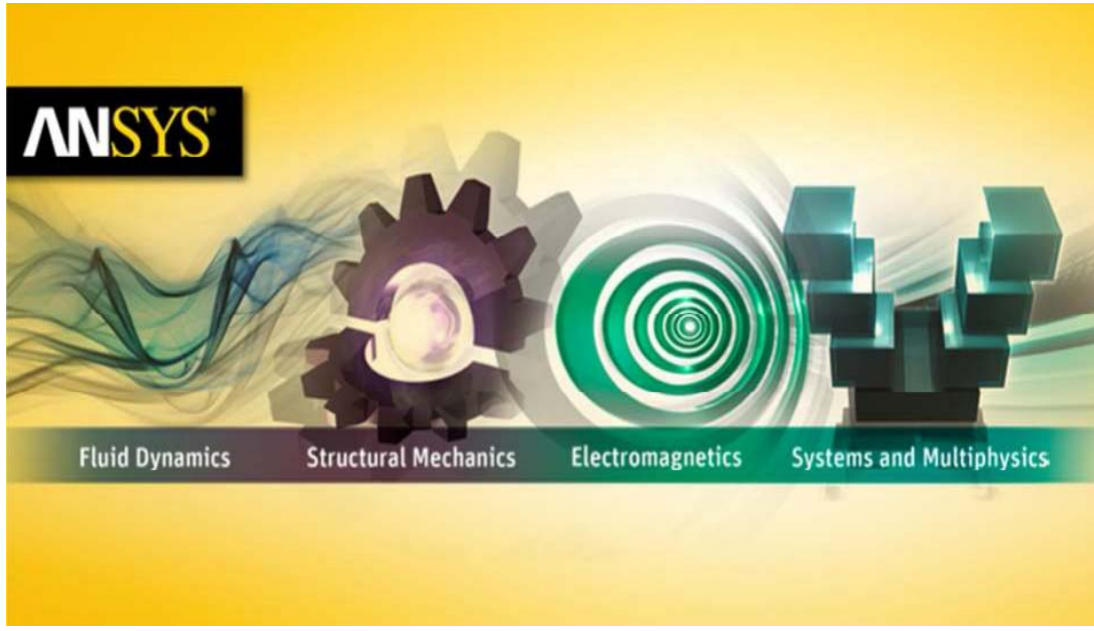


ANSYS Electronics Desktop™: Scripting Guide

ANSYS, Inc.
Southpointe
2600 ANSYS Drive
Canonsburg, PA 15317
ansysinfo@ansys.com
http://www.ansys.com
(T) 724-746-3304
(F) 724-514-9494

June 2015
ANSYS Electromagnetics Suite 16.2

ANSYS, Inc. is certified to ISO 9001:2008.

Copyright and Trademark Information

Disclaimer Notice

U.S. Government Rights

Third-Party Software

Published in the U.S.A.

Table of Contents

3. Ansoft Application Object Script Commands

4. Desktop Object Script Commands

5. Project Object Script Commands

6. Material Script Commands

7. Property Script Commands

12. Output Variable Script Commands

13. Reporter Editor Script Commands

16. Analysis Setup Module Script Commands

17. Optimetrics Module Script Commands

18. Solutions Module Script Commands

26. Layout Scripting

29. Definition Editor Script Commands

1 Introduction to VBScript

ANSYS Electronics Desktop uses the Microsoft® Visual Basic® Scripting Edition (VBScript) scripting language to record macros. VBScript is based on the Microsoft Visual Basic programming language.

Using scripts is a fast, effective way to accomplish tasks you want to repeat. When you execute a script, the commands in the script are performed.

You can write a script using any text editor or you can record a script from within the ANSYS Electronics Desktop interface. After recording the script from within ANSYS Electronics Desktop, you can then modify it if necessary using a text editor.

Although ANSYS Electronics Desktop records scripts in VBScript format, it can also execute scripts in JavaScript™ format. If you are running a script from a command prompt, the script can be written in any language that provides the Microsoft COM methods. The ANSYS Electronics Desktop scripting documentation refers to VBScript format only.

This chapter provides an overview of key VBScript components.

Converting Between Data Types

Including Scripts

Aborting Scripts

Interacting with a Script

Recommended VBScript References

For more details about VBScript, please see the *Recommended VBScript References* section at the end of this chapter.

A Sample HFSS Script

Following is an example of an ANSYS Electronics Desktop script. It includes comment lines, which are preceded by either an apostrophe (') or the word REM, that offer explanations for each preceding line or lines. VBScript keywords appear in bold font.

```
'----------------------------------------------
'Script Recorded by Ansoft HFSS Version 10.0
'11:03 AM May 3, 2005
'----------------------------------------------

Dim oDesign
Dim oEditor
Dim oModule
REM Dim is used to declare variables. Dim means dimension. In VBScript you can use Dim,
REM Public, or Private to declare variables. As VBScript has no built-in data types (like
REM integer, string, etc.), all variables are treated as variants, which can store any type of
REM information. In this example, the three variables will be used as objects. When
REM recording scripts in HFSS, variants that will be used as objects always begin with o.

Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
' You can use Set to assign an object reference to a variable. A copy of the object is not
' created for that variable. Here CreateObject is a function that takes a string as input
' and returns an object. The object is assigned to the variable oAnsoftApp.

Set oDesktop = oAnsoftApp.GetAppDesktop()
' GetAppDesktop is a function of oAnsoftApp. This function does not take an input and it
' returns an object. The object is assigned to the variable oDesktop.

oDesktop.NewProject
' In VBScript, a Sub procedure is a procedure that is called by name, can receive arguments,
' and can perform a specific task with a group of statements. Here the Sub procedure
' NewProject of the object oDesktop is called. This Sub does not take an input.

Set oProject = oDesktop.GetActiveProject
oProject.InsertDesign "Hfss", "HFSSDesign1", "DrivenModal", ""
```

```
' In a Sub or Function procedure call, you can group the input parameters inside
' parentheses or without parentheses. Here the four strings are the input parameters of
' the Sub procedure InsertDesign of the object oProject.

Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.CreateBox Array("NAME:BoxParameters", "XPosition:=", _
    "0mm", "YPosition:=", "0mm", "ZPosition:=", "0mm", _
    "XSize:=", "1.6mm", "YSize:=", "1.2mm", "ZSize:=", _
    "0.8mm"), Array("NAME:Attributes", "Name:=", "Box1", "Flags:=", _
    "", "Color:=", "(132 132 193)", "Transparency:=", _
    0.400000005960464, "PartCoordinateSystem:=", _
    "Global", "MaterialName:=", "vacuum", "SolveInside:=", true)
' oEditor.CreateBox is a Sub procedure that takes two array variables as input. The
' first array is for the box's geometric parameters and the second array is for the box's
' attributes. You can modify the italicized entries to create a different box. In VBScript,
' Array is a function that returns a variant containing an array. The underscore
' character ( _ ) here indicates that the statement continues to the next line. The
' underscore character must be placed outside of string constants, or else VBScript will
' recognize the character as part of the string constant rather than an indication that the
' string continues on the next line. Following is an example of proper use of the underscore
' character:
'Msgbox("Please include units when creating variables " & _
'"that require dimensions."
' Following is an example of improper use of the underscore character:
'Msgbox("Please include units when creating variables _
'that require dimensions."
```

For additional ANSYS Electronics Desktop script examples, see *Example Scripts*.

A Sample Designer Script

Following is an example of a Designer script. It includes comment lines, which are preceded by either an apostrophe (') or the word REM, that offer explanations for each preceding line or lines. VBScript keywords appear in bold font.

```
' ------------------------------------------------
```

```
' Script Recorded by Designer/Nexxim
' 8:10 AM  Dec 05, 2003
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule

REM Dim is used to declare variables.  Dim means dimension.  In
VBScript you can use Dim,
REM Public, or Private to declare variables. As VBScript has no built-
in data types (like,
REM integer, string, etc.) all variables are treated as variants,
which can store any type of
REM information. In this example, the variables will be used as
objects. When recording
REM scripts in Designer, variants that will be used as objects always
begin with
REM the letter o.

Set oAnsoftApp = CreateObject("AnsysDesigner.DesignerScript")
```

You can use **Set** to assign an object reference to a variable. A copy of the object is ‘not created for that variable. Here, CreateObject is a function that takes a string ‘as input and returns an object. The object is assigned to the variable oAnsoftApp.

```
Set oDesktop = oAnsoftApp.GetAppDesktop()
‘GetAppDesktop is a function of oAnsoftApp.  This function does not
take an input
‘but returns an object that is then assigned to the variable oDesktop.
oDesktop.RestoreWindow
oDesktop.NewProject
‘In VBScript a Sub procedure is a procedure that is called by name,
can optionally
‘receive arguments, and can perform a specific task with a group of
statements.
```

```
'Here the Sub procedure NewProject of the object oDesktop is called.
'This Sub does not take an input.
Set oProject = oDesktop.GetActiveProject
oProject.InsertDesign "Planar EM", "PlanarEM1",  _
"C:\Program Files\AnsysEM\Designer\<platform>\syslib\PCB - Single-
Sided.asty", ""

'In a Sub or Function procedure call, you can group the input parame-
ters inside
'parentheses or without parentheses.  Here the four strings are input
parameters
'of the Sub procedure InsertDesign of the object oProject.

Set oDesign = oProject.SetActiveDesign("PlanarEM1")
Set oEditor = oDesign.SetActiveEditor("Layout")
oEditor.CreateRectangle Array("NAME:Contents", "rectGeometry:=",
Array("Name:=",  _
  "rect_1", "LayerName:=", "Top", "lw:=", "0mm", "Ax:=", "-22mm",
"Ay:=", "20mm", "Bx:=",  _
"29mm", "By:=", "-4mm", "ang:=", "0deg"))

'oEditor.CreateRectangle is a Sub procedure that takes an array vari-
able
'as an argument.  This array variable contains two string variables
and an array
'containing the geometric parameters of the rectangle.  You can use
variables to
'modify the geometric parameters of the rectangle.

'In VBScript Array is a function that returns a variant containing an
array.  The
'underscore character (_) here indicates that the statement continues
on the next line.
'The underscore character must be placed outside of string constants,
or else VBScript
'will recognize the character as part of the string constant rather
than an indication that the
'string continues on the next line.
```

```
'Following in an example of proper use of the underscore character:
'  MsgBox("Please include units when creating variables" & _
'  "that require dimensions.")

'Following is an example of improper use of the underscore character:
'  MsgBox("Please include units when creating variables _
   '  that require dimensions.")

   oProject.SaveAs "C:\Program
   Files\AnsysEM\Designer\<platform>\Examples\test.adsn", true
```

A Sample Q3D Extractor Script

Following is an example of script. It includes comment lines, which are preceded by either an apostrophe (') or the word REM, that offer explanations for each preceding line or lines. VBScript keywords appear in bold font.

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
REM Dim is used to declare variables and means dimension. In VBScript you can use Dim,
REM Public, or Private to declare variables. As VBScript has no built-in data types (like
REM integer, string, etc.), all variables are treated as variants, which can store any type of
REM information. In this example, the three variables will be used as objects. When
REM recording scripts in Q3D Extractor, variants that will be used as objects always begin
with o.

Set oAnsoftApp = CreateObject("Q3DExtractor.ScriptInterface")
' You can use Set to assign an object reference to a variable. A copy of the object is not
' created for that variable. Here CreateObject is a function that takes a string as input
' and returns an object. The object is assigned to the variable oAnsoftApp.

Set oDesktop = oAnsoftApp.GetAppDesktop()
```

```
' GetAppDesktop is a function of oAnsoftApp. This function does not take an input and it
' returns an object. The object is assigned to the variable oDesktop.

oDesktop.NewProject
' In VBScript, a Sub procedure is a procedure that is called by name, can receive arguments,
' and can perform a specific task with a group of statements. Here the Sub procedure
' NewProject of the object oDesktop is called. This Sub does not take an input.

Set oProject = oDesktop.GetActiveProject
oProject.InsertDesign "Q3D Extractor", "Q3DDesign1", "", ""
' In a Sub or Function procedure call, you can group the input parameters inside
' parentheses or without parentheses. Here the four strings are the input parameters of
' the Sub procedure InsertDesign of the object oProject.

Set oDesign = oProject.SetActiveDesign("Q3DDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.CreateBox Array("NAME:BoxParameters", "XPosition:=", _
    "0mm", "YPosition:=", "0mm", "ZPosition:=", "0mm", _
    "XSize:=", "1.6mm", "YSize:=", "1.2mm", "ZSize:=", _
    "0.8mm"), Array("NAME:Attributes", "Name:=", "Box1", "Flags:=", _
    "", "Color:=", "(132 132 193)", "Transparency:=", _
    0.400000005960464, "PartCoordinateSystem:=", _
    "Global", "MaterialName:=", "vacuum", "SolveInside:=", true)
' oEditor.CreateBox is a Sub procedure that takes two array variables as input. The
' first array is for the box's geometric parameters, and the second array is for the box's
' attributes. You can modify the italicized entries to create a different box. In VBScript,
' Array is a function that returns a variant containing an array. The underscore
' character ( _ ) here indicates that the statement continues to the next line. The
' underscore character must be placed outside of string constants, or else VBScript
' recognizes the character as part of the string constant rather than an indication that the
' string continues on the next line. Following is an example of proper use of the underscore
' character:
'Msgbox("Please include units when creating variables " & _
'"that require dimensions."
' Following is an example of improper use of the underscore character:
```

```
'Msgbox("Please include units when creating variables _
'that require dimensions."
```

For additional ANSYS Electronics Desktop script examples, see *Example Scripts*.

Simple and Composite Names

Components, symbols, footprints, models, and padstacks possess either “simple” names or “composite” names. Composite names are used to distinguish items from libraries that may possess the same simple name. A composite name is created by combining an item’s library name with its simple name. Composite names for definitions are unique, but simple names are not.

- Composite names are used by definition manager script commands to uniquely identify script definitions.
- Materials and scripts do not have composite names, so project definitions for these items must possess a unique simple name.
- The format of a composite name is LibraryName:SimpleDefinitionName. For example, the composite name for the component “CAP_ in” the system library Nexxim Circuit Elements\Capacitors is "Nexxim Circuit Elements\Capacitors:CAP_in."
- The format of a composite name in a project is OriginLibraryName:SimpleDefinitionName. For example, the composite name for the project component “CAP_” that was originally from the system library Nexxim Circuit Elements\Capacitors is "Nexxim Circuit Elements\Capacitors:CAP_".
- Not all definitions in a project have a library of origin. Newly added definitions do not have a library of origin, and project definitions whose names are changed do not have a library of origin (even if they did before the name change). As a result, the composite name for items without a library of origin is the item’s simple name itself. For example, the composite name for the project component “CAP_” that came from a system library and was renamed to “MyCAP_” is “MyCAP_”.

To construct a composite name, select **Tools > Edit Configured Libraries > Components** to open the **Edit Libraries** dialog. The subnames used to construct a composite name can be found in the **Name** and **Origin** columns that correspond to a particular component. The **Origin** column contains the library portion of the composite name, while the **Name** column contains the simple portion of the composite name.

VBScript Variables

A VBScript variable is a placeholder representing information that may change during the time your script is running. Variables are useful because they let you assign a short and easy to remember name to each piece of data you plan to use. Use a variable name in a script to view or modify its value.

Declaring Variables

To declare variables explicitly in a script, use the `Dim`, `Public`, or `Private` statements. For example:

```
Dim box_xsize
```

After declaring a variable, you can assign information to it. For example:

```
box_xsize = "3mm"
```

You can declare multiple variables by separating each variable name with a comma. For example:

```
Dim Top, Bottom, Left, Right
```

You can also declare a variable implicitly by simply using its name in your script. Doing so is not generally a good practice because you could misspell the variable name in one or more places, causing unexpected results when your script is run. For that reason, the **Option Explicit** statement is available to require explicit declaration of all variables. The **Option Explicit** statement should be the first statement in your script.

Variable Naming Conventions

You should use names that are short but intuitive and easy to remember. Use the following conventions for naming variables in VBScript:

- Begin with an alphabetic character.
- Cannot contain an embedded period.
- Must not exceed 255 characters.
- Must be unique in the scope in which it is declared.
- Do not use VBScript keywords.

Scope and Lifetime of Variables

Variables at the script level are available to all procedures within the script. At the procedure level, variables are available only within the procedure. It has local scope and is a procedure-level variable.

The lifetime of a variable depends on how long it exists. The script-level variables exist from declaration until the end of the script. A procedure-level variable exists only as long as you are in the procedure and is destroyed when the procedure exits.

Array Variables

Create an array variable when you want to assign more than one related value to a single variable. An array variable contains a series of values. For example:

```
Dim Primitives(2)
```

All arrays in VBScript are zero-based, so the array above actually contains 3 elements. You assign data to each of the array's elements using an index into the array. Data can be assigned to the elements of an array as follows:

```
Primitives(0) = "Box1"
Primitives(1) = "Cone1"
Primitives(2) = "Cylinder1"
```

Similarly, the data can be retrieved from any element using an index into a particular array element. For example:

```
one_prim = Primitives(1)
```

You can also use the `Array` function to assign an array of elements to a variable. For example:

```
Dim Primitives
Primitives = Array ("Box1", "cone1", "Cylinder1")
```

Note When using the `Array` function, do not use parentheses on the variable when it is declared. For example, use `Dim myarray`, not `Dim myarray()`.

If you do not know the size of the array at declaration or the size changes during the time your script is running, you can use dynamic arrays. They are declared without size or number of dimensions inside the parentheses. For example:

```
Dim FirstArray()
ReDim SecondArray()
```

To use a dynamic array, you must subsequently use **ReDim** to determine the number of dimensions and the size of each dimension. You can also use the **Preserve** keyword to preserve the contents of the array as the resizing takes place.

```
ReDim FirstArray(25)
ReDim Preserve FirstArray(30)
```

VBScript Operators

VBScript provides operators, which are grouped into these categories: arithmetic operators, comparison operators, and logical operators.

Please see the online *VBScript User's Guide* for more details.

Operator Precedence

When several operations occur in an expression, each part is evaluated and resolved in a predetermined order, called operator precedence. You can use parentheses to override the order of precedence and force some parts of an expression to be evaluated before others. Operations within parentheses are always performed before those outside the parentheses. Within parentheses, however, standard operator precedence is maintained.

When expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators are evaluated next, and logical operators are evaulated last. Comparison operators all have equal precedence, that is, they are evaluated in the left-to-right order in which they appear. Arithmetic and logical operators are evaluated in the following order of precedence.

Arithmetic Operators

Following is a list of VBScript's arithmetic operators.

Symbol	Description
^	Exponentiation
-	Unary negation
*	Multiplication
/	Division
\	Integer division
Mod	Modulus arithmetic
+	Addition
-	Subtraction
&	String concatenation

Comparison Operators

Following is a list of VBScript's comparison operators.

Symbol	Description
=	Equality
<>	Inequality
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
Is	Object equivalence

Logical Operators

Following is a list of VBScript's logical operators:

Symbol	Description
Not	Logical negation
And	Logical conjunction
Or	Logical disjunction
Xor	Logical exclusion
Eqv	Logical equivalence
Imp	Logical implication

Controlling Program Execution

You can use conditional statements to control the flow of a script. There are two types of conditional statements in VBScript:

- `If...Then...Else`
- `Select Case`

Using If...Then...Else

Following is an example that demonstrates the `If...Then...Else` conditional statement:

```
If obj = "Box1" Then
    <statements to execute>
ElseIf obj = "Cylinder1" Then
```

```
    <statements to execute>
Else
    <statements to execute>
End If
```

Using Select Case

Following is an example that demonstrates the `Select Case` conditional statement:

```
Select Case primitive_name
    Case "Box1"
      <statements to execute>
    Case "Cylinder1"
      <statements to execute>
    Case Else
      <statements to execute>
End Select
```

Looping Through Code

Looping allows you to run a group of statements repeatedly. There are two types of loops:

- For…Next: Uses a counter to run statements a specified number of times.
- Do…Loop: Loops while or until a condition is True.

When using conditional statements that test for zero voltage/current, it is important to note that a real voltage or current should not be trusted to be exactly zero, even when it should be. Typically, the voltage or current is often on the order of 'epsilon' (1e-16) or smaller; hence, it is nonzero in value.

Using a For...Next Loop

The `For...Next` type of loop allows you to run a group of statements repeatedly. It uses a counter to run statements a specified number of times. Following is an example that demonstrates the `For...Next` loop:

```
For variable = start To end
    <statements to execute>
Next
```

You can exit early from a `For...Next loop` with the `Exit For` statement.

Using a Do Loop

You can use **Do...Loop** statements to run a block of statements until (or while) a condition is true.

Repeating Statements While a Condition is True

Use the While keyword to check a condition in a Do...Loop statement. The syntax is as follows:

```
Do While condition
    <statements to execute>
Loop
```

Repeating a Statement Until a Condition Becomes True

Following is the syntax:

```
Do Until condition
    <statements to execute>
Loop
```

You can exit early from a loop by using the `Exit For` statement.

VBScript Procedures

In VBScript, there are two kinds of procedures, Sub and Function. These procedures are called by name, they can receive arguments, and each performs a specific task with a group of VBScript statements. If there is no argument, then the Sub or Function statement must include an empty set of parentheses.

Function Procedures

A Function returns a value by assigning a value to its name in one or more statements. Following is the syntax of a Function:

```
Function FunctionName([arguments])
    <Function statements>
End Function
```

Sub Procedures

A Sub procedure is like a function procedure, except that it does not return a value through its name. Following is the syntax of a Sub:

```
Sub ProcedureName([arguments])
    <Procedure statements>
    End Sub
```

Converting Between Data Types

To convert data from one subtype to another, use the following VBScript functions:

CStr Syntax: `CStr(variablename)`.

Converts `variablename` to a string. For example, it can be used to convert the number 2.5 to the string `"2.5"`.

CBool Syntax: `CBool(variablename)`.

Converts `variablename` to a boolean. If `variablename` is `0` or `"0"`, `CBool` returns `False`. Otherwise it returns `True`.

CDbl Syntax: `CDbl(variablename)`.

Converts `variablename` to a double precision number. For example, it can be used to convert the string `"2.5"` to the number 2.5.

CInt Syntax: `CInt(variablename)`.

Converts `variablename` to an integer.

Including Scripts

HFSS allows you to include one script within another using the following command:

```
#include "<scriptfilename>"
```

Where scriptfilename is the full path name to a file that contains script text, or is the name of a script in the project library or script library (listed in the project window under the Definitions/Scripts directory).

The command works for VBScript, JScript, and for the following:

- Scripts in the project library that are run by right-clicking the script icon in the project window and choosing "Run Script"
- Scripts in files that are external to HFSS and are run by choosing "Tools/Run Script..."
- Scripts that are specified as callbacks in the Property dialog
- Scripts that are run to draw parameterized footprints in layout

An include command can be placed anywhere in a script, but for readability it is recommended that commands be placed at the beginning of a file. The same script can be included multiple times without error, and circular inclusions will be ignored.

Aborting Scripts

You can abort a script that is running in the desktop simply by pressing the ESC key. Terminating a script in this manner works for each of the following:

- Scripts in the project library that are run by right-clicking the script icon in the project win-

dow and choosing "Run Script"

- Scripts in files that are external to HFSS and are run by choosing “Tools/Run Script...”
- Scripts that are specified as callbacks in the Property dialog
- Scripts that are run to draw parameterized footprints in layout

Interacting with a Script

VBScript provides two functions that enable you to interact with a script while it is running: the `InputBox` function and the `MsgBox` function.

The `InputBox` function displays a dialog box with an input field. The value that is typed into the input field is returned. For example:

```
Dim users_string
users_string = InputBox ("text prompt", "title of the pop-up dialog _
    box", "default text for the input box")
```

The last two arguments to the function are optional.

The `MsgBox` function shows a message and returns a number based on the button the user presses. For example:

```
MsgBox ("message text")
```

Recommended VBScript References

Microsoft Corporation. *VBScript User’s Guide.*

Available http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/vbstutor.asp.

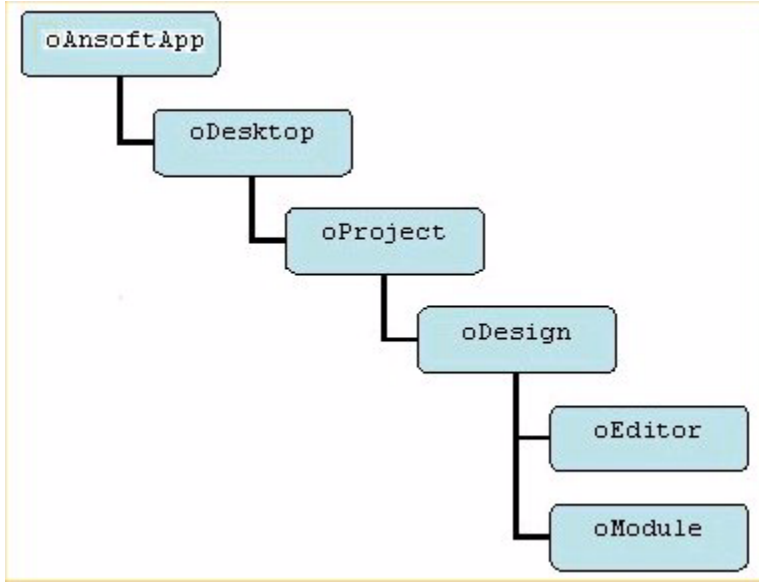
Childs, M., Lomax, P., and Petrusha, R. *VBScript in a Nutshell: A Desktop Quick Reference*. May 2002. O’Reilly & Associates. ISBN: 1-56592-720-6.

2 ANSYS Electronics Desktop and VBScript

This chapter provides an overview of ANSYS Electronics Desktop scripting using VBScript. Information is included on the following topics:

Overview of ANSYS Electronics Desktop Script Variables

When you record an ANSYS Electronics Desktop script, the beginning of the script has some standard commands as shown in the following chart. The commands in the chart are meant to define the variables used by ANSYS Electronics Desktop in the script and assign values to the variables. The variables are used in the following hierarchy.



First the commands are described followed by examples.

oAnsoftApp

The `oAnsoftApp` object provides a handle for VBScript to access the `AnsoftHfss` product.

One example of accessing this object is:

```
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
```

oDesktop

The `oDesktop` object is used to perform desktop-level operations, including project management.

One example of accessing this object is:

```
Set oDesktop = oAnsoftApp.GetAppDesktop()
```

See the chapter *Desktop Object Script Commands*, for details about script commands recognized by the `oDesktop` object.

oProject

The `oProject` object corresponds to one project open in the product. It is used to manipulate the project and its data. Its data includes variables, material definitions and one or more designs. One example of accessing this object is:

```
Set oProject = oDesktop.GetActiveProject()
```

See the following chapters for details about the script commands recognized by the `oProject` object:

- *Project Object Script Commands*
- *Material Script Commands*
- *Property Script Commands*
- *Dataset Script Commands*

oDesign

The `oDesign` object corresponds to an instance of a design in the project. This object is used to manipulate the design and its data. Its data includes variables, modules, and editors.

One example of accessing this object is:

```
Set oDesign = oProject.GetActiveDesign()
```

See the following chapters for details about the script commands recognized by the `oDesign` object:

- *Design Object Script Commands*
- *Output Variable Script Commands*
- *Reporter Editor Script Commands*

oEditor

The `oEditor` object corresponds to an editor, such as the 3D Modeler, layout or schematic editors. This object is used to add and modify data in the editor.

One example of accessing this object is:

```
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
```

An HFSS layout example of accessing this object is:

```
Set oEditor = oDesign.SetActiveEditor("Layout")
```

The AnsoftHfss product scripting supports the following editors:

Editor	Name in Script
3D Modeler Editor	"3D Modeler"
Reporter Editor	There is no Reporter editor object in the script. Instead, Reporter editor commands are executed by the ANSYS Electronics Desktop design object oDesign.

See the chapter *3D Modeler Editor Script Commands*, for details about the script commands recognized by the oEditor object and the chapter *Reporter Editor Script Commands* for details about Reporter editor commands.

oModule

The oModule object corresponds to a module in the design. Modules are used to handle a set of related functionality.

One example of accessing this object is:

```
Set oModule = oDesign.GetModule("BoundarySetup")
```

The software scripting supports the following modules:

Module	Name in Script	Chapter Title
Boundary/Excitations/Nets Module Corresponds to the **Boundaries, Excitations or Nets** branches in the project tree.	"BoundarySetup"	*Boundary and Excitation Module Script Commands*
Mesh Operations Module Corresponds to the **Mesh Operations** branch in the project tree.	"MeshSetup"	*Mesh Operations Module Script Commands*
Analysis Module Corresponds to the **Analysis** branch in the project tree.	"AnalysisSetup"	*Analysis Module Script Commands*

Optimetrics Module Corresponds to the **Optimetrics** branch in the project tree.	`"Optimetrics"`	*Optimetrics Script Commands*
Solutions Module Corresponds to the operations in the **Solution Data** dialog box, which is accessed by clicking **HFSS>Results> Solution Data**.	`"Solutions"`	*Solutions Module Script Commands*
Field Overlays Module Corresponds to the **Field Overlays** branch in the project tree.	`"FieldsReporter"`	*Field Overlays Module Script Commands*
Radiation Module Corresponds to the **Radiation** branch in the project tree.	`"RadField"`	*Radiation Module Script Commands*
Reduce Matrix Module Corresponds to the **Reduce Matrix** branch in the project tree.	`"RaduceMatrix"`	*Reduce Matrices Module Script Commands*

Examples of HFSS, Layout Editor, and Q3D Extractor scripts are described as follows.

Example: HFSS Script

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
```

```
Set oProject = oDesktop.SetActiveProject("Project1")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
Set oModule = oDesign.GetModule("BoundarySetup")
```

Example: HFSS Layout Editor Script

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsysDesigner.DesignerScript")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project1")
oProject.InsertDesign "Planar EM", "PlanarEM1",  _
     "C:\testinstall\HFSS\syslib\PCB - SingleSided.asty", ""
Set oDesign = oProject.SetActiveDesign("PlanarEM1")
Set oEditor = oDesign.SetActiveEditor("Layout")
```

Example: Q3D Extractor Script

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Q3DExtractor.ScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
Set oProject = oDesktop.SetActiveProject("")
Set oDesign = oProject.SetActiveDesign("")
Set oModule = oDesign.GetModule("Solutions")
```

The lines above define the variables used by the script and assigns values to the variables.

Recording a Script

After you begin to record a script, each subsequent interface action you take is recorded and saved to a text file or project folder you have specified — i.e., each interface command is comprised of one or more associated script commands, and each action is recorded and saved to a single script. The recorded script is saved in VBScript format (.vbs).

To Record a Script to a File:

1. On the **Tools** menu, click **Record Script to File**.
   The **Save As** dialog box appears.
2. Use the file browser to locate and double-click the folder where you wish to save the script, such as C:\Program Files\AnsysEM\HFSS\<platform>\Scripts.
3. Type the name of the script in the **File name** text box and then click **Save**.
4. Perform the steps you want to record.
5. When you have finished recording the script, click **Tools > Stop Script Recording**.
   The recorded script is then saved to *filename.vbs* in the folder specified folder.

To Record a Script to a Project:

1. On the **Tools** menu, click **Record Script to Project**.
   The **Save Script to Project** dialog box appears.
2. Type the name of the script in the text box and then click **OK**.
   Perform the steps you want to record.
3. When you have finished, click **Tools > Stop Script Recording**.
   The recorded script is then saved to ***scriptname.vbs*** in the Scripts library and can be accessed by expanding the definitions/Scripts folder in the Project tree.

Stopping Script Recording

- Click**Tools>Stop Script Recording**.
  ANSYS Electronics Desktop stops recording to the script.

Running a Script

1. Click **Tool>Run Script**.
   The **Open** dialog box appears.
2. Use the file browser to locate the folder in which you saved the script, and then double-click the folder's name.
3. Type the name of the script in the **File name** text box, or click its name, and then click **Open**.
   ANSYS Electronics Desktop executes the script.

To supply script arguments when running from **Tools>Run Script**, use the **Edit** field at the bottom of the file selection dialog. You can access the script arguments using the **AnsoftScriptHost.arguments** collection from vbscript. This is a standard COM collection.

To run a script from a command line (as described in the ANSYS Electronics Desktop Online Help in the Running ANSYS Electronics Desktop from a Command Line section), use:

-runscriptandexit or **-runscript** arguments to the ANSYS Electronics Desktop command line syntax.

You can give **-scriptargs** parameter to the script and specify the arguments described in the ANSYS Electronics Desktop online help.

If you run the script from DOS prompt as a .vbs file (that is, you do not launch ANSYS Electronics Desktop, but just launch vbs directly, or use wscript.exe or cscript.exe), the arguments will be in the WSH.arguments collection, not the AnsoftScriptHost.arguments collection. To handle this, use the following script:

```
on error resume next
dim args
Set args = AnsoftScript.arguments
if(IsEmpty(args)) then
Set args = WSH.arguments
End if
on error goto 0
'At this point, args has the arguments no matter if you are
running
'under windows script host or Ansoft script host
msgbox "Count is " & args.Count
for i = 0 to args.Count - 1
  msgbox args(i)
next
```

Pausing and Resuming a Script

To pause a script during its execution:

- Click **Tools>Pause Script**.

To resume a script after pausing it:

- Click **Tools>Resume Script**.

Stopping a Script

- On the **Tools** menu, click **Stop Script**.
  ANSYS Electronics Desktop stops executing the script that has been paused.

Modifying a Script for Easier Playback

In the sample scripts, note that the `oProject` variable is set to `"Project1"`. That means that the script must be played back within Project1 to operate correctly. Alternatively, `oProject` could be set to the active project without specifying a project name.

For example:

```
Set oProject = oDesktop.GetActiveProject()
```

Using the line above, the script can be played back in any project.

The modified sample script is as follows:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
Set oProject = oDesktop.SetActiveProject()
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
Set oModule = oDesign.GetModule("BoundarySetup")
```

ANSYS Electronics Desktop Scripting Conventions

A number of conventions exist for ANSYS Electronics Desktop regarding syntax, arguments and numerical values. These conventions are as follows:

- Syntax conventions
- Script command conventions
- Named arguments
- Setting numerical values

Syntax Conventions

The following data types will be used throughout this scripting guide:

<string>	A quoted string.
<bool>	A boolean value. Should be set to either `true` or `false` (no quotes). Example 1: `"SolveInside:=", true` Example 2: `"PortSolver:=", true`
<double>	A double precision value. Example: `1.2`
<int>	An integer. Example: `1`

<value>	Can be a number, a VBScript variable, or a quoted string containing a valid ANSYS Electronics Desktop expression.
	Examples:
	`-"XSize:=", 1`
	`-"XSize:=", "3mm"`
	`-"XSize:=", VBScript_Var`
	`-"XSize:=", "Hfss_Var + 10mm"`

Script Command Conventions

The majority of this guide lists individual script commands. The following conventions are used to describe them:

Script Command Name

Use:	Describes the function of the script command.
Command:	Lists the interface command that corresponds to the script command. Menu commands are separated by carats. For example, **HFSS>Excitations>Assign>Wave Port**.
Syntax:	`Demonstrates the correct syntax for the command. Carat brackets < > enclose information or arguments that you must enter.`
Return Value:	Describes the return value, if any.
Parameters:	`Describes the arguments or information in the syntax description, if an explanation is needed.`
Example:	`Provides a working example of the script command, if needed.`

Passing Arguments to Scripts

There are two ways to pass arguments to scripts:

1. When running from command line using **-runscriptandexit** or **-runscript**, give **-scriptargs** parameters and specify arguments.

   All arguments following the **-scriptargs** flag are enclosed in double quotes. For example

   ```
   C:\AnsysEM\AnsysEMn\Win64\hfss.exe -RunScriptAndExit _
   C:\scripts\test.vbs -scriptsargs "arg1 arg2 arg3"
   ```

2. When running from **Tools>Run script**, there is an edit field at the bottom of the file selection dialog that you can use to enter script arguments.

You can access the script arguments using the **AnsoftScriptHost.arguments** collection from vbscript. This is a standard COM collection.

If you run the script from DOS prompt as a .vbs file (that is, you do not open ANSYS Electronics Desktop, but launch vbs directly, or use **wscript.exe** or **cscript.exe**), the arguments are in the **WSH.arguments** collection, not the **AnsoftScriptHost.arguments** collection. To handle this, write this script:

```
on error resume next
dim args
Set args = AnsoftScript.arguments
if(IsEmpty(args)) then
Set args = WSH.arguments
End if
on error goto 0

'At this point, args has the arguments no matter if you are running
'under windows script host or Ansoft script host
msgbox "Count is " & args.Count
for i = 0 to args.Count - 1
msgbox args(i)
next
```

Named Arguments

Many ANSYS Electronics Desktop script commands use named arguments. The names can appear in three ways:

1. Named data, name precedes data.

For example: `...,"SolveInside:=", true, ...`

2. Named Array, name precedes array.
For example: `...,"Attributes:=", Array(...),...`

3. Named Array, name inside array.
For example: `..., Array("NAME:Attributes",...),...`

In the first and second examples, the name is formatted as `"<Name>:="`. This signals ANSYS Electronics Desktop that this is a name for the next argument in the script command. In the third example, the name is formatted as `"NAME:<name>"` and is the first element of the Array. The names are used both to identify what the data means to you and to inform ANSYS Electronics Desktop which data is being given. The names must be included or the script will not play back correctly. However, if you are writing a script, you do not need to pass in every piece of data that the command can take. For example, if you are modifying a boundary, the script will be recorded to include every piece of data needed for the boundary, whether or not it was modified. If you are writing a script by hand, you can just add the data that changed and omit anything that you do not want to change. ANSYS Electronics Desktop will use the names to determine which data you provided.

For example, when editing an impedance boundary, ANSYS Electronics Desktop records the 'edit impedance boundary' command as follows:

```
oModule.EditImpedance "Imped1", Array("NAME:Imped1", _
    "Resistance:=", "100", "Reactance:=", "50", _
    "InfGroundPlane:=", false)
```

If you only want to change the resistance, then you can leave out the other data arguments when you are manually writing a script:

```
oModule.EditImpedance "Imped1", Array("NAME:Imped1", _
    "Resistance:=", "100")
```

Another example corresponding to ANSYS Electronics Desktop layout is described below:
When editing a port excitation, ANSYS Electronics Desktop records the 'edit port' command as follows:

```
oModule.Edit "Port1", Array("NAME:Port1",Array("NAME:Properties",
"PortSolver:=", "true", "Phase:=", "0deg", "Magnitude:=", "2mA",
"Impedance:=", "50Ohm", "Theta:=", "0deg", "Phi:=", "0deg", "PostPro-
cess:=", "false", "Renormalize:=", "50Ohm + 0i Ohm", "Deembed:=",
"0mm", "RefToGround:=", "false"), "Type:=", "EdgePort", "IsGap-
```

```
Source:=", true, "UpperProbe:=", false, "LayoutObject:=", "Port1",
"Pin:=", "", "ReferencePort:=", "")
```

If you only want to change the magnitude, you can leave out the other data arguments when you are manually writing a script:

```
oModule.Edit "Port1", Array("NAME:Port1", Array ("Magnitude:=",
"1mA"))
```

Setting Numerical Values

For script arguments that expect a number, the following options are possible:

- Pass in the number directly. For example:

```
oModule.EditVoltage "Voltage1", Array("NAME:Voltage1", _
    "Voltage:=", 3.5)
```

- Pass in a string containing the number with units. For example:

```
oModule.EditVoltage "Voltage1", Array("NAME:Voltage1", _
    "Voltage:=", "3.5V" )
```

- Pass in an ANSYS Electronics Desktop/Layout/Q3D Extractor defined variable name. For example:

```
oModule.EditVoltage "Voltage1", Array("NAME:Voltage1", _
    "Voltage:=", "$var1" )
```

- Pass in a VBScript variable. For example:

```
vb_var = "3.5V"
oModule.EditVoltage "Voltage1", Array("NAME:Voltage1", _
     "Voltage:=", vb_var)
```

ANSYS Electronics Desktop Layout Scripts and the Active Layer

The active layer is the layer that is used for object creation and placement during adding operations in the user interface. Adding operations include paste and placement of instances, as well as object creation. Usually there will be an active layer, but it is not required and can not be assumed. Adding operations are responsible for ensuring that the active layer exists and meets the particular requirements (such as layer type) for the operation. Adding operations may change the active layer to a different layer that meets requirements. The user is notified if the active layer is changed. If no layer is available to be active, the operation is not done.

The active layer is not used during script adding operations. Script adding operations are responsible for ensuring that the specified layer exists and meets the particular requirements (such as layer type) for the operation. If there is a problem with using the specified layer, the operation is not done. The active layer is always visible and selectable. These attributes are reset, if needed, when a layer is made active. The current active layer is indicated by a combo box display in the toolbar. The list for the combo box contains all layers that may be set active.

The active text style is related to the active layer. If there is no active layer, there is no active text style. Objects on the active layer have priority during snapping.

Scripts and Locked Layers

The locked attribute of a layer is defined to mean that you may not edit, delete, or add objects on the layer, either directly or with scripts (i.e., scripts run on layout or footprint definitions). This includes not being able to change properties of objects on the layer. Note, however, that parameter changes can alter objects on locked layers.

The locked attribute of a layer is configurable using script commands and is user-editable via the **Edit Layers Dialog** in the HFSS Layout Editor.

Event Callback Scripting

Event Callback scripting allows you to define custom JavaScript and VBScript routines that will run automatically after a triggering event is detected, events such as placing a component or running a simulation. When you define an Event Callback script, you specify one or more scripts that will be run after a particular event is detected. For more information see Event Callbacks in the online help.

A callback script can only access functions and other scripts defined by its callback definition. For example, a callback script can call PropHost.GetValue — and all other PropHost functions — but only from scripts defined in the Property Dialog callback. As a result, "PropHost" is a script item that is only visible in "Property" callback scripts. For more information, see Callback Scripting Using PropHost Object and Callback Scripting Using CompInstance Object.

The following table lists allowable callback events, items that are visible from the associated callback script, and the set of accessible functions that can be called.

Callback Event	Scripts Visible from the Event Callback Script	Functions Callable from the Visible Script
Place Component	CompInstance	**CompInstance.GetParentDesign()** — Returns a oDesign item that can be used to call Design functions.
		CompInstance.GetPropserverName() — Returns a CompInstance identification name that can be used in oEditor property-method scripts, such as GetPropertyValue(), SetPropertyValue(), etc.
		CompInstance.GetComponentName() — Returns the component name, e.g. "MS_TRL".
Simulate Component	CompInstance	**CompInstance.GetDesign()** — Returns the interface to the specified design simulation.
		CompInstance.GetProgress() — Returns the completion percentage (from 0 to 100) of the specified design simulation.
		CompInstance.GetRunStatus() — Returns the status number of the specified design simulation.
		CompInstance.Abort() — Aborts the specified design simulation.

The function, **ExecuteAnsoftScript(<ScriptName>)**, searches the configured ANSYS Electronics Desktop script libraries by name for the script passed to it, and invokes the found ScriptName. The invoked script will run with the same set of visible script items as the originally called script. That is, **CompInstance** is visible from the invoked sub-script, ScriptName, and CompInstance's functions can be called from ScriptName.

Executing a Script from Within a Script

ANSYS Electronics Desktop provides a script command that enables you to launch another script from within the script that is being executed:

```
oDesktop.RunScript  <ScriptName>
```

If the full path to the script is not specified, ANSYS Electronics Desktop searches for the specified script in the following locations, in this order:

- Personal library directory.
  This is the **PersonalLib** subdirectory in the project directory. The project directory can be specified in the **General Options** dialog box (click **Tools>Options>General Options** to open this dialog box) under the **Project Options** tab.
- User library directory.
  This is the **userlib** subdirectory in the library directory. The library directory can be specified in the **General Options** dialog box (click **Tools>Options>General Options** to open this dialog box) under the **Project Options** tab.
- System library directory.
  This is the syslib subdirectory in the library directory. The library directory can be specified in the **General Options** dialog box (click **Tools>Options>General Options** to open this dialog box) under the **Project Options** tab.
- ANSYS Electronics Desktop installation directory.

Editing Properties

Any data that is shown in the dockable **Properties** dialog box or in the modal **Properties** pop-up window is called a property. For example, project and local variables are properties. The **XSize** of a box in the Geometry editor is also a property. See the chapter, *Property Script Commands*, for an explanation of how to manipulate properties in a script.

3 Ansoft Application Object Script Commands

The Application object commands allow you to set parameters for RAM and processor use. Application object commands should be executed by the `oAnsoftApp` object.

`oAnsoftApp.<`*`CommandName`*`> <args>`

The depricated commands are no longer supported and produce an error if used.

GetAppDesktop

Use: `GetAppDesktop` is a function of `oAnsoftApp`. This function does not take an input and it returns an object. The object is assigned to the variable `oDesktop`.

Syntax: `GetAppDesktop()`

Return Value: Object.

Parameters: `None`

Example:

```
Set oDesktop = oAnsoftApp.GetAppDesktop()
```

GetDesiredRamMBLimit(deprecated)

GetHPCLicenseType(deprecated)

GetMaximumRamMBLimit (deprecated)

GetMPISpawnCmd(deprecated)

GetMPIVendor(deprecated)

GetNumberOfProcessors(deprecated)

GetUseHPCForMP(deprecated)

SetDesiredRamMBLimit(deprecated)

SetHPCLicenseType(deprecated)

SetMaximumRamMBLimit (deprecated)

SetMPISpawnCmd (deprecated)

SetMPIVendor (deprecated)

SetNumberOfProcessors (deprecated)

SetUseHPCForMP (deprecated)

4 Desktop Object Script Commands

Desktop commands should be executed by the `oDesktop` object. Some new commands permit you to query objects when you do not know the names.

