PAIZI

SailWind Logic 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
Logic Automation Server

This section describes the Automation Server features contained in SailWind Logic.

OLE Background

“The fundamental question OLE addresses: How can a system be designed such that binary components from different vendors - written in different parts of the world at different times and using different programming languages - are guaranteed to interoperate?”

Dr. Dobbs’s Journal, January 1995.

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

Automation is a feature of OLE that defines a protocol for applications to share their data and functionality with any other application using that same protocol. Automation involves a server and a client. A server is the entity that makes available some data and functionality. A client, also called the controller, is the entity that uses the server's data and functionality.

Different people at different organizations can author servers and clients at different times, using different development tools and languages. The only required commonality is that they use the same Automation communication protocol. SailWind Logic is an Automation server because it makes some data (its database) and some functionality (such as opening design files and selecting objects) available to other applications. A client, such as Excel, can access SailWind Logic's data and functionality using the Automation protocol.

The Automation implementation in SailWind Logic allows third party companies and users to:

- Integrate their products with SailWind Logic.
- Expand SailWind Logic's set of functionality.
- Customize existing SailWind Logic features.
- Automate SailWind Logic tasks using standard scripting languages (such as Basic).

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

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.

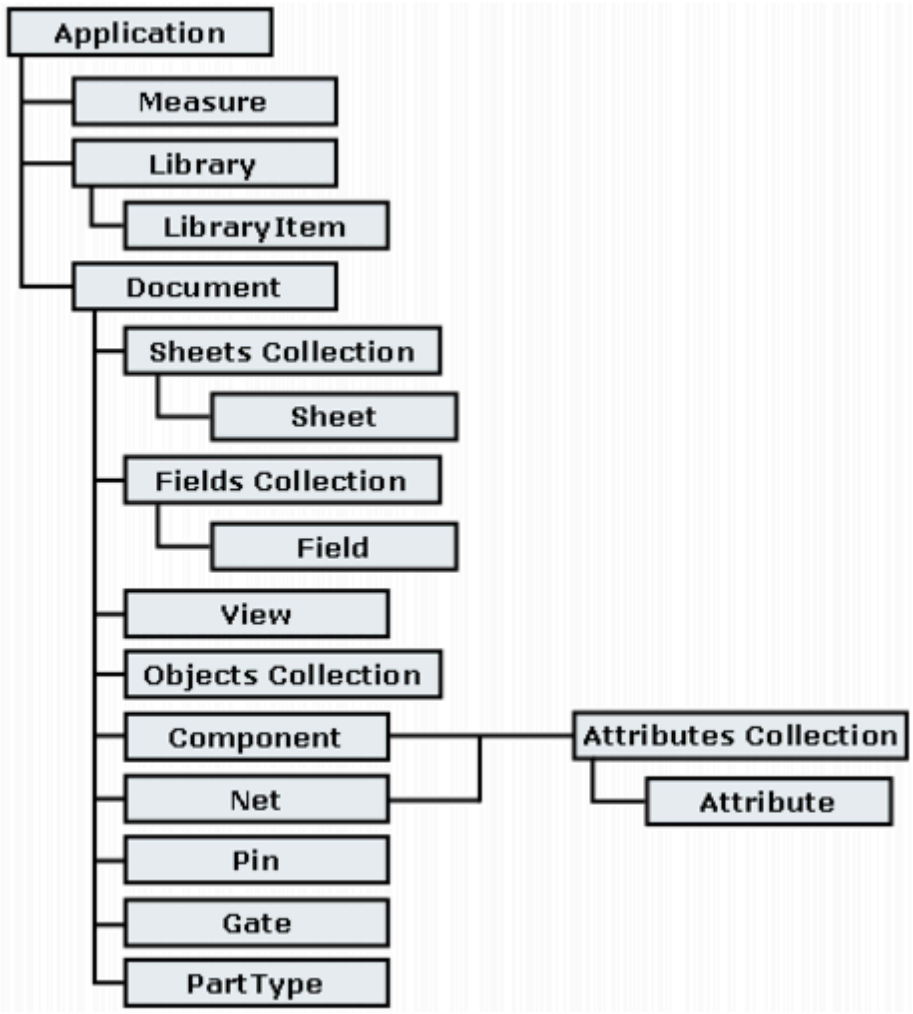
- COM automation scripts — These scripts extend the functionality of the main authoring applications (for example, SailWind Layout). These scripts connect to an instance of a tool to access its functionality. That functionality is contained in COM objects within the applications. It is these COM objects 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

The following figure shows the Object Hierarchy.

Figure 1. Object Hierarchy Diagram



Related Topics

Automation Server Reference

Automation Server Sample

The general sample in this topic is provided to make it easy for you to develop Automation clients designed for SailWind Logic, or to integrate their existing applications with SailWind Logic.

This sample demonstrates the following techniques:

- Basics of creating an Automation client for the SailWind Logic server. How to connect to an existing (or a new) instance of SailWind Logic, how to disconnect from it, and how to access simple data from SailWind Logic.
- Adding events, which notify clients when the state of SailWind Logic changes.
- Access to SailWind Logic database.
- Full-duplex cross-probing feature between the client and SailWind Logic.

Tips:

- You need to install the proper development tool on your system to properly run these samples. In other words, to run the Microsoft Excel version of any of the sample, install Excel on your system. To run the Visual Basic version, install Visual Basic on your system. To run the Visual C++ version, install Visual C++ on your system.
- If you encounter difficulty with one of the samples or with your own client application, see the following topic, which provides hints on common Automation problems.

Troubleshooting
SailWind Logic Automation Client Sample

Troubleshooting

If a sample doesn't work as described, here are some things to check.

- Make sure that only one instance of SailWind Logic is running. By definition a server is unique. Several instances of the same server may create confusion for clients.
- Exiting and restarting SailWind Logic may help in some cases because SailWind Logic self-registers as an Automation server upon start up. Make sure that you don't have another instance of SailWind Logic running in the background, using the Task Manager.
- Make sure to disconnect all clients from SailWind Logic before exiting the client application.
- You may need to reboot the machine if the above doesn't help.

SailWind Logic Automation Client Sample

This topic discusses the Automation sample in detail.

Goal

To introduce the basics of writing a SailWind Logic Automation client application using Microsoft Visual C ++ 5.0 (with MFC), Microsoft Visual Basic 5.0, and Microsoft Excel 97. One sample is provided for each of these applications.

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

- Prepare the communication between your client and the SailWind Logic Automation Server.
- Prepare a pointer to SailWind Logic's top level Automation object.
- Connect and disconnect from SailWind Logic, regardless of whether SailWind Logic is running or not.
- Access simple data from SailWind Logic.
- Add client notifications, also called client callbacks, to notify the client when something changes in SailWind Logic.
- Access the SailWind Logic database components and retrieve detailed information about each component.
- Add full-duplex cross-probing and demonstrate the server locking mechanism to improve Automation performance.

Tips

- The source code for this sample is in the C:\SailWind Projects\Samples\Scripts\Logic \AutomationClient folder, if you installed the OLE samples during SailWind Logic installation (Typical). The code is well commented, and you may use it when developing your own client application.
- Make sure you copy the OLE sample from the distribution CD to your hard drive before running the sample.
- To run the sample client, start SailWind Logic and open a Schematic file.

Specification

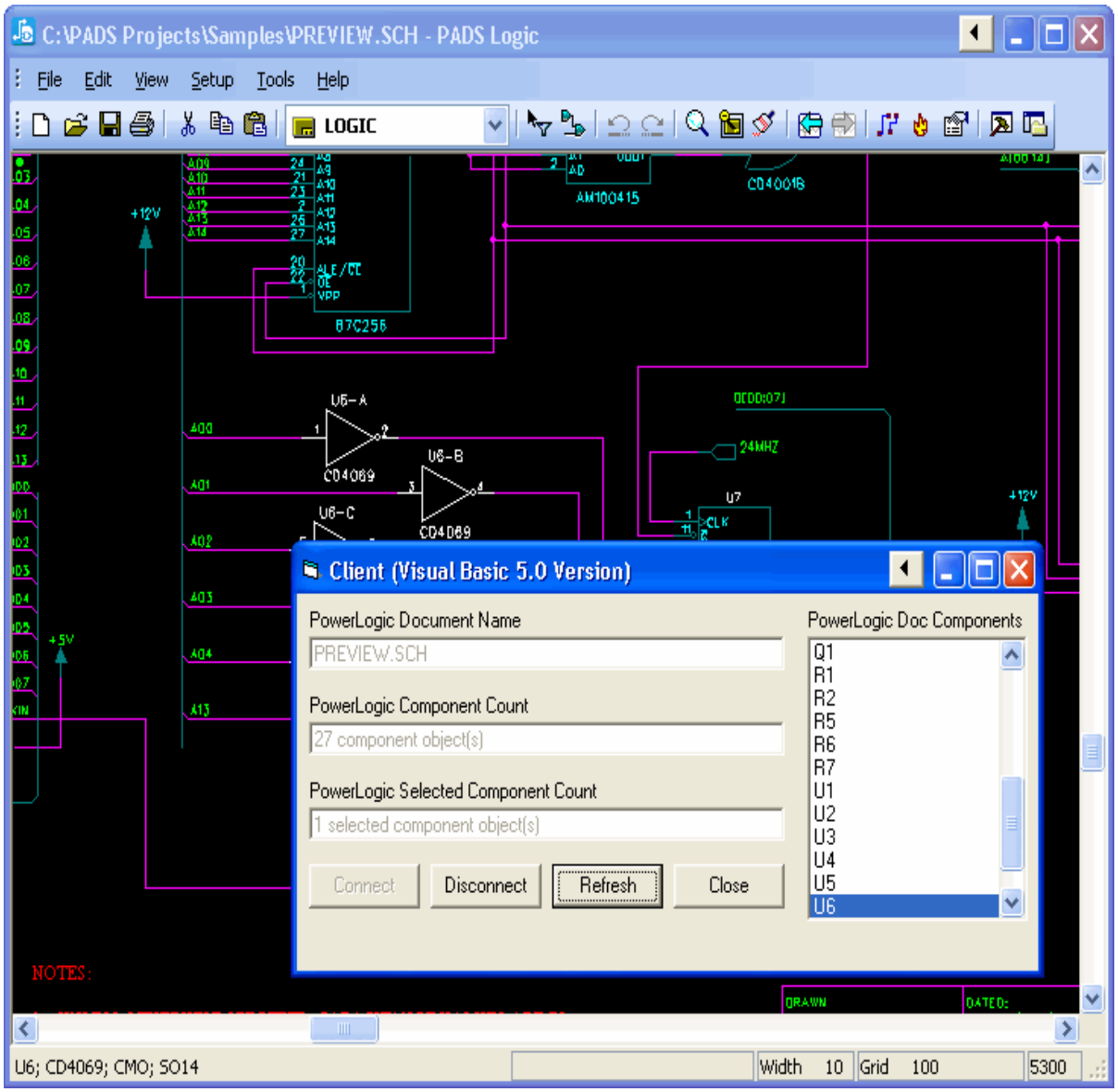
The client sample is a dialog box that can connect and disconnect at runtime from the SailWind Logic server. It can connect to an existing, running instance or a new instance of SailWind Logic.

An example of the Visual Basic version appears in Figure 2. The Visual C++ version of this sample looks almost identical. The Excel 97 version is different because Excel is primarily an application, not a development environment. The Excel version of this sample outputs information in an Excel worksheet.

The client sample allows selection cross-probing between the client and the server; the list of selected objects in SailWind Logic matches the list of selected items in the Client list box at all times, and vice-versa. The selection cross-probing supports multiple selection. If you double-click an item in the list box, extended information about the object appears in a dialog box.

The client refreshes its values automatically, matching the selected components in SailWind Logic at all times. Similarly, a user selection change in the client changes the SailWind Logic selection list.

Figure 2. Client Sample of Visual Basic



Code Samples

Many topics in this Help file provide Basic code samples to illustrate how to use Automation properties, methods, or events.

You can run code samples as described in the following topics.

The code sample is always in the following form:

```
Sub Main
```

```
' Do something
```

```
End Sub
```

Disclaimer:

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

Running Code Sample in SailWind Logic

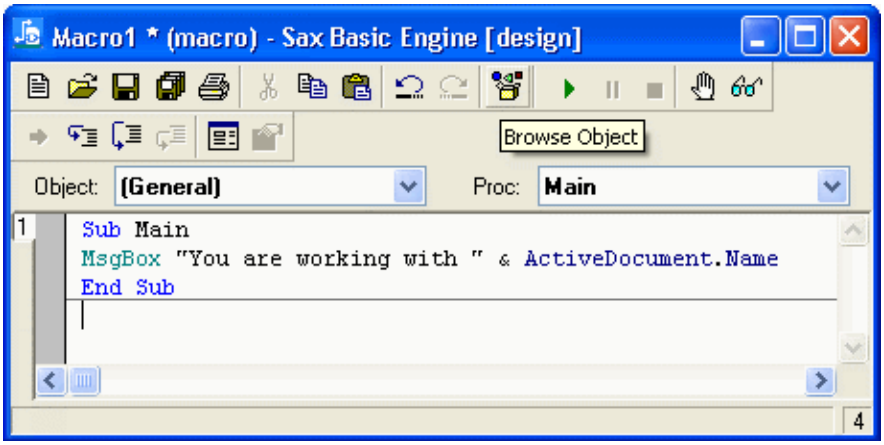
Run a sample using the Sax Basic Engine.

Procedure

1. Select the sample, including the Sub Main and End Sub statements.
2. Copy it to the Clipboard using Edit menu > Copy.
3. Choose Tools menu > Basic Scripts > Basic Script Editor in SailWind Logic. The Sax Basic Engine dialog box appears.
4. Paste the code sample into the Basic Editor using Edit menu > Paste.
5. Choose the Start icon from the toolbar in the Sax Basic Engine dialog box to run the sample.

6. The following graphic shows the code sample for the Application.ActiveDocument Property property pasted into the Sax Basic Engine dialog box.

Figure 3. Code Sample Pasted into Sax Basic Engine Dialog Box



Running Code Samples Outside of SailWind Logic

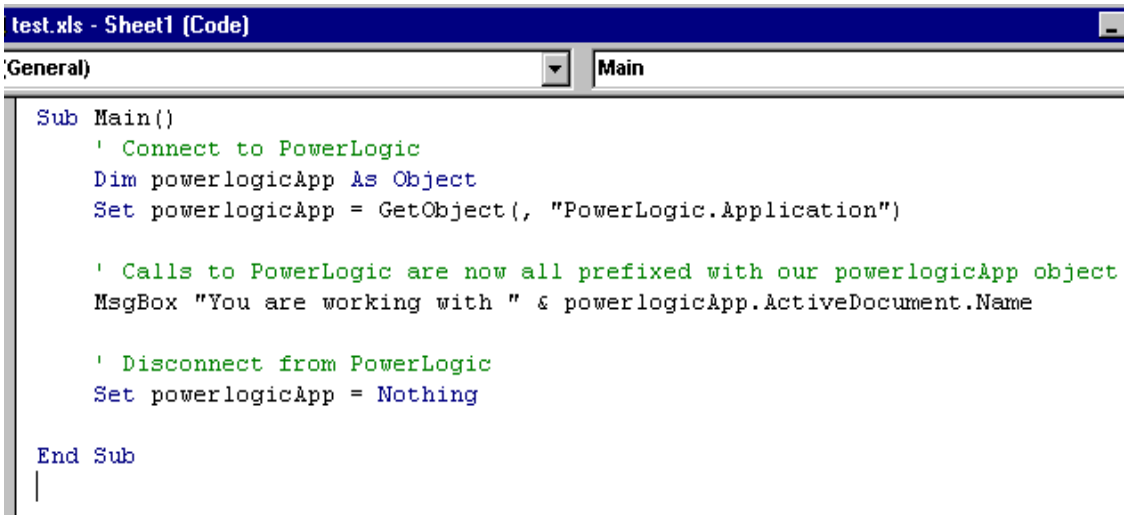
Use the code sample with other Basic scriptable applications (also called host applications), such as Visual Basic, Microsoft Excel, and Microsoft Word.

Procedure

1. Import the SailWind Logic Automation server references into the host application. For example to use Excel as the host application:
   a. Choose Tools/References in Excel. The References dialog box appears.
   b. In the list box, turn on the SailWind Logic Type Library check box.
   c. Choose OK.
2. Paste the code sample into the Basic Editor of the host application. To do this in Excel:
   a. Choose View/Code.
   b. Choose Edit/Paste.
3. Add code to the beginning of the code sample to connect the host application to SailWind Logic using the Basic GetObject function.
4. Add the object returned by the Basic GetObject function to the beginning of each SailWind Logic Automation method or property in the code sample.
5. Add code to the end of the sample to disconnect the application from SailWind Logic.

6. The following graphic shows the code sample for the Application.ActiveDocument Property property pasted and modified in the Microsoft Excel Basic editor. See the host application help files for more information.

Figure 4. Code Sample in Microsoft Excel 97 Basic Editor



Enhancing Sample Code

The code samples in the SailWind Logic Automation Server Help are reduced to the minimum number of lines necessary to quickly illustrate how to use SailWind Logic Automation properties and methods. You can use these code samples as a base for developing your own SailWind Logic features.

The following list provides ideas for possible enhancements:

- Add strict type checking, using the Option Explicit declaration at the beginning of the code, and declare all variables with the Dim Basic keyword. This ensures that SailWind Logic can interpret your variables properly and can generate compiling errors where a potential problem may exist.
- Add error checking using the Basic On Error keyword. Error checking improves how your code reacts to run-time errors.
- Output information in custom dialog boxes using the Basic UserDialog keyword, instead of outputting into simple dialog boxes using the Basic MsgBox keyword.

Disclaimer for Code Samples

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

Troubleshooting the Code Samples

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

- Make sure a design is open in SailWind Logic.

  Almost all code samples require a schematic file to be open at the time the sample is run.

- Check for any assumptions the samples make about the design files.

  For example, some code samples may assume that component U1 exists. These assumptions are clearly stated in the text preceding each sample. If these assumptions are not true, the sample code will not run properly. You can adapt the sample to your schematics.

- If you modified the sample code to run outside of SailWind Logic, make sure that you properly applied all required changes to the code, as described in the topic “Code Samples” on page 21.

Common mistakes include forgetting to import SailWind Logic references and forgetting to prefix SailWind Logic Automation constants.

Disclaimer

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

Chapter 2
Automation Server Reference

The SailWind Logic Automation object hierarchy strictly follows Microsoft standards.

The root-level object of the Object Hierarchy, the Application on page 401 object, including the Measure on page 195, Library on page 185, and LibraryItem on page 177 objects, identifies the SailWind Logic application and provides a way for Automation clients to bind to and to navigate the application's exposed objects, methods and properties. The Measure object The Document on page 419 object handles all document-centered operations, the Sheet on page 265 object handles all sheet-centered operations, and the View on page 294 object handles all view-centered operations.

All SailWind Logic data objects, such as the Component on page 77, Net on page 208, Pin on page 246, PartType on page 234, and Gate on page 155 objects, are accessed either directly through the Document on page 419 object or through the Objects on page 219 collection object. The Objects on page 219 collection object provides a convenient way of working with a set of SailWind Logic data objects rather than on a per object basis. The Attributes Collection on page 59 and Attribute on page 69 objects are accessed through either the Component or Net objects.

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

The following list identifies the Automation objects:

- Application Object
- Attributes Collection Object Property
- Attribute Object
- Component Object
- Document Object
- Field Object
- Fields Collection Object
- Gate Object
- Library Object
- LibraryItem Object
- Measure Object
- Net Object
- Objects Collection Object
- PartType Object
- Pin Object
- Sheets Collection Object
- Sheet Object
- View Object

The following list identifies the Automation Constants:

- PlogObjectType
- PlogUnit
- PlogGridType
- PlogPinElectricalType
- PlogASCIIVersion
- PlogNetListVersion
- PlogGateVisibility
- PlogDefaultPosition
- PlogMeasureFormat
- PlogLibraryItemType

Application Object

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

Application.ActiveDocument Property

This property returns the active SailWind Logic document.

Usage

```
ActiveDocument As Document on page 419
```

Arguments

None

Description

The active document represents the open schematic.

Examples

The following sample code retrieves the name of the open schematic using the Document.Name Property property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
End Sub
```

Related Topics

Application.OpenDocument Event

Document.Name Property

Application.Application Property

This property returns the SailWind Logic Application object.

Usage

```
Application As Application on page 401
```

Arguments

None

Description

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

Application.DefaultFilePath Property

This property sets or returns the path that SailWind Logic use to open schematic files.

Usage

```
DefaultFilePath As String
```

Arguments

None

Description

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

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

Examples

The following sample code changes the SailWind Logic default file path and notifies the client of this change. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main
```

```
oldPath = DefaultFilePath
```

```
DefaultFilePath="C:\TEMP"
```

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

```
End Sub
```

Application.FullName Property

This property returns the filename of the SailWind Logic application, including its path.

Usage

```
FullName As String
```

Arguments

None

Description

For example, this function can return the string “C:\<install_folder>\<*version*>\Programs \SailWindLogic.exe”.

Examples

The following sample code displays the common name and the actual .exe name of SailWind Logic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
End Sub
```

Related Topics

Application.Name Property

Application.Version Property

Application.Libraries Property

This property returns the collection of available libraries, or a specific library.

Usage

```
Libraries as Collection
```

```
Libraries(name as String) as Library Object
```

Arguments

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

Description

None

Examples

This sample displays the number of available libraries. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
End Sub
```

Application.Name Property

This property returns the name of the SailWind Logic application.

Usage

```
Name As String
```

Arguments

None

Description

For example, in SailWind Logic this property returns the string “PowerLogic”.

This property is the default property for the Application object.

Examples

The following sample code displays the common name, the version, and the actual .exe name for SailWind Logic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
End Sub
```

Related Topics

Application.FullName Property

Application.Version Property

Application.ObjectType Property

This property returns the type of this object.

Usage

```
ObjectType As PlogObjectType
```

Arguments

None

Description

None

Examples

The following sample tests the ObjectType property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set items = GetLibraryItems()

For Each item In items

If item.ObjectType <> plogObjectTypeLibraryItem Then

MsgBox "Test failed"

End If

Next item

End Sub
```

Application.Parent Property

This property returns the parent of the object.

Usage

```
Parent As Application.Application Property
```

Arguments

None

Description

None

Application.ProgressBar Property

This property sets or returns current value of the Progress Bar, in percentages.

Usage

```
ProgressBar As Integer
```

Arguments

None

Description

This property allows you to retrieve current progress of long batch process running in SailWind Logic, or to show progress of long Basic scripts.

Set the value greater than 0 or less than 100 to deactivate the Progress Bar. Use this property with Application.StatusBarText Property.

Examples

The following sample demonstrates this property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

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

 For i = 0 to 100

  ProgressBar = i

 Next

 ProgressBar = -1 'deactivate progress bar

 StatusBarText="" 'hide progress text

End Sub
```

Related Topics

Application.StatusBarText Property

Application.ProgressChange Event

Application.StatusBarText Property

This property sets or returns the text displayed on SailWind Logic Status Bar.

Usage

```
StatusBarText As String
```

Arguments

None

Description

To set the SailWind Logic Status Bar text to nothing, set this property to an empty string (“ ”).

Examples

The following sample displays a message on SailWind Logic Status Bar. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
End Sub
```

Related Topics

Application.ProgressBar Property

Application.ProgressChange Event

Application.Version Property

This property returns the SailWind Logic version.

Usage

```
Version As String
```

Arguments

None

Description

This property returns the application version as a string with the following format: <major>. <minor>, for example “3.0”.

Examples

The following sample displays application name and version. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
End Sub
```

Related Topics

Application.Name Property

Application.FullName Property

Application.Visible Property

This property sets or returns whether SailWind Logic is visible.

Usage

```
Visible As Boolean
```

Arguments

None

Description

This property is usually used in the following cases:

- When an Automation client starts the SailWind Logic 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, therefore, use this property to make SailWind Logic visible if needed.
- When a client attempts to shut down SailWind Logic (see Application.Quit Method), by making SailWind Layout invisible, disconnecting from it, and letting the server shut down appropriately.
- When a client needs to make the server window the active window, making it appear on top of other application windows.

Examples

The following sample makes SailWind Logic invisible, waits a second, and then makes it visible again. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
Visible = False
```

```
Wait 1
```

```
Visible = True
```

```
End Sub
```

Application.CreateLibrary Method

This method returns a specific library or a collection of available libraries.

Usage

```
Libraries as Collection
```

```
Libraries(Name as String) as Library
```

Arguments

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

Description

None

Examples

The following sample displays the number of available libraries. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
End Sub
```

Application.ExportLibraryItems Method

This method generates a PADS-format ASCII file from the library items.

Usage

```
ExportLibraryItems (filename as String, items as Collection)
```

Arguments

	Argument	Description
	filename	Name of the file to which to export the library item. Do not specify an extension.
	items	Optional on page 314. Collection of items to export. If omitted, all items from the collection are exported.

Description

Up to four files can be generated.

Examples

This sample subroutine exports library items beginning with “R” from a library to a specified file. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set coll = GetLibraryItems(, "R*")

ExportLibraryItems("C:\sample", coll)

End Sub
```

Application.GetConfigParamInt Method

This method retrieves the integer value from the specified parameter and section in the SailWindlogic.ini file.

Usage

```
GetConfigParamInt(
```

```
sectionName as String,
```

```
paramName as String,
```

```
defaultValue as Integer) as Integer
```

Arguments

	Argument	Description
	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

The parameter value or default value.

Examples

```
MsgBox Application.GetConfigParamInt("general", " Object Selector Filter",
 0)
```

Application.GetConfigParamString Method

This method retrieves the string from the specified parameter and section in the SailWindlogic.ini file.

Usage

```
GetConfigParamString(
```

```
sectionName as String,
```

```
paramName as String,
```

```
defaultValue as String) as String
```

Arguments

	Argument	Description
	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

The parameter value or default value.

Examples

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

Application.GetLibraryItems Method

This method returns the collection of library items in all available libraries, a collection of all items of a given type, or a specific item.

Usage

```
GetLibraryItems (type as PlogLibraryItemType, name as String)
 as LibraryItem Object
```

Arguments

	Argument	Description
	type	Type of item(s) to retrieve. Optional; default value is plogLibraryItemTypeAll.
	name	Name of the item to retrieve. Optional on page 314; may include wildcards, ranges, lists.

Description

None

Examples

This sample displays the number of available library items. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main

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

End Sub
```

Application.LockServer Method

This method locks the SailWind Logic Automation server.

Usage

```
LockServer()
```

Arguments

None

Description

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

CAUTION:
Never forget to unlock a locked server.

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

You typically lock the server when your client needs to make more than hundreds of consecutive calls to the server. Do not keep the server locked for too long (not more than a few minutes). It is not necessary to lock the server when your client needs to make just a few OLE calls; speed does not greatly improve in this case.

Examples

The following sample code shows how to use this method. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

LockServer

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

UnlockServer

End Sub
```

Related Topics

Application.UnlockServer Method

Application.Measure Method

This method creates and returns measure value object.

