeLayerColorError	= 6
blazeLayerColorRefDes	= 7

blazeLayerColorPinNumber	= 8
blazeLayerColorKeepout	= 9
blazeLayerColorOutlineTop	= 10
blazeLayerColorOutlineBottom	= 11
blazeLayerColorPlacementTop	= 12
blazeLayerColorPlacementBottom	= 13

blazeLineStyle

blazeLineStyleSolid	= 0
blazeLineStyleDashed	= 1
blazeLineStyleDotted	= 2
blazeLineStyleDashDotted	= 3
blazeLineStyleDashDoubleDotted	= 4
blazeLineStyleMultiple	= 5

blazeObjectType

blazeObjectTypeUnknown	= 0 The server returns this value to indicate an invalid object. The client may use this value when working with empty object collections.
blazeObjectTypeComponent	= 1
blazeObjectTypeNet	= 2
blazeObjectTypePin	= 3
blazeObjectTypeVia	= 4
blazeObjectTypeNetClass	= 9
blazeObjectTypePadStackLayer	= 31
blazeObjectTypePad	= 32
blazeObjectTypeThermalPad	= 33
blazeObjectTypeAntiPad	= 34
blazeObjectTypeLayer	= 35

blazeObjectTypeError	= 36
blazeObjectTypeErrorConflict	= 37
blazeObjectTypeAssociatedNet	= 38
blazeObjectTypeAll	= 9999 All Automation database object types, including Component, Net, NetClass, Pin, and Via.

blazePadCornerType

blazePadCornerType90Degree	= 0
blazePadCornerTypeChamfered	= 1
blazePadCornerTypeRounded	= 2

blazePadShape

blazePadShapeOvalFinger	= 0
blazePadShapeRectangularFinger	= 1
blazePadShapeRound	= 2
blazePadShapeSquare	= 3
blazePadShapeAnnular	= 4
blazePadShapeOdd	= 5

blazePadStackLayerType

blazePadStackLayerTypeMounted	= -2
blazePadStackLayerTypeInner	= -1
blazePadStackLayerTypeOpposite	= 0

blazePlaneType

blazePlaneTypeNoPlane	= 0
blazePlaneTypeCAMPlane	= 1
blazePlaneTypeSplitMixedPlane	= 2

blazeRoutingDirection

blazeRoutingDirectionHorizontal	= 0
blazeRoutingDirectionVertical	= 1
blazeRoutingDirectionAny	= 2
blazeRoutingDirectionDiagonal45	= 3
blazeRoutingDirectionDiagonal135	= 4
blazeRoutingDirectionValue	= 5

blazeSegmentType

blazeSegmentUnknown	=0
blazeSegmentLine	=1
blazeSegmentArc	=2

blazeThermalPadShape

blazeThermalPadShapeRound	= 6
blazeThermalPadShapeSquare	= 7

Chapter 2
Macros

You can edit, run, and debug macros in the Macro tab. A macro is any combination of commands, keystrokes, and mouse clicks that you record to later replay. You can record virtually any set of procedural steps for replay, thus simplifying redundant activities such as setting preferences and settings.

Macro Features

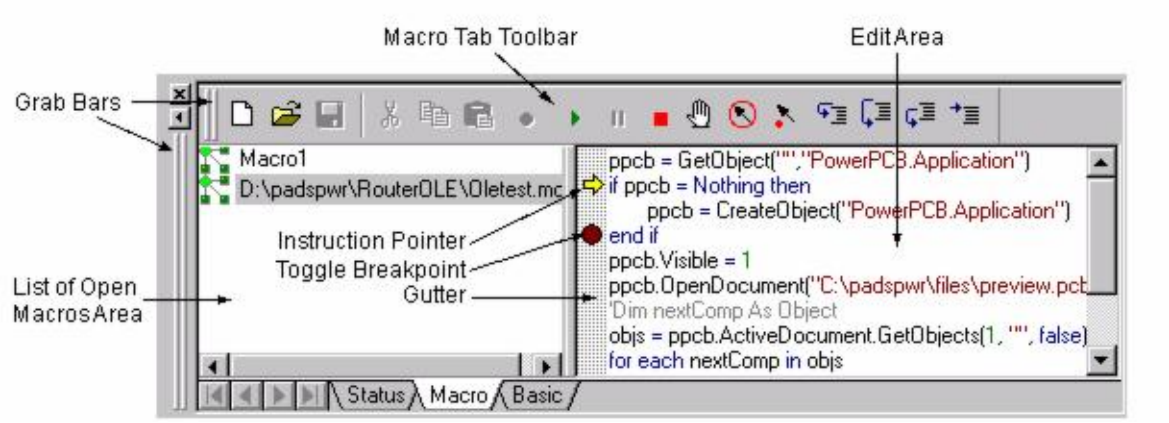
The Macro editor supports standard editing features. It includes unlimited Undo/Redo operations, syntax highlighting, and provides context-sensitive help on the Macro language. You can also open multiple macros in the macro editor. You can move between open macros and edit them. You can nest macros within other macros.