```
Set oDesktop =
     CreateObject("AnsoftHfss.HfssScriptInterface")
oDesktop.CommandName <args>
```

ClearMessages
CloseAllWindows
CloseProject
CloseProjectNoForce
Count
EnableAutoSave
GetActiveProject
GetAutoSaveEnabled
GetDesigns
GetDistributedAnalysisMachines
GetName
GetLibraryDirectory
GetMessages
GetPersonalLibDirectory
GetProjects
GetProjectDirectory

GetProjectList
GetSysLibDirectory
GetTempDirectory
GetUserLibDirectory
GetVersion
ImportANF
ImportAutoCAD
ImportGDSII
ImportODB
NewProject
OpenAndConvertProject
OpenMultipleProjects
OpenProject
PauseScript
Print
QuitApplication
RestoreWindow
RunProgram
RunScript
SetActiveProject
SetActiveProjectByPath
SetLibraryDirectory
SetProjectDirectory
SetTempDirectory
Sleep

Also see:

Desktop Commands For Registry Values

Clear Messages

Use: Clears messages, optionally specifying severity and design.

Command: Clear Messages

Syntax: `ClearMessages(<project>, <design>, <severity>)`

Return Value: None

Parameters:

`project`

Type: <string>

project name, an empty string will clear all project messages.

`design`

Type: <string>

design name, will be ignored if project name is empty; an empty string will clear messages is all Designs.

`severity`

Type: <int>

0 -- clear all info messages;

1 -- clear all info and warning messages;

2-- clear all info, warning and error messages;

3 -- clear all messages included info, warning, error, and fatal-error.

Example:

```
oDesktop.ClearMessages(MyProject, Mydesign,3)
```

CloseAllWindows

Use: Closes all MDI child windows on the desktop.

Command: From main menu, **Window>CloseAll**.

Syntax: `CloseAllWindows()`

Return Value: `None`

Parameters: None

Example:

```
oDesktop.CloseAllWindows()
```

Example:

```
-----------------------------------
Example Script
-----------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
```

```
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.CloseAllWindows()
```

CloseProject

Use: Closes a specified project. Changes to the project are not saved. Save the project using the Project command `Save` or `SaveAs` before closing to save changes.

Command: **File>Close**

Syntax: `CloseProject <ProjectName>`

Return Value: None

Parameters: `<ProjectName>`

Type: <string>

Example:

```
oDesktop.CloseProject "Project1"
```

Example:

```
-----------------------------------
Example Script
-----------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.CloseProject "Project1"
```

CloseProjectNoForce

Use: Closes a specified project unless there are simulations ongoing. Changes to the project will not be saved. Save the project using the Project command `Save` or `SaveAs` before closing to save changes.

Command: **File>Close**

Syntax: `CloseProjectNoForce <ProjectName>`

Return Value: None

Parameters: `<ProjectName>`

Type: <string>

Example:

```
oDesktop.CloseProjectNoForce "Project1"
```

Example:

```
-----------------------------------
Example Script
-----------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.CloseProjectNoForce "Project9"
```

Count

Use: Gets the total number of queried projects or designs obtained by GetProjects() and GetDesigns() commands. See the example query.

Syntax:

Return Value: Returns an integer value.

Parameters: None

Example:

```
set projects = oDesktop.GetProjects()
numprojects = projects.
```

Example:

```
------------------------------------------------------------
Example Script: iterate through using integer index
------------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Dim oProjects
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.RestoreWindow
            Dim projects
            set projects = oDesktop.GetProjects()
            for i = 0 to projects.Count - 1
              msgbox projects(i).GetName()
              dim designs
            set designs = projects(i).GetDesigns()
            for j = 0 to designs.Count - 1
            msgbox designs(j).GetName()
            next
            next
```

EnableAutoSave

Use: Enable or disable autosave feature.

Syntax: EnableAutoSave(bool)

Return Value: None

Parameters: None

Example:

```
oDesktop.EnableAutoSave(true)
```

```
------------------------------------------------------------------------
In this example message box returns 1 since autosave is enabled.
```

```
-----------------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.RestoreWindow
            Set oProject = oDesktop.GetActiveProject()
            Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
            oDesktop.EnableAutoSave(true)
            msgbox(oDesktop.GetAutoSaveEnabled())
```

GetActiveProject

Use: Returns the project that is active in the desktop.

Command: None

Syntax: `GetActiveProject`

Return Value: The project that is active in the desktop.

Parameters: None

Example:

```
            Set oProject = oDesktop.GetActiveProject()
```

Note: GetActiveProject returns normally if there are no active objects.

Example:

```
-----------------------------------
Example Script
-----------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
```

```
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.GetActiveProject()
```

GetAutoSaveEnabled

Use: Checks to see if the autosave feature is enabled.

Command: None

Syntax: `GetAutoSaveEnabled`

Return Value: Boolean

Parameters: `None`

Example:

```
oDesktop.GetAutoSaveEnabled()
```

Example:

```
-----------------------------------
Example Script
-----------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.GetActiveProject()
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
msgbox(oDesktop.GetAutoSaveEnabled())
```

GetDesigns

Use: For querying designs within a queried project obtained by the GetProjects() command. Once you have the designs you can iterate through them using standard VBScript methods. See the example query. Returns the designs in a given project.

Syntax: `GetDesigns()`

Return Value: Returns a COM collection of designs in the given project.

Parameters: `None`

Example:

```
set projects = oDesktop.GetProjects()
set designs = projects(0).GetDesigns()
```

Example:

```
------------------------------------------------------------------------
Object in design(0) is edited. designs(0) is one of several designs in
this project. design(0) implies first design; design(1) implies sec-
ond design and so on.
------------------------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.RestoreWindow
            Set oProject = oDesktop.SetActiveProject("Project9")
            set projects = oDesktop.GetProjects()
            set designs = projects(0).GetDesigns()
            Set oEditor = designs(0).SetActiveEditor("3D Modeler")
            oEditor.ChangeProperty Array("NAME:AllTabs",
            Array("NAME:Geometry3DCmdTab", Array("NAME:PropServ-
            ers",  _
              "Box1:CreateBox:1"), Array("NAME:ChangedProps",
            Array("NAME:ZSize", "Value:=", "1mm"))))
```

GetDistributedAnalysisMachines

Use: Gets a list of machines used for distributed analysis. You can iterate through the list using standard VBScript methods.

Syntax: GetDistributedAnalysisMachines()

Return Value: Returns a COM collection of machines used for distributed analysis.

Parameters: None

Example:

```
for each machine in oDesktop.GetDistributedAnalysisMachines()
    msgbox machine
    next
```

Example:

```
-----------------------------------
Example Script
-----------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Dim oProjects
            Dim omachine
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.RestoreWindow
            for each machine in oDesktop.GetDistributedAnalysisMa-
            chines()
            msgbox machine
            next
```

GetName [Desktop]

Use: Gets names of queried projects or designs obtained by GetProjects() and GetDesigns() commands. See the example query.

Syntax: GetName()

Return Value: Returns a name of type string.

Parameters: None

Example:

```
set projects = oDesktop.GetProjects()
project_name = projects(0).GetName()
```

Example:

```
------------------------------------------------------------
In this example, message box returns project name. projects(0) is the
first of the several projects. Similarly projects(1) displays name of
second project
------------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Dim oProjects
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.RestoreWindow
            set projects = oDesktop.GetProjects()
            project_name = projects(0).GetName()
            msgbox(project_name)
```

Example:

```
------------------------------------------------------
message box returns the names of the projects
------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
```

```
Dim oModule
Dim oProjects
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
erface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
for each prj in oDesktop.GetProjects()
msgbox prj.GetName()
next
```

GetLibraryDirectory

Use: Gets the library directory path.

Syntax: `GetLibraryDirectory`

Return Value: Returns a directory path.

Type: <string>

Parameters: None

Example:

```
libdir = oDesktop.GetLibraryDirectory
```

Example:

```
------------------------------------------------------
message box returns the path in this example
------------------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
erface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
libdir = oDesktop.GetLibraryDirectory
msgbox(oDesktop.GetLibraryDirectory())
```

GetMessages

Use: Collects the messages from a specified project and design.

Command: None

Syntax: `GetMessages <ProjectName>, <DesignName>, <SeverityName>`

Return Value: A simple array of strings.

Parameters:

`<ProjectName>`

Type:<string>

Name of the project for which to collect messages.

An incorrect project name results in no messages (design is ignored)

An empty project name results in all messages (design is ignored)

`<DesignName>`

Type: <string>

Name of the design in the named project for which to collect messages

An incorrect design name results in no messages for the named project

An empty design name results in all messages for the named project

`<SeverityName>`

Type: <integer>

Severity is 0-3, and is tied in to info/warning/error/fatal types as follows:

0 is info and above

1 is warning and above

2 is error and fatal

3 is fatal only (rarely used)

Example:

```
oDesktop.GetMessages(project,design,severity)
```

Example:

```
-----------------------------------------------------
For severity =1, the message box returns the first
warning message for Project9, HFSSDesign1
-----------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
```

```
Dim oEditor
Dim oModule
Dim oProjects
Dim omachine
Dim oseverity
Dim var
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
erface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
var = oDesktop.GetMessages(Project9,HFSSDesign1,1)
msgbox var(0)
```

Example:

```
------------------------------------------------------------
For severity =2, the message box returns the
first error message for Project9, HFSSDesign1
------------------------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Dim oProjects
Dim omachine
Dim oseverity
Dim var
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
erface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
var = oDesktop.GetMessages(Project9,HFSSDesign1,1)
msgbox var(0)
```

Example:

```
'----------------------------------------------------
using for loop display all the messages for
Project9, HFSSDesign1
'----------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Dim oProjects
            Dim omachine
            Dim var
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.RestoreWindow
            for each var in oDesktop.GetMessages("Project9","HFSSDe-
            sign1",1)
            msgbox var
            next
```

GetPersonalLibDirectory

Use: Informational

Command: None

Syntax: `GetPersonalLibDirectory`

Return Value: The directory path string.

Parameters: `None.`

Example:

```
            sys = oDesktop.GetPersonalLibDirectory()
```

Example:

```
'----------------------------------------------------
message box returns the PersonalLib directory path
```

```
------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.RestoreWindow
            sys = oDesktop.GetPersonalLibDirectory()
            msgbox(oDesktop.GetPersonalLibDirectory())
```

GetProjects

Use: For querying projects. Once you have the projects you can iterate through them using standard VBScript methods. See the example query.

Syntax: `GetProjects()`

Return Value: Returns a COM collection of opened projects.

Parameters: `None`

Example:

```
            set projects = oDesktop.GetProjects()
```

Example:

```
------------------------------------------------------
Example Script
------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Dim oProjects
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
```

```
oDesktop.RestoreWindow
for each prj in oDesktop.GetProjects()
msgbox prj.GetName()
for each design in prj.GetDesigns()
msgbox design.GetName()
next
next
```

GetProjectDirectory

Use: Gets the project directory path.

Syntax: `GetProjectDirectory`

Return Value: Returns a directory path.

Type: <string>

Parameters: None

Example:

```
projdir = oDesktop.GetProjectDirectory()
```

Example:

```
------------------------------------------------------
message box returns the project directory path
------------------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
erface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
projdir = oDesktop.GetProjectDirectory()
msgbox(oDesktop.GetProjectDirectory())
```

GetProjectList

Use: Returns a list of all projects that are open in the desktop.

Command: None

Syntax: `GetProjectList()`

Return Value: An array of strings, the names of all open projects in the desktop.

Parameters: None

Example:

```
list_of_projects = oDesktop.GetProjectList()
```

Example:

```
------------------------------------------------------
message box returns list of project names
------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Dim oProjects
            Dim omachine
            Dim lpj
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.RestoreWindow
            for each lpj in oDesktop.GetProjectList()
              msgbox lpj
            next
```

GetSysLibDirectory

Use: Informational.

Command: None.

Syntax: `GetSysLibDirectory`

Return Value: The directory path string

Parameters: `None`

Example:

```
dim sys
sys = oDesktop.GetSysLibDirectory()
```

Example:

```
-----------------------------------------------------
message box returns system library directory path
-----------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.RestoreWindow
            sys = oDesktop.GetSysLibDirectory()
            msgbox(oDesktop.GetSysLibDirectory())
```

GetTempDirectory

Use: Gets the temp directory path.

Syntax: `GetTempDirectory`

Return Value: Returns a directory path.

Type: <string>

Parameters: `None`

Example:

```
tempdir = oDesktop.GetTempDirectory
```

Example:

```
-----------------------------------------------------
message box returns temp directory path
-----------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
```

```
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
tempdir = oDesktop.GetTempDirectory()
msgbox(oDesktop.GetTempDirectory())
```

GetUserLibDirectory

Use: Informational.

Command: None

Syntax: `GetUserLibDirectory`

Return Value: The directory path string

Parameters: `None`

Example:

```
dim sys
sys = oDesktop.GetUserLibDirectory()
```

Example:

```
------------------------------------------------------
message box returns userlib directory path
------------------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
userlib = oDesktop.GetUserLibDirectory()
msgbox(oDesktop.GetUserLibDirectory())
```

GetVersion

Use: Returns a string representing the version.

Syntax: GetVersion()

Return Value: string

Parameters: None

Example:

msgbox(oDesktop.GetVersion()), displays "10.0"

Example:

```
------------------------------------------------------
message box displays version number
------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Dim oProjects
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.RestoreWindow
            msgbox(oDesktop.GetVersion())
```

ImportANF

Use: Import an ANF file into a new project.

Command: File > Import > ANF

Syntax: ImportANF<"ANF_filename">

Return Value: None

Parameters: <"ANF_filename">

Type: text

Example:

```
            oDesktop.RestoreWindow()
            Set oTool = oDesktop.GetTool("ImportExport")
            oTool.ImportANF("C:/AnsTranslator/results/package 4.anf")
```

Example:

```
------------------------------------------------------
Example shows how to import an ANF file.
------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Dim oProjects
            Dim omachine
            Dim oTool
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.RestoreWindow
            Set oTool = oDesktop.GetTool("ImportExport")
            oTool.ImportANF("C:\Program Files\AnsysEM\Ansy-
            sEM16.0\Win64\Examples\Package\package_board.anf")
```

ImportAutoCAD

Use: Import an AutoCAD file into a new project.

Command: File >Import > AutoCAD

Syntax: `ImportAutoCAD <"AutoCADfilename">, <"Controlfilename">`

Return Value: None

Parameters: `<"AutoCADfilename">`

Type: text

Description: name of the AutoCAD file to import

<"OutputEDBfilename">

Type: text

Description: name of the EDB file to create during the import.

<"Controlfilename">

Type: text

Description: name of the xml control file to use to guide the import.

This string can be empty ("") if no control file is to be used.

Example:

```
Set oTool = oDesktop.GetTool("ImportExport")
oTool.ImportDXF"a4lines.dxf", "a4lines.aedb.edb"
"a4lines.xml"
```

ImportGDSII

Use: Import a GDSII file into a new project.

Command: File > Import > GDSII

Syntax:

```
ImportGDSII
<"GDSIIfilename">
<"OutputEDBfilename">,
<"Controlfilename">,
<"PropertyMappingfilename">
```

Return Value: None

Parameters:

<"GDSIIfilename">

Type: text

Description: name of the GDSII file to import

<"OutputEDBfilename">

Type: text

Description: name of the EDB file to create during the import.

<"Controlfilename">

Type: text

Description: name of the xml control file to use to guide the import.

This string can be empty ("") if no control file is to be used.

<"PropertyMappingfilename">

Type: text

Description: name of the property mapping file to use to guide the import.

This string can be empty ("") if no control file is to be used.

Example:

```
Set oTool = oDesktop.GetTool("ImportExport")
oTool.ImportGDSII  "test.gds",  "test.aedb.edb",
"test.xml",  "test.txt"
```

ImportODB

Use: Import an ODB++ file into a new project.

Command: File > Import > ODB++

Syntax:
```
ImportODB++ <"ODB++filename">,   <"OutputEDBfilename>,
<"Controlfilename">
```

Return Value: None

Parameters:

<"ODB++filename">

Type: text

Description: name of the ODB++ file to import

<"OutputEDBfilename">

Type: text

Description: name of the EDB file to create during the import.

<"Controlfilename">

Type: text

Description: name of the xml control file to use to guide the import.

This string can be empty ("") if no control file is to be used.

Example:

```
Set oTool = oDesktop.GetTool("ImportExport")
oTool.ImportODB "test.tgz", "test.aedb.edb", "test.xml"
```

NewProject

Use: Creates a new project. The new project becomes the active project.

Command: **File>New**

Syntax:
```
NewProject
```

Return Value: An object reference to the newly-added project.

Parameters: None

Example:

```
Set oProject = oDesktop.NewProject
```

Example:

```
------------------------------------------------------
Example creates a new project.
------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
```

```
Dim oDesign
Dim oEditor
Dim oModule
Dim oProjects
Dim omachine
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
erface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.NewProject
```

OpenAndConvertProject

Use: Opens a legacy project and converts or copies to .aedt format.

Command: **File>Open**, and choose a legacy project.

Return Value: An object reference to the newly-opened project which has the .aedt extension.

Syntax: `OpenAndConvertProject(`*`filePath, legacyChoice`*`)`

Parameters: *filePath*: full path of the legacy project to open

legacyChoice: integer with one of the following values:

0: show conversion dialog, (same as **File>Open** of a legacy file)

1: convert, will convert the project/results to project/results with the .aedt extension in the same directory, the original project will no longer be available

2: copy, will copy the project/results to a new project/results with the .aedt extension in the same directory, the originals will still be available.

Warning: If project file / results with the same name and .aedt extension already exist in the same directory, they will be overwritten.

Example:

```
' convert original to OptimTee.aedt
set oProject = oDesktop.OpenAndConvertProject("C:/legacy/
OptimTee.hfss", 1)

' create OptimTee.aedt, leave original
set oProject = oDesktop.OpenAndConvertProject("C:/legacy/
OptimTee.hfss", 2)
```

OpenMultipleProjects

Use: Opens all files of a specified type in a specified directory.

Command: **File>Multiple Open**

Syntax: `OpenMultipleProjects <Directory> <FileType>`

Return Value: None

Parameters: `<Directory>`

Type: <string>

`<FileType>`

Type: <string>

Example:

```
oDesktop.OpenMultipleProjects "D:/Projects", "*.hfss"
```

OpenProject

Use: Opens a specified project.

Command: **File>Open**

Syntax: `OpenProject <FileName>`

Return Value: An object reference to the newly-opened project.

Parameters: `<FileName>`: Full path of the project to open.

Type: <string>

Example:

```
oDesktop.OpenProject "D:/Projects/Project1.hfss"
```

Example:

```
------------------------------------------------------
Example opens a specified project.
------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Dim oProjects
            Dim omachine
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
```

```
oDesktop.RestoreWindow
oDesktop.OpenProject "E:/helical_antenna.hfss"
```

PauseScript

Use: Pauses the script's execution and displays a message in a pop-up dialog box to the user. The script execution will not resume until the user clicks **Tools>Resume Script**.

Command: **Tools>Pause Script**

Syntax: `PauseScript <Message>`

Return Value: None

Parameters: `<Message>`

Type: <string>

Example:

```
oDesktop.PauseScript "Text to display in pop-up dialog box"
```

Example:

```
------------------------------------------------------
Example pauses a script. Resume script to run it.
------------------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Dim oProjects
Dim omachine
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
oDesktop.PauseScript "Script is paused. Click OK to continue"
Set oProject = oDesktop.NewProject
oDesktop.OpenProject "E:/helical_antenna.hfss"
```

Print

Use: Prints the contents of the active view window.

Command: **File>Print**

Syntax: `Print`

Return Value: None

Parameters: None

Example:

```
oDesktop.Print
```

Example:

```
------------------------------------------------------
Example prints the contents of the active window.
------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Dim oProjects
            Dim omachine
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.RestoreWindow
            oDesktop.Print
```

QuitApplication

Use: Exits the desktop.

Command: **File>Exit**

Syntax: `QuitApplication`

Return Value: None

Parameters: None

Example:

```
oDesktop.QuitApplication
```

```
-----------------------------------------------------
Example : quit an application.
-----------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Dim oProjects
            Dim omachine
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.RestoreWindow
            oDesktop.QuitApplication
```

RestoreWindow

Use: Restores a minimized HFSS window.

Command: None

Syntax: `RestoreWindow`

Return Value: None

Parameters: None

Example:

```
            oDesktop.RestoreWindow
-----------------------------------------------------
Example : restores minimized HFSS window.
-----------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
```

```
Dim oProjects
Dim omachine
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
```

RunProgram

Use: Runs an external program.

Command: None

Syntax: `RunProgram <ProgName>, <ProgPath>, <WorkPath>, <ArgArray>`

Return Value: None

Parameters:

`<ProgName>`

Type: <string>

Name of the program to run.

`<ProgPath>`

Type: <string>

Location of the program. Pass in an empty string to use the system path.

`<WorkPath>`

Type: <string>

Working directory in which program will start.

<ArgArray>

Type: Array of strings

Arguments to pass to the program. If no arguments, pass in `None`.

Example:

```
oDesktop.RunProgram "winword.exe", _
   "C:\Program Files\Microsoft Office\Office10",_
   "", None
```

RunScript

Use: Launches another script from within the script currently being executed.

Command: **Tools>Run Script**

Syntax: `RunScript <ScriptPath>`

Return Value: None

Parameters: `<ScriptPath>`

Type: <string>

Name or full path of the script to execute. If the full path to the script is not specified, HFSS searches for the specified script in the following locations, in this order:

- Personal library directory.

  This is the **PersonalLib** subdirectory in the project directory. The project directory can be specified in the **General Options** dialog box (click **Tools>Options>General Options** to open this dialog box) under the **Project Options** tab.

- User library directory.

  This is the **userlib** subdirectory in the library directory. The library directory can be specified in the **General Options** dialog box (click **Tools>Options>General Options** to open this dialog box) under the **Project Options** tab.

- System library directory.

  This is the syslib subdirectory in the library directory. The library directory can be specified in the **General Options** dialog box (click **Tools>Options>General Options** to open this dialog box) under the **Project Options** tab.

- HFSS installation directory.

Example:

```
oDesktop.RunScript("C:/Project/test1.vbs")
```

SetActiveProject

Use: Returns a specified project as the active project in the desktop.

Command: None

Syntax: `SetActiveProject <ProjectName>`

Return Value: The specified project becomes active in the desktop.

Parameters: `<ProjectName>`

Type: <string>

Example:

```
Set oProject = oDesktop.SetActiveProject ("Project1")
```

Example:

```
------------------------------------------------------
Example sets an existing project as active.
------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
```

```
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Dim oProjects
Dim omachine
Dim lpj
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject ("Project9")
msgbox "Active Project Set to " + oDesktop.GetActiveProject().GetName()
```

SetActiveProjectByPath

Use: If a user has two projects open with the same name, the result of SetActiveProject is ambiguous (The first one listed in selected). This command permits unambiguous specification of the active project.

Syntax: `SetActiveProjectByPath()`

Return Value: The specified project becomes active in the desktop.

Parameters: `<fullPathProjectName>`

Example:

```
Set oProject = oDesktop.SetActiveProjectByPath("C:\working\tee.hfss")
```

Example:

```
------------------------------------------------------
Example sets an existing project as active by path.
------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Dim oProjects
            Dim omachine
```

```
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProjectByPath("E:/helical_antenna.hfss")
msgbox "Active Project Set to " + oDesktop.GetActiveProject().GetName()
```

SetLibraryDirectory

Use: Sets the library directory path. The specified directory must already exist and contain a syslib folder.

Syntax: `SetLibraryDirectory <DirectoryPath>`

Return Value: None

Parameters: `<DirectoryPath>`

Type: <string>

Example:

```
oDesktop.SetLibraryDirectory "c:\libraries"
```

SetProjectDirectory

Use: Sets the project directory path. The directory will be automatically created if it does not already exist.

Syntax: `SetProjectDirectory <DirectoryPath>`

Return Value: None

Parameters: `<DirectoryPath>`

Type: <string>

Example:

```
oDesktop.SetProjectDirectory "c:\projects"
```

SetTempDirectory

Use: Sets the temp directory path. The directory will be automatically created if it does not already exist.

Syntax: `SetTempDirectory <DirectoryPath>`

Return Value: None

Parameters: `<DirectoryPath>`

Type: <string>

Example:

```
oDesktop.SetTempDirectory "c:\temp"
```

Example:

```
------------------------------------------------------
Example sets Temp directory path.
------------------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Dim oProjects
Dim omachine
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
oDesktop.SetTempDirectory("C:\temp")
msgbox "Temp Directory Set to " + oDesktop.GetTempDirectory()
```

Sleep

Use: Suspends execution of HFSS for the specified number of milliseconds, up to 60,000 milliseconds (1 minute).

Command: none

Syntax: `Sleep <TimeInMilliseconds>`

Return Value: None

Parameters: `<TimeInMilliseconds>`

Type: <int>

Example:

```
oDesktop.Sleep 1000
```

Desktop Commands For Registry Values

The ANSYS Registry is stored as XML format file. By default it is located at C:\Users\<User-Name>\Documents\Ansoft\<AnsysProductNameversion>\config\<PC_NAME>_user.XML. Most of the ANSYS product configuration information is stored in this XML file. These methods allow you to change the product configuration in VB-script or Python script.

For example, to set the DSO & HPC analysis setup with Python script:

1. In a PC start HFSS go to the DSO and HPC options dialog, create a setup name it as "test"
2. Export the setup to a file c:\tem\test.acf.
3. Copy the exported file to a target PC f:\temp\test.acf.
4. Run the following script.