Usage

```
Measure(Value As Variant, [DefaultUnit As String=""]) As Measure on
 page 195
```

Arguments

	Argument	Description
	Value	String or number representing a measure value including optional prefix and optional physical unit.
	DefaultUnit	[Optional on page 314] String that contains default prefix and/or physical unit.

Description

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

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

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

If the parser cannot recognize measure in the Value parameter, it creates a dummy Measure Object with the Measure.Value Property property equal to 0.0 and the Measure.Text Property property equal to the *Value* parameter.

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

Examples

The following sample adds a measure type attribute to part “C1” assuming that it exists in open design. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
C1.Attributes.Add "Capacitance", Measure("500pF")
```

```
MsgBox C1.Attributes("Capacitance") 'displays 500pF
```

```
End Sub
```

Related Topics

Attribute.Measure Method

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 the active sheet to the second sheet. See "Code Samples" for more information on running this sample.

```
Sub Main
```

```
ModelessCmd("sh 2")
```

```
End Sub
```

Application.OpenDocument Method

This method opens a SailWind Logic schematic file.

Usage

```
OpenDocument(filename As String) As Document on page 419
```

Arguments

	Argument	Description
	filename	Name of the file to open.

Return Values

If the function succeeds, the return value is the newly opened Document Object.

If the function fails, the return value is Nothing.

Description

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

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

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

This method does not check if the currently opened file was saved or not. It is the client's responsibility to use the Document.Saved Property property to check if the open schematic needs to be saved.

This method generates an Exception if a SailWind Logic failure has occurred while processing it.

Examples

The following sample code opens *DEMO.SCH*, assuming that it exists in the folder specified in the Application.DefaultFilePath Property property. The sample then displays the name of the opened file. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
OpenDocument(DefaultFilePath & "\Samples\DEMO.SCH")
```

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

```
End Sub
```

Related Topics

Application.ActiveDocument Property

Application.DefaultFilePath Property

Document.Name Property

Document.Saved Property

Application.OpenDocumentNoLock Method

This method opens a SailWind Logic schematic file without file locking.

Usage

```
OpenDocumentNoLock(filename As String) As Document on page 419
```

Arguments

	Argument	Description
	filename	Name of the file to open.

Return Values

If the function succeeds, the return value is the newly opened Document.

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

Application.OpenTempDocument Method

This method works in the same way as OpenDocumentNoLock but additionally doesn't add a filename to the MRU (Most Recently Used) list. This method is mainly intended for macro tests.

Usage

```
OpenTempDocument(filename As String) As Document on page 419
```

Arguments

	Argument	Description
	filename	Name of the file to open.

Return Values

If the function succeeds, the return value is the newly opened Document.

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

Application.Quit Method

This method shuts down SailWind Logic.

Usage

```
Quit ()
```

Arguments

None

Description

CAUTION:
Do not use this method.

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

- A server cannot shut down until all clients have disconnected 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 therefore 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 SailWind Logic shutdown, an Automation client needs to make SailWind Logic invisible using the Application.Visible Property property, and then disconnect from it. If no other clients are connected to SailWind Logic at that time, SailWind Logic automatically shuts down. If an Automation client is a Basic script running in the Sax Basic Engine, it can never successfully shut down the SailWind Logic Automation server.

Related Topics

Application.Quit Event

Application.RunMacro Method

This method runs a SailWind Logic macro.

Usage

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

Arguments

	Argument	Description
	filename	Name of the macro file to use.
	macroname	Name of the macro to run.

Description

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

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

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

This method generates an Exception if a SailWind Logic failure has occurred while processing it.

Examples

The following sample code runs the SailWind Logic macro MACRO1, recorded in the macro file *MACROS.MCR*, assuming that the macro exists in the folder specified by the Application.DefaultFilePath Property property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
End Sub
```

Application.UnlockServer Method

This method unlocks the SailWind Logic Automation server.

Usage

```
UnlockServer()
```

Arguments

None

Return Values

None

Examples

The following sample code shows how to use this method. See Code Samples for more information on running this sample.

```
Sub Main
```

```
LockServer
```

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

```
UnlockServer
```

```
End Sub
```

Related Topics

Application.LockServer Method

Application.OpenDocument Event

This event occurs after the program opens a new document.

Usage

```
Application_OpenDocument (Doc As Document on page 419)
```

Arguments

	Argument	Description
	Doc	Document object that was opened.

Description

This event occurs after SailWind Logic opens a new document.

Related Topics

Application.OpenDocument Method

Application.ProgressChange Event

This event occurs after the status bar value is changed.

Usage

```
Application_ProgressChange
```

Arguments

None

Description

This event occurs after the Progress Bar value is changed.

Application.Quit Event

This event shuts down the program.

Usage

```
Application_Quit ()
```

Arguments

None

Description

This event occurs before SailWind Logic exits.

Attributes Collection Object Property

The Attributes collection object is a collection of attributes of a SailWind Logic Component Object.

The Component Object being the Component.Attributes Property property.

The following list identifies the Attributes Properties and Methods:

Attributes.Application Property
Attributes.Count Property
Attributes.Item Property
Attributes.Parent Property
Attributes.Add Method
Attributes.Delete Method

Attributes.Application Property

This property returns the SailWind Logic Application object.

Usage

Application As Application on page 401

Arguments

None

Description

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

Attributes.Count Property

This property returns the number of attributes.

Usage

Count As Long

Arguments

None

Description

None

Examples

The following sample code retrieves the number of attributes for the part with name U1. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
 MsgBox "Part U1 attribute count is " &
 ActiveDocument.Components("U1").Attributes.Count
```

```
End Sub
```

Attributes.Item Property

This property returns an attribute, given its index or its name.

Usage

Item(*index* As Long) As Attribute on page 69

Item(*name* As String) As Attribute on page 69

Arguments

	Argument	Description
	index	Index (in the collection) of the attribute to retrieve.
	name	Name of the attribute to retrieve.

Description

This is the default member of the Attributes collection object.

This property generates an Exception if the *index* or *name* argument is not valid.

Examples

The following samples show two different methods for iterating through all attributes of a Component object in the open schematic. The second method is generally preferred because it is cleaner and faster. See “Code Samples” on page 21 for more information on running this sample.

```
' Method 1: Use the Object.Item property

Sub Main

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

For I=1 To attrs.Count

Set thisAttr = attrs.Item(I)

' Do something with the attribute thisAttr

Next I

End Sub
```

```
' Method 2: Do not use the Object.Item property (preferred method)

Sub Main

For Each nextAttr in ActiveDocument.Components("U1").Attributes

' Do something with the attribute nextAttr

Next nextAttr

End Sub
```

Attributes.Parent Property

This property returns the parent of the object.

Usage

Parent As Document on page 419

Arguments

None

Description

None

Attributes.Add Method

This method adds a new attribute.

Usage

Add(*name* As String, [*value* As Variant]) As Attribute Objects

Arguments

	Argument	Description
	name	Name of the new attribute.
	value	[Optional on page 314] Value of the new attribute. The attribute may be of type Boolean, Byte, Single, Integer, PortInt, Long, Double, String, or Measure object.

Return Values

The new attribute packaged as Attribute Objects.

Description

SailWind Logic supports only string attributes value, so any other type is converted to String type.

This property generates an Exception if the name argument is already an existing attribute or if it is not a valid attribute name.

Examples

The following sample code adds several attributes of different types to component U1 and net +5V, assuming they exist in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

With ActiveDocument.Components("U1").Attributes

' Ignore exceptions generated when that attribute already exists

On Error Resume Next

.Add("COST", 12.99)

.Add("HEIGHT", 120)

.Add("COMMENT", "TO DO: Get spec sheet")
```

```
End With
```

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

```
End Sub
```

Related Topics

Attributes.Delete Method

Attribute.Measure Method

Application.Measure Method

Attributes.Delete Method

This method deletes an attribute.

Usage

Delete(*index* As Long)

Delete(*name* As String)

Arguments

	Argument	Description
	index	Index (in the collection) of the attribute to delete.
	name	Name of the attribute to delete.

Return Values

None

Description

This property generates an Exception in the following cases:

- If the name argument is not an existing attribute.
- If the name argument is not a valid attribute to delete.
- If the index argument is higher than the number of existing attributes.

Examples

The following sample code deletes the COST attribute of all components in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

For Each nextComp In ActiveDocument.Components

' Ignore exceptions generated when that attribute does not exist

On Error Resume Next

nextComp.Attributes.Delete("COST")
```

```
Next nextComp
```

```
End Sub
```

Related Topics

Attributes.Add Method

Attribute Object

The Attribute object represents an attribute of a SailWind Logic Component Object (part).

This object is usually retrieved from the Attributes Collection Object Property.

The following list identifies the Application Properties and Method:

- Attribute.Application Property
- Attribute.Name Property
- Attribute.Parent Property
- Attribute.Value Property
- Attribute.Measure Method

Attribute.Application Property

This property returns the SailWind Logic Application object.

Usage

Application As Application Object

Arguments

None

Description

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

Attribute.Name Property

This property returns the name of the attribute.

Usage

Name As String

Arguments

None

Description

None

Examples

The following sample code lists all attributes for component U1 and places that list in a custom dialog box. This sample uses the UserDialog Editor in the Sax Basic Engine in SailWind Logic. See Sax Basic Editor On-line Help for more information.

See “Code Samples” on page 21 for more information on running this sample.

```
Dim ListAttrs$(10000)

Sub Main

index = 0

For Each nextAttr In ActiveDocument.Components("U1").Attributes

ListAttrs$(index) = nextAttr.Name

index = index + 1

Next nextAttr

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

 Begin Dialog UserDialog 180,238,"U1 Attributes" ' %GRID:10,7,1,1

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

  OKButton 10,210,160,21
```

```
 End Dialog

Dim dlg As UserDialog

Dialog dlg

End Sub
```

Related Topics

Attribute.Value Property

Attribute.Parent Property

This property returns the parent of the object.

Usage

Parent As Document Object

Arguments

None

Description

SailWind Logic supports attributes for components and nets only; therefore, the parent can only be a component or a net.

Attribute.Value Property

This property sets or returns the value of the attribute.

Usage

Value As Variant

Arguments

None

Description

This is a default property.

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

This property generates an Exception if there is a type mismatch between the value type and the type of the attribute.

Examples

The following sample code changes the COST attribute of all LED components to US$3.99, in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

For Each nextComp In ActiveDocument.Components

If nextComp.PartType="LED" Then

' avoid exceptions generated when that attribute does not exist

if nextComp.Attributes("COST") is Nothing Then

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

   End If

End If

Next nextComp

End Sub
```

Related Topics

Attribute.Name Property

Attribute.Measure Method

This method creates and returns measure value object.

Usage

Measure As Application.Measure Method

Arguments

None

Description

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

If the internal parser cannot recognize measure in the attribute, it creates a dummy Application.Measure Method object with the Measure.Value Property property equal to 0.0 and the Measure.Text Property property equal to Attribute.Value Property property.

Examples

The following sample shows a quantity name of the capacitor's “Value” attribute (“Capacitance”) of part C1 assuming that it exists in open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set C1 = ActiveDocument.Components("C1")

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

End Sub
```

Related Topics

Application.Measure Method

Measure.Name Property

Component Object

The Component object represents a physical part that exists in the open schematic. The Component object consists of one or more visible or unused Gate objects.

The following buttons present lists of the properties and methods in the Component object.

The following list identifies the Component Properties and Method:

Component.Application Property

This property returns the SailWind Logic Application object.

Usage

Application As Application on page 401

Arguments

None

Description

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

Component.Attributes Property

This property returns the collection of all attributes of the component.

Usage

Attributes As Attributes

Attributes(*name* As String) As Attribute on page 69

Arguments

	Argument	Description
	name	Name of an existing component attribute.

Description

When an existing attribute name is passed to this property, it returns that component Attribute on page 69 object. Otherwise, it returns the collection of all component attributes in an Attributes Collection Object Property.

Examples

The following sample code retrieves the number of attributes for component U1, assuming it exists in the open schematic, using theAttributes.Count Property property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

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

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

End Sub
```

Component.Gates Property

This property returns object collection of used gates in this component.

Usage

Gates As Objects on page 219

Gates(*name* As String) As Gate on page 155

Arguments

	Argument	Description
	name	Name of an existing gate.

Description

The property returns only used gates of the part.

When an existing gate name is passed to this property, it returns that Gate Object. Otherwise, it returns the collection of all used gates in the component in an Objects Collection Object.

Examples

The following sample iterates through all used gates in the component U1. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 For Each nextGate In ActiveDocument.Components("U1").Gates

  'Do something with gate

 Next

End Sub
```

Component.Name Property

This property returns the name of the component.

Usage

Name As String

Arguments

None

Description

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

This property is the default property for the Component object.

Examples

The following sample code retrieves all components in the open schematic and places that list in a custom dialog list box. When a component is selected in the list box, the sample selects that component in SailWind Logic. This sample uses the UserDialog Editor in the Sax Basic Engine in SailWind Logic. See Sax Basic Editor On-line Help for more information.

See “Code Samples” on page 21 for more information on running this sample.

```
Dim ListComps$(10000)

Sub Main

index = 0

For Each nextComp In ActiveDocument.Components

ListComps$(index) = nextComp.Name

index = index + 1

Next nextComp

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

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

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

```
OKButton 10,210,160,21

End Dialog

Dim dlg As UserDialog

Dialog dlg

End Sub
```

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

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

Select Case Action%

Case 2 ' Value changing or button pressed

If DlgItem$="ListBox1" Then

ActiveDocument.SelectObjects(plogObjectTypeAll, , False)

'get part by name

Set comp = ActiveDocument.Components(ListComps(SuppValue%))

'select part

comp.Selected = True

'activate sheet where first gate of the part is located

comp.Gates(1).Sheet.Activate

End If

End Select

End Function
```

Component.ObjectType Property

This property returns the type of the object.

Usage

ObjectType As PlogObjectType

Arguments

None

Description

This property returns PlogObjectType.

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

This property is generally used:

- To identify the kind of SailWind Logic database objects in a heterogeneous Objects Collection Object.
- When implementing a generic routine that depends on the type of the SailWind Logic database object passed as an argument. For example:

```
Sub DoSomething(dbObject As Object)
```

```
Select Case dbObject.ObjectType
```

```
Case plogObjectTypeComponent
```

```
' Do something specific to component objects
```

```
Case plogObjectTypeNet
```

```
' Do something specific to net objects
```

```
Case plogObjectTypePin
```

```
' Do something specific to pin objects
```

```
Case plogObjectTypeGate
```

```
' Do something specific to gate objects
```

```
  Case Else
```

```
MsgBox "Not a SailWind Logic database object"
```

```
End Select
```

```
End Sub
```

Component.Parent Property

This property returns the parent of the object.

Usage

Parent As Document on page 419

Arguments

None

Description

None

Component.PartType Property

This property returns the part type of the component.

Usage

PartType As String

Arguments

None

Examples

The following sample code retrieves the part type of component U1, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

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

End Sub
```

Related Topics

Component.PartTypeObject Property

Component.PartTypeLogic Property

This property returns the logic family of the part type of the component.

Usage

PartTypeLogic As String

Arguments

None

Description

None

Examples

The following sample code retrieves the logic family of the part type for component U1, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
End Sub
```

Related Topics

PartType.Logic Property

Component.PartTypeObject Property

This property returns the part type object of this component.

Usage

PartTypeObject As PartType on page 234

Arguments

None

Description

None

Examples

The following sample code retrieves the part type object for component U1, assuming it exists in the open schematic. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main
```

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

```
End Sub
```

Related Topics

Component.PartType Property

Component.PCBDecal Property

This property returns the PCB decal of the component.

Usage

PCBDecal As String

Arguments

None

Description

None

Examples

The following sample code retrieves the current decal for component C1, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 MsgBox "C1 PCB decal is " & ActiveDocument.Components("C1").PCBDecal

End Sub
```

Component.Pins Property

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

Usage

Pins As Objects on page 219

Pins(*name* As String) As Pin on page 246

Arguments

	Argument	Description
	name	Name of an existing pin.

Description

Returns all pins of the component including:

- Visible used gate pins (pins of used gates).
- Invisible “signal” pins (do not belong to any gate, but are associated with some signal).
- Invisible unused gate pins (belong to unused gate).
- Invisible unused part pins (do not belong to any gate and are not associated with any signal).

When an existing pin *name* is passed to this property, it returns that Pin Object. Otherwise, it returns the collection of all pins of the component in an Objects Collection Object.

Examples

The following sample calculates the number of all types of pins in part U1.

```
Sub Main

 SigPins = 0

 AllPins = 0

 UsedGatePins = 0

 UnusedGatePins = 0

 UnusedPartPins = 0
```

```
UnconnectedPins = 0

Set U1 = ActiveDocument.Components("U1")

'get total pin count

AllPins = U1.Pins.Count

For Each p In U1.Pins

 If p.Gate Is Nothing Then   'if pin does not belong to any gate

  If p.Net Is Nothing Then  'if pin is unconnected then it is unused

   UnusedPartPins = UnusedPartPins + 1

  Else

   SigPins = SigPins + 1 'otherwise it is a signal pin

  End If

 Else

  If p.Gate.Sheet Is Nothing Then 'if pin belongs to gate but gate is not
installed

   UnusedGatePins = UnusedGatePins + 1

  Else

   UsedGatePins = UsedGatePins + 1

  End If

 End If

 'if pin is not connected

 If p.Net Is Nothing Then
```

```
    UnconnectedPins = UnconnectedPins + 1

   End If

  Next

  MsgBox "Total Pin Count is " & AllPins & vbCr & _

    "Signal Pin Count is " & SigPins & vbCr & _

    "Used Gate Pin Count is " & UsedGatePins & vbCr & _

    "Unused Gate Pin Count is " & UnusedGatePins & vbCr & _

    "Unused Part Pin Count is " & UnusedPartPins & vbCr & _

    vbCr & _

    "Unconnected Pin Count is " & UnconnectedPins, _

    "Pin Info of U1 part"

End Sub
```

Component.Selected Property

This property sets or returns whether the component is selected.

Usage

Selected As Boolean

Arguments

None

Description

The component is considered selected when one or more of its gates is selected. You can also select a SailWind Logic database object using the Document.SelectObjects Method or Objects.Select Method methods.

Examples

The following sample code selects component U1 only, assuming it exists in the open schematic, and activates the sheet on which it resides. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

ActiveDocument.SelectObjects(,,False)

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

 ActiveDocument.Components("U1").Gates(1).Sheet.Activate

End Sub
```

Related Topics

Document.SelectionChange Event

Component.UnusedGates Property

This property returns the object collection of all unused gates in this component.

Usage

UnusedGates As Objects on page 219

UnusedGates(*name* As String) As Gate on page 155

Arguments

	Argument	Description
	name	Name of an existing gate.

Description

The property returns only unused gates of the part.

When an existing gate name is passed to this property, it returns that Gate Object. Otherwise, it returns the collection of all unused gates of the component in an Objects Collection Object.

Examples

The following sample retrieves the total number of gates in component U1. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main

 Set u1 = ActiveDocument.Components("U1")

 MsgBox "Total gate count of part U1 is " & (u1.Gates.Count +
 u1.UnusedGates.Count)

End Sub
```

Component.Delete Method

This method deletes this part (all its gates) from the schematic.

Usage

Delete

Arguments

None

Description

None

Examples

The following sample deletes the part U1 from the schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 ActiveDocument.Components("U1").Delete

End Sub
```

Document Object

The Document object represents a schematic file that is currently open in SailWind Logic.

This object is usually retrieved using the Application.ActiveDocument Property property.

The following list identifies the Document Properties, Methods, and Events:

Document.ActiveSheet Property
Document.ActiveView Property
Document.AncestorSheets Property
Document.Application Property
Document.Components Property
Document.Fields Property
Document.FullName Property
Document.Gates Property
Document.GridX Property
Document.GridY Property
Document.Name Property
Document.Nets Property
Document.Parent Property
Document.PartTypes Property
Document.Path Property
Document.Pins Property
Document.Saved Property
Document.Sheets Property
Document.Activate Method
Document.ExportASCII Method
Document.ExportNetList Method
Document.GenerateECO Method
Document.GetColor Method
Document.GetObjects Method
Document.ImportASCII Method
Document.ImportECO Method
Document.IntegrityTest Method
Document.Save Method
Document.SaveAs Method
Document.SaveAsNoLock Method
Document.SaveAsTemp Method
Document.SaveNoLock Method
Document.SaveTemp Method
Document.SelectObjects Method
Document.SetColor Method
Document.Save Event

Document.ActiveSheet Property

This property returns the active sheet in this document.

Usage

ActiveSheet As Sheet on page 265

Arguments

None

Description

None

Examples

The following sample retrieves the name of active sheet. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 MsgBox "Active Sheet is " & ActiveDocument.ActiveSheet.Name

End Sub
```

Document.ActiveView Property

This property returns the active SailWind Logic view.

Usage

ActiveView As View on page 294

Arguments

None

Description

The active view represents the main view of the active sheet in the open schematic.

Examples

The following sample code pans the current SailWind Logic view to the origin of the active sheet (using the View.Pan Method method). See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
End Sub
```

Document.AncestorSheets Property

This property returns a collection of sheets in this document that are not child sheets of any other sheet.

Usage

AncestorSheets As Sheets on page 265

AncestorSheets(*name* As String) As Sheet on page 265

Arguments

- *name*

  Name of an existing sheet.

Description

When an existing sheet with *name* is passed to this property, this property returns that sheet packaged as a Sheet Object. Otherwise, this property returns the collection of root sheets existing in the open schematic, packaged as a Sheets Collection Object.

Examples

The following sample retrieves the total number of ancestor sheets in the active document. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
 MsgBox "Number of ancestor sheets is " &
 ActiveDocument.AncestorSheets.Count
```

```
End Sub
```

Document.Application Property

This property returns the SailWind Logic Application object.

Usage

Application As Application on page 401

Arguments

None

Description

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

Document.Components Property

This property returns the collection of all components.

Usage

Components As Objects on page 219

Components(*name* As String) As Component on page 77

Arguments

	Argument	Description
	name	Name of an existing component.

Description

When an existing component name is passed to this property, it returns that Component Object. Otherwise, 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 schematic using the Objects.Count Property property. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main

Set comps = ActiveDocument.Components

MsgBox "There are " & comps.Count & " part(s) in " & ActiveDocument.Name

End Sub
```

The following sample code retrieves the number of pins of component U1, assuming it exists in the open schematic, using the Component.Pins Property and Objects.Count Property properties. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main

Set compU1 = ActiveDocument.Components("U1")

MsgBox "Component " & compU1.Name & " has " & compU1.Pins.Count & "
 pin(s)."

End Sub
```

Related Topics

Document.GetObjects Method

Sheet.Components Property

Document.Fields Property

This property returns the collection of fields.

Usage

Fields As Objects on page 219

Arguments

None

Description

None

Examples

The following sample retrieves the number of fields in the document. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
MsgBox "The number of fields in this document is " &
 ActiveDocument.Fields.Count()
```

```
End Sub
```

Document.FullName Property

This property returns the filename of the document, including its path.

Usage

FullName As String

Arguments

None

Description

None

Examples

The following sample retrieves the name and location of the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
End Sub
```

Related Topics

Document.Name Property

Document.Path Property

Document.Gates Property

This property returns the collection of all gates.

Usage

Gates As Objects on page 219

Gates(*name* As String) As Gate on page 155

Arguments

	Argument	Description
	name	Name of an existing gate.

Description

When an existing gate name is passed to this property, it returns that Gate Object. If the gate name does not exist, this property returns the collection of all gates in an Objects Collection Object.

Examples

The following sample code retrieves the number of gates in the open schematic using the Objects.Count Property property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set gates = ActiveDocument.Gates

MsgBox "There are " & gates.Count & " gate(s) in " & ActiveDocument.Name

End Sub
```

Related Topics

Document.GetObjects Method

Sheet.GetObjects Method

Sheet.Gates Property

Document.GridX Property

This property sets or returns the X grid of the document.

Usage

GridX([*type* As PlogGridType = plogGridDesign], [*unit* As PlogUnit]) As Double

Arguments

	Argument	Description
	type	[Optional on page 314] Type of grid to set or return.
	unit	[Optional] Unit in which the grid value is set or returned.

Examples

The following sample code sets the display grid of the open schematic to the same value as the design grid. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

curGridX = ActiveDocument.GridX

curGridY = ActiveDocument.GridY

ActiveDocument.GridX(plogGridDisplay) = curGridX

ActiveDocument.GridY(plogGridDisplay) = curGridY

End Sub
```

Tip
You cannot assign different values for GridX and GridY. The property is not writable for GridY, so GridY takes on the value assigned to GridX.

Related Topics

Document.GridY Property

Document.GridY Property

This property sets or returns the Y grid of the document.

Usage

GridY([*type* As PlogGridType = plogGridDesign], [*unit* AsPlogUnit]) As Double

Arguments

	Argument	Description
	type:	[Optional on page 314] Type of grid to set or return.
	unit:	[Optional] Unit in which the grid value is set or returned.

Examples

The following sample code sets the display grid of the open schematic to the same value as the design grid. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

curGridX = ActiveDocument.GridX

curGridY = ActiveDocument.GridY

ActiveDocument.GridX(plogGridDisplay) = curGridX

ActiveDocument.GridY(plogGridDisplay) = curGridY

End Sub
```

Tip
You cannot assign different values for GridX and GridY. The property is not writable for GridY, so GridY takes on the value assigned to GridX.

Related Topics

Document.GridX Property

Document.Name Property

This property returns the name of the document.

Usage

Name As String

Arguments

None

Description

For example, if the current schematic file is \My Documents\SailWind Projects\Samples\demo.sch, this function returns the string “demo.sch”.

This property is the default property for the Document object.

Examples

The following sample code retrieves the name and location of the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
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 on page 219

Nets(*name* As String) As Net on page 208

Arguments

	Argument	Description
	name	Name of an existing net.

Description

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 in an Objects Collection Object.

Examples

The following sample code retrieves the number of nets in the open schematic using the Objects.Count Property property. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main

Set nets = ActiveDocument.Nets

MsgBox "There are " & Nets.Count & " net(s) in " & ActiveDocument.Name

End Sub
```

The following sample code retrieves the number of pins connected to net VCC, assuming it exists in the open schematic, using the Net.Pins Property and Objects.Count Property properties. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main

Set netVCC = ActiveDocument.Nets("VCC")

MsgBox "Net " & netVCC.Name & " connects " & netVCC.Pins.Count & "
 pin(s)."

End Sub
```

Related Topics

Document.GetObjects Method

Sheet.Nets Property

Document.Parent Property

This property returns the parent of the object.

Usage

Parent As Application on page 401

Arguments

None

Description

None

Document.PartTypes Property

This property returns the collection of all part types.

Usage

PartTypes As Objects on page 219

PartTypes(name As String) As PartType on page 234

Arguments

	Argument	Description
	name	Name of an existing part type.

Description

When an existing part type *name* is passed to this property, it returns that PartType on page 77 object. Otherwise, this property returns the collection of all part types in an Objects Collection Object.

Examples

The following sample code retrieves the number of part types in the open schematic using the Objects.Count Property property. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main
```

```
Set pkgs = ActiveDocument.PartTypes
```

```
MsgBox "There are " & pkgs.Count & " part type(s) in " &
 ActiveDocument.Name
```

```
End Sub
```

Related Topics

Document.GetObjects Method

Document.Path Property

This property returns the path of the document.

