SIEMENS EDA

Layout Automation Reference (Part 1)

Examples, PCB Applications, Project Integration, Output Window Add-In

Release X-ENTP VX.2.13
Document Revision 10

SIEMENS

About Siemens Digital Industries Software

Siemens Digital Industries Software is a leading global provider of product life cycle management (PLM) software and services with 7 million licensed seats and 71,000 customers worldwide. Headquartered in Plano, Texas, Siemens Digital Industries Software works collaboratively with companies to deliver open solutions that help them turn more ideas into successful products. For more information on Siemens Digital Industries Software products and services, visit www.siemens.com/plm.

Support Center: support.sw.siemens.com
Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Revision History ISO-26262

Revision	Changes	Status/Date
10	Modifications to improve the readability and comprehension of the content. Approved by Kevin Chupp. All technical enhancements, changes, and fixes listed in the *Xpedition Enterprise Flow Release Notes* for this product are reflected in this document. Approved by Mike Bare.	Released March 2023
9	Modifications to improve the readability and comprehension of the content. Approved by Kevin Chupp. All technical enhancements, changes, and fixes listed in the *Xpedition Enterprise Flow Release Notes* for this product are reflected in this document. Approved by Mike Bare.	Released September 2022
8	Modifications to title page to reflect the latest product version supported. Approved by Kevin Chupp. All technical enhancements, changes, and fixes listed in the *Xpedition Enterprise Flow Release Notes* for this product are reflected in this document. Approved by Mike Bare.	Released March 2022
7	Modifications to title page to reflect the latest product version supported. Approved by Kevin Chupp. All technical enhancements, changes, and fixes listed in the *Xpedition Enterprise Flow Release Notes* for this product are reflected in this document. Approved by Mike Bare.	Released September 2021

Author: In-house procedures and working practices require multiple authors for documents. All associated authors for each topic within this document are tracked within the Siemens documentation source. For specific topic authors, contact the Siemens Digital Industries Software documentation department.

Revision History: Released documents include a revision history of up to four revisions. For earlier revision history, refer to earlier releases of documentation on Support Center.

Table of Contents

Chapter 1
Examples of PCB Automation

The section provides working examples of PCB automation.

Refer to the scripts used in these examples to help understand how you can use data objects, methods, properties, and events to automate tasks that you typically perform manually.

Note:
If you need information on the PCB batch engines refer to the separate manual, *Automation Pro Batch Engines Reference*.

Automation Script Samples

All of the script samples that are embedded in this document and included with the application software are written in VBScript. VBScript version 5 or higher is required to run the samples.

You can find the samples included in this document in the subdirectory:

standard\examples\pcb\Automation\HelpSamples

The examples show how to use a given method or property. They are not full-featured scripts. There is basic error checking in most examples, however it might be insufficient to run in all circumstances.

The samples must be run internally unless you run them using **mgcscript** on command line. Refer to "Running and Debugging MGCPCB Scripts" on page 14 for more information on running scripts.

There are additional scripting samples located in the subdirectory:

standard\examples\pcb\Automation\Scripts

A description of each of this files is listed in "Table 1: Scripting Samples" on page 12.

Note:
There are no samples for the C++ programming language. You can use several languages to write your scripts. If the language supports COM object models you can use it with the automation server.

Table 1. Scripting Samples

File	Description
ActiveLayerKeyBindings.vbs	Uses the Key Binding Server for mapping number keys to the change active layer command.
AIS.vbs	Creates an AIS bill of materials.
AutoRoute.vbs	Shows how to run the Auto Router using the RoutePass object.
BoardOutline.vbs	Shows the coordinates of the board outline.
CommandBars.vbs	Shows how to use the command bar server to customize menu bar and tool bar.
Connectivity.vbs	Shows how to extract the connectivity for an object.
DeleteAllPlaneShapes.vbs	Shows how to delete the plane shapes on a layer.
EventHandler.vbs	Shows how to create an event handler in VBScript (must be run external to the application).
GenerateBOM.vbs	Creates a bill of materials (bom.txt)
IsGeometryInside.vbs	Uses the MaskEngine Automation Controller to test whether a geometry is inside the board outline or another geometry.
KeyBindings.vbs	Shows how to use the key binding server to assign shortcut keys to menu command and to run user scripts.
NewUserLayer.vbs	Shows how to create a new user layer.
PadstackEditor.vbs	Shows how to access and run the Padstack Editor Automation.

Table 1. Scripting Samples (continued)

PDBEditor.vbs	Shows how to access and run the PDB Editor Automation.
Pick.vbs	Shows how to select elements with the Document.Pick method.
PlaceBoards.vbs	Shows how to query the user for board name, rotation and flip status, then fill the a panel with the entire board.
PlaceDetailedViews.vbs	Creates detailed views from the board or panel four quadrants.
RouteBoard.vbs	Shows how to run the Auto Router using the Gui.ProcessCommand on page 238 method.
Spiral.vbs	Shows how to create a spiral trace. Also shows the use of classes in VBScript.
WireReport.vbs	Shows how to iterate through a collection of bond wire objects, read the beginning and ending coordinate values of each bond wire, and write the values to a text file.

Scripting Languages Compatible With Automation

COM technology allows you to develop scripts in several languages.

Each language has strengths and weaknesses. You should always choose the language that is appropriate for the task at hand. VBScript is often the default choice; it is the easiest language to understand and work with. All examples of Automation script in the PCB Automation documentation use VBScript. Jscript is often the second language choice; however some common tasks, such as iterating collections, are tedious. Both VBScript and Jscript run on Windows and UNIX environments.

For information on COM scripting languages refer to the wealth of information available on the internet.

Running and Debugging MGCPCB Scripts

The VBScript and JScript programming languages are the most generic programming platforms for the MGCPCB Automation layer.

You can create script files with any text editor and they are compatible across operating systems (Unix, Windows).

You can run scripts in the following ways:

- **Externally** — In this mode you launch the script file outside the MGCPCB application. This is done by running the script file with mgcscript command on command line:

  mgcscript <name_of_the_script_file>

- **Internally** — In this mode you run the script inside the application. This is done by using the keyin:

  run <name_of_the_script_file>

  (The ProcessScript property behaves the same as the run command). On Windows, you can also run the script by drag-and-drop the script file to the application.

Note that each mode has its limitations. These are listed below.

External Mode Limitations

External Mode imposes some limitations on execution speed and the use of enumerated types.

- **Execution speed** — Running a script externally creates an out-of-process relationship between the script and the application. This means that scripts that are launched this way will execute somewhat slower than scripts that were run internally. If you have scripts that require heavy interaction with the application server it is advisable to write them such that they can be run internally.

- **Enumerated types** — The MGCPCB object model exposes many enumerated types (epcbxxx...) for its commands. These are constants that make the code more readable with less room for errors. However, these enumerated types are not recognized if the script is run externally. There are a few solutions to this problem:

  a. Replace the epcbxxx constant names with their numerical value. Although this is a way to solve the problem you will need to be careful to use the correct numerical values. Furthermore, the code will become less readable.

     The example below shows the difference for the method that writes information to the status field:

```
Call app.Gui.StatusBarText("Information-script failed
 ",epcbStatusField1)
Call app.Gui.StatusBarText("Information-script failed",1)
```

b. Use the command Scripting.AddTypeLibrary in your script. By adding Scripting.AddTypeLibrary("MGCPCB.Application") after connecting to the server application, your script recognizes enumerated types defined in MGCPCB type library. For example:

```
Dim app
Set app = GetObject(,"MGCPCB.ExpeditionPCBApplication")
Scripting.AddTypeLibrary("MGCPCB.ExpeditionPCBApplication")
```

Internal Mode Limitations

Internal Mode imposes some limitations on collection of arguments.

- **No arguments collected** — It is not possible to collect the arguments the user may have supplied when running a script internally.

Debugging Scripts

It is beyond the scope of this document to explain in full detail how you can debug your scripts. However, a few useful tips can get you started:

- Add debug lines to your script. This is the simplest way to get feedback from your script. You can use the "Gui.Display" on page 226 command for this purpose.
- Use Microsoft script debugger (Windows only).
- Use the Microsoft Office script debugger (Windows only). If you are a Microsoft office user you can use the debugger that is delivered with Microsoft Office. It offers more functionality than the script debugger.

Using mgcscript

Mgcscript provides a cross-platform way to run script files on MGCPCB Automation layer.

Another advantage of using mgcscript is it provides a Scripting object which allows attaching events when running script internally and externally. This section describes the use of mgcscript.

For more information on running script internally and externally refer to “Running and Debugging MGCPCB Scripts” on page 14. For more information on the Scripting object refer to Scripting object description.

Running Scripts with mgcscript

You can use mgcscript to run your script externally on the command line.

The following example shows how to run your script externally on the command line:

```
mgcscript <name_of_the_script_file> <optional_arguments_list>
```

If you pass any arguments to your script, you can collect the arguments using the ScriptHelper object which is automatically created and added to the currently running script when using mgcscript.

ScriptHelper Properties

ScriptHelper has two properties; Arguments and Echo.

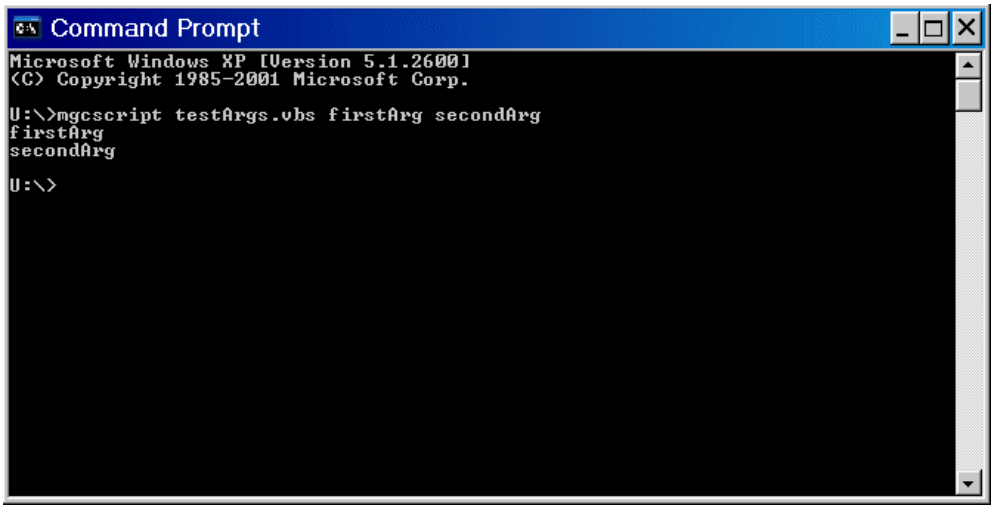
Note:
ScriptHelper.Arguments collects all arguments supplied by user, including mgcscript command itself. Therefore, the first argument to the script will be the third item in the collection returned by ScriptHelper.Arguments.

The example below shows how to use two of the ScriptHelper object properties, Arguments and Echo.

Example 1. ScriptHelper Arguments

```
Set ScriptArgs = ScriptHelper.Arguments
arg1 = ScriptArgs.Item(3)
' The first argument is the third item in the collection
arg2 = ScriptArgs.Item(4)

ScriptHelper.Echo arg1
ScriptHelper.Echo arg2
```



Using Startup Scripts

Startup scripts are defined in the scripts.ini file.

When an Auto Active application is invoked the default script directories are searched for files named *scripts.ini*. The scripts listed in the *scripts.ini* files are then run at either document startup or application startup depending on what section of the *scripts.ini* file the script is listed in. If the command line option **-nostart** is specified to the application no startup scripts will be run.

Default script directories are the paths that the*scripts.ini* files are searched in and the paths scripts are searched in if an explicit path is not given. The directories are searched in this order:

1. Product area (*%SDD_HOME%\standard\automation\startup*)
2. Design area (directory containing the design file, i.e. *.pcb*)
3. User defined areas
   - Defined by the **WDIR** environment variable
   - Allows multiple, semi-colon separated paths, for example:

     ```
     set WDIR = c:\jobs; c:\temp
     ```

   - Paths defined earlier in the **WDIR** variable take precedence over paths defined later

Scripts.ini File Format

The *scripts.ini* files list the startup scripts to run on a per product basis. The general format of this file is:

```
[application]
Script#<N>=<script_name>
```

For example, to run the script M*yStartup.vbs* and *MyStartup2.vbs* when Xpedition Layout invokes, the *scripts.ini* file would specify:

```
[Expedition PCB]
Script#0=C:\scripts\MyStartup.vbs
Script#1=C:\scripts\MyStartup2.vbs
```

Adding “- Document” after the application name causes the script to run when a document opens instead of when the application invokes.

```
[Expedition PCB - Document]
Script#0=C:\scripts\MyStartup.vbs
Script#1=C:\scripts\MyStartup2.vbs
```

If the path to a script is not defined in the *scripts.ini* file the script will be looked for in default script directories. Once a script by that name is found the search stops and that script is run. Additionally, environment variables can be used in the *scripts.ini* files. The environment variables can be specified in two formats:

```
%var_name%
$var_name
```

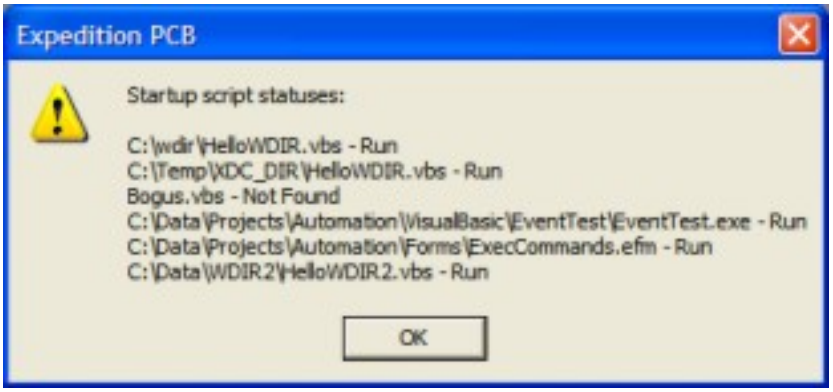
The “~” character can be used to identify the user home directory. The “*” character can be used as a wildcard character. This character will only run the first script found that matches this character, It will not run all scripts that match the expression.

It is also possible to run files other than scripts on startup. On Windows platforms, *.efm* (Script Forms), *.exe*, and *.bat* files can be defined in the *scripts.ini* file. On UNIX platforms, anything with execute permissions can be defined in the *scripts.ini* file.

Command Line Options and Debugging

Two command line options are available to modify the behavior of the startup scripts and to help users debug the usage of these scripts. The command line option **-nostart** will invoke the application without running any startup scripts. The command line option **-dbgstart** will display a message box listing each script that was found in a *scripts.ini* file and whether that script was run or not. If the script was run a “- Run” will be next to the script name. If the script could not be found and was therefore not run “- Not Found” will be next to the script name.

The following is an example of the message box:



The following is a list of the product names to use for the application field in *scripts.ini* files.

Table 2. scripts.ini Application Keys

Application Name	Gui Display	Application Key
BoardStationRE	Board Station RE - XXX	Board Station RE
BoardStationXE	Board Station XE - XXX	Board Station XE
DrawingEditor	Xpedition FabLink: Drawing Editor	Drawing Editor
ExpeditionPCB	Xpedition Layout - XXX	Expedition PCB
FablinkXE	Xpedition FabLink	FabLink XE
PCBBrowser	Layout Browser	PCB Browser
PlannerPCB	Xpedition Layout Planner	Discovery PCB
SFXRE	SFX RE - XXX	SFX RE
ViewerPCB	Xpedition Layout Viewer	Discovery PCB
PackageDesigner	Xpedition Package Designer	Xpedition PD
LibraryManager	EDM Library Tools	Library Manager
CellEditor	Cell Editor - XXX	Cell Editor

Note:
For PCB Browser, you can use automation to interact with the GUI (for example: **SuppressTrivialDialogs = True**). However, you cannot use automation to interact with the layout design data.

Connecting to the MGCPCB Server

In order to use the MGCPCB automation layer you must connect to the MGCPCB automation server. Like most automation models the server exposes an application object for this purpose. However, the automation server is a multipurpose server for several PCB applications. Therefore, dedicated application objects exist for the different applications. This includes the following applications:

- Application (generic AutoActive application)
- BoardStationREApplication
- BoardStationXEApplication
- CellEditorApplication
- DrawingEditorApplication
- ExpeditionPCBApplication
- FablinkXEApplication
- PlannerPCBApplication
- SFXREApplication
- ViewerPCBApplication

Dedicated application objects allow you to connect to a specific application whereas the generic application object connects to the current running instance of the server. The following shows a few examples (using VBScript syntax):

- Check if any of the PCB applications are running and return a reference to it (generates an error if none of the applications are running):

```
Set app = GetObject(,"MGCPCB.Application")
```

- Check if the BoardStation RE application is running and return a reference to it (generates an error if BoardStation RE is not running):

```
Set app = GetObject(,"MGCPCB.BoardStationREApplication")
```

- You should not use the following because the server does not know which application type to create:

```
Set app = CreateObject("MGCPCB.Application")
```

- This creates a new instance for the BoardStation RE application and returns a reference to the newly created instance:

```
Set app = CreateObject("MGCPCB.BoardStationREApplication")
```

- This creates a new instance for the Xpedition Layout application for version X-ENTP VX.x (COM version 9) and returns a reference to the newly created instance:

```
Set app = CreateObject("MGCPCB.ExpeditionPCBApplication.9")
```

Note:
You must specify the COM version for machines with multiple software installs.

Feature Support
Validating References

Feature Support

The fact that the automation server supports several applications does not imply that all methods and properties will work for all applications.

If a method or property is not supported, the following error will be generated when you try to use it.

Error epcbErrCodeFunctionNotSupported: This function is not supported for <product_name>

Note:
The documentation does not contain any information on the support level for a given method or property for an application.

Validating References

It is common practice to define an application reference that is used virtually everywhere as a global variable in a script or application program. It is important to make sure the reference is still valid at the time you use or reuse it. Use the following procedure(s).

Procedure

1. Check the validity of the object through its IsValid Method (Applications). As long as this method returns True you can safely use the application reference.

   As long as this method returns True you can safely use the application reference.

2. You can create event handlers that keep track of the state of the application through the Quit Event and OnOpenDocument Event.

Use the Quit event handler to set the reference to Nothing if the application exits.

Use the OnOpenDocument event handler to reassign the active document and to perform custom document initialization and verification on newly opened documents. (Document object references are mostly defined as global because of their frequent use).

Using Automation Licenses and Document Validation

Every time you work with a document object you must request a license from the automation license server. If you do not request a license you will not be able to call Automation methods on the document.

Procedure

1. Retrieve a reference to the document object. Typically, this is done using the ActiveDocument Property or the OpenDocument Method:

```
Set docObj = app.ActiveDocument
```

2. Generate an identification key for the document object by passing 0 to its Validate method:

```
key = docobj.Validate(0)
```

3. Create an instance of the automation license server object:

```
Set licenseServer =
 CreateObject("MGCPCBAutomationLicensing.Application")
```

4. Generate a license token for the document object using the key you generated in step 2:

```
licenseToken = licenseServer.GetToken(key)
```

5. Validate the document object using the generated license token:

```
docObj.Validate(licenseToken)
```

6. If the document validates correctly, no error results. If the license token is incorrect, the error epcbErrCodeValidationFailed results.

Examples

For your convenience, you can use the function shown in Example 2: GetLicensedDoc Function in your application scripts. It retrieves a licensed document from the application object and generates the appropriate errors if license retrieval fails. Note that every sample this documentation uses this function.

Example 2. GetLicensedDoc Function

```
‘ This function retrieves an automation license for a document
‘
‘ Returns:
‘ - ‘Nothing’ if licensing failed .
‘ - A reference to the active document of the application if
‘ licensing succeeded.
‘
Public Function GetLicensedDoc(app)
 On Error Resume next
 Dim key,licenseServer,licenseToken,docObj
 Set GetLicensedDoc = Nothing
 ‘ collect the active document
 Set docObj = app.ActiveDocument
 If (Err) Then
  Call app.Gui.StatusBarText(“No active document: “ + _
   Err.Description,epcbStatusFieldError)
  Exit Function
 End If
 ‘ Ask document for the key
 key = docObj.Validate(0)
 ‘ Get token from license server
 Set licenseServer = _
  CreateObject(“MGCPCBAutomationLicensing.Application”)
 licenseToken = licenseServer.GetToken(key)
 Set licenseServer = Nothing
 ‘ Ask the document to validate the license token
 Err.Clear
 docObj.Validate(licenseToken)
 If (Err) Then
  Call app.Gui.StatusBarText(“No active document license: “ + _
   Err.Description,epcbStatusFieldError)
  Exit Function
 End If
 ‘ everything is OK, return document
 Set GetLicensedDoc = docObj
End Function
‘
‘ Testing code for the getlicenced function
‘
Dim app
Dim docObj
Set app = GetObject(,”MGCPCB.Application”)
‘ get the document
Set docObj = GetLicensedDoc(app)
MsgBox docObj.name
```

Using Server Locking

The MGCPCB server is created such that you can easily interact with a running instance of the PCB tool. The server services interactive input as well as the input and interrupts you provide when you process a script and/or interact with the server.

If you do not take special precautions, the server interaction can be very time consuming because the server has to grant access to its data structures. It does this by making sure that other input sources cannot interfere with your operations. If the interface is processing many calls to the server you will notice that performance slows down because the overhead of “access granting” every time the script interacts with the server.

You can resolve this problem by enclosing operation(s) with LockServer Method and UnlockServer Method. Typically, a server locking code segment appears as follows:

```
If (App.LockServer = True) Then
 ...
 <perform your server actions here>
 App.UnlockServer
End If
```

In the example, the script first tries to lock the server. If this succeeds, the script can process the server actions. The server processes these operations faster because the script locked it such that no other inputs are serviced. Once the script is done, it must release the lock and free up the server for other processing.

LockServer and UnlockServer methods can also be nested. However, care must always be taken that one LockServer statement always matches a corresponding UnlockServer statement.

Limitation of Server Locking

It might be tempting to enclose all operations with LockServer and UnlockServer methods. However, there are cases where this is not advisable.

Graphics updates, event processing and autorouting are delayed because of server locking.

It is important to be aware of these delayed processes. For example, newly created or deleted elements may not display correctly until the server is unlocked.

In the following cases it is advisable to use the LockServer and UnlockServer methods:

- Reading many elements (for example; traversing the Nets collection to find all net names in a design).
- Deleting many elements.

In the following cases it is not advisable to use the LockServer and UnlockServer methods:

- For operations that are very small and/or do not require much server CPU time.
- For operations that require DRC checks on the element.

- For operations where immediate display update is a requirement.

CAUTION:
You should not place GUI operations, other than trivial operations such as status bar updates, inside LockServer/UnlockServer blocks. Unpredictable behavior may occur.

If you require control over DRC and undo operations you should use transactions. Refer to “Using Transactions” on page 25 for more information.

Using Transactions

Grouping operations gives you more control over transactions.

It can be useful to group a set of operations together in such a way that you can:

- Control the level of DRC when an element is added or changed.
- Decide if the changes should be kept or discarded.
- Place the operation on the Undo stack of the PCB application so that the user, if desired, can undo them.

The MGCPCB automation layer uses transactions to achieve these requirements. Typically, a transaction source code segment resembles the following:

```
If (doc.TransactionStart(<drc_mode>) = True) Then
 ...
 <perform your transaction operations here>

 If (noProblems = True) then
  doc.TransactionEnd(True) ' All OK, commit the transaction
 Else
  doc.TransactionEnd(False) ' Something went wrong. Do not commit.
 End if
End If
```

In the code segment the transaction is started with the TransactionStart method. At this point you indicate the level of DRC you want to apply to the changes inside transaction. The <drc_mode> can be **epcbDRCModeNone** (no checking), **epcbDRCModeDRC** (check DRC) or **epcbDRCModeResolve** (try to automatically resolve DRC problems)

Once the transaction starts, the script can add or change objects inside of it. Note that any modification is allowed regardless of the DRC mode that is set at the beginning of the transaction. The actual DRC checks (if any) are performed when the transaction completes.

To complete a transaction, use the TransectionEnd method. If you supply False to this method, none of the changes are written to the design. If you supply True, DRC is performed on the modified elements according to the DRC mode that is set with TransactionStart. If DRC violations occur, TransactionEnd returns False and the elements that change do not write to the design. A return value of True indicates that DRC found no violations.

The element changes that are written during TransactionEnd are placed on the Undo stack as a single entity allowing you to reset the changes if required.

- You can combine transactions with “server locking” on page 24 to speed up operations.
- Transactions can be nested and may use different DRC modes.
- Any TransactionStart statement must always have a matching TransactionEnd statement.
- DRC mode does not influence component modification commands. It is controlled with the settings of the EditorControl.PartsOnlineDRC property.

CAUTION:
You should not place GUI operations, other than trivial operations such as status bar updates, inside TransactionStart/TransactionEnd blocks. Unpredictable behavior may occur. These methods include Gui.ProcessCommand, Gui.ProcessKeyin and Button.Click.

Customizing Menus

You can customize the PCB application menus or menu bar by adding and removing commands and menus.

The command bar server provides these functions. The sample script “Table 1: Scripting Samples” on page 12 shows how to use the command bar server to customize menu, menu bar and toolbars.

Connecting to the command bar server

The “CommandBars property” on page 222 of the “Gui object” on page 216 returns a reference to the CommandBars collection object and from which you can start customizing the menu or menu bar:

```
Dim cmdBars
Set cmdBars = Gui.CommandBars
```

By specifying the menu bar name when using the “CommandBars property” on page 222, a reference to the CommandBar object will be returned. Note that the “Document Menu Bar” is the default menu bar. All products support the “Document Menu Bar”. To modify the menu bar for Layout (DFL mode) use “DFL Document Menu Bar”. To edit the menu bar that is used when a *.pcb* file is opened in FabLink DWG Editor FabLink use “PCB Document Menu Bar”.

```
Dim docMenuBar
Set docMenuBar = Gui.CommandBars("Document Menu Bar")
```

```
Dim cmdBars
Set cmdBars = Gui.CommandBars
```

By specifying the menu bar name when using the “CommandBars property” on page 222, a reference to the CommandBar object will be returned. Note that the “Document Menu Bar” is the default menu bar. All products support the “Document Menu Bar”. To modify the menu bar for Layout (DFL mode) use “DFL Document Menu Bar”. To edit the menu bar that is used when a *.pcb* file is opened in Xpedition FabLink use “PCB Document Menu Bar”.

```
Dim docMenuBar
Set docMenuBar = Gui.CommandBars("Document Menu Bar")
```

Adding a menu or a command

The Add method of the CommandBarControls collection object allows you to add a new menu or command to a menu or the menu bar. The following example adds a new menu at the end of the menu bar and names it “My Menu”:

```
Dim myMenu
Set myMenu = docMenuBar.Controls.Add(cmdControlPopup,,,-1)
myMenu.Caption = "My Menu"
```

You can add a menu to another menu by calling the same methods. The following example adds a menu to the menu created in the previous example, and names it “Edit”:

```
Set myEditMenu = myMenu.Controls.Add(cmdControlPopup,,,-1)
myEditMenu.Caption = "Edit"
```

To add a command to a menu, from the menu object, return its CommandBarControls collection, then use the Add method to add commands. The following example adds three built-in commands to the “Edit” menu created in the previous example:

```
Set myEditCntrls = myEditMenu.Controls
Set cmd1 = myEditCntrls.Add
cmd1.Caption = "&Highlight"
cmd1.OnAction = "32867"
Set cmd2 = myEditCntrls.Add
cmd2.Caption = "&Fix"
cmd2.OnAction = "32866"
Set cmd3 = myEditCntrls.Add
cmd3.Caption = "&Lock"
cmd3.OnAction = "32868"
```

Note:
The built-in command id used in OnAction property can be returned by using the CommandBarButton.Id property.

Adding a separator

You can add a separator to a menu by specifying the type parameter (CmdControlType) in the Add method of the CommandBarControls collection object:

```
myMenu.Controls.Add(cmdControlButtonSeparator)
```

Delete a command from a menu

Any menu or command can be deleted using the Delete method of the CommandBarControl object. For example, the following deletes the first menu or command in the myMenu object:

```
myMenu.Controls.Item(1).Delete
```

Renaming a menu or command

To rename a menu or a command, use the Caption Property of the CommandBarControl object. You can only rename a menu or command created by the command bar server, but not a built-in menu or command. The following example will name the first menu or command in the myMenu object as “My Command”:

```
myMenu.Controls.Item(1).Caption = "My Command"
```

Adding a command to a popup menu

To add a command to a popup menu, you call a pre-popup function that allows you to modify the popup menu before displaying it. Then, use the ExecuteMethod function to execute your command from the popup.

Example 3. Adding a Command to a Popup Menu

```
Option Explicit

'How to add a user command to popup  menu

Scripting.DontExit=True
RegisterPrePopupClient(Application)

Sub RegisterPrePopupClient(application)
 Dim gui
 Set gui = application.Gui
 If Not ( gui Is Nothing ) Then
  Dim commandBars
  Set commandBars = gui.commandbars
  If Not ( commandBars Is Nothing ) Then
   'call the function "OnUpdatePopupMenuCallback" before displaying
   'the popup to allow the function to modify the popup menu
   commandBars.RegisterOnPreDisplayPopupClient "AnyPopup", _
           ScriptEngine, "OnUpdatePopupMenuCallback"
  End If
 End If
End Sub

'Dynamic Creation of a popup menu item
Function OnUpdatePopupMenuCallback( menu, title )
 Dim cntrls
 Set cntrls = menu.controls

 Dim button
   'Add a separator button to the menu
   Set button = cntrls.Add(cmdControlButtonSeparator,,,-1)
   'Add a command button to the menu
 Set button = cntrls.Add(cmdControlButton,,,-1)
 If Not Err.Number Then
```

```
  button.Caption = "Menu entry added by Automation test"
      'Target COM object is this scripting engine
  button.Target = ScriptEngine
      'Call function below with this name
      button.ExecuteMethod = "onClickCallback"
  button.UpdateMethodEx = "onUpdateClickCallback"
 End If
End Function

Function onClickCallback(nID)
 MsgBox("Executing function")
End Function

Function onUpdateClickCallback(button)
   'This is called before and after the ExecuteMethod
 button.Checked = False
 button.Enabled = True
End Function

'=====================================================
'Not used in this example, but can be used when required to
'remove the prepopup call, such as when Closing a Document.
Sub TestUnRegisterPrePopupClient(application)
 Dim gui
 Set gui = application.Gui
 If Not ( gui Is Nothing ) Then
  Dim commandBars
  Set commandBars = gui.commandbars
  If Not ( commandBars Is Nothing ) Then
   commandBars.UnRegisterOnPreDisplayPopupClient(ScriptEngine)
  End If
 End If
End Sub
```

Customizing Toolbars

You can customize the PCB application toolbars by adding and removing buttons.

The command bar server provides the functions. The sample script “Table 1: Scripting Samples” on page 12 shows how to use the command bar server to customize menu, menu bar and toolbars.

Connecting to the CommandBarServer

The “CommandBars property” on page 222 of the “Gui object” on page 216 returns a reference to the CommandBars collection object and from which you can start customizing toolbars:

```
Dim cmdBars
Set cmdBars = Gui.CommandBars
```

By specifying the tool bar name when using the “CommandBars” on page 222 property, a reference to the CommandBar object will be returned:

```
Dim stdToolBar
Set stdToolBar = Gui.CommandBars("Standard")
```

Adding a button to a toolbar

The Add method of the CommandBarControls collection object allows you to add a new button to a toolbar. The following example adds a new button to the “Standard” toolbar to run the “Fit All” command:

```
Dim stdTBCtrls, btn1
Set stdTBCtrls = stdToolBar.Controls
Set btn1 = stdTBCtrls.Add(cmdControlButton,,,14)
btn1.OnAction = "33128"
btn1.TooltipText = "View->Fit All"
btn1.DescriptionText = "Fit all data to screen "
btn1.BitmapFile = "c:\temp\viewall.bmp"
```

Adding a separator on a toolbar

You can add a separator on a toolbar by specifying the Type argument (CmdControlType) and the Before argument in the Add method of the CommandBarControls collection object. The following example adds a separator before the 15th button, including separators, on the “Standard” toolbar:

```
stdToolBar.Controls.Add(cmdControlButtonSeparator,,,15)
```

Removing a toolbar button

Any toolbar button in a PCB application can be deleted using the Delete method of the CommandBarControl object. For example, the following deletes the first button on the “Standard” toolbar:

```
stdToolBar.Controls.Item(1).Delete
```

Running Commands with Automation

You can use automation to choose menu commands found in the application pull-down menus.

Examples of this operation include the generation of Design Status information, Gerber Output, Forward Annotation, etc.

The automation model offers a generic procedure to achieve this:

Procedure

1. Use the “Gui.ProcessCommand” on page 238(“<command_name>”) method. The command name references a name on the pull-down menus. This method behaves exactly the same as if you select the command from the menu. If a dialog displays, refer to the following steps to manipulate it.

2. If a dialog displays in step 1, locate it with the “Gui.FindDialog” on page 231(“<caption of dialog>”) method. This method returns a reference to the dialog object if it is found.

3. (Optional) If the dialog object itself contains pull-down menus use the "Dialog.ProcessCommand" on page 212("<command_name>") to run a pull-down command. Next, repeat step 2 if the command you run displays a dialog.

4. Locate buttons and checkboxes on the dialog. Find buttons with "Dialog.FiindButton" on page 210("<button_caption>"). This method returns a "Button" on page 155 object. Find checkboxes with "Dialog.FindCheckBox" on page 211("<caption>"). This method returns a "checkbox" on page 158 object.

5. You emulate a button click with the "Button.click" on page 156 method. You change or verify a checkbox with the "checkbox.Checked" on page 159 property.

6. (Optional) Terminate the current interactive command with the "Gui.TerminateCommand" on page 250 method.

   Notes:

   - Not all of the dialog elements can be manipulated. For example, complex list selections cannot be achieved with the methods outlined above.

   - Several dialogs have their own object so that you can modify the settings of the dialog more easily. These include: View.DisplayControl, Document.EditorControl, and Document.NewRoutePass.

   - Some commands may display confirmation boxes or the Notepad editor when they complete. You can suppress these dialogs using "Gui.SuppressNotepadDialogs" on page 247 and "Gui.SuppressTrivialDialogs" on page 248. The following shows sample code to add to your scripts to suppress confirmation boxes and the notepad editor:

```
 ' Do not display the notepad editor at the end of the command
Gui.SuppressNotepadDialogs = True
 ' Do not display confirmation dialogs
Gui.SuppressTrivialDialogs = True
```

CAUTION:
When you utilize a name from an application's GUI, the automation results cannot be guaranteed from software release to software release. This occurs when the application's GUI changes. These names include: menu commands (Gui.ProcessCommand), dialog control names (Gui.FindDialog, Dialog.FindButton, Dialog.FindCheckbox). If the application's GUI changes, you must update the corresponding Gui object names in your scripts.

Examples

The following examples show how to run commands and setup dialog boxes in the application environment.

Example 4. Generating Gerber Using ProcessCommand

```
Option Explicit
'
' The example below opens the Gerber output dialog and generates
' gerber with the current settings
'
Sub Execute_GerberOut ()
 Dim resultMess
 Dim retBool,diaObj
 Dim butObj
 ' Display the Gerber output dialog
 retBool = app.Gui.ProcessCommand("Gerber",True)
 If (retBool = False) Then
  Call app.Gui.StatusBarText("Export " + _
   Err.Description,epcbStatusFieldError)
  Exit Sub
 End If
 ' Find the dialog
 Set diaObj = app.Gui.FindDialog("Gerber Output")
 If (Err Or (diaObj Is Nothing)) Then
  Call app.Gui.StatusBarText("Gerber output dialog not found " + _
   Err.Description,epcbStatusFieldError)
  Exit Sub
 End If
 ' Find the process button
 Set butObj = diaObj.FindButton("Process Checked Output Files")
 If (butObj Is Nothing) Then
  Call app.Gui.StatusBarText("Gerber output-Process button not " +_
   "found " +   Err.Description,epcbStatusFieldError)
  Exit Sub
 End if
 ' Click the Process button. This will generate the gerber
 Call butObj.Click
 ' Close the dialog
 Set butObj = diaObj.FindButton("Close")
 If (Not butObj Is Nothing) Then
 Call butObj.ClickresultMess = "Gerber output was created"
 ' Update status bar
 Call app.Gui.StatusBarText(resultMess,epcbStatusField1)
End Sub
'
' Testing code for the sample
'
Dim app
Set app = GetObject(,"MGCPCB.Application")
Call Execute_GerberOut
```

Example 5. Generating a Design Status Report

```
Option Explicit
'
' The example below generates the design status
'
'
Sub Execute_DesignStatus ()
 Dim resultMess
 Dim retBool,diaObj
 Dim cboxObj,butObj
 ' Make sure the notepad does not appear
 app.Gui.SuppressNotepadDialogs = True
 ' Display the dump dialog by executing the command
 retBool = app.Gui.ProcessCommand("Design Status", True)
 ' reinstate notepads
 app.Gui.SuppressNotepadDialogs = False
 If (retBool = False) Then
  Call app.Gui.StatusBarText("Design status - " + _
   Err.Description,epcbStatusFieldError)
  Exit Sub
 End If
 resultMess = "Design status was generated"
 ' Update status bar
 Call app.Gui.StatusBarText(resultMess,epcbStatusField1)
End Sub
'
' Testing code for the sample
'
Dim app
Set app = GetObject(,"MGCPCB.Application")
Call Execute_DesignStatus
```

Creating New Commands

You can extend a PCB application command set by creating your own commands. These commands behave as if they are part of the application.

This section describes the procedure to create commands.

Registering the command

To notify the server you are starting a new command you need to register the command:

```
Dim ccmd
Dim app
```

```
Set ccmd = app.Gui.RegisterCommand("<command_name>", True)
```

"RegisterCommand" on page 243 returns a "Command" on page 161 object. Concurrently, the command starts.

Define the event handlers

If you declare the variable that holds the Command object reference ("ccmd" in the example), you need to call Scripting.AttachEvents to bind the events of the Command object to its event handlers defined with the given prefix:

```
Call Scripting.AttachEvents(<object_reference_variable>,
 "<event_function_prefix>")
```

The Command object supports most of the interactive events that can occur during interactive operations. Refer to the "Command" on page 161 object events for more information.