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.

Figure 3 shows the screen elements that appear in the Macro tab when you open a macro.

Figure 3. Screen Elements on the Macro Tab



- List of Open Macros Area lists the path and file name of any open macros. To switch to another open macro, select the macro in the list.

- Edit Area is for editing and creating macros.

Macro Editing Shortcuts

You use standard Windows editing functions when editing macros. Use the mouse to select areas of text.

The following table shows the default shortcut keys for macro editing functions.

Table 15. 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.

Table 15. Default Macro Editing Shortcuts (continued)

Command	Description
Alt+Backspace, Ctrl+Z	Undoes the last edit. This is a multilevel undo. See also "Undo and redo" in SailWind Router Help
Ctrl+Shift+Z	Restores the edit rescinded by the Undo command. See also "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.

Reporting Run-Time Errors

If a run-time error occurs, the debugger suspends running the macro, and displays a message on the status bar. The Instruction pointer is set on the line that produced the error. After fixing the error, you can resume running the macro.

Recording Mouse Movements

When recording a macro, you can record mouse movements in these ways:

- Record movement from coordinate A to coordinate B without recording the intermediate coordinates (compressed).
- Record movement from coordinate A to coordinate B including all of the intermediate coordinates (uncompressed).
- Record movement from coordinate A to coordinate B using incremental coordinates (relative).
- Record movement from coordinate A to coordinate B using absolute coordinates.

Compressed Mouse Mode

Compressed mouse mode records only the start point and end point of a mouse movement. Compression is recommended under most circumstances because it significantly reduces the size of your macro file. Recording intermediate mouse movements increases the file size, but documents coordinate information if required for a special application.

Relative Mouse Mode

Relative mouse mode records the start point and end point of a movement in incremental coordinates instead of absolute coordinates.

Recording a session

You can record a session upon start-up using command line switches. You can record macros in the background or load and run a macro when you start the program.

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.

To enable this feature, modify the shortcut and add a "–log" command line parameter.

Tip
If you use the SailWind Router Link from SailWind Layout, the .log file is not written.

For more information, refer to "Recording a Session" in SailWind Router Help

Using Command Line Switches with Macros

Using command line switches, you can record your session, load and run a macro, or record a macro to a specific file and location when you start SailWind Router.

The following descriptions are included in this topic:

Recording a Session

You can record your session of working in SailWind Router.

Procedure

1. In the Start menu, navigate to the shortcut for SailWind Router.
2. Right-click the shortcut and click **Properties**.
3. Click the **Shortcut** tab and click in the Target box.
4. At the end of the existing shortcut, type: -log. Make sure to type a space between for example "SailWindRouter.exe" and the "-log" command.
5. Click **OK**. When you run the program, a log is created.
6. **Restriction:** SailWind Router log files are not created when you use the link to SailWind Router in SailWind Layout to start the program.

Recording Log Files to a Specific File and Location

You can specify the filename and location to record the log files.

Procedure

1. In the Start menu, navigate to the shortcut for SailWind Router.
2. Right-click the shortcut and click **Properties**.
3. Click the **Shortcut** tab and click in the Target box.
4. At the end of the existing shortcut, type: -log:[path and file name]. For example: -log:C:\SailWind Projects\mylog.txt. Make sure to type a space between "SailWindRouter.exe" and the "-log" command.
5. Click **OK**. When you run SailWind Router, a log is created in the specified location using the specified name.
6. **Restriction:** SailWind Router log files are not created when you use the link to SailWind Router in SailWind Layout to start the program.

Running a Macro When You Start the Program

You can run a macro at start-up when you start SailWind Router.

Procedure

1. In the Start menu, navigate to the shortcut for SailWind Router.
2. Right-click the shortcut and click **Properties**.
3. Click the **Shortcut** tab and click in the Target box.
4. At the end of the existing shortcut, type:

   -run=[path and file name]

   For example:

   -run=C:\SailWind Projects\Samples\mymacro.mcr

   Requirement: Make sure to type a space between "SailWindRouter.exe" and the "-run" command.

   Alternative: You can use a / instead of the hyphen (for example, /run=mymacro.mcr).
5. Click **OK**. When you run SailWind Router, the specified macro runs as soon as it starts.
6. **Restriction:** SailWind Router log files are not created when you use the link to SailWind Router in SailWind Layout to start the program.