Usage

Path As String

Arguments

None

Description

None

Examples

The following sample code retrieves the name and path of the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
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 the collection of all pins.

Usage

Pins As Objects on page 219

Pins(*name* As String) As Pin on page 246

Arguments

	Argument	Description
	name	Name of an existing pin

Description

When an existing pin *name* is passed to this property, it returns that Pin Object. Otherwise, it returns the collection of all pins in an Objects Collection Object.

Examples

The following sample code retrieves the number of pins in the open schematic using the Objects.Count Property property. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main

Set pins = ActiveDocument.Pins

MsgBox "There are " & Pins.Count & " pin(s) in " & ActiveDocument.Name

End Sub
```

Related Topics

Document.GetObjects Method

Sheet.Nets Property

Document.Saved Property

This property sets or returns whether the document is saved.

Usage

Saved As Boolean

Arguments

None

Description

This property is usually used just before opening a new schematic in SailWind Logic, using the Application.OpenDocument Method, to ensure that the message *Save old file before reloading?* does not occur.

Examples

The following sample code sets the saved status of the open schematic to True, then opens demo.sch, assuming that it exists in the folder pointed by the Application.DefaultFilePath Property property. It then retrieves the name of the newly opened schematic. See Code Samples for more information on running this sample.

```
Sub Main

ActiveDocument.Saved = True

OpenDocument(DefaultFilePath & "\DEMO.SCH")

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

End Sub
```

Related Topics

Document.Save Method

Document.SaveAs Method

Document.Sheets Property

This property returns the collection of all sheets.

Usage

Sheets As Sheets on page 265

Sheets(*name* As String) As Sheet on page 265

Arguments

	Argument	Description
	name	Name of an existing sheet.

Description

When an existing sheet with *name* is passed to this property, it returns that sheet packaged as a Sheet Object. Otherwise, it returns the collection of all sheets existing in the open schematic, packaged as a Sheets Collection Object.

Examples

The following sample code retrieves the number of sheets in the open schematic using the Sheets.Count Property property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set shts = ActiveDocument.Sheets

MsgBox "There are " & shts.Count & " sheet(s) in " & ActiveDocument.Name

End Sub
```

Document.Activate Method

This method activates the window associated with the document.

Usage

Activate ()

Arguments

None

Return Values

None

Description

This is a Microsoft requirement. However, since SailWind Logic is a SDI (Single Document Interface) server application, this function has no effect.

Document.ExportASCII Method

This method generates a SailWind Logic ASCII file from the current schematic.

Usage

ExportASCII(*path* As String)

Arguments

	Argument	Description
	path	Path to the output file.

Description

None

Examples

The following sample creates an ASCII file from active document. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 ActiveDocument.ExportASCII DefaultFilePath & "\ascii.txt"

End Sub
```

Document.ExportNetList Method

This method generates PADS-format netlist from the current schematic.

Usage

ExportNetList(*path* As String, [*ver* As PlogNetListVersion])

Arguments

	Argument	Description
	path	Name of the file to which to export the netlist.
	ver	[Optional on page 314] Version in which to export (for backward compatibility).

Return Values

None

Description

If the specified file *name* does not exist the function creates new file. If the file *name* does exist, this method overwrites the existing file.

This property generates an Exception if the function failed.

Examples

The following sample code creates a netlist file with the specified name (padsnet.asc). See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

ActiveDocument.ExportNetList(DefaultFilePath & "\padsnet.asc")

End Sub
```

Document.GenerateECO Method

This method compares the open schematic with specified netlist or other schematic and generates ECO file.

Usage

GenerateECO(*fileToComp* As String, *path* As String)

Arguments

	Argument	Description
	fileToComp	Name of the schematic or netlist file to compare with the open schematic.
	path	Name of the resulting ECO file.

Return Values

None

Description

None

Examples

The following sample code compares PADS-format netlist file (padsnet.asc) and creates an ECO file with the specified name (eco2pcb.eco). See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.GenerateECO(DefaultFilePath & "\padsnet.asc",
 DefaultFilePath & "\eco2pcb.eco")
```

```
End Sub
```

Document.GetColor Method

This method returns the color of the specified document element.

Usage

GetColor(*colorType* as PlogDocumentColor) as Integer

Arguments

	Argument	Description
	colorType	Specifies the document element.

Return Values

The index of the color in the palette.

Examples

This sample also includes the SetColor method.

```
doc = Application.ActiveDocument
```

```
msg=""
```

```
msg = msg & doc.GetColor(plogDocumentColorBackground) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorSelection) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorConnection) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorBus) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorLine) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorPart) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorHierarchicalComp) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorText) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorTextBox) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorRefDes) & ", "
msg = msg & doc.GetColor(plogDocumentColorRefDesBox) & ", "
msg = msg & doc.GetColor(plogDocumentColorPartType) & ", "
msg = msg & doc.GetColor(plogDocumentColorPartTypeBox) & ", "
msg = msg & doc.GetColor(plogDocumentColorPartText) & ", "
msg = msg & doc.GetColor(plogDocumentColorPartTextBox) & ", "
msg = msg & doc.GetColor(plogDocumentColorPinNumber) & ", "
msg = msg & doc.GetColor(plogDocumentColorPinNumberBox) & ", "
msg = msg & doc.GetColor(plogDocumentColorNetName) & ", "
msg = msg & doc.GetColor(plogDocumentColorNetNameBox) & ", "
msg = msg & doc.GetColor(plogDocumentColorField) & ", "
msg = msg & doc.GetColor(plogDocumentColorFieldBox)

MsgBox msg

curr_bkg_color = doc.GetColor(plogDocumentColorBackground)
doc.SetColor(plogDocumentColorBackground, 2)

MsgBox "Press any key"

doc.SetColor(plogDocumentColorBackground, curr_bkg_color)
```

Document.GetObjects Method

This method returns a collection of SailWind Logic database objects.

Usage

GetObjects([*type* As PlogObjectType = plogObjectTypeAll], [*value* As String], [*selected* As Boolean = False]) As Objects on page 219

Arguments

	Argument	Description
	type	[Optional on page 314] Type of SailWind Logic database object to get.
	name	[Optional] Value or name of the object(s) to get.
	selected	[Optional] True to get selected objects only. False to get all objects.

Return Values

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

Description

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

Name argument supports wildcarding (“U*”), lists of items delimited by comma (“U1, U2, R1”), ranges specified by two object names and the dash character (“U1 - U10, U12, R1 - R20”). Dash must be surrounded by spaces since the dash is a legal symbol in an object name. Only one wildcard per name is allowed and you cannot specify wildcards in a range. You can pass *name* such as “U*, R*, C1 – C100” but you cannot pass *name* such as “U*1*” or “C1* - C10*”.

To get all objects of the same type, use the corresponding object Document property instead of this method. For example, to get all gates in the open schematic, use Document.Gates Property instead of Document.GetObjects(plogObjectTypeGate).

This property generates an Exception if the *type* argument is not a valid SailWind Logic database object type.

Examples

The following sample code shows different ways to use this method, displaying the number of objects retrieved for each way, using the Objects.Count Property property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
Dim objs As Object
```

```
' Ex1: Get all objects of all types

Set objs = ActiveDocument.GetObjects

MsgBox "Ex1: " & objs.Count & " objects."

' Ex2: Get all selected objects of all types

Set objs = ActiveDocument.GetObjects(,,True)

MsgBox "Ex2: " & objs.Count & " selected objects."

' Ex3: Get all net objects

Set objs = ActiveDocument.GetObjects(plogObjectTypeNet)

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(plogObjectTypeNet, "VCC")

MsgBox "Ex4: " & objs.Count & " VCC net objects."

' Ex5: Get all part objects which names begin with U

Set objs = ActiveDocument.GetObjects(plogObjectTypeComponent, "U*")

MsgBox "Ex3: " & objs.Count & " U* part objects."

End Sub
```

Related Topics

Document.SelectObjects Method

Document.Components Property

Document.Gates Property

Document.Nets Property

Document.Pins Property

Document.PartTypes Property

Sheet.GetObjects Method

Sheet.Components Property

Sheet.Gates Property

Sheet.Nets Property

Sheet.PartTypes Property

Document.ImportASCII Method

This method imports a SailWind Logic ASCII file.

Usage

ImportASCII(*name* As String)

Arguments

	Argument	Description
	name	Name of an existing ASCII file to import.

Description

The function fails and generates an Exception if the specified file name does not exist or its format is incorrect.

Examples

The following sample code imports the specified file (demo.txt) to the current schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.ImportASCII(DefaultFilePath & "\demo.txt")
```

```
End Sub
```

Related Topics

Document.ExportNetList Method

Document.ExportASCII Method

Document.ImportECO Method

This method imports an ECO file.

Usage

ImportECO(*name* As String)

Arguments

	Argument	Description
	name	Name of an existing ECO file to import.

Description

The function fails and generates an Exception if the specified file name does not exist or its format is incorrect.

Examples

The following sample code imports the specified ECO file (*eco2sch.eco*) to the current schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

ActiveDocument.ImportECO(DefaultFilePath & "\eco2sch.eco")

End Sub
```

Related Topics

Document.ExportNetList Method

Document.IntegrityTest Method

This method runs the integrity test.

Usage

IntegrityTest() as Boolean

Arguments

None

Return Values

TRUE if the test passed without errors, FALSE in other cases.

Document.Save Method

This method saves the document.

Usage

Save

Arguments

None

Description

None

Examples

The following sample code saves, if needed, the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
If ActiveDocument.Saved = False Then ActiveDocument.Save
```

```
End Sub
```

Related Topics

Document.SaveAs Method

Document.Saved Property

Document.Save Event

Document.SaveAs Method

This method saves the document under a new name.

Usage

Save(*name* As String)

Arguments

	Argument	Description
	name	Name of the new file.

Examples

The following sample code creates a custom backup of the open schematic. If the schematic file name is *XXX.SCH*, the backup file is saved in the same folder with the name XXX (Backup on 12-28-1998 4h40).SCH. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

curSchematicName = ActiveDocument.FullName

theDate = Month(Date) & "-" & Day(Date) & "-" & Year(Date) & " at " &
 Hour(Now) & "h" & Minute(Now)

bakSchematicName = Left$(curSchematicName, Len(curSchematicName)-4) &
 " (Backup on " & theDate & ").sch"

ActiveDocument.SaveAs(bakSchematicName)

OpenDocument(curSchematicName)

End Sub
```

Related Topics

Document.Save Method

Document.Saved Property

Document.Save Event

Document.SaveAsNoLock Method

This method saves the document with a new name and releases the file lock.

Usage

SaveAsNoLock(*filename* as String)

Arguments

	Argument	Description
	filename	New name for the file.

Return Values

This method generates an exception if a SailWind Logic security failure occurs during processing.

Document.SaveAsTemp Method

This method works in the same way as SaveAsNoLock but additionally it doesn't add the filename to the MRU (Most Recently Used) list. This method is mainly intended for macro tests.

Usage

SaveAsTemp(*filename* as String)

Arguments

	Argument	Description
	filename	New name for the file.

Return Values

This method generates an exception if a SailWind Logic security failure occurs during processing.

Document.SaveNoLock Method

This method saves the document and releases the file lock.

Usage

SaveNoLock

Arguments

None

Return Values

This method generates an exception if a SailWind Logic security failure occurs during processing.

Document.SaveTemp Method

This method works in the same way as SaveNoLock but additionally it doesn't add the filename to the MRU (Most Recently Used) list. This method is mainly intended for macro tests.

Usage

SaveTemp

Arguments

None

Return Values

This method generates an exception if a SailWind Logic security failure occurs during processing.

Document.SelectObjects Method

This method selects or deselects SailWind Logic database objects.

Usage

SelectObjects([*type* As PlogObjectType = plogObjectTypeAll], [*name* As String], [*select* As Boolean = True])

Arguments

	Argument	Description
	type	[Optional on page 314] Type of SailWind Logic database object to select/deselect.
	name	[Optional] Value or name of the object(s) to select/deselect.
	select	[Optional] True to select. False to deselect.

Description

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

Name argument supports wildcarding (“U*”), lists of items delimited by comma (“U1, U2, R1”), ranges specified by two object names and the dash character (“U1 - U10, U12, R1 - R20”). Dash must be surrounded by spaces since the dash is a legal symbol in an object name. Only one wildcard per name is allowed and you cannot specify wildcards in a range. You can pass *name* such as “U*, R*, C1 – C100” but you cannot pass *name* such as “U*1*” or “C1* - C10*”.

This property generates an Exception if the *type* argument is not a valid SailWind Logic database object type.

Examples

The following sample code shows different ways to use this method, displaying the number of objects selected for each way, using the Document.GetObjects Method method and the Objects.Count Property property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Dim objs As Object

' Ex1: Select all objects of all types

ActiveDocument.SelectObjects

Set objs = ActiveDocument.GetObjects(,,True)
```

```
MsgBox "Ex1: " & objs.Count & " selected objects (all)."

' Ex2: Unselelect 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(plogObjectTypeNet)

Set objs = ActiveDocument.GetObjects(,,True)

MsgBox "Ex3: " & objs.Count & " selected objects (all nets)."

' Ex4: Unselect net VCC

ActiveDocument.SelectObjects(plogObjectTypeNet, "VCC", False)

Set objs = ActiveDocument.GetObjects(,,True)

MsgBox "Ex4: " & objs.Count & " selected objects (all nets except VCC)."

' Ex5: Select only part objects which names begin with U

ActiveDocument.SelectObjects(,,False)

 ActiveDocument.SelectObjects(plogObjectTypeComponent, "U*")

Set objs = ActiveDocument.GetObjects(,,True)

MsgBox "Ex5: " & objs.Count & " selected U* part objects."

End Sub
```

Related Topics

Document.GetObjects Method

Document.SelectionChange Event

Document.SetColor Method

This method sets the color for the specified document element.

Usage

SetColor(*colorType* as PlogDocumentColor, *colorIndex* as Integer)

Arguments

	Argument	Description
	colorType	Specifies the document element.
	colorIndex	The index of the color in the palette. Must be between 0 and 31.

Examples

This sample also includes the GetColor method.

```
doc = Application.ActiveDocument
```

```
msg=""
```

```
msg = msg & doc.GetColor(plogDocumentColorBackground) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorSelection) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorConnection) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorBus) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorLine) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorPart) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorHierarchicalComp) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorText) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorTextBox) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorRefDes) & ", "
```

```
msg = msg & doc.GetColor(plogDocumentColorRefDesBox) & ", "

msg = msg & doc.GetColor(plogDocumentColorPartType) & ", "

msg = msg & doc.GetColor(plogDocumentColorPartTypeBox) & ", "

msg = msg & doc.GetColor(plogDocumentColorPartText) & ", "

msg = msg & doc.GetColor(plogDocumentColorPartTextBox) & ", "

msg = msg & doc.GetColor(plogDocumentColorPinNumber) & ", "

msg = msg & doc.GetColor(plogDocumentColorPinNumberBox) & ", "

msg = msg & doc.GetColor(plogDocumentColorNetName) & ", "

msg = msg & doc.GetColor(plogDocumentColorNetNameBox) & ", "

msg = msg & doc.GetColor(plogDocumentColorField) & ", "

msg = msg & doc.GetColor(plogDocumentColorFieldBox)

MsgBox msg

curr_bkg_color = doc.GetColor(plogDocumentColorBackground)

doc.SetColor(plogDocumentColorBackground, 2)

MsgBox "Press any key"

doc.SetColor(plogDocumentColorBackground, curr_bkg_color)
```

Document.Save Event

This event occurs after the document is saved.

Usage

```
Document_Save ()
```

Arguments

None

Description

This event occurs after the document has been saved.

Related Topics

Document.Save Method

Document.SaveAs Method

Document.SelectionChange Event

This event occurs when the current selection changes.

Usage

Document_SelectionChange ()

Arguments

None

Description

This event occurs when the current selection changes.

Related Topics

Document.SelectObjects Method

Objects.Select Method

Field Object

The Field object contains properties required by the operating system for a schematic file. This object enables you to retrieve information for the field name, value, and parent.

The following list identifies the Field Properties:

- Field.Application Property
- Field.Name Property
- Field.Parent Property
- Field.Value Property

Field.Application Property

This property returns the Application object.

Usage

Application As SailWind Logic.Application on page 401

Arguments

None

Description

This property identifies the object as an Automation object. This is a Microsoft-required property.

Examples

The following sample code retrieves the name of the application using the first field in the collection. See "Code Samples" on page 21 for more information on running this sample.

```
MsgBox ActiveDocument.Fields(1).Application.Name
```

Field.Name Property

This property returns the name of the field object.

Usage

Name As String on page 401

Arguments

None

Description

This is a Microsoft-required property.

Examples

The following sample code retrieves the name of the first field in the fields collection. See “Code Samples” on page 21 for more information on running this sample.

```
MsgBox ActiveDocument.Fields(1).Name
```

Field.Parent Property

This property returns the parent of a field. The parent is the current Document object.

Usage

Parent As Object on page 401

Arguments

None

Description

This is a Microsoft-required property.

Examples

The following sample code retrieves the name of the parent object using the first field in the collection. See “Code Samples” on page 21 for more information on running this sample.

```
MsgBox ActiveDocument.Fields(1).Parent.Name
```

Field.Value Property

This default property sets or returns the value of an Application field.

Usage

Value As Variant on page 401

Arguments

None

Description

This property enables you to view or set the value of an Application field.

Tip
You can neither delete nor change the value of a system field. When you perform an operation on a field, it affects all field labels that refer to the field in the current schematic.

Examples

The following sample code sets the value of a field.

Tip
The FieldName field should exist in the document.

```
ActiveDocument.Fields ("FieldName").Value="FieldValue"
```

See “Code Samples” on page 21 for more information on running these samples.

The following sample code sets the value of a field, using the default Value property.

```
ActiveDocument.Fields ("FieldName")="FieldValue"
```

Fields Collection Object

The Fields Collection object contains properties and methods to provide a means of adding to and deleting fields from a collection in an open schematic.

This object is usually retrieved using the Application.ActiveDocument Property object.

Tips:

- When you perform an operation on a field, it affects all field labels that refer to the field in the current schematic.
- You can neither delete nor change the value of a system field.

The following list identifies the Field Properties and Methods:

Fields.Application Property
Fields.Count Property
Fields.Item Property
Fields.Parent Property
Fields.Add Method
Fields.Delete Method

Fields.Application Property

This property returns the Application object.

Usage

Application As SailWind Logic.Application on page 401

Arguments

None

Description

This property identifies the object as an Automation object. This is a Microsoft-required property.

Examples

The following sample code retrieves the name of the application using the fields collection object. See "Code Samples" on page 21 for more information on running this sample.

```
MsgBox ActiveDocument.Fields.Application.Name
```

Fields.Count Property

This property returns the number of fields in the fields collection of the current Document.

Usage

Count As Long on page 401

Arguments

None

Description

This property returns the number of fields in the fields collection of the current Document. This is a Microsoft-required property.

Examples

The following sample code displays the number of fields in the document. See “Code Samples” on page 21 for more information on running this sample.

```
MsgBox ActiveDocument.Fields.Count
```

Fields.Item Property

This default property returns a member of a fields collection. The member can be returned either by its position or its name.

Usage

Item as SailWind Logic.Field on page 401

Arguments

	Argument	Description
	Index	A required variant, which can have one of two values
	Value	• The string name of a field • A number with a value from one to the value of the collection's Count property

Description

This property returns a member of a fields collection in a schematic. This is a Microsoft-required property for collection objects.

Tip
You can neither delete nor change the value of a system field.

Examples

The following sample code displays the value of the given field in an open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
MsgBox ActiveDocument.Fields.Item
```

```
("FieldName").Value
```

The following sample displays the value of the given field, using the default item property. See “Code Samples” on page 21 for more information on running this sample.

```
MsgBox ActiveDocument.Fields("AttrName").Value
```

Fields.Parent Property

This property returns the parent Application of the fields collection object.

Usage

Parent As Object on page 401

Arguments

None

Description

This property returns the parent of the fields collection object. The parent object is a document. This is a Microsoft-required property.

Examples

The following sample code retrieves the name of the parent of the fields collection object. (The parent of the fields collection object is the current document.) See “Code Samples” on page 21 for more information on running this sample.

```
ActiveDocument.Fields.Parent.Name
```

Fields.Add Method

This method adds a new field to the collection of fields in the current Document and returns a new field object.

Usage

Add(*name* As String, value As Variant) As SailWind Logic.Field on page 143

Arguments

	Argument	Description
	Value	A variant, which specifies the value of a field.

Description

This method generates an Exception if the *name* argument is an existing field or if it is not a valid field name.

Examples

The following sample code adds a new field to a document. See “Code Samples” on page 21 for more information on running this sample.

```
ActiveDocument.Fields.Add "FieldName","StringValue"
```

Related Topics

Fields.Delete Method

Fields.Delete Method

This method deletes the field, by index, from the fields collection.

Usage

Delete(*index* As Variant) As SailWind Logic.Field on page 143

Arguments

	Argument	Description
	Index	A required variant, which can have one of two values: • The string name of a field • A number with a value from one to the value of the collection's Count property

Description

This method generates an Exception if the *index* argument is outside the limits, if it is not a valid index name, or if it points to a system field.

Examples

The following sample code deletes the field by its name. See "Code Samples" on page 21 for more information on running this sample.

```
ActiveDocument.Fields.Delete("FieldName")
```

Related Topics

Fields.Add Method

Gate Object

The Gate object represents a physical gate that exists in the open schematic. Gates can be used or unused. If the gate is unused, its Sheet property is *Nothing*.

The following button presents a list of the properties in the Gate object.

The following list identifies the Gate Properties and Methods:

Gate.Application Property

This property returns the SailWind Logic Application object.

Usage

Application As Application on page 401

Arguments

None

Description

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

Gate.Component Property

This property returns the component of which the gate is a part.

Usage

Component As Component on page 77

Arguments

None

Examples

The following sample code retrieves the component of which the gate U6-A is a part, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 OpenDocument DefaultFilePath & "\preview.sch"

MsgBox "Gate U6-A belongs to component " &
 ActiveDocument.Gates("U6-A").Component.Name

End Sub
```

Gate.Name Property

This property returns the name of the gate.

Usage

Name As String

Arguments

None

Description

This property is the default property for the Gate object.

Examples

The following sample code retrieves all gates, by name, for the active sheet and lists them in a custom dialog box. When a gate is selected in the list box, the sample selects that gate in SailWind Logic.

This sample uses the UserDialog Editor in the Sax Basic Engine in SailWind Logic. See Sax Basic Editor On-line Help for more information. See “Code Samples” on page 21 for more information on running this sample.

```
Dim ListGates$(10000)

Sub Main

index = 0

for Each nextGate In ActiveDocument.ActiveSheet.Gates

ListGates$(index) = nextGate.Name

index = index + 1

Next nextGate

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

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

ListBox 10,7,160,203,ListGates(),.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(plogObjectTypeAll, , False)

ActiveDocument.SelectObjects(plogObjectTypeGate, ListGates(SuppValue%))

End If

End Select

End Function
```

Gate.Number Property

This property returns the index of this gate in the part.

Usage

Number As Long

Arguments

None

Description

U1-A gate is index number 1; U1-B is index number 2; and so forth.

Examples

The following sample code retrieves the index of gate U6-A, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 OpenDocument DefaultFilePath & "\preview.sch"

 Set U6A = ActiveDocument.Gates("U6-A")

MsgBox "Gate U6-A has gate index " & U6A.Number & " in component " &
 U6A.Component.Name

End Sub
```

Gate.ObjectType Property

This property returns the type of the object.

Usage

ObjectType As PlogObjectType

Arguments

None

Description

This property returns PlogObjectTypeGate.

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

This property is generally used:

- To identify the kind of SailWind Logic database objects in a heterogeneous Objects Collection Object.
- When implementing a generic routine that depends on the type of the SailWind Logic database object passed as an argument. For example:

```
Sub DoSomething(dbObject As Object)
```

```
Select Case dbObject.ObjectType
```

```
Case plogObjectTypeComponent
```

```
' Do something specific to component objects
```

```
Case plogObjectTypeNet
```

```
' Do something specific to net objects
```

```
Case plogObjectTypePin
```

```
' Do something specific to pin objects
```

```
Case plogObjectTypeGate
```

```
' Do something specific to gate objects
```

```
  Case Else
```

```
MsgBox "Not a SailWind Logic database object"
```

```
End Select
```

```
End Sub
```

Gate.Parent Property

This property returns the parent of the object.

Usage

Parent As Document on page 419

Arguments

None

Description

None

Gate.Pins Property

This property returns the collection of all pin for the gate.

Usage

Pins As Objects on page 219

Pin(*name* As String) As Pin on page 246

Arguments

	Argument	Description
	name	Name of an existing pin.

Description

When an existing pin *name* is passed to this property, it returns that Pin Object. Otherwise, it returns the collection of all pins of the gate in an Objects Collection Object.

Examples

The following sample retrieves the total number of pins in gate U6-A. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 OpenDocument DefaultFilePath & "\preview.sch"

 MsgBox "Total pin count in the gate U6A is " &
 ActiveDocument.Gates("U6-A").Pins.Count

End Sub
```

Gate.PositionX Property

This property returns the x-coordinate of the component.

Usage

PositionX([*unit* As PlogUnit]) As Double

Arguments

	Argument	Description
	unit	[Optional on page 314] Unit in which the x coordinate is returned.

Description

None

Examples

The following sample code retrieves the location of gate U6-A in current design units. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 OpenDocument DefaultFilePath & "\preview.sch"

Set U6A = ActiveDocument.Gates("U6-A")

MsgBox "U6-A position is (" & U6A.PositionX & ", " & U6A.PositionY & ")"

End Sub
```

Gate.PositionY Property

This property returns the y-coordinate of the component.

Usage

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

Arguments

	Argument	Description
	unit	[Optional on page 314] Unit in which the y coordinate is returned.

Description

None

Examples

The following sample code retrieves the location of gate U6-A in current design units. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 OpenDocument DefaultFilePath & "\preview.sch"

Set U6A = ActiveDocument.Gates("U6-A")

MsgBox "U6-A position is (" & U6A.PositionX & ", " & U6A.PositionY & ")"

End Sub
```

Gate.ReflectedX Property

This property sets mirroring or returns whether the gate is mirrored along the X axis.

Usage

ReflectedX As Boolean

Arguments

None

Description

This property generates an Exception if gate is unused.

Examples

The following sample rotates and mirrors gate U6-A. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 OpenDocument DefaultFilePath & "\preview.sch"

 Set U6A = ActiveDocument.Gates("U6-A")

 U6A.ReflectedX = True

 U6A.ReflectedY = True

 U6A.Rotated90 = True

End Sub
```

Related Topics

Gate.Rotated90 Property

Gate.ReflectedY Property

Gate.ReflectedY Property

This property sets mirroring or returns whether the gate is mirrored along the Y axis.