Although you do not need to handle all events, it is advisable to define at least the "OnTerminate" on page 175 event to make sure the command environment is reset correctly. Also, if your application exits and your command is still running you should use "Gui.TerminateCommand" on page 250 to make sure the command exits.

Refer to "Working with Events" on page 38 for more information on event handling.

Optionally, you can add motion graphics during command execution. For more information, refer to the following example and "Using Motion Graphics" on page 36.

Example 6. Creating a New Command

```
'
' The example below shows the framework to create a new command
' - In the ConnectServer routine all references to the server are set
 up.
' The command is started with RegisterCommand
' - The ccmd_OnMouseClk routine handles left click mouse entries.
' On every click the motion graphics attached to the cursor gets
 changed.
' - Motion graphics is defined with the MotionGFX_Show routine
Dim ccmd
Dim docObj
Dim tagID
Dim delta
Dim clickCount
Dim doNotExit
doNotExit = True
' ================================
' Connect to server
' ================================
Private Sub ConnectServer()
 ' collect document object
 Set docObj = GetLicensedDoc(app)
 If (docObj Is Nothing) Then Exit Sub
 tagID = -1
 delta = 20
 ' Start the command
 Set ccmd = app.Gui.RegisterCommand("Showing motion graphics", True)
```

```
 Call Scripting.AttachEvents(ccmd, “ccmd”)
 ccmd.Unit = docObj.CurrentUnit
End Sub


‘ ================================
‘ Event handler for a mouse click
‘ ================================
Function ccmd_OnMouseClk(eButton, eFlags, dX, dY)
 ‘ switch the motion graphics on every mouse click
 If (eButton = epcbMouseButtonLeft) Then
  clickCount = clickCount + 1
  Call MotionGFX_Show(clickCount, dX, dY)
  If (clickCount >= 3) Then clickCount = 0
   ccmd_OnMouseClk = True ‘ return True to indicate
  End If
End Function
‘ ========================================
‘ Event handler for terminating the command
‘ ========================================
Sub ccmd_OnTerminate()
 ‘ terminate the command
 Call MotionGFX_Clear
 Set ccmd = Nothing
 ‘ exit the infinite loop
 doNotExit = False
End Sub
‘ ==============================
‘ Clearing the motion graphics
‘ ==============================
Public Sub MotionGFX_Clear()
 If (tagID < 0) Then Exit Sub
 If (docObj Is Nothing) Then Exit Sub
 Call docObj.ActiveViewEx.MotionGfxDeleteByTag(tagID)
 tagID = -1
End Sub


‘ ======================================
‘ Showing/switching the motion graphics
‘ ======================================
Public Sub MotionGFX_Show(mode, pntX, pntY)
 Dim v
 Dim color
 Dim width
 Dim delta2
 delta2 = delta / 2
 Set v = docObj.ActiveViewEx
 ‘ clear the graphics
 Call MotionGFX_Clear
 ‘ create a new color object
 Set color = app.Utility.NewColor
 ‘ set motion graphics width
```

```
 width = 0
 With v
 If (mode = 1) Then
  ‘ Draw a rectangle around the cursor
  tagID = .MotionGfxGetTag
  Call .MotionGfxPutRect(pntX - delta2, pntY - delta2, _
   epcbMotionGfxPointMoveXY, _
   pntX + delta2, pntY + delta2, epcbMotionGfxPointMoveXY, _
   tagID, width, color)
 ElseIf (mode = 2) Then
  ‘ Draw a circle around the cursor
  tagID = .MotionGfxGetTag
  Call .MotionGfxPutArcByCenter(pntX - delta2, pntY, _
   pntX, pntY, epcbMotionGfxPointMoveXY, _
   pntX + delta2, pntY, epcbMotionGfxPointMoveXY, _
   True, tagID, width, color)
 ElseIf (mode = 3) Then
  ‘ Draw a line at 45 degrees
  tagID = .MotionGfxGetTag
  Call .MotionGfxPutLine(pntX - delta2, pntY - delta2, _
   epcbMotionGfxPointMoveXY, _
   pntX + delta2, pntY + delta2, epcbMotionGfxPointMoveXY, _
   tagID, width, color)
 End If
 End With
End Sub
‘
‘ Testing code for the sample
‘
Dim app
Set app = GetObject(,”MGCPCB.Application”)
Call Scripting.AddTypeLibrary(“MGCPCB.Application”)
Call ConnectServer
‘ An infinite loop to prevent script from exiting
Do While doNotExit
 Scripting.Sleep(300)
Loop
```

Using Motion Graphics

The PCB applications use motion graphics to provide dynamic feedback to the user on the operation being performed. For instance, when you move a graphical element its ghost image is attached to the cursor until you place it or cancel the operation. These “ghost images” are called motion graphics.

The automation layer allows you to define your own motion graphic and remove it when it is no longer required. The View object has a set of motion graphics methods to achieve this.

To create motion graphics, use the following procedure:

Procedure

1. Request a tag id for the graphics you are about to create with the MotionGfxGetTag method.
2. Define the motion graphics geometry with one of the motion graphics geometry creation methods. The tag ID argument of these methods should be the ID you obtained in step 1. The coordinates you use for the motion graphics can be absolute (not related to the cursor position) or relative (to the current cursor position) or a combination of both, which allows for the creation of dynamically growing elements. Refer to the EPcbMotionGfxPointType for more information on coordinate specification.
3. (Optional) By definition, the graphics you define refreshes every time the mouse moves. If you want to paint the graphics immediately after creation, use the MotionGfxDrawAnyUndrawn method.
4. To remove the motion graphics, use the MotionGfxDeleteByTag method and supply it with the tag ID you obtained in 1.
5. (Optional) You can add additional motion graphics geometries by repeating the steps above. There are no limits to the number of elements you can define but too many elements may have an impact on interactive response times because they are redrawn for every mouse move.

Note:
It is important to assure the motion graphics is removed when you exit your command or application. Failing to do so will leave the motion graphics active and there is no way to clean it up afterwards.

You are not required to have a different tag ID for each separate motion graphics geometry. One tag can be used for several geometries. In this way you can logically group a set of motion graphics objects.

Interaction with Mouse Events

Motion graphics are usually used in relation with mouse events.

These mouse events are triggered when you “create a new command” on page 33 with an event handler. In general, the interaction between events of a command and motion graphics is as follows:

1. The “OnMouseClk” on page 169 event fires. Depending on the state of the command (first point, second point, etc.) the motion graphics is defined or adjusted (steps 1-3 above). If the last point is entered you can decide to remove the motion graphics (step 4 above).
2. The “OnMouseMove” on page 174 event fires. In most cases you do not need to adjust the motion graphics for this event. As soon as it is defined, the server will make sure it is refreshed and relocated automatically without the need to do this in a mouse move event.
3. The “OnTerminate” on page 175 event fires. The command is about to be terminated. Remove any motion graphics that is currently active.

Note:
The above describes the simplest case but for other events like the drag events the same rules apply.

Working with Events

While you are interacting with the automation server it is often desirable to be informed of the state changes in the server. The automation server allows you to create event handlers for this purpose.

Several object types generate events when their state is changing or has been changed. This section explains how to set-up and use event handlers.

CAUTION:
Processing done by event handlers is process intensive and causes the application to slow. To keep performance of the application high, reduce time spent in handlers. This is particularly true of events the fire often including: OnChange, OnSelectionChange, OnNotify, and OnDisplayViewExtentsChange.

Setting up the Event Handler

Event handlers in VBScript takes the form of id_event, where id is the event function prefix and event is the event name of the object.

For example, to define event handlers for application object events, OnOpenDocument, and Quit:

```
Sub appEvents_OnOpenDocument(Flags)
Sub appEvents_Quit()
```

To bind object events to event handlers, call the Scripting.AttachEvents method after creating the object. Scripting.AttachEvents method has two parameters, the object reference variable and the event function prefix.

```
Scripting.AttachEvents(<object_reference_variable>,
"<event_function_prefix>")
```

The following example attaches a Layout application object to its event handlers:

```
Dim appPCB
Set appPCB = GetObject(, "MGCPCB.ExpeditionPCBApplication")
Call Scripting.AttachEvents(appPCB, "appEvents")
```

Creating Event Handler Contents

Inside the event handler function you define the actions to take when the event occurs. While most events do not require a return value, some do.

For example “Command.OnMouseClk” on page 169, the returned value indicates if the event handler handled the event (True) or if you want the automation server to handle the event (False).

Note:
The automation server blocks until an event handler function exits. To avoid long response times, assure that the operations inside the event handler functions are kept to a minimum.

Exiting With Event Handlers

Normally a script exits after executing the last statement of the script. If a script includes event handlers, there must be an infinite loop that prevents the script from exiting, and a way to exit the infinite loop.

One way to construct an infinite loop is using Scripting.Sleep method, this infinite loop with an exit structure is shown in Example 7: Scripting.Sleep. Another way to prevent script from exiting is to use Scripting.DontExit property.

Example 7. Scripting.Sleep

```
' An infinite loop to prevent script from exiting
Dim doNotExit
doNotExit = True
Do While doNotExit
 Scripting.Sleep(300)
Loop
```

```
'Exit loop on the occurrence of some event by setting doNotExit to false
Sub <some_event>
 doNotExit = False
End Sub
```

Example 8: Application Event Handler Framework shows an example event handler framework for an application.

Example 8. Application Event Handler Framework

```
'
' The example below shows the event handler framework for an
' Layout application object.
'
Dim docPCB
Dim doNotExit
doNotExit = True
' ========================================
' The app is opening or reloading a document.
' Do design initialization or verification if new open.
' Make sure the docPCB object is reassigned if new open.
' ========================================
Sub appEvents_OnOpenDocument(Flags)
 Select Case Flags
  Case epcbOnOpenDocOpen
   Set docObj = app.ActiveDocument
   ' Custom document initialization & verifications goes here
```

```
  Case epcbOnOpenDocReload
    ' Skip the custom init & verify steps
 End Select

End Sub
```

```
' ================================
' The application is quitting
' ================================
Sub appEvents_Quit()
 MsgBox "Quitting the Application"
 'Exit the infinite loop
 doNotExit = False
 ' Make sure all objects are released
 Set docPCB = Nothing
 Set appPCB = Nothing
End Sub
```

```
'
' Testing code for the sample
'
Dim appPCB
Set appPCB = GetObject(,"MGCPCB.ExpeditionPCBApplication")
Call Scripting.AttachEvents(appPCB, "appEvents")
' collect document object
Set docPCB = GetLicensedDoc(appPCB)
' An infinite loop to prevent script from exiting
Do While doNotExit
 Scripting.Sleep(300)
Loop
```

Object Lifetime and the IsValid Method

The life span of an object that the automation layer returns is limited. In general, the simplest modification of an element will cause a deletion of the old element followed by the creation of a new element. In practice, this means the old element and its object references are no longer valid.

In order to notify the user of an object that it is no longer valid, the IsValid method is available for most of the objects. When the method returns a value of False, you should not use the object. If you do, errors will result.

Use the following rules of thumb to determine if you should verify the object validity:

- When a object reference is retrieved and no error occurs you can safely assume the object is valid. In the following, if the FindNet method returns a net object without an error, by definition the net object IsValid method will return True.

  ```
  Set netObj = pcbDoc.FindNet("GND")
  ```

- Object references that are assigned to global variables should be checked with IsValid if they are used elsewhere in the code.

- Changing the Selected and Highlighted properties on an object will not invalidate the object.
- Component modification operations Move method and Orientation property do not invalidate the object.
- Objects that are removed from the design by using their Delete method are invalid by default.
- When in doubt, use IsValid to make sure the object is still valid.

Modifying Graphical Elements

Graphical elements can be modified using the automation layer.

Component objects do not follow the rules that are explained in this section. Refer to “Modifying Components” on page 47 for more details on component manipulations.

Deleting elements

To delete an object from the design, you should use its Delete method. If the Delete method is applied to a collection, all elements in the collection are deleted.

Note:
Once an element is deleted, its object is no longer valid and its IsValid method returns False.

Creating elements

To create an object, use the Put... method. The majority of these methods are found in the Document object.

Modifying elements

Element modification is basically a create operation of a new element that is followed by a delete operation of the original element.

Example 9. Replacing fabrication layer text

```
‘
‘ The example below demonstrates the element manipulations
‘ on a fabrication layer text.
‘ It changes the text string of the incoming element ‘orgTextObj’
‘ to ‘newTextString’ by creating a new text element.
‘ The new element is placed in the design. The original element is
‘ deleted.
‘ ========================================================================
Sub FabTextReplace  (orgTextObj,newTextString)Dim docObj,resultMess
 Dim newTextObj‘ collect document object

 Set docObj = GetLicensedDoc(app)
 If (docObj Is Nothing) Then Exit Sub
```

```
 ‘ if the incoming element is empty then exit
 If (orgTextObj Is Nothing) Then Exit Sub

 ‘ create the new text element
 If (app.LockServer(True)) Then
  Set newTextObj = docObj.PutFabricationLayerText( newTextString, _
        orgTextObj.PositionX ,orgTextObj.PositionY, _
        orgTextObj.Type,orgTextObj.Side, _
        orgTextObj.Format.Height, _
        orgTextObj.Format.Orientation, _
        orgTextObj.Format.PenWidth, _
        orgTextObj.Format.Font, _
        orgTextObj.Format.Attributes, _
        orgTextObj.Format.HorizontalJust, _
        orgTextObj.Format.VerticalJust, _
        orgTextObj.Component, epcbUnitCurrent, _
         epcbAngleUnitDegrees)

  If (Err) Then
   Call app.Gui.StatusBarText(“An error occured while creating “ + _
   “new text : “ + vbNewLine + Err.Description,epcbStatusFieldError)
   app.UnlockServer(False)
   Exit Sub
  Else
   ‘ delete the original text
   orgTextObj.Delete
   If (Err) Then
    Call app.Gui.StatusBarText(“Could not delete original “ + _
    “text : “ + vbNewLine + Err.Description, _
    epcbStatusFieldError)
    app.UnlockServer(False)
    Exit Sub
   End if
   resultMess = “Fabrication text change was successfull”
   ‘ select the new text object and fit it on the board
   newTextObj.selected = True
   docObj.ActiveView.SetExtentsToSelection
   ‘ Update status bar
   Call app.Gui.StatusBarText(resultMess,epcbStatusField1)
  End If
  app.UnlockServer(True)
 End if
End Sub
‘
‘ Testing code for the sample
‘
Dim app
Set app = GetObject(,”MGCPCB.Application”)
Dim textObj

‘ Code to retrieve or create a fabrication text object goes here
  ‘....
  ‘....
```

```
‘ Execute text replacement on the textObject (uncomment the line below)
‘ Call FabTextReplace  (textObj,”Text String was replaced”)
```

Example 10. Moving user layer graphics elements

```
O’
‘ The example below moves all user layer graphics objects on
‘ user layer ‘Reserved_Area’ with a deltaX and deltaY
‘ (graphics inside the components is not included
‘Function  Move_UserGfx(gfxObj,deltaX,deltaY)
‘
 ========================================================================
===
‘ Description : Move a graphics object. This will recreate the graphics
‘ object in its new location
‘
‘ Parameters:
‘    - deltaX : Displacement in X
‘    - deltaY : Displacement in Y
‘
‘ Return values 1: If success
‘               -1: If failure
‘
‘
 ========================================================================
===
 on error resume next
 Dim retValue:retValue = 0
 Dim docObj,resultMess,geomObj,pntsArr,i,numPoints,newObj,layerObj

 ‘ collect document object
 Set docObj = GetLicensedDoc(app)
 If (docObj Is Nothing) Then Exit Function

 ‘ collect the points array of the user layer gfx
 Set geomObj = gfxObj.Geometry
 pntsArr = geomObj.PointsArray
 numPoints = UBound(pntsArr,2) +1

 ‘ displace the points array in x and y
 For i = 0 To numPoints - 1
  pntsArr(0,i) = pntsArr(0,i) + deltaX
  pntsArr(1,i) = pntsArr(1,i) + deltaY
 Next

 ‘ recreate the element in the document
 Set layerObj = docObj.UserLayers.Item(gfxObj.LayerName)
 Set newObj = docObj.PutUserLayerGfx( layerObj,geomObj.LineWidth, _
                numPoints,pntsArr, _
          geomObj.DisplayFilled,gfxObj.Component)
 If (Err) Then
  Call app.Gui.StatusBarText(“An error occured while creating a “ + _
```

```
            “user layer graphics element: “ + _
        vbNewLine + Err.Description,epcbStatusFieldError)
  Move_UserGfx = -1
 Else
  ‘ delete the original object
  gfxObj.Delete
  Move_UserGfx = 1
 End if
End Function

Sub Move_UserLayer(layerName,deltaX,deltaY)
‘
 ========================================================================
===
‘ Description : Move a user layer
‘               in its new location
‘
‘ Parameters:
‘    - layerName : the name of the user layer to move
‘    - deltaX    : Displacement in X
‘    - deltaY    : Displacement in Y
‘
‘ Return values 1: If success
‘               -1: If failure
‘
 ========================================================================
===
 Dim docObj,resultMess,gfxColl,i,gfxObj,retVal
 ‘ collect document object
 Set docObj = GetLicensedDoc(app)
 If (docObj Is Nothing) Then Exit Sub

 ‘ Collect the user layer graphics elements
 If (app.LockServer(True) = True ) Then
  Set gfxColl = docObj.UserLayerGfxs(epcbSelectAll,layerName,False)
  If (Err) Then
   Call app.Gui.StatusBarText(“An error occured while “ + _
            “collecting user layer graphics: “ + _
        vbNewLine + Err.Description,epcbStatusFieldError)
   app.UnlockServer False
   Exit Sub
  Else       ‘ Translate the elements
   ‘
   For i =1 To gfxColl.Count
    Set gfxObj = gfxColl.Item(i)
    retVal = Move_UserGfx(gfxObj,deltaX,deltaY)
   Next      ‘ unlock the server
   app.UnlockServer True
  End if
 End if
End Sub
‘
‘ Testing code for the sample
```

```
'
Dim app
Set app = GetObject(,"MGCPCB.Application")
Call Move_UserLayer("Reserved Area",100,100)
```

Working with the Object Collections

In some cases, the application automation model returns a collection of items that are not of the same type.

For example, if you use the ConnectedObjects Property on a Pin object, a collection returns that can contain a diversity of elements.

The returned collection in these cases is always an Objects Collection. The nature of this collection is such that its Item property always returns objects of type Object. This is the base type COM automation uses to store an object of any type.

To determine the object's type, use the following procedure:

Procedure

1. Determine the class type of the object. You do this by checking its ObjectClass property.

   This property returns the object's class type (EPcbObjectClassType). Based on this type you can cast the object to the parent class for the returned object type (refer to the description of EPcbObjectClassType for an overview of available parent classes).

2. Once you have the parent class, you can determine the actual object type with the Type2 property. This will allow you to cast the parent to the actual object (this rule does not apply for UserLayerText. This parent object has no sub objects.)

Examples

Example 11. Diverse Object Collection

```
Option Explicit
'
' The example below verifies all objects that are
' connected to a pin
'
Public Sub CheckConnectedObjects(pinObj)
```

```
Dim connObjsColl          ' The connected objects collection
Dim tempObj , condGfxObj
Dim objClType             ' The class type of the object
Dim i
```

```
Set connObjsColl = pinObj.ConnectedObjects
```

```
For i = 1 To connObjsColl.Count
 Set tempObj = connObjsColl.Item(i)
```

```
 '
 ' Check if we have an element that is conductor graphics
 ' (check the object class)
 If (tempObj.ObjectClass = epcbObjectClassConductorLayerGfx) Then
 ' 'cast' to a conductor layer graphics parent object
 Set condGfxObj = tempObj
```

```
 Select Case condGfxObj.Type2
```

```
  Case epcbGfxGeneratedPlane
   Dim genPlaneObj
   Set genPlaneObj = condGfxObj
   app.Gui.Display "Generated plane connected to " +_
    genPlaneObj.Net.Name
```

```
  Case epcbGfxBoardOutline
   Dim boutlineObj
   Set boutlineObj = condGfxObj
   app.Gui.Display "Board outline element "
  End Select
 End If
Next
End Sub
```

```
'
' The multiple use of variables
' can be avoided
' The routine below shows the reduced use of variables
'
Public Sub CheckConnectedObjects(pinObj)
```

```
Dim objType,tempObj
```

```
For Each tempObj In pinObj.ConnectedObjects
 ' we immediately check for the object type.
 ' However , there may by objects which do not
 ' have the Type2 property so one must check for
 ' error
 On Error Resume next
 objType = tempObj.Type2
 If (Err = 0) Then
  Select Case objType
   Case epcbGfxGeneratedPlane
    app.Gui.Display "Generated plane connected to " +_
     tempObj.Net.Name

   Case epcbGfxBoardOutline
    app.Gui.Display "Board outline element "
```

```
  End Select
 End If
Next
End Sub
```

Modifying Components

Component objects have their own set of modification methods that are different from the modification commands of regular graphical elements.

All operations will use the DRC rules to determine if the modification can be done. If not, an error issues.

A collection of drawing cells and mechanical cells can be constructed with the Document.Components property. This will return all drawing and/or mechanical cells that exists in the design. These can then be modified with the operations below. However, these types of cells cannot be placed from the cell library. They can only be manipulated if they are placed in the design.

Placing a component

The Component.Place method allows a component to be placed by specifying all the positional parameters such as X, Y, rotation and side. The method is used to place a component that is unplaced (if Component.Placed returns False) or to relocate a placed component. Refer to Example 12: Placing a component .

Note:
Apart from the Unplace operation, all other operations listed below are shorthand methods for the Component.Place method but they only execute a portion of this method.

Placing components interactively

To place components interactively you can use the Component.Place method and set the placement method argument value equal to epcbCompPlaceMethodOneAtATime. Refer to Example 13: Placing components interactively.

Moving a placed component

To move a placed component to an X, Y location use the Component.Move method.

Changing the rotation of a component

To change the rotation of a component, use the Component.Orientation property.

Changing the side of a component

To change the side of a component use the Component.Place method. There is no dedicated property that changes the side of a component. Refer to Example 14: Changing the side of a component .

Unplacing components

To unplace a single component, use the Component.Delete method. To unplace a collection of components, use the Components.Delete method on the collection. (see Component_Unplace example below)

Example 12. Placing a component

```
‘ The example below places a component with the supplied positional
‘ parameters
‘  Refdes: The reference designator of the component to place
‘  X,Y : The X and Y coordinates of the component.
‘  Angle: The angle of the component.
‘  topSide:Flag indicating if the component must be pushed to the
 bottom.
‘
Sub Component_Place (Refdes,X,Y,Angle,topSide)
```

```
Dim docObj,resultMess
Dim compObj
```

```
‘ collect document object
Set docObj = GetLicensedDoc(app)
If (docObj Is Nothing) Then Exit Sub
```

```
‘ Find the component in the components collection
Set compObj = docObj.FindComponent(Refdes)
If (compObj Is Nothing) Then
 Call app.Gui.StatusBarText(“Could not locate component : “ + RefDes _
  ,epcbStatusFieldError)
 Exit Sub
End If
```

```
‘ Place the component in the correct location
Call compObj.Place( X,Y,Angle,topSide,epcbAnchorNone, _
  epcbUnitCurrent,epcbAngleUnitDegrees)
If (Err) Then
 Call app.Gui.StatusBarText(“An error occured while placing “ + _
  Refdes + vbNewLine + Err.Description,epcbStatusFieldError)
 Exit Sub
Else
 resultMess = “Component ‘” + Refdes + “‘ was placed”
```

```
 ‘ select the collection and fit it on the board
 compObj.selected = True
 docObj.ActiveView.SetExtentsToSelection
```

```
 ‘ Update status bar
 Call app.Gui.StatusBarText(resultMess,epcbStatusField1)
```

```
End If
```

```
End Sub
‘
‘ Testing code for the sample
‘
Dim app
Set app = GetObject(,”MGCPCB.Application”)
Call Component_Place  (“R1”,0.0,-700.0,120.0,True)
Call Component_Place  (“R1”,0.0,-700.0,90.0,False)
```

Example 13. Placing components interactively

```
‘ The example below interactively places all components that match
‘ regular expression ‘RefdesExpr’
‘ The components are attached one by one to the cursor for placement
‘
‘ parameters :
‘  RefdesExpr : A regular expression defining the components that
‘               must be placed interactively
‘
Sub Component_Place_Interactive (RefdesExpr)
```

```
Dim docObj,resultMess
Dim compColl
```

```
‘ collect document object
Set docObj = GetLicensedDoc(app)
If (docObj Is Nothing) Then Exit Sub
```

```
‘ Find the component in the components collection
Set compColl = docObj.Components(epcbSelectAll,epcbCompAll, _
 epcbCelltypePackage,RefdesExpr)
If (compColl Is Nothing) Then
 Call app.Gui.StatusBarText(_
  “An error occured while collecting components: “ + _
  vbNewLine + Err.Description,epcbStatusFieldError)
 Exit Sub
End If
```

```
‘ Unplace the components that were collected
compColl.Sort
Call compColl.Place(epcbCompPlaceMethodOneAtATime)
```

```
If (Err) Then
 Call app.Gui.StatusBarText(“An error occured while placing “ + _
  RefdesExpr + vbNewLine + Err.Description,epcbStatusFieldError)
 Exit Sub
Else
 resultMess = “Placing ‘” + RefdesExpr + “‘ ...”
```

```
 ‘ Update status bar
 Call app.Gui.StatusBarText(resultMess,epcbStatusField1)
End if
```

```
End Sub
‘
‘ Testing code for the sample
‘
Dim app
Set app = GetObject(,”MGCPCB.Application”)
Call Component_Place_Interactive(“R1”)
```

Example 14. Changing the side of a component

```
‘
‘ The example below pushes a component to the opposite side
‘ Refdes   : The component to push
‘
Sub Component_Push (Refdes)
```

```
Dim docObj,resultMess
Dim compObj,topFlag
```

```
‘ collect document object
Set docObj = GetLicensedDoc(app)
If (docObj Is Nothing) Then Exit Sub
```

```
‘ Find the component in the components collection
Set compObj = docObj.FindComponent(Refdes)
If (compObj Is Nothing) Then
 Call app.Gui.StatusBarText(“Could not locate component: “ + _
  RefDes,epcbStatusFieldError)
 Exit Sub
End If
```

```
‘
‘ To push the component we collect the previous
‘ settings and flip the top/bottom flag
‘
If (compObj.Layer = 1) Then
 topFlag = False
Else
 topFlag = True
End if
```

```
Call compObj.Place( compObj.PositionX,compObj.PositionY, _
    compObj.Orientation,topFlag,compObj.Anchor, _
    epcbUnitCurrent,epcbAngleUnitDegrees)
If (Err) Then
 Call app.Gui.StatusBarText("An error occured while pushing " + _
  Refdes + vbNewLine + Err.Description,epcbStatusFieldError)
 Exit Sub
Else
 resultMess = "Component '" + Refdes + "' was pushed"
```

```
 ' select the collection and fit it on the board
 compObj.selected = True
 docObj.ActiveView.SetExtentsToSelection
```

```
 ' Update status bar
 Call app.Gui.StatusBarText(resultMess,epcbStatusField1)
```

```
End If
```

```
End Sub
'
' Testing code for the sample
'
Dim app
Set app = GetObject(,"MGCPCB.Application")
Call Component_Push   ("R1")
```

Using PointsArrays

All geometry types that are used by the automation layer use a point array to represent the element's geometry.

Each entry in the point array has three values that define the X, Y and R values of a point. X and Y are the coordinates of the point and R is the radius if an arc needs to be defined.

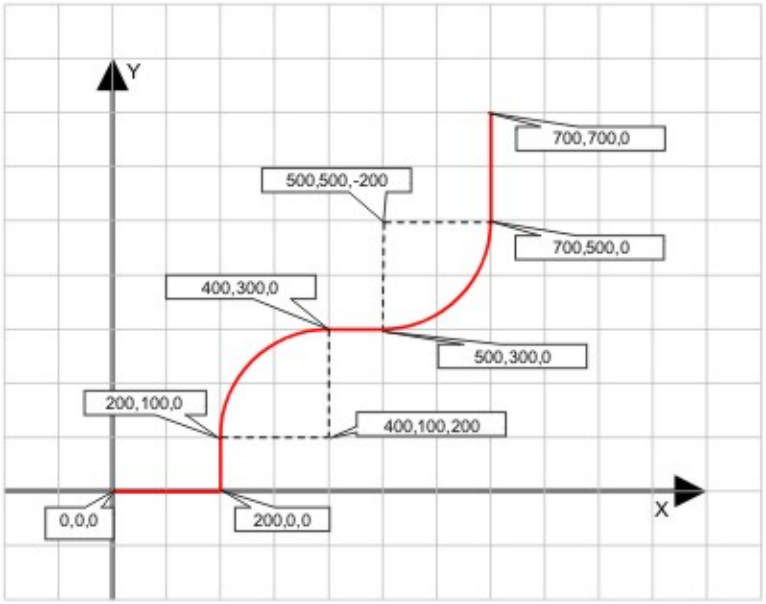
Figure 1: Points Array shows the conventions for defining point arrays:

- **Non-arcs** — points are defined with an X, Y coordinate, The R value is always 0.
- **Arcs** — are defined with 3 points.
  - The first point is the start point of the arc (R value is 0).
  - The second point is the center point of the arc. The R value for this point is equal to the radius of the arc. If the R value is negative, a counter clockwise arc is defined. If the R value is positive, a clockwise arc is defined.
  - The third point is the end point of the arc (R value is 0).

- **Circles** — are defined with three points, a point on the circumference (R=0), the circle center (R=radius) and finally back to the starting point (R=0) on the circumference. The “Utility.CreateCircleXYR” on page 267 method constructs the points array for a circle.

- **Rectangles** — are defined using the “Utility.CreateRectXYR” on page 268 method. It constructs the points array for a rectangle so that it is recognized as a rectangle and not as a regular polygon.

Figure 1. Points Array



Example 15. Points Array

```
‘
‘ The example below shows how to define the points array for the
‘ the element shown in the drawing. It is placed in the design as a
‘ Assembly layer top element
‘
‘
Sub Document_PutFabricationLayerGfx  ()
```

```
Dim docObj,resultMess
Dim pntsArr(),numPoints
Dim fabGfxObj
```

```
‘ collect document object
Set docObj = GetLicensedDoc(app)
If (docObj Is Nothing) Then Exit Sub
```

```
‘ fill the array with points
ReDim pntsArr(2,8)
numPoints = 9
pntsArr(0,0) = 0.0   :pntsArr(1,0) = 0.0   :pntsArr(2,0) =    0.0
pntsArr(0,1) = 200.0 :pntsArr(1,1) = 0.0   :pntsArr(2,1) =    0.0
pntsArr(0,2) = 200.0 :pntsArr(1,2) = 100.0 :pntsArr(2,2) =    0.0
pntsArr(0,3) = 400.0 :pntsArr(1,3) = 100.0 :pntsArr(2,3) =  200.0
pntsArr(0,4) = 400.0 :pntsArr(1,4) = 300.0 :pntsArr(2,4) =    0.0
pntsArr(0,5) = 500.0 :pntsArr(1,5) = 300.0 :pntsArr(2,5) =    0.0
pntsArr(0,6) = 500.0 :pntsArr(1,6) = 500.0 :pntsArr(2,6) = -200.0
pntsArr(0,7) = 700.0 :pntsArr(1,7) = 500.0 :pntsArr(2,7) =    0.0
pntsArr(0,8) = 700.0 :pntsArr(1,8) = 700.0 :pntsArr(2,8) =    0.0
```

```
set fabGfxObj = docObj.PutFabricationLayerGfx( epcbFabAssembly, _
     epcbSideTop,4.0,numPoints,pntsArr, _
     False,Nothing,epcbUnitCurrent)
```

```
If (Err) Then
 Call app.Gui.StatusBarText(_
  “An error occured while creating the the object: “ + _
  vbNewLine + Err.Description,epcbStatusFieldError)
 Exit Sub
Else
 resultMess = “Fabrication layer element was created”
```

```
 ‘ select the object and fit it on the board
 fabGfxObj.selected = True
 docObj.ActiveView.SetExtentsToSelection
```

```
 ‘ Update status bar
 Call app.Gui.StatusBarText(resultMess,epcbStatusField1)
```

```
End If
```

```
End Sub
‘
‘ Testing code for the sample
‘
Dim app
Set app = GetObject(,”MGCPCB.Application”)
Call Document_PutFabricationLayerGfx
```

Using Wildcard Expressions

You can use wildcard characters to form wildcard expressions.

The following wildcard characters are allowed to create a wildcard expression:

%	A single character.
*	Zero or more characters.

The following shows a few examples of wildcard expressions:

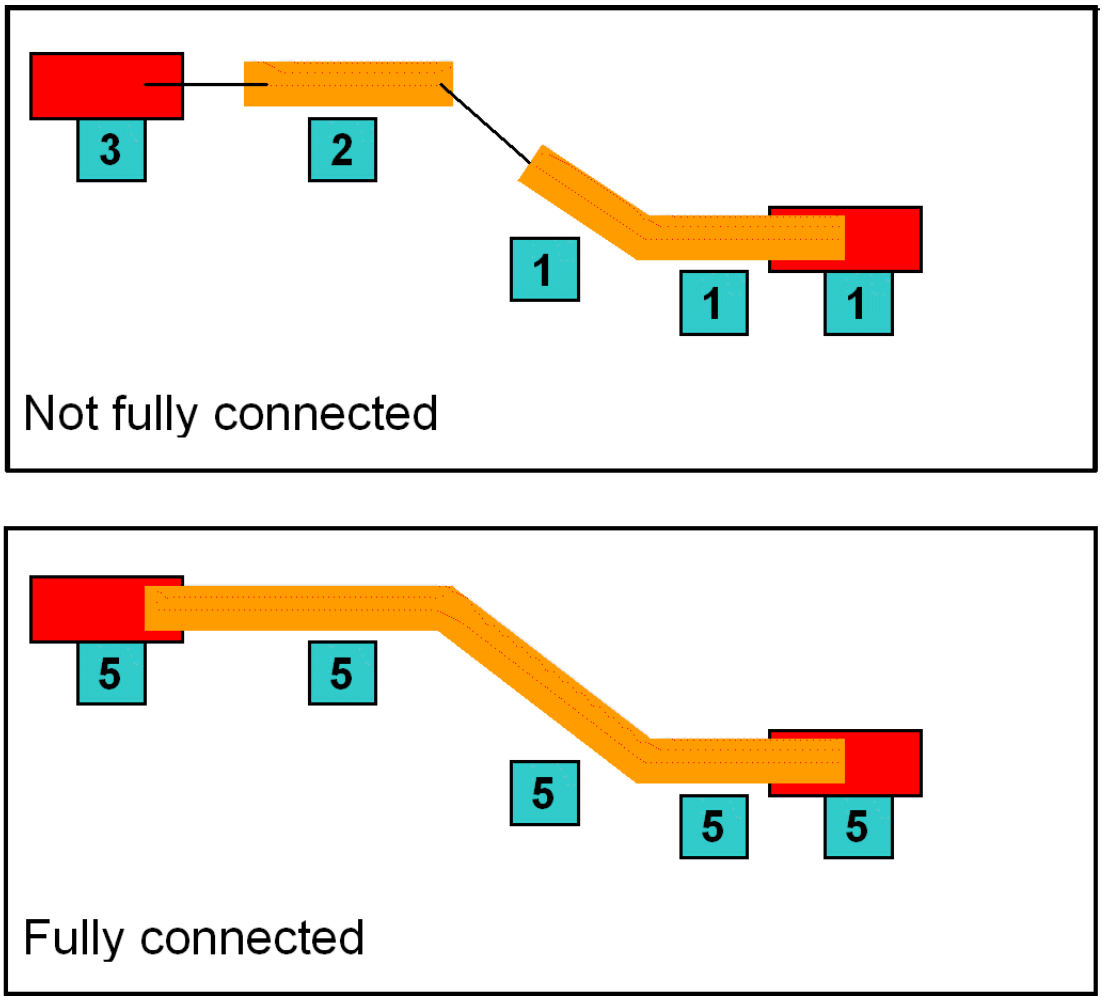
String	Expression	Result
PN234590	%%%%%%%%	PN234590
PN1234	%%%%%%%%	<no match>
PN234590	PN*	PN234590
PN1234	PN*	PN1234
PN1234	PN%%%%	PN1234
PN12345	PN%%%%	<no match>

Understanding the SubNet Property

All conductive items in PCB are assigned to a net. However, the net and its net name offer no additional information to indicate if two or more items are physically connected. The subnet number provides this information.

All objects with the same subnet number are physically connected together. Figure 2: SubNets Example shows this principle:

Figure 2. SubNets Example



In the top example the net is not fully connected. The elements that are connected have the same subnet number (in this example, 1 2 and 3). During routing, these subnet numbers may change to reflect the new connected status. In the bottom example the net is fully connect. All elements have the same subnet number (in this example, 5).

Note:
The absolute value of the subnet number is irrelevant, it is the equality of the subnet numbers that is used to verify if two items are connected together.

Working with Padstacks

The automation layer offers several methods to access padstack information.

The methods are described in this section.

Collecting padstacks in the design

To collect the padstacks that are used in the design you can use Document.Padstacks("<name_expr>") where <name_expr> indicates specific padstack names. This property returns a collection of Padstack objects.

The Padstack.IsModified method determines if the padstack has been changed compared to its definition in the padstack library. If it returns False, the object describes the library padstack. If it returns True, the object describes the modifications to the padstack. Modifications may have been introduced by the pads processor.

You may find several padstack objects with the same padstack name in a Padstacks collection, each with modifications. Only one padstack has an IsModified method with a value of False. That padstack represents the padstack in the library.

You can use the Padstack.Pins and Padstack.Vias to determine which elements in the design use the padstack.

Creating new padstacks

You can create new padstacks using the Document.PutPadstack method. This method also allows you to add the padstack to the central library.

Note:
Negative pads that are added with the planes processor are not stored in the Pads collection of a PadStack object. This type of pad can be retrieved with the PadstackObject.Pads property.

Manipulating objects that use padstacks

If you need to collect objects that use padstacks you should use the Document.PadstackObjects property. The property returns a collection of PadstackObject objects.