Introducing 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
- The full set of standard arithmetic and Boolean Expressions
- Functions and subroutines
- Statements
- Operators
- , properties, and methods
- Automation Support for internal and external automation objects
- Internal Macro Objects

Variables

The macro engine of this program supports variables, which can either be Null or contain values of the following types:

- Numeric
- Logical
- String
- Object

Value types are converted to each other according to these rules:

Table 16. & 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

Table 16. & Operator Arguments (continued)

From...To...	Logical	Numeric	String
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.

Note: The Numeric and String value types are interchangeable; they automatically convert into one another upon assignment.

For more information, see Str function.

Double

Represents numeric value.

The double and string types are interchangeable; that is, they automatically convert into one another upon assignment.

Object

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.

The syntax for both object types is the same:

```
Object.Method arg1, ..., argn
```

```
var = Object.Method( arg1, ..., argn )
```

Expressions

The macro engine of this program uses either of the following 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 uses the following operators:

& Operator

Forces string concatenation of two expressions.

Usage

```
result = expression1 & expression2
```

Arguments

The & operator has these 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.

Examples

```
S="abc" & "123"
```

* Operator

Multiplies two numbers.

Usage

```
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.

Usage

```
result = expression1 + expression2
```

Arguments

The + operator syntax has these arguments:

result	[Required] Any numeric variable
expression1	[Required] Any expression
expression2	[Required] Any expression

Examples

```
X = y + z
```

/ Operator

Divides one number by a second number and returns a floating-point result.

Usage

```
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.

Usage

```
result = number1 - number2
```

The - operator is the arithmetic subtraction operator, which finds the difference between two numbers.

```
-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
```

Or another example:

```
-x
```

= Operator

Assigns a value to a variable or property.

Usage

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.

Usage

```
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.

Usage

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 determined:

Table 17. And Operator Results

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

Compare expressions.

Usage

```
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:

Table 18. Comparison Operators and Results

Comparison Operator	True if	False if
< (Less than)	*expression1* < *expression2*	*expression1* >= *expression2*
<= (Less than or equal to)	*expression1* <= *expression2*	*expression1* > *expression2*
> (Greater than)	*expression1* > *expression2*	*expression1* <= *expression2*
>= (Greater than or equal to)	*expression1* >= *expression2*	*expression1* < *expression2*
= (Equal to)	*expression1* = *expression2*	*expression1* <> *expression2*
<> (Not equal to)	*expression1* <> *expression2*	*expression1* = *expression2*

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.

Usage

```
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.

Usage

```
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:

Table 19. Not Operator Results

If expression is	Then result is
True	False
False	True

Examples

```
x = Not y
```

Or Operator

Performs a logical disjunction on two expressions.

Usage

```
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:

Table 20. Or Operator Results

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.

Usage

```
[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:

Table 21. Xor Operator Results

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 following VBScript and other statements.

Call

Transfers control to a sub procedure or function procedure.

Usage

```
[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.

Usage

```
Close [filenumberlist]
```

Arguments

The Close statement has this argument:

filenumberlist	[Optional] Can be one or more file numbers using the following syntax, where *filenumber* 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.

Usage

```
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.

Usage

```
Do [{While | Until} condition]
[statements]
[Exit Do]
[statements]
Loop
```

And another usage:

```
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 *condition* is Null, *condition* is treated as False.
statements	One or more statements that are repeated while, or until, *condition* 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.

Usage

```
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. *Counter* cannot be a Boolean element or an array element.
start	[Required] Initial value of *counter*.
end	[Required] Final value of *counter*.
step	[Optional] The number by which *counter* 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 *counter*-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:

Table 22. For-Next Statement Loop Counter

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.

Usages

```
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.

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:

Table 23. Function Statement arglist Syntax

Part	Description
ByVal	Indicates that the argument is passed by value.
ByRef	Indicates that the argument is passed by reference.
Varname	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.

Usage

```
If condition Then [statements] [Else [elsestatements]]
```

Block Syntax:

```
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 *statements* executed when associated *condition-n* is True.
elsestatements	[Optional] One or more *statements* executed when no previous condition or when *condition-n* 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, ElseIf, 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 ElseIf clauses are both optional. You can have as many ElseIf 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 ElseIf condition is evaluated in turn. When a True condition is found, the statements immediately following the associated Then are executed. If none of the ElseIf conditions are True (or if there are no ElseIf 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.

Usage

```
Input #filenumber, varlist
```

Arguments

The Input # statement has these arguments:

filenumber	[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 SailWind 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.

Usage

```
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.

Usage

```
Open pathname For mode [Access access] [lock] As [#]filenumber
 [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.

Usage

```
Print # filenumber, [outputlist]
```

Arguments

The Print # statement has these arguments:

filenumber	[Required] Any valid file number.
outputlist	[Optional] An expression or list of expressions to print. Nothing is written to the file when *outputlist* is empty. However, when *outputlist* is Null, Null is written to the file.