Usage

ReflectedY As Boolean

Arguments

None

Description

This property generates an Exception if gate is unused.

Examples

The following sample rotates and mirrors gate U6-A. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 OpenDocument DefaultFilePath & "\preview.sch"

 Set U6A = ActiveDocument.Gates("U6-A")

 U6A.ReflectedX = True

 U6A.ReflectedY = True

 U6A.Rotated90 = True

End Sub
```

Related Topics

Gate.Rotated90 Property

Gate.ReflectedX Property

Gate.Rotated90 Property

This property sets rotation or returns whether the gate is rotated.

Usage

Rotated90 As Boolean

Arguments

None

Description

This property generates an Exception if the gate is unused.

Examples

The following sample rotates and mirrors gate U6-A. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 OpenDocument DefaultFilePath & "\preview.sch"

 Set U6A = ActiveDocument.Gates("U6-A")

 U6A.ReflectedX = True

 U6A.ReflectedY = True

 U6A.Rotated90 = True

End Sub
```

Related Topics

Gate.ReflectedX Property

Gate.ReflectedY Property

Gate.Selected Property

This property sets or returns whether the gate is selected.

Usage

Selected As Boolean

Arguments

None

Description

You can also select a SailWind Logic database object using the Document.SelectObjects Method or Objects.Select Method methods.

Examples

The following sample code selects a gate, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
 ActiveDocument.Gates("U1-A").Selected = True
```

```
End Sub
```

Related Topics

Document.SelectionChange Event

Gate.Sheet Property

This property returns the sheet on which the gate is located.

Usage

Sheet As Sheet on page 265

Arguments

None

Description

If the gate is unused, the return value is *Nothing*. You can check the Gate for unused status using this property.

Examples

The following sample retrieves the first gate of component U1 and retrieves the name of sheet on which the gate is located. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 Set gate = ActiveDocument.Components("U1").Gate(1)

 If gate.Sheet Is Nothing then

  MsgBox "This Gate is Unused!"

 Else

  MsgBox "First gate of part U1 is located in sheet " & gate.Sheet

 End If

End Sub
```

Gate.SwapClass Property

This property returns the Swap Class of this gate.

Usage

SwapClass As Long

Arguments

None

Description

None

Examples

The following sample code retrieves the swap class of gate U6-A, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 OpenDocument DefaultFilePath & "\preview.sch"

 Set U6A = ActiveDocument.Gates("U6-A")

MsgBox "Gate U1-A has swap class " & U6A.SwapClass

End Sub
```

Related Topics

Gate.Number Property

Gate.Visibility Property

This property sets visibility or returns whether the gate is visible.

Usage

Visibility([*Item* As PlogGateVisibility = plogAttrVisibility], [*AttrName* As String=" "]) As Boolean

PlogGateVisibility items. Possible values are:

plogAttrVisibility = 0

plogAttrNameVisibility = 1

plogRefDesVisibility = 2

plogPartTypeVisibility = 3

plogPinNumberVisibility = 4

plogPinNameVisibility = 5

plogPCBDecalVisibility = 6

plogPCBDecalNameVisibility = 7

Arguments

	Argument	Description
	Item	[Optional on page 314] Visibility item. By default, it specifies the visibility of the attribute specified in AttrName.
	AttrName	[Optional] Attribute name. It can be used with items plogAttrVisibility and plogAttrNameVisibility. It should be omitted for other items.

Description

This function does not redraw the gate. Use the View.Refresh Method method to update the screen.

Examples

The following sample code sets the visibility of gate U6-A. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 OpenDocument DefaultFilePath & "\preview.sch"

 Set U6A = ActiveDocument.Gates("U6-A")
```

```
 'Show attribute "MFG #1" if it exists

 If Not U6A.Component.Attributes("MFG #1") Is Nothing Then

  U6A.Visibility(,"MFG #1") = True

 End If

 U6A.Visibility(plogRefDesVisibility) = False 'Hide reference designator

 ActiveDocument.ActiveView.Refresh

End Sub
```

Related Topics

View.Refresh Method

Gate.Delete Method

This method deletes this gate from the schematic.

Usage

Delete

Arguments

None

Description

None

Examples

The following sample deletes gate U6-A from the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 OpenDocument DefaultFilePath & "\preview.sch"

 ActiveDocument.Gates("U6-A").Delete

End Sub
```

Gate.Move Method

This method moves this gate to new location.

Usage

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

Arguments

	Argument	Description
	x	X-coordinate of the new component position.
	y y	Y-coordinate of the new component position.
	unit	[Optional on page 314] Unit in which the x and y coordinates are given.

Description

This property generates an Exception if the arguments are not valid.

Examples

The following sample code moves all gates to the left on the current design grid. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 For Each nextGate In ActiveDocument.ActiveSheet.Gate

  nextGate.Move nextGate.PositionX – ActiveDocument.GridX,
 nextGate.PositionY

Next nextGate

End Sub
```

Related Topics

Gate.PositionX Property

Gate.PositionY Property

LibraryItem Object

The LibraryItem object represents an item existing in a particular part library.

The following list identifies the LibraryItem Properties:

LibraryItem.Application Property

This property returns the Application object.

Usage

Application As Application on page 401

Arguments

None

Description

This is a Microsoft-required property.

LibraryItem.Library Property

This property returns a library to the Library Item to which it belongs.

Usage

Library As Object on page 219

Arguments

None

Description

None

Examples

The following sample code displays the name of the library to which an item belongs. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set items = GetLibraryItems()

for Each item In items

MsgBox "The item " & item & " belongs to " & item.Library & " library"

Exit For

Next item

End Sub
```

LibraryItem.Name Property

This default property returns the Library Item's name.

Usage

Name As String

Arguments

None

Description

This is a Microsoft-required property.

Examples

The following sample displays a message box showing the name of the first item in the available libraries. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set items = GetLibraryItems()

for Each item In items

MsgBox "The first item name is " & item.Name

Exit For

Next item

End Sub
```

LibraryItem.ObjectType Property

This property returns the object type.

Usage

ObjectType As PlogObjectType

Arguments

None

Description

None

Examples

The following sample tests the ObjectType property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set items = GetLibraryItems()

for Each item In items

If item.ObjectType <> plogObjectTypeLibraryItem Then

MsgBox "Test failed"

End If

Next item

End Sub
```

LibraryItem.Parent Property

This property returns the parent of this object.

Usage

Parent As Application on page 401

Arguments

None

Description

This is a Microsoft-required property. The Parent property of a LibraryItem object always returns Application.

LibraryItem.Type Property

This property returns the LibraryItem object type.

Usage

Type as PlogLibraryItemType

Arguments

None

Description

None

Examples

The following sample displays the type of a library item. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set items = GetLibraryItems()

for Each item In items

Select Case item.Type

Case plogLibraryItemTypePartType

MsgBox "Item type: PartType"

Case plogLibraryItemType

MsgBox "Item type: Decal"

Case plogLibraryItemType

MsgBox "Item type: LogicDrawing"

Case plogLibraryItemType

MsgBox "Item type: Drawing"
```

```
End Select
```

```
Exit For
```

```
Next item
```

```
End Sub
```

Library Object

The Library object represents a library included in the library list.

The following list identifies the Library Properties and Methods:

Library.Application Property

This property returns the Application object.

Usage

Application As Application on page 401

Arguments

None

Description

This is a Microsoft-required property.

Library.FullName Property

This property returns the full name of library files, including path and name.

Usage

FullName As String

Arguments

None

Description

None

Examples

The following sample displays a message box showing the name of the first available library. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

for Each lib In Libraries

MsgBox "The full name of the first available library is " & lib.FullName

Exit For

Next lib

End Sub
```

Library.Name Property

This default property returns the Library's name.

Usage

Name As String

Arguments

None

Description

This is a Microsoft-required property.

Examples

The following sample displays a message box showing the name of the first available library. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

for Each lib In Libraries

MsgBox "The name of the first available library is " & lib.Name

Exit For

Next lib

End Sub
```

Library.ObjectType Property

This property returns the type of this object.

Usage

ObjectType As PlogObjectType

Arguments

None

Description

None

Examples

The following sample tests the ObjectType property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

for Each lib In Libraries

If lib.ObjectType <> plogObjectTypeLibrary Then

MsgBox "Test failed"

End If

Next lib

End Sub
```

Library.Parent Property

This property returns the parent of this object.

Usage

Parent As Application on page 401

Arguments

None

Description

This is a Microsoft-required property.

Library.Path Property

This property returns the path to the Library.

Usage

Path As String

Arguments

None

Description

None

Examples

The following sample displays a message box showing the name of the first available library. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

for Each lib In Libraries

MsgBox "The path to the first available library is " & lib.Path

Exit For

Next lib

End Sub
```

Library.GetLibraryItems Method

This method returns an object collection of all items in this library, or a specified item.

Usage

GetLibraryItems (*Type* as PlogLibraryItemType, *Name* as String) As Collection

Arguments

	Argument	Description
	Name	Name of a LibraryItem object to retrieve. May contain wildcards, lists, and ranges. Optional on page 314; if omitted, matches any name.
	Type	[Optional]. Parameter specifying the type of object to retrieve. It is plogLibraryItemTypeAll by default.

Description

None

Examples

This sample displays the number of items in a library. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

for Each lib In Libraries

count = lib.GetLibraryItems(plogLibraryItemTypeDecal, "DIP*").Count

MsgBox "Library " & lib.Name & " has " & count & " DIP decals"

Exit For

Next lib

End Sub
```

Library.ImportLibraryItems Method

This method reads library items from PADS-format ASCII file(s) and returns a collection of the most recently imported items.

Usage

ImportLibraryItems (*Filename* as String) as Collection

Arguments

	Argument	Description
	Name	Name of the file from which to import library items.

Description

None

Examples

This sample imports library items from a file, and displays the number of imported items. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

for Each lib In Libraries

Set coll = lib.ImportLibraryItems("C:\sample")

MsgBox coll.Count & " items are successfully imported"

Exit For

Next lib

End Sub
```

Library.ImportLibraryItems2 Method

This method reads library items from PowerPCB or SailWind Layout format ASCII files.

Usage

ImportLibraryItems2 (*Filename* as String, ImportOption as PcbImportLibMode) as Collection

Arguments

	Argument	Description
	Filename	Name of the file from which to import library items.
	ImportOption	specifies the preference for overwriting existing items.

Return Values

Returns the collection of just imported items.

Description

If ImportOption is pcbImportLibModePrompt, then the Application object fires the OverwriteLibraryItemPrompt event. The client application must process this event to specify whether the existing library item should be overwritten.

Measure Object

The Measure object provides an access to the internal SailWind Logic unit parser.

The Measure object can be either constructed from Application object (see Application.Measure method) or obtained from the object (see Attribute.Measure property). From the measure object, you can extract information about real value and unit.

The following list identifies the Measure Properties and Method:

Measure.Application Property

This property returns the SailWind Logic Application object.

Usage

Application As Application on page 401

Arguments

None

Description

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

Measure.Name Property

This property returns the name of the quantity represented by the measure.

Usage

Name As String

Arguments

None

Description

This property returns the name of a quantity represented by the measure object, for example “Capacitance” for measure “10pF”, “Voltage” for “5V”.

Examples

The following sample code shows the quantity name of the “10pF” measure (the Capacitance). See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
MsgBox Measure("10pF").Quantity
```

```
End Sub
```

Related Topics

Measure.Value Property

Measure.Number Property

This property returns the number to combine with prefix and unit.

Usage

Number As Double

Arguments

None

Description

This property returns the left numeric part of the measure. You can format this number before output using standard methods available in programming language you use for creating a client. Use this property with Measure.Prefix Property and Measure.Unit Property.

Examples

The following sample outputs formatted measure (keeping three digits after point) using standard VB formatting. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
Set Cap = Measure (1.12345e-12, "F")
```

```
MsgBox Format(Cap.Number, "#.###") & Cap.Prefix & Cap.Unit
```

```
End Sub
```

Related Topics

Measure.Prefix Property

Measure.Unit Property

Measure.Parent Property

This property returns the SailWind Logic Application object.

Usage

Parent As Application on page 401

Arguments

None

Description

None

Measure.Prefix Property

This property returns the unit prefix to combine with number and unit.

Usage

Prefix([Format As PlogMeasureFormat = plogMeasureFormatStandard]) As String

Arguments

	Argument	Description
	Format	[Optional on page 314] Parameter that indicates format of the prefix representation.

Description

This property returns the prefix currently used in the measure. Use this property with Measure.Number Property and Measure.Unit Property.

Possible Format values:

- plogMeasureFormatStandard—to return standard prefix representation (e.g., p for Pico, k for Kilo)
- plogMeasureFormatCurrent—to return prefix in format currently used in this Measure value
- plogMeasureFormatShort—to return short prefix (e.g., p for Pico, k for Kilo)
- plogMeasureFormatLong—to return short unit name (e.g., Pico, Kilo, Mega)

Examples

See sample for Measure.Number Property

Related Topics

Measure.Number Property

Measure.Unit Property

Measure.Text Property

This property sets or returns the exact text value of the measure.

Usage

Text As String

Arguments

None

Description

This property defines custom format of a measure. Text value consists of number followed by optional prefix and unit. Text value always matches Attribute.Value Property property if an attribute represents a measure

Examples

The following sample code demonstrates the difference between Measure.Text Property, Measure.Value Property and Attribute.Value Property properties.

See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set C1 = ActiveDocument.Components("C1")

Set Cap = Measure("500pF")

C1.Attributes.Add "Capacitance", Cap

MsgBox Cap 'displays 0.0000000005 (the default Value property)

MsgBox Cap.Value 'displays 0.0000000005

MsgBox Cap.Text 'displays 500pF

MsgBox C1.Attributes("Capacitance") 'displays 500pF

MsgBox C1.Attributes("Capacitance").Value 'displays 500pF (long form)

End Sub
```

Related Topics

Measure.Text Property

Attribute.Value Property

Measure.Value Property

Measure.Unit Property

This property returns the physical unit name of the measure.

Usage

Unit([Format As PlogMeasureFormat = plogMeasureFormatStandard]) As String

Arguments

	Argument	Description
	Format	Optional on page 314 parameter that indicates the format of the unit representation.

Description

This property returns the name of the physical unit without prefix.

Possible Format values:

- plogMeasureFormatStandard—to return standard unit representation (e.g., F for Capacitance)
- plogMeasureFormatCurrent—to return unit in format currently used in this Measure value
- plogMeasureFormatShort—to return short unit name (e.g., F for Capacitance)
- plogMeasureFormatLong—to return short unit name (e.g., Farad for Capacitance)

Examples

See sample for Measure.Number Property

Related Topics

Measure.Number Property

Measure.Prefix Property

Measure.Value Property

This property sets or returns the real value of the measure.

Usage

Value As Double

Arguments

None

Description

This property returns floating-point number for a measure taking into account unit prefix. For example, 10000 for “10K” or 2e-10 for 200pF.

This is a default property, so it can be omitted in Basic scripts.

If you set new Value then Measure gets automatically normalized (See Normalize method).

Examples

The following sample demonstrates how to work with measures. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
'1 – Compare two existing measure attributes (.Value call may be omitted)
```

```
Set U1_Val =
 ActiveDocument.Components("U1").Attributes("Value").Measure.Value
```

```
Set U2_Val =
 ActiveDocument.Components("U2").Attributes("Value").Measure.Value
```

```
Set U1_Quantity =
 ActiveDocument.Components("U1").Attributes("Value").Measure.Name
```

```
Set U2_Quantity =
 ActiveDocument.Components("U2").Attributes("Value").Measure.Name
```

```
If U1_Quantity <> U2_Quantity then
```

```
MsgBox "Cannot compare values with different physical units"
```

```
ElseIf U1_Val < U2_Val then
```

```
MsgBox "U1 value is less than U2 value"

ElseIf U1_Val > U2_Val then

MsgBox "U1 value is greater than U2 value"

Else

MsgBox "U1 value is equal to U2 value"

End If

'2 - Check that the Resistor's Value is in range between 100K and 10M

Set R1_Val = ActiveDocument.Components("R1").Attributes("Value").Measure

If R1_Val >= Measure("100k") And R1_Val <= Measure("10M") And
 R1_Val.Name="Resistence" Then

MsgBox "Resistor Is In Range [100k, 10M]

End If

'3 – Calculate total thermal dessipation for all parts

'make sure attribute exists for all parts

For Each part In ActiveDocument.Components

If part.Attributes("Thermal.Dissipation") Is Nothing Then

Part.Attributes.Add "Thermal.Dissipation", Measure("10mW")

End If

Next

Dim Total As Measure 'declare Total as Measure object explicitly!

Set Total = Measure("0mW") 'create Measure to accamulate the total
```

```
For Each part In ActiveDocument.Components

Total = Total + part.Attributes("Thermal.Dissipation").Measure

Next

MsgBox "Total Thermal Dissipation=" & Total.Text

End Sub
```

Related Topics

Measure.Text Property

Measure.Name Property

Measure.Normalize Method

This method normalizes the text value of the measure and returns a new text.

Usage

Normalize As String

Arguments

None

Description

This method selects appropriate unit prefix and appends a unit if it is missed. For example it converts text of the time measure from “5e-9” to “5ns”.

Examples

The following sample normalizes a capacitance measure. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set Cap = Measure(2e-10, "F")

MsgBox Cap.Normalize ' shows 200pF

End Sub
```

Related Topics

Measure.Text Property

Net Object

The Net object represents a physical net that exists in the open schematic.

The following list identifies the Net Properties:

Net.Application Property

This property returns the SailWind Logic Application object.

Usage

Application As Application on page 401

Arguments

None

Description

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

Net.Attributes Property

This property returns the collection of all attributes of the net.

Usage

Attributes As Attributes

Attributes(*name* As String) As Attribute on page 69

Arguments

	Argument	Description
	name	Name of an existing component attribute.

Description

When an existing attribute *name* is passed to this property, it returns that component Attribute on page 69 object. Otherwise, it returns the collection of all net attributes in an Attributes Collection Object Property.

Examples

The following sample code retrieves the number of attributes for net GND, assuming it exists in the open schematic, using the Attributes.Count Property property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set attrs = ActiveDocument.Nets("GND").Attributes

MsgBox "There are " & attrs.Count & " attribute(s) in net GND."

End Sub
```

Net.Name Property

This property returns the name of the net.

Usage

Name As String

Arguments

None

Description

For example, this property returns the string “GND” for net GND.

This property is the default property for the Net object.

Examples

The following sample code retrieves all nets in the open schematic and places that list in a custom dialog box. When a net is selected in the list box, the sample selects that net in SailWind Logic.

This sample uses the UserDialog Editor in the Sax Basic Engine in SailWind Logic. See Sax Basic Editor On-line Help for more information. See “Code Samples” on page 21 for more information on running this sample.

```
Dim ListNets$(10000)

Sub Main

index = 0

for Each nextNet In ActiveDocument.Nets

ListNets$(index) = nextNet.Name

index = index + 1

Next nextNet

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

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

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

```
OKButton 10,210,160,21

End Dialog

Dim dlg As UserDialog

Dialog dlg

End Sub

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

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

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

Select Case Action%

Case 2 ' Value changing or button pressed

If DlgItem$="ListBox1" Then

ActiveDocument.SelectObjects(plogObjectTypeAll, , False)

ActiveDocument.SelectObjects(plogObjectTypeNet, ListNets(SuppValue%))

End If

End Select

End Function
```

Net.ObjectType Property

This property returns the type of the object.

Usage

ObjectType As PlogObjectType

Arguments

None

Description

This property returns PlogObjectTypeNet.

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

This property is generally used:

- To identify the kind of SailWind Logic database objects in a heterogeneous Objects Collection Object.
- When implementing a generic routine that depends on the type of the SailWind Logic database object passed as an argument. For example:

```
Sub DoSomething(dbObject As Object)
```

```
Select Case dbObject.ObjectType
```

```
Case plogObjectTypeComponent
```

```
' Do something specific to component objects
```

```
Case plogObjectTypeNet
```

```
' Do something specific to net objects
```

```
Case plogObjectTypePin
```

```
' Do something specific to pin objects
```

```
Case plogObjectTypeGate
```

```
' Do something specific to gate objects
```

```
  Case Else
```

```
MsgBox "Not a SailWind Logic database object"
```

```
End Select
```

```
End Sub
```

Net.Parent Property

This property returns the parent of the object.

Usage

Parent As Document on page 419

Arguments

None

Description

None

Net.Pins Property

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

Usage

Pins As Objects on page 219

Pins(*name* As String) As Pin on page 246

Arguments

	Argument	Description
	name	Name of an existing pin.

Description

When an existing pin name is passed to this property, it returns that Pin Object. Otherwise, it 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 GND, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

MsgBox "GND connects " & ActiveDocument.Nets("GND").Pins.Count & " pins."

End Sub
```

Net.Selected Property

This property sets or returns whether the net is selected.

Usage

Selected As Boolean

Arguments

None

Description

You can also select a SailWind Logic database object using the Document.SelectObjects Method or Objects.Select Method methods.

Examples

The following sample code selects net GND only, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

ActiveDocument.SelectObjects(,,False)

ActiveDocument.Nets("GND").Selected = True

End Sub
```

Related Topics

Document.SelectionChange Event

Net.Width Property

This property returns the width of the specified net.

Usage

Width([*Unit* As PlogUnit]) As Double

Arguments

	Argument	Description
	Unit	[Optiona on page 314l] Unit in which the width is returned.

Description

None

Examples

The following sample code retrieves the width of net GND, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 MsgBox "Width of GND net is " & ActiveDocument.Nets("GND").Width

End Sub
```

Objects Collection Object

The Objects collection object is a collection of homogeneous or heterogeneous SailWind Logic database objects, such as Component objects, Gate objects, Net objects, and Pin objects, existing in the open schematic.

This object is usually retrieved using the Document.GetObjects Method method or using database object-specific properties such as Document.Components Property, Document.Fields Property, Document.Nets Property, Document.Pins Property, and Document.Gates Property.

The following list identifies the Objects Properties and Methods:

- Objects.Application Property
- Objects.Count Property
- Objects.Item Property
- Objects.ItemType Property
- Objects.Next Property
- Objects.Parent Property
- Objects.Add Method
- Objects.Merge Method
- Objects.Remove Method
- Objects.Reset Method
- Objects.Select Method
- Objects.Sort Method

Objects.Application Property

This property returns the SailWind Logic Application object.

Usage

Application As Application on page 401

Arguments

None

Description

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

Objects.Count Property

This property returns the number of objects in the collection.

Usage

Count As Long

Arguments

None

Description

None

Examples

The following sample code retrieves the number of selected items in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
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.

Usage

Item(*index* As Long) As Object

Item(*name* As String) As Object

Arguments

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

Description

This is the default member of the Objects collection object.

This property generates an Exception if the index or name argument is not valid.

Examples

The following samples show two different methods for iterating through all SailWind Logic database objects contained in the open schematic using Component Object. The second method is generally preferred because it is cleaner and faster. See “Code Samples” on page 21 for more information on running this sample.

```
' Method 1: Use the Object.Item property

Sub Main

Set comps = ActiveDocument.Components

for I=1 To comps.Count

Set thisComp = comps.Item(I)

' Do something with the component thisComp

Next I

End Sub
```

```
' Method 2: Do not use the Object.Item property (preferred method)

Sub Main

for Each nextComp in ActiveDocument.Components

' Do something with the component nextComp

Next nextComp

End Sub
```

Related Topics

Objects.ItemType Property

Objects.ItemType Property

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

Usage

ItemType(*index* As Long) As PlogObjectType

ItemType(*name* As String) As PlogObjectType

Arguments

	Argument	Description
	index	Index of the object in the collection to query.
	name	Name of the object to retrieve.

Description

This property generates an Exception if the *index* or *name* argument is not valid.

Related Topics

Objects.Item Property

Objects.Next Property

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

Usage

Next(*index* As long, *type* As PlogObjectType) As Long

Arguments

	Argument	Description
	index	Index of the object in the collection to query.
	type	Type of the object to query.

Description

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

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

Objects.Parent Property

This property returns the parent of the object.

Usage

Parent As Document on page 419

Arguments

None

Description

None

Objects.Add Method

This method adds an object to the collection.

Usage

Add(*object* As object)

Arguments

	Argument	Description
	object	Object to add to the collection. It must be a SailWind Logic data object such as Component on page 77, Net on page 208, Pin on page 246, or Gate on page 155.

Description

This property generates an Exception if the *object* argument is not a SailWind Logic database object.

Examples

The following sample code creates a collection of all U* parts in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 'Create empty collection

Set UComps = ActiveDocument.GetObjects(plogObjectTypeUnknown)

for Each nextComp In ActiveDocument.Components

If Left(nextComp.Name, 1)="U" Then UComps.Add(nextComp)

Next nextComp

MsgBox "There are " & UComps.Count & " U* parts"

End Sub
```

Related Topics

Objects.Remove Method

Objects.Merge Method

This method merges two object collections.

Usage

Merge(objects As Objects on page 219)

Arguments

	Argument	Description
	objects	The collection with objects to merge with the current object collection.

Description

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

Objects.Remove Method

This method removes an object from the collection.

Usage

Remove(*index* As Long)

Remove(*name* As String)

Arguments

	Argument	Description
	index	Index of the object to remove.
	name	Name of the object to remove.

Description

This property generates an Exception if the *index* or *name* argument is not valid.

Related Topics

Objects.Add Method

Objects.Reset Method

This method resets the object collection.

Usage

Reset()

Arguments

None

Description

None

Related Topics

Objects.Remove Method

Objects.Select Method

This method selects or deselects all objects in the collection.

Usage

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

Arguments

	Argument	Description
	bSelect	[Optional on page 314] True to select. False to unselect.

Description

None

Examples

The following sample code quickly selects all gates in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.Gates.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

Description

None

Examples

The following sample creates collection of pins, sorts them by name, and displays them in a list box. See “Code Samples” on page 21 for more information on running this sample.

```
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 SailWind Logic 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
```

PartType Object

The PartType object represents a part type (package) of a physical part that exists in the open schematic.

The following list identifies the PartType Properties:

PartType.Application Property

This property returns the SailWind Logic Application object.

Usage

Application As Application on page 401

Arguments

None

Description

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

PartType.Components Property

This property returns an object collection of all components of this part type.

Usage

Components As Objects on page 219

Components (*name* As String) As Component on page 77

Arguments

	Argument	Description
	name	Name of an existing component.

Description

When an existing pin *name* is passed to this property, it returns that Component Object. Otherwise, it returns the collection of all components of the part type as Objects Collection Object.

Examples

The following sample displays total number of 7400 parts.