Note:
There are other objects that expose a PadstackObjects property. However, Document.PadstackObjects is used in most cases.

Figure 3. Padstack Object and Padstacks

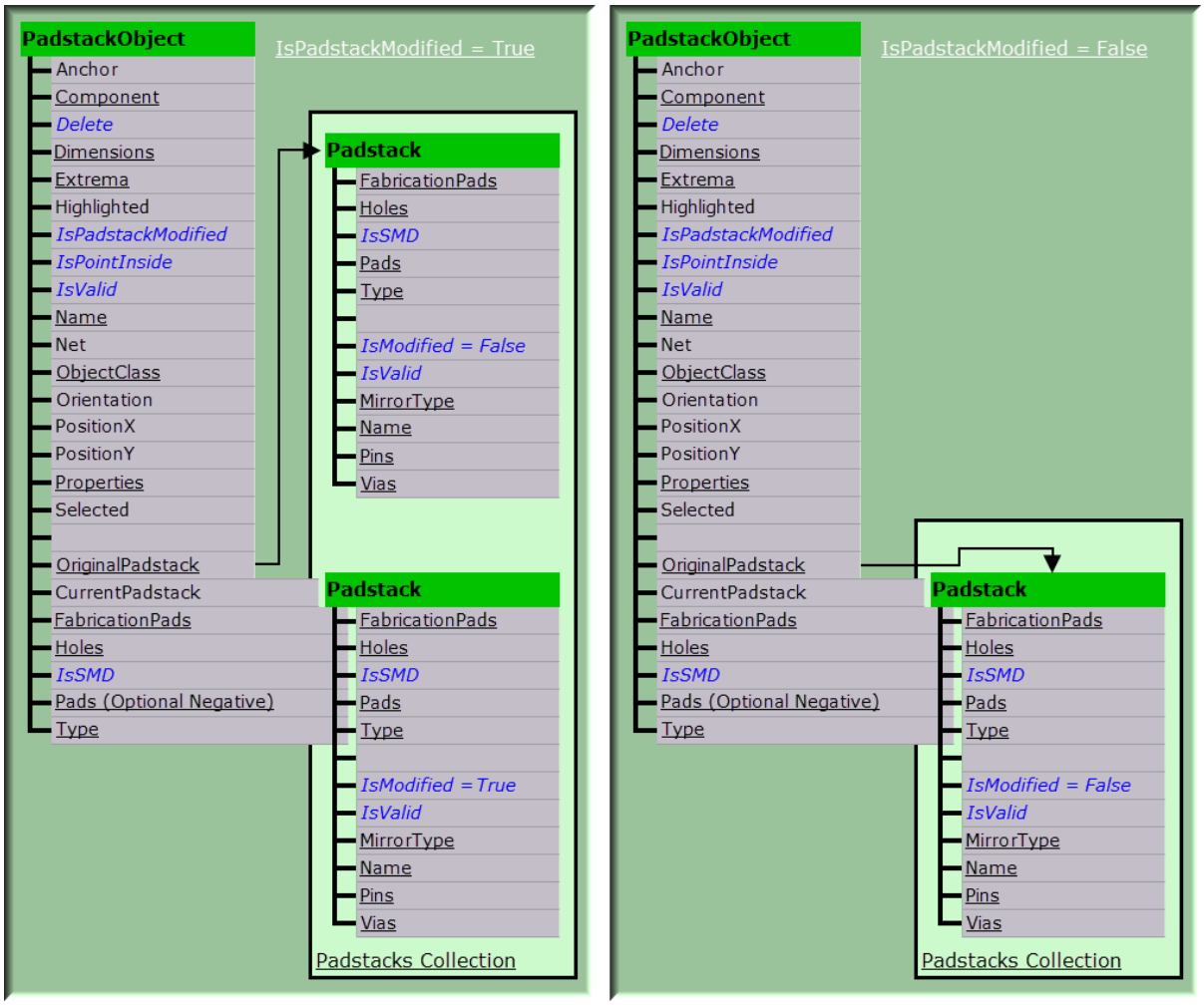


Figure 3: Padstack Object and Padstacks shows the relations between a padstack object and its padstack(s).

- If the IsPadstackModified method returns True, the padstack for the object has been modified compared to the definition of the padstack in the padstack library. The CurrentPadstack and OriginalPadstack (as in the library) objects will be different.

- If the IsPadstackModified method returns False, the padstack for the object has not been changed. The CurrentPadstack and OriginalPadstack (as in the library) objects are identical and reference the padstack definition as it was defined in the padstack library.

Note:
As can be seen in the figure that some properties and methods in the PadstackObject match those of the CurrentPadstack object. For example, **pstackObj.FabricationPads** is the same as pstackObj.CurrentPadstack.FabricationPads. They have been exposed this way to make it easier to access them. The only exception to this rule is the PadstackObject.Pads property. Apart from exposing the pads in the padstack, you have the option to review the negative pads that may have been added by the planes generator.

The remaining properties and methods of the PadstackObject object can be used to collect details on the element that uses padstacks.

Display Control Automation

Use automation to set the Display Control options to update the display of layers and objects in the design and control object selectivity.

The automation interface for manipulating the Display Control settings consists of these properties:

- **Four-state properties** — used to communicate the On/Off Enabled/Disabled state of the controls:
  - Option
  - Visible
  - Selectable

    This example uses the Visible property to set the Layer 1 visibility to the OnEnabled state:

    ```
    doc.ActiveViewEx.DisplayControl.Visible("LayerControl.1") =
      epcbGraphicsItemStateOnEnabled
    ```

- **String properties** — used to communicate the state of complex items such as sliders, radio buttons, and combo boxes:
  - StringOption

    This example uses the StringOption property to set the Pan Sensitivity slider value:

    ```
    doc.ActiveViewEx.DisplayControl.StringOption(
      "Global.Option.PanSensitivity") = "2"
    ```

  - StringOption for color settings (3D tab only)

    The 3D tab, which appears in the Display Control if you add a 3D view in Layout, requires using the StringOption property to set color values. All other tabs use the Global.Color property to set the color.

    This example uses the StringOption property to set the Origin Marker color on the 3D tab:

    ```
    doc.ActiveViewEx.DisplayControl.StringOption(
      "[Virtual].Global.3D.Options.OriginMarker.Color") = "9900CC"
    ```

    The StringOption property requires a hexadecimal value to set color in the 3D tab. To determine the color number, you can calculate the hex color from RGB values. There are several color wheel calculator websites where you can translate your RGB color values into hex values.

    Optionally, if you enable the Display Control Automation Log Mode (see Setting Layer Visibility in Display Control for an example of using this mode), you can click a color icon in the 3D tab, select the desired color in the Color Fill Pattern dialog box, and then observe the hex value that is echoed to the Message Window.

- **IMGCPCBColorPattern properties** — used to communicate display color, pattern, and transparency settings:

  - Global.Color

    This example uses the Global.Color property to set the LayerControl.1 color:

```
doc.ActiveViewEx.DisplayControl.Global.Color("LayerControl.1") =
  application.Utility.NewColorPattern(0, 102, 255, 100, 0, False,
    True)
```

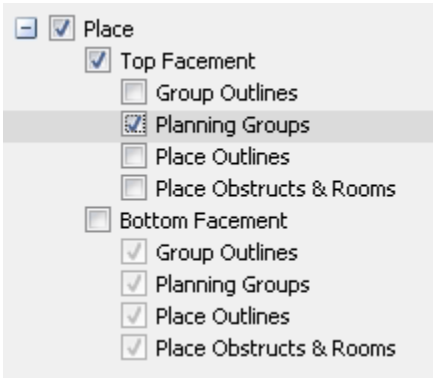
The four-state values represent the four possible states of any given check box in the Display Control. Figure 4: Display Control Graphic Item States shows each value with its corresponding Display Control item state:

Figure 4. Display Control Graphic Item States

```
enum EPcbGraphicsItemState {
    epcbGraphicsItemStateOffEnabled         = 0, ->
    epcbGraphicsItemStateOnEnabled          = 1, ->
    epcbGraphicsItemStateOffNotEnabled      = 2, ->
    epcbGraphicsItemStateOnNotEnabled       = 3, ->
} EPcbGraphicsItemState;
```

When a NotEnabled item is set to one of the Enabled values, all of its parent states are automatically set to OnEnabled to maintain a valid state in the dialog box and internal settings. Similarly, when an item is set to any value except OnEnabled, all of its child states are automatically set to NotEnabled. See Figure 5: Display Control Enabled and NotEnabled Parent-Child States for examples of these conditions.

Figure 5. Display Control Enabled and NotEnabled Parent-Child States



For example, the following code sets the visibility of the Planning Groups item under Top Facement to OnEnabled:

```
doc.ActiveViewEx.DisplayControl.Visible("Group.Outline.Top") =
  epcbGraphicsItemStateOnEnabled
```

If Top Facement and Place visibility were in the NotEnabled state, they would automatically be set to OnEnabled as a result of the change in the state of the child, Planning Groups.

The following example turns off visibility for the Place and Route Objects items on the Edit tab:

```
With doc.ActiveViewEx.DisplayControl
  .Option("Option.PlaceObjects")= epcbGraphicsItemStateOffEnabled
  .Option("Option.RouteObjects.Enabled")= epcbGraphicsItemStateOffEnabled
End With
```

Child items that were previously in the OnEnabled state are automatically set to OnNotEnabled (Figure 6: Route Objects (Edit Tab)). Thus, their visibility can be controlled by the parent's visibility setting.

Figure 6. Route Objects (Edit Tab)

As an aid to learning the Display Control automation interface, you can enable the Display Control Automation Log Mode by setting the MGC_ENABLE_DC_AUTOMATION_LOG environment variable to 1. This will echo all Display Control user actions or automation to the Message Window as VBScript code snippets. See Setting Layer Visibility in Display Control for an example of using this mode.

For an example that uses an MS Excel spreadsheet to read and write Display Control values, see *DisplayControl.xls* in the *\SDD_HOME\standard\examples\pcb\Automation\Excel* directory.

Display Control Dialog Box - Edit Tab Controls
Display Control Dialog Box - Objects Tab Controls
Display Control Dialog Box - Graphic Tab Controls
Display Control Dialog Box - Fab Tab Controls
Display Control Dialog Box - 3D Tab Controls
Setting Layer Visibility in Display Control

Display Control Dialog Box - Edit Tab Controls

To access: Document.ActiveViewEx.DisplayControl.***property.control_id***, where ***property*** is either either Option, Visible, Selectable, StringOption, or Global.color, depending on the control (see Display Control Automation for more information).

Use automation to set options on this tab to control the display of layers and the global display settings.

Note:
Tables in this section do not include color controls. See Usage Notes for more information.

Objects

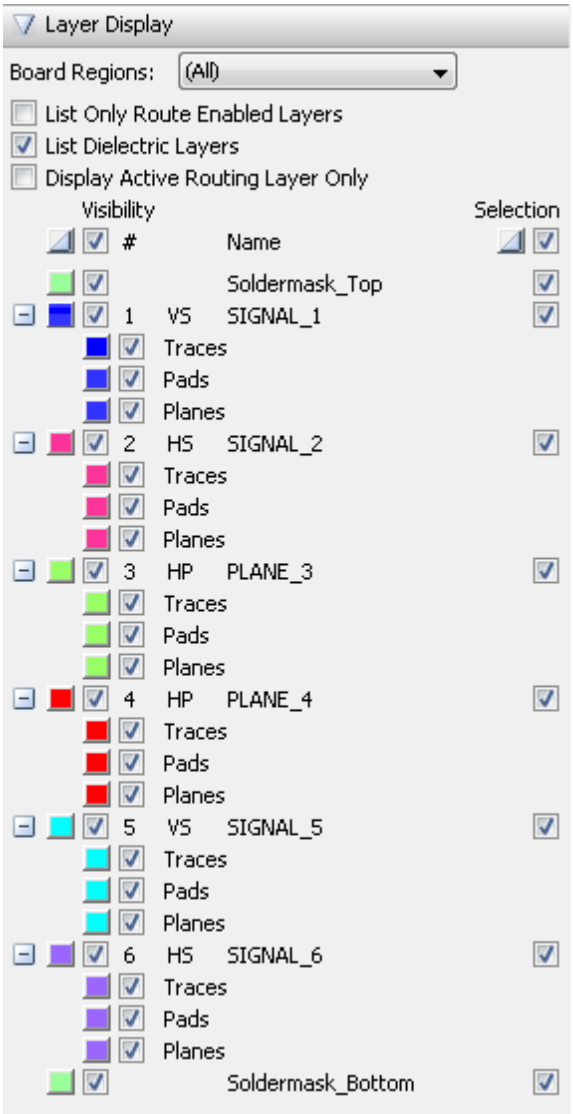


Table 3. Layer Display Section

Object	Description	Control ID
Board Regions[1]	Dropdown list	None
List Only Route Enabled Layers	Check box	None
List Dielectric Layers[1]	Check box	None
Display Active Routing Layer Only	Check box	Option.ActiveLayerOnly
Visibility	Check box	None
Selection[2]	Check box	None
Soldermask_Top (Visibility)[1]	Check box	RigidFlex.SolderMask.1657324667167310656
Soldermask_Top (Selection)[1 2]	Check box	RigidFlex.SolderMask.1657324667167310656
1 VS SIGNAL_1 (Visibility)	Check box	LayerControl.1
1 VS SIGNAL_1 (Selection)[2]	Check box	LayerControl.1
Traces	Check box	Copper.Trace.1
Pads	Check box	Copper.Pad.1
Planes	Check box	Copper.Plane.Data.1
2 HS SIGNAL_2 (Visibility)	Check box	LayerControl.2
2 HS SIGNAL_2 (Selection)[2]	Check box	LayerControl.2
Traces	Check box	Copper.Trace.2
Pads	Check box	Copper.Pad.2
Planes	Check box	Copper.Plane.Data.2
3 HP PLANE_3 (Visibility)	Check box	LayerControl.3
3 HP PLANE_3 (Selection)[2]	Check box	LayerControl.3
Traces	Check box	Copper.Trace.3
Pads	Check box	Copper.Pad.3
Planes	Check box	Copper.Plane.Data.3
4 HP PLANE_4 (Visibility)	Check box	LayerControl.4

1. Available only with RigidFlex designs.
2. Available only in Select Mode.

Table 3. Layer Display Section (continued)

Object	Description	Control ID
4 HP PLANE_4 (Selection)[2]	Check box	LayerControl.4
Traces	Check box	Copper.Trace.4
Pads	Check box	Copper.Pad.4
Planes	Check box	Copper.Plane.Data.4
5 VS SIGNAL_5 (Visibility)	Check box	LayerControl.5
5 VS SIGNAL_5 (Selection)[2]	Check box	LayerControl.5
Traces	Check box	Copper.Trace.5
Pads	Check box	Copper.Pad.5
Planes	Check box	Copper.Plane.Data.5
6 HS SIGNAL_6 (Visibility)	Check box	LayerControl.6
6 HS SIGNAL_6 (Selection)[2]	Check box	LayerControl.6
Traces	Check box	Copper.Trace.6
Pads	Check box	Copper.Pad.6
Planes	Check box	Copper.Plane.Data.6
Soldermask_Bottom (Visibility)[1]	Check box	RigidFlex.SolderMask.1657324718706918208
Soldermask_Bottom (Selection)[1 2]	Check box	RigidFlex.SolderMask.1657324718706918208

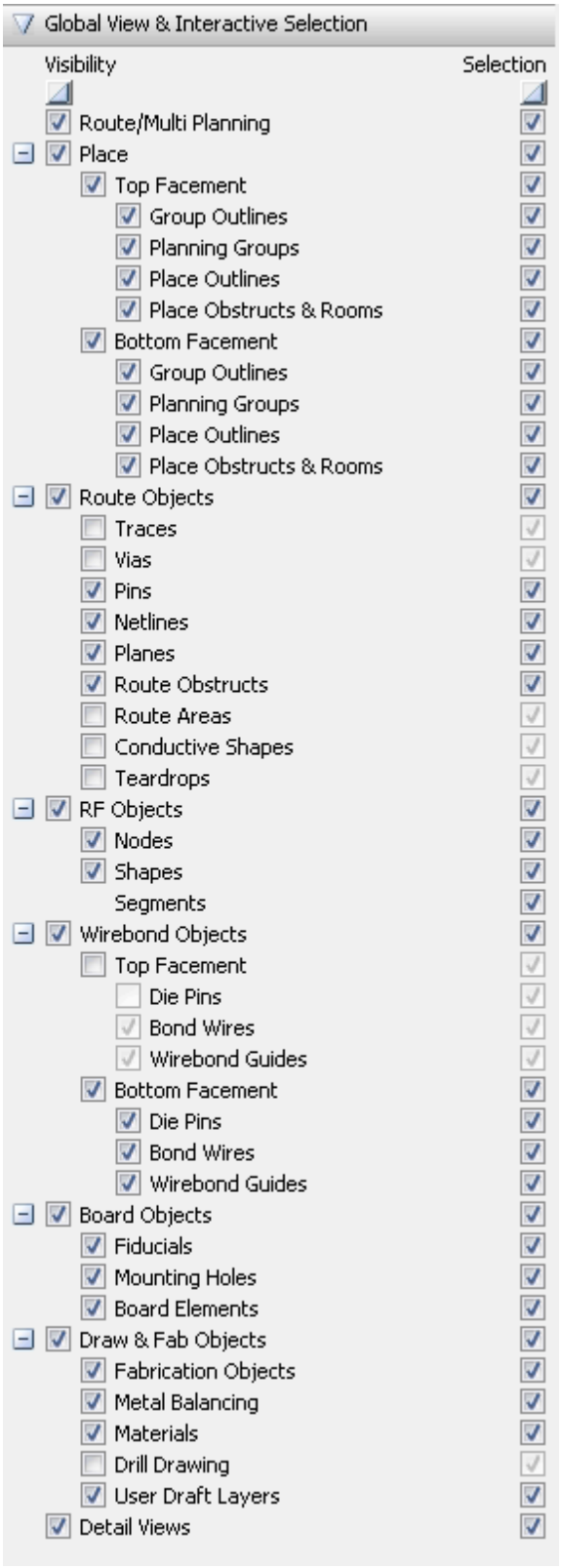

Global View & Interactive Selection
Visibility
Selection
Route/Multi Planning
Place
Top Facement
Group Outlines
Planning Groups
Place Outlines
Place Obstructs & Rooms
Bottom Facement
Group Outlines
Planning Groups
Place Outlines
Place Obstructs & Rooms
Route Objects
Traces
Vias
Pins
Netlines
Planes
Route Obstructs
Route Areas
Conductive Shapes
Teardrops
RF Objects
Nodes
Shapes
Segments
Wirebond Objects
Top Facement
Die Pins
Bond Wires
Wirebond Guides
Bottom Facement
Die Pins
Bond Wires
Wirebond Guides
Board Objects
Fiducials
Mounting Holes
Board Elements
Draw & Fab Objects
Fabrication Objects
Metal Balancing
Materials
Drill Drawing
User Draft Layers
Detail Views

Table 4. Global View & Interactive Selection Section

Object	Description	Control ID
Visibility	Button	None
Selection*	Button	None
Route/Multi planning (Visibility)	Check box	Option.Planning.Enabled
Route/Multi planning (Selection)*	Check box	Option.Planning.Selectable
Place (Visibility)	Check box	Option.PlaceObjects
Place (Selection)*	Check box	Option.PlaceObjects.Selectable
Top Facement (Visibility)	Check box	Option.PlaceObjects.Parts.Top
Top Facement (Selection)*	Check box	Option.PlaceObjects.Parts.Top.Selectable
Group Outlines (Visibility)	Check box	Group.Outline.Top
Group Outlines (Selection)*	Check box	Group.Outline.Top
Planning Groups (Visibility)	Check box	Group.Outline.Bubble.Top
Planning Groups (Selection)*	Check box	Group.Outline.Bubble.Top
Place Outlines (Visibility)	Check box	Part.PlaceOutline.Top
Place Outlines (Selection)*	Check box	Part.PlaceOutline.Top
Place Obstructs & Rooms (Visibility)	Check box	Option.PlaceObjects.ObstructsAndRooms.Top
Place Obstructs & Rooms (Selection)*	Check box	Option.PlaceObjects.ObstructsAndRooms.Top.Selectable
Bottom Facement (Visibility)	Check box	Option.PlaceObjects.Parts.Bottom
Bottom Facement (Selection)*	Check box	Option.PlaceObjects.Parts.Bottom.Selectable
Group Outlines (Visibility)	Check box	Group.Outline.Bottom
Group Outlines (Selection)*	Check box	Group.Outline.Bottom
Planning groups (Visibility)	Check box	Group.Outline.Bubble.Bottom
Planning groups (Selection)*	Check box	Group.Outline.Bubble.Bottom
Place Outlines (Visibility)	Check box	Part.PlaceOutline.Bottom
Place Outlines (Selection)*	Check box	Part.PlaceOutline.Bottom
Place Obstructs & Rooms (Visibility)	Check box	Option.PlaceObjects.ObstructsAndRooms.Bottom
Place Obstructs & Rooms (Selection)*	Check box	Option.PlaceObjects.ObstructsAndRooms.Bottom.Selectable

Table 4. Global View & Interactive Selection Section (continued)

Object	Description	Control ID
Route Objects (Visibility)	Check box	Option.RouteObjects.Enabled
Route Objects (Selection)*	Check box	Option.RouteObjects.Selectable
Traces (Visibility)	Check box	Option.Traces.Enabled
Traces (Selection)*	Check box	Option.Traces.Selectable
Vias (Visibility)	Check box	Option.Vias.Enabled
Vias (Selection)*	Check box	Option.Vias.Selectable
Pins (Visibility)	Check box	Option.Pins.Enabled
Pins (Selection)*	Check box	Option.Pins.Selectable
Netlines (Visibility)	Check box	Option.Netlines.Enabled
Netlines (Selection)*	Check box	Option.Netlines.Selectable
Planes (Visibility)	Check box	Option.Planes.Enabled
Planes (Selection)*	Check box	Option.Planes.Selectable
Route Obstructs (Visibility)	Check box	Option.RouteObstructs.Enabled
Route Obstructs (Selection)*	Check box	Option.RouteObstructs.Selectable
Route Areas (Visibility)	Check box	Option.RouteAreas.Enabled
Route Areas (Selection)*	Check box	Option.RouteAreas.Selectable
Conductive Shapes (Visibility)	Check box	Option.ConductiveShapes.Enabled
Conductive Shapes (Selection)*	Check box	Option.ConductiveShapes.Selectable
Teardrops (Visibility)	Check box	Option.Teardrops.Enabled
Teardrops (Selection)*	Check box	Option.Teardrops.Selectable
RF Objects (Visibility)	Check box	Option.RFObjects.Enabled
RF Objects (Selection)*	Check box	Option.RFObjects.Selectable
Nodes (Visibility)	Check box	Option.RFNodes.Enabled
Nodes (Selection)*	Check box	Option.RFNodes.Selectable
Shapes (Visibility)	Check box	Option.RFShapes.Enabled
Shapes (Selection)*	Check box	Option.RFShapes.Selectable

Table 4. Global View & Interactive Selection Section (continued)

Object	Description	Control ID
Segments (Selection)*	Check box	Option.RFSegments.Selectable
Wirebond Objects (Visibility)	Check box	Option.WirebondItems.Enabled
Wirebond Objects (Selection)*	Check box	Option.WirebondItems.Selectable
Top Facement (Visibility)	Check box	Option.WirebondItemsTop.Enabled
Top Facement (Selection)*	Check box	Option.WirebondItemsTop.Selectable
Die Pins (Visibility)	Check box	Option.DiePinsTop.Enabled
Die Pins (Selection)*	Check box	Option.DiePinsTop.Selectable
Bond Wires (Visibility)	Check box	Option.BondWiresTop.Enabled
Bond Wires (Selection)*	Check box	Option.BondWiresTop.Selectable
Wirebond Guides (Visibility)	Check box	WirebondObjects.WirebondGuides.Top
Wirebond Guides (Selection)*	Check box	WirebondObjects.WirebondGuides.Top
Bottom Facement (Visibility)	Check box	Option.WirebondItemsBottom.Enabled
Bottom Facement (Selection)*	Check box	Option.WirebondItemsBottom.Selectable
Die Pins (Visibility)	Check box	Option.DiePinsBottom.Enabled
Die Pins (Selection)*	Check box	Option.DiePinsBottom.Selectable
Bond Wires (Visibility)	Check box	Option.BondWiresBottom.Enabled
Bond Wires (Selection)*	Check box	Option.BondWiresBottom.Selectable
Wirebond Guides (Visibility)	Check box	WirebondObjects.WirebondGuides.Bottom
Wirebond Guides (Selection)*	Check box	WirebondObjects.WirebondGuides.Bottom
Board Objects (Visibility)	Check box	Option.BoardObjects.Enabled
Board Objects (Selection)*	Check box	Option.BoardObjects.Selectable
Fiducials (Visibility)	Check box	Option.Fiducials.Enabled
Fiducials (Selection)*	Check box	Option.Fiducials.Selectable
Mounting Holes (Visibility)	Check box	Option.Holes.Enabled
Mounting Holes (Selection)*	Check box	Option.Holes.Selectable
Board Elements (Visibility)	Check box	Option.BoardElements.Enabled

Table 4. Global View & Interactive Selection Section (continued)

Object	Description	Control ID
Board Elements (Selection)*	Check box	Option.BoardElements.Selectable
Draw & Fab Objects (Visibility)	Check box	Option.DrawFabObjects.Enabled
Draw & Fab Objects (Selection)*	Check box	Option.DrawFabObjects.Selectable
Fabrication Objects (Visibility)	Check box	Option.FabricationObjects
Fabrication Objects (Selection)*	Check box	Option.FabricationObjects.Selectable
Metal Balancing (Visibility)	Check box	Option.CopperBalancing.Enabled
Metal Balancing (Selection)*	Check box	Option.CopperBalancing.Selectable
Materials (Visibility)	Check box	Option.Materials.Enabled
Materials (Selection)*	Check box	Option.Materials.Selectable
Drill Drawing (Visibility)	Check box	Option.Fabrication.DrillDrawing
Drill Drawing (Selection)*	Check box	Option.Fabrication.DrillDrawing.Selectable
User Draft Layers (Visibility)	Check box	Option.UserDraftLayers.Enabled
User Draft Layers (Selection)*	Check box	Option.UserDraftLayers.Selectable
DetailViews (Visibility)	Check box	Fabrication.DetailViews
DetailViews (Selection)*	Check box	Fabrication.DetailViews

* Available only in Select Mode.

Usage Notes

- Most color settings in Display Control use the following pattern:

  ```
  Global.Color(<control_id>)
  ```

  For example:

  ```
  doc.ActiveViewEx.DisplayControl.Global.Color("LayerControl.1") =
    application.Utility.NewColorPattern(0, 102, 255, 100, 0, False,
   True)
  ```

  Although they typically match, the control IDs of a color setting and its associated check box could be different. To verify a control ID, enable Display Control Automation Log Mode to echo the control ID to the Message Window. See Setting Layer Visibility in Display Control for more information.

- RigidFlex.Soldermask values in Table 1-3 (Soldermask_Top and Soldermask_Bottom layers) represent the unique id of the layer. The Soldermask is an insulation layer. You acquire this value by getting either the MultiBoard layer stackup or the Document.LayerStackup(true), both of which include the insulation layers.

  To enable RigidFlex functionality in the design, set the Environment.DesignTechnology property to epcbDesignTechnologyRigidFlexDesign. RigidFlex requires either an Xpedition Layout 301 license or an Xpedition Advanced Technologies license.

Display Control Dialog Box - Objects Tab Controls

To access: Document.ActiveViewEx.DisplayControl.***property.control_id***, where ***property*** is either either Option, Visible, Selectable, StringOption, or Global.color, depending on the control (see Display Control Automation for more information).

Use automation to set options on this tab to control the display of design objects.

Note:
Tables in this section do not include color controls. See Usage Notes for more information.

Objects

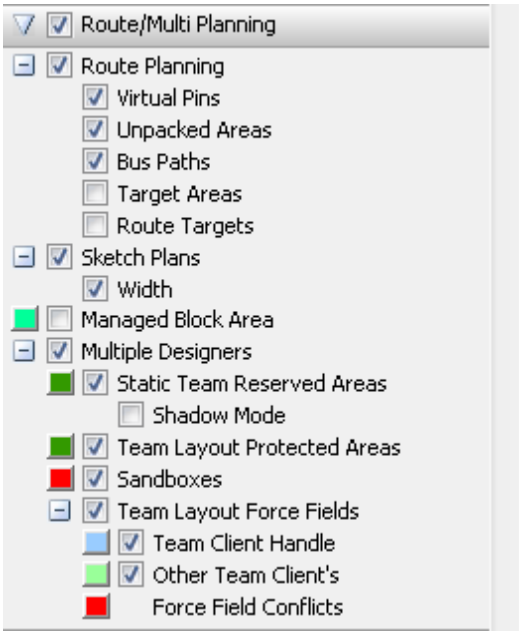


Table 5. Route/Multi Planning Pane

Object	Description	Control ID
Route/Multi Planning	Check box	Option.Planning.Enabled
Route Planning	Check box	Option.RoutePlanning
Virtual Pins	Check box	Option.VirtualPins.Enabled
Unpacked Areas	Check box	Option.UnpackedAreas.Enabled
Bus Paths	Check box	Option.BusPaths.Enabled
Target Areas	Check box	Option.TargetAreas.Enabled
Route Targets	Check box	Option.RouteTargets.Enabled

Table 5. Route/Multi Planning Pane (continued)

Object	Description	Control ID
Sketch Plans	Check box	Option.SketchPlans.Enabled
Width	Check box	Option.SketchPlans.Width
Managed Block Area	Check box	Option.ReuseArea.Enabled
Multiple Designers	Check box	Option.MultipleDesigners
Static Team Reserved Areas	Check box	General.MultipleDesigners.TeamPCB.ReservedAreas
Shadow Mode	Check box	Option.MultipleDesigners.TeamPCB.ShadowMode
Team Layout Protected Areas	Check box	General.MultipleDesigners.Xtreme.ProtectedAreas
Sandboxes	Check box	Board.Sandbox

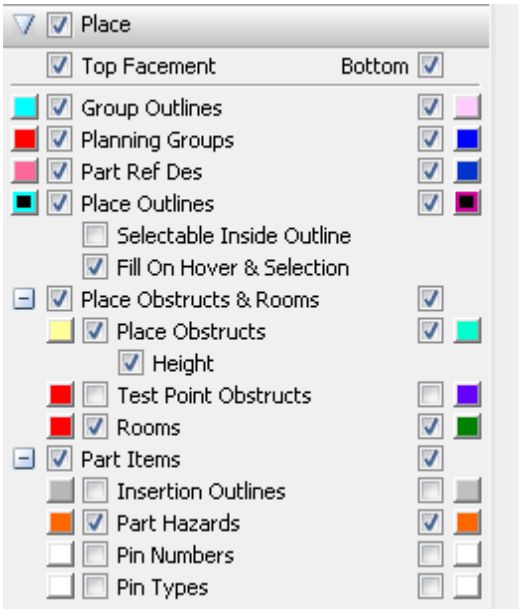


Table 6. Place Pane

Object	Description	Control ID
Place	Check box	Option.PlaceObjects
Top Facement	Check box	Option.PlaceObjects.Parts.Top
Bottom	Check box	Option.PlaceObjects.Parts.Bottom
Group Outlines (Top)	Check box	Group.Outline.Top
Group Outlines (Bottom)	Check box	Group.Outline.Bottom

Table 6. Place Pane (continued)

Object	Description	Control ID
Planning Groups (Top)	Check box	Group.Outline.Bubble.Top
Planning Groups (Bottom)	Check box	Group.Outline.Bubble.Bottom
Part Ref Des (Top)	Check box	Place.Part.Text.RefDes.Top
Part Ref Des (Bottom)	Check box	Place.Part.Text.RefDes.Bottom
Place Outlines (Top)	Check box	Part.PlaceOutline.Top
Place Outlines (Bottom)	Check box	Part.PlaceOutline.Bottom
Selectable Inside Outline	Check box	Option.SelectableInsidePartOutline
Fill On Hover & Selection	Check box	Option.FillPartOutlineOnSelection
Place Obstructs & Rooms (Top)	Check box	Option.PlaceObjects.ObstructsAndRooms.Top
Place Obstructs & Rooms (Bottom)	Check box	Option.PlaceObjects.ObstructsAndRooms.Bottom
Place Obstructs (Top)	Check box	Board.Obstruct.Part.Top
Place Obstructs (Bottom)	Check box	Board.Obstruct.Part.Bottom
Height	Check box	Option.PlaceObstructsHeight
Test Point Obstructs (Top)	Check box	Board.Obstruct.TestPoint.Top
Test Point Obstructs (Bottom)	Check box	Board.Obstruct.TestPoint.Bottom
Rooms (Top)	Check box	Board.Room.Top
Rooms (Bottom)	Check box	Board.Room.Bottom
Part Items (Top)	Check box	Option.PlaceObjects.PartItems.Top
Part Items (Bottom)	Check box	Option.PlaceObjects.PartItems.Bottom
Insertion Outlines (Top)	Check box	Part.InsertionOutline.Top
Insertion Outlines (Bottom)	Check box	Part.InsertionOutline.Bottom
Part Hazards (Top)	Check box	Part.Hazard.Top
Part Hazards (Bottom)	Check box	Part.Hazard.Bottom
Pin Numbers (Top)	Check box	Option.Pin.Number.Top
Pin Numbers (Bottom)	Check box	Option.Pin.Number.Bottom
Pin Types (Top)	Check box	Option.Pin.Type.Top

Table 6. Place Pane (continued)

Object	Description	Control ID
Pin Types (Bottom)	Check box	Option.Pin.Type.Bottom

Table 7. Vias Pane

Object	Description	Control ID
Vias	Check box	Option.Vias.Enabled
All Vias Same Color	Check box	Option.Pad.Via.AllSameColor
Via Pads	Check box	Option.ViaPads.Enabled
Via Holes	Check box	Fabrication.Hole.Via
Via Span Numbers	Check box	General.Via.SpanNumbers
Internal Skip Via Pads	Button	General.Via.InternalSkipViaPad
Inactive Blind/Buried Vias	Check box	General.Via.InactiveBlindBuriedPad

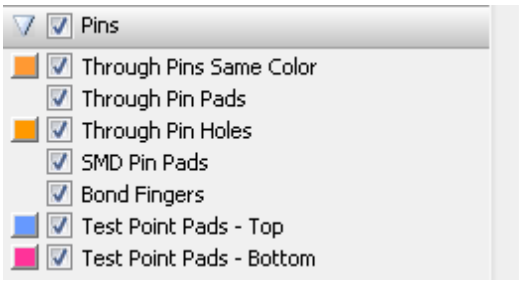


Table 8. Pins Pane

Object	Description	Control ID
Pins	Check box	Option.Pins.Enabled
Through Pins Same Color	Check box	Option.Pad.Through.AllSameColor
Through Pin Pads	Check box	Option.Pin.Through.Enabled
Through Pin Holes	Check box	Fabrication.Hole.Pin

Table 8. Pins Pane (continued)

Object	Description	Control ID
SMD Pin Pads	Check box	Option.SMDPinPads.Enabled
Bond Fingers	Check box	Option.BondPads.Enabled
Test Point Pads - Top	Check box	Copper.Pad.TestPoint.Top
Test Point Pads - Bottom	Check box	Copper.Pad.TestPoint.Bottom

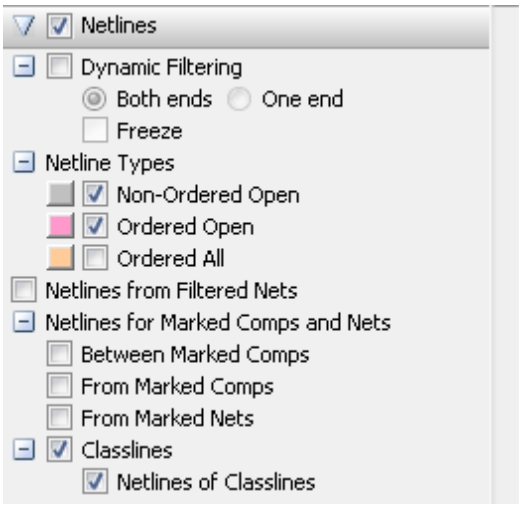


Table 9. Netlines Pane

Object	Description	Control ID
Netlines	Check box	Option.Netlines.Enabled
Dynamic Filtering	Check box	Option.Netlines.DynamicFiltering
Both ends One end	Radio buttons	Option.Netlines.DynamicFiltering.Mode
Freeze	Check box	Option.Netlines.DynamicFiltering.Freeze
Non-Ordered Open	Check box	Netline.NonOrderedOpen
Ordered Open	Check box	Netline.OrderedOpen
Ordered All	Check box	Netline.OrderedAll
Netlines from Filtered Nets	Check box	Option.Netlines.DisplayFromFilteredNets
Between Marked Comps	Check box	Option.Netlines.DisplayBetweenMarkedComponents
From Marked Comps	Check box	Option.Netlines.DisplayFromMarkedComponents
From Marked Nets	Check box	Option.Netlines.DisplayFromMarkedNets

Table 9. Netlines Pane (continued)

Object	Description	Control ID
Classlines	Check box	Option.Classlines
Netlines of Classlines	Check box	Option.Classlines.Netlines

Table 10. Planes Pane

Object	Description	Control ID
Planes	Check box	Option.Planes.Enabled
Data	Check box	Option.Planes.Data.Enabled
Fill / Hatch	Check box	Option.Planes.Data.Fill
Shapes	Check box	Option.Planes.Shape.Enabled
Sketches	Check box	Option.Planes.Sketch.Enabled

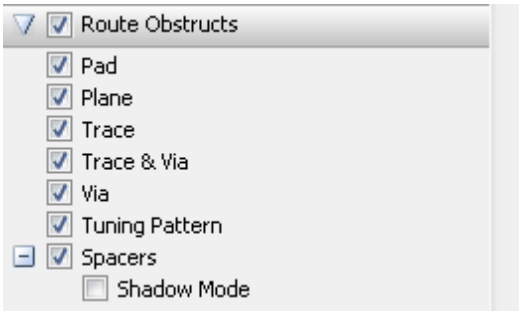


Table 11. Route Obstructs Pane

Object	Description	Control ID
Route Obstructs	Check box	Option.RouteObstructs.Enabled
Pad	Check box	Option.RouteObstructs.Pad.Enabled
Plane	Check box	Option.RouteObstructs.Plane.Enabled
Trace	Check box	Option.RouteObstructs.Trace.Enabled
Trace & Via	Check box	Option.RouteObstructs.TraceVia.Enabled
Via	Check box	Option.RouteObstructs.Via.Enabled