If you omit *outputlist* and include only a list separator after *filenumber*, 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 24 describes the outputlist syntax elements:

Table 24. 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.
expression	Numeric or string expressions to print. You can separate multiple expressions with either a space or a semicolon.

Table 24. Print # Statement outputlist Syntax (continued)

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 #.

And another example:

```
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 se ReDim repetedly to change the number of elements and dimensions in an array.

Note:
If you re-declare a dimension for an array variable whose size is explicitly specified in a DIM statement, an error occurs.

Usage

```
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: `[lower To] upper [,[lower To] upper] . . .` 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.

Usage

```
Set objectvar = {objectexpression | Nothing}
```

Arguments

The Set statement has these arguments:

objectvar	[Required] Name of the variable, or property; follows standard variable-naming conventions
objectexpression	[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 *objectvar* with any specific object. Assigning Nothing to *objectvar* 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.

Usages

```
Sub name [(arglist)]
[statements]
[Exit Sub]
[statements]
End Sub
```

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 25 describes the arglist syntax elements:

Table 25. Sub Statement arglist Syntax

Part	Description
ByVal	Indicates that the argument is passed by value.

Table 25. Sub Statement arglist Syntax (continued)

ByRef	Indicates that the argument is passed by reference.
varname	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.

Usage

```
While condition
[statements]
Wend
```

Arguments

The While Wend statement has these arguments:

condition	[Required] Any expression. If the expression is not Logical, it is converted to a Logical type.
statements	[Optional] One or more statements executed while *condition* 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.

Usage

```
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 the following 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.

Usage

```
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.

For more information, see Double on page 451.

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).

Usages

```
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.

Usage

```
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.

Usage

```
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\SailWindRouter.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.

Usage

```
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.

Usage

```
CreateObject(class,[servername])
```

Arguments

The CreateObject function syntax has these arguments:

class	[Required] String The application name and the class of the object to create.
servername	[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:

appname	[Required] String The name of the application providing the object.
objecttype	[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.

Usage

```
CurDir [(drive)]
```

Arguments

The CurDir function has this argument:

drive	[Optional] Any string expression that specifies an existing drive. If no drive is specified or if *drive* 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.

Usage

```
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.

Usage

```
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.

Usage

```
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 that is generated when attempting to get input after the end of a file.

Usage

```
Eof [(filenumber)]
```

Arguments

The Eof function has this argument:

filenumber	[Optional] An integer containing any valid file number

Return Values

Table 26 lists the Eof function returned values:

Table 26. 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
File opened for binary access and Get statement is not able to read an entire record	True
File opened for output	True

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.

For more information, see Double on page 451.

Usages

```
Exp(number)
```

The Exp function has this argument:

number	[Required] A double or any valid numeric expression. When the value of *number* exceeds 709.782712893, an error occurs.

Examples

```
x = exp(y)
```

GetObject

This function returns a reference to an object provided by an ActiveX component.

Usage

```
GetObject([pathname] [, class])
```

Arguments

The GetObject function has these arguments:

pathname	[Optional] Variant string The full path (folder, and drive) and name of the file containing the object to retrieve. If *pathname* is omitted, class is required.
class	[Optional] Variant string A string representing the class of the object.

The *class* argument uses the syntax *appname.objecttype*, which has these elements:

appname	[Required] Variant string The name of the application providing the object.
objecttype	[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.

Usage

```
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.

Usage

```
InStr([start, ]string1, string2)
```

Arguments

The InStr function has these arguments:

start	[Optional] Numeric expression that sets the starting position for each search. When *start* is omitted, the search begins at the first character position.
string1	[Required] String expression to search.
string2	[Required] String expression to find.

Return Values

Table 27. 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.

Usage

```
InStrRev(string1, string2, [start])
```

Arguments

The InStrRev function has these arguments:

string1	[Required] String expression being searched.
string2	[Required] String expression to find.
start	[Optional] Numeric expression that sets the starting position for each search. If omitted, the search begins at the last character position.

Return Values

Table 28. InStrRev Function Return Values

If	InStrRev returns
string1 is zero-length	0
string2 is zero-length	Start
string2 is not found	0
string2 is found within string1	Position at which match is found
start > Len(string2)	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.

Usage

```
Left(string, length)
```

Arguments

The Left function has these arguments:

string	[Required] A string expression from which the characters from the left side are returned
length	A numeric expression indicating the number of characters to return If *length* is 0, Left returns a zero-length string (" "). If *length* 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).

Usage

```
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.

Usage

```
Mid (string, start, [length])
```

Arguments

The Mid function has these arguments:

string	[Required] A string expression from which the characters are returned
start	[Required] The character position in *string* at which the part to return begins. If start is greater than the number of characters in *string*, Mid returns a zero-length string ("")
length	[Optional] The number of characters to return. If omitted, when there are less characters in the string (including the character at start), than *length*, *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.

Usage

```
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.