```
Sub Main
```

```
MsgBox Str(ActiveDocument.PartTypes("7400").Components.Count)
```

```
End Sub
```

PartType.ECORegistered Property

This property sets or returns the ECO Registration status for the PartType.

Usage

Selected As Boolean

Arguments

None

Description

None

Examples

```
The following sample code sets the ECO registration status to unregistered
 for the Part Type "87C256" if it exists, then checks to see if it is
 registered displays a message box indicating that it is either ECO
 registered or not.
```

```
Sub Main
 ActiveDocument.PartTypes("87C256").ECORegistered = False

 If ActiveDocument.PartTypes("87C256").ECORegistered Then
  MsgBox "Part 87C256 is ECO Registered."
 Else
  MsgBox "Part 87C256 is not ECO Registered."
 End If
End Sub
```

PartType.Logic Property

This property returns the logic family of the part type.

Usage

Logic As String

Arguments

None

Description

None

Examples

The following sample code retrieves the logic family of the part type 7400, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
MsgBox "The Logic family of 7400 is " &
 ActiveDocument.PartTypes("7400").Logic
```

```
End Sub
```

PartType.Name Property

This property returns the name of the part type.

Usage

Name As String

Arguments

None

Description

For example, this property returns the string “7402” for part type 7402.

This property is the default property for the Component object.

Examples

The following sample code retrieves all part types in the open schematic and places that list in a custom dialog list box. When a part type is selected in the list box, the sample selects all parts of that part type in SailWind Logic. This sample uses the UserDialog Editor in the Sax Basic Engine in SailWind Logic. See Sax Basic Editor On-line Help for more information.

See “Code Samples” on page 21 for more information on running this sample.

```
Dim ListPkgs$(10000)

Sub Main

index = 0

for Each nextPkg In ActiveDocument.PartTypes

ListPkgs$(index) = nextPkg.Name

index = index + 1

Next nextPkg

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

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

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

OKButton 10,210,160,21

End Dialog

Dim dlg As UserDialog

Dialog dlg

End Sub

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

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

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

Select Case Action%

Case 2 ' Value changing or button pressed

If DlgItem$="ListBox1" Then

ActiveDocument.SelectObjects(plogObjectTypeAll, , False)

'get part by name

Set pkg = ActiveDocument.PartTypes(ListPkgs(SuppValue%))

'select part

pkg.Selected = True

'activate sheet where first gate of the part is located

pkg.Components(1).Gates(1).Sheet.Activate

End If
```

```
End Select
```

```
End Function
```

PartType.ObjectType Property

This property returns the type of the object.

Usage

ObjectType As PlogObjectType

Arguments

None

Description

This property returns PlogObjectTypePartType.

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

This property is generally used:

- To identify the kind of SailWind Logic database objects in a heterogeneous Objects Collection Object.
- When implementing a generic routine that depends on the type of the SailWind Logic database object passed as an argument. For example:

```
Sub DoSomething(dbObject As Object)
```

```
Select Case dbObject.ObjectType
```

```
Case plogObjectTypeComponent
```

```
' Do something specific to component objects
```

```
Case plogObjectTypeNet
```

```
' Do something specific to net objects
```

```
Case plogObjectTypePin
```

```
' Do something specific to pin objects
```

```
Case plogObjectTypeGate
```

```
' Do something specific to gate objects
```

```
Case plogObjectTypePartType

' Do something specific to part type objects

Case Else

MsgBox "Not a SailWind Logic database object"

End Select

End Sub
```

PartType.Parent Property

This property returns the parent of the object.

Usage

Parent As Document on page 419

Arguments

None

Description

None

PartType.Selected Property

This property sets or returns whether the components of the part type are selected.

Usage

Selected As Boolean

Arguments

None

Description

The part type is considered selected when one or more components of this part type is selected. You can also select a SailWind Logic database object using the Document.SelectObjects Method or Objects.Select Method methods.

Examples

The following sample code selects part type 7400 only, assuming it exists in the open schematic, and activates the sheet on which it resides. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

ActiveDocument.SelectObjects(,,False)

 ActiveDocument.PartTypes("7400").Selected = True

 ActiveDocument.PartTypes("7400").Components(1).Gates(1).Sheet.Activate

End Sub
```

Related Topics

Document.SelectionChange Event

Pin Object

The Pin object represents a physical pin that exists in the open schematic.

The following list identifies the Pin Properties:

Pin.AlphaNumber Property

This property returns the alphanumeric of the pin.

Usage

Alpha Number As String

Arguments

None

Description

None

Examples

The following sample code retrieves the alphanumeric pin number for pin U1.1, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 Set U1_1 = ActiveDocument.Pins("U1.1")

MsgBox "Pin U1.1 has pin number " & U1_1.AlphaNumber & " in component " &
 U1_1.Component.Name

End Sub
```

Pin.Application Property

This property returns the SailWind Logic Application object.

Usage

Application As Application on page 401

Arguments

None

Description

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

Pin.Component Property

This property returns the component to which the pin belongs.

Usage

Component As Component on page 77

Arguments

None

Description

None

Examples

The following sample code displays the component to which pin U1.1 belongs, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
End Sub
```

Pin.ElectricalType Property

This property returns the gate type of the pin.

Usage

ElectricalType As PlogPinElectricalType

Arguments

None

Description

None

Examples

The following sample code retrieves the electrical type of pin U1, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
MsgBox "Pin U1.1 has type " &
 ElectricalTypeName(ActiveDocument.Pins("U1.1").ElectricalType)
```

```
End Sub
```

```
Function ElectricalTypeName(theType As Long) As String
```

```
Select Case theType
```

```
Case plogElectricalTypeUnknown
```

```
ElectricalTypeName="unknown"
```

```
Case plogElectricalTypeSource
```

```
ElectricalTypeName="source"
```

```
Case plogElectricalTypeBidirectional
```

```
ElectricalTypeName="bidirectional"
```

```
Case plogElectricalTypeOpenCollector
```

```
ElectricalTypeName="open collector"

Case plogElectricalTypeOrTieableSource

ElectricalTypeName="tiable source"

Case plogElectricalTypeTristate

ElectricalTypeName="tristate"

Case plogElectricalTypeLoad

ElectricalTypeName="load"

Case plogElectricalTypeTerminator

ElectricalTypeName="terminator"

Case plogElectricalTypePower

ElectricalTypeName="power"

Case plogElectricalTypeGround

ElectricalTypeName="ground"

Case Else

ElectricalTypeName="unknown"

End Select

End Function
```

Pin.FunctionName Property

This property returns the function name of the pin.

Usage

FunctionName As String

Arguments

None

Description

None

Examples

The following sample code retrieves the function name of pin U1, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
MsgBox "Pin U1.1 function name is " &
 ActiveDocument.Pins("U1.1").FunctionName
```

```
End Sub
```

Pin.Gate Property

This property returns the gate to which the pin belongs.

Usage

Gate As Gate on page 155

Arguments

None

Description

This property returns Nothing if the pin does not belong to any gate.

Examples

The following sample code retrieves the gate to which pin U1.1 belongs, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 Set U1_1 = ActiveDocument.Pins("U1.1")

 If U1_1.Gate Is Nothing Then

  MsgBox "Pin U1.1 does not belong to any gate"

 Else

  MsgBox "Pin U1.1 belongs to gate " & U1_1.Gate.Name

 End If

End Sub
```

Pin.GatePinName Property

This property returns the name of this pin through its gate.

Usage

GatePinName As String

Arguments

None

Description

This property returns the name consisting of Gate name and local Pin name. For example, U1-B.1, U2-B.A.

If a pin does not belong to any gate, the return value matches Pin.Name Property.

Examples

The following sample code retrieves the gate pin name for pin U1.1, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
MsgBox "Pin U1.1 has gate pin name " &
 ActiveDocument.Pins("U1.1").GatePinName
```

```
End Sub
```

Pin.Name Property

This property returns the name of the pin.

Usage

Name As String

Arguments

None

Description

For example, this property returns the string “U1.1” for pin U1.1.

This property is the default property for the Pin object.

Examples

The following sample code lists all pins, by name, in a custom dialog box. When a pin is selected in the list box, the sample selects that pin in SailWind Logic as well.

This sample uses the UserDialog Editor in the Sax Basic Engine in SailWind Logic. See Sax Basic Editor On-line Help for more information. See “Code Samples” on page 21 for more information on running this sample.

```
Dim ListPins$(10000)

Sub Main

index = 0

for Each nextPin In ActiveDocument.Pins

ListPins$(index) = nextPin.Name

index = index + 1

Next nextPin

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

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

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

```
OKButton 10,210,160,21

End Dialog

Dim dlg As UserDialog

Dialog dlg

End Sub

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

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

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

Select Case Action%

Case 2 ' Value changing or button pressed

If DlgItem$="ListBox1" Then

ActiveDocument.SelectObjects(plogObjectTypeAll, , False)

ActiveDocument.SelectObjects(plogObjectTypePin, ListPins(SuppValue%))

End If

End Select

End Function
```

Pin.Net Property

This property returns the net connected to the pin.

Usage

Net As Net on page 208

Arguments

None

Description

This property returns Nothing if the pin is not connected.

Examples

The following sample code retrieves the net connected to pin U1.1, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
End Sub
```

Pin.ObjectType Property

This property returns the type of the object.

Usage

ObjectType As PlogObjectType

Arguments

None

Description

This property returns PlogObjectTypePin.

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

This property is generally used:

- To identify the kind of SailWind Logic database objects in a heterogeneous Objects Collection Object.
- When implementing a generic routine that depends on the type of the SailWind Logic database object passed as an argument. For example:

```
Sub DoSomething(dbObject As Object)
```

```
Select Case dbObject.ObjectType
```

```
Case plogObjectTypeComponent
```

```
' Do something specific to component objects
```

```
Case plogObjectTypeNet
```

```
' Do something specific to net objects
```

```
Case plogObjectTypePin
```

```
' Do something specific to pin objects
```

```
Case plogObjectTypeGate
```

```
' Do something specific to gate objects
```

```
  Case Else
MsgBox "Not a SailWind Logic database object"
End Select
End Sub
```

Pin.Parent Property

This property returns the parent of the object.

Usage

Parent As Document on page 419

Arguments

None

Description

None

Pin.PositionX Property

This property returns the x-coordinate of the pin.

Usage

PositionX([*unit* As PlogUnit]) As Double

Arguments

	Argument	Description
	unit	[Optional on page 314] Unit in which the x coordinate is returned.

Description

This property generates an Exception if the pin is unused or signal.

Examples

The following sample code retrieves the coordinates of pin U1.1, assuming it exists in the open schematic, in current design units. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

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

```
End Sub
```

Related Topics

Pin.PositionY Property

Pin.PositionY Property

This property returns the y-coordinate of the pin.

Usage

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

Arguments

	Argument	Description
	unit	[Optional on page 314] Unit in which the y coordinate is returned.

Description

This property generates an Exception if the pin is unused or signal.

Examples

The following sample code retrieves the coordinates of pin U1.1, assuming it exists in the open schematic, in current design units. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

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

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

End Sub
```

Related Topics

Pin.PositionX Property

Pin.Selected Property

This property sets or returns whether the pin is selected.

Usage

Selected As Boolean

Arguments

None

Description

You can also select a SailWind Logic database object using theDocument.SelectObjects Method or Objects.Select Method methods.

Examples

The following sample code selects pin U1.1 only, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

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

```
ActiveDocument.Pins("U1.1").Selected = True
```

```
End Sub
```

Related Topics

Document.SelectionChange Event

Pin.SwapClass Property

This property returns the Swap Class of this pin.

Usage

SwapClass As Long

Arguments

None

Description

None

Examples

The following sample code retrieves the swap class of the pin U1.1, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 Set U1_1 = ActiveDocument.Pins("U1.1")

MsgBox "Pin U1.1 has swap class " & U1_1.SwapClass

End Sub
```

Related Topics

Gate.SwapClass Property

Sheet Object

The Sheet object represents a single SailWind Logic schematic sheet.

This object is usually retrieved from the Sheets Collection Object.

The following list identifies the Sheet Properties and Methods:

Sheet.Application Property

This property returns the SailWind Logic Application object.

Usage

Application As Application on page 401

Arguments

None

Description

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

Sheet.ChildSheets Property

This property returns the collection of child sheets in this sheet.

Usage

ChildSheets As Sheets on page 265

ChildSheets(*name* As String) As Sheet on page 265

Arguments

	Argument	Description
	name	Name of an existing sheet.

Description

When an existing sheet with name is passed to this property, it returns that sheet packaged as a Sheet Object. Otherwise, it returns the collection of all child sheets, packaged as a Sheets Collection Object.

Examples

The following sample code retrieves the number of child sheets for the active sheet using the Sheets.Count Property property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
Set shts = ActiveDocument.ActiveSheet.ChildSheets
```

```
MsgBox "There are " & shts.Count & " child sheet(s) in the " &
 ActiveDocument.ActiveSheet.Name
```

```
End Sub
```

Sheet.Components Property

This property returns the collection of all components in this sheet.

Usage

Components As Objects on page 219

Components(*name* As String) As Component on page 77

Arguments

	Argument	Description
	name	Name of an existing component.

Description

When an existing component *name* is passed to this property, it returns that Component Object. Otherwise, it returns the collection of all components in an Objects Collection Object.

Examples

The following sample code retrieves the number of components in active sheet using the Objects.Count Property property. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main

Set comps = ActiveDocument.ActiveSheet.Components

MsgBox "There are " & comps.Count & " part(s) in the sheet " &
 ActiveDocument.ActiveSheet.Name

End Sub
```

The following sample code retrieves the number of pins of component U1, assuming it exists in active sheet, using the Component.Pins Property and Objects.Count Property properties. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main

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

MsgBox "Component " & compU1.Name & " has " & compU1.Pins.Count & "
 pin(s)."

End Sub
```

Related Topics

Document.GetObjects Method

Sheet.GetObjects Method

Document.Components Property

Sheet.Gates Property

This property returns the collection of all gates in this sheet.

Usage

Gates As Objects on page 219

Gates(*name* As String) As Gate on page 155

Arguments

	Argument	Description
	name	Name of an existing gate.

Description

When an existing gate name is passed to this property, it returns that Gate Object. Otherwise, it returns the collection of all gates in an Objects Collection Object.

Examples

The following sample code retrieves the number of gates in the active sheet using the Objects.Count Property property. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main
```

```
Set gates = ActiveDocument.ActiveSheet.Gates
```

```
MsgBox "There are " & gates.Count & " gate(s) in the sheet " &
 ActiveDocument.ActiveSheet.Name
```

```
End Sub
```

Related Topics

Document.GetObjects Method

Sheet.GetObjects Method

Document.Gates Property

Sheet.Name Property

This property returns or sets the name of the sheet.

Usage

Name As String

Arguments

None

Description

This property is the default property for the Sheet object.

Examples

The following sample code retrieves the name of the active sheet. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
MsgBox "Hi, you are using " & ActiveDocument.ActiveSheet.Name
```

```
End Sub
```

Sheet.Nets Property

This property returns the collection of all nets in this sheet.

Usage

Nets As Objects on page 219

Nets(*name* As String) As Net on page 208

Arguments

	Argument	Description
	name	Name of an existing net.

Description

When an existing net *name* is passed to this property, it returns that Net Object. Otherwise, it returns the collection of all nets in anObjects Collection Object.

Examples

The following sample code retrieves the number of nets in the active sheet using the Objects.Count Property property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set nets = ActiveDocument.ActiveSheet.Nets

MsgBox "There are " & nets.Count & " net(s) in the sheet " &
 ActiveDocument.ActiveSheet.Name

End Sub
```

The following sample code retrieves the number of pins connected to net VCC, assuming it exists in the active sheet, using the Net.Pins Property and Objects.Count Property properties. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set netVCC = ActiveDocument.ActiveSheet.Nets("VCC")

MsgBox "Net " & netVCC.Name & " connects " & netVCC.Pins.Count & "
 pin(s)."

End Sub
```

Related Topics

Document.GetObjects Method

Sheet.Parent Property

This property returns the parent of the object.

Usage

Parent As Application on page 401

Arguments

None

Description

None

Sheet.ParentSheet Property

This property returns the parent sheet of this sheet.

Usage

Parent As Application on page 401

Arguments

None

Description

If this sheet is an ancestor, the return value is Nothing.

Examples

The following sample retrieves the name of the parent sheet for the active sheet. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main

 If Not ActiveDocument.ActiveSheet.ParentSheet Is Nothing Then

  MsgBox ActiveDocument.ActiveSheet.ParentSheet

 Else

  MsgBox "Sheet " & ActiveDocument.ActiveSheet.Name & " is ancestor"

 End If

End Sub
```

Sheet.PartTypes Property

This property returns the collection of all part types in this sheet.

Usage

PartTypes As Objects on page 219

PartTypes(*name* As String) As PartType on page 234

Arguments

	Argument	Description
	name	Name of an existing part type.

Description

When an existing part type *name* is passed to this property, it returns that PartType Object. Otherwise, this property returns the collection of all part types in an Objects Collection Object.

Examples

The following sample code retrieves the number of part types in the current sheet of an open schematic using the Objects.Count Property property. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main
```

```
Set pkgs = ActiveDocument.ActiveSheet.PartTypes
```

```
MsgBox "There are " & pkgs.Count & " part type(s) in the current sheet of "
 & ActiveDocument.Name
```

```
End Sub
```

Related Topics

Document.GetObjects Method

Sheet.Pins Property

This property returns the collection of all pins in this sheet.

Usage

Pins As Objects on page 219

Pins(*name* As String) As Pin on page 246

Arguments

	Argument	Description
	name	Name of an existing pin.

Description

When an existing pin *name* is passed to this property, it returns that Pin Object. Otherwise, it returns the collection of all pins in anObjects Collection Object.

Examples

The following sample code retrieves the number of pins in the active sheet using the Objects.Count Property property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set pins = ActiveDocument.ActiveSheet.Pins

MsgBox "There are " & Pins.Count & " pin(s) in the sheet " &
 ActiveDocument.ActiveSheet.Name

End Sub
```

Related Topics

Document.GetObjects Method

Sheet.Activate Method

This method activates the specified sheet.

Usage

Activate

Arguments

None

Description

None

Examples

The following sample activates the first sheet in the active document. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
 ActiveDocument.Sheets(1).Activate
```

```
End Sub
```

Sheet.AddComponent Method

This method adds a new part (all its gates) to the active sheet, and returns new part object.

Usage

AddComponent (*PartType* As String, [*RefDes* As String], [*PositionX* As Double = PlogDefaultPositionX], [*PositionY* As Double = plogDefaultPositionX], [*DeltaX* As Double = plogDefaultPositionX], [*DeltaY* As Double = plogDefaultPositionX], [*Unit* As PlogUnit]) As Component on page 77

Arguments

	Argument	Description
	PartType:	Required part type name.
	RefDes:	[Optional on page 314] Name of the part to which this Gate belongs. If RefDes is not specified, the next reference designator is used.
	PositionX:	[Optional] x-coordinate of the new gate. If PositionX is not specified, an x-coordinate is selected within the current sheet view.
	PositionY:	[Optional] y-coordinate of the new gate. If PositionY is not specified, a y-coordinate is selected within the current sheet view.
	DeltaX:	[Optional] x-coordinate increment between gates. If DeltaX is not specified, it is calculated automatically.
	DeltaY:	[Optional] y-coordinate increment between gates. If DeltaY is not specified, it is calculated automatically.
	Unit:	[Optional] Unit in which the position is given. If Unit is not specified, the current unit is used.

Return Values

New part object packaged as Component on page 77.

Description

This property generates an Exception if the arguments are not valid.

Examples

The following sample adds a new component to the active sheet and selects it. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 Set comp = ActiveDocument.ActiveSheet.AddComponent("7402", "U2")
```

```
 Comp.Selected = True
```

```
End Sub
```

Related Topics

Sheet.AddGate Method

Component.Delete Method

Sheet.AddGate Method

This method adds a new gate to the active sheet, and returns a new gate object.

Usage

AddGate(*PartType* As String, [*RefDes* As String], [*GateIndex* As Long], [*PositionX* As Double = PlogDefaultPositionX], [*PositionY* As Double = plogDefaultPositionY], [*Unit* As PlogUnit]) As Gate on page 155

Arguments

	Argument	Description
	PartType	Required part type name.
	RefDes	[Optional on page 314] Name of the part to which this Gate belongs. If RefDes is not specified, the next available reference designator is used
	GateIndex	[Optional] Gate index in the part. If GateIndex is not specified, the next available gate index is used.
	PositionX	[Optional] x-coordinate of the new Gate. If PositionX is not specified, an x-coordinate is selected within the current sheet view.
	PositionY	[Optional] y-coordinate of the new Gate. If PositionY is not specified, a y-coordinate is selected within the current sheet view.
	Unit	[Optional] Unit in which the position is given. If Unit is not specified, the current unit is used.

Return Values

New gate object packaged as Gate on page 155.

Description

This property generates an Exception if the arguments are not valid.

Examples

The following sample adds the first unused gate of the part to the active sheet and selects it. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

 Set part = ActiveDocument.Components("U1")

 Set sht = ActiveDocument.ActiveSheet
```

```
 For Each gt in part.UnusedGates

  Set newGate = sht.AddGate(part.PartType, part.Name, gt.Number)

  newGate.Selected = True

  Exit For

 Next

End Sub
```

Related Topics

Gate.Delete Method

Sheet.AddComponent Method

Sheet.GetObjects Method

This method returns a collection of SailWind Logic database objects in this sheet.

Usage

GetObjects([*type* As PlogObjectType = plogObjectTypeAll], [*name* As String], [*selected* As Boolean = False]) As Objects on page 219

Arguments

	Argument	Description
	type	[Optional on page 314] Type of SailWind Logic database object to get.
	name	[Optional] Value or name of the object(s) to get.
	selected	[Optional] True to get selected objects only. False to get all objects.

Return Values

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

Description

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

Name argument supports wildcarding (“U*”), lists of items delimited by comma (“U1, U2, R1”), ranges specified by two object names and the dash character (“U1 - U10, U12, R1 - R20”). Dash must be surrounded by spaces since the dash is a legal symbol in an object name. Only one wildcard per name is allowed and you cannot specify wildcards in a range. You can pass *name* such as “U*, R*, C1 – C100” but you cannot pass name such as “U*1*” or “C1* - C10*”.

To get all objects of the same type, use the object Sheet property. For example, to get all gates in a sheet, use Sheet.Gates Property instead of Sheet.GetObjects Method (PlogObjectType).

This property generates an Exception if the type argument is not a valid SailWind Logic database object type.

Examples

The following sample code shows different ways to use this method, retrieving the number of objects for each way using the Objects.Count Property property. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
Dim objs As Object
```

```
' Ex1: Get all objects of all types

Set objs = ActiveDocument.ActiveSheet.GetObjects

MsgBox "Ex1: " & objs.Count & " objects."

' Ex2: Get all selected objects of all types

Set objs = ActiveDocument.ActiveSheet.GetObjects(,,True)

MsgBox "Ex2: " & objs.Count & " selected objects."

' Ex3: Get all net objects

Set objs = ActiveDocument.ActiveSheet.GetObjects(plogObjectTypeNet)

MsgBox "Ex3: " & objs.Count & " net objects."

' Ex4: Get all net objects of name "VCC" (there is at least 1 of course)

Set objs = ActiveDocument.ActiveSheet.GetObjects(plogObjectTypeNet, "VCC")

MsgBox "Ex4: " & objs.Count & " VCC net objects."

' Ex5: Get all part objects which names begin with U

Set objs = ActiveDocument.ActiveSheet.GetObjects(plogObjectTypeComponent,
 "U*")

MsgBox "Ex3: " & objs.Count & " U* part objects."

End Sub
```

Related Topics

Document.SelectObjects Method

Document.Components Property

Document.Gates Property

Document.Nets Property

Document.Pins Property

Document.PartTypes Property

Document.GetObjects Method

Sheet.Components Property

Sheet.Gates Property

Sheet.Nets Property

Sheet.Pins Property

Sheet.PartTypes Property

Sheets Collection Object

The Sheets collection object is the collection of all the sheets existing in the open schematic.

This object is usually retrieved using the Document.Sheets Property property.

The following list identifies the Sheets Properties and Methods:

- Sheets.Application Property
- Sheets.Count Property
- Sheets.Item Property
- Sheets.Parent Property
- Sheets.Add Method
- Sheets.Delete Method

Sheets.Application Property

This property returns the SailWind Logic Application object.

Usage

Application As Application on page 401

Arguments

None

Description

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

Sheets.Count Property

This property returns the number of sheets.

Usage

Count As Long

Arguments

None

Description

None

Examples

The following sample code retrieves the number of sheets in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

Set shts = ActiveDocument.Sheets

MsgBox "There are " & shts.Count & " sheet(s) in " & ActiveDocument.Name

End Sub
```

Sheets.Item Property

This property returns a sheet given its index or its name.

Usage

Item(*index* As Long) As Sheet on page 265

Item(*name* As String) As Sheet on page 265

Arguments

	Argument	Description
	index	Index (in the collection) of the sheet to retrieve.
	name	Name of the sheet to retrieve.

Description

This is the default member of the Sheets collection object.

This property generates an Exception if the index or name argument is not valid.

Examples

The following sample code shows two different methods for iterating through all sheets in the open schematic. The second method is generally preferred because it is cleaner and faster. See “Code Samples” on page 21 for more information on running this sample.

```
' Method 1: Use the Object.Item property

Sub Main

Set shts = ActiveDocument.Sheets

for I=1 To shts.Count

Set thisSht = shts.Item(I)

' Do something with the sheet

Next I

End Sub
```

```
' Method 2: Do not use the Object.Item property (preferred method)

Sub Main

for Each nextSht in ActiveDocument.Sheets

' Do something with the sheet

Next nextSht

End Sub
```

Sheets.Parent Property

This property returns the parent of the object.

Usage

Parent As Document on page 419

Arguments

None

Description

None

Sheets.Add Method

This method adds a new sheet to the schematic.

Usage

Add As Sheet on page 265

Arguments

None

Return Values

The new sheet packaged as a Sheet Object.

Related Topics

Sheets.Delete Method

Sheets.Delete Method

This method deletes a sheet.

Usage

Delete(*index* As Long)

Delete(*name* As String)

Arguments

	Argument	Description
	index	Index (in the collection) of the sheet to delete.
	name	Name of the sheet to delete.

Description

This property generates an Exception in the following cases:

- If the *name* argument is not an existing sheet.
- If the *name* argument is not a valid sheet to delete.
- If the *index* argument is higher than the number of existing sheets.

Examples

The following sample code deletes sheet “MYSHEET” from the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
 ActiveDocument.Sheets.Delete("MYSHEET")
```

```
End Sub
```

Related Topics

Sheets.Add Method

View Object

The View object represents the current SailWind Logic view window, showing the active sheet in the open schematic.

This object is usually retrieved using the Document.ActiveView Property property.

The following list identifies the View Properties, Methods, and Event:

View.Application Property
View.BottomRightX Property
View.BottomRightY Property
View.Name Property
View.Parent Property
View.PointerX Property
View.PointerY Property
View.TopLeftX Property
View.TopLeftY Property
View.Pan Method
View.Refresh Method
View.SetExtents Method
View.SetExtentsToAll Method
View.SetExtentsToSheet Method
View.Change Event

View.Application Property

This property returns the SailWind Logic Application object.

Usage

Application As Application on page 401

Arguments

None

Description

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

View.BottomRightX Property

This property returns the X value of the lower right corner of the view.

Usage

BottomRightX([*unit* As PlogUnit]) As Double

Arguments

	Argument	Description
	unit	[Optional on page 314] Unit in which the X lower right value is returned.

Description

None

Examples

The following sample code retrieves the coordinates of the current view in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

x0 = Format$(ActiveDocument.ActiveView.TopLeftX, "Fixed")

y0 = Format$(ActiveDocument.ActiveView.TopLeftY, "Fixed")

x1 = Format$(ActiveDocument.ActiveView.BottomRightX, "Fixed")

y1 = Format$(ActiveDocument.ActiveView.BottomRightY, "Fixed")

MsgBox "View is (" & x0 & ", " & y0 & ") - (" & x1 & ", " & y1 & ")

End Sub
```

Related Topics

View.BottomRightY Property

View.TopLeftX Property

View.TopLeftY Property

View.BottomRightY Property

This property returns the Y value of the lower right corner of the view.