Table 11. Route Obstructs Pane (continued)

Object	Description	Control ID
Tuning Pattern	Check box	Option.RouteObstructs.TuningPattern.Enabled
Spacers	Check box	Option.Spacers.Enabled
Shadow Mode	Check box	Option.Spacers.ShadowModeEnabled

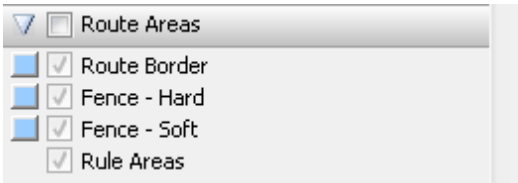


Table 12. Route Areas Pane

Object	Description	Control ID
Route Areas	Check box	Option.RouteAreas.Enabled
Route Border	Check box	Board.RouteBorder
Fence - Hard	Check box	Board.RouteFence.Hard
Fence - Soft	Check box	Board.RouteFence.Soft
Rule Areas	Check box	Option.RuleAreas.Enabled

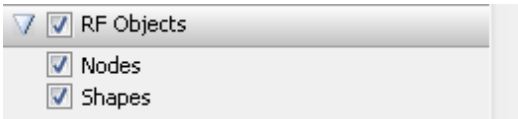


Table 13. RF Objects Pane

Object	Description	Control ID
RF Objects	Check box	Option.RFObjects.Enabled
Nodes	Check box	Option.RFNodes.Enabled
Shapes	Check box	Option.RFShapes.Enabled

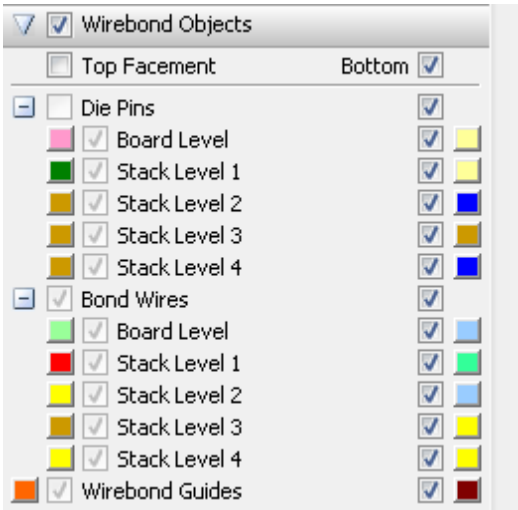


Table 14. Wirebond Objects Pane

Object	Description	Control ID
Wirebond Objects	Check box	Option.WirebondItems.Enabled
Top Facement	Check box	Option.WirebondItemsTop.Enabled
Bottom	Check box	Option.WirebondItemsBottom.Enabled
Die Pins (Top)	Check box	Option.DiePinsTop.Enabled
Die Pins (Bottom)	Check box	Option.DiePinsBottom.Enabled
Board Level (Top)	Check box	Fabrication.DiePins.Level0.Top
Board Level (Bottom)	Check box	Fabrication.DiePins.Level0.Bottom
Stack Level 1 (Top)	Check box	Fabrication.DiePins.Level1.Top
Stack Level 1 (Bottom)	Check box	Fabrication.DiePins.Level1.Bottom
Stack Level 2 (Top)	Check box	Fabrication.DiePins.Level2.Top
Stack Level 2 (Bottom)	Check box	Fabrication.DiePins.Level2.Bottom
Stack Level 3 (Top)	Check box	Fabrication.DiePins.Level3.Top
Stack Level 3 (Bottom)	Check box	Fabrication.DiePins.Level3.Bottom
Stack Level 4 (Top)	Check box	Fabrication.DiePins.Level4.Top
Stack Level 4 (Bottom)	Check box	Fabrication.DiePins.Level4.Bottom
Bond Wires (Top)	Check box	Option.BondWiresTop.Enabled
Bond Wires (Bottom)	Check box	Option.BondWiresBottom.Enabled

Table 14. Wirebond Objects Pane (continued)

Object	Description	Control ID
Board Level (Top)	Check box	Fabrication.BondWires.Level0.Top
Board Level (Bottom)	Check box	Fabrication.BondWires.Level0.Bottom
Stack Level 1 (Top)	Check box	Fabrication.BondWires.Level1.Top
Stack Level 1 (Bottom)	Check box	Fabrication.BondWires.Level1.Bottom
Stack Level 2 (Top)	Check box	Fabrication.BondWires.Level2.Top
Stack Level 2 (Bottom)	Check box	Fabrication.BondWires.Level2.Bottom
Stack Level 3 (Top)	Check box	Fabrication.BondWires.Level3.Top
Stack Level 3 (Bottom)	Check box	Fabrication.BondWires.Level3.Bottom
Stack Level 4 (Top)	Check box	Fabrication.BondWires.Level4.Top
Stack Level 4 (Bottom)	Check box	Fabrication.BondWires.Level4.Bottom
Wirebond Guides (Top)	Check box	WirebondObjects.WirebondGuides.Top
Wirebond Guides (Bottom)	Check box	WirebondObjects.WirebondGuides.Bottom

Usage Notes

Most color settings in Display Control use the following pattern:

```
Global.Color(<control_id>)
```

For example:

```
doc.ActiveViewEx.DisplayControl.Global.Color("LayerControl.1") =
  application.Utility.NewColorPattern(0, 102, 255, 100, 0, False, True)
```

Although they typically match, the control IDs of a color setting and its associated check box could be different. To verify a control ID, enable Display Control Automation Log Mode to echo the control ID to the Message Window. See Setting Layer Visibility in Display Control for more information.

Display Control Dialog Box - Graphic Tab Controls

To access: Document.ActiveViewEx.DisplayControl.***property.control_id***, where ***property*** is either either Option, Visible, Selectable, StringOption, or Global.color, depending on the control (see Display Control Automation for more information).

Use automation to set options on this tab to control the display of graphic objects.

Note:
Tables in this section do not include color controls. See Usage Notes for more information.

Objects

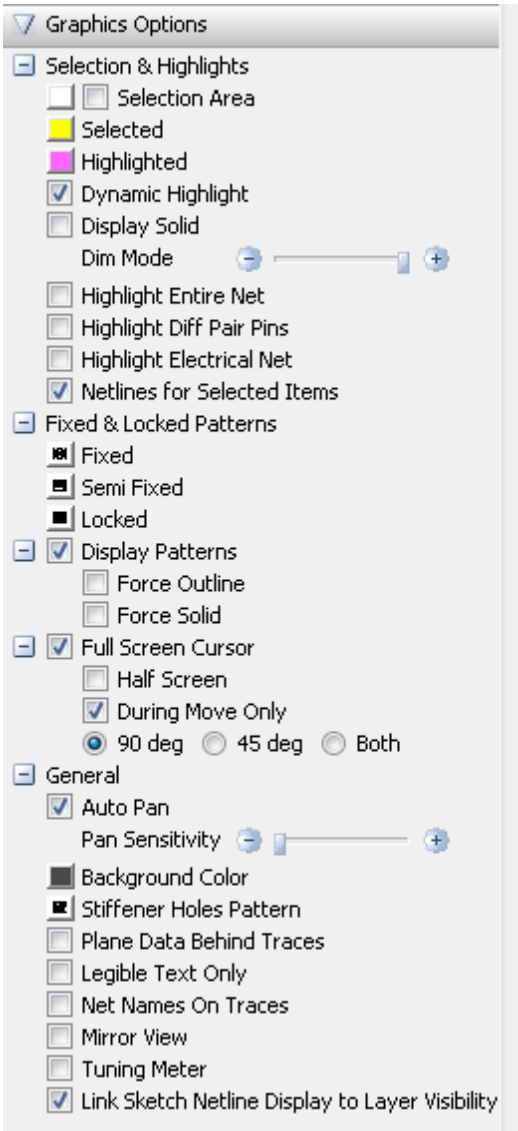


Table 15. Graphics Options Pane

Object	Control Type	Control ID
Selection Area	Check box	General.Color.SelectionShape
Selected	Button	General.Color.Selection
Highlighted	Button	General.Color.Highlight
Dynamic Highlight	Check box	Global.Option.Selection.DynamicHighlight
Display Solid	Check box	Global.Option.Selection.DisplaySolid
Dim Mode	Slider	Global.Option.DimMode
Highlight Entire Net	Check box	Option.SelectionAndHighlights.EntireNetOnSelect
Highlight Diff Pair Pins	Check box	Option.SelectionAndHighlights.DiffPairPinsOnSelect
Highlight Electrical Net	Check box	Option.SelectionAndHighlights.ElectricalNetOnSelect
Netlines for Selected Items	Check box	Option.NetlinesForSelectedItems.Enabled
Fixed	Button	General.Pattern.Fixed
Semi Fixed	Button	General.Pattern.SemiFixed
Locked	Button	General.Pattern.Locked
Display Patterns	Check box	Option.FillPatterns
Force Outline	Check box	Option.ForceOutline
Force Solid	Check box	Option.ForceSolid
Full Screen Cursor	Check box	Global.Option.FullScreenCursor
Half Screen	Check box	Global.Option.HalfScreenCursor
During Move only	Check box	Global.Option.FullScreenCursorDuringMoveOnly
90 deg 45 deg Both	Radio buttons	Global.Option.FullScreenCursorStyle
Auto Pan	Check box	Global.Option.AutoPan
Pan Sensitivity	Slider	Global.Option.PanSensitivity
Background Color	Button	General.Color.Background
Stiffener Holes Pattern	Button	General.Pattern.StiffenerHole
Plane Data Behind Traces	Check box	Global.Option.PlaneDataBehindTraces
Legible Text only	Check box	Option.LegibleTextOnly

Table 15. Graphics Options Pane (continued)

Object	Control Type	Control ID
Net Names On traces	Check box	Global.Option.NetNamesOnTraces
Mirror View	Check box	Option.MirrorView
Tuning Meter	Check box	Global.Option.TuningMeter
Link Sketch Netline Display to Layer Visibility	Check box	Global.Option.LinkSketchNetlineDisplayToLayerVisibility

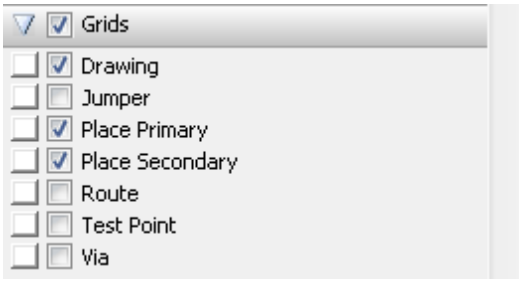


Table 16. Grids Pane

Object	Control Type	Control ID
Grids	Check box	Option.Grids.Enabled
Drawing	Check box	Grid.Draw
Jumper	Check box	Grid.Jumper
Place Primary	Check box	Grid.Part.Primary
Place Secondary	Check box	Grid.Part.Secondary
Route	Check box	Grid.Route
Test Point	Check box	Grid.TestPoint
Via	Check box	Grid.Via

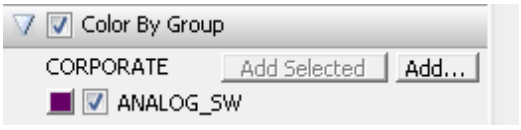


Table 17. Color by Group Pane

Object	Control Type	Control ID
Color By Group	Check box	Global.Option.ColorByGroup.Enabled
<*Group_name*>	Check box	Group.Outline.ColorOverride.<*number*>

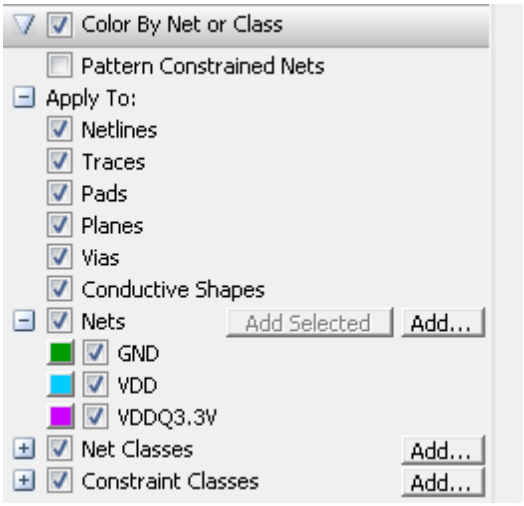


Table 18. Color by Net or Class Pane

Object	Control Type	Control ID
Color By Net or Class	Check box	Global.Option.ColorByNetClass.Enabled
Pattern Constrained Nets	Check box	Global.Option.ColorByNetClass.PatternConstrainedNets
Netlines	Check box	Global.Option.ColorByNetClass.Netlines
Traces	Check box	Global.Option.ColorByNetClass.Traces
Pads	Check box	Global.Option.ColorByNetClass.Pads
Planes	Check box	Global.Option.ColorByNetClass.Planes
Vias	Check box	Global.Option.ColorByNetClass.Vias
Conductive Shapes	Check box	Global.Option.ColorByNetClass.ConductiveShapes
Nets	Check box	Global.Option.ColorByNetClass.Nets.Enabled

<*Net_name*>	Check box	Visible("[Net].<*net_name*>")
		Examples:
		`doc.ActiveViewEx.DisplayControl.Visible( "[Net].VDD" ) = epcbGraphicsItemStateOnEnabled`
		`doc.ActiveViewEx.DisplayControl.Visible( "[Net].$1N4_IO_Port1" ) = epcbGraphicsItemStateOnEnabled`

Net Classes	Check box	Global.Option.ColorByNetClass.NetClasses.Enabled
<*Netclass_name*>	Check box	Visible("[NetClass].<*netclass_name*>")
Constraint Classes	Check box	Global.Option.ColorByNetClass.ConstraintClasses.Enabled

Table 18. Color by Net or Class Pane (continued)

Object	Control Type	Control ID
<*Constraintclass_name*>	Check box	Visible(“[ConstraintClass].<*constraintclass_name*>”)

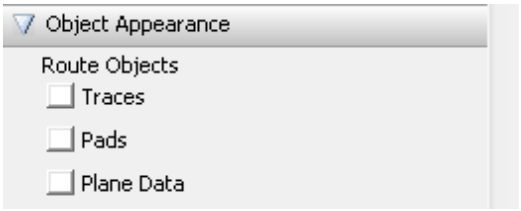


Table 19. Object Appearance Pane

Object	Control Type	Control ID
Traces	Button	Copper.Trace
Pads	Button	Copper.Pad
Plane Data	Button	Copper.Plane.Data

Usage Notes

Most color settings in Display Control use the following pattern:

```
Global.Color(<control_id>)
```

For example:

```
doc.ActiveViewEx.DisplayControl.Global.Color("LayerControl.1") =
  application.Utility.NewColorPattern(0, 102, 255, 100, 0, False, True)
```

Although they typically match, the control IDs of a color setting and its associated check box could be different. To verify a control ID, enable Display Control Automation Log Mode to echo the control ID to the Message Window. See Setting Layer Visibility in Display Control for more information.

Display Control Dialog Box - Fab Tab Controls

To access: Document.ActiveViewEx.DisplayControl.***property.control_id***, where ***property*** is either either Option, Visible, Selectable, StringOption, or Global.color, depending on the control (see Display Control Automation for more information).

Use automation to set options on this tab to control the display of fabrication objects and drafting layers.

Note:
Tables in this section do not include color controls. See Usage Notes for more information.

Objects

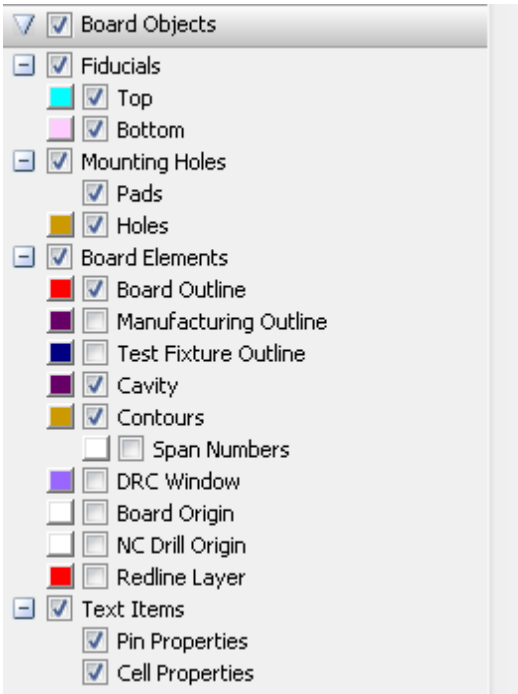


Table 20. Board Objects Pane

Object	Description	Control ID
Board Objects	Check box	Option.BoardObjects.Enabled
Fiducials	Check box	Option.Fiducials.Enabled
Top	Check box	Copper.Pad.Fiducial.Top
Bottom	Check box	Copper.Pad.Fiducial.Bottom
Mounting Holes	Check box	Option.Holes.Enabled
Pads	Check box	Option.MountingHolePads.Enabled

Table 20. Board Objects Pane (continued)

Object	Description	Control ID
Holes	Check box	Fabrication.Hole.Mounting
Board Elements	Check box	Option.BoardElements.Enabled
Board Outline	Check box	Board.BoardOutline
Manufacturing Outline	Check box	Board.ManufacturingOutline
Test Fixture Outline	Check box	Board.FixtureOutline
Bend Areas	Check box	Board.BendArea
Cavity	Check box	Board.Cavity
Contours	Check box	Fabrication.Hole.Contour
Span Numbers	Check box	Fabrication.Hole.Contour.SpanNumbers
DRC Window	Check box	Board.DRCWindow
Board Origin	Check box	Board.Origin.Board
NC Drill Origin	Check box	Board.Origin.NCDrill
Redline Layer	Check box	Fabrication.RedlineLayer
Text Items	Check box	Option.TextItems.Enabled
Pin Properties	Check box	Option.TextItems.PinProperties.Enabled
Cell Properties	Check box	Option.TextItems.CellProperties.Enabled

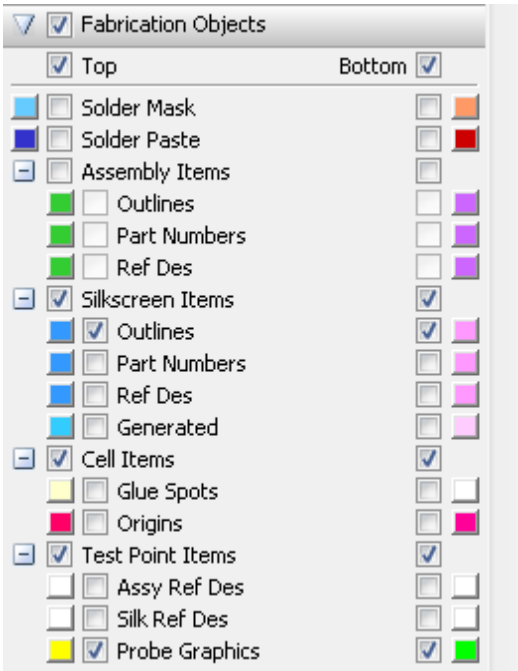


Table 21. Fabrication Objects Pane

Object	Description	Control ID
Fabrication Objects	Check box	Option.FabricationObjects
Top	Check box	Option.FabricationObjects.Top
Bottom	Check box	Option.FabricationObjects.Bottom
Solder Mask (Top)	Check box	Fabrication.Soldermask.Top
Solder Mask (Bottom)	Check box	Fabrication.Soldermask.Bottom
Solder Paste (Top)	Check box	Fabrication.Solderpaste.Top
Solder Paste (Bottom)	Check box	Fabrication.Solderpaste.Bottom
Assembly Items (Top)	Check box	Option.Fabrication.AssemblyItems.Top
Assembly Items (Bottom)	Check box	Option.Fabrication.AssemblyItems.Bottom
Outlines (Top)	Check box	Fabrication.Assembly.Part.Outline.Top
Outlines (Bottom)	Check box	Fabrication.Assembly.Part.Outline.Bottom
Part Numbers (Top)	Check box	Fabrication.Assembly.Part.Text.PartNumber.Top
Part Numbers (Bottom)	Check box	Fabrication.Assembly.Part.Text.PartNumber.Bottom
Ref Des (Top)	Check box	Fabrication.Assembly.Part.Text.RefDes.Top

Table 21. Fabrication Objects Pane (continued)

Object	Description	Control ID
Ref Des (Bottom)	Check box	Fabrication.Assembly.Part.Text.RefDes.Bottom
Silkscreen Items (Top)	Check box	Option.Fabrication.SilkscreenItems.Top
Silkscreen Items (Bottom)	Check box	Option.Fabrication.SilkscreenItems.Bottom
Outlines (Top)	Check box	Fabrication.Silkscreen.Part.Outline.Top
Outlines (Bottom)	Check box	Fabrication.Silkscreen.Part.Outline.Bottom
Part Numbers (Top)	Check box	Fabrication.Silkscreen.Part.Text.PartNumber.Top
Part Numbers (Bottom)	Check box	Fabrication.Silkscreen.Part.Text.PartNumber.Bottom
Ref Des (Top)	Check box	Fabrication.Silkscreen.Part.Text.RefDes.Top
Ref Des (Bottom)	Check box	Fabrication.Silkscreen.Part.Text.RefDes.Bottom
Generated (Top)	Check box	Fabrication.Silkscreen.Generated.Top
Generated (Bottom)	Check box	Fabrication.Silkscreen.Generated.Bottom
Cell Items (Top)	Check box	Option.Fabrication.CellItems.Top
Cell Items (Bottom)	Check box	Option.Fabrication.CellItems.Bottom
Glue Spots (Top)	Check box	Part.Cell.GlueSpot.Top
Glue Spots (Bottom)	Check box	Part.Cell.GlueSpot.Bottom
Origins (Top)	Check box	Part.Cell.Origin.Top
Origins (Bottom)	Check box	Part.Cell.Origin.Bottom
Test Point Items (Top)	Check box	Option.Fabrication.TestPointItems.Top
Test Point Items (Bottom)	Check box	Option.Fabrication.TestPointItems.Bottom
Assy Ref Des (Top)	Check box	Fabrication.Assembly.TestPoint.Text.RefDes.Top
Assy Ref Des (Bottom)	Check box	Fabrication.Assembly.TestPoint.Text.RefDes.Bottom
Silk Ref Des (Top)	Check box	Fabrication.Silkscreen.TestPoint.Text.RefDes.Top
Silk Ref Des (Bottom)	Check box	Fabrication.Silkscreen.TestPoint.Text.RefDes.Bottom
Probe Graphics (Top)	Check box	Fabrication.Silkscreen.TestPoint.Probe.Top
Probe Graphics (Bottom)	Check box	Fabrication.Silkscreen.TestPoint.Probe.Bottom

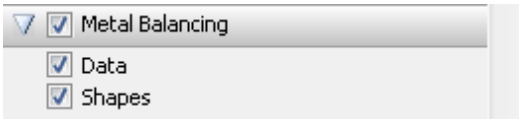


Table 22. Metal Balancing Pane

Object	Description	Control ID
Metal Balancing	Check box	Option.CopperBalancing.Enabled
Data	Check box	Option.CopperBalancing.Data
Shapes	Check box	Option.CopperBalancing.Shapes

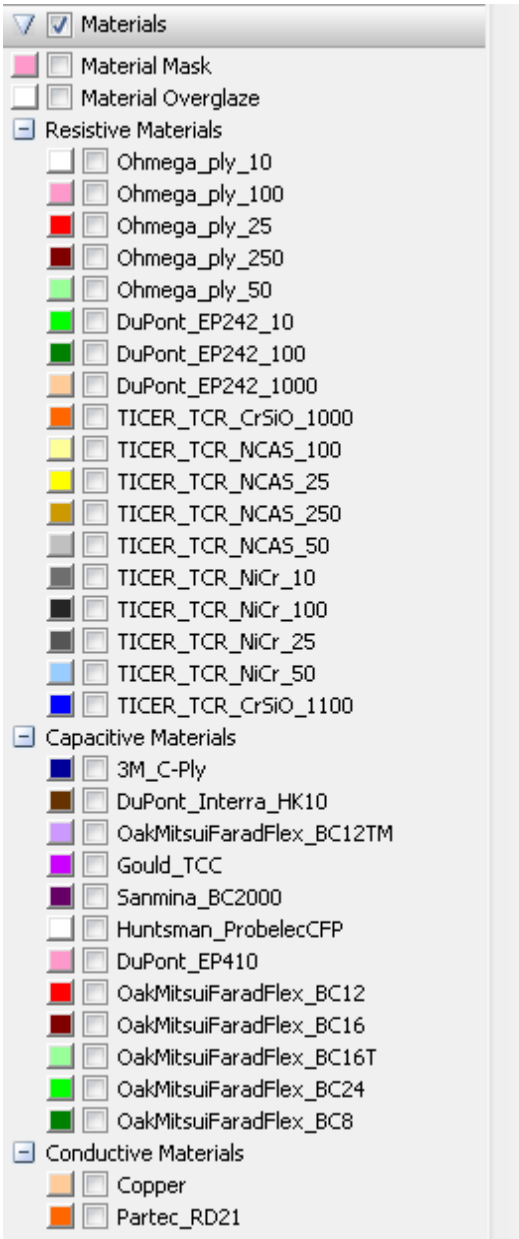


Table 23. Materials Pane

Object	Description	Control ID
Materials	Check box	Option.Materials.Enabled
Material Mask	Check box	Option.Material.Mask.Enabled
Material Overglaze	Check box	Option.Material.Overglaze.Enabled
Ohmega_ply_10	Check box	Option.Material.Ohmega_ply_10
Ohmega_ply_100	Check box	Option.Material.Ohmega_ply_100

Table 23. Materials Pane (continued)

Object	Description	Control ID
Ohmega_ply_25	Check box	Option.Material.Ohmega_ply_25
Ohmega_ply_250	Check box	Option.Material.Ohmega_ply_250
Ohmega_ply_50	Check box	Option.Material.Ohmega_ply_50
DuPont_EP242_10	Check box	Option.Material.DuPont_EP242_10
DuPont_EP242_100	Check box	Option.Material.DuPont_EP242_100
DuPont_EP242_1000	Check box	Option.Material.DuPont_EP242_1000
TICER_TCR_CrSiO_1000	Check box	Option.Material.TICER_TCR_CrSiO_1000
TICER_TCR_NCAS_100	Check box	Option.Material.TICER_TCR_NCAS_100
TICER_TCR_NCAS_25	Check box	Option.Material.TICER_TCR_NCAS_25
TICER_TCR_NCAS_250	Check box	Option.Material.TICER_TCR_NCAS_250
TICER_TCR_NCAS_50	Check box	Option.Material.TICER_TCR_NCAS_50
TICER_TCR_NiCr_10	Check box	Option.Material.TICER_TCR_NiCr_10
TICER_TCR_NiCr_100	Check box	Option.Material.TICER_TCR_NiCr_100
TICER_TCR_NiCr_25	Check box	Option.Material.TICER_TCR_NiCr_25
TICER_TCR_NiCr_50	Check box	Option.Material.TICER_TCR_NiCr_50
TICER_TCR_CrSiO_1100	Check box	Option.Material.TICER_TCR_CrSiO_1100
3M_C-Ply	Check box	Option.Material.3M_C-Ply
DuPont_Interra_HK10	Check box	Option.Material.DuPont_Interra_HK10
OakMitsuiFaradFlex_BC12TM	Check box	Option.Material.OakMitsuiFaradFlex_BC12TM
Gould_TCC	Check box	Option.Material.Gould_TCC
Sanmina_BC2000	Check box	Option.Material.Sanmina_BC2000
Huntsman_ProbelecCFP	Check box	Option.Material.Huntsman_ProbelecCFP
DuPont_EP410	Check box	Option.Material.DuPont_EP410
OakMitsuiFaradFlex_BC12	Check box	Option.Material.OakMitsuiFaradFlex_BC12
OakMitsuiFaradFlex_BC16	Check box	Option.Material.OakMitsuiFaradFlex_BC16
OakMitsuiFaradFlex_BC16T	Check box	Option.Material.OakMitsuiFaradFlex_BC16T

Table 23. Materials Pane (continued)

Object	Description	Control ID
OakMitsuiFaradFlex_BC24	Check box	Option.Material.OakMitsuiFaradFlex_BC24
OakMitsuiFaradFlex_BC8	Check box	Option.Material.OakMitsuiFaradFlex_BC8
Copper	Check box	Option.Material.Copper
Partec_RD21	Check box	Option.Material.Partec_RD21

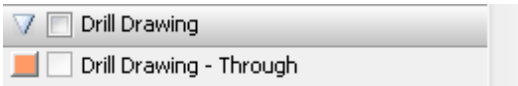


Table 24. Drill Drawing Pane

Object	Description	Control ID
Drill Drawing	Check box	Option.Fabrication.DrillDrawing
Drill Drawing - Through	Check box	Fabrication.DrillDrawingThrough

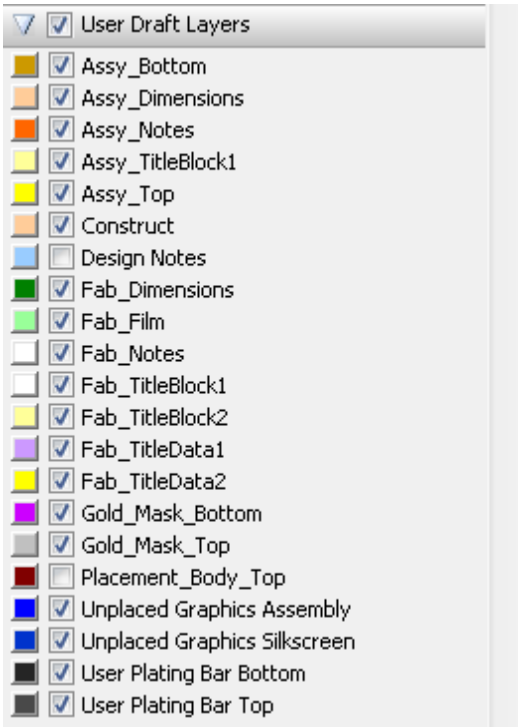


Table 25. User Draft Layers Pane

Object	Description	Control ID
User Draft Layers	Check box	Option.UserDraftLayers.Enabled

Table 25. User Draft Layers Pane (continued)

Object	Description	Control ID
Assy_Bottom	Check box	User.Assy_Bottom
Assy_Dimensions	Check box	User.Assy_Dimensions
Assy_Notes	Check box	User.Assy_Notes
Assy_TitleBlock1	Check box	User.Assy_TitleBlock1
Assy_Top	Check box	User.Assy_Top
Construct	Check box	User.Construct
Design Notes	Check box	User.Design Notes
Fab_Dimensions	Check box	User.Fab_Dimensions
Fab_Film	Check box	User.Fab_Film
Fab_Notes	Check box	User.Fab_Notes
Fab_TitleBlock1	Check box	User.Fab_TitleBlock1
Fab_TitleBlock2	Check box	User.Fab_TitleBlock2
Fab_TitleData1	Check box	User.Fab_TitleData1
Fab_TitleData2	Check box	User.Fab_TitleData2
Gold_Mask_Bottom	Check box	User.Gold_Mask_Bottom
Gold_Mask_Top	Check box	User.Gold_Mask_Top
Placement_Body_Top	Check box	User.Placement_Body_Top
Unplaced Graphics Assembly	Check box	User.Unplaced Graphics Assembly
Unplaced Graphics Silkscreen	Check box	User.Unplaced Graphics Silkscreen
User Plating Bar Bottom	Check box	User.User Plating Bar Bottom
User Plating Bar Top	Check box	User.User Plating Bar Top

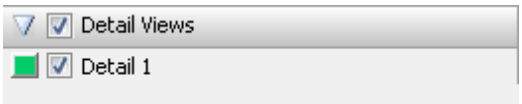


Table 26. Detail Views Pane

Object	Description	Control ID
Detail Views	Check box	Fabrication.DetailViews

Table 26. Detail Views Pane (continued)

Object	Description	Control ID
<*DetailedView_name*>	Check box	Visible(“[DetailedView].<*detailedview_name*>”)

Usage Notes

Most color settings in Display Control use the following pattern:

```
Global.Color(<control_id>)
```

For example:

```
doc.ActiveViewEx.DisplayControl.Global.Color("LayerControl.1") =
  application.Utility.NewColorPattern(0, 102, 255, 100, 0, False, True)
```

Although they typically match, the control IDs of a color setting and its associated check box could be different. To verify a control ID, enable Display Control Automation Log Mode to echo the control ID to the Message Window. See Setting Layer Visibility in Display Control for more information.

Display Control Dialog Box - 3D Tab Controls

To access: Document.ActiveViewEx.DisplayControl.***property.control_id***, where ***property*** is either either Option, Visible, Selectable, StringOption, or Global.color, depending on the control (see Display Control Automation for more information).

Use automation to set options on this tab to control the display of design objects and set 3D display parameters.

Note:
Tables in this section do not include color controls. See Usage Notes for more information.

Objects

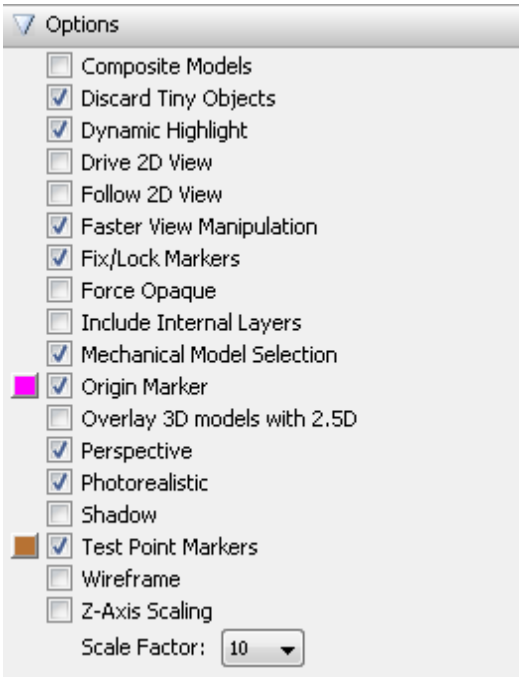


Table 27. Options Pane

Object	Description	Control ID
Composite Models	Check box	[Virtual].3D.Options.CompositeModels
Discard Tiny objects	Check box	[Virtual].3D.Options.DiscardTinyObjects
Dynamic Highlight	Check box	[Virtual].3D.Options.DynamicHighlight
Drive 2D View	Check box	[Virtual].3D.Options.Drive2DView
Follow 2D View	Check box	[Virtual].3D.Options.Follow2DView
Faster View Manipulation	Check box	[Virtual].3D.Options.FasterViewManipulation

Table 27. Options Pane (continued)

Object	Description	Control ID
Fix/Lock Markers	Check box	[Virtual].3D.Options.FixLockMarkers
Force Opaque	Check box	[Virtual].3D.Options.ForceOpaque
Include Internal Layers	Check box	[Virtual].3D.Options.IncludeInternalLayers
Mechanical Model Selection	Check box	[Virtual].3D.Options.AssemblySelection
Origin Marker	Check box	[Virtual].3D.Options.OriginMarker
Overlay 3D models with 2.5D	Check box	[Virtual].3D.Options.Overlay3DModelsWith25D
Perspective	Check box	[Virtual].3D.Options.Perspective
Photorealistic	Check box	[Virtual].3D.Options.Photorealistic
Shadow	Check box	[Virtual].3D.Options.Shadow
Test Point Markers	Check box	[Virtual].3D.Options.TestPointMarkers
Wireframe	Check box	[Virtual].3D.Options.Wireframe
Z-Axis Scaling	Check box	[Virtual].3D.Options.Z-AxisScaling
Scale Factor	Dropdown list	[Virtual].3D.Options.ScaleFactor

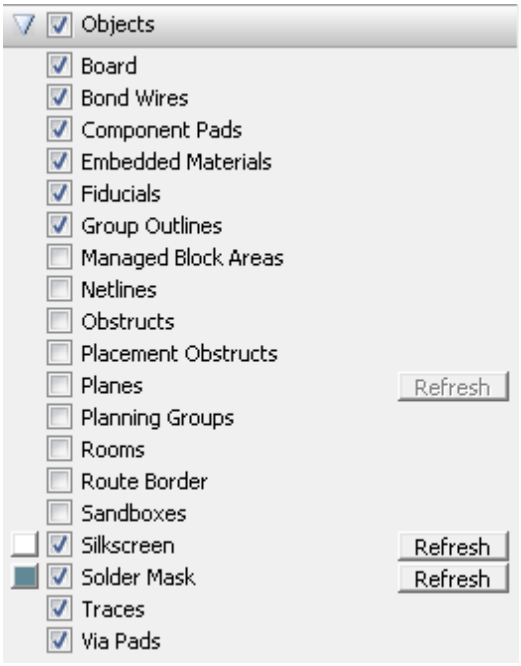


Table 28. Objects Pane

Object	Description	Control ID
Objects	Check box	[Virtual].3D.Objects
Board	Check box	[Virtual].3D.Objects.Board
Bond Wires	Check box	[Virtual].3D.Objects.BondWires
Component Pads	Check box	[Virtual].3D.Objects.ComponentPads
Embedded Materials	Check box	[Virtual].3D.Objects.EmbeddedMaterials
Fiducials	Check box	[Virtual].3D.Objects.Fiducials
Group Outlines	Check box	[Virtual].3D.Objects.GroupOutlines
Managed Block Areas	Check box	[Virtual].3D.Objects.ReuseAreas
Netlines	Check box	[Virtual].3D.Objects.Netlines
Obstructs	Check box	[Virtual].3D.Objects.RouteObstructs
Placement Obstructs	Check box	[Virtual].3D.Objects.PlacementObstructs
Planes	Check box	[Virtual].3D.Objects.Planes
Refresh	Button	[Virtual].3D.Objects.Planes.Refresh
Planning Groups	Check box	[Virtual].3D.Objects.PlanningGroups
Rooms	Check box	[Virtual].3D.Objects.Rooms
Route Border	Check box	[Virtual].3D.Objects.RouteBorder
Sandboxes	Check box	[Virtual].3D.Objects.Sandboxes
Silkscreen	Check box	[Virtual].3D.Objects.Silkscreen
Refresh	Button	[Virtual].3D.Objects.Silkscreen.Refresh
Solder Mask	Check box	[Virtual].3D.Objects.SolderMask
Refresh	Button	[Virtual].3D.Objects.SolderMask.Refresh
Traces	Check box	[Virtual].3D.Objects.Traces
Via Pads	Check box	[Virtual].3D.Objects.ViaPads

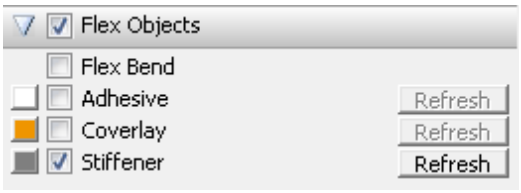


Table 29. Flex Objects Pane

Object	Description	Control ID
Flex Objects	Check box	[Virtual].3D.FlexObjects
Flex Bend	Check box	[Virtual].3D.FlexObjects.FlexBend
Adhesive	Check box	[Virtual].3D.FlexObjects.Adhesive
Refresh	Button	[Virtual].3D.FlexObjects.Adhesive.Refresh
Coverlay	Check box	[Virtual].3D.FlexObjects.Coverlay
Refresh	Button	[Virtual].3D.FlexObjects.Coverlay.Refresh
Stiffener	Check box	[Virtual].3D.FlexObjects.Stiffener
Refresh	Button	[Virtual].3D.FlexObjects.Stiffener.Refresh

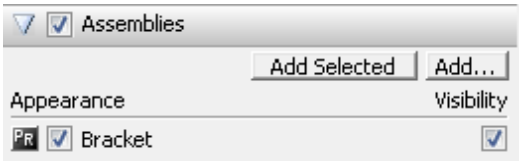


Table 30. Assemblies Pane

Object	Description	Control ID
Assemblies	Check box	[Virtual].3D.Assemblies
Appearance	Check box	[Virtual].3D.Assembly.<*assembly_id*>.Appearance
Visibility	Check box	[Virtual].3D.Assembly.<*assembly_id*>.Visibilty

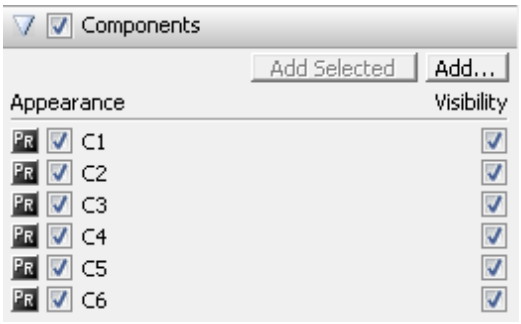


Table 31. Components Pane

Object	Description	Control ID
Components	Check box	Virtual].3D.Components
Appearance	Check box	[Virtual].3D.Component.<*component_id*>.Appearance
Visibility	Check box	[Virtual].3D.Component.<*component_id*>.Visibility

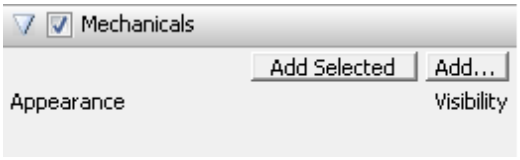


Table 32. Mechanicals Pane

Object	Description	Control ID
Mechanicals	Check box	[Virtual].3D.Mechanicals
Appearance	Check box	[Virtual].3D.Mechanical.<*mechanical_id*>.Appearance
Visibility	Check box	[Virtual].3D.Mechanical.<*mechanical_id*>.Visibility

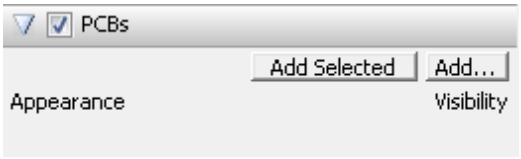


Table 33. PCBs Pane

Object	Description	Control ID
PCBs	Check box	[Virtual].3D.PCBs
Appearance	Check box	[Virtual].3D.PCB.<*pcb_id*>.Appearance
Visibility	Check box	[Virtual].3D.PCB.<*pcb_id*>.Visibility

Usage Notes

Color settings in the 3D tab use the following pattern:

```
StringOption("[Virtual].Global.3D.Options.<control_id>.Color")
```

The StringOption property requires a hexadecimal value to set color in the 3D tab. For example:

```
doc.ActiveViewEx.DisplayControl.StringOption(
"[Virtual].Global.3D.Options.OriginMarker.Color") = "9900CC"
```

Although they typically match, the control IDs of a color setting and its associated check box could be different. To verify a control ID, enable Display Control Automation Log Mode to echo the control ID to the Message Window. See Setting Layer Visibility in Display Control for more information.