Usage

```
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.

Usage

```
MsgBox(prompt [, buttons] [, title])
```

Arguments

The MsgBox function has these arguments:

prompt	[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 *prompt* 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.
buttons	[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 *buttons* is 0.
title	[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:

Table 29. 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.
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 30. MsgBox Return Values

Constant	Value	Description

Table 30. MsgBox Return Values (continued)

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.

Usage

```
Right(string, length)
```

Arguments

The Right function has these arguments:

string	[Required] A string expression from which the characters on the right side are returned
length	A numeric expression indicating how many characters to return If *length* is 0, Right returns a zero-length string (" "). If *length* 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.

For more information, see Double on page 451.

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.

Usages

```
Sin(number)
```

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.

For more information see Print # statement or the Print method.

Usages

```
Spc(n)
```

The Spc function has this argument:

n	[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.

Usage

```
Str(number)
```

Arguments

The Str function has the following argument:

number	[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.

Usages

```
Tab[(n)]
```

For more information see Print # statement or the Print method.

The Tab function has this argument:

n	[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 nth 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 right most 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.

Usage

```
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 this program uses the following dialog box controls:

CheckBox

This control 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.

State

This method sets the state of the check box.

Usage

```
checkbox.State(iState)
```

Arguments

State has this argument:

iState	[Required] A numeric expression, representing the check box state to set

iState may have one of the following values:

Table 31. CheckBox.State istate Values

Value	Description
0, or False	Unchecked
1, or True	Checked
2	Indeterminate

Value Property

This method returns or sets the state of the check box control.

Usage

```
checkbox.Value[=iState]
```

Arguments

The Value Property has this argument:

iState	[Required] A numeric expression, representing the check box state to set

iState may have one of the following values::

Table 32. CheckBox.Value istate Values

Value	Description
0, or False	Unchecked
1, or True	Checked
2	Indeterminate

CheckListBox

This control represents a check list box on a dialog box.

State

This method sets the selection state of the check list box.

Usage

```
CheckListBox.State(string)
```

Arguments

The State method has this argument:

string	[Required] Any valid string expression, containing the list of items to select

SetCheck

This method sets the check state of the check list box.

Usage

```
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.

ListCount Property

This method returns the number of items in the check list box.

Usage

```
CheckListBox.ListCount
```

SelCount Property

This method returns the number of selected items in the check list box.

Usage

```
CheckListBox.SelCount
```

Selected Property

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.

Usage

```
CheckListBox.Selected(index)[=boolean]
```

Check Property

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.

Usage

```
CheckListBox.Check(index)[=boolean]
```

Text Property

This method returns the text of the item currently selected in the check list box.

Usage

```
CheckListBox.Text
```

ComboBox

This control represents a combo box on a dialog box.

Select

This method sets the selection state of the combo box.

Usage

```
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")
```

Edit

This method sets the edit state of the combo box.

Usage

```
ComboBox.Edit(string)
```

The Edit method has this argument:

string	[Required] A string expression, containing a string to insert into the edit box of the combo box

Text Property

This method returns or sets the combo box text.

Usage

```
ComboBox.Text[=string]
```

Arguments

The Text method has this argument:

string	Required A string expression, containing the text to set into the combo box

List Property

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.

Usage

```
ComboBox.List(index)
```

SelStart Property

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.

Usage

```
ComboBox.SelStart[=index]
```

SelLength Property

This method returns or sets the number of characters selected.

Usage

```
ComboBox.SelLength[=number]
```

SelText Property

This method returns or sets the string containing the currently selected text. If no characters are selected, this method returns a zero length string ("").

Usage

```
ComboBox.SelText[=string]
```

Arguments

The argument is a string expression, containing the text to set into the edit box.

EditBox

This control represents an edit box on a dialog box. You can use the Control method of a dialog box to reference a particular edit box.

State

This method sets the state of the edit box.

Usage

```
EditBox.State(string)
```

Arguments

State has this argument:

string	[Required] A string expression of the text to display in the edit box

Text Property

This method returns or sets the edit box text.

Usage

```
EditBox.Text[=string]
```

The *string* argument is a string expression, containing the text to set into the edit box.

SelStart Property

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.

Usage

```
EditBox.SelStart[=index]
```

SelLength Property

This method returns or sets the number of characters selected.

Usage

```
EditBox.SelLength[=number]
```

SelText Property

This method returns or sets the string containing the currently selected text. If no characters are selected this method returns a zero length string ("").

Usage

```
EditBox.SelText[=string]
```

Arguments

The SelText method has this argument:

string	A string expression, containing the text to set into the edit box

GridControl

This control represents a grid control on a dialog box. You can use the control method to reference a particular grid control.

ListBox

This control represents a list box on a dialog box.

State

This method sets the selection state of the list box.

Usage

```
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.