Usage

BottomRightY([*unit* As PlogUnit]) As Double

Arguments

	Argument	Description
	unit	[Optional on page 314] Unit in which the Y lower right value is returned.

Description

None

Examples

The following sample code retrieves the coordinates of the current view in the open schematic. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main

x0 = Format$(ActiveDocument.ActiveView.TopLeftX, "Fixed")

y0 = Format$(ActiveDocument.ActiveView.TopLeftY, "Fixed")

x1 = Format$(ActiveDocument.ActiveView.BottomRightX, "Fixed")

y1 = Format$(ActiveDocument.ActiveView.BottomRightY, "Fixed")

MsgBox "View is (" & x0 & ", " & y0 & ") - (" & x1 & ", " & y1 & ")

End Sub
```

Related Topics

View.BottomRightX Property

View.TopLeftX Property

View.TopLeftY Property

View.Name Property

This property returns the name of the view.

Usage

Name As String

Arguments

None

Description

For example, in SailWind Logic this function returns the string “Current View”.

This property is the default property for the View object.

Examples

The following sample code retrieves the name of the current view. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
MsgBox ActiveDocument.ActiveView.Name
```

```
End Sub
```

View.Parent Property

This property returns the parent of the object.

Usage

Parent As Document on page 419

Arguments

None

Description

None

View.PointerX Property

This property returns the pointer's X position.

Usage

PointerX (*unit* as PlogUnit) as Double

Arguments

	Argument	Description
	unit	[Optional on page 314] Unit in which the value is returned. This optional argument is plogUnitCurrent by default.

Examples

This sample also contains the PointerY property.

```
' load preview.sch

Application.ModelessCommand("s")

DlgModelessCmd.Command="s U5-A"

DlgModelessCmd.OnOk()


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 PlogUnit) as Double

Arguments

	Argument	Description
	unit	[Optional on page 314] Unit in which the value is returned. This optional argument is plogUnitCurrent by default.

Examples

This sample also contains the PointerX property.

```
' load preview.sch
```

```
Application.ModelessCommand("s")
```

```
DlgModelessCmd.Command="s U5-A"
```

```
DlgModelessCmd.OnOk()
```

```

```

```
doc = Application.ActiveDocument
```

```
view = doc.ActiveView
```

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

View.TopLeftX Property

This property returns the x-coordinate of the upper left corner of the view.

Usage

TopLeftX([*unit* As PlogUnit]) As Double

Arguments

	Argument	Description
	unit	[Optional on page 314] Unit in which the x coordinate is returned.

Description

None

Examples

The following sample code retrieves the coordinates of the current view of the open schematic. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main

x0 = Format$(ActiveDocument.ActiveView.TopLeftX, "Fixed")

y0 = Format$(ActiveDocument.ActiveView.TopLeftY, "Fixed")

x1 = Format$(ActiveDocument.ActiveView.BottomRightX, "Fixed")

y1 = Format$(ActiveDocument.ActiveView.BottomRightY, "Fixed")

MsgBox "View is (" & x0 & ", " & y0 & ") - (" & x1 & ", " & y1 & ")

End Sub
```

Related Topics

View.BottomRightX Property

View.BottomRightY Property

View.TopLeftY Property

View.TopLeftY Property

This property returns the y-coordinate of the upper left corner of the view.

Usage

TopLeftY([*unit* As PlogUnit]) As Double

Arguments

	Argument	Description
	unit	[Optional on page 314] Unit in which the Y upper left value is returned.

Description

None

Examples

The following sample code retrieves the coordinates of the current view of the open schematic. See "Code Samples" on page 21 for more information on running this sample.

```
Sub Main

x0 = Format$(ActiveDocument.ActiveView.TopLeftX, "Fixed")

y0 = Format$(ActiveDocument.ActiveView.TopLeftY, "Fixed")

x1 = Format$(ActiveDocument.ActiveView.BottomRightX, "Fixed")

y1 = Format$(ActiveDocument.ActiveView.BottomRightY, "Fixed")

MsgBox "View is (" & x0 & ", " & y0 & ") - (" & x1 & ", " & y1 & ")

End Sub
```

Related Topics

View.BottomRightX Property

View.BottomRightY Property

View.TopLeftX Property

View.Pan Method

This method pans the view to a specified location.

Usage

Pan(*x* As Double, *y* As Double, [*unit* As PlogUnit])

Arguments

	Argument	Description
	x	x-coordinate of the point to which to pan.
	y	y-coordinate of the point to which to pan.
	unit	[Optional on page 314] Unit in which the X and Y values are given.

Description

None

Examples

The following sample code centers the view to the location of the first gate of the component U1, assuming it exists in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

x = ActiveDocument.Components("U1").Gate(1).PositionX

y = ActiveDocument.Components("U1").Gate(1).PositionY

ActiveDocument.ActiveView.Pan(x,y)

End Sub
```

Related Topics

View.Change Event

View.Refresh Method

This method refreshes the view.

Usage

Refresh

Arguments

None

Description

None

Examples

The following sample code redraws the SailWind Logic workspace. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.ActiveView.Refresh
```

```
End Sub
```

View.SetExtents Method

This method sets the view extents.

Usage

SetExtents(*tlx* As Double, *tly* As Double, *brx* As Double, *bry* As Double, [*unit* As PlogUnit])

Arguments

	Argument	Description
	tlx	x-coordinate of the top left corner of the new view.
	tly	y-coordinate of the top left corner of the new view
	brx	x-coordinate of the bottom right corner of the new view.
	bry	y-coordinate of the bottom right corner of the new view
	unit	[Optional on page 314] Unit in which the tlx, tly, brx, and bry values are given.

Description

None

Examples

The following sample code sets the view to the extents of all pins connected to first net in the active sheet. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main

xMin = 100000000.0

yMin = 100000000.0

xMax = -100000000.0

yMax = -100000000.0

ActiveDocument.ActiveSheet.Net(1).Selected =True

for Each nextPin In ActiveDocument.ActiveSheet.Net(1).Pin

 'ensure pin is visible i.e. belongs to gate
```

```
 If Not nextPin.Gate Is Nothing Then
  'ensure gate is used
  If Not nextPin.Gate.Sheet Is Nothing Then
   If nextPin.PositionX < xMin Then xMin = nextPin.PositionX
   If nextPin.PositionX > xMax Then xMax = nextPin.PositionX
   If nextPin.PositionY < yMin Then yMin = nextPin.PositionY
   If nextPin.PositionY > yMax Then yMax = nextPin.PositionY
   End If
  End If
Next nextPin
ActiveDocument.ActiveView.SetExtents(xMin, yMin, xMax, yMax)
End Sub
```

Related Topics

View.SetExtentsToAll Method

View.SetExtentsToSheet Method

View.Change Event

View.SetExtentsToAll Method

This method sets the view extents to all objects in the schematic.

Usage

SetExtentsToAll ()

Arguments

None

Description

None

Examples

The following sample code sets the view extents to all objects in the open schematic. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.ActiveView.SetExtentsToAll
```

```
End Sub
```

Related Topics

View.SetExtents Method

View.SetExtentsToSheet Method

View.Change Event

View.SetExtentsToSheet Method

This method sets the view extents to the active sheet extents.

Usage

SetExtentsToSheet ()

Arguments

None

Description

None

Examples

The following sample code sets the view extents to the active sheet. See “Code Samples” on page 21 for more information on running this sample.

```
Sub Main
```

```
ActiveDocument.ActiveView.SetExtentsToSheet
```

```
End Sub
```

Related Topics

View.SetExtents Method

View.SetExtentsToAll Method

View.Change Event

View.Change Event

This event occurs when the current view changes.

Usage

View_Change ()

Arguments

None

Description

This event occurs after the current view changes or active sheet changes.

Related Topics

View.Pan Method

View.SetExtents Method

View.SetExtentsToSheet Method

View.SetExtentsToAll Method

Constants

The following constants are available.

PlogDocumentColor

Possible values are:

plogDocumentColorBackground	= 0,
plogDocumentColorSelection	= 1,
plogDocumentColorConnection	= 2,
plogDocumentColorBus	= 3,
plogDocumentColorLine	= 4,
plogDocumentColorPart	= 5,
plogDocumentColorHierarchicalComp	= 6,
plogDocumentColorText	= 7,
plogDocumentColorTextBox	= 8,

plogDocumentColorRefDes	= 9,
plogDocumentColorRefDesBox	= 10,
plogDocumentColorPartType	= 11,
plogDocumentColorPartTypeBox	= 12,
plogDocumentColorPartText	= 13,
plogDocumentColorPartTextBox	= 14,
plogDocumentColorPinNumber	= 15,
plogDocumentColorPinNumberBox	= 16,
plogDocumentColorNetName	= 17,
plogDocumentColorNetNameBox	= 18,
plogDocumentColorField	= 19,
plogDocumentColorFieldBox	= 20,

PlogObjectType

SailWind Logic database object type. Possible values are:

- plogObjectTypeUnknown = 0 -

  The server may return this value to indicate an invalid object. The client may use this value when trying to work with empty object collections.

- plogObjectTypeComponent = 1
- plogObjectTypeNet = 2
- plogObjectTypePin = 3
- plogObjectTypeGate = 4
- plogObjectTypePartType = 8
- plogObjectTypeLibrary = 18
- plogObjectTypeLibraryItem = 19
- plogObjectTypeApplication = 20

- plogObjectTypeAll = 9999 -

  All SailWind Logic Automation database object types, including the Component, Net, Pin, Gate, and PartType objects.

PlogUnit

SailWind Logic unit types. Possible values are:

Unit types	Description
plogUnitCurrent = 0	Current unit in use in the SailWind Logic.
plogUnitDatabase = 1	Internal SailWind Logic database unit.
plogUnitMils = 2	Mils unit (1/1000th of an inch).
plogUnitInch = 3	Inch unit.
plogUnitMetric = 4	Metric unit (1/1000th of a meter).
plogUnitDrawArea = 99	Internal SailWind Logic workspace unit. Never use this value.

PlogGridType

SailWind Logic grid type. Possible values are:

Grid types	Description
plogGridNone = 0	None.
plogGridDesign = 1	SailWind Logic design grid.
plogGridDisplay = 3	SailWind Logic display grid.
plogGridAll = 9999	All SailWind Logic grids.

PlogPinElectricalType

SailWind Logic gate electrical types. Possible values are:

- plogElectricalTypeUnknown = 0
- plogElectricalTypeSource = 1
- plogElectricalTypeBidirectional = 2
- plogElectricalTypeOpenCollector = 3
- plogElectricalTypeOrTieableSource = 4

- plogElectricalTypeTristate = 5
- plogElectricalTypeLoad = 6
- plogElectricalTypeTerminator = 7
- plogElectricalTypePower = 8
- plogElectricalTypeGround = 9

PlogASCIIVersion

SailWind Logic ASCII versions. Possible values are:

- plogASCIIVerCurrent = 0
- plogASCIIVer1_2 = 1
- plogASCIIVer3_0 = 2
- plogASCIIVer3_5 = 3

PlogNetListVersion

SailWind Logic/SailWind Layout netlist versions. Possible values are:

- plogPowerPCBNetListVerCurrent = 0
- plogPowerPCBNetListVer2_1 = 2
- plogPowerPCBNetListVer3_0 = 3
- plogPowerPCBNetListVer3_5 = 4

PlogGateVisibility

SailWind Logic gate visibility items. Possible values are:

- plogAttrVisibility = 0
- plogAttrNameVisibility = 1
- plogRefDesVisibility = 2
- plogPartTypeVisibility = 3
- plogPinNumberVisibility = 4

- plogPinNameVisibility = 5
- plogPCBDecalVisibility = 6
- plogPCBDecalNameVisibility = 7

PlogDefaultPosition

SailWind Logic default position coordinates. Possible values are:

plogDefaultPositionX = &H80000000

plogDefaultPositionY = &H80000000

PlogMeasureFormat

SailWind Logic default position coordinates. Possible values are:

plogMeasureFormatStandard = 0

plogMeasureFormatCurrent = 1

plogMeasureFormatShort = 2

plogMeasureFormatLong = 3

PlogLibraryItemType

SailWind Logic library item types. Possible values are:

- plogLibraryItemTypePartType = 0
- plogLibraryItemTypeDecal = 1
- plogLibraryItemTypeLogicDrawing = 2
- plogLibraryItemTypeDrawing = 3
- plogLibraryItemTypeAll = 9999

Optional Arguments

An optional argument is an argument to a property or a method that can be omitted because a default value is used in SailWind Logic. The default value is the value that is statistically most used or one that is most representative for that argument.

For example, if the method M([arg1], [arg2]) has both arg1 and arg2 arguments set as optional, you can call the method M in four different ways as shown below:

Table 1. Optional Argument Examples

Argument Examples	Description
M()	The default values of both arg1 and arg2 are passed.
M(<*value1*>)	Arg1's <*value1*> and arg2's default value are passed.
M(, <*value2*>)	Arg1's default value and arg2's <*value2*> are passed.
M(<*value1*>, <*value2*>)	Both arg1's <*value1*> and arg2's <*value2*> are passed.

Exception

An Automation exception is special notification from the SailWind Logic server that signals the Automation client of an error. For example, if a Basic script is trying to delete an attribute that doesn't exist, SailWind Logic generates an exception.

When the script receives the exception, the following occurs:

- If an exception handler is implemented using the Basic On Error statement, the flow of execution of the client code is rerouted to the exception handler.
- If an exception handler in not implemented, the client's default handler is called. In all Basic interpreters, the default handler is a break point at the line that generated the exception.

Variant

A Variant is a data type that can contain or represent any kind of data, such as a Boolean value, an Integer value, a Long value, a Double value, a String value, or an Array.

The Variant data type is used in SailWind Logic Automation in two different situations:

- When the argument of a property or method is a data that can be represented as different values and value types. For example, a specific object, in a collection of objects, is referenced by its index in the collection (a Long value) or by its name (a String value).
- When the argument or the return value of a property or a method is a complex type, not explicitly defined by SailWind Logic Basic object. For example, an array of points is represented as a Variant.

Chapter 3
Macros

Macros are recordings of your session of working that can be reused.

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 the program.

This topoic includes the following descriptions:

Recording a Session

You can record your session of working in SailWind Logic.

Procedure

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

   “-log”

   Ensure you type a space between “powerlogic.exe” and the “-log” command
5. Click **OK**. When you run the program, a log is created.

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 Logic.
2. Right-click the shortcut and click the **Properties** popup menu item.
3. Click the **Shortcut** tab and click in the “Target” box.
4. At the end of the existing shortcut, type the following:

   “-log:[path and file name]”

   Ensure you type a space between “powerlogic.exe” and the “-log” command.
5. Click **OK**. When you run the program, a log is created in the specified location using the specified name.

Running a Macro When You Start the Program

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

Procedure

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

   “-run=[path and file name]”

   For example:

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

   Ensure you type a space between "powerlogic.exe" and the "-run" command. Alternatively, you can use a / instead of the hyphen (such as, /run=mymacro.mcr).
5. Click **OK**. When you run the program, the specified macro runs as soon as the program starts.

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
- Objects, 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
- Double
- Object

Value types are converted to each other according to these rules, see Table 2.

Table 2. & Operator Arguments

From...To...	Logical	Numeric	String
Null	False	0	Empty string
Logical	None	0 if False, 1 if True	0 if False, 1 if True
Numeric	False if 0, True if not 0	None	String representation of the numeric value
String	Converted to Numeric first	If the beginning of the string can be interpreted as a number, then the value of the number is used. Otherwise it is 0.	None

Numeric

The Numeric value represents a floating-point number.

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

Logical

The Logical value can be True or False.

String

The String value represents a character string.

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.

Syntax

```
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.

Syntax

```
result = number1 * number2
```

Arguments

The * operator has these arguments:

result	Required Any numeric variable.
number1	Required Any expression. If the expression value is not numeric, it is converted to a numeric value.
number2	Required Any expression. If the expression value is not numeric, it is converted to a numeric value.

Examples

```
X = y * z
```

+ Operator

Sums two numbers.

Syntax

```
result = expression1 + expression2
```

Arguments

The + operator syntax has these arguments:

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

Note:
When you use the + operator, you may not be able to determine whether addition or string concatenation will occur. To force string concatenation, use the & operator instead. This will eliminate ambiguity and provide self-documenting code.

The following table describes the behavior of the + operator for the three combinations of types:

Table 3. + Operator Behavior

Arguments	Action
Both expressions are numeric	Add
Both expressions are string	Concatenate
One expression is numeric and the other is a string	Add

Examples

```
X = y + z
```

/ Operator

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

Syntax

```
result = number1 / number2
```

Arguments

The / operator has these arguments:

result	Required Any numeric variable.
number1	Required Any expression. If the expression value is not numeric, it is converted to a numeric value.
number2	Required Any expression. If the expression value is not numeric, it is converted to a numeric value.

Examples

```
x = y/z
```

- Operator

Finds the difference between two numbers or indicates the negative value of a numeric expression.

Syntax

```
result = number1 - number2
```

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

```
-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
```

And

```
-x
```

= Operator

Assigns a value to a variable or property.

Syntax

variable **=** value

Arguments

The = operator has these arguments:

variable	Can only be a variable or a writable property. Can be a simple scalar variable or an element of an array.
Value	Any numeric expression, string expression, literal, or constant

Examples

```
a = 1
```

^ Operator

Raises a number to the power of an exponent.

Syntax

```
result = number ^ exponent
```

Arguments

The ^ operator has these arguments:

result	Required Any numeric variable
number	Required Any expression
exponent	Required Any numeric expression A number can be negative only if the exponent is an integer.

When more than one exponentiation is performed in a single expression, the ^ operator is evaluated as it is encountered from left to right.

Examples

```
x = y ^ z
```

And Operator

Performs a logical conjunction of two expressions.

Syntax

result = expression1 **And** expression2

Arguments

The And operator has these arguments:

result	Required Any numeric variable.
expression1	Required Any expression. If the expression value is not logical, it is converted to a logical value.
expression2	Required Any expression. If the expression value is not logical, it is converted to a logical value.

The following table illustrates how the result is determine:

Table 4. 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.

Syntax

```
result = expression1 comparisonoperator expression2
```

Arguments

Comparison operators have these arguments:

result	Required Any variable.
expression1	Required Any expression.
expression2	Required Any expression.
comparisonoperator	Required Can be any comparison operator. For more information, see the table below.

The following table lists the comparison operators and the conditions that determine whether result is True, False, or Null:

Table 5. 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.

Syntax

```
result = number1 Mod number2
```

Arguments

The Mod operator has these arguments:

result	Required Any numeric variable
number1	Required Any numeric expression
number2	Required Any numeric expression

Examples

```
x = y Mod z
```

Not Operator

Performs a logical negation on an expression.

Syntax

```
result = Not expression
```

Arguments

The Not operator has these arguments:

result	Required Any numeric variable.
expression	Required Any expression. If the expression value is not logical, it is converted to a logical value.

The following table illustrates how result is determined:

Table 6. 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.

Syntax

```
result = expression1 Or expression2
```

Arguments

The Or operator has these arguments:

result	Required Any numeric variable.
expression1	Required Any expression. If the expression value is not logical, it is converted to a logical value.
expression2	Required Any expression. If the expression value is not logical, it is converted to a logical value.

The following table illustrates how result is determined:

Table 7. 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.

Syntax

```
[result =] expression1 Xor expression2
```

Arguments

The Xor operator has these arguments:

result	Optional Any numeric variable.
expression1	Required Any expression. If the expression value is not logical, it is converted to a logical value.
expression2	Required Any expression. If the expression value is not logical, it is converted to a logical value.

The following table illustrates how result is determined:

Table 8. 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
- Close
- Dim
- Do...Loop
- For-Next
- Function
- If...Then...Else statement
- Input #
- Modal
- Open
- Print #
- ReDim
- Set
- Sub
- While...Wend
- Width #

Call

Transfers control to a sub procedure or function procedure.

Syntax

```
[Call] name [argumentlist]
```

When you specify the Call keyword to call a procedure that requires arguments, you must enclose argumentlist in parentheses. See example below.

Arguments

The Call statement has these arguments:

Call	Optional keyword You are not required to use the Call keyword when calling a procedure. If you omit the Call keyword, you must also omit the parentheses around argumentlist. If you use either the Call syntax to call any intrinsic or user-defined function, the function's return value is discarded.
name	Required Name of the procedure to call
argumentlist	Optional Comma-delimited list of variables, arrays, or expressions to pass to the procedure.

Examples

```
Call MyProc(0)
```

Related Topics

Statements

Close

Concludes input/output (I/O) to a file opened using the Open statement. When you close files that were opened for Output or Append, the final buffer of output is written to the operating system buffer for that file; All buffer space associated with the closed file is released; and the association of a file with its file number ends.

Syntax

```
Close [filenumberlist]
```

Arguments

The Close statement has this argument:

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

Syntax

```
Dim varname[([subscripts])] [,varname[([subscripts])]] . . .
```

Arguments

The Dim statement has these arguments:

varname	Required The variable name follows standard variable-naming conventions.
subscripts	Optional Dimensions of an array variable. The subscripts argument uses the following syntax: [lower To] upper [, [lower To] upper] . . . When not explicitly stated, the lower bound is zero.

You can also use the Dim statement, with empty parentheses, to declare a dynamic array. After declaring a dynamic array, use the ReDim statement to define the number of dimensions and elements in the array.

Examples

```
Dim x(10), y(20)
```

Related Topics

Statements

Do...Loop

Repeats a block of statements while a condition is True or until a condition becomes True.

Syntax

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

And

```
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.

Syntax

```
For counter = start To end [Step step]
[statements]
[Exit For]
[statements]
Next [counter]
```

Arguments

The For-Next statement has these arguments:

counter	Required Numeric variable used as a loop counter. ***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 9. 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.

Usage

```
Function name [(arglist)]
```

```
[statements]
```

```
[name = expression]
```

```
[Exit Function]
```

```
[statements]
```

```
[name = expression]
```

```
End Function
```

The Exit Function statement causes an immediate exit from a function procedure. Program execution continues with the statement that follows the statement that called the function procedure. You can add any number of Exit Function statements in a function procedure.

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 10. 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.

Syntax

```
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.

Syntax

```
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.

Syntax

```
modal variablename
```

Arguments

The modal keyword has these arguments:

variablename	Required The name of the variable you want to make accessible.

Examples

```
modal docname
```

Open

Enables input and output to a file. You must open a file before performing any I/O operation on it. The Open opeartion allocates an I/O buffer for the file and determines the mode of access to use with the buffer. If the file is already opened by another process and the specified type of access is not allowed, the Open operation fails and an error occurs.

Syntax

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

Syntax

```
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 11 describes the outputlist syntax elements:

Table 11. Print # Statement outputlist Syntax

Setting	Description
Spc(n)	Inserts space characters in the output, where ***n*** is the number of spaces to insert.
Tab(n)	Positions the insertion point at an absolute column number, where ***n*** is the column number. Use Tab with no argument to position the insertion point at the beginning of the next print zone. Because Print # writes an image of the data to the file, you must delimit the data so it prints correctly. If you use Tab with no arguments to move the print position to the next print zone, Print # also writes the spaces between print fields to the file.

Table 11. Print # Statement outputlist Syntax (continued)

expression	Numeric or string expressions to print. You can separate multiple expressions with either a space or a semicolon.
charpos	Specifies the insertion point for the next character. If you omit charpos, the next character prints on the next line. Use a semicolon to position the insertion point immediately after the last displayed character.

Data written with Print # is usually read from a file with Input #.

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 use ReDim repeatedly to change the number of elements and dimensions in an array.

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

Syntax

```
ReDim [Preserve] varname(subscripts) [, varname(subscripts)] . . .
```

Arguments

The ReDim statement has these arguments:

Preserve	Optional Keyword used to preserve the data in an existing array when you change the size of the last dimension.
varname	Required Name of the variable; follows standard variable-naming conventions.
subscripts	Required Dimensions of an array variable The subscripts argument uses the following syntax: `[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.

Syntax

```
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.

Usage

```
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 12 describes the arglist syntax elements:

Table 12. Sub 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.

Note:
You cannot define a sub procedure inside any other procedure such as a function or another sub procedure.

For specific information about calling sub procedures, see the Call statement.

Related Topics

Statements

While...Wend

Executes a series of statements as long as a given condition is True.

Syntax

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

Arguments

The While Wend statement has these arguments:

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.

Syntax

```
Width #filenumber, width
```

Arguments

The Width # statement has these arguments:

filenumber	Required Any valid file number
width	Required Numeric expression in the range 0 to 255, inclusive. Indicates how many characters appear on a line before a new line starts. If width equals 0, there is no limit to the length of a line. The default value for width is 0.

Examples

```
Width #2, 100
```

Related Topics

Statements

Open

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.

Syntax

```
Asc(string)
```

Arguments

The Atn function has this argument:

string	Required Any valid string expression If the string contains no characters, a run-time error occurs.

Examples

```
i = Asc("abc")
```

Atn

This function returns a Double specifying the arctangent of a number.

For more information see Double.

The range of the result is - pi/2 to pi/2 radians. To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

The Atn function takes the ratio of two sides of a right triangle and returns the corresponding angle in radians. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

Atn is the inverse trigonometric function of Tan, which takes an angle as its argument and returns the ratio of two sides of a right triangle. Do not confuse Atn with cotangent, which is the inverse of a tangent (1/tangent).

Usage

```
Atn(number)
```

The Atn function has this argument:

number	Required Any expression. If the expression is not Logical, it is converted to a Logical type.

Examples

```
f = Atn(2)
```

Chr

This function returns a string containing the character associated with the specified character code.

Syntax

```
Chr(charcode)
```

Arguments

The Chr function has this argument:

charcode	Required A long that identifies a character. Values from 0 to 31 are standard, nonprintable ASCII codes. For example, Chr(10) returns a linefeed character. The normal range for charcode is 0 to 255, inclusive.

Examples

```
c = chr(64)
```

Command

This function returns the command line used to start the program, including the path to the executable file and any subsequent arguments.

Syntax

```
Command
```

When you launch the program from the command line, the command line is available to Macro scripts.

Arguments

None

Examples

Assuming that the program is launched by the following command:

```
BlazeRouter log:logfile.log preview.pcb
```

The Command function returns:

```
"C:\<install_folder>\<version>\Programs\BlazeRouter.exe log:logfile.log
 preview.pcb"
```

Cos

This function returns a Double specifying the cosine of an angle.

For more information see Double.

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)
```

The Cos function has this argument:

number	Required A double or any valid numeric expression expressing an angle in radians

Examples

```
x = Cos(1.57)
```

CreateObject

This function creates and returns a reference to an ActiveX object.