Setting Layer Visibility in Display Control

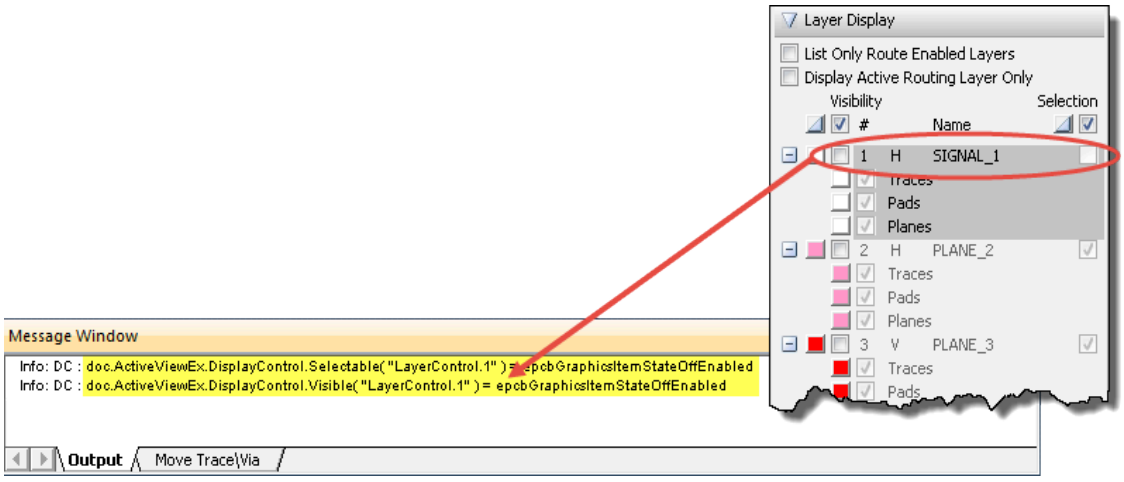
To set layer visibility or selectability in Display Control, you need to know the layer's unique identifier in the layer stackup. There is no built-in function to retrieve the layer ID. You can enable the MGC_ENABLE_DC_AUTOMATION_LOG environment variable to write the Display Control automation calls to the Message Window, where you can copy the calls and paste them into your script.

Procedure

1. Set the environment variable MGC_ENABLE_DC_AUTOMATION_LOG to 1 (on).
2. Launch Xpedition Layout, and then display the Message Window and Display Control.
3. Select one or more layers in the Display Control - Layer Display options.

   The automation calls appear in the Message Window.

Figure 7. Display Control Automation Calls in Message Window



4. Copy the automation calls from the Message Window and paste them into your script.

Example 16. Turn off layer visibility and selectability

```
'Turn off signal layer visibility and selectability.
'
With doc.ActiveViewEx.DisplayControl
   .Selectable( "LayerControl.1" ) =
 epcbGraphicsItemStateOffEnabled
   .Visible( "LayerControl.1" ) = epcbGraphicsItemStateOffEnabled
End With
```

Note:
Reset the MGC_ENABLE_DC_AUTOMATION_LOG environment variable before running the script in another design (possibly having a different generated unique id).

Chapter 2
PCB Applications

This section describes PCB Application objects and the associated methods, properties and events you can access in the MGCPCB interface. Application is the top-level object in all automation scripts.

For information on PCB design data objects, refer to PCB Data Objects A—C, PCB Data Objects D—G, PCB Data Objects H—P, and PCB Data Objects R—Z.

Each Application object contains identical methods, properties, and events (see the following table).

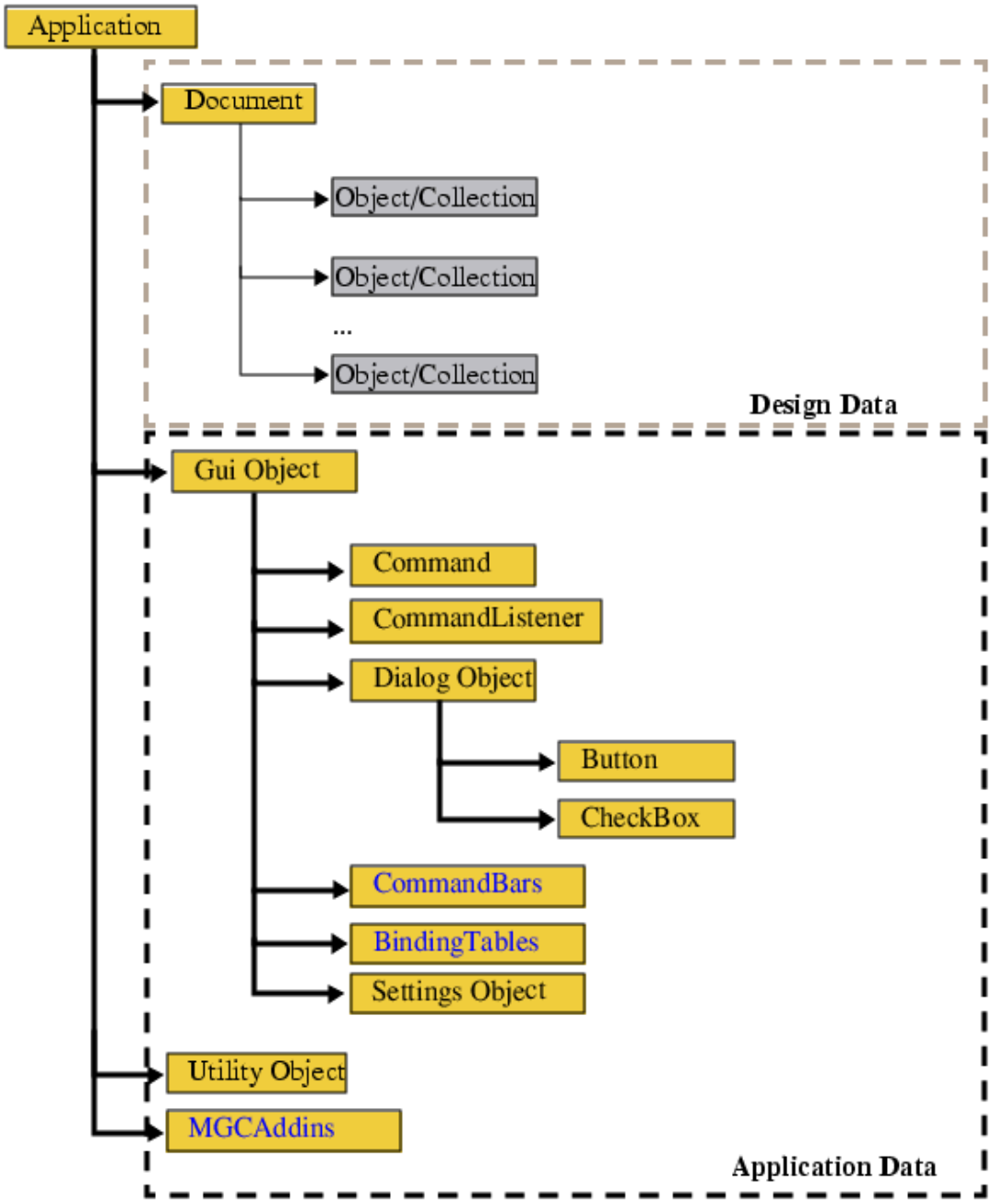
Note:
There are no events for the CellEditorApplication Object.

Table 34. Application Objects

Method, Property, or Event	Description
Application Object	The Generic AutoActive Application.
BoardStationREApplication Object	The BoardStationRE Application.
BoardStationXEApplication Object	The BoardStationXE Application.
CellEditorApplication Object	The CellEditor Application.
DrawingEditorApplication Object	The FabLink Drawing Editor Application.
ExpeditionPCBApplication Object	The Layout Application.
FablinkXEApplication Object	The FabLink Application.
PlannerPCBApplication Object	The Layout Planner Application.
SFXREApplication Object	The SFXRE Application.
SolidEdgePCBApplication Object	The SolidEdgePCB Application
ViewerPCBApplication Object	The Layout Viewer Application.

Figure 8: Application and Design Data shows the top level PCB Application and Document structure. This section describes the Application objects, including Application, Gui and Utility. Later sections describe the Document object.

Figure 8. Application and Design Data



Application Object Methods, Properties, and Events

All objects methods, properties, and events shared between applications.

Table 35. Application Object Methods, Properties, and Events

Method, Property, or Event	Description
AcquireLicenseEx Method	Attempts to acquire the specified license features.
IsLicenseAcquired Method	Checks if the specified license is already acquired in this session.
IsLicenseAvailable Method	Checks if the specified license is available on the license server.
IsValid Method (Applications)	Checks if the object is still valid in the design.
IsXtremeClient Method	Checks if the application is a Team client.
IsXtremeServer Method	Checks if the application is a Team server.
LockServer Method	Locks the server to improve performance.
OpenDocument Method	Opens (and returns) a new document object.
OpenReference Method	Returns the Application object of the reference application.
ProcessScript Method	Runs a script file.
Quit Method	Closes all open documents and exits the application.
ReleaseLicense Method	Releases the specified license feature.
RemoveAddinVersionInfo Method	Removes add-in version information. The About dialog displays the version information.
SetAddinVersionInfo Method	Sets the add-in version information. The About dialog displays the version information.
UnlockServer Method	Unlocks the server that has been previously locked by a LockServer Method.
ActiveDocument Property	Returns the active document object for the application. The document object must be validated before you use it in subsequent operations. Refer to “Using Automation Licenses and Document Validation” on page 1 for more information.
Addins Property	Returns the collection of Addins.
DefaultFilePath Property	Returns the path the application uses to locate script files.
EditorType Property	Returns the editor type of the application. This can be a Layout Template, a Reusable Block, or the standard version of the editor.
FullName Property (Applications)	Returns the full name (including path) of the application.
Gui Property	Returns the Gui object for the application.
InstanceGuid Property	Returns a GUID to uniquely identify the application instance.
IsDBLocked Property	Returns True when an internal command is updating the database or when a transaction is in progress.
IsDocumentReady Property	Returns true if the application has a document ready for automation.
IsOpenGLOn Property	Returns the status of OpenGL.
IsReferenceApplication Property	Returns true if the application is a reference application.

Table 35. Application Object Methods, Properties, and Events (continued)

Method, Property, or Event	Description
Name Property	Returns the name of the application.
ProductBinDirectory Property	Returns the path to the product bin directory.
ProductConfigDirectory Property	Returns the path to the product config directory.
ProductHelpDirectory Property	Returns the path to the product help directory.
ProductStandardDirectory Property	Returns the path to the product standard directory.
RecentFiles Property	Returns an array of recent files.
Type Property (Application)	Returns the application type.
UserHandle Property	Return user handle for Team Layout or a Sandbox.
Utility Property	Returns the utility object for the application.
Version Property	Returns version of the application.
Visible Property	Sets or returns the Application's visible property.
OnIdle Event	Fired when idle processing is allowed. This event fires frequently.
OnOpenDocument Event	Fired after a new document opens in the application.
OnPreLaunchKeyinDlg Event	Fired before launching the keyin dialog when there is no active document. This event is fired for each keystroke unless the keyin dialog has focus..
OnProcessKeyin Event	Fired when keyin data is processed. The client gets the first chance to process it.
Quit Event	Fired before the application exits.

AcquireLicenseEx Method

Prerequisites: None

Attempts to acquire the specified license features.

Usage

```
Application.AcquireLicenseEx(ByVal eType As EPcbLicenseFeature) As Boolean
```

Arguments

- eType

  The license feature (EPcbLicenseFeature) to acquire.

Return Values

Boolean. If True, the requested feature could be acquired. If False, the requested feature could not be acquired.

IsLicenseAcquired Method

Prerequisites: None

Checks if the specified license is already acquired in this session.

Usage

Application.IsLicenseAquired(ByVal eType As EPcbLicenseFeature) As Boolean

Arguments

- eType

  The license feature (EPcbLicenseFeature Enum) to check.

Return Values

Boolean. If True, the license is already acquired. If False, the license is not already acquired.

IsLicenseAvailable Method

Prerequisites: None

Checks if the specified license is available on the license server.

Usage

```
Application.IsLicenseAvailable(ByVal eType As EPcbLicenseFeature) As Boolean
```

Arguments

- eType

  The license feature (EPcbLicenseFeature Enum) to check.

Return Values

Boolean. If True, the license is available. If False, the license is not available.

IsValid Method (Applications)

Prerequisites: None

Checks if the object is still valid in the design.

Usage

Application.IsValid() As Boolean

Arguments

None

Return Values

Boolean. Returns True if the object is still valid. If False is returned, any attempt to access the object will result in a runtime error.

Examples

The following checks if an object is valid. If not a message is output and the subroutine is exited.

```
If Not obj.IsValid Then
 app.Gui.StatusBarText(“The object is invalid”)
 Exit Sub
End If
```

IsXtremeClient Method

Scope: Xpedition Enterprise

Prerequisites: None

Checks if the application is a Team client.

Usage

Application.IsXtremeClient() As Boolean

Arguments

None

Return Values

Boolean. Returns True if this is a Team client. Returns False if this is not a Team client.

Examples

The following checks and outputs whether an application is a Team client or server.

```
s = s & "Team Client: " & app.IsXtremeClient & vbCrLf
s = s & "Team Server: " & app.IsXtremeServer & vbCrLf
```

IsXtremeServer Method

Scope: Xpedition Enterprise

Prerequisites: None

Checks if the application is a Team server.

Usage

Application.IsXtremeServer() As Boolean

Arguments

None

Return Values

Boolean. If True, this is a Team server. If False, this is not a Team server.

Examples

The following checks and outputs whether an application is a Team client or server.

```
s = s & "Team Client: " & app.IsXtremeClient & vbCrLf
s = s & "Team Server: " & app.IsXtremeServer & vbCrLf
```

LockServer Method

Prerequisites: None

Locks the server to improve performance.

Usage

Application.LockServer([ByVal bReservedForFutureUse As Boolean = False]) As Boolean

Arguments

- bReservedForFutureUse

  (Optional) Reserved for future use.

Return Values

Boolean. Returns True if the server is locked. Returns False if the server locking failed.

Examples

The following shows the typical usage for server locking.

```
If (App.LockServer = True) Then
 ...
 <perform your server actions here>
 App.UnlockServer
End If
```

OpenDocument Method

Prerequisites: None

Opens (and returns) a new document object.

Usage

Application.OpenDocument(ByVal filename As String) As IMGCPCBDocument

Arguments

- filename

  A string containing the name of the file to open. This must be a file with the *.pcb* extension.

Return Values

IMGCPCBDocument. Returns the Document that is opened. Returns Nothing if the open operation fails.

Description

The document object must be validated before it can be used in subsequent operations. Refer to “Using Automation Licenses and Document Validation” on page 22 for more information.

Examples

The following attempts to open a specified document and exits the subroutine if it fails.

```
Dim design_name
design_name =“C:\designs\simple_des\pcb\mydesign.pcb”
Set docObj = app.OpenDocument(design_name)
If (Err) Then
 Call app.Gui.StatusBarText(“Unable to open document: " + design_name
  Err.Description,epcbStatusFieldError)
 Exit Sub
End If
‘ Validate the document
```

OpenReference Method

Prerequisites: None

Access: Read-Only

Returns the Application object of the reference application.

Usage

Application.OpenReference(ByVal refDesign As String) As IMGCPCBApplication

Arguments

- refDesign

  String containing the full path to and name of the reference design.

Return Values

IMGCPCBApplication. The reference application.

Examples

```
'CreateObject ("MGCPCB.ExpeditionPCBApplication")

Option Explicit     ' This means that all variables must be declared using
' the 'Dim' statement

'--------------------------------------
'Begin Main program
'
' Testing three calls or methods
'
' Application.IsReferenceApplication
'  Application.IsDocumentReady
'  Application.OpenReference
'
'--------------------------------------

' Get the application object
Dim pcbApp
Set pcbApp = GetObject(,"MGCPCB.ExpeditionPCBApplication")

' Get the active document
Dim pcbDoc
Set pcbDoc = pcbApp.ActiveDocument

' Create the output logfile
Dim fso, txtfile
Set fso = CreateObject("Scripting.FileSystemObject")
Set txtfile = fso.OpenTextFile("LogFiles\AutoOpenRefFile.txt",8,True)
```

```
' Test current settings before Reference file is Opened
txtfile.Write "Check current settings before reference file is opened" & _
 vbCrLf
txtfile.Write "" & vbcrlf
txtfile.Write "Testing Application.IsReferenceApplication before " & _
 "Reference is Open: " & "( " & pcbApp.IsReferenceApplication & " )" & _
 vbCrLf
txtfile.Write "Testing Application.IsDocumentReady for the current " & _
 "Document: " & "( " & pcbApp.IsDocumentReady & " )" & vbCrLf
txtfile.Write "And the design name is " & "( " & pcbDoc.Name & " )" & _
 vbCrLf
txtfile.Write "" & vbcrlf
' License the document
If (ValidateServer(pcbDoc) = 1) Then

 ' add a reference to the MGCPCB type library in order to use enums
 'Scripting.AddTypeLibrary("MGCPCB.ExpeditionPCBApplication")


 '---------------------------------------
 'add your code here
 '---------------------------------------
'    MsgBox pcbApp.FullName

    Dim app2
 Set app2 = pcbApp.OpenReference("..\Jumpers_Ref\pcb\Reference.pcb")

 ' Check current settings after Reference file is Opened
 txtfile.Write "Check current settings after Reference file is " & _
  "Opened" & vbCrLf
 txtfile.Write " " & vbCrLf
 txtfile.Write "Application.OpenReference is now set to Reference " & _
  "file ( ..\Jumpers_Ref\pcb\Reference.pcb )"  & vbCrLf
 txtfile.write "Application.IsReferenceApplication is now: " & "( " & _
  app2.IsReferenceApplication & " )" & vbCrLf

 If app2 Is Nothing Then
  MsgBox "no ptr returned"
 Else
  If app2.IsReferenceApplication Then
   Do While app2.IsDocumentReady <> True
    Scripting.sleep(500)
   Loop
   Set pcbDoc = app2.ActiveDocument

    txtfile.Write "The Application.IsDocumentReady should now " & _
    "be the reference? " & "( " & app2.IsDocumentReady & " )" & _
     vbCrLf
   txtfile.Write "And the design name is " & "( " & _
    pcbDoc.Name & " )"
   txtfile.Write "" & vbCrLf
   app2.Quit
  End If
 End If
```

```
Else
 Msgbox("Could not validate the server. Exiting program.")
End If

'---------------------------------------
'End Main program
'---------------------------------------

'---------------------------------------
'Add local functions, subroutines,
'and methods here
'---------------------------------------


'---------------------------------------
' Begin Validate Server Function
'---------------------------------------
Private Function ValidateServer(doc)

 Dim key, licenseServer, licenseToken

 ' Ask document for the key
 key = doc.Validate(0)

 ' Get license server
 Set licenseServer =
  CreateObject("MGCPCBAutomationLicensing.Application")

 ' Ask the license server for the license token
 licenseToken = licenseServer.GetToken(key)

 ' Release license server
 Set licenseServer = nothing

 ' Turn off error messages. Validate may fail if the token is incorrect
 On Error Resume Next
 Err.Clear

 ' Ask the document to validate the license token
 doc.Validate(licenseToken)
 If Err Then
  ValidateServer = 0
 Else
  ValidateServer = 1
 End If

End Function
'---------------------------------------
' End Validate Server Function
'---------------------------------------
```

ProcessScript Method

Prerequisites: None

Runs a script file.

Usage

```
Application.ProcessScript(ByVal filename As String , ByVal bSilent As Boolean) As Boolean
```

Arguments

- filename

  The name of the script to process. This must be a script that uses the VBScript or JScript syntax.

- bSilent

  If this argument is set to true, the runtime error messages and warnings will not be echoed.

Return Values

Boolean. Returns True if the script ran successfully. Returns False if the script failed.

Quit Method

Prerequisites: None

Closes all open documents and exits the application.

Usage

Application.Quit()

Arguments

None

Examples

The following subroutine closes the passed document and quits the application.

```
Sub UnloadApplication(doc)
 ' release the app and document
 If Not (doc Is Nothing) Then
  doc.Close (False)
  Set doc = Nothing
 End If
 If Not (app Is Nothing) Then
  app.Quit
  Set app = Nothing
 End If
End Sub
```

ReleaseLicense Method

Prerequisites: None

Releases the specified license feature.

Usage

```
Application.ReleaseLicense(ByVal eType As EPcbLicenseFeature) As Boolean
```

Arguments

- eType

  A license feature type (EPcbLicenseFeature).

Return Values

Boolean. If True, the license feature was successfully released. If False, the license feature release failed.

RemoveAddinVersionInfo Method

Prerequisites: None

Removes add-in version information. The About dialog displays the version information.

Usage

```
Application.RemoveAddinVersionInfo(ByVal AddinName As String)
```

Arguments

- AddinName

  A string that contains the name of the add-in.

SetAddinVersionInfo Method

Prerequisites: None

Sets the add-in version information. The About dialog displays the version information.

Usage

```
Application.SetAddinVersionInfo(ByVal AddinName As String ,
        ByVal VersionInformation As String)
```

Arguments

- AddinName

  A string that contains the name of the add-in.

- VersionInformation

  A string that contains the version information to display in the About dialog.

UnlockServer Method

Prerequisites: None

Unlocks the server that has been previously locked by a LockServer Method.

Usage

Application.UnlockServer([ByVal bReservedForFutureUse As Boolean = True]) As Boolean

Arguments

- bReservedForFutureUse

  (Optional) Reserved for future use.

Return Values

Boolean. Returns True if the server is unlocked. Returns False if the server unlock failed.

Description

Refer to “Using Server Locking” on page 24 for more information.

Examples

The following shows the typical usage for server locking.

```
If (App.LockServer = True) Then
 ...
 <perform your server actions here>
 App.UnlockServer
End If
```

ActiveDocument Property

Prerequisites: None

Access: Read-Only

Returns the active document object for the application. The document object must be validated before you use it in subsequent operations. Refer to “Using Automation Licenses and Document Validation” on page 1 for more information.

Usage

Application.ActiveDocument

Arguments

None

Return Values

IMGCPCBDocument. An object (Document) representing the active document.

Examples

The following gets the active document and validates.

```
Dim key, licenseServer, licenseToken, docObj
```

```
' collect the active document
Set docObj = app.ActiveDocument
If (Err) Then
 Call app.Gui.StatusBarText("No active document: " + _
  Err.Description,epcbStatusFieldError)
 Exit Sub
End If
```

```
' Ask document for the key
key = docObj.Validate(0)
```

```
' Get token from license server
Set licenseServer = CreateObject("MGCPCBAutomationLicensing.Application")
licenseToken = licenseServer.GetToken(key)
Set licenseServer = Nothing
```

```
' Ask the document to validate the license token
Err.Clear
docObj.Validate(licenseToken)
If (Err) Then
```

```
 Call app.Gui.StatusBarText("No active document license: " + _
  Err.Description,epcbStatusFieldError)
 Exit Sub
End If
```

Addins Property

Prerequisites: None

Access: Read-Only

Returns the collection of Addins.

Usage

Application.Addins

Arguments

None

Return Values

Object. Returns an Addins Collection.

DefaultFilePath Property

Prerequisites: None

Access: Read-Only

Returns the path the application uses to locate script files.

Usage

Application.DefaultFilePath

Arguments

None

Return Values

String. A string that contains the path that is used to locate script files.

Examples

The following output the path the application uses to locate script files.

```
s = s & "Script location: " & app.DefaultFilePath & vbCrLf
```

EditorType Property

Prerequisites: None

Access: Read-Only

Returns the editor type of the application. This can be a Layout Template, a Reusable Block, or the standard version of the editor.

Usage

Application.EditorType

Description

Use this property to distinguish between a standard Xpedition Layout, Layout Template, and Reusable Block Editor when attempting to connect to an already running Xpedition Layout application. These editors all appear as Xpedition Layout applications at the process level.

Arguments

None

Return Values

EPcbEditorType. The editor type (EPcbEditorType)

Examples

The following provides the framework for distinguishing the application editor type.

```
Dim edType

Select Case app.EditorType
 Case epcbEditorGeneral
  edType = "Standard "
 Case epcbEditorReusableBlock
  edType = "Reusable block "
 Case epcbEditorTemplate
  edType = "Layout template "
End Select
s = s& "Editor Type: " & edType & vbCrLf
```

FullName Property (Applications)

Prerequisites: None

Access: Read-Only

Returns the full name (including path) of the application.

Usage

Application.FullName

Arguments

None

Return Values

String. A string that contains the full name and path of the application.

Description

Use the Name Property to return the name of the application without its path.

Examples

The following output the full name and location of the application.

```
s = s & "Full Name: " & app.FullName & vbCrLf
```

Gui Property

Prerequisites: None

Access: Read-Only

Returns the Gui object for the application.

Usage

Application.Gui

Arguments

None

Return Values

IMGCPCBGui. The gui object (“Gui” on page 216) for the application.

Examples

The following outputs an error message to the status bar.

```
set GUI = app.Gui
If (Err) Then
 Call GUI.StatusBarText(_
  "An error occurred while collecting objects: " + _
  vbNewLine + Err.Description,epcbStatusFieldError)
 Exit Sub
```

InstanceGuid Property

Prerequisites: None

Access: Read-Only

Returns a GUID to uniquely identify the application instance.

Usage

Application.InstanceGuid

Arguments

None

Return Values

String. A string that contains the GUID for the application instance.

Examples

The following outputs the GUID for the application.

```
s = s & "GUI ID: " & app.InstanceGuid & vbCrLf
```

IsDBLocked Property

Prerequisites: None

Access: Read-Only

Returns True when an internal command is updating the database or when a transaction is in progress.

Usage

```
Application.IsDBLocked
```

Arguments

None

Return Values

Boolean. If True, the database is updating. If False, the database is unlocked.

IsDocumentReady Property

Prerequisites: None

Access: Read-Only

Returns true if the application has a document ready for automation.

Usage

```
Object.IsDocumentReady As Boolean
```

Arguments

None

Return Values

Boolean. Returns True if if the application has a document ready for automation.

IsOpenGLOn Property

Prerequisites: None

Access: Read-Only

Returns the status of OpenGL.

Usage

```
Object.IsOpenGLOn As Boolean
```

Arguments

None

Return Values

Boolean. Returns True if OpenGL is on.

IsReferenceApplication Property

Prerequisites: None

Access: Read-Only

Returns true if the application is a reference application.

Usage

Object.IsReferenceApplication As Boolean

Arguments

None

Return Values

Boolean. Returns True if the application is a reference application.

Name Property

Prerequisites: None

Access: Read-Only

Returns the name of the application.

Note:
Use the “FullName Property” on page 129 to retrieve the name and path of the application.

Usage

Application.Name

Arguments

None

Return Values

String. A string that contains the name of the application, not including the path.

Examples

The following outputs the application name and full name (includes path).

```
s = s & "Application Name: " & app.Name & vbCrLf
s = s & "Full Name: " & app.FullName & vbCrLf
```

ProductBinDirectory Property

Prerequisites: None

Access: Read-Only

Returns the path to the product bin directory.

Usage

Application.ProductBinDirectory

Arguments

None

Return Values

String. Returns the full path to the application *bin* directory.

Examples

The following outputs the full path to the application bin directory.

```
s = s & "Bin Directory: " & app.ProductBinDirectory & vbCrLf
```

ProductConfigDirectory Property

Prerequisites: None

Access: Read-Only

Returns the path to the product config directory.

Usage

Application.ProductConfigDirectory

Arguments

None

Return Values

String. Returns the full path to the application config directory.

Examples

The following outputs the full path to the application config directory.

```
s = s & "Config Directory: " & app.ProductConfigDirectory & vbCrLf
```

ProductHelpDirectory Property

Prerequisites: None

Access: Read-Only

Returns the path to the product help directory.

Usage

Application.ProductHelpDirectory

Arguments

None

Return Values

String. Returns the full path to the application help directory.

Examples

The following outputs the full path to the application help directory.

```
s = s & "Help Directory: " & app.ProductHelpDirectory & vbCrLf
```

ProductStandardDirectory Property

Prerequisites: None

Access: Read-Only

Returns the path to the product standard directory.

Usage

Application.ProductStandardDirectory

Arguments

None

Return Values

String. Returns the full path to the application standard directory.

Examples

The following outputs the full path to the application standard directory.

```
s = s & "Standard Directory: " & app.ProductStandardDirectory & vbCrLf
```

RecentFiles Property

Prerequisites: None

Access: Read-Only

Returns an array of recent files.

Usage

Object.RecentFiles

Arguments

None

Return Values

Array

Type Property (Application)

Prerequisites: None

Access: Read-Only

Returns the application type.

Usage

Application.Type

Arguments

None

Return Values

EPcbApplicationType. The application type (EPcbApplicationType).

The following summarizes the return value of the property for the specified application.

Table 36. Summary of Type Property Return Type Values

Object	Type Value
Application	Depends on application
BoardStationREApplication	epcbAppBoardStationRE
BoardStationXEApplication	epcbAppBoardStationXE
ExpeditionPCBApplication	epcbAppExpedition
FablinkXEApplication	epcbAppFablinkXE
PlannerPCBApplication	epcbAppPlanner
SFXREApplication	epcbAppSFXRE
ViewerPCBApplication	epcbAppViewer

UserHandle Property

Scope: Xpedition Enterprise

Prerequisites: None

Access: Read-Only

Return user handle for Team Layout or a Sandbox.

Usage

Application.UserHandle

Arguments

None

Return Values

String. A string that contains the user handle.

Utility Property

Prerequisites: None

Access: Read-Only

Returns the utility object for the application.

Usage

Application.Utility

Arguments

None

Return Values

IMGCPCBUtility. The Utility Object for the application.

Examples

The following uses the utility object to convert units.

```
' find all things within 50mils around this point
delta = app.Utility.ConvertUnit(500.0, epcbUnitMils, doc.CurrentUnit)
' pick and select all conductor layer gfx and pad objects around the point
Dim obs
Set obs = doc.Pick(x-delta, y-delta, x+delta, y+delta, _
  epcbObjectClassConductorLayerGfx + epcbObjectClassPadstackObject, _
  Nothing, False, True, epcbUnitCurrent)
```

Version Property

Prerequisites: None

Access: Read-Only

Returns version of the application.

Usage

Application.Version

Arguments

None

Return Values

String. A string that contains the version of the application.

Examples

The following outputs the version of the application.

```
s = s & "Version: " & app.Version & vbCrLf
```

Visible Property

Prerequisites: None

Access: Read/Write.

Sets or returns the Application's visible property.

Usage

Application.Visible = True | False

Arguments

None

Return Values

True | False. If True, the application is visible. If False, the application is not visible.

Examples

The following makes the application visible if it is invisible.

```
If Not app.Visible Then
 app.Visible = True
End If
```

OnIdle Event

Prerequisites: None

Fired when idle processing is allowed. This event fires frequently.

Usage

```
Sub Application_OnIdle()
```

Arguments

None

OnOpenDocument Event

Prerequisites: None

Fired after a new document opens in the application.

Usage

Sub Application_OnOpenDocument(ByVal Flags As EPcbOnOpenDocumentFlags)

Arguments

- Flags

  The OnOpenDocument flag types (EPcbOnOpenDocumentFlags).

Description

Reload is True (epcbOnOpenDocReload) if the event is the result of a reload.

For more information on event handlers refer to “Working with Events” on page 38.

Examples

The example below shows an event handler for Xpedition Layout opening or reloading a document.

```
Dim docPCB
Dim doNotExit: doNotExit = True
```

```
' ========================================
' The app is opening or reloading a document.
' Do design initialization or verification if new open.
' Make sure the docPCB object is reassigned if new open.
' ========================================
Sub appEvents_OnOpenDocument(Flags)
 Select Case Flags
  Case epcbOnOpenDocOpen
   Set docObj = app.ActiveDocument
   ' Custom document initialization & verifications goes here
  Case epcbOnOpenDocReload
   ' Skip the custom init & verify steps
 End Select

End Sub
```

```
‘ Get the application object
Dim appPCB
Set appPCB = GetObject(,"MGCPCB.ExpeditionPCBApplication")
```

```
‘ Attach events to the application
Call Scripting.AttachEvents(appPCB, "appEvents")
```

```
' collect document object
Set docPCB = GetLicensedDoc(appPCB)
```

```
' An infinite loop to prevent script from exiting
Do While doNotExit
 Scripting.Sleep(300)
Loop
```

OnPreLaunchKeyinDlg Event

Prerequisites: None

Fired before launching the keyin dialog when there is no active document. This event is fired for each keystroke unless the keyin dialog has focus..

Usage

```
Function Application_OnPreLaunchKeyinDlg() As Boolean
```

Arguments

None

OnProcessKeyin Event

Prerequisites: None

Fired when keyin data is processed. The client gets the first chance to process it.

Usage

```
Function Application_OnProcessKeyin(ByVal keyinText As String) As Boolean
```

Arguments

- keyinText

  A string that contains the keyin text.

Description

Return False to indicate the system should process. Returns True to keep it from being processed.

Quit Event

Prerequisites: None

Fired before the application exits.

Usage

Sub Application_Quit()

Arguments

None

Examples

The follows show the event handler for quitting the Xpedition Layout application.

```
Dim docPCB
Dim doNotExit
doNotExit = True

' ================================
' The application is quitting
' ================================
Sub appEvents_Quit()
 MsgBox "Quitting the Application"
'Exit the infinite loop
 doNotExit = False
' Make sure all objects are released
 Set docPCB = Nothing
 Set appPCB = Nothing
End Sub


‘ Get the application object
Dim appPCB
Set appPCB = GetObject(,"MGCPCB.ExpeditionPCBApplication")

‘ Attach events to the application
Call Scripting.AttachEvents(appPCB, "appEvents")

' collect document object
Set docPCB = GetLicensedDoc(appPCB)

' An infinite loop to prevent script from exiting
Do While doNotExit
 Scripting.Sleep(300)
Loop
```

Note:
For more information on event handlers, refer to “Working with Events” on page 38.