```
#import the setup
oDesktop.SetRegistryFromFile("f:\\temp\\test.acf")
# Set Active Setup to "test"
oDesktop.SetRegistryString("Desktop/ActiveDSOConfigurations/HFSS",
"test")
```

DoesRegistryValueExist
GetRegistryInt
GetRegistryString
SetRegistryFromFile
SetRegistryInt
SetRegistryString

DoesRegistryValueExist

Use: Determine if a registry value exists.

Command: None

Syntax: `DoesRegistryValueExist <KeyName>`

Return Value: Boolean. True if key exists. False otherwise.

Parameters: `<KeyName> - registry key name with the full path.`

Example:

```
bExist = oDesktop.DoesRegistryValueExist("Desktop/ActiveDSOConfigura-
tions/HFSS")
```

GetRegistryInt

Use: Obtain RegistryKey integer value.

Command: None

Syntax: `GetRegistryIn(<KeyName>)`

Return Value: Integer for success, if the integer value is found. Return as Bad-Argument-Value, if Registry key does not exist or it is not an integer value.

Parameters: `<KeyName>`

Registry key name with the full path.

Example:

```
num = oDesktop.GetRegistryInt("Desktop/Settings/ProjectOptions/HFSS/
UpdateReportsDynamicallyOnEdits")
```

GetRegistryString

Use: Obtain RegistryKey string value.

Command: None.

Syntax: `GetRegistryString <KeyName> <PVal>`

Return Value: String for success. Bad argument if they key is not defined or has an integer value.

Parameters: `<KeyName>`

Registry key name with the full path.

Example:

```
activeDSO = oDesktop.GetRegistryString("Desktop/ActiveDSOConfigurations/
HFSS")
```

SetRegistryFromFile

Use: Configures a registry by specifying an Analysis Configuration file which must have been exported from the HPC and Analysis panel.

Command: None.

Syntax: `SetRegistryFromFile "<filePath>"`

Return Value: Success if Analysis configuration is imported. Bad argument value if the file is not found or does not contain valid analysis configuration data.

Parameters: `<filePath>`

Full path to an Analysis Configuration file.

Example:

```
oDesktop.SetRegistryFromFile "c:/temp/test.acf"
```

SetRegistryInt

Use: Sets a registry key with an integer value.

Command: None

Syntax: `SetRegistryInt "<KeyName>", <int>`

Return Value: Success if the key is defined as an integer. Bad argument value if a key is not defined, or if the value is a text string.

Parameters: `<KeyName>`

Registry key name with full path.

<int>

New integer value.

Example:

```
oDesktop.SetRegistryInt "Desktop/Settings/ProjectOptions/HFSS/Updat-
eReportsDynamicallyOnEdits", 0
```

SetRegistryString

Use: Sets a string value for a specified registry key.

Command: None.

Syntax: `SetRegistryString "<KeyPath>" "<val>"`

Return Value: Success, if the key is defined as a text string. Bad argument value if the key is not defined or requires an integer value.

Parameters: `<KeyName>`

Registry key name with full path.

<val>

New string value.

Example:

```
oDesktop.SetRegistryString "Desktop/ActiveDSOConfigurations/HFSS",
"Local"
```

5 Project Object Script Commands

Project commands should be executed by the `oProject` object. One example of accessing this object is:

```
Set oProject = oDesktop.GetActiveProject()
```

Close
CopyDesign
CutDesign
DeleteDesign
GetActiveDesign
GetDesign
GetName [Project]
GetPath
GetTopDesignList
InsertDesign
Paste
Redo
Save
SaveAs
SetActiveDesign
SimulateAll
Undo

Close

Use: Closes the active project. Unsaved changes will be lost.

Command: None

Syntax: `Close`

Return Value: None

Parameters: `None`

Example:

```
oProject.Close
```

CopyDesign

Use: Copies a design.

Command: **Edit>Copy**

Syntax: `CopyDesign <DesignName>`

Return Value: None

Example:

```
oProject.CopyDesign "HFSSDesign1"
```

Example:

```
------------------------------------------------------
Example Script
------------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Dim oProjects
            Dim omachine
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.RestoreWindow
            Set oProject = oDesktop.GetActiveProject()
            oProject.CopyDesign "Differential"
```

CutDesign

Use: Cuts a design from the active project. The design is stored in memory and can be pasted in any HFSS or Q3D Extractor project.

Command: **Edit>Cut**

Syntax: `CutDesign <DesignName>`

Return Value: None

Example:

```
oProject.CutDesign "HFSSDesign1"
```

Example:

```
oProject.CutDesign "Q3DDesign1"
```

Warning This is a legacy command that is no longer supported and should not be used as it may have unintended effects on solved designs.

DeleteDesign

Use: Deletes a specified design in the project.

Command: **Edit>Delete**

Syntax: `DeleteDesign <DesignName>`

Return Value: None

Example:

```
oProject.DeleteDesign "HFSSDesign2"
```

GetActiveDesign

Use: Returns the design in the active project.

Command: None

Syntax: `GetActiveDesign`

Return Value: The active design.

Parameters: None

Example:

```
Set oDesign = oProject.GetActiveDesign ()
```

GetDesign

Use: Returns the specified design.

Command: None

Syntax: `GetDesign <DesignName>`

Return Value: The specified design.

Parameters: `<DesignName>`

Type: <string>

Name of the design to return.

Example:

```
Set oDesign = oProject.GetDesign ("HFSSDesign1")
```

GetName [Project]

Use: Returns the project name.

Command: None

Syntax: `GetName`

Return Value: The active project's name.

Parameters: None

Example:

```
name = oProject.GetName ()
```

GetPath

Use: Returns the location of the project on disk.

Command: None

Syntax: `GetPath`

Return Value: The path to the project, which does not include the project name.

Parameters: None

Example:

```
path = oProject.GetPath ()
```

GetTopDesignList

Use: Returns a list of the names of the top-level designs.

Command: None

Syntax: `GetTopDesignList`

Return Value: An array of strings that are the names of the top-level designs. Returns null if there are no top level designs.

Parameters: None

Example:

```
name_list = oProject.GetTopDesignList ()
```

InsertDesign

Use: Inserts a new design in the project. In HFSS scripts, the last argument will always be empty.

Command: **Project>Insert HFSS Design**

Syntax: `InsertDesign "HFSS", <DesignName>, <SolutionType>, ""`

Return Value: Object

Parameters: `<DesignName>`

Type: <string>

Name of the new design.

`<SolutionType>`

Type: <string>

Solution type of the new design. Can be `"DrivenModal"`, `"DrivenTerminal"`, or `"Eigenmode"`.

Example:

```
Set oDesign = oProject.InsertDesign("HFSS", "HFSSDesign3",_
              "DrivenModal", "")
```

For Insert EMDesigns, Insert Circuit NetList Design, and Insert Filter Design items the command details are as follows.

Use: Inserts a new design in the project. In Designer scripts, the last argument will always be empty.

Command: **Project>Insert Designer Design**

Syntax: InsertDesign <DesignType>, <DesignName>, <TechnologyFile>:<SubCircuitID>,“”

Return Value: None.

Parameters:

<DesignType>

Type: <string>

Possible Values:

“Circuit”, “Nexxim Circuit”, “System”, “Nexxim Netlist”, “Planar EM”

<DesignName>

Type: <string>

Name of the new design

<TechnologyFile>:<SubCircuitID>

Type: <string>

The path to the Designer technology file to be used in this design. Use a pair of empty double quotes (““) for none. <SubCircuitID> is optional and must be preceded by a colon if included along with the Technology File name. No colon is necessary when the subcircuit ID is omitted.

Example: oProject.InsertDesign "Nexxim Circuit","MyDesigner", "C:\Program Files\AnsysEM\Designer\"

oProject.InsertDesign "Nexxim Circuit","MyDesigner:12", "C:\Program Files\AnsysEM\Designer\

For Q3D Extractor the InsertDesign command details are as follows:

Use: Inserts a new design in the project. In HFSS scripts, the last argument will always be empty.

Command: **Project>Insert Q3D Extractor Design**

Syntax: `InsertDesign "Q3D Extractor", <DesignName>, "", ""`

Return Value: None

Parameters:

```
<DesignName>
   Type: <string>
   Name of the new design.
```

Example:

```
oProject.InsertDesign "Q3D Extractor","Q3DDesign2", "",
```

Paste [Design]

Use: Pastes a design in the active project.

Command: **Edit>Paste**

Syntax: `Paste`

Return Value: None

Parameters: None

Example:

```
oProject.Paste
```

Example:

```
------------------------------------------------------
Example Script
```

```
'----------------------------------------------------
            Dim oAnsoftApp
            Dim oDesktop
            Dim oProject
            Dim oDesign
            Dim oEditor
            Dim oModule
            Dim oProjects
            Dim omachine
            Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInt-
            erface")
            Set oDesktop = oAnsoftApp.GetAppDesktop()
            oDesktop.RestoreWindow
            Set oProject = oDesktop.GetActiveProject()
            oProject.CopyDesign "Differential"
            oProject.Paste
```

For Q3D Extractor the Paste command works as follows:

Use: Pastes copied objects and returns an array of pasted objects from the 3D model editor.

Command: **Edit>Paste**

Syntax: `Paste`

Return Value: One dimensional array of pasted object names. The order is not guaranteed to be alphabetical.

Parameters: None

Example: `arrayEntities = oEditor.Paste`

Redo [Project Level]

Use: Reapplies the last project-level command.

Command: **Edit>Redo**

Syntax: `Redo`

Return Value: None

Parameters: `None`

Example:

```
oProject.Redo
```

Save

Use: Saves the active project.

Command: **File>Save**

Syntax: `Save`

Return Value: None

Parameters: `None`

Example:

```
oProject.Save
```

SaveAs

Use: Saves the project under a new name.

Command: **File>Save As**

Syntax: `SaveAs <FileName> <OverWrite>`

Return Value: None

Parameters: `<FileName>`

Type: <string>

New name for the file.

`<OverWrite>`

Type: <bool>

Set to `true` if an existing project by that name should be overwritten.

Example:

```
oProject.SaveAs "D:/projects/project1.aedt", true
```

For Schematic and Layout editor related projects SaveAs functions as follows.

Use: Saves the project under a new name.

Command: **File > Save As**

Syntax: SaveAs <FileName> <OverWrite> <DefaultAction> <OverrideActions>

Return Value: None

Parameters:

<FileName>

Type: <string>

New name for the file

<OverWrite>

Type: <bool>

Set to true if an existing project by that name should be overwritten

<DefaultAction>

Type: <string>

Set to one of the following action strings: ef_overwrite , ef_copy_no_overwrite, ef_-make_path_absolute or empty string

<OverrideActions>

Type: Array("Name: OverrideActions", <Files>, <Files>, ...)

<Files>

Type: Array("Name: <Action>", <FileName>, <FileName>, ...)

<DAction>

Type: <string>

Set to one of the following action strings: ef_overwrite , ef_copy_no_overwrite, ef_-make_path_absolute or empty string

Example:

```
oProject.SaveAs
  "F:\Designer Projects\TA33097\HighSpeedChannel.aedt",
true, "ef_overwrite", Array("NAME:OverrideActions",
Array("NAME:ef_copy_no_overwrite",
Array("NAME:Files","$PROJECTDIR/circuit_models.inc")),
Array("NAME:ef_make_path_absolute",
Array("NAME:Files","$PROJECTDIR\SL_6s.sp")))
```

Note

The Action and DefaultAction strings correspond to the following actions:

- ef_overwrite — Copy File to New Project Directory and Overwrite
- ef_copy_no_overwrite — Copy File to New Project Directory and Don't Overwrite
- ef_make_path_absolute — Change Reference to Point to File in Old Project Directory
- empty string — Do Nothing

The DefaultAction is applied to all files that are NOT explicitly listed in the OverrideActions array. Those in the OverrideActions array are separate arrays for actions that are different from the default action; those actions are applied to the files listed in the same array:

- If OverrideActions are not specified, then DefaultAction is applied to ALL files in project directory.
- If DefaultAction is not specified, then nothing is done (action is Do Nothing).

SetActiveDesign

Use: Sets a new design to be the active design.

Command: None

Syntax: `SetActiveDesign <DesignName>`

Return Value: The named design becomes active.

Parameters: `<DesignName>`

Type: <string>

Name of the design to set as the active design.

Example:

```
Set oDesign = oProject.SetActiveDesign ("HFSSDesign2")
```

SimulateAll

Use: Runs the `SimulateAll` project-level script command from the script, which will simulate all HFSS solution setups and Optimetrics setups for all design instances in the project.

Command: None

Syntax: None

Return Value: `SimulateAll` script command.

Parameters: None

Example:

```
oProject.SimulateAll
```

Undo [Project]

Use: Cancels the last project level command.

Command: **Edit>Undo**

Syntax: `Undo`

Return Value: None

Parameters: `None`

Example:

```
oProject.Undo
```

6 Material Script Commands

Material commands other than the DoesMaterialExist command should be executed by the `oProject` object. The DoesMaterialExist command is executed by the `o`DefinitionManager object. Material commands apply to all products.

```
Set oProject = oDesktop.SetActiveProject("Project1")
oProject.CommandName <args>
```

AddMaterial

Use: Adds a local material.

Command: Add Material command in the material editor.

Syntax:
```
AddMaterial Array("NAME:<MaterialName>",
   <MatProperty>, <MatProperty>, ...)
```

Return Value: None

Parameters:
```
<MatProperty> (simple material)
   "<PropertyName>:=", <value>

<MatProperty> (anisotropic material)
   Array("NAME:<PropertyName>",
      "property_type:=", "AnisoProperty",
      "unit:=", <string>",
      "component1:=", <value>,
      "component2:=", <value>,
      "component3:=", <value>))
```

`<PropertyName>`
Type: <string>
Should be one of the following: `"permittivity"`,
`"permeability"`, `"conductivity"`
`"dielectric_loss_tangent"`,
`"magnetic_loss_tangent"`, `"saturation_mag"`,
`"lande_g_factor"`, `"delta_H"`

`property_type`
Type: <string>
Should be `"AnisoProperty"`.

`unit`
Type: <string>
Possible values:
`delta_H`: `"Oe"`
`saturation_mag`: `"Gauss"`, `"uGauss"`, `"Tesla"`, `"uTesla"`
other properties: `""` (empty string)

Example:

```
Set oDefinitionManager = oProject.GetDefinitionManager()
oDefinitionManager.AddMaterial Array("NAME:Material2",_
  "dielectric_loss_tangent:=", "44",
  Array("NAME:saturation_mag",_
    "property_type:=", "AnisoProperty",_
    "unit:=", "Gauss",_
    "component1:=", "11", _
    "component2:=", "22", _
    "component3:=", "33"), _
  "delta_H:=", "44Oe")
```

An example related to Q3D Extractor is given below.

Example:

```
oProject.AddMaterial Array("NAME:Material2",_
  "dielectric_loss_tangent:=", "44",
  Array("NAME:saturation_mag",_
    "property_type:=", "AnisoProperty",_
    "unit:=", "Gauss",_ "component1:=", "11", _
    "component2:=", "22", _ "component3:=", "33"), _
  "delta_H:=", "44Oe")
```

CloneMaterial

Use: Clones a local material.

Command: None

Syntax: `CloneMaterial(<name>, <newName>)`

Return Value: True if Material is cloned successfully. False if the existing material is not found or if there is a name conflict with the new material name.

Parameters:

`name`

Type: <string>

Existing material name.

`<newName>`

Type: <string>

The new material name for newly cloned material.

Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project1")
Set oDefinitionManager = oProject.GetDefinitionManager()
oDefinitionManager.CloneMaterial("copper1", "copper3")
```

DoesMaterialExist

Use: Checks for the presence of a material in the library by name

Command: None

Syntax: DoesMaterialExist(<MaterialName>)

Return Value: Boolean

Parameters:

`<materialName>`

Type: <string>

Name of the material to search for in the material database

Example:

```
Set oProject = oDesktop.SetActiveProject("cir_th")
Set oDesign = oProject.SetActiveDesign("Design1")
Set oDefinitionManager = oProject.GetDefinitionManager()
if(oDefinitionManager.DoesMaterialExist("vacuum")) then
    MsgBox("It Exists")
else
    MsgBox("It Doesn't Exist...Creating")
    oDefinitionManager.AddMaterial Array("NAME:Material5", _
    "CoordinateSystemType:=", "Cartesian", _
    Array("NAME:AttachedData"), _
    Array("NAME:magnetic_coercivity",
    "property_type:=", "VectorProperty", _
    "Magnitude:=", "-1A_per_meter", _
    "DirComp1:=", "1", _
    "DirComp2:=", _
    "2", "DirComp3:=", "3"))
end if
```

An example related to Q3D Extractor is as follows:

Example:

```
Set oProject = oDesktop.SetActiveProject("cir_th")
Set oDesign = oProject.SetActiveDesign("Design1")
Set oDefinitionManager = oProject.GetDefinitionManager()
if(oProject.DoesMaterialExist("modified_epoxy")) then
  MsgBox("It Exists")
else
  MsgBox("It Doesn't Exist...Creating")
  oDefinitionManager.AddMaterial
  Array("NAME:modified_epoxy",
  "CoordinateSystemType:=", _ "Cartesian",
  Array("NAME:AttachedData"),
  Array("NAME:ModifierData"), "permittivity:=", "4.8",
  "dielectric_loss_tangent:=", "0.025")
```

EditMaterial

Use: Modifies an existing material.

Command: **View/Edit Materials** command in the material editor.

Syntax:
```
EditMaterial <OriginalName>, Array("NAME:<NewName>",
   <MatProperty>, <MatProperty>, ...)
```

Return Value: None

Parameters:

`<OriginalName>`

Type: <string>

Name of the material before editing.

`<NewName>`

Type: <string>

New name for the material.

ExportMaterial

Use: Exports a local material to a library.

Command: **Export to Library** command in the material editor.

Syntax:
```
ExportMaterial <ExportData>, <Library location>
```

Return Value: None

Parameters:
```
<ExportData>
   Array("NAME:<LibraryName>",
      <MaterialName>, <MaterialName>, ...)
```

Example:
```
oProject.ExportMaterial Array("NAME:mo0907b",_
   "Material1", "Material2", "Material3"),_
   "UserLib"
```

RemoveMaterial

Use: Removes a material from a library.

Command: **Remove Material(s)** command in the material editor.

Syntax:
```
RemoveMaterial <MaterialName>, <IsProjectMaterial>,
    <LibraryName>, <LibraryLocation>
```

Return Value: None

Parameters:

`<MaterialName>`

Type: <string>

Name of the material to be removed.

<IsProjectMaterial>

Type: <bool>

If true, HFSS (or Q3D Extractor)assumes the material is a project material. In this case, the last two parameters will be ignored.

<LibraryName>

Type: <string>

The name of the user or personal library where the material resides.

<LibraryLocation>

Type: <string>

Should be "UserLib" or "PersonalLib".

Example:

```
oProject.RemoveMaterial "Material1", false, "mo0907", "UserLib"
oProject.RemoveMaterial "Material1", true, "Local", "Project"
```

7 Property Script Commands

Property commands should be executed by the `oProject` object.

```
Set oProject = oDesktop.SetActiveProject("Project1")
oProject.CommandName <args>
```

Some of the topics covered in this chapter are as follows:

Conventions Used in this Chapter

- Property

  Refers to a single item that can be modified in the dockable **Properties** dialog box or in the modal **Properties** pop-up window.

- `<PropServer>`

  Refers to the item whose properties are being modified. This is usually a compound name giving all the information needed by the editor, design, or project to locate the

item being edited.

- `<PropTab>`

  Corresponds to one tab in the **Property** dialog box – the tab under which properties are being edited.

- `<PropName>`

  Name of a single property .

`<PropServer>` and `<PropTab>` Names

Project

Project Variables:

```
<PropServer>
   "ProjectVariables"

<PropTab>
   "ProjectVariableTab"
```

AnsoftHfss Design and Q3D Extractor Design

Local Variables:

```
<PropServer>
   "LocalVariables"

<PropTab>
   "LocalVariableTab"
```

Passed Parameters

```
<PropServer>
   "Instance:<name of circuit instance>"
<PropTab>
   "PassedParameter Tab"
```

Definition Parameters

```
<PropServer>
   "DefinitionParameters"
<PropTab>
   "DefinitionParameters"
```

Modules and Editors

```
<PropServer>
```

Format is: `<ModuleName>:<ItemName>`, where `<ItemName>` is the boundary name, solution setup name, etc., depending on which module is being edited.
Example: `<PropServer>` for the boundary "PerfE1" is `"BoundarySetup:PerfE1"`

`<PropTab>`
Boundary module: `"HfssTab"`
Mesh Operations module: `"MeshSetupTab"`
Analysis module: `"HfssTab"`
Optimetrics module: `"OptimetricsTab"`
Solutions module: *Does not support properties.*
Field Overlays module: `"FieldsPostProcessorTab"`
Radiation module: `"RadFieldSetupTab"`
Circuit module: "`CCircuitTab`"
System module: "`SystemTab`"
PlanarEM module: "`PlanarEMTab`"
Nexxim module: "`NexximTab`"
Layout elements: "`BaseElementTab`"
Schematic elements: "`ComponentTab`"
Optimetrics module: "`OptimetricsTab`"

`AnsoftHfss` 3D Model Editor

Object in the module:

`<PropServer>`
Name of the object. For example: `"Box1"`.

`<PropTab>`
`"Geometry3DAttributeTab"`

Operation on an object:

`<PropServer>`
Format is `<ObjName>:<OperationName>:<int>`
Concatenation of object name, operation name, and the index of the operation.
For example: `"Box2:CreateBox:2"` refers to the second

"CreateBox" command in Box2's history.

```
<PropTab>
   "Geometry3DCmdTab"
```

Reporter

Operations on Report properties:

Format is <ReportSetup>

For example, to set the Company Name in the plot header to "My Company":

```
Set oModule = oDesign.GetModule("ReportSetup")
oModule.ChangeProperty Array("NAME:AllTabs",_
Array("NAME:Header",_ Array("NAME:PropServers",_
"XY Plot1:Header"), Array("NAME:ChangedProps",_
Array("NAME:Company Name", "Value:=", "My Company"))))
```

Q3D Extractor Modules

```
<PropServer>
   Format is: <ModuleName>:<ItemName>, where <ItemName> is
   the solution setup name, etc., depending on which mod-
   ule is being edited.
<PropTab>
   Boundary module:  "Q3D ExtractorTab" or "2D
   ExtractorTab"
   Mesh Operations module: "MeshSetupTab"
   Analysis module:  "Q3D ExtractorTab" or "2D
   ExtractorTab"
   Optimetrics module:  "OptimetricsTab"
   Solutions module:  Does not support properties.
Field Overlays module: "FieldsPostProcessorTab"
```

Note: For scripted property changes in the various modules and editors, refer to the chapters on the System, PlanarEM, and Nexxim tools, as well as the Layout and Schematic editors.