List Property

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.

Usage

```
ListBox.List
```

ListCount Property

This method returns the number of items in the list box.

Usage

```
ListBox.ListCount
```

SelCount Property

This method returns the number of selected items in a list box.

Usage

```
ListBox.SelCount
```

Selected Property

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.

Usage

```
ListBox.Selected(index) [=boolean]
```

Text Property

This method returns the currently selected (focused) item's text in a list box.

Usage

```
ListBox.Text
```

PushButton

This control represents a push button (also called a command button) on a dialog box.

Click

This method emulates pressing a push button.

Usage

```
button.Click()
```

Examples

```
dlg.control("OK").click()
```

RadioBox

This control represents an option button on a dialog box.

State

This method checks the state of the option button.

Usage

```
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.

Value Property

This method returns or checks the state of the option button.

Usage

```
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

This control represents a slider control on a dialog box.

State

This method sets the state of the slider.

Usage

```
SliderControl.State(iState)
```

Arguments

State has this argument:

iState	[Required] A numeric expression

Value Property

This method returns or sets the current slider position.

Usage

```
Slider.Value[=val]
```

Arguments

Value Property has this argument:

val	[Required] A numeric expression

SpinButton

This control represents a spin button on a dialog box.

State

This method sets the state of the spin button.

Usage

```
SpinButton.State(iState)
```

Arguments

State has this argument:

iState	[Required] A numeric expression

TabControl

This control represents a tab on a dialog box.

State

This method sets the selection state of the tab.

Usage

```
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)
```

Value Property

This method returns or sets the current tab position.

Usage

```
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

TreeItem

This control represents a tree item on a dialog box.

Select

This method sets the selection state of the tree item.

Usage

```
TreeItem.Select(flag)
```

Arguments

The Select method has this argument:

flag	[Required] A numeric expression

flag may have one of the following values:

Table 33. TreeItem.Select flag Values

Value	Description
0, or False	Unselect
1, or True	Select

Examples

```
item.Select(true)
```

Expand

This method sets the expand state of the tree item.

Usage

```
TreeItem.Expand(flag)
```

Arguments

Expand has this argument:

flag	[Required] A numeric expression

flag may have one of the following values:

Table 34. TreeItem.Expand flag Values

Value	Description
0, or False	Collapse
1, or True	Expand

Examples

```
item.Expand(true)
```

Focus

This method sets the tree item focus to the item.

Usage

```
TreeItem.Focus()
```

Examples

```
item.Focus(1)
```

TreeView

This control represents a Tree View on a dialog box.

Item

This method returns a TreeItem object.

Usage

```
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")
```

BeginDrag

This method emulates dragging selected items off the tree.

Usage

```
TreeView.BeginDrag(itemname)
```

Arguments

BeginDrag has this argument:

itemname	[Required] A string expression representing the name of the item to drag

Copy

This method copies the selected item to the Clipboard.

Usage

```
TreeView.Copy(itemname)
```

Arguments

Copy has this argument:

itemname	[Required] A string expression representing the name of the item to copy

Drop

This method emulates dropping the dragged items onto an item.

Usage

```
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")
```

Paste

This method pastes the contents of the Clipboard into the selected branch.

Usage

```
TreeView.Paste(itemname)
```

Arguments

Paste has this argument:

itemname	[Required] A string expression representing the name of the item to paste

CreateNewItem

This method creates a new item in the selected branch. This method returns a TreeItem object that corresponds to the created item.

Usage

```
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 Internal Macro Objects of this program include the following:

Application Object

This object represents applications of the program.

The object has the following methods:

- CreateNewDocument
- ExecuteCommand
- Help
- HelpContents
- HelpPane
- OpenCustomizeDialog
- OpenDocument
- OpenOptionsDialog
- OpenPropertiesDialog
- Quit
- RunMacro

CreateNewDocument

This method creates an empty document.

Usage

```
Application.CreateNewDocument
```

Arguments

None

ExecuteCommand

This method executes one of the commands of the program.

Usage

```
Application.ExecuteCommand(command, [arg1,...])
```

Arguments

The ExecuteCommand method has these arguments:

command	[Required] A string expression representing a SailWind product command
arg1,...	[Optional] Represents optional arguments that are passed to *command*

Examples

```
Application.ExecuteCommand("ID_VIEW_BOARD")
```

And another example:

```
Application.ExecuteCommand("Open", "C:\SailWind Projects\preview.pcb")
```

Help

This method invokes the *Help*.

Usage

```
Application.Help()
```

Arguments

None

HelpContents

This method returns the *Help* contents in the Help Contents window.

Usage

```
Application.HelpContents
```

Arguments

None

Examples

```
Set var = Application.HelpContents
```

HelpPane

This method returns the Help window.