Other Application Objects

This section includes an alphabetical listing of objects that the Application object contains and that are not considered design data.

This includes objects associated with the Gui Object and Utility Object.

Table 37. Other Application Objects

Object	Description
Button Object	The Button object describes a dialog button.
checkbox Object	The CheckBox object describes a dialog check box.
Command Object	The Command object is an event-based object that allows clients to create interactive commands.
CommandListener Object	The CommandListener object is an event-based object that allows clients to Pre and Post-process existing interactive commands.
Dialog Object	The Dialog object describes a dialog.
Gui Object	The Gui object describes the application's Graphical User Interface (GUI).
Settings Object	The Settings object provides a mechanism for storing and retrieving settings.
Utility Object	The Utility object contains utility functions for conversion and object creation.

Button Object

The Button object describes a dialog button.

Table 38. Button Object Methods

Method	Description
Click Method (Button Object)	Clicks the button on the dialog box.
Click2 Method (Button Object)	Clicks the button on the dialog box.

Click Method (Button Object)

Prerequisites: None

Object: Button Object

Clicks the button on the dialog box.

Usage

```
Button.Click()
```

Arguments

None

Click2 Method (Button Object)

Prerequisites: None

Object: Button Object

Clicks the button on the dialog box.

Usage

```
Button.Click2 ([ByVal bWait As Boolean = True])
```

Arguments

- bWait

  (Optional) If True, the action completes before continuing. If False, the action does not wait before continuing. The default is for the action to wait.

checkbox Object

The checkbox object describes a dialog check box.

Table 39. checkbox Object Properties and Methods

Property or Method	Description
Checked Property (checkbox Object)	Sets or returns the state of the checkbox button on the dialog.
IsValid Method (Applications)	Checks if the object is still valid in the design.

Checked Property (checkbox Object)

Prerequisites: None

Object: checkbox Object

Access: Read/Write

Sets or returns the state of the checkbox button on the dialog.

Usage

```
checkbox.Checked = True | False
```

Arguments

None

Return Values

True | False. If True, the check box is checked. If False, the check box is not checked.

IsValid Method (Applications)

Prerequisites: None

Object: checkbox Object

Checks if the object is still valid in the design.

Usage

Application.IsValid() As Boolean

Arguments

None

Return Values

Boolean. Returns True if the object is still valid. If False is returned, any attempt to access the object will result in a runtime error.

Examples

The following checks if an object is valid. If not a message is output and the subroutine is exited.

```
If Not obj.IsValid Then
 app.Gui.StatusBarText(“The object is invalid”)
 Exit Sub
End If
```

Command Object

The Command object is an event-based object that allows clients to create interactive commands.

Table 40. Command Object Methods, Properties, and Events

Method, Property, or Event	Description
IsValid Method (Applications)	Checks if the object is still valid in the design.
Name Property (Command Object)	Returns the command name.
Unit Property (Command Object)	Sets or returns the units coordinates use in the command events.
OnChar Event (Command Object)	Fired when a character is typed on the keyboard.
OnLayerChanged Event (Command Object)	Fired when either the route or component active layer is changed.
OnMouseBeginDrag Event (Command Object)	Fired when a mouse button is held down and the mouse starts moving.
OnMouseCancelDrag Event (Command Object)	Fired when user cancels the drag by returning to the start point and takes their finger off the button.
OnMouseClk Event (Command Object)	Fired when a mouse button is clicked (the up-event).
OnMouseDblClk Event (Command Object)	Fired when a mouse button is double-clicked.
OnMouseDown Event (Command Object)	Fired when a mouse button is pressed down.
OnMouseDrag Event (Command Object)	Fired when the mouse moves while a drag is in progress.
OnMouseEndDrag Event (Command Object)	Fired when user terminates the drag by taking their finger off the mouse button.
OnMouseMove Event (Command Object)	Fired when the mouse is moved.
OnTerminate Event (Command Object)	Fired when the command is exiting.
OnUndo Event (Command Object)	Fired when a request for Undo or Redo is made.
OnVChar Event (Command Object)	Fired when a virtual character is typed on the keyboard.

IsValid Method (Applications)

Prerequisites: None

Object: Command Object

Checks if the object is still valid in the design.

Usage

Application.IsValid() As Boolean

Arguments

None

Return Values

Boolean. Returns True if the object is still valid. If False is returned, any attempt to access the object will result in a runtime error.

Examples

The following checks if an object is valid. If not a message is output and the subroutine is exited.

```
If Not obj.IsValid Then
 app.Gui.StatusBarText(“The object is invalid”)
 Exit Sub
End If
```

Name Property (Command Object)

Prerequisites: None

Object: Command Object

Returns the command name.

Usage

```
Command.Name
```

Arguments

None

Return Values

String. A string that contains the name of the command.

Unit Property (Command Object)

Prerequisites: None

Object: Command Object

Access: Read/Write.

Sets or returns the units coordinates use in the command events.

Usage

```
Command.Unit = EPcbUnit
```

Arguments

None

Return Values

EPcbUnit. The units (EPcbUnit) to use.

Note:
epcbUnitCurrent cannot be used as a value for this property.

Description

The command object does not inherit units from the active document, so you must set the Command units separately.

OnChar Event (Command Object)

Prerequisites: None

Object: Command Object

Fired when a character is typed on the keyboard.

Usage

```
Function Command_OnChar(ByVal nChar As Long) As Boolean
```

Arguments

- nChar

  The key code that has been typed on the keyboard.

Return Values

Boolean. If True, the character was processed inside the event. If False, the character was not processed inside the event.

OnLayerChanged Event (Command Object)

Prerequisites: None

Object: Command Object

Fired when either the route or component active layer is changed.

Usage

```
Sub Command_OnLayerChanged(ByVal nLayer As Long ,
        ByVal bRouteLayerChanged As Boolean)
```

Arguments

- nLayer

  The number of the layer that has become active.

- bRouteLayerChanged

  If True, a route layer was changed. If False, a component layer was changed.

OnMouseBeginDrag Event (Command Object)

Prerequisites: None

Object: Command Object

Fired when a mouse button is held down and the mouse starts moving.

Usage

```
Function Command_OnMouseBeginDrag(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dXStart As Double , ByVal dYStart As Double ,
        ByVal dXNow As Double , ByVal dYNow As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was pressed (EPcbMouseButton)
- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.
- dXStart

  The starting X position of the mouse.
- dYStart

  The starting Y position of the mouse.
- dXNow

  The X position of the mouse.
- dYNow

  The Y position of the mouse.

Return Values

Boolean. If True, the start of the drag operation has been processed inside the event. If False, the start of the drag operation has not been processed inside the event.

OnMouseCancelDrag Event (Command Object)

Prerequisites: None

Object: Command Object

Fired when user cancels the drag by returning to the start point and takes their finger off the button.

Usage

```
Function Command_OnMouseCancelDrag(
        ByVal eButton As EPcbMouseButton) As Boolean
```

Arguments

- eButton

  The mouse button (EPcbMouseButton) that was released.

Return Values

Boolean. If True, the cancel drag operation has been processed inside the event. If False, the cancel drag operation has not been processed inside the event.

OnMouseClk Event (Command Object)

Prerequisites: None

Object: Command Object

Fired when a mouse button is clicked (the up-event).

Usage

```
Function Command_OnMouseClk(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dX As Double , ByVal dY As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was clicked (EPcbMouseButton)

- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.

- dX

  The X position of the mouse.

- dY

  The Y position of the mouse.

Return Values

Boolean. If True, the mouse click has been processed inside the event. If False, the mouse click has not been processed inside the event.

OnMouseDblClk Event (Command Object)

Prerequisites: None

Object: Command Object

Fired when a mouse button is double-clicked.

Usage

```
Function Command_OnMouseDblClk(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dX As Double , ByVal dY As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was double clicked (EPcbMouseButton)

- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.

- dX

  The X position of the mouse.

- dY

  The Y position of the mouse.

Return Values

Boolean. If True, the mouse double click has been processed inside the event. If False, the mouse double click has not been processed inside the event.

OnMouseDown Event (Command Object)

Prerequisites: None

Object: Command Object

Fired when a mouse button is pressed down.

Usage

```
Function Command_OnMouseDown(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dX As Double , ByVal dY As Double) As Boolean
```

Arguments

- eButton

  The mouse button that is down (EPcbMouseButton)

- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.

- dX

  The X position of the mouse.

- dY

  The Y position of the mouse.

Return Values

Boolean. If True, the mouse down has been processed inside the event. If False, the mouse down has not been processed inside the event.

OnMouseDrag Event (Command Object)

Prerequisites: None

Object: Command Object

Fired when the mouse moves while a drag is in progress.

Usage

```
Function Command_OnMouseDrag(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dXStart As Double , ByVal dYStart As Double ,
        ByVal dXNow As Double , ByVal dYNow As Double) As Boolean
```

Arguments

- eButton

  The mouse button that is active during drag (EPcbMouseButton)
- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.
- dXStart

  The starting X position of the mouse.
- dYStart

  The starting Y position of the mouse.
- dXNow

  The X position of the mouse.
- dYNow

  The Y position of the mouse.

Return Values

Boolean. If True, the mouse drag operation has been processed inside the event. If False, the mouse drag operation has not been processed inside the event.

OnMouseEndDrag Event (Command Object)

Prerequisites: None

Object: Command Object

Fired when user terminates the drag by taking their finger off the mouse button.

Usage

```
Function Command_OnMouseEndDrag(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dXStart As Double , ByVal dYStart As Double ,
        ByVal dXNow As Double , ByVal dYNow As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was active during the drag (EPcbMouseButton)
- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.
- dXStart

  The starting X position of the mouse.
- dYStart

  The starting Y position of the mouse.
- dXNow

  The X position of the mouse.
- dYNow

  The Y position of the mouse.

Return Values

Boolean. If True, the mouse end drag operation has been processed inside the event. If False, the mouse end drag operation has not been processed inside the event.

OnMouseMove Event (Command Object)

Prerequisites: None

Object: Command Object

Fired when the mouse is moved.

Usage

```
Function Command_OnMouseMove(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dX As Double , ByVal dY As Double) As Boolean
```

Arguments

- eButton

  The mouse button that is pressed during move (EPcbMouseButton)

- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.

- dX

  The X position of the mouse.

- dY

  The Y position of the mouse.

Return Values

Boolean. If True, the mouse move has been processed inside the event. If False, the mouse move has not been processed inside the event.

OnTerminate Event (Command Object)

Prerequisites: None

Object: Command Object

Fired when the command is exiting.

Usage

```
Sub Command_OnTerminate()
```

Arguments

None

OnUndo Event (Command Object)

Prerequisites: None

Object: Command Object

Fired when a request for Undo or Redo is made.

Usage

```
Function Command_OnUndo(ByVal bExecute As Boolean,
        ByVal bIsUndoEvent As Boolean) As Boolean
```

Arguments

- bExecute

  Typically, the server will fire this event in two steps:

  a. The event is fired with bExecute = False. In this mode, the server is requesting if the command can handle an undo event. If this is the case you should return True, if this is not the case you should return False.

  b. If the return value of the first event was True, the server will fire a second event with bExecute = True. In this mode it is requesting to handle the undo operation. Return True if the undo operation was successful, return False if it failed.

- bIsUndoEvent

  If True, the event is an Undo-event. If False, the event is a Redo-event.

Return Values

Boolean. If True, the event is an Undo-event. If False, the event is a Redo-event.

OnVChar Event (Command Object)

Prerequisites: None

Object: Command Object

Fired when a virtual character is typed on the keyboard.

Usage

```
Function Command_OnVChar(ByVal nChar As Long) As Boolean
```

Arguments

- nChar

  The key code of the virtual character that was pressed.

Return Values

Boolean. If True, the character has been processed inside the event. If False, the character has not been processed inside the event.

CommandListener Object

The CommandListener object is an event-based object that allows clients to Pre and Post-process existing interactive commands.

Table 41. CommandListener Object Properties and Events

Property or Event	Description
id Property (CommandListener Object)	Returns the current interactive command ID.
KeyboardFlags Property (CommandListener Object)	Returns the state of the Alt, Shift, and Ctrl modifier keys.
Name Property (CommandListener Object)	Returns the current interactive command name.
Unit Property (CommandListener Object)	Sets or returns the units coordinates use in the CommandListener events.
PostOnChar Event (CommandListener Object)	Fired after the current command processes the key press event.
PostOnCommand Event (CommandListener Object)	Fired after a command has been initialized.
PostOnEscape Event (CommandListener Object)	Fired after the current command processes the escape key press.
PostOnMouseBeginDrag Event (CommandListener Object)	Fired after the current command processes the begin drag event.
PostOnMouseCancelDrag Event (CommandListener Object)	Fired after the current command processes the cancel drag event.
PostOnMouseClk Event (CommandListener Object)	Fired after the current command process the click event (the up-event).
PostOnMouseDblClk Event (CommandListener Object)	Fired after the current command process the double-click event.
PostOnMouseDown Event (CommandListener Object)	Fired after the current command processes the mouse down event.
PostOnMouseDrag Event (CommandListener Object)	Fired after the current command processes the drag event.
PostOnMouseEndDrag Event (CommandListener Object)	Fired after the current command processes the end drag event.
PostOnMouseMove Event (CommandListener Object)	Fired after the current command processes the mouse move.
PostOnTerminate Event (CommandListener Object)	Fired after a command has been terminated.
PostOnVChar Event (CommandListener Object)	Fired after the current command processes the virtual key press.
PreOnChar Event (CommandListener Object)	Fired before the current command processes the key press event.
PreOnCommand Event (CommandListener Object)	Fired before a command is run.
PreOnEscape Event (CommandListener Object)	Fired before the current command processes the escape key press.

Table 41. CommandListener Object Properties and Events (continued)

Property or Event	Description
PreOnMouseBeginDrag Event (CommandListener Object)	Fired before the current command processes the begin drag event.
PreOnMouseCancelDrag Event (CommandListener Object)	Fired before the current command processes the cancel drag event.
PreOnMouseClk Event (CommandListener Object)	Fired before the current command process the click event (the up-event).
PreOnMouseDblClk Event (CommandListener Object)	Fired before the current command process the double-click event.
PreOnMouseDown Event (CommandListener Object)	Fired before the current command processes the mouse down event.
PreOnMouseDrag Event (CommandListener Object)	Fired before the current command processes the drag event.
PreOnMouseEndDrag Event (CommandListener Object)	Fired before the current command processes the end drag event.
PreOnMouseMove Event (CommandListener Object)	Fired before the current command process the mouse move.
PreOnVChar Event (CommandListener Object)	Fired before the current command processes the virtual key press.

id Property (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Access: Read-Only

Returns the current interactive command ID.

Usage

CommandListener.id

Arguments

None

Return Values

Long. A long that contains the command ID.

KeyboardFlags Property (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Access: Read-Only

Returns the state of the Alt, Shift, and Ctrl modifier keys.

Usage

```
CommandListener.KeyboardFlags
```

Description

This applies to the current active event (the event the automation client is processing). If there is no active event, the property returns the state of Alt, Shift, and Ctrl.

Arguments

None

Return Values

EPcbKeyboardFlags. The keyboard modifiers (EPcbKeyboardFlags).

Name Property (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Access: Read-Only

Returns the current interactive command name.

Usage

CommandListener.Name

Arguments

None

Return Values

String. A string that contains the command name.

Unit Property (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Access: Read/Write

Sets or returns the units coordinates use in the CommandListener events.

Usage

```
CommandListener.Unit = EPcbUnit
```

Arguments

None

Return Values

EPcbUnit. The units (EPcbUnit) to use.

Note:
epcbUnitCurrent cannot be used as a value for this property.

Description

The CommandListener object does not inherit units from the active document, so you must set the CommandListener units separately.

PostOnChar Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired after the current command processes the key press event.

Usage

```
Function CommandListener_PostOnChar(ByVal nChar As Long) As Boolean
```

Arguments

- nChar

  The key code that has been typed on the keyboard.

Return Values

Boolean. If True, the character was processed inside the event. If False, the character was not processed inside the event.

PostOnCommand Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired after a command has been initialized.

Usage

```
Sub CommandListener_PostOnCommand(ByVal CommandName As String ,
        ByVal CommandId As Long)
```

Arguments

- CommandName

  As string that contains the command name.

- CommandId

  A long that contains the command id.

PostOnEscape Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired after the current command processes the escape key press.

Usage

```
Function CommandListener_PostOnEscape() As Boolean
```

Arguments

None

Return Values

Boolean. If True, the escape key was processed inside the event. If False, the escape key was not processed inside the event.

PostOnMouseBeginDrag Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired after the current command processes the begin drag event.

Usage

```
Function CommandListener_PostOnMouseBeginDrag(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dXStart As Double , ByVal dYStart As Double ,
        ByVal dXNow As Double , ByVal dYNow As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was pressed (EPcbMouseButton)
- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.
- dXStart

  The starting X position of the mouse.
- dYStart

  The starting Y position of the mouse.
- dXNow

  The X position of the mouse.
- dYNow

  The Y position of the mouse.

Return Values

Boolean. If True, the start of the drag operation has been processed inside the event. If False, the start of the drag operation has not been processed inside the event.

PostOnMouseCancelDrag Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired after the current command processes the cancel drag event.

Usage

```
Function CommandListener_PostOnMouseCancelDrag(
        ByVal eButton As EPcbMouseButton) As Boolean
```

Arguments

- eButton

  The mouse button (EPcbMouseButton) that was released.

Return Values

Boolean. If True, the cancel drag operation has been processed inside the event. If False, the cancel drag operation has not been processed inside the event.

PostOnMouseClk Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired after the current command process the click event (the up-event).

Usage

```
Function CommandListener_PostOnMouseClk(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dX As Double , ByVal dY As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was clicked (EPcbMouseButton)

- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.

- dX

  The X position of the mouse.

- dY

  The Y position of the mouse.

Return Values

Boolean. If True, the mouse click has been processed inside the event. If False, the mouse click has not been processed inside the event.

PostOnMouseDblClk Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired after the current command process the double-click event.

Usage

```
Function CommandListener_PostOnMouseDblClk(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dX As Double , ByVal dY As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was double clicked. (EPcbMouseButton)

- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.

- dX

  The X position of the mouse.

- dY

  The Y position of the mouse.

Return Values

Boolean. If True, the mouse double click has been processed inside the event. If False, the mouse double click has not been processed inside the event.

PostOnMouseDown Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired after the current command processes the mouse down event.

Usage

```
Function CommandListener_PostOnMouseDown(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dX As Double , ByVal dY As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was pressed (EPcbMouseButton)

- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.

- dX

  The X position of the mouse.

- dY

  The Y position of the mouse.

Return Values

Boolean. If True, the mouse down has been processed inside the event. If False, the mouse down has not been processed inside the event.

PostOnMouseDrag Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired after the current command processes the drag event.

Usage

```
Function CommandListener_PostOnMouseDrag(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dXStart As Double , ByVal dYStart As Double ,
        ByVal dXNow As Double , ByVal dYNow As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was pressed (EPcbMouseButton)
- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.
- dXStart

  The starting X position of the mouse.
- dYStart

  The starting Y position of the mouse.
- dXNow

  The X position of the mouse.
- dYNow

  The Y position of the mouse.

Return Values

Boolean. If True, the mouse drag operation has been processed inside the event. If False, the mouse drag operation has not been processed inside the event.

PostOnMouseEndDrag Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired after the current command processes the end drag event.

Usage

```
Function CommandListener_PostOnMouseEndDrag(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dXStart As Double , ByVal dYStart As Double ,
        ByVal dXNow As Double , ByVal dYNow As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was pressed (EPcbMouseButton)
- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.
- dXStart

  The starting X position of the mouse.
- dYStart

  The starting Y position of the mouse.
- dXNow

  The X position of the mouse.
- dYNow

  The Y position of the mouse.

Return Values

Boolean. If True, the mouse end drag operation has been processed inside the event. If False, the mouse end drag operation has not been processed inside the event.

PostOnMouseMove Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired after the current command processes the mouse move.

Usage

```
Function CommandListener_PostOnMouseMove(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dX As Double , ByVal dY As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was pressed (EPcbMouseButton)

- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.

- dX

  The X position of the mouse.

- dY

  The Y position of the mouse.

Return Values

Boolean. If True, the mouse move has been processed inside the event. If False, the mouse move has not been processed inside the event.

PostOnTerminate Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired after a command has been terminated.

Usage

```
Sub CommandListener_PostOnTerminate(
        ByVal CommandName As String ,
        ByVal CommandId As Long)
```

Arguments

- CommandName

  As string that contains the command name.

- CommandId

  A long that contains the command id.

PostOnVChar Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired after the current command processes the virtual key press.

Usage

```
Function CommandListener_PostOnVChar(ByVal nChar As Long) As Boolean
```

Arguments

- nChar

  The key code of the virtual character that was pressed.

Return Values

Boolean. If True, the character has been processed inside the event. If False, the character has not been processed inside the event.

PreOnChar Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired before the current command processes the key press event.

Usage

```
Function CommandListener_PreOnChar(ByVal nChar As Long) As Boolean
```

Arguments

- nChar

  The key code that has been typed on the keyboard.

Return Values

Boolean. If True, allows the application to process the event. If False, prevents the application from processing the event.

PreOnCommand Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired before a command is run.

Usage

```
Sub CommandListener_PreOnCommand(ByVal CommandName As String ,
        ByVal CommandId As Long)
```

Arguments

- CommandName

  As string that contains the command name.

- CommandId

  A long that contains the command id.

Return Values

Boolean. If True, allows the application to process the event. If False, prevents the application from processing the event.

PreOnEscape Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired before the current command processes the escape key press.

Usage

```
Function CommandListener_PreOnEscape() As Boolean
```

Arguments

None

Return Values

Boolean. If True, allows the application to process the event. If False, prevents the application from processing the event.

PreOnMouseBeginDrag Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired before the current command processes the begin drag event.

Usage

```
Function CommandListener_PreOnMouseBeginDrag(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dXStart As Double , ByVal dYStart As Double ,
        ByVal dXNow As Double , ByVal dYNow As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was pressed (EPcbMouseButton)
- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.
- dXStart

  The starting X position of the mouse.
- dYStart

  The starting Y position of the mouse.
- dXNow

  The X position of the mouse.
- dYNow

  The Y position of the mouse.

Return Values

Boolean. If True, allows the application to process the event. If False, prevents the application from processing the event.

PreOnMouseCancelDrag Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired before the current command processes the cancel drag event.

Usage

```
Function CommandListener_PreOnMouseCancelDrag(
        ByVal eButton As EPcbMouseButton) As Boolean
```

Arguments

- eButton

  The mouse button (EPcbMouseButton) that was released.

Return Values

Boolean. If True, allows the application to process the event. If False, prevents the application from processing the event.

PreOnMouseClk Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired before the current command process the click event (the up-event).

Usage

```
Function CommandListener_PreOnMouseClk(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dX As Double , ByVal dY As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was clicked (EPcbMouseButton)

- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.

- dX

  The X position of the mouse.

- dY

  The Y position of the mouse.

Return Values

Boolean. If True, allows the application to process the event. If False, prevents the application from processing the event.

PreOnMouseDblClk Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired before the current command process the double-click event.

Usage

```
Function CommandListener_PreOnMouseDblClk(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dX As Double , ByVal dY As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was double clicked. (EPcbMouseButton)

- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.

- dX

  The X position of the mouse.

- dY

  The Y position of the mouse.

Return Values

Boolean. If True, allows the application to process the event. If False, prevents the application from processing the event.

PreOnMouseDown Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired before the current command processes the mouse down event.

Usage

```
Function CommandListener_PreOnMouseDown(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dX As Double , ByVal dY As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was pressed (EPcbMouseButton)

- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.

- dX

  The X position of the mouse.

- dY

  The Y position of the mouse.

Return Values

Boolean. If True, allows the application to process the event. If False, prevents the application from processing the event.

PreOnMouseDrag Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired before the current command processes the drag event.

Usage

```
Function CommandListener_PreOnMouseDrag(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dXStart As Double , ByVal dYStart As Double ,
        ByVal dXNow As Double , ByVal dYNow As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was pressed (EPcbMouseButton)
- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.
- dXStart

  The starting X position of the mouse.
- dYStart

  The starting Y position of the mouse.
- dXNow

  The X position of the mouse.
- dYNow

  The Y position of the mouse.

Return Values

Boolean. If True, allows the application to process the event. If False, prevents the application from processing the event.

PreOnMouseEndDrag Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired before the current command processes the end drag event.

Usage

```
Function CommandListener_PreOnMouseEndDrag(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dXStart As Double , ByVal dYStart As Double ,
        ByVal dXNow As Double , ByVal dYNow As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was pressed (EPcbMouseButton)
- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.
- dXStart

  The starting X position of the mouse.
- dYStart

  The starting Y position of the mouse.
- dXNow

  The X position of the mouse.
- dYNow

  The Y position of the mouse.

Return Values

Boolean. If True, allows the application to process the event. If False, prevents the application from processing the event.

PreOnMouseMove Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired before the current command process the mouse move.

Usage

```
Function CommandListener_PreOnMouseMove(
        ByVal eButton As EPcbMouseButton ,
        ByVal eFlags As EPcbKeyboardFlags ,
        ByVal dX As Double , ByVal dY As Double) As Boolean
```

Arguments

- eButton

  The mouse button that was pressed (EPcbMouseButton)

- eFlags

  The flags (EPcbKeyboardFlags) that indicate which keyboard buttons were pressed.

- dX

  The X position of the mouse.

- dY

  The Y position of the mouse.

Return Values

Boolean. If True, allows the application to process the event. If False, prevents the application from processing the event.

PreOnVChar Event (CommandListener Object)

Prerequisites: None

Object: CommandListener Object

Fired before the current command processes the virtual key press.

Usage

```
Function CommandListener_PreOnVChar(ByVal nChar As Long) As Boolean
```

Arguments

- nChar

  The key code of the virtual character that was pressed.

Return Values

Boolean. If True, allows the application to process the event. If False, prevents the application from processing the event.

Dialog Object

The Dialog object describes a dialog.

CAUTION:
When you utilize a name from an application's GUI, the automation results cannot be guaranteed from software release to software release. This occurs when the application's GUI changes. These names include: menu commands (Gui.ProcessCommand), dialog control names (Gui.FindDialog, Dialog.FindButton, Dialog.FindCheckbox and Dialog.ProcessCommand). If the application's GUI changes, you must update the corresponding Gui object names in your scripts.

Table 42. Dialog Object Methods and Events

Method or Event	Description
FindButton Method (Dialog Object)	Find the specified button on the dialog.
FindCheckBox Method (Dialog Object)	Find the specified check box on the dialog.
ProcessCommand Method (Dialog Object)	Starts the pulldown menu command on the dialog.
OnApply Event (Dialog Object)	Fired after the Apply button is clicked.
OnClosed Event (Dialog Object)	Fired after the dialog closes.
OnOK Event (Dialog Object)	Fired after the OK button is clicked.

FindButton Method (Dialog Object)

Prerequisites: None

Object: Dialog Object

Find the specified button on the dialog.

Usage

Dialog.FindButton(ByVal vButton As Variant) As IMGCPCBButton

Arguments

- vButton

  The button to look for. Supply the caption of the button you are looking for.

Return Values

IMGCPCBButton. A reference to a button (“Button” on page 155). Returns Nothing if the button was not found.

Related Topics

Running Commands with Automation

Running Commands with Automation

FindCheckBox Method (Dialog Object)

Prerequisites: None

Object: Dialog Object

Find the specified check box on the dialog.

Usage

```
Dialog.FindCheckBox(ByVal vCheckBox As Variant) As IMGCPCBCheckBox
```

Arguments

- **vCheckBox**

  The checkbox to look for. Supply the caption of the checkbox you are looking for.

Return Values

IMGCPCBCheckBox. A reference to a check box (“checkbox” on page 158). Returns Nothing if the check box was not found.

ProcessCommand Method (Dialog Object)

Prerequisites: None

Object: Dialog Object

Starts the pulldown menu command on the dialog.

Usage

```
Dialog.ProcessCommand(ByVal vCommand As Variant) As Boolean
```

Arguments

- vCommand

  This is either the ID (long) for the command to run or the name of the command to run.

  The name that is supplied is equal to the name that can be found on the pull down menus (excluding the ' …').

  If the name is unique for all menus you do not need to specify its hierarchical path. If it is not unique you must identify the hierarchical path using the '->' separator.

  Examples:

  Design Status, File->Import->DXF, File->Export->DXF.

Return Values

Boolean. Returns True if the command starts. Returns False if the command does not start.

Description

This method is identical to the "Gui.ProcessCommand" on page 238 method. The only difference is that Gui.ProcessCommand references the pulldown menu of the application and Dialog.ProcessCommand references the pulldown menu of a dialog.

OnApply Event (Dialog Object)

Prerequisites: None

Object: Dialog Object

Fired after the Apply button is clicked.

Usage

```
Sub Dialog_OnApply()
```

Arguments

None

OnClosed Event (Dialog Object)

Prerequisites: None

Object: Dialog Object

Fired after the dialog closes.

Usage

```
Sub Dialog_OnClosed()
```

Arguments

None

OnOK Event (Dialog Object)

Prerequisites: None

Object: Dialog Object

Fired after the OK button is clicked.

Usage

```
Sub Dialog_OnOK()
```

Arguments

None

Gui Object

The Gui object describes the application's Graphical User Interface (GUI).

CAUTION:
When you utilize a name from an application's GUI, the automation results cannot be guaranteed from software release to software release. This occurs when the application's GUI changes. These names include: menu commands (Gui.ProcessCommand), dialog control names (Gui.FindDialog, Dialog.FindButton, Dialog.FindCheckbox). If the application's GUI changes, you must update the corresponding Gui object names in your scripts.

Table 43. Gui Object Properties and Methods

Property or Method	Description
ActiveMode Property (Gui Object)	Sets or returns the active mode.
ActivePlaceLayer Property (Gui Object)	Sets or returns the active place layer.
ActiveRouteLayer Property (Gui Object)	Sets or returns the active route layer.
Bindings Property (Gui Object)	Maps key binding to command.
CommandBars Property (Gui Object)	Returns the collection of command bars.
CommandListener Property (Gui Object)	Returns a command listener object.
CursorBusy Method (Gui Object)	Sets the cursor to be displayed as busy or not busy.
DirectoryBrowser Method (GUI Object)	Displays a modal Directory browser dialog box.
Display Method (Gui Object)	Displays a string in a message box.
DisplayActionKeybar Method (Gui Object)	Sets a new Action Keybar.
DisplayMessage Method (Gui Object)	Displays a string in a message box or to the Output message window if the add-in is enabled.
FileBrowser Method (GUI Object)	Displays a modal File browser dialog box. If OpenFileDialog is set to False, displays the SaveAs dialog box.
FindDialog Method (Gui Object)	Finds an application dialog that is currently open.
HWND Property (Gui Object)	Returns the window handle of the server's main window.
InputBox Method (Gui Object)	Display an input box and returns value retrieved.
IsActionObjectCmd Method (Gui Object)	Returns true if an action object command is active.
KeyInCommandText Property (Gui Object)	Sets or returns the text in the keyin command field.
MeasureReadoutText Property (Gui Object)	Sets or returns the text in the Measure readout toolbar.
PopToFront Method (Gui Object)	Pops the application to the top of the window stack.

Table 43. Gui Object Properties and Methods (continued)

Property or Method	Description
ProcessCommand Method (Gui Object)	Starts supplied command.
ProcessKeyin Method (Gui Object)	Runs a keyin command.
ProcessKeyPressEvent Method (Gui Object)	Runs the shortcut key.
ProgressBar Method (Gui Object)	Updates the progress bar.
ProgressBarInitialize Method (Gui Object)	Initializes the progress bar with a caption and min and max values.
RegisterCommand Method (Gui Object)	Starts a new interactive command. This is a command that extends the command set of the application.
Settings Property (Gui Object)	Returns the Settings object.
ShowWindow Method (Gui Object)	Set state of application's main window.
StatusBarText Method (Gui Object)	Displays text in the application status bar.
SuppressNotepadDialogs Property (Gui Object)	Enable or disable the display of the notepad editor.
SuppressTrivialDialogs Property (Gui Object)	Enable or disable the display of trivial dialogs.
XYReadoutText Property (Gui Object)	Sets or returns the text in the XY readout toolbar.
TerminateCommand Method (Gui Object)	Terminates the active built-in command.
TerminateCommandEx Method (Gui Object)	Terminates the active built-in or automation command.

ActiveMode Property (Gui Object)

Prerequisites: None

Object: Gui Object

Access: Read/Write

Sets or returns the active mode.

Usage

```
Gui.ActiveMode = EPcbMode
```

Arguments

None

Return Values

EPcbMode. The active mode (EPcbMode) of the GUI.

ActivePlaceLayer Property (Gui Object)

Prerequisites: None

Object: Gui Object

Access: Read/Write

Sets or returns the active place layer.

Usage

```
Gui.ActivePlaceLayer = EPcbPlaceLayer
```

Arguments

None

Return Values

EPcbPlaceLayer. The active place layer (EPcbPlaceLayer) of the gui.

ActiveRouteLayer Property (Gui Object)

Prerequisites: None

Object: Gui Object

Access: Read/Write

Sets or returns the active route layer.

Usage

```
Gui.ActiveRouteLayer = Long
```

Arguments

None

Return Values

Long. A long that contains the layer number of the active route layer.

Bindings Property (Gui Object)

Prerequisites: None

Object: Gui Object

Access: Read-Only

Maps key binding to command.

Usage

```
Gui.Bindings (ByVal BTableName As String)
```

Arguments

- BTableName

  The name of the binding table. Available binding tables are “Application”, “Document” and “Accelerators”.

Return Values

BindingTables. Returns a Binding Tables Collection.

CommandBars Property (Gui Object)

Prerequisites: None

Object: Gui Object

Access: Read-Only

Returns the collection of command bars.

Usage

```
Gui.CommandBars
```

Arguments

None

Return Values

Object. Returns a collection of command bar objects.

CommandListener Property (Gui Object)

Prerequisites: None

Object: Gui Object

Access: Read-Only

Returns a command listener object.

Usage

Gui.CommandListener

Arguments

None

Return Values

IMGCPCBCommandListener. Returns a “CommandListener Object” on page 178.

CursorBusy Method (Gui Object)

Prerequisites: None

Object: Gui Object

Sets the cursor to be displayed as busy or not busy.

Usage

```
Gui.CursorBusy([ByVal bCursorBusy As Boolean = True])
```

Arguments

- bCursorBusy

  (Optional) If True the cursor is displayed as busy. If False the cursor will be displayed as not busy. If not specified the cursor is displayed as busy.

DirectoryBrowser Method (GUI Object)

Scope: Xpedition Layout, Xpedition Fablink, Xpedition FabLink DWG Editor, CellEditor.

Prerequisites: None

Object: Gui Object

Displays a modal Directory browser dialog box.

Usage

```
Gui.DirectoryBrowser([ByVal initialDirectory As String],
        [ByVal title As String = “SelectDirectory”]) As String
```

Arguments

- initialDirectory

  (Optional) A string that defines the initial directory in which to browse. If not specified, the default file location (e.g., Windows’ Documents library) is used.

- title

  (Optional) String that defines the text that appears in the title bar of the browser. The default is “Select Directory”.

Return Values

String. A string that contains the name of the selected directory. Returns an empty string if the selection is canceled.

Display Method (Gui Object)

Prerequisites: None

Object: Gui Object

Displays a string in a message box.

Usage

Gui.Display(ByVal Text As String)

Arguments

- Text

  A string containing the information to display in the message box.

DisplayActionKeybar Method (Gui Object)

Prerequisites: None.

Sets a new Action Keybar.

Usage

```
Gui.DisplayActionKeybar(
        ByVal keybarName As String ,
        [ByVal State As EPcbAKBState = epcbAKBStateDisplay]) As Boolean
```

Description

Gui is an object expression that evaluates to a Gui object. For more information about this object, refer to “Gui Object” on page 216.

Arguments

- keybarName

  The string that contains the keybar name.

- State

  (Optional) The action keybar state (EPcbAKBState).

Return Values

Boolean. Returns True if keybar is displayed.

DisplayMessage Method (Gui Object)

Prerequisites: None

Object: Gui Object

Displays a string in a message box or to the Output message window if the add-in is enabled.

Note:
Only informational messages are output to the Output message window.

Usage

```
Gui.DisplayMessage(
        ByVal Message As String ,
        [ByVal ReservedForFutureUse As String] ,
        [ByVal Type As EPcbMsgBoxType = epcbMsgBoxOk] ,
        [ByVal Icon As EPcbMsgBoxIcon = epcbMsgBoxIconExclamation] ,
        [ByVal DefaultButton As EPcbMsgBoxDefaultButton = epcbMsgDefaultBtn1]) As
        EPcbMsgBoxAnswer
```

Arguments

- Message

  The string that is output for display

- ReservedForFutureUse

  (Optional) For future use.

- Type

  (Optional) The message box type (EPcbMsgBoxType). The default is MsgBoxOK.

- Icon

  (Optional) The message box icon (EPcbMsgBoxIcon). The default is MsgBoxIconExclamation.

- DefaultButton

  (Optional) The default button (EPcbMsgBoxDefaultButton). The default is epcbMsgDefaultBtn1.

Return Values

Answer. The answer that is returned (EPcbMsgBoxAnswer).

Examples

```
Dim g
Set g = pcbApp.Gui
g.DisplayMessage("OK Information button1", "", _
        epcbMsgBoxOk, epcbMsgBoxIconInformation, epcbMsgDefaultBtn1)
g.DisplayMessage("OkCancel, Question, button2",  "", _
        epcbMsgBoxOkCancel, epcbMsgBoxIconQuestion, epcbMsgDefaultBtn2)
```