Callback Scripting Using PropHost Object

Callback scripts are scripts that can be set in the Property Dialog for individual properties by clicking the button in the Callback column and choosing a script that is saved with the project. Callback scripts can contain any legal script commands including general ANSYS script function calls (e.g., GetApplicationName()). In addition, Callback scripts can also call functions on a

special object named PropHost. The PropHost represents the PropServer (owner of properties) that contains the Property that is calling the Callback script. Therefore, the Callback script can use the PropHost's functions to query or set other properties in the same PropServer.

Definitions

<**propName**> = text string

<**value**> = double

<**valueText**> = text string

<**fileName**> = full path file name

<**choices**> = string containing menu choices separated by commas

<**initialChoice**> = string containing initial choice for menu; must be one of the <choices>

<**scriptName**> = string containing name of script stored in project

<**bool**> is 1 for true or 0 for false

<propType>	<propTypeName>	Property Description
0	TextProp	Text
1	MenuProp	Menu
2	CheckboxProp	Checkbox
3	VariableProp	Variable
4	VPointProp	VPoint
5	V3DPointProp	V3DPoint
6	NumberProp	Number
7	ColorProp	Color
8	PointProp	Point
9	ValueProp	Value
10	ButtonProp	Button
11	SeparatorProp	Separator

12	NetlistProp	Netlist
13	FileNameProp	FileName

<tabType>	Description	Objects with this tab type
0	DefaultTab	
1	PassedParameterTab	Instances of designs, components, geometric objects
2	DefinitionParameterTab	Definitions of designs, components
3	LocalVariableTab	Definitions of designs, components
4	ProjectVariableTab	Projects
5	ConstantsTab	Projects
6	BaseElementTab	Geometric objects
7	ComponentTab	Designs, components
8	PropertyTab	
9	CircuitTab	
10	SystemTab	
11	PlanarEMTab	
12	HfssTab	HFSS objects
13	OptimetricsTab	Optimetrics data
14	AltraSimTab	

15	Report3DTab	Report3d
16	FieldsPostProcessorTab	Fieldspostprocessor
17	MeshSetupTab	Manual meshing setup
18	RadFieldSetupTab	Radiation field geometry setup
19	Geometry3DAttributeTab	Geometry3D
20	Geometry3DCmdTab	Geometry3D
21	Geometry3DPolylineTab	Geometry3D
22	Geometry3DCSTab	Geometry3D
23	Geometry3DPlaneTab	Geometry3D
24	Geometry3DPointTab	Geometry3D
25	Geometry3DListTab	Geometry3D
26	StandardPropTab	
27	PropDisplayPropTab	
28	CustomPropTab	

ChangeProperty

Different forms of this command are documented for HFSS as well Schematic and Layout Editors.

Use: Changes to properties are scripted using the `ChangeProperty` command. This command can be executed by the `oEditor` to change editor properites, by the `oDesign` to change design level properties, and by the `oProject` to change project level properties. The command can be used to create, edit, and/or remove properties. In HFSS, only Variable and Separator properties can be deleted.

Use the script recording feature and edit a property, and then view the resulting script entry or use `GetPropertyValue` for the desired property to see the expected format.

Command: None

Syntax:

```
ChangeProperty  Array("Name:AllTabs", <PropTabArray>,
<PropTabArray>, ...)
ChangeProperty(<modulename>:<setup name>:<sweep name>)
```

Return Value: None

Parameters:

```
<PropTabArray>
   Array("Name:<PropTab>",
      <PropServersArray>,
      <NewPropsArray>,
      <ChangedPropsArray>,
      <DeletedPropsArray>)

<PropServersArray>
   Array("Name:PropServers", <PropServer>,
   <PropServer>, ...)

<NewPropsArray>
   Array("Name:NewProps", <PropDataArray>,
   <PropDataArray>,...)

<ChangedPropsArray>
   Array("Name:ChangedProps",<PropDataArray>,
   <PropDataArray>, ...)

<DeletedPropsArray>
   Array("Name:DeletedProps", <PropName>,
   <PropName>, ...)
```

```
<PropDataArray>
   Array("NAME:<PropName>",
      "PropType:=", <PropType>,
      "NewName:=", <string>,
      "Description:=", <string>,
      "NewRowPosition:=", <int>,
      "ReadOnly:=", <bool>,
      "Hidden:=", <bool>,
      <PropTypeSpecificArgs>)
```

`<PropType>`

Type: string

Identifies the type of property when a new property is added. In HFSS, only separator properties and variable properties can be added.

```
"SeparatorProp"
"VariableProp"
"TextProp"
"NumberProp"
"ValueProp"
"CheckboxProp"
"MenuProp"
"PointProp"
"VPointProp"
"V3DPointProp"
"ButtonProp"
```

`NewName`

Specify the new name of a property if the property's name is being edited. In HFSS, the name can only be changed for separators and variables.

`Description`

Specify a description of the property. In HFSS, the description can only be changed for separators and variables.

`NewRowPosition`

Used to reorder rows in the **Property** dialog box. In HFSS, this only applies to the **Project>Project Variables** panel and the **Hfss>Design Properties** panel. Specify the new zero-based row index of the variable or separator.

`ReadOnly`

Used to mark a property as "read only" so it can not be modified. In HFSS, this flag can only be set for variables and separators.

`Hidden`

Used to hide a property so it can not be viewed outside of the **Property** dialog box. In HFSS, this flag can only be set for variables and separators.

```
<PropTypeSpecificArgs>
   SeparatorProp: no arguments
   TextProp: "Value:=", <string>
   NumberProp: "Value:=", <double>
   ValueProp: "Value:=", <value>
   CheckboxProp: "Value:=", <bool>
   MenuProp: "Value:=", <string>
   PointProp"X:=", <double>, "Y:=", <double>
   VPointProp: "X:=", <value>, "Y:=", <value>
   V3DPointProp: "X:=",<value>, "Y:=",<value>,
      "Z:=",<value>
   Material Button: "Material:=", <string>
   Color Button: "R:=",<int>,"G:=",<int>,"B:=",<int>
   Transparency Button:"Value:=", <double>
```

`<PropTypeSpecificArgs>` for `VariableProps`

Syntax:

```
"Value:=", <value>, <OptimizationFlagsArray>,
<TuningFlagsArray>, <SensitivityFlagsArray>,
<StatisticsFlagsArray>
```

Parameters:

```
<OptimizationFlagsArray>
   Array("NAME:Optimization",
```

```
            "Included:=", <bool>,
            "Min:=", <value>,
            "Max:=", <value>)

      <Tuning flagsArray>
         Array("NAME:Tuning",
            "Included:=", <bool>,
            "Step:=", <value>,
            "Min:=", <value>,
            "Max:=", <value>)

      <SensitivityFlagsArray>
         Array("NAME:Sensitivity",
            "Included:=", <bool>,
            "Min:=", <value>,
            "Max:=", <value>,
            "IDisp:=", <value> )

      <StatisticsFlagsArray>
         Array("NAME:Statistical",
            "Included:=", <bool>,
            "Dist:=", <Distribution>,
            "StdD:=", <value>,
            "Min:=", <value>,
            "Max:=", <value>,
            "Tol:=", <string>)
```

`<Distribution>`
Type: string
Value should be `"Gaussian"` or `"Uniform"`

`StdD`
Standard deviation.

`Min`

Low cut-off for the distribution.

`Max`
High cut-off for the distribution.

`Tol`
Tolerance for uniform distributions. Format is `"<int>%"`.
Example: `"20%"`.

Example: Adding a new project level variable `"$width"`:

```
oProject.ChangeProperty Array("NAME:AllTabs",_
  Array("NAME:ProjectVariableTab",_
    Array("NAME:PropServers", "ProjectVariables"),_
    Array("NAME:NewProps",_
      Array("NAME:$width",_
        "PropType:=", "VariableProp",_
        "Value:=", "3mm",_
        "Description:=", "my new variable"))))
```

Example: Deleting the design level variable `"height"`:

```
oDesign.ChangeProperty Array("NAME:AllTabs",_
  Array("NAME:LocalVariableTab",_
    Array("NAME:PropServers", "DefinitionParameters"),_
    Array("NAME:DeletedProps", "height"))
```

Example: Changing a property's value. If the following command were executed, then the value of the property `"XSize"` of the `PropServer "Box1:CreateBox:1"` on the `"Geometry3DCmdTab"` tab would be changed. (`oEditor` is the `Geometry3D editor` in HFSS.)

```
oEditor.ChangeProperty Array("NAME:AllTabs",_
  Array("NAME:Geometry3DCmdTab",_
      Array("NAME:PropServers","Box1:CreateBox:1"),_
      Array("NAME:ChangedProps",_
        Array("NAME:XSize", "Value:=", "1.4mil"))))
```

Example: Changing the Company Name, Design Name, the background color, and the Axis scaling in a Report.

```
Set oProject = oDesktop.SetActiveProject("wgcombiner")
Set oDesign = oProject.SetActiveDesign("HFSSDesign2")
Set oModule = oDesign.GetModule("ReportSetup")
oModule.ChangeProperty Array("NAME:AllTabs", Array("NAME:Header", _
Array("NAME:PropServers", "XY Plot1:Header"), _
Array("NAME:ChangedProps", Array("NAME:Company Name", _
"Value:=", "My Company"))))
oModule.ChangeProperty Array("NAME:AllTabs", Array("NAME:Header",_
Array("NAME:PropServers", "XY Plot1:Header"), _
Array("NAME:ChangedProps", Array("NAME:Design Name", _
"Value:=", "WG Combiner"))))
oModule.ChangeProperty Array("NAME:AllTabs", Array("NAME:General",_
Array("NAME:PropServers", "XY Plot1:General"), _
Array("NAME:ChangedProps", Array("NAME:Back Color", _
"R:=", 128, "G:=", 255, "B:=", 255))))
oModule.ChangeProperty Array("NAME:AllTabs", Array("NAME:Axis", _
Array("NAME:PropServers", "XY Plot1:AxisX"), _
Array("NAME:ChangedProps", Array("NAME:Axis Scaling", _
"Value:=", "Log"))))
```

For Schematic Editor and Layout Editor, the ChangeProperty command details are as follows:

Use: Changes to properties are scripted using the **ChangeProperty** command. This command can be executed by the **oEditor** to change editor properties, by the **oDesign** to change design level properties, and by the **oProject** to change project level properties. The command can be used to create, edit, and/or remove properties. In Designer, only Variable and Separator properties can be deleted.

Command: None

Syntax: `ChangeProperty Array("Name:AllTabs", <PropTabArray>, <PropTabArray>, ...)`

Return Value: None

Parameters:

```
<PropTabArray>
   Array("Name:<PropTab>",
   <PropServersArray>,
   <NewPropsArray>,
   <ChangedPropsArray>,
```

```
   <DeletedPropsArray>)

<PropServersArray>
   Array("Name:PropServers", <PropServer>,
<PropServer>, ...)

<NewPropsArray>
   Array("Name:NewProps", <PropDataArray>,
<PropDataArray>,...)

<ChangedPropsArray>
   Array("Name:ChangedProps",<PropDataArray>,
<PropDataArray>, ...)

   <DeletedPropsArray>
   Array("Name:DeletedProps", <PropName>,
<PropName>, ...)
OR (for PropDisplay deletions only)
   Array("Name:DeletedProps",<PropDataArray>,
<PropDataArray>, ...)


<PropDataArray>
Array("NAME:<PropName>",
     "PropType:=", <PropType>,
     "NewName:=", <string>,
     "Description:=", <string>,
     "Callback:=", <string>,
     "NewRowPosition:=", <int>,
     "ReadOnly:=", <bool>,
     "Hidden:=", <bool>,
<PropTypeSpecificArgs>)OR (for PropDisplays only)
Array("Name:<PropName>",<PropDisplayData>)

<PropDisplayData>
```

for adding, changing, deleting PropDisplays

<PropDisplayAttributes>

for changing PropDisplays only

<PropDisplayNewAttributes>

<PropDisplayAttributes>

Layer & Location only used for PropDisplays in layout

For adding PropDisplays, this will add a single PropDisplay with attributes as shown; if an attribute is missing, a default value will be assigned. Adding PropDisplay to schematic with attributes that are identical to one already existing there will fail without an error message.

For deleting PropDisplays, these attributes are used to identify an existing PropDisplay to delete. If there doesn't exist a PropDisplay that matches the given attributes, then nothing will be deleted. If multiple PropDisplays match the given attributes, then all of them will be deleted. If an attribute is missing, then all PropDisplays match that missing attribute. For example, if Layer is missing, then PropDisplays on all layers that match the remaining given attributes will be deleted.

For changing PropDisplays, these attributes are used to identify an existing PropDisplay to change. If no PropDisplay matching the attributes is found, no changes will be made. If multiple PropDisplays match the attributes, all of them will be changed. If an attribute is missing, it matches all PropDisplays. For example, to change the format of PropDisplays that are on the bottom, but have any layer, style or format to show the name only, this command should have Location set to "Bottom" and all other attributes omitted.

"Format:=", <PropDisplayType>,

"Location:=", <PropDisplayLocation>,

"Layer:=", <string>,

"Style:=", <string>

<PropDisplayNewAttributes>

NewLayer & NewLocation only used for PropDisplays in layout

For changing PropDisplays, these attributes are used to identify which attributes to change and what the new value is. If the attribute should not be changed, the corresponding entry should be omitted.

"NewName:=", <string>,
"NewFormat:=", <PropDisplayType>,
"NewLocation:=", <PropDisplayLocation>,
"NewLayer:=", <string>,
"NewStyle:=", <string>

<PropDisplayType>
  Type: string
Identifies the format of PropDisplay.
"Name"
"Value"
"NameAndValue"
"EvaluatedValue"
"NameAndEvaluatedValue"

<PropDisplayLocation>
  Type: string
Identifies where PropDisplay is located with respect to object

"Left"
"Top"
"Right"
"Bottom"
"Custom"

<PropType>
Type: string
Identifies the type of property when a new property is added. In Designer, only separator properties and variable properties can be added.

"SeparatorProp"
"VariableProp"
"TextProp”
"NumberProp"

"ValueProp"

"CheckboxProp"

"MenuProp”

"PointProp"

"VPointProp"

"ButtonProp"

NewName

Specify the new name of a property if the property’s name is being edited. In Designer, the name can only be changed for separators and variables.

Description

Specify a description of the property. In Designer, the description can only be changed for separators and variables.

Callback

Specify the name of the script callback to be run when the property value is changed.

NewRowPosition

Used to reorder rows in the Property dialog box. In Designer, this only applies to the Project>Project Variables panel and the Designer>Design Properties panel. Specify the new zero-based row index of the variable or separator.

ReadOnly

Used to mark a property as “read only” so it can not be modified. In Designer, this flag can only be set for variables and separators.

Hidden

Used to hide a property so it can not be viewed outside of the Property dialog box. In Designer, this flag can only be set for variables and separators.

<PropTypeSpecificArgs>

SeparatorProp: no arguments

TextProp: “Value:=”, <string>

NumberProp: “Value:=”, <double>

ValueProp: “Value:=”, <value>

```
CheckboxProp: “Value:=”, <bool>
MenuProp: “Value:=”, <string>
PointProp“X:=", <double>, "Y:=", <double>
VPointProp: "X:=", <value>, "Y:=", <value>
Material Button: “Material:=”, <string>
Color Button: "R:=",<int>,"G:=",<int>,"B:=",<int>
Transparency Button:“Value:=”, <double>
```

<PropTypeSpecificArgs> for MenuProps

Syntax for NewProps array: "AllChoices:=",

<"choice1,choice2,..."> or <Array("choice1" "choice2", ...)>,

"Value:=", <string>

Syntax for ChangedProps array: "Value:=", <string>

<PropTypeSpecificArgs> for VariableProps

Syntax:

“Value:=”, <value>, <OptimizationFlagsArray>, <TuningFlagsArray>, <SensitivityFlagsArray>, <StatisticsFlagsArray>

Parameters:

<OptimizationFlagsArray>

```
Array("NAME:Optimization",
   "Included:=", <bool>,
   "Min:=", <value>,
   "Max:=", <value>)
```

<TuningFlagsArray>

```
Array("NAME:Tuning",
   "Included:=", <bool>,
   "Step:=", <value>,
   "Min:=", <value>,
   "Max:=", <value>)
```

```
   <SensitivityFlagsArray>
Array("NAME:Sensitivity",
   "Included:=", <bool>,
   "Min:=", <value>,
   "Max:=", <value>,
   "IDisp:=", <value> )

<StatisticsFlagsArray>
Array("NAME:Statistical",
   "Included:=", <bool>,
   "Dist:=", <Distribution>,
   "StdD:=", <value>,
   "Min:=", <value>,
   "Max:=", <value>,
   "Tol:=", <string>)

<Distribution>
```

Type: string
Value should be "Gaussian" or "Uniform"

`StdD`
Standard deviation.

`Min`
Low cut-off for the distribution.

`Max`
High cut-off for the distribution.

`Tol`
Tolerance for uniform distributions. Format is "<int>%".
Example: "20%".

Example: `Adding a new project level variable "$width":`

```
oProject.ChangeProperty Array("NAME:AllTabs",_
Array("NAME:ProjectVariableTab",_
Array("NAME:PropServers", "ProjectVariables"),_
Array("NAME:NewProps",_
Array("NAME:$width",_
"PropType:=", "VariableProp",_
"Value:=", "3mm",_
"Description:=", "my new variable"))))
```

Example:

```
Deleting the design level variable "height":
oDesign.ChangeProperty Array("NAME:AllTabs",_
Array("NAME:LocalVariableTab",_
Array("NAME:PropServers", "DefinitionParameters"),_
Array("NAME:DeletedProps", "height"))
```

Example:

```
Changing a property's value. If the following command
were executed, then the value of the property "XSize" of
the PropServer "Box1:CreateBox:1" on the
"Geometry3DCmdTab" tab would be changed. (oEditor is the
Geometry3D editor in Designer.)
oEditor.ChangeProperty Array("NAME:AllTabs",_
Array("NAME:Geometry3DCmdTab",_
Array("NAME:PropServers","Box1:CreateBox:1"),_
Array("NAME:ChangedProps",_
Array("NAME:XSize", "Value:=", "1.4mil"))))
```

Example:

```
Changing a property's value.  If the following command
were executed, then the values of Callback and L on the
PassedParameterTab would be changed.
```

Example:

```
oEditor.ChangeProperty Array("NAME:AllTabs", _
Array("NAME:PassedParameterTab", _
Array("NAME:PropServers",  "CHOKE2"), _
Array("NAME:ChangedProps", _
Array("NAME:L", "Callback:=", "ac", "OverridingDef:=",
true), _
Array("NAME:L", "Value:=", "1nH"))))
```

Example: Changing the Company Name, Design Name, the background color, and the Axis scaling in a Report.

```
Set oProject = oDesktop.SetActiveProject("wgcombiner")
Set oDesign = oProject.SetActiveDesign("DesignerDesign2")
Set oModule = oDesign.GetModule("ReportSetup")
oModule.ChangeProperty Array("NAME:AllTabs", Array("NAME:Header", _
Array("NAME:PropServers", "XY Plot1:Header"), _
Array("NAME:ChangedProps", Array("NAME:Company Name", _
"Value:=", "My Company"))))
oModule.ChangeProperty Array("NAME:AllTabs", Array("NAME:Header",_
Array("NAME:PropServers", "XY Plot1:Header"), _
Array("NAME:ChangedProps", Array("NAME:Design Name", _
"Value:=", "WG Combiner"))))
oModule.ChangeProperty Array("NAME:AllTabs", Array("NAME:General",_
Array("NAME:PropServers", "XY Plot1:General"), _
Array("NAME:ChangedProps", Array("NAME:Back Color", _
"R:=", 128, "G:=", 255, "B:=", 255))))
oModule.ChangeProperty Array("NAME:AllTabs", Array("NAME:Axis", _
Array("NAME:PropServers", "XY Plot1:AxisX"), _
Array("NAME:ChangedProps", Array("NAME:Axis Scaling", _
"Value:=", "Log"))))
```

Example: Changing a property's value. Note that the AllChoices parameter is only used when the MenuProp is being added. Also note that either a string of choices separated by commas or an Array("choice1", "choice2", "choice3") works for the AllChoices parameter.

```
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
oEditor.ChangeProperty Array("NAME:AllTabs",
Array("NAME:PassedParameterTab", Array("NAME:PropServ-
ers",  _
  "CompInst@CAP_;2"), Array("NAME:NewProps",
Array("NAME:xxxx", "PropType:=",  _
  "MenuProp", "AllChoices:=", Array("aa", "bb", "cc", "dd"),
"Value:=", "bb"))))
```

PropHost Functions

Following commands can be used to manipulate properties from a Property script.

Abort

Use: Aborts the specified design simulation.

Command: None

Syntax: `Abort(<designName>)`

Return Value: String

Example:
```
a = PropHost.Abort(<designName>);
```

AddMenuProp

Use: Creates a new Menu property in tabType with name specified; choices are set to the values in choices; initial selection is initialChoice.

Command: None

Syntax:
```
AddMenuProp(<tabType>, <propName>, <choices>,
<initialChoice>)
```

Return Value: None.

Example:
```
PropHost.AddProp(2, "ResChoices",
"inline,upfront,parallel,series", "parallel"); creates a
new MenuProp in the DefinitionParameters tab named
ResChoices with choices inline, upfront, parallel, and
series. The initial choice shown in the Menu is
"parallel"
```

AddMenuProp2

Use: Creates a new Menu property in tabTypeName with name specified; choices are set to the values in choices; initial selection is initialChoice.

Command: None

Syntax:
```
AddMenuProp2(<tabTypeName>, <propName>, <choices>,
<initialChoice>)
```

Return Value: None.

Example:
```
PropHost.AddMenuProp2("DefinitionParameterTab",
"ResChoices", "inline,upfront,parallel,series",
"parallel"); creates a new MenuProp in the
DefinitionParameters tab named ResChoices with choices
inline, upfront, parallel, and series. The initial choice
shown in the Menu is "parallel".
```

AddProp

Use: Creates a new propType property in tabType with name and value specified.

Command: None

Syntax:
```
AddProp(<tabType>, <propType>, <propName>, <valueText>)
```

Return Value: None.

Example:
```
PropHost.AddProp(2, 3, "W1", "10mm"); creates a new
VariableProp in the DefinitionParameters tab named W1
with value 10mm.
```

Note

AddProp adds a new property using the <propName> preceded by a separator "--". For instance, a new property added with the <propName> "xxx" will be added with the name "--xxx". Consequently, all subsequent script functions that wish to access the added property must now use the name "--xxx".

For example:

PropHost.AddProp 1, 11, "Time_Domain_Options",""

Prophost.SetHidden "--Time_Domain_Options",1

AddProp2

Use: Creates a new propTypeName property in tabTypeName with name and value specified.