Usage

```
Application.HelpPane
```

Arguments

None

Examples

```
Set var = Application.HelpPane
```

OpenCustomizeDialog

This method opens the Customize modal dialog box.

Usage

```
Application.OpenCustomizeDialog()
```

Arguments

None

OpenDocument

This method opens an existing document identified by the path argument.

Usage

```
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

This method opens the Options modeless dialog box.

Usage

```
Application.OpenOptionsDialog()
```

Arguments

None

OpenPropertiesDialog

This method opens the Properties modeless dialog box.

Usage

```
Application.OpenPropertiesDialog()
```

Arguments

None

Quit

This method exits the application.

Usage

```
Application.Quit()
```

Arguments

None

RunMacro

This method executes one of the program commands.

Usage

```
Application.RunMacro(path[, function [, arg1, ...]])
```

Arguments

The RunMacro method has these arguments:

path	[Required] A string expression containing the file path of the macro to run
function	[Optional] Name of a function or a sub in the macro file to be called. If the *function* is specified, RunMacro returns what the function returns. If the *function* is not specified, or it is a sub, which returns nothing, RunMacro returns nothing.
arg1,...	[Optional] Arguments that are passed onto *function*

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:

Focus
Control
CloseHelpPane
OpenHelpPane
ShowHelpFor

Focus

This method sets focus to a dialog box control.

Usage

```
Dialog.Focus(controlname)
```

Arguments

The Focus method has this argument:

controlname	[Required] A string expression representing the name of the control

Related Topics

Control

Control

This method returns a dialog box control.

Usage

```
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

CloseHelpPane

This method closes the open help pane in a dialog box.

Usage

```
Dialog.CloseHelpPane
```

Arguments

None

Examples

```
Dialog.CloseHelpPane
```

OpenHelpPane

This method displays the help pane for the dialog box.

Usage

```
Dialog.OpenHelpPane
```

Arguments

None

Examples

```
Dialog.OpenHelpPane
```

ShowHelpFor

This method displays help for the specified control.

Usage

```
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

This method prints the document.

Usage

```
Document.Print()
```

Arguments

None

PrintSetup

This method opens the Print Setup dialog box.

Usage

```
Document.PrintSetup()
```

Arguments

None

RepeatLastAction

This method repeats the last action performed during the current session.

Usage

```
Document.RepeatLastAction()
```

Arguments

None

Save

This method saves the document, if it has been modified.

Usage

```
Document.Save()
```

Arguments

None

SaveAs

This method saves the document to a user-defined name or path location.

Usage

```
Document.SaveAs(path)
```

Arguments

The SaveAs method has this argument:

path	[Required] A string expression representing the path to which to save the document

HelpContents Object

The HelpContents object represents the *Help* Contents window.

Item

The Item property finds the location of the Help contents item in the contents tree.

Usage

```
Application.HelpContents.Item (path)
```

Arguments

None

Examples

```
Set item = Application.HelpContents.Item("File Operations\To Restore
 Files")
```

HelpContentsItem Object

This 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

This property returns the location of the item.

Usage

```
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

This property returns the name of the item.

Usage

```
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

This method selects the item.

Usage

```
Item.Select
```

Arguments

None

Examples

```
Set item = Application.HelpContents.Item("File Operations").SubItem(3)
```

```
item.Select
```

SubItem

This property 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.

Usage

```
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

This property returns the number of sub items within the item.

Usage

```
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

This 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

This method displays the Active Layer combo box.

Usage

```
MainView.ActiveLayer
```

Arguments

None

Examples

```
set layerCombo = MainView.ActiveLayer
```

ToggleFullScreen

This method turns the Full Screen mode on.

Usage

```
MainView.ToggleFullScreen()
```

Arguments

None

MouseDown

This method emulates pressing a mouse button.

Usage

```
MainView.MouseDown(x, y, button)
```

Arguments

The MouseDown method has these arguments:

x	[Required] Any numeric expression X-coordinate of pointer
y	[Required] Any numeric expression Y-coordinate of pointer
button	[Required] String expression Pressed button, if any

The *button* argument may contain one or more of the following values and modifiers:

Table 35. MainView.MouseDown button Values

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

This method emulates ending a mouse drag operation.

Usage

```
MainView.MouseEndDrag(x, y, button)
```

Arguments

The Mouse End Drag method has these arguments:

x	[Required] Any numeric expression X-coordinate of pointer
y	[Required] Any numeric expression Y-coordinate of pointer
button	[Required] String expression Pressed button, if any

The *button* argument may contain one or more of the following values and modifiers:

Table 36. MainView.MouseEndDrag button Values

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

This method emulates moving the mouse.

Usage

```
MainView.MouseMove(x, y, button)
```

Arguments

The Mouse Move method has these arguments:

x	[Required] Any numeric expression X-coordinate of pointer
y	[Required] Any numeric expression Y-coordinate of pointer
button	[Required] String expression + indicates relative mode Pressed button, if any

The *button* argument may contain one or more of the following values and modifiers:

Table 37. MainView.MouseMove button Values

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")
```

MouseStartDrag

This method emulates starting a mouse drag operation.