```
g.DisplayMessage("YesNo Exclamation button2",  "", _
        epcbMsgBoxYesNo, epcbMsgBoxIconExclamation, epcbMsgDefaultBtn2)
g.DisplayMessage("YesNoCancel Stop button3",  "", _
        epcbMsgBoxYesNoCancel, epcbMsgBoxIconStop, epcbMsgDefaultBtn3)
```

FileBrowser Method (GUI Object)

Scope: Xpedition Layout, Xpedition Fablink, Xpedition FabLink DWG Editor, CellEditor.

Prerequisites: None

Object: Gui Object

Displays a modal File browser dialog box. If OpenFileDialog is set to False, displays the SaveAs dialog box.

Usage

```
Gui.FileBrowser([ByVal OpenFileDialog As Boolean = True],
        [ByVal initialDirectory As String],
        [ByVal filter As String = “All Files (*.*)|*.*||”],
        [ByVal Extension As String],
        [ByVal filename As String],
        [ByVal eOptions As EPcbFileBrowserOptions = 14],
        [ByVal title As String = “Select File”]) As String
```

Arguments

- OpenFileDialog

  (Optional) Boolean. If set to True, displays File browser dialog box. If False, displays the SaveAs dialog box. Default is True.

- initialDirectory

  (Optional) A string that defines the initial directory in which to browse. If not specified, the default file location (e.g., Windows’ Documents library) is used.

- filter

  (Optional) A string that defines a filter. If not specified, the dialog box lists all files in the directory.

- Extension

  (Optional) A string that specifies a file extension.

- filename

  (Optional) A string that specifies a filename.

- eOptions

  (Optional) The options (EPcbFileBrowserOptions) to which this setting applies. The default is 14. See EPcbFileBrowserOptions Enum for more information.

- title

  (Optional) String that defines the text that appears in the title bar of the browser. The default is “Select File”.

Return Values

String. A string that contains the name of the selected file. Returns an empty string if the selection is canceled.

FindDialog Method (Gui Object)

Prerequisites: None

Object: Gui Object

Finds an application dialog that is currently open.

Usage

```
Gui.FindDialog(ByVal vDialog As Variant) As IMGCPCBDialog
```

Arguments

- vDialog

  The dialog to find. Supply the title string of the dialog you are looking for.

Return Values

IMGCPCBDialog. Returns a dialog box (“Dialog” on page 209) if found. Returns Nothing if the dialog was not found.

HWND Property (Gui Object)

Prerequisites: None

Object: Gui Object

Access: Read-Only

Returns the window handle of the server's main window.

Usage

```
Gui.HWND
```

Description

Use this handle to define a parent for any dialog that needs to behave as a child window in the PCB application.

Arguments

None

Return Values

Long. A long that contains the window handle of the server's main window.

InputBox Method (Gui Object)

Prerequisites: None

Object: Gui Object

Display an input box and returns value retrieved.

Usage

```
Gui.InputBox(ByVal sDescription As String , [ByVal sTitle As String] ,
        [ByVal sDefault As String]) As String
```

Arguments

- sDescription

  The description of the information that is requested.

- sTitle

  (Optional) The title that will be displayed in the title bar of the input box.

- sDefault

  (Optional) The default value of the information that is required. This value will be presented to the user if the input box is opened. The user can then modify it if required.

Return Values

String. Returns a string that contains the information the user typed in the input box.

IsActionObjectCmd Method (Gui Object)

Prerequisites: None

Object: Gui Object

Returns true if an action object command is active.

Usage

```
Gui.IsActionObjectCmd() As Boolean
```

Arguments

None

Return Values

Boolean. Returns true if an action object command is active. False, otherwise.

KeyInCommandText Property (Gui Object)

Prerequisites: None

Object: Gui Object

Access: Read/Write

Sets or returns the text in the keyin command field.

Note:
Use “ProcessKeyin” on page 239 to run a keyin command.

Usage

```
Gui.KeyInCommandText  = String
```

Arguments

None

Return Values

String. A string that contains the keyin command text.

MeasureReadoutText Property (Gui Object)

Prerequisites: None

Object: Gui Object

Access: Read/Write

Sets or returns the text in the Measure readout toolbar.

Usage

```
Gui.MeasureReadoutText As String
```

Arguments

None

Return Values

String. A string that contains the measure readout text.

PopToFront Method (Gui Object)

Prerequisites: None

Object: Gui Object

Pops the application to the top of the window stack.

Usage

```
Gui.PopToFront()
```

Arguments

None

ProcessCommand Method (Gui Object)

Prerequisites: None

Object: Gui Object

Starts supplied command.

Usage

```
Gui.ProcessCommand(ByVal vCommand As Variant ,
        [ByVal bSynchronous As Boolean = True]) As Boolean
```

Arguments

- vCommand

  This is either the ID (long) for the command to be executed or the name of the command to be executed.

  The name that is supplied is equal to the name that can be found on the pull down menus (excluding the ’ …’).

  If the name is unique for all menus you do not need to specify its hierarchical path. If it is not unique you must identify its hierarchical path using the ‘->’ separator.

  Examples:

  Design Status, File->Import->DXF, File->Export->DXF.

- bSynchronous

  (Optional) If True, the method will not return until the command execution is complete. If False, the requested command will be queued and the method returns immediately.

Return Values

Boolean. Returns True if the command could be started. Returns False if the command could not be started.

Description

The command IDs are used for internal purposes. They are not documented. It is advisable to use the command name to identify the command to be processed.

Refer to “Running Commands with Automation” on page 30 for more information on the use of this method.

ProcessKeyin Method (Gui Object)

Prerequisites: None

Object: Gui Object

Runs a keyin command.

Usage

```
Gui.ProcessKeyin(ByVal KeyinString As String)
```

Arguments

- KeyinString

  A string that contains the keyin command.

ProcessKeyPressEvent Method (Gui Object)

Prerequisites: None

Object: Gui Object

Runs the shortcut key.

Usage

```
Gui.ProcessKeyPressEvent(
        ByVal sKeyName As String ,
        [ByVal eModifiers As EPcbKeyboardFlags = epcbKeyboardFlagsNone])
```

Arguments

- sKeyName

  A string that contains the key name. Valid key names are: A-Z, 0-9, Backspace, Delete, End, Enter, Esc, Home, Insert, PageDown, PageUp, Space, Up, Down, Left, or Right.

  Note:
  The function keys, F1-F12 are not valid.

- eModifiers

  (Optional) Keyboard modifiers (EPcbKeyboardFlags), (Alt, Shift, and Ctrl keys). The default is no keyboard modifiers.

Examples

```
trace.Selected = True
‘ if a trace is selected, copy it and place it on the cursor
Gui.ProcessKeyPressEvent("c", epcbKeyboardFlagsCntrlPressed)
```

ProgressBar Method (Gui Object)

Prerequisites: None

Object: Gui Object

Updates the progress bar.

Usage

```
Gui.ProgressBar(ByVal val As Long)
```

Arguments

- val

  A long that contains the new value for the progress bar. The supplied value has to be between the minimum and maximum values that have been defined during progress bar initialization. (“ProgressBarInitialize Method” on page 242).

Description

Use the “ProgressBarInitialize Method” on page 242 to initialize the progress bar.

ProgressBarInitialize Method (Gui Object)

Prerequisites: None

Object: Gui Object

Initializes the progress bar with a caption and min and max values.

Usage

```
Gui.ProgressBarInitialize([ByVal bDisplay As Boolean = True] ,
    [ByVal BCaption As String = “Progress”] , [ByVal nMaxVal As Long = 100] ,
    [ByVal nMinVal As Long])
```

Arguments

- bDisplay

  (Optional) Set the display flag for the progress bar. If set to False, a previously initialized progress bar will be hidden.

- BCaption

  (Optional) Sets the caption of the progress bar.

- nMaxVal

  (Optional) Defines the maximum value that can be set.

- nMinVal

  (Optional) Defines the minimum value that can be set.

Description

Use the “ProgressBar Method” on page 241 to update the initialized progressbar with new values.

RegisterCommand Method (Gui Object)

Prerequisites: None

Object: Gui Object

Starts a new interactive command. This is a command that extends the command set of the application.

Usage

```
Gui.RegisterCommand(ByVal cmdName As String ,
        [ByVal bTerminateCurrent As Boolean = True]) As IMGCPCBCommand
```

Arguments

- cmdName

  This is the command name of the command you want to register. This name will be echoed in the prompt field to notify the user that the command is active

- bTerminateCurrent

  (Optional) If True, the current command is terminated before the new command is started.

Return Values

IMGCPCBCommand. Returns the Command Object for the command registered.

Description

Refer to “Creating New Commands” on page 33 for more information on this subject.

Settings Property (Gui Object)

Prerequisites: None

Object: Gui Object

Access: Read-Only

Returns the Settings object.

Usage

```
Gui.Settings
```

Arguments

None

Return Values

IMGCPCBSettings. Returns a Settings Object.

ShowWindow Method (Gui Object)

Prerequisites: None

Object: Gui Object

Set state of application’s main window.

Usage

```
Gui.ShowWindow(ByVal eWS As EPcbWindowState)
```

Arguments

- eWS

  The application’s main window state (EPcbWindowState).

StatusBarText Method (Gui Object)

Prerequisites: None

Object: Gui Object

Displays text in the application status bar.

Usage

```
Gui.StatusBarText(ByVal BValue As String ,
        [ByVal eField As EPcbStatusField = epcbStatusFieldPrompt])
```

Arguments

- BValue

  The string to be displayed.

- eField

  (Optional) The status field (EPcbStatusField) to use to display the text.

SuppressNotepadDialogs Property (Gui Object)

Prerequisites: None

Object: Gui Object

Access: Write-Only

Enable or disable the display of the notepad editor.

Usage

```
Gui.SuppressNotepadDialogs = True | False
```

Arguments

None

Return Values

True | False. If True, the notepad editor will not be displayed. If False, the notepad editor will be displayed.

Description

This editor is popped up after certain commands have run. (For example, Design Status). By default, this property is set to True.

Refer to “Running Commands with Automation” on page 30 for more information on the use of this method.

Examples

The following suppresses notepad dialog boxes.

```
Gui.SuppressNotepadDialogs = True
```

SuppressTrivialDialogs Property (Gui Object)

Prerequisites: None

Object: Gui Object

Access: Write-Only

Enable or disable the display of trivial dialogs.

Usage

```
Gui.SuppressTrivialDialogs = True | False
```

Arguments

None

Return Values

True | False. If True, trivial dialogs are not displayed. If False, trivial dialogs are displayed.

Description

Trivial dialogs are dialogs that are displayed after command completion informing the user of the completion status of a command. By suppressing these dialogs there is no user interaction required to dismiss these dialogs.

Examples

The following suppresses trivial dialog boxes.

```
Gui.SuppressTrivialDialogs = True
```

XYReadoutText Property (Gui Object)

Prerequisites: None

Object: Gui Object

Access: Read/Write

Sets or returns the text in the XY readout toolbar.

Usage

```
Gui.XYReadoutText As String
```

Arguments

None

Return Values

String. A string containing the text in the readout toolbar.

TerminateCommand Method (Gui Object)

Prerequisites: None

Object: Gui Object

Terminates the active built-in command.

Usage

```
Gui.TerminateCommand()
```

Arguments

None

Description

Refer to “Creating New Commands” on page 33 for more information on this subject.

TerminateCommandEx Method (Gui Object)

Prerequisites: None

Object: Gui Object

Terminates the active built-in or automation command.

Usage

```
Gui.TerminateCommandEx()
```

Arguments

None

Description

Refer to “Creating New Commands” on page 33 for more information on this subject.

Settings Object

The Settings object provides a mechanism for storing and retrieving settings.

Table 44. Settings Object Methods

Method	Description
GetDouble Method (Settings Object)	Returns the double setting.
GetDoubleArray Method (Settings Object)	Returns the double array setting.
GetInteger Method (Settings Object)	Returns the integer setting.
GetIntegerArray Method (Settings Object)	Returns the integer array setting.
GetString Method (Settings Object)	Returns the string setting.
GetStringArray Method (Settings Object)	Returns the string array setting.
PutDouble Method (Settings Object)	Sets the double setting.
PutDoubleArray Method (Settings Object)	Sets the double array setting.
PutInteger Method (Settings Object)	Sets the integer setting.
PutIntegerArray Method (Settings Object)	Sets the integer array setting.
PutString Method (Settings Object)	Sets the string setting.
PutStringArray Method (Settings Object)	Sets the string array setting.

GetDouble Method (Settings Object)

Prerequisites: None

Object: Settings Object

Returns the double setting.

Usage

```
Settings.GetDouble(ByVal ClientId As String ,
        ByVal SettingId As Long ,
        ByVal SettingType As EPcbSettingsType ,
        [ByVal DefaultValue As Double]) As Double
```

Arguments

- ClientId

  A string that defines the client identifier.
- SettingId

  A long that defines the setting identifier.
- SettingType

  The setting type (EPcbSettingsType Enum), user global, user design, or design.
- DefaultValue

  (Optional) A double that contains the default value.

Return Values

Double. A double that contains the setting value.

GetDoubleArray Method (Settings Object)

Prerequisites: None

Object: Settings Object

Returns the double array setting.

Usage

```
Settings.GetDoubleArray(ByVal ClientId As String ,
        ByVal SettingId As Long ,
        ByVal SettingType As EPcbSettingsType)
```

Arguments

- ClientId

  A string that defines the client identifier.
- SettingId

  A long that defines the setting identifier.
- SettingType

  The setting type (EPcbSettingsType Enum), user global, user design, or design.

Return Values

Variant. An array of double setting values.

GetInteger Method (Settings Object)

Prerequisites: None

Object: Settings Object

Returns the integer setting.

Usage

```
Settings.GetInteger(ByVal ClientId As String ,
        ByVal SettingId As Long ,
        ByVal SettingType As EPcbSettingsType ,
        [ByVal DefaultValue As Long]) As Long
```

Arguments

- ClientId

  A string that defines the client identifier.
- SettingId

  A long that defines the setting identifier.
- SettingType

  The setting type (EPcbSettingsType Enum), user global, user design, or design.
- DefaultValue

  (Optional) A long that contains the default value.

Return Values

Long. A long that contains the setting value.

GetIntegerArray Method (Settings Object)

Prerequisites: None

Object: Settings Object

Returns the integer array setting.

Usage

```
Settings.GetIntegerArray(ByVal ClientId As String ,
        ByVal SettingId As Long ,
        ByVal SettingType As EPcbSettingsType)
```

Arguments

- ClientId

  A string that defines the client identifier.
- SettingId

  A long that defines the setting identifier.
- SettingType

  The setting type (EPcbSettingsType Enum), user global, user design, or design.

Return Values

Variant. An array of integer setting values.

GetString Method (Settings Object)

Prerequisites: None

Object: Settings Object

Returns the string setting.

Usage

```
Settings.GetString(ByVal ClientId As String ,
        ByVal SettingId As Long ,
        ByVal SettingType As EPcbSettingsType ,
        [ByVal DefaultValue As String = "0"]) As String
```

Arguments

- ClientId

  A string that defines the client identifier.
- SettingId

  A long that defines the setting identifier.
- SettingType

  The setting type (EPcbSettingsType Enum), user global, user design, or design.
- DefaultValue

  (Optional) A string that contains the default value. If not specified the value is “0”.

Return Values

String. A string that contains the setting value.

GetStringArray Method (Settings Object)

Prerequisites: None

Object: Settings Object

Returns the string array setting.

Usage

```
Settings.GetStringArray(ByVal ClientId As String ,
        ByVal SettingId As Long ,
        ByVal SettingType As EPcbSettingsType)
```

Arguments

- ClientId

  A string that defines the client identifier.
- SettingId

  A long that defines the setting identifier.
- SettingType

  The setting type (EPcbSettingsType Enum), user global, user design, or design.

Return Values

Variant. An array of string setting values.

PutDouble Method (Settings Object)

Prerequisites: None

Object: Settings Object

Sets the double setting.

Usage

```
Settings.PutDouble(ByVal ClientId As String ,
        ByVal SettingId As Long ,
        ByVal SettingType As EPcbSettingsType ,
        ByVal DoubleVal As Double)
```

Arguments

- ClientId

  A string that defines the client identifier.
- SettingId

  A long that defines the setting identifier.
- SettingType

  The setting type (EPcbSettingsType Enum), user global, user design, or design.
- DoubleVal

  A double that contains the setting value.

PutDoubleArray Method (Settings Object)

Prerequisites: None

Object: Settings Object

Sets the double array setting.

Usage

```
Settings.PutDoubleArray(ByVal ClientId As String ,
        ByVal SettingId As Long ,
        ByVal SettingType As EPcbSettingsType ,
        ByVal DoubleArray As Variant)
```

Arguments

- ClientId

  A string that defines the client identifier.
- SettingId

  A long that defines the setting identifier.
- SettingType

  The setting type (EPcbSettingsType Enum), user global, user design, or design.
- DoubleArray

  An array of double setting values.

PutInteger Method (Settings Object)

Prerequisites: None

Object: Settings Object

Sets the integer setting.

Usage

```
Settings.PutInteger(ByVal ClientId As String ,
        ByVal SettingId As Long ,
        ByVal SettingType As EPcbSettingsType ,
        ByVal Integer As Long)
```

Arguments

- ClientId

  A string that defines the client identifier.
- SettingId

  A long that defines the setting identifier.
- SettingType

  The setting type (EPcbSettingsType Enum), user global, user design, or design.
- Integer

  A long that contains the setting value.

PutIntegerArray Method (Settings Object)

Prerequisites: None

Object: Settings Object

Sets the integer array setting.

Usage

```
Settings.PutIntegerArray(ByVal ClientId As String ,
        ByVal SettingId As Long ,
        ByVal SettingType As EPcbSettingsType ,
        ByVal IntegerArray As Variant)
```

Arguments

- ClientId

  A string that defines the client identifier.
- SettingId

  A long that defines the setting identifier.
- SettingType

  The setting type (EPcbSettingsType Enum), user global, user design, or design.
- IntegerArray

  An array of long setting values.

PutString Method (Settings Object)

Prerequisites: None

Object: Settings Object

Sets the string setting.

Usage

```
Settings.PutString(ByVal ClientId As String ,
        ByVal SettingId As Long ,
        ByVal SettingType As EPcbSettingsType ,
        ByVal StringVal As String)
```

Arguments

- ClientId

  A string that defines the client identifier.
- SettingId

  A long that defines the setting identifier.
- SettingType

  The setting type (EPcbSettingsType Enum), user global, user design, or design.
- StringVal

  A string that contains the setting value.

PutStringArray Method (Settings Object)

Prerequisites: None

Object: Settings Object

Sets the string array setting.

Usage

```
Settings.PutStringArray(ByVal ClientId As String ,
        ByVal SettingId As Long ,
        ByVal SettingType As EPcbSettingsType ,
        ByVal StringArray As Variant)
```

Arguments

- ClientId

  A string that defines the client identifier.
- SettingId

  A long that defines the setting identifier.
- SettingType

  The setting type (EPcbSettingsType Enum), user global, user design, or design.
- StringArray

  An array of string setting values.

Utility Object

The Utility object contains utility functions for conversion and object creation.

Table 45. Utility Object Properties and Methods

Property or Method	Description
ConvertUnit Method (Utility Object)	Returns the value converted to the specified unit.
CreateCircleXYR Method (Utility Object)	Creates a point array for a circle using a center point and a radius.
CreateRectXYR Method (Utility Object)	Creates a point array for a rectangle.
FindApplication Method (Utility Object)	Returns the application object of a running application instance.
GetUniqueIdString Method (Utility Object)	Converts a 64-bit integer Unique Id (or Change Record) into a string.
Globals Property (Utility Object)	Returns the Struct (Globals) object.
IsEqual Method (Utility Object)	Checks if two Unique Ids (or Change Records) are equal.
NewColor Method (Utility Object)	Returns a new color object.
NewColorPattern Method (Utility Object)	Returns a new ColorPattern object.
NewComponents Method (Utility Object)	Returns a new empty Components collection.
NewExtrema Method (Utility Object)	Returns a new Extrema object.
NewGroups Method (Utility Object)	Returns a new empty Groups collection.
NewLayerObject Method (Utility Object)	Returns a new, un-initialized LayerObject object.
NewLayerObjects Method (Utility Object)	Returns a new, empty LayerObjects collection.
NewLayerRange Method (Utility Object)	Returns a new LayerRange object.
NewLayerRanges Method (Utility Object)	Returns a new, empty LayerRanges collection.
NewNets Method (Utility Object)	Returns a new empty nets collection.
NewObjectFilter Method (Utility Object)	Creates an ObjectFilter object that contains the set of all objects.
NewObjects Method (Utility Object)	Returns a new, empty Objects collection.
NewPhysicalReuseLibraryCircuits Method (Utility Object)	Returns a new empty PhysicalReuseLibraryCircuits collection.

ConvertUnit Method (Utility Object)

Prerequisites: None

Object: Utility Object

Returns the value converted to the specified unit.

Usage

```
Utility.ConvertUnit(ByVal dValue As Double , ByVal eFromUnit As EPcbUnit ,
    ByVal eToUnit As EPcbUnit) As Double
```

Arguments

- dValue

  A double that contains the value to be converted.
- eFromUnit

  The units dValue uses.
- eToUnit

  The units to convert to.

Return Values

Double. A double that contains the converted value.

CreateCircleXYR Method (Utility Object)

Prerequisites: None

Object: Utility Object

Creates a point array for a circle using a center point and a radius.

Usage

```
Utility.CreateCircleXYR(ByVal dCenterX As Double , ByVal dCenterY As Double ,
    ByVal dRadiues As Double)
```

Arguments

- dCenterX

  A double containing the X-coordinate of the circle.
- dCenterY

  A double containing the Y-coordinate of the circle.
- dRadiues

  A double containing the radius of the circle

CreateRectXYR Method (Utility Object)

Prerequisites: None

Object: Utility Object

Creates a point array for a rectangle.

Usage

```
Utility.CreateRectXYR(ByVal dMinX As Double , ByVal dMinY As Double ,
      ByVal dMaxX As Double , ByVal dMaxY As Double)
```

Arguments

- dMinX

  A double containing the X-coordinate of the lower left corner.
- dMinY

  A double containing the Y-coordinate of the lower left corner.
- dMaxX

  A double containing the X-coordinate of the upper right corner.
- dMaxY

  A double containing the Y-coordinate of the upper right corner.

FindApplication Method (Utility Object)

Prerequisites: None

Object: Utility Object

Returns the application object of a running application instance.

Usage

Utility.FindApplication(ByVal vAppDescriptor As Variant) As IMGCPCBApplication

Arguments

- vAppDescriptor

  A string that specifies the full path of a document or an integer that specifies the index of the application instance (0 based indexing).

Return Values

IMGCPCBApplication. The object (Application) that matches the index number or document specified in vAppDescriptor.

Examples

```
' This is a template script for a script that will get
' a running instance of the server with a job already open
' There is no error checking of the pcbDoc or pcbApp objects

Option Explicit

' Get the application object
Dim pcbApp
Set pcbApp = GetObject(,"MGCPCB.ExpeditionPCBApplication")

' add a reference to the MGCPCB type library in order to use enums
Scripting.AddTypeLibrary("MGCPCB.ExpeditionPCBApplication")

Dim appNmArr
appNmArr = GetRunningDocumentNames(pcbApp)Dim i

Dim msgBoxStr
msgBoxStr = "What application instance would you like to attach to?" &_
    vbCrLf
For i = 0 To UBound(appNmArr, 1)
    msgBoxStr = msgBoxStr & i + 1 & ". " & appNmArr(i) & vbCrLf
Next

dim retVal
retVal = InputBox(msgBoxStr)

' Pass the document name into the FindApplication function to get the
' specific application
```

```
Set pcbApp = pcbApp.Utility.FindApplication(appNmArr(CInt(retval) - 1))

If pcbApp Is Nothing Then
    MsgBox "Application instance not found."
Else
    pcbApp.Gui.Display "Attached to " & pcbApp.ActiveDocument.FullName
End If

'-------------------------------------------------------------------------
' Local functions
Private Function GetRunningDocumentNames(anApp)

Dim nmArr: ReDim nmArr(32)
Dim aDoc
Dim cnt: cnt = 0
Dim localApp: Set localApp = anApp

' Get the first application in the list.
Set localApp = localApp.Utility.FindApplication(cnt)

While Not localApp Is Nothing
' Put the document name in the list.
    Set aDoc = localApp.ActiveDocument
    If Not aDoc Is Nothing Then
        nmArr(cnt) = aDoc.FullName
    Else
        nmArr(cnt) = ""
    End If
    ' Increment the index
    cnt = cnt + 1
    ' Get the application object at the next index
    Set localApp = localApp.Utility.FindApplication(cnt)
Wend

' Reset the size to be the correct size
ReDim Preserve nmArr(cnt - 1)

' Set the return value
GetRunningDocumentNames = nmArr
End Function
```

GetUniqueIdString Method (Utility Object)

Prerequisites: None

Object: Utility Object

Converts a 64-bit integer Unique Id (or Change Record) into a string.

Usage

```
Utility.GetUniqueIdString(
        ByVal UniqueId As Variant) As String
```

Arguments

- UniqueId

  A 64-bit integer

Return Values

String. A string that contains the Unique Id

Globals Property (Utility Object)

Prerequisites: None

Object: Utility Object

Access: Read-Only

Returns the Struct (Globals) object.

Usage

Utility.Globals

Description

This property is equivalent to Scripting.Globals. Refer to the Struct Object for addition information on this object. Refer to Scripting.Globals for examples of use.

Arguments

None

Return Values

Object. The Struct Object.

IsEqual Method (Utility Object)

Prerequisites: None

Object: Utility Object

Checks if two Unique Ids (or Change Records) are equal.

Usage

```
Utility.IsEqual(
        ByVal UniqueId1 As Variant ,
        ByVal UniqueId2 As Variant) As Boolean
```

Arguments

- UniqueId1

  A variant containing the Unique Id of the first object, or a Change Record. The variant can be a 64-bit integer or string.

- UniqueId2

  A variant containing the Unique Id of the second object, or a Change Record. The variant can be a 64-bit integer or string.

Return Values

Boolean. If True, the Ids are equal. If False, they are not equal.

NewColor Method (Utility Object)

Prerequisites: None

Object: Utility Object

Returns a new color object.

Usage

```
Utility.NewColor([ByVal nRed As Integer = 255] ,
        [ByVal nGreen As Integer = 255] ,
        [ByVal nBlue As Integer = 255]) As IMGCPCBColor
```

Arguments

- nRed

  (Optional) The red component of the color
- nGreen

  (Optional) The green component of the color.
- nBlue

  (Optional) The blue component of the color.

Return Values

IMGCPCBColor. A color object (Color) representing the requested color.

NewColorPattern Method (Utility Object)

Prerequisites: None

Object: Utility Object

Returns a new ColorPattern object.

Usage

```
Utility.NewColorPattern([ByVal nRed As Integer = 255] ,
        [ByVal nGreen As Integer = 255] ,
        [ByVal nBlue As Integer = 255] ,
        [ByVal nAlphaBlend As Integer = 100] ,
        [ByVal nPattern As Integer] ,
        [ByVal bTransparent As Boolean = False] ,
        [ByVal bOutline As Boolean = False]) As IMGCPCBColorPattern
```

Arguments

- nRed

  (Optional) The red component of the color
- nGreen

  (Optional) The green component of the color.
- nBlue

  (Optional) The blue component of the color.
- nAlphaBlend

  (Optional) The transparency value of the color.
- nPattern

  (Optional) Index of the pattern to use. This index number references the index numbers that are used in the *GraphicsPatterns.txt* file in the MGCPCB *config* directory.
- bTransparent

  (Optional) If True, the pattern is transparent.
- bOutline

  (Optional) If True, an outline is drawn around the pattern.

Return Values

IMGCPCBColorPattern. A color pattern (ColorPattern) representing the supplied arguments.

NewComponents Method (Utility Object)

Prerequisites: None

Object: Utility Object

Returns a new empty Components collection.

Usage

```
Utility.NewComponents() As IMGCPCBComponents
```

Arguments

None

Return Values

IMGCPCBComponents. An empty collection of component objects (Components)

Description

Use this utility method if you need to group a set of components into one collection.

NewExtrema Method (Utility Object)

Prerequisites: None

Object: Utility Object

Returns a new Extrema object.

Usage

```
Utility.NewExtrema(ByVal MinX As Double , ByVal MinY As Double ,
        ByVal MaxX As Double , ByVal MaxY As Double ,
        [ByVal eUnit As EPcbUnit = epcbUnitCurrent]) As IMGCPCBExtrema
```

Arguments

- MinX

  The lower X-coordinate of the extrema.
- MinY

  The lower Y-coordinate of the extrema.
- MaxX

  The upper X-coordinate of the extrema.
- MaxY

  The upper Y-coordinate of the extrema.
- eUnit

  (Optional) The units to use for the X,Y arguments that define the new object.

Note:
You should specify a unit type other than **epcbUnitCurrent** for this argument, otherwise the method uses current design units.

Return Values

IMGCPCBExtrema. An extrema object (Extrema) with the specified extents.

NewGroups Method (Utility Object)

Prerequisites: None

Object: Utility Object

Returns a new empty Groups collection.

Usage

```
Utility.NewGroups() As IMGCPCBGroups
```

Arguments

None

Return Values

IMGCPCBGroups. An empty Groups collection (Groups).

NewLayerObject Method (Utility Object)

Prerequisites: None

Object: Utility Object

Returns a new, un-initialized LayerObject object.

Usage

```
Utility.NewLayerObject() As IMGCPCBLayerObject
```

Arguments

None

Return Values

IMGCPCBLayerObject. A layer object (LayerObject).

NewLayerObjects Method (Utility Object)

Prerequisites: None

Object: Utility Object

Returns a new, empty LayerObjects collection.

Usage

```
Utility.NewLayerObjects() As IMGCPCBLayerObjects
```

Arguments

None

Return Values

IMGCPCBLayerObjects. An empty LayerObjects collection.

NewLayerRange Method (Utility Object)

Prerequisites: None

Object: Utility Object

Returns a new LayerRange object.

Usage

Utility.NewLayerRange(ByVal StartLayer As Long , ByVal EndLayer As Long) As IMGCPCBLayerRange

Arguments

- StartLayer

  A long that contains the start layer number.
- EndLayer

  A long that contains the end layer number.

Return Values

IMGCPCBLayerRange. A layer range object (LayerRange).

NewLayerRanges Method (Utility Object)

Prerequisites: None

Object: Utility Object

Returns a new, empty LayerRanges collection.

Usage

```
Utility.NewLayerRanges () As IMGCPCBLayerRanges
```

Arguments

None

Return Values

IMGCPCBLayerRanges. An empty LayerRanges collection.

NewNets Method (Utility Object)

Prerequisites: None

Object: Utility Object

Returns a new empty nets collection.

Usage

```
Utility.NewNets() As IMGCPCBNets
```

Arguments

None

Return Values

IMGCPCBNets. An empty collection of net objects. (Nets).

Description

Use this utility method if you need to group a set of nets in one collection.

NewObjectFilter Method (Utility Object)

Prerequisites: None

Object: Utility Object

Creates an ObjectFilter object that contains the set of all objects.

Usage

```
Utility.NewObjectFilter() As IMGCPCBObjectFilter
```

Arguments

None

Return Values

IMGCPCBObjectFilter. A new ObjectFilter object.

NewObjects Method (Utility Object)

Prerequisites: None

Object: Utility Object

Returns a new, empty Objects collection.

Usage

```
Utility.NewObjects() As IMGCPCBObjects
```

Arguments

None

Return Values

IMGCPCBObjects. An empty collection of objects. (Objects)

Description

Use this utility method when you need to create an empty collection of objects. This supports the MoveRelative method to allow you to move objects of different types. Examples include adding vias and traces to the same collection and then moving as a group.

NewPhysicalReuseLibraryCircuits Method (Utility Object)

Prerequisites: None

Object: Utility Object

Returns a new empty PhysicalReuseLibraryCircuits collection.

Usage

```
Utility.NewPhysicalReuseLibraryCircuits() As IMGCPCBPhysicalReuseLibraryCircuits
```

Arguments

None

Return Values

IMGCPCBPhysicalReuseLibraryCircuits. An empty collection of PhysicalReuseLibraryCircuit objects. (PhysicalReuseLibraryCircuits).

Description

Use this utility method if you need to group a set of physical reuse library circuits in one collection.

Chapter 3
Project Integration

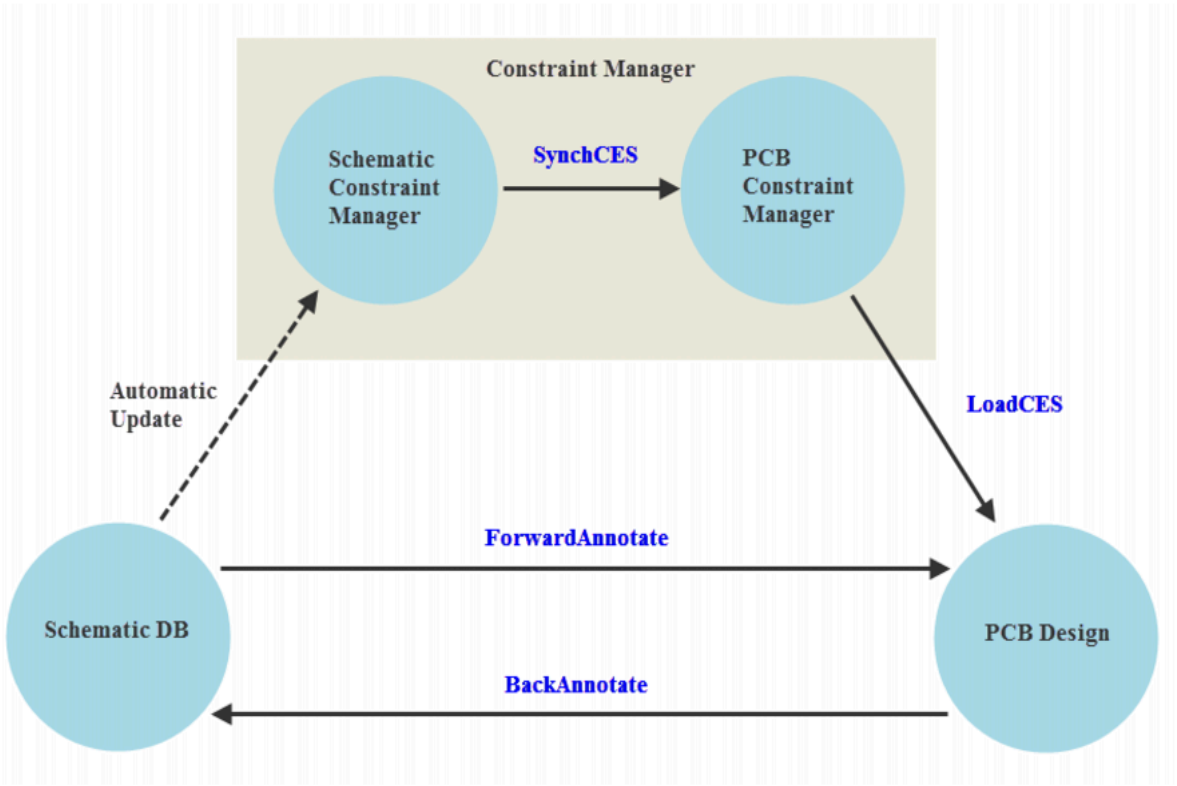
PCB Automation supports project integration.

ProjectIntegration Object

The ProjectIntegration object establishes a link between a design entry tool, a PCB layout tool and the Constraint Manager). You use it to define how data flows between the databases, through Back Annotation, Forward Annotation, the loading and synchronization of constraints defined in Constraint Manager.

The following shows the data flow between the Schematic, PCB Design, and Constraint Manager databases.

Figure 9. Project Integration Data Flow



The following is an example that shows the use of the ProjectIntegration methods and properties.

Example 17. ProjectIntegration Example

```
Dim docObj
Dim prjIntObj
Dim s, success
s=""

' Define project integration enums, dll is not registered.
Const eprjintStatusInSynch = 2
Const eprjintStatusRequired = 1
Const eprjintStatusNoCES = 3
```

```
' collect document object
Set docObj = GetLicensedDoc(app)
If (docObj Is Nothing) Then Exit Sub

' get the project integration
Set prjIntObj = docObj.ProjectIntegration

s = s + "Project File: " + prjIntObj.ProjectFile + vbCrLf

' Synchronize Constraint Manager schematic data
If prjIntObj.IsSynchCESAllowed Then
 Select Case prjIntObj.SynchCESStatus
  Case eprjintStatusRequired
   s = s + "Synchronizing Constraint Manager schematic" + vbCrLf +_
   "changes into PCB Constraint Manager..." + vbCrLf
   success = prjIntObj.SynchCES
   If success Then
    s = s + "Synchronize Constraint Manager successful" + vbCrLf
   Else
    s = s + "Synchronize Constraint Manager unsuccessful" + vbCrLf
   End If
  Case eprjintStatusInSynch
   s = s + "Constraint Manager data is already synchronized" +_
   vbCrLf
  Case eprjintStatusNoCES
   s = s + "Design flow does not use Constraint Manager" + vbCrLf
 End Select
Else
 s = s + _
 "Synchronizing Constraint Manager schematic changes" + vbCrLf +_
 "into PCB Constraint Manager not allowed" + vbCrLf
End If

' Load Constraint Manager data into PCB design
If prjIntObj.IsLoadCESAllowed Then
 Select Case prjIntObj.LoadCESStatus
  Case eprjintStatusRequired
   s = s + "Loading Constraint Manager changes into" + vbCrLf +_
   "PCB Design..." + vbCrLf
   prjIntObj.LoadCES
  Case eprjintStatusInSynch
   s = s + "Constraint Manager data is already " + vbCrLf +_
   "up to date in PCB Design" + vbCrLf
  Case eprjintStatusNoCES
   s = s + "Design flow does not use Constraint Manager" + vbCrLf
 End Select
Else
 s = s + "Loading Constraint Manager changes into" + vbCrLf +_
 "PCB design is not allowed" + vbCrLf
End If

' Back annotation
If prjIntObj.IsBackAnnotationAllowed Then
```

```
 Select Case prjIntObj.BackAnnotationStatus
  Case eprjintStatusRequired
   s = s + "Running Back Annotation..." + vbCrLf
   prjIntObj.BackAnnotate
  Case eprjintStatusInSynch
   s = s + "Back Annotation not required" + vbCrLf
 End Select
Else
 s = s + "Back Annotation not allowed" + vbCrLf
End If