Command: None

Syntax:
```
AddProp2(<tabTypeName>, <propTypeName>, <propName>,
<valueText>)
```

Return Value: None.

Example:
```
PropHost.AddProp2(("DefinitionParameterTab",
"VariableProp", "W1", "10mm"); creates a new VariableProp
in the DefinitionParameters tab named W1 with value 10mm.
```

ExecuteScript

Use: Finds the named script in the Definitions/Scripts folder and runs that script; the script being run can also use the PropHost object.

Command: None

Syntax:
```
ExecuteScript(<scriptName>)
```

Return Value: None.

Example: `PropHost.ExecuteScript("PropChangeScript"); runs the script named PropChangeScript`

GetApplication

Use: Returns the application object currently running the script.

Command: None

Syntax: `GetApplication()`

Return Value: Application object.

Example: `Set a = PropHost.GetApplication(); returns currently running application.`

GetCallback

Use: Finds named property and returns name of Callback script.

Command: None

Syntax: `GetCallback(<propName>)`

Return Value: String

Example: `a = PropHost.GetCallback( "W1"); returns "SynchronizeResistors"`

GetChangedProperty

Use: If the script was called by a Callback associated with a property, this function returns the name of that property.

Command: None

Syntax: `GetChangedProperty()`

Return Value: String

Example: `pn = PropHost.GetChangedProperty(); returns "C" if the script was a Callback associated with the property named "C" and the script was called in response to the property "C" changing value.`

GetDescription

Use: Finds named property and returns description string.

Command: None

Syntax: `GetDescription(<propName>)`

Return Value: String

Example: `a = PropHost.GetDescription( "W1"); returns "this is the width of the gate"`

GetDesign

Use: Returns the interface to the specified design simulation.

Command: None

Syntax: `GetDesign <DesignName>`

Return Value: Interface to the specified design simulation.

Parameters: `<DesignName>`

Type: <string>

Example: `Set oDesign = oPropHost.GetDesign ("DesignerModel1")`

GetEditor

Use: Returns an interface to the editor requested IF the PropServer behind the PropHost is contained within that type of editor.

Command: None

Syntax: `GetEditor(<editorName>)`

Return Value: String

Example: `Set oLayout2 = PropHost.GetEditor("Layout"); returns the interface to the layout containing a selected component. This interface can be used to call Layout Scripting functions.`

GetEvaluatedText

Use: Finds the propName and returns its value. If the value is an expression, GetEvaluatedText evaluates and returns the value of the expression.

Command: None

Syntax: `GetEvaluatedText(<propName>)`

Return Value: String

Example: `a = PropHost. GetEvaluatedText ("bitPattern");`

`If bitPattern= "10100101" then a = "10100101".`

`If bitPattern= "bitPatterns[1]", where bitPatterns is an array variable (bitPatterns = ["10111"."0000"], then a = "0000".`

GetFileName

Use: Finds the full path name to propName.

Example: None

Syntax: GetFileName(<propName>)

Return Value: String.

Example:

```
a = PropHost.GetFileName("SubstrateFile");
```

Returns the full pathname associated with the ButtonProp “SubstrateFile”.

If SubstrateFile=$projectdir/info.txt, then a = "c:/data/info.txt" (if the project directory is set to c:/data).

NOTE: Directory variables can be used in the property's value (e.g. $projectdir, $userlib, $syslib, $personallib) and these will be expanded to the correct path. GetFileName always returns a path string; if propName actually contains a variable expression, that expression is evaluated to a constant string before returning.

GetHidden

Use: Finds named property and returns its Hidden flag.

Command: None

Syntax: GetHidden(<propName>)

Return Value: Returns 1 if property is hidden and 0 if it is not.

Example:

```
a = PropHost.GetHidden( "W1"); returns 1
```

GetProgress

Use: Returns the completion percentage (from 0 to 100) of the specified design simulation.

Command: None

Syntax: GetProgress(<designName>)

Return Value: String

Example:

```
a = PropHost.GetProgress(<designName>);
```

GetPropHost

Use: Returns the PropServer (owner of properties) of the propTypeName that contains the Property that is calling the Callback script.

Command: None

Syntax: `GetPropHost(<propTypeName>)`

Return Value: Returns string.

Example: `objects = PropHost.GetPropHost("VariableProp");`

GetPropServers

Use: Returns array of objects that have properties showing on tabTypeName.

Command: None

Syntax: `GetPropServers(<tabTypeName>)`

Return Value: Returns string.

Example:
```
objects = PropHost.GetPropServers("PassedParameterTab");
returns array containing PropServers that have properties
shown on PassedParameterTab; this would include only
components and designs; individual properties can be
accessed using standard notation, e.g. objects(0) might
contain "CompInst@CAP_;1".
```

GetPropTabType

Use: Finds named property and returns the id of the tab it is in.

Command: None

Syntax: `GetPropTabType(<propName>)`

Return Value: Returns string.

Example:
```
a = PropHost.GetPropTabType( "W1"); returns 2 for
property W1 since it is on the DefinitionParams tab
```

GetReadOnly

Use: Finds named property and returns its ReadOnly flag.

Command: None

Syntax: `GetReadOnly(<propName>)`

Return Value: Returns 1 if property is read-only and 0 if it is not.

Example: `a = PropHost.GetReadOnly( "W1"); returns 1`

GetRunStatus

Use: Returns the status number of the specified design simulation.

Command: None

Syntax: `GetRunStatus(<statusNumber>)`

Return Value: Returns string.

Example:
```
a = PropHost.GetRunStatus(<statusNumber>);
```

GetProgress

Use: Returns the percentage (from 0 to 100) of the simulation completed.

Command: None

Syntax: `GetProgress(<simProgress>)`

Return Value: String

Example:
```
a = PropHost.GetDesign(<simProgress> );
```

GetTabTypeName

Use: Finds named property and returns the name of the tab it is on.

Command: None

Syntax: `GetTabTypeName(<propName>)`

Return Value: Returns string.

Example:
```
a = PropHost.GetTabTypeName( "W1"); returns
"DefinitionParameterTab" for property W1 since it is on
the DefinitionParams tab.
```

GetText

Use: Finds property in any tab and returns its value as a text string.

Command: None

Syntax: `GetText(<propName>)`

Return Value: Returns string.

Example:
```
a = PropHost.GetText("C"); a contains "13pF"
```

GetValue

Use: Finds property in any tab and returns its value as a double.

Command: None

Syntax: `GetValue(<propName>)`

Return Value: Returns double.

Example:
```
a = PropHost.GetValue("C") ;
```

Note Values are returned in SI units. Compound SI units are, in general, not supported. Temperature values are returned in Celcius

IsValueConstant

Use: Returns True if propName is evaluated to be a constant; returns False if propName is evaluated to be an expression.

Command: None

Syntax: `IsValueConstant(<propName>)`

Return Value: Boolean.

Example:
```
a = PropHost.IsValueConstant("C");
If C=x+1 then a = 0.
If C=2pF then a=1.
```

PropertyExists

Use: Finds named property and returns its property type.

Command: None

Syntax: `PropertyExists(<propName>)`

Return Value: Returns 1 if property exists in any tab, 0 if it does not.

Example:
```
a = PropHost.PropertyExists( "W1"); returns 1 since this
property is present on DefinitionParams tab
```

RemoveProp

Use: Removes the named property from whichever tab it is found.

Command: None

Syntax: `RemoveProp(<propName>)`

Return Value: None.

Example: `PropHost.RemoveProp("W1"); removes the property named W1 from whatever tab it is in`

SetCallback

Use: Finds named property and sets its Callback script.

Command: None

Syntax: `SetCallback(<propName>, <scriptName>)`

Return Value: None.

Example: `PropHost.SetCallback( "W1", "SynchronizeResistors"); sets the Callback script for property W1 to SynchronizeResistors`

SetDescription

Use: Finds named property and sets its description text.

Command: None

Syntax: `SetDescription(<propName>, <valueText>)`

Return Value: None.

Example: `PropHost.SetDescription( "W1", "this is the width of the gate"); sets the description for property W1 to "this is the width of the gate"`

SetHidden

Use: Finds named property and sets its Hidden flag.

Command: None

Syntax: `SetHidden(<propName>, <bool>)`

Return Value: None.

Example: `PropHost.SetHidden( "W1", 1); makes property W1 invisible in Property Window`

SetReadOnly

Use: Finds named property and sets its ReadOnly flag.

Command: None

Syntax: `SetReadOnly(<propName>, <bool>)`

Return Value: None.

Example: `PropHost.SetReadOnly( "W1", 1); makes property W1 read-only`

SetText

Use: Finds property in any tab and sets its value to a text string.

Command: None

Syntax: `SetText(<propName>, <valueText>)`

Return Value: None.

Example: `PropHost.SetText("C", "22nF"); sets C to 22nF`

SetValue

Use: Finds property in any tab and sets its value to a double.

Command: None

Syntax: `SetValue(<propName>, <value>)`

Return Value: None.

Example: `PropHost.SetValue("C", 2e-9); sets C to 2e-9`

Additional Property Scripting Commands

Following are other commands that can be used to manipulate properties from a script.

GetProperties

Use: Gets a list of all the properties belonging to a specific `PropServer` and `PropTab`. This can be executed by the `oProject`, `oDesign`, or `oEditor` variables.

Command: None

Syntax: `GetProperties( <PropTab>, <PropServer> )`

`GetProperties(<modulename>:<setup name>:<sweep name>)`

Return Value: Variant array of strings – the names of the properties belonging to the prop server.

Example:

```
Dim all_props
all_props = oDesign.GetProperties("HfssTab",_
   "BoundarySetup:WavePort1")
```

Example:

```
Dim all_props
all_props = oDesign.GetProperties(“BaseElementTab”,_
"rect_1")
```

Example:

```
Dim all_props
all_props = oDesign.GetProperties("Q3DTab",_
"BoundarySetup:Source1")
```

GetPropertyValue

Use: Gets the value of a single property. This can be executed by the `oProject`, `oDesign`, or `oEditor` variables.

Use the script recording feature and edit a property, and then view the resulting script to see the format for that property.

Command: None

Syntax:

```
GetPropertyValue(<PropTab>, <PropServer>, <PropName>)
GetPropertyValue(<modulename>:<setup name>:<sweep name>)
```

Return Value: String representing the property value.

Example:

```
value_string = _
oEditor.GetPropertyValue("Geometry3DCmdTab",_
   "Box1:CreateBox:1", "XSize")
```

Example:

```
value_string = _
oEditor.GetPropertyValue(“BaseElementTab”,_
“rect_1”, “Name”)
```

GetArrayVariables

Use: Returns a list of array variables. To get a list of indexed Project variables, execute this command using `oProject`. To get a list of indexed local variables, use `oDesign`.

Syntax:

```
GetArrayVariables()
```

Return Value: Variant array of strings – the names of the array variables.

Example:

```
Dim var_array
project_var_array = oProject.GetArrayVariables()
local_var_array = oDesign.GetArrayVariables()
```

GetVariables

Use: Returns a list of all variables. To get a list of non-indexed Project variables, execute this command using `oProject`. To get a list of non-indexed local variables, use `oDesign`.

Syntax: `GetVariables()`

Return Value: Variant array of strings – the names of the variables.

Example:

```
Dim var_array
project_var_array = oProject.GetVariables()
local_var_array = oDesign.GetVariables()
```

GetVariableValue

Use: Gets the value of a single variable. To get the value of Project variables, execute this command using `oProject`. To get the value of local variables, use `oDesign`.

Command: None

Syntax: `GetVariableValue(<VarName>)`

Return Value: A string representing the value of the variable.

Parameters: `<VarName>`

Type: string

Name of the variable to access.

Example:

```
project_var_value_string = oProject.GetVariableValue("var_name")
```

Example:

```
local_var_value_string = oDesign.GetVariableValue("var_name")
```

SetPropertyValue

Use: Sets the value of one property. This is not supported for properties of the following types: `ButtonProp, PointProp, V3DPointProp`, and `VPointProp`. Only the `ChangeProperty` command can be used to modify

these properties. This can be executed by the `oProject`, `oDesign`, or `oEditor` variables.

Use the script recording feature and edit a property, and then view the resulting script entry or use `GetPropertyValue` for the desired property to see the expected format.

Command: None

Syntax: `SetPropertyValue <PropTab>, <PropServer>, <PropName>, <PropValue>`

Return Value: None

Parameters: `<PropValue>`

Type: String

Contains the value to set the property. The formatting is different depending on what type of property is being edited.

Example:

```
oEditor.SetPropertyValue _
"Geometry3DCmdTab","Box1:CreateBox:1",_
   "XSize", "3mm"
```

Example:

```
oEditor.SetPropertyValue _
"BaseElementTab","rect_1",_
"LineWidth", "3mm"
```

SetVariableValue

Use: Sets the value of a variable. To set the value of a Project variable, execute this command using `oProject`. To set the value of a local variable, use `oDesign`.

Syntax: `SetVariableValue <VarName>, <VarValue>`

Return Value: None

Parameters: `<VarValue>`

Type: <value>

New value for the variable.

Example:

```
oProject.SetVariableValue "$Var1", "3mm"
```

Example:

```
var_value = "2Ohm"
oDesign.SetVariableValue "Var2", var_value
```

Additional Property Scripting Example

Following is a sample script that uses the `GetPropertyValue`, `SetPropertyValue`, and `GetProperties` functions. The script gets all the properties of the first `CreateBox` command of `"Box1"`. It then loops through the properites and for each one, shows the user the current value and asks if the value should be changed.

Example:

```
Dim all_props
Dim prop
all_props = oEditor.GetProperties("Geometry3DCmdTab",_
   "Box1:CreateBox:1")
For Each prop In all_props
   val = oEditor.GetPropertyValue("Geometry3DCmdTab",_
      "Box1:CreateBox:1", prop)
   new_val = InputBox("New Value of " + prop + ":",_
      "Current Value of '" + prop + "' is " + val, val)
   If new_val <> val Then
        oEditor.SetPropertyValue "Geometry3DCmdTab",_
          "Box1:CreateBox:1", prop, new_val
        val = _
          oEditor.SetPropertyValue("Geometry3DCmdTab",_
          "Box1:CreateBox:1", prop)
        MsgBox("Now the value of '" + prop + "' is " + val)
   End If
Next
```

The following is a sample script that creates a PlanarEM design, draws a rectangle in the layout editor and uses the `GetPropertyValue`, `SetPropertyValue` and `GetProperties` functions. The script gets all properties of the rectangle. It then loops through the properties and for each one, shows the user the current value and asks if the value should be changed. Note that the last call to `GetPropertyValue` in the script will fail if you change the name of the rectangle from the script.

```
-----------------------------------------------------------------
' Script Recorded by Designer/Nexxim
```

```
' 8:10 AM  Dec 05, 2003
'----------------------------------------------------------------
```

Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsysDesigner.Design-
erScript")
Set oDesktop = oAnsoftApp.GetAppDesktop()
'
oDesktop.RestoreWindow
oDesktop.NewProject
Set oProject = oDesktop.GetActiveProject
'
' CREATE A RECTANGLE IN PLANAR_EM
'
oProject.InsertDesign "Planar EM", "PlanarEM1",  _
  "C:\testinstall\Designer\syslib\PCB - Single-
Sided.asty", ""
Set oDesign = oProject.SetActiveDesign("PlanarEM1")
Set oEditor = oDesign.SetActiveEditor("Layout")
oEditor.CreateRectangle Array("NAME:Contents", _
       "rectGeometry:=", Array("Name:=",  _
       "rect_1", "LayerName:=", "Top", "lw:=", _
       "0mm", "Ax:=", "-22mm", "Ay:=", "20mm", "Bx:=",  _
       "29mm", "By:=", "-4mm", "ang:=", "0deg"))
'
' GET ALL PROPERTIES OF THE RECTANGLE
'
Dim all_props
Dim prop
Dim val
Dim new_val
```

```
'
all_props = oEditor.GetProperties("BaseElement-
Tab","rect_1")
'
' LOOP OVER ALL PROPERTIES
'
For Each prop in all_props
  val = oEditor.GetPropertyValue("BaseElement-
Tab","rect_1",prop)
'
' DISPLAY VALUE TO THE USER
'
  new_val = InputBox("New Value of "+prop+":",_
                     "Current Value of "+prop+" is
"+val,val)
'
' CHANGE THE VALUE IF DESIRED
'
  If new_val <> val Then
oEditor.SetPropertyValue _
   "BaseElementTab","rect_1",prop,new_val
    val = _
       oEditor.GetPropertyValue("BaseElement-
Tab","rect_1",prop)

    MsgBox("Now the value of "+prop+" is "+val)
  End If
'
Next
'
```

Example Use of Record Script and Edit Properties

A simple way to see how to format the string arguments for a design object or property of interest is to use the script recording command in HFSS or 2D Extractor, and then to edit the property. Open the script file and look at the o.Editor.ChangeProperty entry to see the string arguments.

```
' ----------------------------------------------
```

```
' Script Recorded by Ansoft HFSS Version 10.0
' 2:44 PM  Nov 18, 2005
' ------------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("wg_combiner")
Set oDesign = oProject.SetActiveDesign("HFSSModel1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.ChangeProperty Array("NAME:AllTabs", Array("NAME:Geometry3DAttribu-
teTab", Array("NAME:PropServers",  _
  "Polyline1"), Array("NAME:ChangedProps", Array("NAME:Display Wireframe",
"Value:=", true), Array("NAME:Display Wireframe", "Value:=",  _
  false), Array("NAME:Transparent", "Value:=", 0.2))))
```

8 Dataset Script Commands

Dataset commands should be executed by the `oProject` object:

```
Set oProject = oDesktop.SetActiveProject("Project1")
oProject.CommandName <args>
```

AddDataSet
DeleteDataSet
EditDataSet
ImportDataSet

AddDataset

Use: Adds a dataset.

Command: **Project>Datasets>Add**

Syntax:
```
AddDataset <DatasetDataArray>
```

Return Value: None

Parameters:
```
<DatasetDataArray>
   Array("NAME:<DatasetName>",
      Array("NAME:Coordinates", <CoordinateArray>,
      <CoordinateArray>, ...)
```

`<DatasetName>`
Type: <string>
Name of the dataset.

```
<CoordinateArray>
   Array("NAME:Coordinate",
      "X:=", <double>, "Y:=", <double>)
```

Example:
```
oProject.AddDataset Array("NAME:ds1",_
   Array("NAME:Coordinates",_
      Array("NAME:Coordinate", "X:=", 1, "Y:=", 2,_
      Array("NAME:Coordinate", "X:=", 3, "Y:=", 4),_
      Array("NAME:Coordinate", "X:=", 5, "Y:=", 7),_
      Array("NAME:Coordinate", "X:=", 6, "Y:=", 20)))
```

DeleteDataset

Use: Deletes the specified dataset.

Command: **Project>Datasets>Remove**

Syntax:
```
DeleteDataset <DatasetName>
```

Return Value: None

EditDataset

Use: Modifies a dataset. When a dataset is modified, its name as well as its data can be changed.

Command: **Project>Datasets>Edit**

Syntax: `EditDataset <OriginalName> <DatasetDataArray>`

Return Value: None

Parameters: `<OriginalName>`

Type: <string>

Name of the dataset before editing.

Example:

```
oProject.EditDataset "ds1" Array("NAME:ds2",_
  Array("NAME:Coordinates",_
    Array("NAME:Coordinate", "X:=", 1, "Y:=", 2),_
    Array("NAME:Coordinate", "X:=", 3, "Y:=", 4)))
```

ImportDataset

Use: Imports a dataset from a named file.

Command: Project>Dataset...>Import

Syntax: `ImportDataset <path>`

Return Value: None

Parameters: `<Path>`

Type: <string>

Path of the dataset.

Example:

```
Set oProject = oDesktop.SetActiveProject("OptimTee")
oProject.ImportDataset "F:/work/BSR_1_dk.tab"

Where BSR_1_dk.tab contains a numbered list of tab separated data:

      1   18.5
      2   18.5
      3   18.5
      4   18.5
      5   18.5
      6   18.5
      ...
```

In the case of Q3D Extractor, the **ImportDataset** command details are as follows.

Use: Imports a dataset.

Command: **Project>Datasets>Import**

Syntax: `ImportDataset <DatasetFileFullPath>`

Return Value: None

Parameters:

```
<DatasetFileFullPath>
   Type: <string>
   The full path to the file containing the dataset val-
   ues.
```

Example:

```
oProject.ImportDataset ("e:\tmp\dsdata.txt")
```

9 Design Object Script Commands

Design object commands should be executed by the `oDesign` object.

```
oDesign.CommandName <args>
```

Another example is as follows:

```
Set oDesign = oProject.SetActiveDesign("Q3DExtractorDe-
sign1")
oDesign.CommandName <args>
```

Conventions Used in this Chapter

`<ModuleName>`

Name used to access one of the following modules:

- Boundary module: `"BoundarySetup"`
- Mesh Operations module: `"MeshSetup"`
- Analysis module: `"AnalysisSetup"`
- Optimetrics module: `"Optimetrics"`
- Solutions module: `"Solutions"`
- Field Overlays module: `"FieldsReporter"`
- Radiation module: `"RadField"`

DeleteVariation
DeleteFieldVariation
DeleteFullVariation
DeleteLinkedDataVariation
ExportConvergence
ExportMatrixData
ExportMatrixData(2D Extractor)
ExportMeshStats
ExportProfile
ExportNetworkData
ExportNMFData
GetEdit SourcesCount
GetExcitations
GetModule
GetName
GetNominalVariation
GetSelections
GetSolutionType
GetSolveInsideThreshold
GetSourceContexts
GetVariationVariableValue
Redo
RenameDesignInstance
ResetToTimeZero
SARSetup
SetActiveEditor
SetBackgroundMaterial
SetDesignSettings
SetLengthSettings
SetSolutionType
SetSolveInsideThreshold
SetSourceContexts
Solve
RunToolkit
Undo

ApplyMeshOps

Use: If there are any mesh operations that were defined and not yet performed in the current variation for the specified solution setups, they will be applied to the current mesh. If necessary, an initial mesh will be computed first. No further analysis will be performed.

Command: **HFSS>Analysis Setup>Apply Mesh Operations**

Syntax: `ApplyMeshOps <SetupNameArray>`

Return Value: <SetupNameArray>

Type: <int>

-1: completed with error

0: completed successfully

Example:

```
status = oDesign.ApplyMeshOps Array("Setup1","Setup2")
```

For Q3D Extractor the command is as follows:

Command: **Q3D Extractor or 2D Extractor>Analysis Setup>Apply Mesh Operations**

Analyze

Use: Solves a single solution setup and all of its frequency sweeps.

Command: Right-click a solution setup in the project tree, and then click **Analyze** on the shortcut menu.

Syntax: `Analyze(<SetupName>)`

Parameters: `<setupName>`

Return Value: None

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 12.0.0
' 2:54 PM  Dec 15, 2008
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
```

```
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Tee")
Set oDesign = oProject.SetActiveDesign("TeeModel")
oDesign.Analyze "Setup1"
```

AnalyzeDistributed

Use: Perform a distributed analysis.

Command: None

Syntax: `AnalyzeDistributed <SetupName>`

Return Value: <AnalysisStatus>

Type: <int>

-1: completed with error

0: completed successfully

Parameters: `<SetupName>`

Example:

```
For frequency sweeps:
oDesign.AnalyzeDistributed "Setup1"
```

AssignDCThickness

Use: Assign DC Thickness to more accurately compute DC resistance of a thin conducting object for which Solve Inside is not selected.