Usage

```
MainView.MouseStartDrag(x, y, button)
```

Arguments

The MouseStartDrag method has these arguments:

x	[Required] Any numeric expression X-coordinate of pointer
y	[Required] Any numeric expression Y-coordinate of pointer
button	[Required] String expression Pressed button, if any

The *button* argument may contain one or more of the following values and modifiers:

Table 38. MainView.MouseStartDrag button Values

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

This method emulates releasing a mouse button.

Usage

```
MainView.MouseUp(x, y, button)
```

Arguments

The MouseUp method has these arguments:

x	[Required] Any numeric expression X-coordinate of pointer
y	[Required] Any numeric expression Y-coordinate of pointer
button	[Required] String expression Pressed button, if any

Description

The *button* argument may contain one or more of the following values and modifiers:

Table 39. MainView.MouseUP button Values

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

This method prints the current view.

Usage

```
MainView.Print()
```

Arguments

None

PrintPreview

This method turns the Print Preview mode on.

Usage

```
MainView.PrintPreview()
```

Arguments

None

ObjectView Object

The ObjectView object only applies to SailWind Router and represents the view(s) of the Project Explorer. Since the Project Explorer can have a split view, the object is either ObjectView1 or ObjectView2. If there is no split view, the object is labeled ObjectView1. This object uses the following methods:

- AllowSelection
- Copy
- CreateNewItem
- DeleteSelected
- Drop
- Item.Expand
- Item.Select
- Paste
- Select
- SortByRules
- ZoomToSelection

AllowSelection

This method toggles the Allow Selection feature which allows the item(s) selected in the Project Explorer to also be selected in the design.

Usage

```
ObjectView1.AllowSelection()
```

Arguments

None

Copy

This method is used to copy the item(s) selected in the Project Explorer.

Usage

```
ObjectView1.Copy()
```

Arguments

None

CreateNewItem

This method is used to create new item(s) of the selected object in the Project Explorer. Applies to Net Classes, Matched Length Net Groups, Matched Length Pin Pair Groups, and Pin Pair Groups.

Usage

```
ObjectView1.CreateNewItem()
```

Arguments

None

DeleteSelected

This method is used to delete item(s) of the selected object in the Project Explorer. Applies to Net Classes, Matched Length Net Groups, Matched Length Pin Pair Groups, Pin Pair Groups, Conditional Rules, and Differential Pairs objects.

Usage

```
ObjectView1.DeleteSelected()
```

Arguments

None

Drop

This method is used to (drag and) drop the selected Project Explorer item into a valid object. Applies to Net Classes, Matched Length Net Groups, Matched Length Pin Pair Groups, Pin Pair Groups, Conditional Rules, and Differential Pairs objects.

Usage

```
ObjectView1.Drop()
```

Arguments

None

Examples

```
ObjectView1.Drop("Net Objects\Matched Length Net Groups")
```

Item.Expand

This method is used to expand or collapse the tree/branch of a Project Explorer object.

Usage

```
ObjectView1.Item().Expand()
```

Arguments

None

Examples

```
ObjectView1.Item("Net Objects\Nets").Expand(True)
```

Item.Select

This method is used to add to, or remove items from a selection in the Project Explorer. It follows an ObjectView1.Select() when additional items are added to or removed from the selection using Ctrl+click.

Usage

```
ObjectView1.Item().Select()
```

Arguments

None

Examples

```
ObjectView1.Item("Net Objects\Nets\5-2").Select(True)
```

Paste

This method is used to paste item(s) that are copied in the Project Explorer.

Usage

```
ObjectView1.Paste()
```

Arguments

None

Select

This method is used to select items in the Project Explorer.

Usage

```
ObjectView1.Select()
```

Arguments

None

Examples

The following example selects the net A04 in the Nets branch of the Net Objects tree.

```
ObjectView1.Select("Net Objects\Nets\A04")
```

The following example selects the nets A04 through A12 in the Nets branch of the Net Objects tree and is similar to a Shift+click.

```
ObjectView1.Select("Net Objects\Nets\A04", "Net Objects\Nets\A12")
```

SortByRules

This method toggles the Sort By Rules feature of the Project Explorer where objects with custom rules are listed at the top of every branch.

Usage

```
ObjectView1.SortByRules()
```

Arguments

None

ZoomToSelection

This method toggles the Zoom To Selection feature which zooms into the component in the design that matches the selected component(s) in the Project Explorer.

Usage

```
ObjectView1.ZoomToSelection()
```

Arguments

None