Syntax

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

Arguments

The CreateObject function syntax has these arguments:

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.

Syntax

```
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.

Syntax

```
Dir(pathname)
```

Arguments

The Dir function has this argument:

pathname	Optional String expression that specifies a file name. May also include folder and drive names. A zero-length string ("") is returned if pathname is not found.

Dir supports the use of multiple character (*) and single character (?) wildcards to specify multiple files.

Examples

To get the first file in the c: drive root:

```
Dir("C:\*.*")
```

To get the next file in the same path:

```
Dir
```

To start another search in C:\SailWind Projects\Samples:

```
Dir("C:\SailWind Projects\Samples\*.*")
```

DoEvents

This function passes control to the operating system. The operating system returns control after it finishes processing the events in its queue.

Syntax

```
DoEvents()
```

DoEvents may be useful if the macro is performing long calculations. Inserting DoEvents calls every second or more may prevent the accumulation of unprocessed events in the queue.

Arguments

None

Environ

This function returns the string associated with an operating system environment variable.

Syntax

```
Environ [(envstring)]
```

Arguments

The Environ function has this argument:

Envstring	Optional String expression containing the name of an environment variable.

If envstring can't be found in the environment-string table, a zero-length string ("") is returned.

Environ returns the text assigned to the specified envstring; that is, the text which follows the equal sign (=) in the environment-string table for that environment variable.

Examples

```
s = Environ ("path")
```

Eof

This function returns an integer, which represents the Boolean value True, when the end of a file opened for random or sequential input has been reached.

Use the Eof function to avoid the error generated by attempting to get input after the end of a file.

Table 13 lists the Eof function returned values:

Table 13. Eof Function Returned Values

Condition	Eof function returns
Before end of file is reached	False
After end of file is reached	True
File opened for random access and Get statement is able to read an entire record	False
File opened for random access and Get statement is not able to read an entire record	True
File opened for binary access and Get statement is able to read an entire record	False

Table 13. Eof Function Returned Values (continued)

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

Usage

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

The Eof function has this argument:

filenumber	Optional An integer containing any valid file number

Examples

```
If Eof (1) then ....
```

Exp

This function returns a Double specifying e (the base of natural logarithms) raised to a power. The constant *e* is approximately 2.718282.

For more information see Double.

Note:
The Exp function complements the Log function and may be referred to as the antilogarithm.

Usage

```
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.

Syntax

```
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.

Syntax

```
GetTmpFileName(string)
```

Arguments

The GetTmpFileName function has this argument:

string	Required A string expression

Examples

```
s = GetTmpFileName("d:\tmp")
```

InStr

This function returns the position of the first occurrence of one string within another string.

Syntax

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

Arguments

The InStr function has these arguments:

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 14. InStr Function Return Values

If	InStr returns
string1 is zero-length	0
string2 is zero-length	start
string2 is not found	0
string2 is found within string1	position at which a match is found
start > string	0

Examples

```
i = InStr(1,"cbc","c")
```

InStrRev

This function returns the position of an occurrence of one string within another, from the end of the string.

Syntax

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

Arguments

The InStrRev function has these arguments:

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 15. 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.

Syntax

```
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).

Syntax

```
Len(string)
```

Arguments

The Len function has this argument:

string	Required Any valid string expression

Examples

The following sample code returns 4.

```
Len("abcd")
```

Mid

This function returns a specified number of characters from a string.

Syntax

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

Arguments

The Mid function has these arguments:

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.

Syntax

```
MkDir (path)
```

Arguments

The MkDir function has this argument:

path	Required A string expression that identifies the folder to create. The path may include the drive. When no drive is specified, MkDir creates the new folder on the current drive.

Examples

```
MkDir ("D:\newdir")
```

MoveFile

This function moves the file identified by path1 argument to the location specified in path2.

Syntax

```
MoveFile(path1, path2)
```

Arguments

The MoveFile function has these arguments:

path1	Required A string expression
path2	Required A string expression

Examples

```
MoveFile("C:\data.bin","D:\")
```

MsgBox

This function displays a message in a dialog box, waits for the user to click a button, and returns an integer indicating which button the user clicked.

Usage

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

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.

The *buttons* argument settings are:

Table 16. 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.

When adding numbers to create a final value for the buttons argument, use only one number from each group.

- 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.

Returns

Table 17. MsgBox Return Values

Constant	Value	Description
mbOK	1	OK
mbCancel	2	Cancel*
mbAbort	3	Abort
mbRetry	4	Retry
mbIgnore	5	Ignore
mbYes	6	Yes
mbNo	7	No

*If the dialog box displays a Cancel button, pressing Esc has the same effect as clicking Cancel.

Examples

```
MsgBox("Hello",mbOK, "This is a message box")
```

Right

This function returns a specified number of characters from the right side of a string.

Syntax

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

Arguments

The Right function has these arguments:

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.

The Sin function takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse.

The result lies in the range -1 to 1.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

Usage

```
Sin(number)
```

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 Print # statement or the Print method to position output.

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

Usage

```
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.

Syntax

```
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.

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

Usage

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

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 left-most 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.

Syntax

```
Val(string)
```

Arguments

The Val function has this argument:

string	Required Any valid string expression

The Val function stops reading the string at the first character it can't recognize as part of a number. However, the function recognizes the radix prefixes &O (for octal) and &H (for hexadecimal). Blanks, tabs, and linefeed characters are stripped from the argument. The Val function recognizes only the period (.) as a valid decimal separator.

Examples

```
i=Val("123")
```

Automation Support

The macro engine in the program supports Automation through its automation objects. After an object is created using CreateObject( ) or connected to using GetObject( ), the object's methods may be called using the usual syntax.

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

Related Topics

CreateObject

GetObject

Dialog Box Controls

The macro language of 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.

Syntax

```
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 18. CheckBox.State istate Values

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

Table 18. CheckBox.State istate Values (continued)

2	Indeterminate

Value Property

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

Syntax

```
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 19. 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.

Syntax

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

Arguments

The State method has this argument:

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

SetCheck for CheckListBox

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

Syntax

```
CheckListBox.SetCheck(string)
```

Arguments

The SetCheck method has this argument:

string	Required A string expression, containing the list of the items to check. All the items that do not appear in the list are not checked.

ListCount Property for CheckListBox

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

Syntax

```
CheckListBox.ListCount
```

SelCount Property for CheckListBox

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

Syntax

```
CheckListBox.SelCount
```

Selected Property for CheckListBox

This method returns or sets the selection status of an item in the check list box. This property is an array of Boolean values with the same number of items as the List property.

Syntax

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

Check Property for CheckListBox

This method returns or sets the checked status of an item in a check list box. This property is an array of Boolean values with the same number of items as the list property.

Syntax

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

Text Property for CheckListBox

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

Syntax

```
CheckListBox.Text
```

ComboBox

This control represents a combo box on a dialog box.

Select

This method sets the selection state of the combo box.

Syntax

```
ComboBox.Select(string)
```

Arguments

The Select method has this argument:

string	Required A string expression, containing a string to select in the combo box

Examples

```
ActiveLayer.Select("Top")
```

Edit for ComboBox

This method sets the edit state of the combo box.

Syntax

```
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 for ComboBox

This method returns or sets the combo box text.

Syntax

```
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 for ComboBox

This method returns or sets the items contained in the combo box list. This list is a string array in which each element is a list item.

Syntax

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

SelStart Property for ComboBox

This method returns or sets the starting point of the selected text. If no text is selected, this method indicates the position of the insertion point.

Syntax

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

SelLength Property for ComboBox

This method returns or sets the number of characters selected.

Syntax

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

SelText Property for ComboBox

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

Syntax

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

Arguments

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

EditBox

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.

Syntax

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

Arguments

State has this argument:

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

Text Property for EditBox

This method returns or sets the edit box text.

Syntax

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

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

SelStart Property for EditBox

This method returns or sets the starting point of selected text. If no text is selected this method indicates the position of the insertion point.

Syntax

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

SelLength Property for EditBox

This method returns or sets the number of characters selected.

Syntax

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

SelText Property for EditBox

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

Syntax

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

Arguments

The SelText method has this argument:

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

GridControl

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.

Syntax

```
ListBox.State(string)
```

Arguments

The State method has this argument:

string	Required A string expression, containing the item numbers to select. All the items that do not appear in the list are not selected.

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.

Syntax

```
ListBox.List
```

ListCount Property

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

Syntax

```
ListBox.ListCount
```

SelCount Property

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

Syntax

```
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.

Syntax

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

Text Property

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

Syntax

```
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.

Syntax

```
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.

Syntax

```
RadioBox.State(iState)
```

Arguments

The State method has this argument:

iState	Required A numeric expression, representing the position of an option button to check. -1 means that no button is selected.

Value Property for RadioBox

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

Syntax

```
RadioBox.Value[=iState]
```

Arguments

Value Property has this argument:

iState	Required A numeric expression, representing the position of an option button to check. -1 means that no button is selected.

SliderControl

This control represents a slider control on a dialog box.

State

This method sets the state of the slider.

Syntax

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

Arguments

State has this argument:

iState	Required A numeric expression

Value Property for SliderControl

This method returns or sets the current slider position.

Syntax

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

Arguments

Value Property has this argument:

val	Required A numeric expression

SpinButton

This control represents a spin button on a dialog box.

State

This method sets the state of the spin button.

Syntax

```
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.

Syntax

```
TabControl.State(iState)
```

Arguments

State has this argument:

iState	Required A numeric expression, which represents the position of a tab.

Examples

```
dlg.Control("Tab").State(3)
```

Value Property

This method returns or sets the current tab position.

Syntax

```
TabControl.Value[=tab]
```

Arguments

Value Property has this argument:

tab	Required Either a numeric expression representing the tab position or a string expression representing the tab caption

TreeItem

This control represents a tree item on a dialog box.

Select for TreeItem

This method sets the selection state of the tree item.

Syntax

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

Arguments

The Select method has this argument:

flag	Required A numeric expression

flag may have one of the following values:

Table 20. TreeItem.Select flag Values

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

Examples

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

Expand for TreeItem

This method sets the expand state of the tree item.

Syntax

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

Arguments

Expand has this argument:

flag	Required

	A numeric expression

flag may have one of the following values:

Table 21. TreeItem.Expand flag Values

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

Examples

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

Focus for TreeItem

This method sets the tree item focus to the item.

Syntax

```
TreeItem.Focus()
```

Examples

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

TreeView

This control represents a Tree View on a dialog box.

Item

This method returns a TreeItem object.

Syntax

```
TreeView.Item(itemname)
```

Arguments

Item has this argument:

itemname	Required A string expression representing the name of the item

Examples

```
item = tree.Item("Net Objects\Nets\end")
```

BeginDrag

This method emulates dragging selected items off the tree.

Syntax

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

Arguments

BeginDrag has this argument:

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

Copy

This method copies the selected item to the Clipboard.

Syntax

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

Arguments

Copy has this argument:

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

Drop

This method emulates dropping the dragged items onto an item.

Syntax

```
TreeView.Drop(itemname)
```

Arguments

The Drop method has this argument:

itemname	Required A string expression representing the name of the item onto which to drop the dragged items

Examples

```
tree.Drop("Net Objects\Net classes")
```

Paste

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

Syntax

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

Arguments

Paste has this argument:

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

CreateNewItem

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

Syntax

```
TreeView.CreateNewItem(itemname)
```

Arguments

CreateNewItem has this argument:

itemname	Required A string expression representing the name of the item to create

Internal Macro Objects

The 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.

Syntax

```
Application.CreateNewDocument
```

Arguments

None

ExecuteCommand

This method executes one of the commands of the program.

Syntax

```
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

```
Application.ExecuteCommand("Open", "C:\SailWind Projects\preview.pcb")
```

Help

This method invokes the Help.

Syntax

```
Application.Help()
```

Arguments

None

HelpContents

This method returns the Help contents in the Help Contents window.

Syntax

```
Application.HelpContents
```

Arguments

None

Examples

```
Set var = Application.HelpContents
```

HelpPane

This method returns the Help window.

Syntax

```
Application.HelpPane
```

Arguments

None

Examples

```
Set var = Application.HelpPane
```

OpenCustomizeDialog

This method opens the Customize modal dialog box.

Syntax

```
Application.OpenCustomizeDialog()
```

Arguments

None

OpenDocument

This method opens an existing document identified by the path argument.

Syntax

```
Application.OpenDocument(path)
```

Arguments

The OpenDocument method has this argument:

path	Required A string expression containing the path of the document to open

Examples

```
Application.OpenDocument("C:\SailWind Projects\preview.pcb") \
```

OpenOptionsDialog

This method opens the Options modeless dialog box.

Syntax

```
Application.OpenOptionsDialog()
```

Arguments

None

OpenPropertiesDialog

This method opens the Properties modeless dialog box.

Syntax

```
Application.OpenPropertiesDialog()
```

Arguments

None

Quit

This method exits the application.

Syntax

```
Application.Quit()
```

Arguments

None

RunMacro

This method executes one of the program commands.

Syntax

```
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:

- Control
- Focus
- CloseHelpPane
- OpenHelpPane
- ShowHelpFor

Control

This method returns a dialog box control.

Syntax

```
Dialog.Control(controlname)
```

Arguments

Control has this argument:

controlname	Required A string expression representing the name of the control

Examples

This example returns the OK button.

```
set obj = dlg.Control("OK")
```

Related Topics

Focus

Focus

This method sets focus to a dialog box control.

Syntax

```
Dialog.Focus(controlname)
```

Arguments

The Focus method has this argument:

controlname	Required A string expression representing the name of the control

Related Topics

Control

CloseHelpPane

This method closes the open help pane in a dialog box.

Syntax

```
Dialog.CloseHelpPane
```

Arguments

None

Examples

```
Dialog.CloseHelpPane
```

OpenHelpPane

This method displays the help pane for the dialog box.

Syntax

```
Dialog.OpenHelpPane
```

Arguments

None

Examples

```
Dialog.OpenHelpPane
```

ShowHelpFor

This method displays help for the specified control.

Syntax

```
Dialog.ShowHelpFor(controlname)
```

Arguments

The ShowHelpFor method has this argument:

controlname	Required The name of the control

Examples

This example displays Help for the Apply button.

```
Dialog.ShowHelpFor("Apply")
```

Document Object

The Document object represents any currently loaded design.

This object has the following methods:

- Print
- PrintSetup
- RepeatLastAction
- Save
- SaveAs

Print

This method prints the document.

Syntax

```
Document.Print()
```

Arguments

None

PrintSetup

This method opens the Print Setup dialog box.

Syntax

```
Document.PrintSetup()
```

Arguments

None

RepeatLastAction

This method repeats the last action performed during the current session.

Syntax

```
Document.RepeatLastAction()
```

Arguments

None

Save

This method saves the document, if it has been modified.

Syntax

```
Document.Save()
```

Arguments

None

SaveAs

This method saves the document to a user-defined name or path location.

Usage

```
Document.SaveAs(path)
```

The SaveAs method has this argument:

path	Required A string expression representing the path to which to save the document

Arguments

None

HelpContents Object

The HelpContents object represents the Help Contents window. The Item property finds the location of the Help contents item in the contents tree.

Syntax

```
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.

Syntax

```
Item.Location
```

Arguments

None

Examples

In this example, the item_loc variable is assigned a value of "*its:C:\<install_folder>\<version>\Documentation\Router\BlazeRouter.chm::/fileops/To_Restore_Files.htm*".

```
Set item = Application.HelpContents.Item("File Operations\To Restore
 Files")
```

```
item_loc = item.Location
```

Name

This property returns the name of the item.

Syntax

```
Item.Name
```

Arguments

None

Examples

In this example, the item_name variable is assigned the value of "To Restore Files."

```
Set item = Application.HelpContents.Item("File Operations\To Restore
 Files")
```

```
item_name = item.Name
```

Select

This method selects the item.

Syntax

```
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.

Syntax

```
Item.SubItem(pos)
```

Arguments

None

Examples

In this example, the item_name variable is assigned value of "To Restore Files".

```
Set item = Application.HelpContents.Item("File Operations").SubItem(3)
```

```
item_name = item.Name
```

SubItemCount

This property returns the number of sub items within the item.

Syntax

```
Item.SubItemCount
```

Arguments

None

Examples

In this example, the *count* variable is assigned a value of 10.

```
Set item = Application.HelpContents.Item("File Operations")
```

```
count = item.SubItemCount
```

HelpPane Object

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.

Syntax

```
MainView.ActiveLayer
```

Arguments

None

Examples

```
set layerCombo = MainView.ActiveLayer
```

ToggleFullScreen

This method turns the Full Screen mode on.

Syntax

```
MainView.ToggleFullScreen()
```

Arguments

None

MouseDown

This method emulates pressing a mouse button.

Syntax

```
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 22. 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.

Syntax

```
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 23. 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.

Syntax

```
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 24. 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.

Syntax

```
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 25. 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)
```

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

The ***button*** argument may contain one or more of the following values and modifiers:

Table 26. 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)

Arguments

None

Print

This method prints the current view.

Syntax

```
MainView.Print()
```

Arguments

None

PrintPreview

This method turns the Print Preview mode on.

Syntax

```
MainView.PrintPreview()
```

Arguments

None

Chapter 4
SPICE Netlist Attribute Glossary

The attributes described in this section can apply to your analog design. All analog attributes are prefixed with sim.analog. when added. For example, you add the sim.analog.model attribute to a part when using a model file. The format is not case sensitive.

A2D

The A2D net attribute specifies a model name (to represent an absent interface component) on an analog–to–digital signal connection in ViewSim/AD.

Placement

On the schematic, attached to a net segment.

Example

A2D=TTLADMOD specifies the TTLADMOD interface component.

ABSTOL

The ABSTOL attribute on the OPTIONS analog report generator sets the accuracy, in amps, of current measurements.

Placement

On the symbol.

Default

The default attribute is ABSTOL=1pa.

Example

ABSTOL=1E–6A indicates that current measurements are accurate to within 1 micro amp. ABSTOL=1E–12A indicates that current measurements are accurate to within 1 pico amp.

AC/DC/TRAN

The AC/DC/TRAN attribute defines what type of analysis (DC, AC, transient) is being done when the Monte–Carlo analysis is run.

Placement

On the symbol.

Attribute Value

Use one of the following keywords in the value field:

Keyword	Definition
AC	AC analysis
DC	DC analysis
TRAN	Transient analysis

Example

AC/DC/TRAN=AC indicates that an AC analysis is to be performed each time simulation is run.

ACCT

The ACCT on the OPTIONS analog report generator outputs a summary of the analysis.

Placement

On the symbol.

Attribute Value

Do not use an attribute value (or equal sign character) when specifying this attribute.

AD

The AD attribute defines the size of the drain area in a FET.

Placement

On the symbol.

The ORDER Attribute

You must identify this attribute in the ORDER attribute for interpretation.

Example

AD=2u indicates a drain area of 2 microns.

AMPLITUDE

The AMPLITUDE attribute sets maximum voltage or current swing on the SFFM and ISFFM analog symbols and the maximum current swing on the ISIN symbol.

Placement

On the symbol.

Attribute Value

The value of this attribute is substituted for “amp” in the waveform formula. Refer to the waveform formula under I and V in the PSpice Help file.

Example

AMPLITUDE=100E-3A indicates a maximum current swing of 100 milliamps.

AS

The AS attribute defines the size of the source area of a FET.

Placement

On the symbol.

The ORDER Attribute

You must identify this attribute in the ORDER attribute for interpretation.

Example

AS=1u indicates a source area of 1 micron.

BULK

The BULK attribute is attached to a 3–pin FET symbol. It specifies where the fourth pin or bulk substrate of a four–pin transistor will be connected. This attribute must be used to include all of the transistor pins in the simulation file.

Placement

On the symbol.

CHGTOL

The CHGTOL attribute reports the best accuracy, in coulombs, of charges on the OPTIONS analog report generator.

For more information, refer to the HSpice or PSpice Help files.

Placement

On the symbol.

Default

The default attribute is CHGTOL=.01pC.

Example

CHGTOL=1E–3C indicates that measurements of charge are accurate to within 1 millicoulomb.
CHGTOL=1E–6C indicates that measurements of charge are accurate to within 1 microcoulomb.

CPTIME

The CPTIME attribute on the OPTIONS analog report generator provides the time, in seconds, allowed for this run.

For more information, refer to the HSpice or PSpice Help files.

Placement

On the symbol.

Default

The default attribute is CPTIME=1E6.

Example

CPTIME=3600 indicates that this run can take up to one hour to complete.

CURRENT

The CURRENT attribute specifies the number of amps present on the IDC analog symbol.

Placement

On the symbol.

The ORDER Attribute

You must identify this attribute in the ORDER attribute for interpretation.

Example

CURRENT=25E-3 indicates that there are 25 milliamps present on the IDC component.

The VOLTAGE Attribute

Use the VOLTAGE attribute on the DC component.

D2A

The D2A net attribute specifies a model name (to represent an absent interface component) on an digital–to–analog signal connection in The Simulator/AD.

Placement

On the schematic, attached to a net segment.

Example

D2A=TTLDAMOD specifies the TTLDAMOD interface component.

DAMPING

The DAMPING attribute specifies the damping factor of the waveform on the SIN and ISIN analog symbols.

Placement

On the symbol.

Attribute Value

The value of this attribute is substituted for “df” in the waveform formula. Refer to the waveform formula under I and V in the PSpice Help file.

The ORDER Attribute

You must identify this attribute in the ORDER attribute for interpretation.

Example

DAMPING=1 indicates a damping factor of 1.

DC

The DC attribute indicates the amount of voltage across the AMMETER or UAMMETER symbol.

Placement

On the symbol

Example

DC=15 indicates that there are 15 volts across this component.

Measuring Current

If you are using a METER type symbol to measure current (such as, AMMETER or UAMMETER), specify the DC and PROBE attributes as follows:

DC=0

PROBE=I

ENDFREQ

The ENDFREQ attribute specifies the final frequency of a frequency sweep on the BODEPLOT analog report generator.

Placement

On the symbol.

Attribute Value

The attribute value must be greater than the value of the STARTFREQ attribute.

Example

ENDFREQ=100MEGHz indicates a final sweep frequency of 100 Megahertz.

ENDVAL

The ENDVAL attribute specifies the end point of the sweep on the SWEEP analog report generator.

Placement

On the symbol.

Attribute Value

Because the sweep can be in either direction, the value of ENDVAL can be greater or less than the value of the STARTVAL attribute.

Example

ENDVAL=5mA indicates an end sweep current of 5 milliamps.

FREQ_CARRIER

The FREQ_CARRIER attribute specifies the carrier frequency of the waveform on the SFFM and ISFFM analog symbols.

For more information, refer to the HSpice or PSpice Help file.

Placement

On the symbol.

Attribute Value

The value of this attribute is substituted for “fc” in the waveform formula. Refer to the waveform formula under I and V in the PSpice Help file.

Example

FREQ_CARRIER=101MEGHz indicates a carrier frequency of 101 Megahertz.

FREQ_SIGNAL

The FREQ_SIGNAL attribute specifies the modulation frequency of the waveform on the SFFM and ISFFM analog symbols.

For more information, refer to the HSpice or PSpice Help file.

Placement

On the symbol.

Attribute Value

The value of this attribute is substituted for “fm” in the waveform formula. Refer to the waveform formula under I and V in the PSpice Help file.

Example

FREQ_SIGNAL=20KHz indicates a modulation frequency of 20 Kilohertz.

FREQUENCY

The FREQUENCY attribute specifies the frequency of the waveform on the SIN and ISIN analog symbols.

For more information, refer to the HSpice or PSpice Help file.

Placement

On the symbol.

Attribute Value

The value of this attribute is substituted for “freq” in the waveform formula. Refer to the waveform formula under I and V in the PSpice Help file.

Example

FREQUENCY=5MEGHz indicates a frequency of 5 Megahertz.

GAIN

The GAIN attribute specifies the amplification caused by the CCCS, VCVS, VCCS, and CCVS analog components.

Placement

On the symbol.

Example

GAIN=3E–3 indicates that this component multiplies a circuit parameter by 0.003.

GENERATOR

The GENERATOR attribute identifies the type of analog analysis that you wish to perform on the analog circuit. The attribute value identifies the Spice statement that the analog netlister will insert into the Spice circuit file, **.cir*.

Notes

All analog test instruments require the GENERATOR attribute as well as the CLASS and PREFIX attributes.

PowerWorks comes supplied with symbols for each analysis type. These symbols possess the appropriate GENERATOR attribute value.

Placement

On the test instrument symbol.

Attribute Value

The GENERATOR attribute accepts the following values:

NOTE: Precede each value with a period (for example, .SENS, not SENS).

The GENERATOR attribute can accept multiple values. For example:

```
GENERATOR = .AC, .DC, .FOUR
```

Table 27. Generator Attribute - Possible Values

Attribute Value	Instrument Name	Symbol Name	Analysis Description
.AC	Bodeplot	BODEPLOT.1	Sweeps each independent AC source in a design through a series of frequencies and calculates the frequency response of the circuit at each of the swept frequencies.
.CSDF\PROBE	CSDF	CSDF.1	Common Simulation Data Format; this is ViewSpice specific; it causes ViewSpice to produce an ASCII file which contains all voltages and currents in a circuit; analyzes voltage, current, reliability.
.DC	DC Sweep	SWEEP.1	Sweeps a circuit's input source through a series of voltages or currents and calculates the bias point of the circuit at each of the specified voltages or currents.
.FOUR	Fourier	FOURIER.1	Performs a decomposition into Fourier components of the result of a transient analysis. Note that this is not the same as the FFT (Fast Fourier Transform) performed by ViewTrace.
.LIB	Library	LIB.1	Indicates that one or more ViewSpice or HSPICE models are defined in an external library.
.LOADBIAS	Loadbias (ViewSpice only)	LBIAS.1	Loads the bias point file saved with SAVEBIAS. This is specific to ViewSpice.
.MC	Monte–Carlo	MC.1	Calculates the effects of device tolerances on the design's performance; also simulates the

Table 27. Generator Attribute - Possible Values (continued)

Attribute Value	Instrument Name	Symbol Name	Analysis Description
			manufacture of multiple copies of the design. This is specific to ViewSpice.
.NOISE	Noise	NOISE.1	Estimates the input and output noise of a circuit.
.OP	Bias point	OP.1	Calculates the operating conditions of devices in the design and lists these calculations in the simulation output file.
.OPTIONS	Options	OPTIONS.1	Lets you set parameters to control how a specified analysis is run or how its results appear.
.PARAM	Parameter (ViewSpice only)	.PARAM	
.SAVEBIAS	Savebias (ViewSpice only)	SBIAS.1	Saves bias points at specified times into a specified file. This is specific to ViewSpice.
.SENS	Sensitivity	SENSITIVITY.1	Calculates the small signal sensitivity of each node voltage to each of the active devices in the design.
.STEP	Step (ViewSpice only)	STEP.1	Re–simulates the circuit with different parameter values. Useful for running multiple transient or AC analyses. This is specific to ViewSpice.
.TEMP	Temperature	TEMP.1	Sets the circuit to a specific Centigrade temperature so an analysis can calculate how the circuit operates at that temperature.
.TF	Transfer Function	TRANSFCT.1	Calculates small signal gain, input impedance and output impedance of a circuit with respect to the circuit's bias point.
.TRAN	Transient Sweep or Oscilloscope	TRANS.1	Also called transient analysis; generates visual and text reports about the circuit's operation over a specified period of time.
.WCASE	Worst Case (ViewSpice only)	WCASE.1	Does sensitivity and worst case analysis of the circuit by making a number of runs, varying one parameter per run. The sensitivity of the output is calculated for each varied parameter and a final run is done with all parameters varied to produce a worst case. This is specific to PSpice.

Example

GENERATOR=.TRAN indicates that you want to include transient analysis information as part of your analog simulation input.

GMIN

The GMIN attribute on the OPTIONS analog report generator specifies the minimum conductance, in mhos, used for any branch.

For more information, refer to the HSpice or PSpice Help file.

Placement

On the symbol.

Example

GMIN=3 indicates a minimum conductance of 3 mhos.

I1, I2, I3, ...

The I1 attribute specifies the amount of current generated by the IPWL analog symbol at the time period specified by the T1 attribute.

Placement

On the symbol.

The ORDER Attribute

You must identify this attribute in the ORDER attribute of the IPWL for interpretation.

The T1 Attribute

The T1 attribute specifies the time period at which the current is generated.

Example

I1=2E–3A indicates that 2 milliamps are generated at the T1 time period.

IC

The IC attribute sets up initial conditions on nets and analog components. You can assign an initial voltage to a node, or you can assign initial conditions across a circuit element.

For more information, refer to the HSpice or PSpice Help file.

Placement

On the schematic, attached to a net or component.

The ORDER Attribute

You must identify this attribute in the ORDER attribute for interpretation.

INCREMENT

The INCREMENT attribute specifies the interval during the sweep at which analyses are taken on the SWEEP analog report generator.

Placement

On the symbol.

Attribute Value

The attribute value must be greater than 0.

Example

INCREMENT=.1V indicates an analysis of the circuit is made each time source voltage is changed by 0.1 volts.

INIT

The INIT attribute on the OSCILLOSCOPE analog report generator instructs the simulator to use specified initial conditions.

For more information, refer to the HSpice or PSpice Help file.

Placement

On the symbol.

Attribute Value

The attribute value is INIT=UIC.

INITIAL

The INITIAL attribute specifies the starting current of the waveform on the IPULSE analog symbol.

Placement

On the symbol.

The ORDER Attribute

You must identify this attribute in the ORDER attribute for interpretation.

Example

INITIAL=1E–6 indicates that the initial value of the IPULSE component is one microamp.

INPUTSOURCE

The INPUTSOURCE attribute specifies the REFDES or label name of the input source component against which gain is measured on the XFERFUNC analog report generator.

For more information, refer to the HSpice or PSpice Help files.

Placement

On the symbol.

Example

INPUTSOURCE=VIN indicates that gain is to be measured against the component VIN.

ITL1

The ITL1 attribute on the OPTIONS analog report generator specifies the total number of times that the system can perform DC or BIAS POINT analysis iterations if the circuit does not contain a NODESET attribute.

For more information, refer to the HSpice or PSpice Help files.

Placement

On the symbol.

Default

The default attribute value is ITL1=100.

Example

ITL1=300. This increase changes the limit of iterations and can help to achieve DC convergence if a failure occurs.

ITL2

The ITL2 attribute on the OPTIONS analog report generator specifies the maximum number of times that, with a NODESET attribute present, the system can perform DC or BIAS POINT analysis iterations.

For more information, refer to the HSpice or PSpice Help file.

Placement

On the symbol.

Default

The default attribute value is ITL2=20.

Example

ITL2=50 indicates that, with a NODESET attribute present, the system can perform DC or BIAS POINT analysis up to 50 times.

ITL4

The ITL4 attribute on the OPTIONS analog report generator specifies the maximum number of iteration times transient analysis will be performed for any one point in analysis.

For more information, refer to the HSpice or PSpice Help file.

Placement

On the symbol.

Default

The default attribute value is ITL4=10.

ITL5

The ITL5 attribute on the OPTIONS analog report generator specifies the maximum number of iteration times that transient analysis will be performed for all points in analysis.

For more information, refer to the HSpice or PSpice Help file.

Placement

On the symbol.

Default

The default attribute value is ITL5=5000.

Example

ITL5=100 indicates that transient analysis can be performed 100 times. ITL5=0 indicates that there is no limit on the number of times transient analysis can be performed.

L

The L attribute defines the length of an analog FET channel in meters.

Placement

On the symbol.

The ORDER Attribute

You must identify this attribute in the ORDER attribute for interpretation.

Example

L=5u indicates that the channel is 5 microns long.

LABEL

The LABEL attribute specifies a unique name you want to assign to this analog source component in order to differentiate it from similar components in a project schematic. Attaching a LABEL attribute to a component has the same effect as creating a label for the component using the **Add > Label** command.

Placement

On the symbol.

Example

The attribute LABEL=VIN assigns the name VIN to this component instance of the symbol, just as if you had attached a label of VIN to the component instance.

LIBRARY

You can only place the LIBRARY attribute on the LIB symbol. The LIBRARY attribute identifies the Spice model library that the LIB symbol is meant to represent.

LIMPTS

The LIMPTS attribute on the OPTIONS analog report generator specifies the maximum number of data points allowed in any analysis.

Placement

On the symbol.

Example

LIMPTS=2000 indicates that a maximum of 2000 points in any print table or plot can be included in an analysis.

LIN/OCT/DEC

The LIN/OCT/DEC attribute specifies the type of sweep to be performed on the BODEPLOT analog report generator.

Placement

On the symbol.

Attribute Value

Use one of the following keywords in the value field to define sweep **type:**

Keyword	Definition

DEC	Decade
LIN	Linear
OCT	Octave

Example

LIN/OCT/DEC=DEC indicates that simulation results are measured in decades.

LIST

The LIST attribute on the OPTIONS analog report generator directs the system to output a summary of components.

For more information, refer to the HSpice or PSpice Help file.

Placement

On the symbol.

Attribute Value

Do not use an attribute value (or equal sign character) when specifying this attribute.

MOD_INDEX

The MOD_INDEX attribute specifies the multiplier of modulated frequency on SFFM and ISFFM analog symbols.

Placement

On the symbol.

Attribute Value

The value of this attribute is substituted for “mod” in the waveform formula. Refer to the waveform formula under I and V in the PSpice Help file.

Example

MOD_INDEX=1 indicates a modulated frequency of 1.

MODEL

The MODEL attribute defines which model file the system uses to simulate this component. Simulation *.mod* files are stored in the Analog Models subdirectory. Each SPICE model must have a separate .mod file. A model file can have subcircuit sections, but multiple models cannot be stored in one file since there is no mechanism for the software to extract the different models from within the file.

Device Types

RAM, ROM

Analog Placement

On the symbol.

Analog Example

MODEL=Q2N3904 tells the system to simulate this component using the “q2n3904.”

NO_PINS

The NO_PINS attribute overrides a symbol’s PINORDER attribute. Using the NO_PINS attributes allows you use the ORDER attribute instead of the PINORDER attribute to determine analog component pin order.

Placement

On the symbol.

Attribute Value

The NO_PINS attribute accepts 1 or TRUE as its value.

The PINORDER and ORDER Attributes

Normally, the PINORDER attribute controls the default sequence. You can also use the ORDER attribute to specify pin sequence. If you specify the NO_PINS attribute on the symbol, this overrides the PINORDER attribute and the system uses the pin sequence in the ORDER attribute.

NODE

The NODE attribute on the OPTIONS analog report generator directs the system to output a list or a table of nets.

For more information, refer to the HSpice or PSpice Help file.

Placement

On the symbol.

Attribute Value

Do not use an attribute value (or equal sign character) when specifying this attribute.

NODESET

The NODESET attribute defines what voltage you think is present on the net. The simulator uses your estimate as an initial condition for determining the actual voltage on the net during calculation of the bias point.

For more information, refer to the HSpice or PSpice Help file.

Placement

On the schematic, attached to a net.

Example

NODESET=3 tells the simulator to guess the initial voltage as 3 volts. This adds the following line in the *.cir* file:

.NODESET V([**net_name**])=**n**V

NOECHO

The NOECHO attribute on the OPTIONS analog report generator directs the system not to list the input file.

For more information, refer to the PSpice Help file.

Placement

On the symbol.

Attribute Value

Do not use an attribute value (or equal sign character) when specifying this attribute.

Example

This attribute adds the following line in the *.cir* file:

.OPTIONS NOECHO

NOISESOURCE

The NOISESOURCE attribute specifies the REFDES or label name of the input signal and noise source component against which noise is measured on the NOISE analog report generator.

Placement

On the symbol.

Example

NOISESOURCE=VIN indicates that noise is to be measured against the amount of noise at the component VIN.

NOPAGE

The NOPAGE attribute on the OPTIONS analog report generator directs the system not to paginate, or print a header/banner for each major section of output.

Placement

On the symbol.

Attribute Value

Do not use an attribute value (or equal sign character) when specifying this attribute.

NUMDGT

The NUMDGT attribute on the OPTIONS analog report generator specifies the number of significant digits printed in each column in a table (each column corresponds to one output variable).

Placement

On the symbol.

Example

NUMDGT=6 indicates that each output variable is precise to 6 digits.

NUMPOINTS

The NUMPOINTS attribute specifies the number of data points in each sweep analysis on the BODEPLOT analog report generator.

Placement

On the symbol.

The LIN/OCT/DEC Attribute

The LIN/OCT/DEC attribute specifies the sweep type that simulation results are measured in.

Example

NUMPOINTS=10 indicates that each sweep analysis contains 10 data points per octave if the LIN/OCT/DEC=OCT attribute exists.

NUMRUN

The NUMRUN attribute on the MC analog report generator specifies the number of times the DC, AC, or transient analysis is run during a Monte Carlo analysis.

Placement

On the symbol.

Example

NUMRUN=10 indicates that, in addition to nominal simulation, 10 passes of DC, AC, or transient analysis are to be run.

OFFSET

The OFFSET attribute specifies the location on the Y axis at which the DC voltage or current is measured on the SFFM and ISFFM analog symbols. This attribute is also used to specify the location on the Y axis at which the current is measured on the ISIN analog symbol.

Placement

On the symbol.

Attribute Value

The value of this attribute is substituted for “voff” in the waveform formula. Refer to the waveform formula under I and V in the PSpice Help file.

Example

OFFSET=50mA indicates that the power generated by this component is measured around the Y axis that corresponds to 50 milliamps.

OPTS

The OPTS attribute on the OPTIONS analog report generator lists the values of all of the options.

For more information, refer to the HSpice Help file.

Placement

On the symbol.

Attribute Value

Do not use an attribute value (or equal sign character) when specifying this attribute.

ORDER

The ORDER attribute specifies what information is included in the simulation file and how the information is formatted. The simulation file includes only those attributes listed in the value field of the ORDER attribute.

Use the ORDER attribute in your design only where necessary. For example, attach ORDER to a symbol to include the symbol definition in the simulation file.

Each SPICE model must have a separate .mod file. A model file can have subcircuit sections, but multiple models cannot be stored in one file since there is no mechanism for the software to extract the different models from within the file.

Placement

On the symbol.

Attribute Value

You can alter the way that attributes are entered into the simulation file by placing one of the following characters at the end of the attribute’s name keyword in the ORDER attribute:

Table 28. Order Attribute - Characters for Controlling Attribute Values

Character	Definition
$	Only the attribute value is entered
=	The entire attribute is entered
&	Inserts the SPICE name of the referenced component, where the reference is by internal name. SPICE name of the referenced component, where the reference is by internal name.
[	Writes the name or number of the net connected to that pin. See the .MC report generator for some examples. This can be used with the attribute NO_PINS = TRUE, which disables default output of node numbers after the component name.
“ ”	Defines literals which are always available as output

It is not necessary to insert spaces in between items listed in the ORDER attribute unless you want the spaces to appear in the SPICE input file. This is because spaces are read as part of the attribute string. For example:

```
ORDER=K1$ K2$K3$" V("K4$")"
```

Where K1=1, K2=2, K3=3, and K4=4, this produces in the SPICE input file:

```
1 23 V(4)
```

Special Case Keywords

The SPICE Netlister recognizes two special case keywords: NOT4ECAD and WL. When it finds the keyword NOT4ECAD within an element:

- for Dracula: it removes this element from the deck.
- for Spice: it removes only the keyword, “NOT4ECAD”

When the SPICE Netlister finds the keyword WL=wval/lval, two parameters are created: W=wval and L=lval. The WL_SPLIT parameter set in the wspice.cfg file enables this feature.

Example

With the BULK=VDD and MODEL=PSS attributes, the ORDER=BULK$MODEL= attribute causes the following entry in the simulation file:

```
VDD MODEL=PSS
```

OUT

The OUT attribute specifies the number of characters per line of text in a print table on the WIDTH analog report generator.

Placement

On the symbol.

Attribute Value

Use 80 or 132 in the value field to specify 80 or 132 characters, respectively.

Example

OUT=80 (default value) indicates that 80 characters are allowed in each line of a print table.

OUTPUT

The OUTPUT attribute specifies how much output information is generated from Monte–Carlo analysis on the MC analog report generator.

For more information, refer to the HSpice or PSpice Help file.

Placement

On the symbol.

Example

OUTPUT=ALL indicates that all of the output from every Monte–Carlo analysis should be generated.

PARNAM

The PARNAM is used by the SPICE Netlister to allow you to pass parameters through a hierarchical netlist. The PARNAM attribute references the component's .SUBCKT line in the hierarchical netlist and adds the desired parameter to the subcircuit call. The following conditions apply to the PARMAN attribute:

You do not need to use the PARNAM attribute to generate a hierarchical netlist. The PARNAM attribute simply adds additional information to the .cir file.

If you are creating a flat netlist the SPICE Netlister ignores the PARNAM attribute. The PARNAM attribute takes as its value the parameters you want to pass. The parameters must be defined in the format acceptable to the analog simulator you intend to use.

Placement

On the schematic, attached to a component.

Example

If an analog symbol intended to be simulated using SPICE and possessed the following attributes:

```
REFDES = X1
| Assume two nodes named In and OUT
PARNAM = PARAMS: @W=1.0U @L=1.0U
ORDER = PARAMS: @W= @L=
W = 5.0U
L = 5.0U
```

The Spice Netlister would produce the following subcircuit calls in the .cir file:

```
.SUBCIT FOO IN OUT PARAMS: W=1.0U L=1.0U
```

```
X1 IN OUT FOO PARAMS: W=5.0u L=5.0U
```

If an analog symbol intended to be simulated using HSpice possessed the following attributes:

```
REFDES = X1
| Assume two nodes named In and OUT
PARNAM = @W=1.0U @L=1.0U
ORDER = @W= @L=
W = 5.0U
L = 5.0U
```

The Spice Netlister would produce the following subcircuit calls in the .cir file:

```
.SUBCIT FOO IN OUT W=1.0U L=1
```

PHASE

The PHASE attribute specifies relative phase, in degrees, of multiple waveforms on AC, IAC, SIN, and ISIN analog symbols.

Placement

On the symbol.

Attribute Value

The value of this attribute is substituted for “phase” in the waveform formula. Refer to the waveform formula under I and V in the PSpice Help file.

Example

PHASE=90 indicates a phase of 90 degrees.

PINORDER

The PINORDER attribute determines the order in which nets attached to exact pins are listed in the SPICE netlist. Only those nets attached to exact pins (pins defined in the pinorder attribute) are listed in the SPICE netlist.

Separators for pins in this attribute value are: " ,\t\n"

Placement

On the symbol.

Attribute Value

The attribute value is the pin name label, not the pin number.

Additional Information

Normally, the PINORDER attribute controls the default sequence. You can also use the ORDER attribute to specify pin sequence. If you specify the NO_PINS attribute on the symbol, this overrides the PINORDER attribute and the system uses the pin sequence in the ORDER attribute.

PIVREL

The PIVREL attribute on the OPTIONS analog report generator specifies the relative magnitude required for pivot in the matrix solution.

Placement

On the symbol.

Additional Information

For more information, refer to the Spice Help file or a math text book for the subjects:

- Solutions to systems of equations
- Gaussian elimination
- Partial pivoting

PREFIX

The PREFIX attribute value is used by the SPICE Netlister to uniquely identify each analog component in a SPICE compatible format. The SPICE Netlister prefixes each component's PREFIX attribute value to its REFDES attribute value and writes the entire string to the .cir file. The .cir is used as input to ViewSpice the analog simulator. This attribute is required for all analog symbols.

Placement

On the symbol.

Attribute Value

The following prefixes are defaults for the corresponding types of components:

Table 29. Prefix Attribute - List of Prefixes

Keyword	Definition
B	Gallium arsenide field effect transistor
C	Capacitor
D	Diode
E	Voltage–controlled voltage source
F	Current–controlled current source
G	Voltage–controlled current source
H	Current–controlled voltage source
I	Independent current source
J	Junction field effect transistor
K	Transformer
L	Inductor
M	Metal–oxide silicon field effect transistor
O	Digital output
Q	Bipolar transistor
R	Resistor
S	Voltage–controlled switch
T	Transmission line
V	Independent voltage source
W	Sub–circuit

Refer to appropriate simulator documentation to determine which prefix to use in the value field of this attribute.

Example

For example is a component has a REFDES = U2 and a PREFIX = R the component is written into the .cir file as RU2 which tells the simulator that component U2 is a resistor.

PROBE

The PROBE attribute specifies if the analog component is measuring current (I) or voltage (V).

Placement

On the schematic, attached to a net or symbol.

Attribute Value

Use I or V in the value field to specify current or voltage, respectively. Attach any of the following keyword suffixes to the letter I or V to precisely define measurement:

Keyword	Definition
DB	Decibel
M	Magnitude
P	Phase

Duplicate Attributes

You cannot attach the same attribute to an object more than once, except for the PROBE attribute on components and net/bus segments.

Example

PROBE=IP indicates that current phase is being measured.

The DC Attribute

If you are using a METER type symbol to measure current (such as AMMETER or UAMMETER), specify 0 for the DC attribute value.

PULSED

The PULSED attribute specifies the maximum current swing of the IPULSE analog symbol.

Placement

On the symbol.

RELTOL

The RELTOL attribute on the OPTIONS analog report generator specifies relative accuracy of voltage and current measurements.

Placement

On the symbol.

Default

The default attribute value is RELTOL=0.001.

Example

RELTOL=.001 indicates that relative voltage and current measurements are accurate to 0.1%.

SOURCENAME

The SOURCENAME attribute specifies the REFDES or label name of the component that is the source of the circuit being swept on the SWEEP analog report generator.

Placement

On the symbol.

Example

SOURCENAME=VIN indicates that a DC sweep analysis is made by varying the value of the component VIN.

STARTFREQ

The STARTFREQ attribute specifies the initial frequency of a frequency sweep on the BODEPLOT analog report generator.

Placement

On the symbol.

Attribute Value

Value must be greater than 0 and less than the value of the attribute.

Example

STARTFREQ=15KHz indicates a starting sweep frequency of 15 kilohertz.

STARTVAL

The STARTVAL attribute specifies the initial voltage or current of the sweep on the SWEEP analog report generator.

Placement

On the symbol.

Attribute Value

The sweep can be in either direction; therefore, the value of STARTVAL can be negative or positive.

Example

STARTVAL=–2.5V indicates a starting sweep voltage of negative 2.5 volts.

TDELAY

The TDELAY attribute specifies the time it takes a PULSE, IPULSE, SIN, and ISIN component to begin a regular waveform.

Placement

On the symbol.

Attribute Value

The value of this attribute is substituted for “td” in the waveform formula. Refer to the waveform formula under I and V in the PSpice Help file.

Example

TDELAY=10ns indicates that the start of a regular waveform is delayed by 10 nanoseconds.

TEMP1, TEMP2, TEMP3, ...

The TEMP1 attribute specifies a centigrade temperature on the TEMP analog report generator to apply to the circuit during simulation. When the specified analysis is performed (such as, DC SWEEP or MONTE–CARLO), the circuit is set to the specified temperature.

Placement

On the symbol.

Attribute Value

If multiple values are defined, the system simulates the circuit at each of the temperatures specified.

Example

TEMP1=125 indicates that the circuit should be simulated at a temperature of 125 degrees centigrade.

Other TEMP Attributes

Each additional TEMP attribute (TEMP2, TEMP3, TEMP4, etc.) defines an additional centigrade temperature to be applied to the circuit during the specified analysis.

TFALL

The TFALL attribute specifies the time it takes to transition from PULSED to INITIAL or VPULSED to VINITIAL on the IPULSE and PULSE analog symbols.

Placement

On the symbol.

Attribute Value

The value of this attribute is substituted for “tf” in the waveform formula. Refer to the waveform formula under I and V in the PSpice Help file.

Example

TFALL=10ns indicates a transition time of 10 nanoseconds on the PULSE and IPULSE analog symbols.

TFINAL

The TFINAL attribute on the TRANS analog report generator specifies the total time for which simulation results are measured.

Placement

On the symbol.

Example

TFINAL=100ns indicates that simulation results will be measured for 100 nanoseconds on the TRANS analog report generator.

TI, T2, T3, ... T9

The T1 attribute specifies the time period at which the voltage or current defined by the V1 or I1 attribute is generated on the PWL or IPWL analog symbol.

Placement

On the symbol.

The ORDER Attribute

You must identify this attribute in the ORDER attribute of the PWL or IPWL for interpretation.

Example

T1=10ns indicates that V1 voltage or I1 current is generated at 10 nanoseconds.

The V1 and I1 Attributes

Use the V1 attribute to specify the voltage on the PWL symbol. Use the I1 attribute to specify the current on the IPWL symbol.

Specifying Time Periods

Each additional set of T1 and V1, or T1 and I1, defines an additional time period and amount of voltage or current generated.

Example

T2 defines the time period during which the voltage or current defined with V2 or I2 is generated.

Time/Value Pairs

The maximum number of time/value pairs is nine. To specify more pairs, use a text file. Place time/value pairs for each data point in the text file. Add a FILE attribute to the PWL or IPWL component on the schematic.

TNOM

The TNOM attribute specifies the default temperature, in centigrade degrees, on the OPTIONS analog report generator.

Placement

On the symbol.

Example

TNOM=25 indicates a default temperature of 25 degrees, centigrade.

TPERIOD

The TPERIOD attribute specifies the length of time of one complete waveform cycle on the PULSE and IPULSE analog symbols.

Placement

On the symbol.

Example

TPERIOD=50ns indicates a 20 Megahertz waveform.

TPULWIDTH

The TPULWIDTH attribute specifies length of time the pulse is active on PULSE and IPULSE analog symbols.

Placement

On the symbol.

Attribute Value

The value of this attribute is substituted for “pw” in the waveform formula. Refer to the waveform formula under I and V in the PSpice Help file.

Example

TPULWIDTH=20ns indicates an active pulse width of 20 nanoseconds.

V1, V2, V3, ..., V9

The V1 attribute specifies amount of voltage generated by the PWL analog symbol at the time period specified by the T1 attribute.

Placement

On the symbol.

The ORDER Attribute

You must identify this attribute in the ORDER attribute of the PWL for interpretation.

The T1 Attribute

The T1 attribute specifies the time period at which the voltage V1 is generated.

Example

V1=2V indicates that 2 volts are generated at the T1 time period.

VALUE

In addition to using the VALUE attribute to specify R, L, or C values, the VALUE attribute specifies relative small–signal amplitude on AC and IAC analog symbols.

Analog Placement

On the symbol.

Analog Example

Attached to an AC component, VALUE=5 indicates a small–signal amplitude of 5 volts.

Analog Example 2

Attached to an R component, VALUE=5.1K indicates a 5.1K ohms resistance value.

VAMPLITUDE

The VAMPLITUDE attribute specifies maximum voltage swing on the SIN analog symbol.

Placement

On the symbol.

Attribute Value

The value of this attribute is substituted for “vampl” in the waveform formula. Refer to the waveform formula under I and V in the PSpice Help file.

Example

VAMPLITUDE=100E–3 indicates a maximum voltage swing of 100 millivolts.

VINITIAL

The VINITIAL attribute specifies the starting voltage of the waveform on the PULSE analog symbol.

Placement

On the symbol.

Example

VINITIAL=5V indicates a starting voltage of 5 volts.

VNTOL

The VNTOL attribute on the OPTIONS analog report generator specifies the best accuracy, in volts, of voltage measurements.

For more information, refer to the HSpice or PSpice Help file.

Placement

On the symbol.

Default

The default attribute value is VNTOL=1uV.

Example

VNTOL=1uV indicates that voltage measurements are accurate to within 1 micro volt.

VOFFSET

The VOFFSET attribute specifies the Y axis point at which DC voltage is measured on the SIN analog symbol.

Placement

On the symbol.

Attribute Value

The value of this attribute is substituted for the “voff” parameter in the waveform formula. Refer to the waveform formula under I and V in the PSpice Help file.

Example

VOFFSET=50mV indicates that the waveform is measured around the Y axis point that corresponds to 50 millivolts.

VOLTAGE

The VOLTAGE attribute specifies voltage across the DC analog component.

Placement

On the symbol.

The ORDER Attribute

You must identify this attribute in the ORDER attribute for interpretation.

The CURRENT Attribute

Use the CURRENT attribute on the IDC analog component.

The VPULSED Attribute

If the design contains conflicting attribute values, simulation results will be unspecified. For example, the value of the VPULSED attribute must agree with the value of the VOLTAGE attribute if both attributes are used in a design.

Example

VOLTAGE=5V indicates that there are 5 volts across this DC component.

VPULSED

The VPULSED attribute specifies the maximum voltage swing of the PULSE analog symbol when it is ON.

Placement

On the symbol.

The VOLTAGE Attribute

If the design contains conflicting attribute values, simulation results will be unspecified. For example, the value of the VPULSED attribute must agree with the value of the VOLTAGE attribute if both attributes are used in a design.

Example

VPULSED=5V indicates a maximum voltage swing of 5 volts.

W

The W attribute defines the width of an analog FET channel in meters.

Placement

On the symbol.

The ORDER Attribute

You must identify this attribute in the ORDER attribute for interpretation.

Example

W=3u indicates that the channel is 3 microns wide.

WIDTH

The WIDTH attribute on the OPTIONS analog report generator specifies the number of characters per line of text in a print table. Use 80 or 132 in the value field to specify 80 or 132 characters, respectively.

Placement

On the symbol.

Example

WIDTH=80 indicates that each line in a print table can contain 80 characters.

YMAX

The YMAX attribute specifies maximum allowable deviation of Monte–Carlo results in the Y direction on the MC analog report generator.

Placement

On the symbol.

Example

YMAX=5 indicates that maximum deviation in the Y direction is 5.