' Forward annotation
If prjIntObj.IsForwardAnnotationAllowed Then
 Select Case prjIntObj.ForwardAnnotationStatus
  Case eprjintStatusRequired
   s = s + "Running Forward Annotation..." + vbCrLf
   prjIntObj.ForwardAnnotate
  Case eprjintStatusInSynch
   s = s + "Forward Annotation not required" + vbCrLf
 End Select
Else
 s = s + "Forward Annotation not allowed" + vbCrLf
End If
MsgBox s
```

The following section defines the methods and properties available in the ProjectIntegration object.

Table 46. ProjectIntegration Object Method and Properties

Method or Property	Description
BackAnnotate Method (ProjectIntegration Object)	Run back annotation.
BackAnnotationStatus Property (ProjectIntegration Object)	Checks if back annotation is required.
ForwardAnnotate Method (ProjectIntegration Object)	Run forward annotation.
ForwardAnnotationStatus Property (ProjectIntegration Object)	Checks if forward annotation is required.
IsBackAnnotationAllowed Method (ProjectIntegration Object)	Checks if back annotation is allowed.
IsForwardAnnotationAllowed Method (ProjectIntegration Object)	Checks if forward annotation is allowed.
IsLoadCESAllowed Method (ProjectIntegration Object)	Checks if loading Constraint Manager changes into the PCB design is allowed.
IsSynchCESAllowed Method (ProjectIntegration Object)	Checks if Constraint Manager connectivity changes can be synchronized into PCB Constraint Manager.
LoadCES Method (ProjectIntegration Object)	Load the Constraint Manager changes into the PCB design.

Table 46. ProjectIntegration Object Method and Properties (continued)

Method or Property	Description
LoadCESStatus Property (ProjectIntegration Object)	Checks if Constraint Manager PCB data has changed and therefore needs to be loaded into PCB design.
ProjectFile Property (ProjectIntegration Object)	Returns the project file name.
SynchCES Method (ProjectIntegration Object)	Synchronize Constraint Manager schematic data into PCB Constraint Manager.
SynchCESStatus Property (ProjectIntegration Object)	Checks if Constraint Manager schematic data has changed and therefore needs to be synchronized into PCB Constraint Manager.
EPrjIntStatus Enum	Project Integration status constants.

BackAnnotate Method (ProjectIntegration Object)

Prerequisites: None

Object: ProjectIntegration Object

Run back annotation.

Usage

```
ProjectIntegration.BackAnnotate() As Boolean
```

Arguments

None

Return Values

Boolean. If True, back annotation ran successfully. If False, back annotation was not error successful.

BackAnnotationStatus Property (ProjectIntegration Object)

Prerequisites: None

Object: ProjectIntegration Object

Access: Read-Only

Checks if back annotation is required.

Usage

ProjectIntegration.BackAnnotationStatus

Arguments

None

Return Values

"EPrjIntStatus" on page 305. Returns the status (EPrjIntStatus) of back annotation.

- If **eprjintStatusInSynch**, there are no PCB design changes to back annotate.
- If **eprjintStatusRequired**, there are PCB design changes to back annotate.

ForwardAnnotate Method (ProjectIntegration Object)

Prerequisites: None

Object: ProjectIntegration Object

Run forward annotation.

Usage

ProjectIntegration.ForwardAnnotate() As Boolean

Arguments

None

Return Values

Boolean. If True, forward annotation ran successfully. If False, forward annotation was not successful.

ForwardAnnotationStatus Property (ProjectIntegration Object)

Prerequisites: None

Object: ProjectIntegration Object

Access: Read-Only

Checks if forward annotation is required.

Usage

ProjectIntegration.ForwardAnnotationStatus

Arguments

None

Return Values

EPrjIntStatus. Returns the status ("EPrjIntStatus" on page 305) of forward annotation.

- If **eprjintStatusInSynch**, there are no connectivity changes to forward annotate to the PCB design.
- If **eprjintStatusRequired** then there are connectivity changes to forward annotate to the PCB design.

IsBackAnnotationAllowed Method (ProjectIntegration Object)

Prerequisites: None

Object: ProjectIntegration Object

Checks if back annotation is allowed.

Usage

```
ProjectIntegration.IsBackAnnotationAllowed() As Boolean
```

Arguments

None

Return Values

Boolean. If True, back annotation is allowed. If False, back annotation is not allowed.

IsForwardAnnotationAllowed Method (ProjectIntegration Object)

Prerequisites: None

Object: ProjectIntegration Object

Checks if forward annotation is allowed.

Usage

```
ProjectIntegration.IsForwardAnnotationAllowed() As Boolean
```

Arguments

None

Return Values

Boolean. If True, forward annotation is allowed. If False, forward annotation is not allowed.

IsLoadCESAllowed Method (ProjectIntegration Object)

Prerequisites: None

Object: ProjectIntegration Object

Checks if loading Constraint Manager changes into the PCB design is allowed.

Usage

```
ProjectIntegration.IsLoadCESAllowed() As Boolean
```

Arguments

None

Return Values

As Boolean. If True, loading Constraint Manager changes is allowed. If False, loading Constraint Manager changes is not allowed.

IsSynchCESAllowed Method (ProjectIntegration Object)

Prerequisites: None

Object: ProjectIntegration Object

Checks if Constraint Manager connectivity changes can be synchronized into PCB Constraint Manager.

Usage

```
ProjectIntegration.IsSynchCESAllowed() As Boolean
```

Arguments

None

Return Values

Boolean. If True, Constraint Manager synchronization is allowed. If False, Constraint Manager synchronization is not allowed.

LoadCES Method (ProjectIntegration Object)

Prerequisites: None

Object: ProjectIntegration Object

Load the Constraint Manager changes into the PCB design.

Usage

```
ProjectIntegration.LoadCES() As Boolean
```

Arguments

None

Return Values

Boolean. If True, the Constraint Manager data loaded successfully. If False, the Constraint Manager data did not load successfully.

LoadCESStatus Property (ProjectIntegration Object)

Prerequisites: None

Object: ProjectIntegration Object

Access: Read-Only

Checks if Constraint Manager PCB data has changed and therefore needs to be loaded into PCB design.

Usage

ProjectIntegration.LoadCESStatus

Arguments

None

Return Values

EPrjIntStatus. Return the status ("EPrjIntStatus" on page 305) of the Constraint Manager changes to PCB.

- If **eprjintStatusNoCES**, the design flow does not use Constraint Manager.
- If **eprjintStatusInSynch**, Constraint Manager data does not need to be loaded into PCB design.
- If **eprjintStatusRequired**, Constraint Manager data has changed and needs to be loaded into PCB design.

ProjectFile Property (ProjectIntegration Object)

Prerequisites: None

Object: ProjectIntegration Object

Access: Read-Only

Returns the project file name.

Usage

ProjectIntegration.ProjectFile

Arguments

None

Return Values

String. A string that contains the name of the project file (*.prj*) including the path.

SynchCES Method (ProjectIntegration Object)

Prerequisites: None

Object: ProjectIntegration Object

Synchronize Constraint Manager schematic data into PCB Constraint Manager.

Usage

```
ProjectIntegration.SynchCES() As Boolean
```

Arguments

None

Return Values

Boolean. If True, the synchronize operation was successful. If False, the synchronize operation failed.

SynchCESStatus Property (ProjectIntegration Object)

Prerequisites: None

Object: ProjectIntegration Object

Access: Read-Only

Checks if Constraint Manager schematic data has changed and therefore needs to be synchronized into PCB Constraint Manager.

Usage

ProjectIntegration.SynchCESStatus

Arguments

None

Return Values

EPrjIntStatus. Returns the status ("EPrjIntStatus" on page 305) of the Constraint Manager synchronization.

- If **eprjintStatusNoCES**, the design flow does not use Constraint Manager.
- If **eprjintStatusInSynch**, there are no Constraint Manager schematic changes that require synchronization into PCB Constraint Manager.
- If **eprjintStatusRequired**, there are Constraint Manager schematic changes that require synchronization into PCB Constraint Manager.

EPrjIntStatus Enum

Prerequisites: None

Object: ProjectIntegration Object

Project Integration status constants.

Usage

EPrjIntStatus.*Constant*

Arguments

- **eprjintStatusInSynch**

  The database operation is in synchronization.

  The numerical value for this constant is 2.

- **eprjintStatusNoCES**

  The design flow does not use Constraint Manager.

  The numerical value for this constant is 3.

- **eprjintStatusRequired**

  A database synchronization is required.

  The numerical value for this constant is 1.

Description

You cannot use the Scripting.AddTypeLibrary property to define these enumerators. Instead, define your enumerators using the appropriate syntax for the scripting language you use. For example:

```
Const eprjintStatusInSynch = 2
```

```
Const eprjintStatusRequired = 1
```

```
Const eprjintStatusNoCES = 3
```

Chapter 4
Output Window Add-in

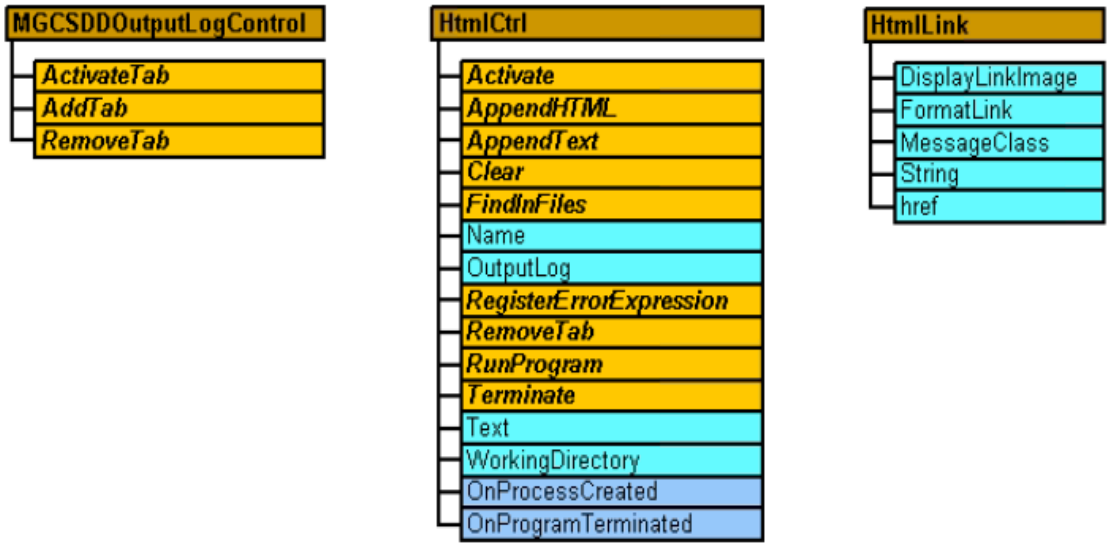
This section describes the Output Window Add-in. You can script the add-in to display and link messages to design data objects and to output hypertext links to external sources that provide information.

Output Window Data Model

Each object has additional methods and properties. However, only the methods and properties described in this section are illustrated.

Figure 10: Output Window Data Model shows the data model for the Output Window Add-in.

Figure 10. Output Window Data Model



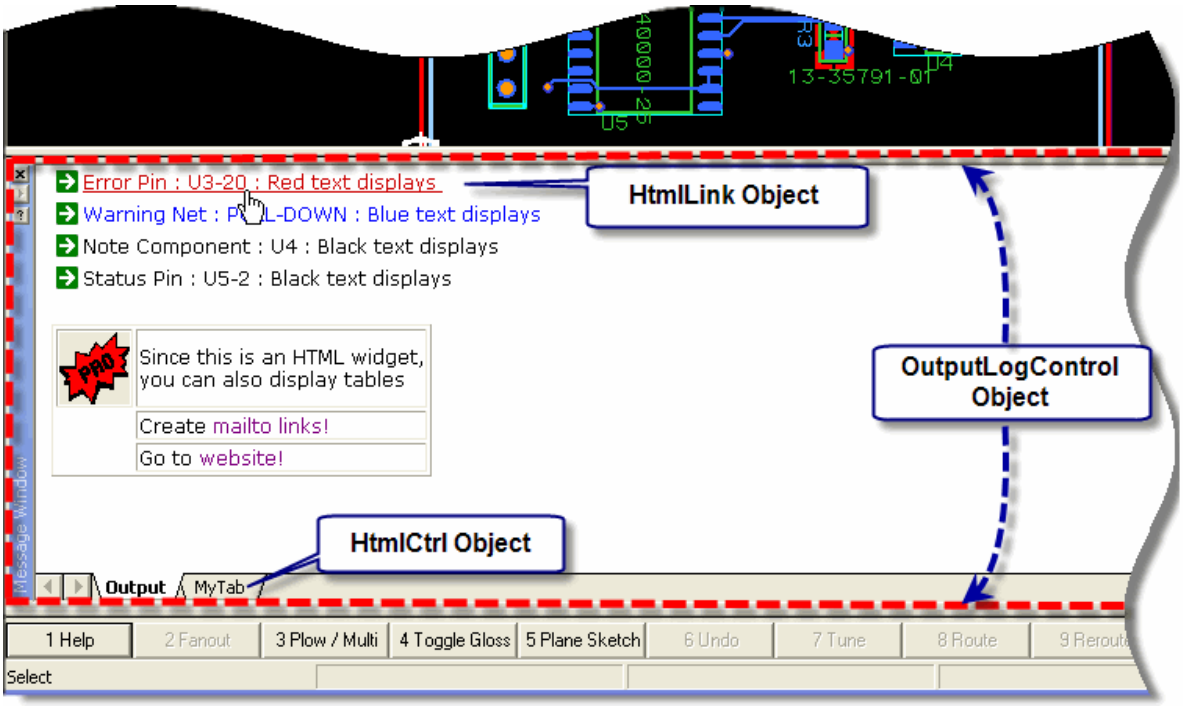
Output Window Object Descriptions

This section provides the descriptions of Output Window objects and the associated methods, properties and events you can access using the interface.

For information on the application Addins collection, refer to MGCAddins Collection in the Common Automation Reference.

For detailed information on the Output Window objects, refer to the individual objects described in this section. Figure 11: Add-in Objects shows the relationship between the Message Window GUI and the COM interface objects.

Figure 11. Add-in Objects



The scripts in Example 18: OutoutWindowCrossProbe.vbs and Example 19: OutputWindowCrossProbeText.vbs must be run consecutively. Example 18: OutoutWindowCrossProbe.vbs watches for text being written to the output window. If the text contains the form UID:<UID>, it is formatted into a hyperlink that will cross probe to the object with the specified UID (a via in this case). Example 19: OutputWindowCrossProbeText.vbs writes to the output window if it detects a via in the design.

Example 18. OutoutWindowCrossProbe.vbs

```
' This is a persistent script. It will continue running and watch for
' text being written to the output tab. When a line is written that
' contains text of the form UID:<UID> it will automatically format that
' text to be a hyperlink for cross probing to the object with the
' specified UID.

' To use this script drop OutputWindowCrossProbe.vbs on your design.
```

```
' Nothing will happen at this time. Next drop
' OutputWindowCrossProbeText.vbs. If you have a via in the design
' it will write a line to the output window, which you can click to
' select and fit the via.

Option Explicit

Dim pcbApp
Dim pcbDoc : Set pcbDoc = Nothing
Dim outputAddin

main

Sub main()
   Set pcbApp = Application
   If RegisterCrossProbeExpression Then
    Scripting.DontExit = True
 End If
 ' Sample usage would be done in a seperate script
 'Output("Padstack UID:23061")
End Sub

' Any time text is added that is a regular expression match to
' what is specified in RegisterCrossProbeExpression this function
' is called to give you the opportunity to format the expression.
Function FormatObject(str, linkObj)
    Dim regEx, Match, Matches
   Dim UID,Text
   Set regEx = New RegExp
   regEx.Pattern = "(.*)UID:([0-9]+)"
 regEx.IgnoreCase = True
   regEx.Global = True
   Set Matches = regEx.Execute(str)
   Set Match = Matches(0)
 Text = Match.SubMatches(0)
 UID = Match.SubMatches(1)
 linkObj.MessageClass = 1
 linkObj.String = Text
 ' linkobj.hRef is what is used in the function called when
 ' the link is clicked. VisitObject will use this. See below.
 linkObj.hRef = "UID:" & UID
 linkObj.FormatLink = True
   linkObj.HelpKey = ""
   linkObj.HelpId = 0
 linkObj.DisplayLinkImage = True
   FormatObject = True
End Function

' Whenever the text is selected that was formated by FormatObject
' this function is called and the linkObj is passed to it.
Function VisitObject(anchor)
    ' Parse the UID.
   Dim url: url = unescape(anchor.hRef)
 Dim UID: UID = Split(url, ":")(1)
```

```
 ' Unselect everything.
 GetDoc.UnselectAll
 ' Find the object with this UID. Select and fit it.
 Dim obj: Set obj = GetDoc.FindObjectById(UID)
 obj.Selected = True
 FitObject(obj)
End Function

' Registers the format and visit callbacks for the UID expression
' on the output tab.
Function RegisterCrossProbeExpression
 Dim outputTab: Set outputTab = GetOutputTab
   If Not (outputTab Is Nothing) Then
      Call outputTab.RegisterErrorExpression("(.*)UID:([0-9]+)",
         ScriptEngine, "VisitObject", "FormatObject")
      RegisterCrossProbeExpression = True
   Else
      MsgBox "Error: Message Window cross prober " & vbCrLf & _
         "can't find the Message Window Addin"
  RegisterCrossProbeExpression = False
   End If
End Function

Function FitObject(obj)
 If Not obj Is Nothing Then
  Dim ext
  Set ext = obj.Extrema
  Call GetDoc.ActiveView.SetExtentsToExtremaEx(ext, True)
 Else
  MsgBox "Object now found."
 End If
End Function

Function Output(str)
 Dim outputTab: Set outputTab = GetOutputTab
 outputTab.AppendHTML(str)
End Function

Function GetOutputWindow()
 Set GetOutputWindow = pcbApp.Addins.Item("Message Window")
End Function

Function GetOutputTab()
 Dim outputWindow: Set outputWindow = GetOutputWindow
 If Not outputWindow Is Nothing Then
  Set GetOutputTab = outputWindow.Control.AddTab("TestCrossProbe")
   Else
  Set GetOutputTab = Nothing
 End If
End Function


Function GetDoc()
 If pcbDoc Is Nothing Then
```

```
  Set pcbDoc = pcbApp.ActiveDocument
  ValidateServer(pcbDoc)
 End If
 Set GetDoc = pcbDoc
End Function

'Server validation function
Private Function ValidateServer(doc)
   Dim key, licenseServer, licenseToken
   ' Ask Expedition's document for the key
   key = doc.Validate(0)
   ' Get license server
   Set licenseServer =
      CreateObject("MGCPCBAutomationLicensing.Application")
   ' Ask the license server for the license token
   licenseToken = licenseServer.GetToken(key)
   ' Release license server
   Set licenseServer = Nothing
   ' Turn off error messages. Validate may fail if the token is
 incorrect.
   On Error Resume Next
   Err.Clear
   ' Ask the document to validate the license token
   doc.Validate (licenseToken)
   If Err Then
      ValidateServer = 0
   Else
      ValidateServer = 1
   End If
End Function
```

Example 19. OutputWindowCrossProbeText.vbs

```
' Caveat: This script is meant to be run in combination with
' OutputWindowCrossProbe.vbs. Run OutputWindowCrossProbe.vbs first;
' it will watch for text being written to the output window. When it
' detects text that contains UID:<UID>, it will make the text into an
' automatic hyperlink that cross probes to the object with the specified
' UID.

Option Explicit

' Add any type libraries to be used.
Scripting.AddTypeLibrary("MGCPCB.ExpeditionPCBApplication")

' Get the application object
Dim pcbApp
Set pcbApp = Application

' Get the active Document
Dim pcbDoc
Set pcbDoc = pcbApp.ActiveDocument
```

```
' License the pcbDoc
ValidateServer(pcbDoc)

Dim viaColl: Set viaColl = pcbDoc.Vias
If viaColl.Count > 0 Then
 Dim viaObj: Set viaObj = viaColl.Item(1)
 ' Any text containing UID:<UID> will automatically cross probe if
 ' OutputWindowCrossProbe.vbs is running.
 Output("Click to cross probe via - " & viaObj.UniqueID & " UID:" & _
      viaObj.UniqueId)
Else
 MsgBox "There are no vias in the design. Place " & vbCrLf & _
      "a single via to run this example."
End If

Function Output(str)
 Dim outputTab: Set outputTab = GetOutputTab
 outputTab.AppendHTML(str)
End Function

Function GetOutputWindow()
 Set GetOutputWindow = pcbApp.Addins.Item("Message Window")
End Function

Function GetOutputTab()
 Dim outputWindow: Set outputWindow = GetOutputWindow
 If Not outputWindow Is Nothing Then
  Set GetOutputTab = outputWindow.Control.AddTab("TestCrossProbe")
 Else
  Set GetOutputTab = Nothing
 End If
End Function



''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'Local functions

' Server validation function
Private Function ValidateServer(doc)
   dim key, licenseServer, licenseToken
   ' Ask Expedition's pcbDoc for the key
   key = doc.Validate(0)
   ' Get license server
   Set licenseServer =
      CreateObject("MGCPCBAutomationLicensing.Application")
   ' Ask the license server for the license token
   licenseToken = licenseServer.GetToken(key)
   ' Release license server
   Set licenseServer = nothing
   ' Turn off error messages. Validate may fail if the token is
 incorrect.
    On Error Resume Next
    Err.Clear
```

```
    ' Ask the pcbDoc to validate the license token
     doc.Validate(licenseToken)
     If Err Then
        ValidateServer = 0
     Else
        ValidateServer = 1
     End If
End Function
```

Table 47. Output Window Objects

Object	Description
HtmlCtrl Object	This object is the Output Window tab. Use it to format and add text and HTML to the tab. You can also use it to change the working directory, to find files and to run programs.
HtmlLink Object	This object is the linked text in an Output Window tab. It defines the string to display, if the text is linked (active), the link's message class (color) and the destination URL or anchor point for the link.
OutputLogControl Object	This object is the Output Window control object. Use it to add, remove and activate window tabs.

HtmlCtrl Object

This object is the Output Window tab. Use it to format and add text and HTML to the tab. You can also use it to change the working directory, to find files and to run programs.

Table 48. HtmlCtrl Object Methods, Properties, and Events

Method, Property, or Event	Description
Activate Method (HtmlCtrl Object)	Activates (pops to the front) the tab.
AppendHTML Method (HtmlCtrl Object)	Appends HTML format text to the window tab.
AppendText Method (HtmlCtrl Object)	Appends text to window tab.
Clear Method (HtmlCtrl Object)	Clears the contents of the window tab.
FindInFiles Method (HtmlCtrl Object)	Finds the files with the specified extensions that match the regular expression.
Name Property (HtmlCtrl Object)	Sets or returns the name of the tab.
OutputLog Property (HtmlCtrl Object)	Returns the Output Window (OutputLogControl object) for the tab.
RegisterErrorExpression Method (HtmlCtrl Object)	Parses a string using a regular expression to match. When matched, the client calls the format function (FormatFuncName) to format the text or calls the visit function (VisitFuncName) if the formatted (linked) text is clicked.
RemoveTab Method (HtmlCtrl Object)	Removes the tab from the window.
RunProgram Method (HtmlCtrl Object)	Runs a command and displays output to the Output Window.
Terminate Method (HtmlCtrl Object)	Terminates the process returned by the OnProcessCreated event.
Text Property (HtmlCtrl Object)	Sets or returns the contents of the window tab as text.
WorkingDirectory Property (HtmlCtrl Object)	Sets or returns the working directory.
OnProcessCreated Event (HtmlCtrl Object)	Fired when the program specified by the RunProgram method starts.
OnProgramTerminated Event (HtmlCtrl Object)	Fired when the program specified by the RunProgram method finishes.

Activate Method (HtmlCtrl Object)

Prerequisites: None

Object: HtmlCtrl Object

Activates (pops to the front) the tab.

Usage

```
HtmlCtrl.Activate()
```

Arguments

None

AppendHTML Method (HtmlCtrl Object)

Prerequisites: None

Object: HtmlCtrl Object

Appends HTML format text to the window tab.

Usage

```
HtmlCtrl.AppendHTML(ByVal HTML As String)
```

Arguments

- HTML

  A string that contains HTML format.

Description

The text is formatted if the “RegisterErrorExpression” on page 322 method is called prior to appending the HTML. The “RegisterErrorExpression” on page 322 calls the FormatFuncName to format the HTML and returns the resulting HtmlLink Object. Refer to “RegisterErrorExpression Method” on page 322 for more information.

AppendText Method (HtmlCtrl Object)

Prerequisites: None

Object: HtmlCtrl Object

Appends text to window tab.

Usage

```
HtmlCtrl.AppendText(ByVal Text As String)
```

Arguments

- Text

  A string that contains text.

Description

The text is formatted if the “RegisterErrorExpression” on page 322 method is called prior to appending the text. The “RegisterErrorExpression” on page 322 method calls the FormatFuncName to format the text and returns the resulting HtmlLink Object.Refer to “RegisterErrorExpression Method” on page 322 for more information.

Clear Method (HtmlCtrl Object)

Prerequisites: None

Object: HtmlCtrl Object

Clears the contents of the window tab.

Usage

HtmlCtrl.Clear()

Arguments

None

Examples

```
outputTab.Clear
```

FindInFiles Method (HtmlCtrl Object)

Prerequisites: None

Object: HtmlCtrl Object

Finds the files with the specified extensions that match the regular expression.

Usage

```
HtmlCtrl.FindInFiles(
        ByVal SearchRegexp As String ,
        ByVal FileTypes As String ,
        ByVal StartingFolder As String ,
        ByVal CaseSensitive As Long ,
        ByVal Recurse As Long)
```

Arguments

- SearchRegexp

  Regular expression to search for.

- FileTypes

  List of extensions.

- StartingFolder

  Starting directory.

- CaseSensitive

  If True (1), the search is case sensitive. If False (0), the search is case insensitive.

- Recurse

  If True (1), search directories below the starting directory. If False (0), search only the starting directory.

Name Property (HtmlCtrl Object)

Prerequisites: None

Object: HtmlCtrl Object

Access: Read/Write

Sets or returns the name of the tab.

Usage

HtmlCtrl.Name = String

Arguments

None

Return Values

String. A string that contains the name of the tab.

Description

By default, the name is the same as the tab name. If you change the Name property, the tab name still remains the same.

Examples

```
OutputTab.Name = “MyTab”
```

OutputLog Property (HtmlCtrl Object)

Prerequisites: None

Object: HtmlCtrl Object

Access: Read/Write

Returns the Output Window (OutputLogControl object) for the tab.

Usage

HtmlCtrl.OutputLog = IOutputLogControl

Arguments

None

Return Values

IOutputLogControl. The Output Window (OutputLogControl Object).

Examples

```
Set OutputWindowCntl = outputTab.OutputLog
```

RegisterErrorExpression Method (HtmlCtrl Object)

Prerequisites: None

Object: HtmlCtrl Object

Parses a string using a regular expression to match. When matched, the client calls the format function (FormatFuncName) to format the text or calls the visit function (VisitFuncName) if the formatted (linked) text is clicked.

Usage

```
HtmlCtrl.RegisterErrorExpression(
        ByVal Regexp As String ,
        ByVal Client As Object ,
        ByVal VisitFuncName As String ,
        ByVal FormatFuncName As String)
```

Arguments

- Regexp

  A regular expression to match.

- Client

  The client that calls the VisitFuncName and FormatFuncName functions.

- VisitFuncName

  The name of the visit function. This is defined in your script. The visit function is defined with one parameter, which is the HtmlLink Object for the linked text when the text is clicked.

- FormatFuncName

  The name of the format function. This is defined in your script. The format function is defined with two parameters. The first parameter is the string that results from matching the Regexp string with the string defined in the “AppendText” on page 317 or “AppendHTML” on page 316 methods. The second parameter is the HtmlLink Object that results.

Description

The string is defined by the “AppendText” on page 317 or “AppendHTML” on page 316 methods.

RemoveTab Method (HtmlCtrl Object)

Prerequisites: None

Object: HtmlCtrl Object

Removes the tab from the window.

Usage

HtmlCtrl.RemoveTab()

Arguments

None

Examples

```
outputTab.Remove
```

RunProgram Method (HtmlCtrl Object)

Prerequisites: None

Object: HtmlCtrl Object

Runs a command and displays output to the Output Window.

Usage

```
HtmlCtrl.RunProgram(
        ByVal Program As String ,
        ByVal Sync As Boolean) As Long
```

Arguments

- Program

  A command.

- Sync

  If True (1), then run the command and wait for it to complete. If False (0), run the command and continue running the script without waiting for the command to complete.

Return Values

As Long. Returns the status of the command.

Examples

```
dirReq = InputBox("Directory to display?")
' Get the tab with name specified by end user.
' If the tab already exists the AddTab method activates it
Set theTab = outputAddin.Control.AddTab("Dir " & dirReq)
theTab.RunProgram "%COMSPEC% /c dir " & dirReq, False
```

Terminate Method (HtmlCtrl Object)

Prerequisites: None

Object: HtmlCtrl Object

Terminates the process returned by the OnProcessCreated event.

Usage

```
HtmlCtrl.Terminate(
        ByVal Pid As Long ,
        ByVal ExitCode As Long)
```

Arguments

- Pid

  Process id.

- ExitCode

  Exit code to process.

Description

See “OnProcessCreated” on page 328 for more information.

Text Property (HtmlCtrl Object)

Prerequisites: None

Object: HtmlCtrl Object

Access: Read/Write

Sets or returns the contents of the window tab as text.

Usage

```
HtmlCtrl.Text = String
```

Arguments

None

Return Values

String. A string that contains the window tab contents.

Description

This property differs from the AppendText method. Setting the property overwrites any existing text and HTML output.

WorkingDirectory Property (HtmlCtrl Object)

Prerequisites: None

Object: HtmlCtrl Object

Access: Read/Write

Sets or returns the working directory.

Usage

HtmlCtrl.WorkingDirectory = String

Arguments

None

Return Values

String. A string that contains the working directory.

Examples

```
currentDir = outTab.WorkingDirectory
outTab.AppendText ("Working directory is: " & currentDir & vbCrLf)
```

OnProcessCreated Event (HtmlCtrl Object)

Prerequisites: None

Object: HtmlCtrl Object

Fired when the program specified by the RunProgram method starts.

Usage

```
Sub HtmlCtrl_OnProcessCreated(
        ByVal Pid As Long ,
        ByVal Program As String)
```

Arguments

- Pid

  Process id.

- Program

  Program run by the RunProgram method.

Description

See “RunProgram” on page 324 for more information.

OnProgramTerminated Event (HtmlCtrl Object)

Prerequisites: None

Object: HtmlCtrl Object

Fired when the program specified by the RunProgram method finishes.

Usage

```
Sub HtmlCtrl_OnProgramTerminated(
        ByVal Pid As Long ,
        ByVal nExitCode As Long)
```

Arguments

- Pid

  Process id.

- nExitCode

  Exit code to process.

Description

See “RunProgram” on page 324 for more information.

HtmlLink Object

This object is the linked text in an Output Window tab. It defines the string to display, if the text is linked (active), the link's message class (color) and the destination URL or anchor point for the link.

Table 49. HtmlLink Object Properties

Property	Description
DisplayLinkImage Property (HtmlLink Object)	Indicates whether to display a link image next to the string. To display the link image the FormatLink property must also be True.
FormatLink Property (HtmlLink Object)	Specifies if the text is linked.
href Property (HtmlLink Object)	Specifies a link.
MessageClass Property (HtmlLink Object)	Sets or returns the message class.
String Property (HtmlLink Object)	The string displayed in the Output Window.

DisplayLinkImage Property (HtmlLink Object)

Prerequisites: None

Object: HtmlLink Object

Access: Read/Write

Indicates whether to display a link image next to the string. To display the link image the FormatLink property must also be True.

Usage

```
HtmlLink.DisplayLinkImage = Boolean
```

Arguments

None

Return Values

Boolean. If True, show a green arrow next to the text. If False, do not show the image. By default, this property is True.

FormatLink Property (HtmlLink Object)

Prerequisites: None

Object: HtmlLink Object

Access: Read/Write

Specifies if the text is linked.

Usage

```
HtmlLink.FormatLink = Boolean
```

Arguments

None

Return Values

Boolean. If True, create linked text. If False, the text is not linked.

href Property (HtmlLink Object)

Prerequisites: None

Object: HtmlLink Object

Access: Read/Write

Specifies a link.

Usage

```
HtmlLink.href = String
```

Arguments

None

Return Values

String. A string that contains a link.

MessageClass Property (HtmlLink Object)

Prerequisites: None

Object: HtmlLink Object

Access: Read/Write

Sets or returns the message class.

Usage

```
HtmlLink.MessageClass = MsgClass
```

Arguments

None

Return Values

MsgClass. The message class (MsgClass Enum).

- Errors (MsgClassERROR or 3) appear as red linked text.
- Warnings (MsgClassWARNING or 2) appear as blue linked text.
- Notes (MsgClassNOTE or 1) and Plain (MsgClassPLAIN or 0) appear as black linked text.‘

String Property (HtmlLink Object)

Prerequisites: None

Object: HtmlLink Object

Access: Read/Write

The string displayed in the Output Window.

Usage

```
HtmlLink.String = String
```

Arguments

None

Return Values

String. The string contained in the Output Window.

OutputLogControl Object

This object is the Output Window control object. Use it to add, remove and activate window tabs.

Table 50. OutputLogControl Object Methods

Method	Description
ActivateTab Method (OutputLogControl Object)	Activates (pops to the front) the named tab.
AddTab Method (OutputLogControl Object)	Adds a tab to the Output Window. The added tab is automatically set as the active tab.
RemoveTab Method (OutputLogControl Object)	Removes the tab from the Output Window.

ActivateTab Method (OutputLogControl Object)

Prerequisites: None

Object: OutputLogControl Object

Activates (pops to the front) the named tab.

Usage

```
OutputLogControl.ActivateTab(ByVal TabName As String)
```

Arguments

- TabName

  A string that contains the tab to activate.

AddTab Method (OutputLogControl Object)

Prerequisites: None

Object: OutputLogControl Object

Adds a tab to the Output Window. The added tab is automatically set as the active tab.

Usage

```
OutputLogControl.AddTab(ByVal TabName As String) As IHtmlCtrl
```

Arguments

- TabName

  A string that contains the name of the tab.

Return Values

IHtmlCtrl. Returns the control (HtmlCtrl Object) for the added tab.

RemoveTab Method (OutputLogControl Object)

Prerequisites: None

Object: OutputLogControl Object

Removes the tab from the Output Window.

Note:
You cannot remove the currently active tab.

Usage

```
OutputLogControl.RemoveTab(ByVal TabName As String)
```

Arguments

- TabName

  A string that contains the name of the tab to remove.

Output Window Enumerates

The following summarizes the Output Window enumerated types.

Table 51. Output Window Enumerates

Enumerate	Description
HtmlCtrlEventIDs Enum	Event IDs.
MsgClass Enum	Error type enumerates.

HtmlCtrlEventIDs Enum

Prerequisites: None

Event IDs.

Usage

HtmlCtrlEventIDs

Arguments

- DISPID_EVENT_ONPROCESSCREATED

  The numerical value for this constant is 2.
- DISPID_EVENT_ONPROGRAMTERMINATED

  The numerical value for this constant is 1.

MsgClass Enum

Prerequisites: None

Error type enumerates.

Note:
You must declare these constants in your script, if the script refers to them by name. Refer to the example below.

Usage

MsgClass

Arguments

- MsgClassERROR

  Linked text will be Red. The numerical value for this constant is 3.
- MsgClassNOTE

  Linked text will be Black. The numerical value for this constant is 1.
- MsgClassPLAIN

  Linked text will be Black. The numerical value for this constant is 0.
- MsgClassWARNING

  Linked text will be Blue. The numerical value for this constant is 2.

Examples

```
MsgClassError = 3
MsgClassWarning = 2
MsgClassNote = 1
MsgClassPlain = 0
...
linkObj.MessageClass = MsgClassError
' The following is not as clear, though equivelent,
'  to the previous statement
linkObj.MessageClass = 3
```

Appendix A
Xpedition Layout Team Server Automation

This section provides information about automation limitations when scripting for Xpedition Layout Team Server.

For more information on Xpedition Layout Team Server refer to the *Xpedition Layout Team Guide* or the *BoardStation XE User's Guide.*

Disabled Methods and Properties

When you run an Xpedition Layout Team Server session, some methods and properties are disabled.

Table 52: Disabled Methods/Properties in Xpedition Layout Team Server lists the disabled methods and properties.

Table 52. Disabled Methods/Properties in Xpedition Layout Team Server

Disabled Methods/Properties
Component.ReplaceCell
Component.ResetCell
Document.NewRoutePass
Document.PutAngularDimension
Document.PutBoardOutline
Document.PutCluster
Document.PutCopperBalancingShape
Document.PutDetailedView
Document.PutDRCWindow
Document.PutFiducial
Document.PutManufacturingOutline
Document.PutMountingHole
Document.PutOrdinateDimension
Document.PutPointToPointDimension

Table 52. Disabled Methods/Properties in Xpedition Layout Team Server (continued)

Disabled Methods/Properties
Document.PutReservedArea
Document.PutRoom
Document.PutRouteBorder
Document.PutRouteFence
Document.PutXAngleDimenstion
EditorControl.PadEntryGlobalRules
Padstack.Clone
Padstack.DeletePad
Padstack.PutPad

Read-Only Objects

Some objects are Read Only during an Xpedition Layout Team Server session.

Table 53: Read-Only Objects in Xpedition Layout Team Server lists the objects that are Read Only during an Xpedition Layout Team Server session.

Note:
All Methods associated with Read-Only objects are disabled.

Table 53. Read-Only Objects in Xpedition Layout Team Server

Read-Only Objects
BoardOutline
Cluster
CopperBalancingShape
DetailedView
Dimension
DRCWindow
Fiducial

Table 53. Read-Only Objects in Xpedition Layout Team Server (continued)

Read-Only Objects
ManufacturingOutline
MountingHole
Padstack
Room
RouteBorder

Third-Party Information

Details on open source and third-party software that may be included with this product are available in the *<your_software_installation_location>/legal* directory.