Command: **HFSS>Assign DC Thickness**

Syntax: `AssignDCThickness Array(<ObjectName>) Array (<ThicknessValue>) <string>`

Return Value: None

Parameters: `<ObjectName>`

Type: <string>

Array of object names. Any objects not specified in the arguments are unchanged after the command is processed.

<ThicknessValue>

Type: <real> or <string>, either “Infinite” or “Effective”.

Array of DC thickness values (including units) corresponding to each object name. You can also specify an infinite thickness or have it calculated automatically.

<string>

Type: <string>

"EnableAuto" is equivalent to checking the checkbox in the second tab of the DC thickness dialog. "DisableAuto" is equivalent to clearing that checkbox. Omitting the string means leaves the checkbox unchanged.

Example:

```
Set oModule = oDesign.GetModule("BoundarySetup")
   oModule.AssignDCThickness Array("Box2"), Array("1mm")
```

The following example includes the use of “Automatic” and “Infinite” settings.

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("hfss_uhf_probe")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AssignDCThickness Array("Arm_1", "Arm_2"), _
Array("" & Chr(34) & "<Effective>" & Chr(34) & "",  _
  "" & Chr(34) & "<Infinite>" & Chr(34) & "")
```

ConstructVariationString

Use: Lists and orders the variables and values associated with a design variation.

Command: None

Syntax: `ConstructVariationString(<ArrayOfVariableNames>, <ArrayOfVariableValuesIncludingUnits>)`

Return Value: Returns variation string with the variables ordered to correspond to the order of variables in design variations. The values for the variables are inserted into the variation string.

Parameters: `<ArrayOfVariableNames>`

Type: string

<ArrayOfVariableValuesIncludingUnits>

Type: string

Example:

```
varstring = oDesign.ConstructVariation-
String(Array("x_size", "y_size"), Array("2mm", "1mm"))
```

DeleteFieldVariation

Use: Delete Field variations, field and mesh, or just field.

Command: [Solver]>Results>Clean Up Solutions

Syntax: `DeleteFieldVariation [All | Array(<parameters>)], boolean, boolean, [boolean]`

Return Value: None

Parameters:

`All`

Deletes all Field and mesh variations

`Array(<parameters>)`

Deletes specified variation

boolean

Example:

```
' ----------------------------------------------
' Script Recorded by ANSYS Electronics Desktop Version 2015.0.0
' 9:42:08  Oct 22, 2014
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("OptimTee")
Set oDesign = oProject.SetActiveDesign("TeeModel")
oDesign.DeleteFieldVariation "All", true, false
```

Example:

```
' ----------------------------------------------
' Script Recorded by ANSYS Electronics Desktop Version 2015.0.0
' 9:51:41  Oct 22, 2014
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("OptimTee")
Set oDesign = oProject.SetActiveDesign("TeeModel")
oDesign.DeleteFieldVariation Array("offset=" & Chr(39) &
"0.0947046688081817in" & Chr(39) & ""),  _
  true, false
```

DeleteFullVariation

Use: Use to selectively make deletions or delete all solution data.

Command: **HFSS>Results>Clean Up Solutions...**

Syntax: `DeleteFullVariation Array(<parameters>), boolean`

Parameters: `All | <DataSpecifierArray>`

If, All, all data of existing variations is deleted.

`Array(<DesignVariationKey>, )`

`<DesignVariationKey>`

Type: <string>

Design variation string.

`<Boolean>`

Type: boolean

Whether to also delete linked data.

Example:

```
Set oDesign = oProject.SetActiveDesign("HFSSModel1")
oDesign.DeleteFullVariation Array(""), false
```

Example:

```
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("ogive")
Set oDesign = oProject.SetActiveDesign("IEDesign1")
oDesign.DeleteFullVariation Array(""), true
```

Example:

```
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Tee")
Set oDesign = oProject.SetActiveDesign("TeeModel")
oDesign.DeleteFullVariation Array("offset=" &Chr(39) & "0.2in" _
& Chr(39) & "", "offset=" & Chr(39) & "0in" & Chr(39) & ""), false
```

DeleteLinkedDataVariation

Use: Deletes cached linked data, either all, or specified variations.

Command: **HFSS>Results>Clean Up Solutions**

Syntax: `DeleteLinkedDataVariation [All | <DesignVariationKey>, <DesignVariationKey>, …]`

Return Value: None

Parameters:

`All`

Deletes All Linked data.

<DesignVariationKey>

Type: <string>

Design variation string.

Example:

```
' ----------------------------------------------
' Script Recorded by ANSYS Electronics Desktop Version 2015.0.0
' 13:37:16  Oct 22, 2014
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("OptimTee")
Set oDesign = oProject.SetActiveDesign("TeeModel")
oDesign.DeleteLinkedDataVariation "All"
```

Example:

```
' ----------------------------------------------
' Script Recorded by ANSYS Electronics Desktop Version 2015.0.0
' 13:56:43  Oct 22, 2014
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("OptimTee")
Set oDesign = oProject.SetActiveDesign("TeeModel")
```

```
oDesign.DeleteLinkedDataVariation Array("offset=" & Chr(39) &
"0.0947046688081817in" & Chr(39) & "",  _
  "offset=" & Chr(39) & "0.2in" & Chr(39) & "", "offset=" & Chr(39) &
"0.3in" & Chr(39) & "",  _
  "offset=" & Chr(39) & "0.4in" & Chr(39) & "", "offset=" & Chr(39) &
"0.5in" & Chr(39) & "",  _
  "offset=" & Chr(39) & "0.6in" & Chr(39) & "", "offset=" & Chr(39) &
"0.7in" & Chr(39) & "",  _
  "offset=" & Chr(39) & "0.8in" & Chr(39) & "", "offset=" & Chr(39) &
"0.9in" & Chr(39) & "",  _
  "offset=" & Chr(39) & "0in" & Chr(39) & "", "offset=" & Chr(39) &
"1in" & Chr(39) & "")
```

DeleteVariation

Use: Obsolete. Use DeleteFullVariation, DeleteFieldVariation, DeleteLinkedDataVariation.

ExportConvergence

Use: Exports convergence data (max mag delta S, E, freq) to file for the given variation.

Command: None

Syntax: `ExportConvergence <SetupName>, <VariationString>, <FilePath> <overwriteIfExists>`

Return Value: None

Parameters:

`<SetupName>`

Type: <string>

`Example: "Setup1"`

`<VariationString>`

Type: <string>

`Example: "radius = 3mm"`

The empty variation string ("") is interpreted to mean the current nominal variation.

`<FilePath>`

Type: <string>

`Example: "c:\convergence.conv"`

`overwriteIfExists <Boolean>`

If "overwriteIfExists" is TRUE, then the playback of the script overwrites an existing file. If FALSE, it does not. The default is "TRUE".

Type: <string>

Example: overwriteIfExists=TRUE

Example:

```
oDesign.ExportConvergence "Setup1", "x_size = 2mm",
"c:\convergence.conv"
```

For Q3D Extractor the ExportConvergence command details are as follows:

Use: Exports convergence data to file for the given variation.

Command: None

Syntax: `ExportConvergence <SetupName>, <VariationString>, <SolnType>, <FilePath>`

Return Value: None

Parameters:

`<SetupName>`

Type: <string>

Example: "Setup1"

`<VariationString>`

Type: <string>

Example: "radius = 3mm"

The empty variation string ("") is interpreted to mean the current nominal variation.

`<SolnType>`

Type: "CG", "DC RL" and "AC RL".

`<FilePath>`

Type: <string>

Example: "c:\convergence.conv"

`overwriteIfExists <Boolean>`

Type: <string>

Example: overwriteIfExists=TRUE

If "overwriteIfExists" is TRUE, then the playback of the script overwrites an existing file. If FALSE, it does not. The default is "TRUE".

Example: `oDesign.ExportConvergence "Setup1", "x_size = 2mm", "AC RL", "c:\convergence.conv"`

ExportMatrixData

Use: Exports matrix in Matlab or spreadsheet format.

Command: In the **Matrix** tab of the **Solution** dialog box, click **Export>RLGC**.

Syntax:
```
ExportMatrixData <FileName>, <SolnType>,
<DesignVariationKey>, <Solution>, <Matrix>, <ResUnit>,
<IndUnit>, <CapUnit>, <CondUnit>, <Frequency>,
<MatrixType>, <PassNumber>, <ACPlusDCResistance>
```

Return Value: none

Parameters:

```
<FileName>
  Type: <string>
  The path and name of the file where the data will be
  exported. The file extension determines the file for-
  mat.
<SolnType>
  Type: <string>
  One of the following "C", "AC RL" and "DC RL"
<DesignVariationKey>
  Type: <string>
  Design variation string
  Example: "radius = 3mm"
  The empty variation string ("") is interpreted to mean
  the current nominal variation.
<Solution>
  <SolveSetup>:<Soln>
  Parameters:
  <SolveSetup>
    Type: <string>
    Name of the solve setup
  <Soln>
    Type: <string>
    Name of the solution at a certain adaptive pass.
<Matrix>
  Type: <string>
  Either "Original" or one of the reduce matrix setup
  names
<ResUnit>
  Type: <string>
```

```
    Unit used for the resistance value
<IndUnit>
    Type: <string>
    Unit used for the inductance value
<CapUnit>
    Type: <string>
    Unit used for the capacitance value
<CondUnit>
    Type: <string>
    Unit used for the conductance value
<Frequency>
    Type: <double>
    Frequency in hertz.
<MatrixType>
    Type: <String>
    Value: "Maxwell", "Spice", "Couple" , (one or all of
    these).
    Matrix type to export.
<PassNumber>
    Type: <integer>
    Pass number.
<ACPlusDCResistance>
    Type: <bool>
    Default Value: false
    Add DC and AC resistance and export the matrix.
```

Example:

```
oDesign.ExportMatrixData "C:/temp/3DScripts/
rlgc_lvl.lvl", "C, DC RL, AC RL", "",  _
"Setup1:LastAdaptive", "Original", "ohm", "nH", "pF",
"mSie", 100000000,  _ "Maxwell,Spice,Couple", 0
```

ExportMatrixData (2D Extractor)

Use: Export matrix in Matlab or spreadsheet format.

Command: In the **Matrix** tab of the **Solution** dialog box, click **Export>S-Parameter**.

Syntax:

```
ExportMatrixData <FileName>, <SolnType>,
<DesignVariationKey>, <Solution>, <ReduceMatrix>,
```

<ResUnit>, <IndUnit>, <CapUnit>, <CondUnit>, <Frequency>, <LengthSettings>,<LumpedLength>,<MatrixType>,<PassNumber>

Return Value: none

Parameters:

<FileName>

Type: <string>

The name of the file. The file extension will determine the file format.

<SolnType>

Type: <string>

Values: "CG", "RL", "CG,RL"

<DesignVariationKey>

Type: <string>

Design variation string

Example: "radius = 3mm"

The empty variation string ("") is interpreted to mean the current nominal variation.

<Solution>

<SolveSetup>:<Soln>

Parameters:

<SolveSetup>

Type: <string>

Name of the solve setup

<Soln>

Type: <string>

Name of the solution at a certain adaptive pass or sweep.

<ReduceMatrix>

Type: <string>

Either "Original" or one of the reduce matrix setup name

<ResUnit>

Type: <string>

Unit used for resistance value

<IndUnit>

Type: <string>

Unit used for inductance value

```
<CapUnit>
   Type: <string>
   Unit used for capacitance value
<CondUnit>
   Type: <String>
   Unit used for conductance value.
<Frequency>
   Type: <double> (always in Hz)
   Frequency used for exporting matrix.
<LengthSetting>
   Type:<String>
   Values: "Distributed", "Lumped"
<LumpedLength>
   Type:<String>
   Length with units, length of the design at which it is
   exported.
<MatrixType>
   Type: <String>
   Values: "Maxwell", "Spice", "Couple" , or any combina-
   tion of these like "Maxwell, Spice, Couple".
<PassNumber>
   Type: <Integer>
   Pass number.
```

Example:

```
oDesign.ExportMatrixData "C:/temp/export1/export1.lvl",
"CG, RL", "",  _ "Setup6:Sweep2", "Original", "ohm",
"nH", "pF", "mSie", 1000000000, "Lumped",  _ "7meter",
"Maxwell,Spice,Couple", 0
```

ExportNetworkData

Use: Exports sparameters.

Command: In the **matrix** tab of the **Solution** dialog box, click **Export>S-Parameter**.

Syntax:

```
ExportNetworkData <DesignVariation> <SolutionName>
<FileName> <ReduceMatrix> <Reference Impedance>
<FrequencyArray> <Format> <Length> <PassNumber>
```

Return Value: None

Parameters:

```
<DesignVariation>
```

```
    Type: <String>
    Design variation at which the solution is exported.
<SolutionName>
    Type: <String>
    Format: <SetupName>:<SolutionName>
    Solution that is exported.
<FileName>
    Type: <String>
    The name of the file. The file extension will deter-
    mine the file format.
<ReduceMatrix>
    Type: <String>
    Either "Original" or one of the reduce matrix setup
    name
<Reference Impedance>
    Type: <Double>
    Reference impedance
<FrequencyArray>
    Type: <Array of doubles>
    Value: Array(<double>,<double>….)
    Frequency points in the sweep that is exported.
<Format>
    Type: <String>
    Values: "MagPhase", "RealImag", "DbPhase".
    The format in which the sparameters will be exported.
<Length>
    Type:<String>
    Length for exporting s-parameters.
<PassNumber>
    Type:<integer>
    Pass number.
```

Example:

```
oDesign.ExportNetworkData "", "Setup7 : LastAdaptive",
"C:/temp/tc.s2p",  _ "Original", 50, Array(10000000,
20000000, 30000000, 40000000, 50000000,  _ 60000000,
70000000, 80000000, 90000000, 100000000),
"MagPhase",0,"7meter"
```

Example:

```
oDesign.ExportNetworkData "", "Setup7 : LastAdaptive",
"C:/temp/leg.szg",  _ "Original", 50, Array(10000000,
20000000, 30000000, 40000000, 50000000,  _ 60000000,
70000000, 80000000, 90000000, 100000000),
MagPhase",0,"7meter"
```

Example:

```
oDesign.ExportNetworkData "", "Setup7 : LastAdaptive",
"C:/temp/citi.cit",  _ "Original", 50, Array(10000000,
20000000, 30000000, 40000000, 50000000,  _ 60000000,
70000000, 80000000, 90000000, 100000000),
"MagPhase",0,"7meter"
```

Example:

```
oDesign.ExportNetworkData "", "Setup7 : LastAdaptive",
"C:/temp/txt.txt",  _ "Original", 50, Array(10000000,
20000000, 30000000, 40000000, 50000000,  _ 60000000,
70000000, 80000000, 90000000, 100000000),
"MagPhase",0,"7meter"
```

Example:

```
oDesign.ExportNetworkData "", "Setup7 : LastAdaptive",
"C:/temp/mat.m",  _ "Original", 50, Array(10000000,
20000000, 30000000, 40000000, 50000000,  _ 60000000,
70000000, 80000000, 90000000, 100000000),
"MagPhase",0,"7meter"
```

ExportNMFData

Use: Exports s-parameters in neutral file format.

Command: In the **matrix** tab of the **Solution** dialog box, click **Export->S-Parameter**. Then select the **Neutral Model** file format.

Syntax:

```
ExportNetworkData <SolutionName> <FileName>
<ReduceMatrix><Reference Impedance> <FrequencyArray>
<DesignVariation> <Format> <Length> <PassNumber>
```

Return Value: None

Parameters:

```
SolutionName>
     Type: <String>
     Format: <SetupName>:<SolutionName>
     Solution that is exported.
<FileName>
     Type:<String>
     The name of the file. The file extension will deter-
     mine the file format.
<ReduceMatrix>
     Type: <string>
     Either "Original" or one of the reduce matrix setup
```

```
        name
    <Reference Impedance>
        Type: <Double>
        Reference impedance
    <FrequencyArray>
        Type: <Array of doubles>
        Value: Array(<double>,<double>….)
        Frequency points in the sweep that is exported.
    <DesignVariation>
      Type: <String>
      Design variation at which the solution is exported.
    <Format>
        Type: <String>
        Values: "MagPhase", "RealImag", "DbPhase".
        The format in which the sparameters will be exported.
    <Length>
        Type:<String>
        Length for exporting sparameters.
    <PassNumber>
        Type:<integer>
        Pass number.
```

Example:

```
oDesign.ExportNMFData "Setup7 : LastAdaptive", "C:/temp/neu.nmf",
"Original",  _ 50, Array(10000000, 20000000, 30000000, 40000000,
50000000, 60000000, 70000000,  _ 80000000, 90000000, 100000000), "",
"MagPhase","7meter"
```

ExportMeshStats

Use: Exports the mesh statistics to a file.

Command: None.

Parameters:

`<SetupName>`

Type: <string>

`Example: "Setup1"`

`<VariationString>`

Type: <string>

`Example: "radius = 3mm"`

The empty variation string ("") is interpreted to mean the current nominal variation.

<FilePath>

Type: <string>

Example: "c:\convergence.conv"

overwriteIfExists <Boolean>

If "overwriteIfExists" is TRUE, then the playback of the script overwrites an existing file. If FALSE, it does not. The default is "TRUE".

Type: <string>

Example: overwriteIfExists=TRUE

Example:

```
oDesign.ExportMeshStats "Setup1", "offset=" & Chr(39) &
"0.09in" & Chr(39) & "","C:\mydir\meshstats.ms" "tat"
```

ExportProfile

Use: Exports a solution profile to file.

Syntax: ExportProfile <SetupName>, <VariationString>, <FilePath>, <overwriteIfExists>

Return Value: None

Parameters:

<SetupName>

Type: <string>

Example: "Setup1"

<VariationString>

Type: <string>

Example: "radius = 3mm"

The empty variation string ("") is interpreted to mean the current nominal variation.

<FilePath>

Type: <string>

Example: "c:\profile.prof"

overwriteIfExists <Boolean>

If "overwriteIfExists" is TRUE, then the playback of the script overwrites an existing file. If FALSE, it does not. The default is "TRUE".

Type: <string>

Example: overwriteIfExists=TRUE

Example:

```
oDesign.ExportProfile "Setup1", "", "c:\profile.prof"
```

GetEdit SourcesCount

Use: Returns the number of sources that are listed in the Edit Sources panel.

Command: None

Syntax: `GetEditSources`

Return Value: <int>

Parameters: `None`

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 15.0.0
' 7:37:36 AM  Jul 27, 2012
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.GetActiveProject
Set oDesign = oProject.GetActiveDesign
Set oModule = oDesign.GetModule("Solutions")
Dim count
count = oModule.GetEditSourcesCount
MsgBox(count)
```

GetExcitationsModule

Use: Query source scalings (mag, phase etc.) from the Edit Source panel.

Command: None

Syntax: `GetExcitationScaling( <port name>, <port index>)`

Return Value: Scaling for excitation.

Parameters: `<port name>`

Port ID

`<port index>`

Port Index if present.

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 2015.0.0
' 10:26:55  Jan 07, 2014
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project15")
Set oDesign = oProject.SetActiveDesign("HfssDesign1")
Set oModule = oDesign.GetModule("Solutions")

' --------- first terminal off of port "1" -------
' data = oModule.GetExcitationScaling( "1" )

' --------- excitations for modal "port1:2" -------
data = oModule.GetExcitationScaling( "port1", 2 )

' -- Second eigenmode (excitation name is irrelevant --
' data = oModule.GetExcitationScaling( "unused", 2 )

' ----------------------------------------------
' sample code to display results
' ----------------------------------------------

buffer = ""
for i = 0 to uBound(data)
   buffer = buffer & data(i)
```

```
    if i < uBound(data) then buffer = buffer + ", "
next
MsgBox( buffer )
```

GetModule

Use: Returns the IDispatch for the specified module.

Command: none

Syntax: `GetModule <ModuleName>`

Return Value: Module object.

Parameters:

```
Type: <string>
Name of the module. One of the following:
   -Boundary module:  "BoundarySetup"
   -Mesh Operations module: "MeshSetup"
   -Analysis module:  "AnalysisSetup"
   -Optimetrics module:  "Optimetrics"
   -Solutions module:  "Solutions"
   -Field Overlays module: "FieldsReporter"
   -Radiation module:  "RadField"
```

Example:

```
Set oModule = oDesign.GetModule "BoundarySetup"
```

For Q3D Extractor the GetModule command details are as follows.

Use: Returns the IDispatch for the specified module.

Command: none

Syntax: `GetModule <ModuleName>`

Return Value: Module object.

Parameters:

```
Type: <string>
Name of the module. One of the following:
   -Boundary module:  "BoundarySetup"
   -Mesh Operations module: "MeshSetup"
   -Reduce Matrix module: "ReduceMatrix"
   -Analysis module:  "AnalysisSetup"
   -Optimetrics module:  "Optimetrics"
   -Solutions module:  "Solutions"
   -Field Overlays module: "FieldsReporter"
```

Example:

```
Set oModule = oDesign.GetModule "MeshSetup"
```

GetName

Use: Returns the name of the Design.

Command: none

Syntax: `GetName`

Return Value: The name of the Design.

Type: <string>

Example:

```
name_string = oDesign.GetName
```

GetNominalVariation

Use: Gets the nominal variation string

Command: None

Syntax: `GetNominalVariation()`

Return Value: Returns a string representing the nominal variation

Parameters: `None`

Example:

```
var = oDesign.GetNominalVariation()
```

GetSelections

Use: Informational.

Command: None

Syntax: `GetSelections`

Return Value: array of IDs

Parameters: `None`

Example:

```
Set oProject = oDesktop.SetActiveProject("Project6")
Set oDesign = oProject.SetActiveDesign("Q3DDesign1")
Set oEditor = oDesign.SetActiveEditor("Modeler")
Dim A
A = Array()
A = oEditor.GetSelections
Dim B
B = Join(A,",")
```

```
'Debug.Write "The Selections are " &B
MsgBox(B)
Dim C
C = Array("NAME:Selections", "Selections:=", B)
oEditor.Delete C
```

GetSolutionType

Use: Returns the solution type for the design. This command command does not apply to HFSS-IE.

Command: none

Syntax: `GetSolutionType`

Return Value: `<SolutionType>`

Type: <string>

Possible values are: `"DrivenModal"`, `"DrivenTerminal"`, `"Eigenmode"`, `"Transient"` or `"Transient Network"`.

Example:

```
oDesign.GetSolutionType
```

GetSolveInsideThreshold

Use: Returns the solve inside threshold. This command command does not apply to HFSS-IE.

Command: none

Syntax: `GetSolveInsideThreshold`

Return Value: Double representing the solve inside threshold.

Example:

```
oDesign.GetSolveInsideThreshold
```

GetSourceContexts

Use: Obtain sources currently enabled as context in the Edit Sources dialog Source Context tab.

Command: None

Syntax: `GetSource Contexts`

Return Value: Array of enabled source names

Parameters: `None`

Example:

```
SetoModule = oDesign.GetModule("Solutions")
oModule.GetSourceContexts
```

GetVariationVariableValue

Use: Finds the value of a variable for a specific variation string.

Command: None

Syntax: `GetVariationVariableValue(<VariationString>, <VariableName>)`

Return Value: Returns a double precision value in SI units, interpreted to mean the value of the variable contained in the variation string.

Parameters:

`<VariationString>`

Type: string

`<VariableName>`

Type: string

Example:

```
Example: varval = _
oDesign.GetVariationVariableValue("x_size = 2mm y_size = 1mm",_ "y_size")
```

Redo [Design]

Use: Reapplies the last design-level command.

Command: **Edit>Redo**

Syntax: `Redo`

Return Value: None

Example:

```
oDesign.Redo
```

RenameDesignInstance

Use: Renames a design instance.

Command: Right click a design instance in the project tree, and then click **Rename** on the shortcut menu.

Syntax: `RenameDesignInstance <OldName>, <NewName>`

Return Value: None

Parameters:

`<OldName>`

Type: <string>

```
    Name of the design instance being renamed.
<NewName>
    Type: <string>
    New name of the design instance.
```

Example:

```
oDesign.RenameDesignInstance "HFSSDesign1", "HFSSDesign2"
```

ResetToTimeZero

Use: To reset a simulation to time zero.

Command: CleanStop

Syntax: `ResetToTimeZero( "<name of setup>" )`

Return Value: None

Parameters: `<Name of Setup>`

Example:

```
# ----------------------------------------------
# Script Recorded by ANSYS Electronics Desktop Version 2015.0.0
# 16:11:48  Nov 14, 2014
# ----------------------------------------------
import ScriptEnv
ScriptEnv.Initialize("Ansoft.ElectronicsDesktop")
oDesktop.RestoreWindow()
oProject = oDesktop.SetActiveProject("TermPorts")
oDesign = oProject.SetActiveDesign("TermPorts2Active")
oDesign.AnalyzeAll()
oModule = oDesign.GetModule("AnalysisSetup")
oModule.ResetToTimeZero("Setup1")
```

SARSetup

Use: Sets up for the specific absorption rate (SAR) computation.This command command does not apply to HFSS-IE.

Command: **HFSS>Fields>SAR Setting**

Syntax: `SARSetup <TissueMass>, <MaterialDensity>, <Tissue object List ID>, <voxel size>, <Average SAR method>`

Return Value: None

Parameters: `<TissueMass>`

Type: <double>

Double between 1 and 10 in grams.

`<MaterialDensity>`

Type: <double>

Positive double in gram/cm^3.

`<Tissue Object listID>`

Type: <integer>

The ID of an object list which will be treated as tissues for SAR calculation.

`<voxel size>`

Type: <double>

The size of a voxel in millimeters.

`<Average SAR method>`

Type: <integer>

0: IEEE std 1528.

1: Gridless, i.e. classical Ansoft method.

Example:

```
oDesign.SARSetup 1, 1, 678, 1, 0
```

SetActiveEditor

Use: Sets the active editor.

Command: None

Syntax: `SetActiveEditor(<EditorName>)`

Return Value: Editor object

Parameters:

```
<EditorName>
   Type: <string>
   The only choice is "3D Modeler"
```

Example:

```
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
```

SetBackgroundMaterial

Use: Sets the background material of the design.

Command: Right click on the design in the project tree and choose "**Set Background Material**".

Syntax: `SetBackgroundMaterial <MatName>`

Return Value: None

Parameters:

```
<MatName>
   Type: <string>
   The name of the background material
```

Example: `oDesign.SetBackgroundMaterial "vacuum"`

Note For a 2D project, you can run this command only if the following conditions are met:

- there is no surface ground in the design, and
- problem type is "open".

SetDesignSettings

Use: To set the design settings for materials override.

Command: **HFSS>Design Settings**

Syntax: `SetDesignSettings <MaterialsOverrideArray>`

Return Value: None

Parameters:

```
<MaterialsOverrideArray>
   Array("NAME:Design Settings Data",
   "Allow Material Override:=", <Boolean>,
   "Calculate Lossy Dielectrics:=", <Boolean>)
```

Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project53")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
```

```
oDesign.SetDesignSettings Array("NAME:Design Settings Data",
"Allow Material Override:=", true,
"Calculate Lossy Dielectrics:=", true)
```

For Q3D Extractor the details for SetDesignSettings are as follows:

Use: To set the design settings for materials override.

Command: Click **Q3D Extractor>Design Settings** to open the **Design Settings** dialog box, and then choose **Enable Material Override**.

Syntax: `SetDesignSettings <MaterialsOverrideArray>`

Return Value: None

Parameters:

```
<MaterialsOverrideArray>
Array("NAME:Design Settings Data",
"Allow Material Override:=", <Boolean>,
"Calculate Lossy Dielectrics:=", <Boolean>)
```

Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftQ3D.Q3DScriptInter-
face")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project53")
Set oDesign = oProject.SetActiveDesign("Q3DExtractorDe-
sign1")
oDesign.SetDesignSettings Array("NAME:Design Settings
Data",
"Allow Material Override:=", true,
"Calculate Lossy Dielectrics:=", true)
```

SetLengthSettings

Use: Sets the distributed and lumped length of the design.

Syntax: `SetLengthSettings`

Return Value: None

Parameters:

```
<DistributedUnits>
   Type:<String>
   Value: Length units
   Length Units used for post processing in the design.
<LumpedLength>
   Type:<String>
   Value: Length (double) with units.
   Length of design used in post processing.
<RiseTime>
   Type: <String>
   Value: Time with units.
   Rise time used in post processing steps.
```

```
oDesign.SetLengthSettings "mm", "7meter",  "1s"
```

SetSolutionType

Use: Sets the solution type for the design. This command command does not apply to HFSS-IE.

Command: **HFSS>Solution Type**

Syntax: `SetSolutionType <SolutionType>`

Return Value: None

Parameters:

`<SolutionType>`

Type: <string>

Possible values are: `"DrivenModal"`, `"DrivenTerminal"`, “Transient”, “Transient Network”, or `"Eigenmode"`

Example:

```
oDesign.SetSolutionType "DrivenTerminal"
```

For Q3D Extractor the command details are as follows:

Use: Sets the solution type for the design.

Command: **2D Extractor>Solution Type**

Syntax: `SetSolutionType <SolutionType>`

Return Value: None

Parameters:

`<SolutionType>`

Type: <string>

Possible values are: `"Open"`, or `"Closed"`

Example:

```
oDesign.SetSolutionType "Open"
```

SetSolveInsideThreshold

Use: Set the solve inside threshold to the supplied double. This command command does not apply to HFSS-IE.

Command: None

Syntax: `SetSolveInsideThreshold(<threshold>)`

Return Value: None

Parameters: `<threshold>`

Type: <double>

Siemens/m

Example:

```
oDesign.SetSolveInsideThreshold(100000)
```

SetSourceContexts

Use: For Near or Far Field projects for Driven Modal or Driven Terminal Network Analysis Solutions, specify the port name and all modes/terminals of that port to be enabled as Source Context.

Command: **Fields>Edit Sources**

Syntax: `SetSourceContexts Array("<sourceID>",...)`

Return Value: None.

Parameters: `<sourceID>`

Type: <string>

Example:

```
SetoModule = oDesign.GetModule("Solutions")
oModule.SetSourceContexts Array("Box1_T1", "Box1_T2",
"Box1_T3", "Current1", "IncPWave1")
```

Solve

Use: Performs a blocking simulation. The next script command will not be executed until the simulation is complete.

Command: **HFSS>Analyze**

Syntax: `Solve <SetupNameArray>`

Return Value: Type: <int>

-1: simulation error

0: normal completion

Parameters: `<SetupNameArray>: Array(<SetupName>, <SetupName>, ...)`

`<SetupName>`

Type: <string>

Name of the solution setup to solve.

Example:

```
return_status = oDesign.Solve Array("Setup1", "Setup2")
```

For Q3D Extractor Solve command details are as follows:

Use: Performs a blocking simulation. The next script command will not be executed until the simulation is complete.

Command: **Q3D Extractor** or **2D Extractor>Analyze**. Or right-click the **Analysis** option in the project tree and choose "**Analyze**".

Syntax: `Solve <SetupNameArray>`

Return Value: Type: <int>

```
-1: command execution error
 1: simulation error
 0: normal completion
```

Parameters: `<SetupNameArray>: Array(<SetupName>, <SetupName>, ...)`

`<SetupName>`

```
Type: <string>
Name of the solution setup to solve.
```

Example:

```
return_status = oDesign.Solve Array("Setup1", "Setup2")
```

RunToolkit

Use: Run a Python toolkit script, applying it to the Active Project. The script itself may have prerequisites, such as sweeps defined, or specific model characteristics, such as port definitions.

Command: HFSS>Toolkit><IronPythonScript>

Syntax: `RunToolkit "<LibName>", "<IronPythonScriptName>", Array()`

Return Value: User Defined Solutions and user Defined Outputs.

Parameters: `<LibName>`

Type: <string>

Name of the library in which the script is located.

`<IronPythonScriptName>`

Type: <string>

Name of the IronPython script.

Array(<array>

Type: <Array>

Additional parameters or files required by the script.

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 16.0.0
' 10:12:18 AM  May 14, 2013
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("test_HAC")
Set oDesign = oProject.SetActiveDesign("RCS1")
oDesign.RunToolkit "SysLib", "HearingAidCompliance", Array()
Set oModule = oDesign.GetModule("AnalysisSetup")
oModule.InsertFrequencySweep "Setup1", Array("NAME:Sweep", "IsEn-
abled:=", true, "SetupType:=",  _
  "LinearStep", "StartValue:=", "1GHz", "StopValue:=", "10GHz",
"StepSize:=",  _
  "0.1GHz", "Type:=", "Interpolating", "SaveFields:=", false, "Sav-
eRadFields:=",  _
  false, "InterpTolerance:=", 0.5, "InterpMaxSolns:=", 250, "Interp-
MinSolns:=",  _
  0, "InterpMinSubranges:=", 1, "ExtrapToDC:=", false, "Inter-
pUseS:=", true, "InterpUsePortImped:=",  _
  false, "InterpUsePropConst:=", true, "UseDerivativeConvergence:=",
false, "InterpDerivTolerance:=",  _
  0.2, "UseFullBasis:=", true, "EnforcePassivity:=", true, "Passivi-
tyErrorTolerance:=",  _
  0.0001)
```

For Q3D Extractor the RunToolkit command details are as follows:

Use: Run a Python toolkit script, applying it to the Active Project.

Command: **Q3D Extractor** or **2D Extractor>Toolkit**

Syntax:
```
RunToolkit "<LibName>", "<IronPythonScriptName>", Array()
```

Return Value: User Defined Solutions and user Defined Outputs.

Parameters:
```
<LibName>
   Type: <string>
      Name of the library in which the script is located.
<IronPythonScriptName>
      Type: <string>
      Name of the IronPython script.
Array(<array>)
      Type: <Array>
      Additional parameters or files required by the-
      script.
```

Full scripting for cable modeling is not supported and no arguments are allowed in the script. The **RunToolkit** command will not automatically create a cable bundle, but will simply open the **Cable Modeling** dialog box. You will have to manually input your parameters.

Currently, the script to launch the cable modeling dialog boxes is:

```
oDesign.RunToolkit "SysLib", "CableModeling/AutomotiveCa-
bleBundle", Array()
oDesign.RunToolkit "SysLib", "CableModeling/Oil-GasCable-
Bundle", Array()
```

Example:
```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Q3DExtractor.ScriptInter-
face")

Set oDesktop = oAnsoftApp.GetAppDesktop()
```

```
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project1")
oProject.InsertDesign "Q3D Extractor", "Q3DDesign1", "",
""
oProject.SaveAs "E:\project_directories\q3d\v13\user_sto-
ries_R15\US53001_cable_modeling\auto_script.q3dx", true
Set oDesign = oProject.SetActiveDesign("Q3DDesign1")

oDesign.RunToolkit "SysLib", "CableModeling/AutomotiveCa-
bleBundle", Array()
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AutoIdentifyNets

Set oModule = oDesign.GetModule("AnalysisSetup")
oModule.InsertSetup "Matrix", Array("NAME:Setup1", "Adap-
tiveFreq:=", "1GHz", "SaveFields:=",  _
  true, "Enabled:=", true, Array("NAME:Cap", "MaxPass:=",
10, "MinPass:=", 1, "MinConvPass:=",  _
  1, "PerError:=", 1, "PerRefine:=", 30, "AutoIncreaseSo-
lutionOrder:=", true, "SolutionOrder:=",  _
  "Normal"), Array("NAME:DC", "Residual:=", 1E-005,
"SolveResOnly:=", false, Array("NAME:Cond", "Max-
Pass:=",  _
  10, "MinPass:=", 1, "MinConvPass:=", 1, "PerError:=",
1, "PerRefine:=", 30), Array("NAME:Mult", "MaxPass:=",  _
  1, "MinPass:=", 1, "MinConvPass:=", 1, "PerError:=", 1,
"PerRefine:=", 30)), Array("NAME:AC", "MaxPass:=",  _
  10, "MinPass:=", 1, "MinConvPass:=", 1, "PerError:=",
1, "PerRefine:=", 30))
oProject.Save
```

Undo [Design]

Use: Cancels the last design-level command.

Command: **Edit>Undo**

Syntax: `Undo`

Return Value: None

Example:

```
oDesign.Undo
```

10 Model Setup Script Commands

Model Setup Script Commands should be executed by:

```
Set oModule = oDesign.GetModule("ModelSetup")
```

These commands are as follows:

AssignArray

DeleteArray

EditArray

Assign Array

Use: Create an array based on a unit cell model.

Command: **HFSS>Model>Create Array**

Syntax: `AssignArray <arrayParameters>`

Return Value: None

Parameters:

```
Aray(NAME:<string>",
"Type:=", "Regular",
"Name:=", "<string>",
"UseAirObjects:=", <boolean>,
"RowMasterBnd:=", "Master<n>",
"ColumnMasterBnd:=", "Master<n>",
"RowDimension:=", <value>,
"ColumnDimension:=", <value>,
"PostProcessRow:=", <value>,
"PostProcessCol:=", <value>,
"Active:=", "All" | [<arraycoords>] |"None" )
```

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("terminalArray")
Set oDesign = oProject.SetActiveDesign("HFSSDesign")
Set oModule = oDesign.GetModule("ModelSetup")
oModule.AssignArray Array("NAME:A",
"Type:=", "Regular",
```

```
    "Name:=", "A",
    "UseAirObjects:=", true,
    "RowMasterBnd:=", "Master1",
    "ColumnMasterBnd:=", "Master2",
    "RowDimension:=", 2,
    "ColumnDimension:=", 2,
    "PostProcessRow:=", 1,
    "PostProcessCol:=", 1,
    "Active:=", "All")
```

DeleteArray

Use: Delete an existing array based on a unit cell model.

Command: **Delete**

Syntax: `DeleteArray`

Return Value: None

Parameters: `None`

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("forcreatearraycube")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("ModelSetup")
oModule.DeleteArray
```

EditArray

Use: To edit properties of an existing array.

Command: **Properties**

Syntax: `EditArray <arrayParameters>`

Return Value: None

Parameters:

```
Aray(NAME:<string>",
"Type:=", "Regular",
"Name:=", "<string>",
"UseAirObjects:=", <boolean>,
"RowMasterBnd:=", "Master<n>",
"ColumnMasterBnd:=", "Master<n>",
"RowDimension:=", <value>,
"ColumnDimension:=", <value>,
"PostProcessRow:=", <value>,
"PostProcessCol:=", <value>,
"Active:=", "All" | [<arraycoords>] |"None" )
```

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("terminalArray")
Set oDesign = oProject.SetActiveDesign("HFSSDesign")
Set oModule = oDesign.GetModule("ModelSetup")
oModule.EditArray Array("NAME:A",
"Type:=", "Regular",
"Name:=", "A",
```

```
"UseAirObjects:=", true,
"RowMasterBnd:=", "Master1",
"ColumnMasterBnd:=", "Master2",
"RowDimension:=", 2,
"ColumnDimension:=", 3,
"PostProcessRow:=", 1,
"PostProcessCol:=", 3,
"Active:=", "[0,0], [0,1], [0,2]")
```

11 3D Modeler Editor Script Commands

3D Modeler commands should be executed by the `"3D Modeler"` editor.

```
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.CommandName <args>
```

Conventions Used in this Chapter

```
<Attributes Array>
   Array("NAME:Attributes",
      "Name:=", <string>,
      "Flags:=", <string>,
      "Color:=", <string>,
      "Transparency:=", <value>,
      "PartCoordinateSystem:=", <string>,
      "MaterialName:=", <string>,
      "Solveinside:=", <bool>)
```

`Flags`

Format is a string containing any of the following flags separated by the # character:

- `NonModel`
- `Wireframe`

Example: `"Flags:=", "NonModel#Wireframe"`

`Color`

Format is a string containing an R,G,B triple formatted as `"(R G B)"`.

Example: `"Color:=", "(255 255 255)"`

`Transparency`

Specify a number between 0 and 1.

`PartCoordinateSystem`

Orientation of the primitive. The name of one of the defined coordinate systems should be specified.

```
<SelectionsArray>
   Array("NAME:Selections",
      "Selections:=", <string>)
```

`Selections`

Comma-separated list of parts on which to perform the operation.

Example: `"Selections:=", "Rect1, Rect2"`

Draw Menu Commands

Create3D Component

Use: Create a 3D component

Command: None

Syntax: `Create3DComponent <Geometry Data>, <Design Data>, <File Name>, <Image File>`

Return Value: None

Parameters: `< Geometry Data>`

Geometry data

`< Design Data>`

Design data

< File Name>

File name of 3D component

< Image File>

File name of 3D component image

Example:

```
oEditor.Create3DComponent Array("NAME:GeometryData",
"ComponentName:=", "Connector",
"Owner:=", "", "Email:=", "", "Company:=", "",
"Version:=", "1.0", "Date:=", "11:41:01 AM  Aug 28, 2014",
"Notes:=", "", "HasLabel:=", false,
"IncludedParts:=", Array( "Box1", "Cylinder1", "Cone1"),
"IncludedCS:=", Array("RelativeCS1"),
"ReferenceCS:=", "Global",
"IncludedParameters:=", Array("htcone", "lr", "htcyl", "zs",
"radcyl", "xs", "$rp", "$con"),
"ParameterDescription:=", Array()),
Array("NAME:DesignData", "Boundaries:=",
Array( "PerfE1", "FiniteCond1"),
"Excitations:=", Array("1"),
"MeshOperations:=", Array()),
"C:/tmp/Connector.a3dcomp",
Array("NAME:ImageFile", "ImageFile:=", "")
```

CreateBondwire

Use: Creates a bondwire primitive.

Command: **Draw>Bondwire**

Syntax: `CreateBondwire <ParametersArray>, <AttributesArray>`

Return Value: The name of the newly created object.

Parameters:

```
<ParametersArray>
   Array("NAME:BondwireParameters",
      "WireType:=", <string>,
      "WireDiameter:=", <value>,
      "NumSides:=", <value>,
      "XPadPos:=", <value>,
      "YPadPos:=", <value>,
      "ZPadPos:=", <value>,
      "XDir:=", <value>,
```

```
"YDir:=", <value>,
"ZDir:=", <value>,
"Distance:=", <value>,
"h1:=", <value>,
"h2:=", <value>,
"alpha:=", <value>,
"beta:=", <value>,
"WhichAxis:=", <string>)
```

`WireType`

Should be one of: `"JEDEC_4Points", "JEDEC_5Points"`

Example: `"WireType:=", "JEDEC_4Points"`

`WhichAxis`

Axis normal to the plane where the wire is drawn. Possible values are: `"X", "Y", "Z"`

Example: `"WhichAxis:=", "Z"` means the bond wire will be drawn on the XY plane.

CreateBox

Use: Creates a box primitive.

Command: **Draw>Box**

Syntax: `CreateBox <BoxParametersArray>, <AttributesArray>`

Return Value: The name of the newly created object.

Parameters:

```
<BoxParametersArray>
   Array("NAME:BoxParameters",
      "XPosition:=", <value>,
      "YPosition:=", <value>,
      "ZPosition:=", <value>,
      "XSize:=",<value>,
      "YSize:=",<value>,
      "ZSize:=",<value>)
```

Example:

```
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.CreateBox Array("NAME:BoxParameters", _
```

```
"CoordinateSystemID:=", -1, "XPosition:=",  _
  "1mm", "YPosition:=", "1mm", "ZPosition:=", "0mm", _
"XSize:=", "1mm", "YSize:=", "1mm", "ZSize:=", "1mm"),_
Array("NAME:Attributes", "Name:=", "Box1", "Flags:=", "", _
"Color:=", "(132 132 193)", "Transparency:=", 0, _
"PartCoordinateSystem:=", "Global", "MaterialName:=",  _
  "vacuum", "SolveInside:=", true)_
oEditor.DuplicateAlongLine Array("NAME:Selections",
"Selections:=", "Box1"), _
Array("NAME:DuplicateToAlongLineParameters",_
"CoordinateSystemID:=", -1, "CreateNewObjects:=", true, _
"XComponent:=", "1mm", "YComponent:=", "1mm", "ZComponent:=",  _
  "0mm", "NumClones:=", "2"), _
Array("NAME:Options", "DuplicateBoundaries:=", true)
```

CreateCircle

Use: Creates a circle primitive.

Command: **Draw>Circle**

Syntax: `CreateCircle <CircleParametersArray>, <AttributesArray>`

Return Value: The name of the newly created object.

Parameters:

```
<CircleParametersArray>
   Array("NAME:CircleParameters",
      "XCenter:=", <value>,
      "YCenter:=", <value>,
      "ZCenter:=", <value>,
      "Radius:=", <value>,
      "WhichAxis:=", <string>
      "NumSegments:=", "<integer>")
```

`WhichAxis`

Axis of normal vector to the circle. Possible values are: `"X"`, `"Y"`, `"Z"`

Example: `"WhichAxis:=", "Z"` means the circle will be drawn in the XY plane.

CreateCone

Use: Creates a cone primitive.

Command: **Draw>Cone**

Syntax:
```
CreateCone <ConeParametersArray>, <AttributesArray>
```

Return Value: The name of the newly created object.

Parameters:
```
<ConeParametersArray>
   Array("NAME:ConeParameters",
      "XCenter:=", <value>,
      "YCenter:=", <value>,
      "ZCenter:=", <value>,
      "WhichAxis:=", <string>,
      "Height:=", <value>,
      "BottomRadius:=", <value>,
      "TopRadius:=", <value>)
```

`WhichAxis`

Axis of the cone. Possible values are: `"X", "Y", "Z"`

Example: `"WhichAxis:=", "Z"`

CreateCutplane

Use: Creates a cutplane. Only the name and color attributes from `<AttributesArray>` are supported.

Command: **Draw>Plane**

Syntax:
```
CreateCutplane <CutplaneParametersArray>,
   <AttributesArray>
```

Return Value: The name of the newly created object.

Parameters:
```
<CutplaneParametersArray>
   Array("NAME:PlaneParameters",
      "PlaneBaseX:=", <value>,
      "PlaneBaseY:=", <value>,
      "PlaneBaseZ:=", <value>,
      "PlaneNormalX:=", <value>,
      "PlaneNormalY:=", <value>),
      "PlaneNormalZ:=", <value>)
```

CreateCylinder

Use: Creates a cylinder primitive.

Command: **Draw>Cylinder**

Syntax:
```
CreateCylinder <CylinderParametersArray>,
   <AttributesArray>
```

Return Value: The name of the newly created object.

Parameters:
```
<CylinderParametersArray>
   Array("NAME:CylinderParameters",
      "XCenter:=", <value>,
      "YCenter:=", <value>,
      "ZCenter:=", <value>,
      "Radius:=", <value>,
      "Height:=", <value>,
      "WhichAxis:=", <string>
      "NumSides:=", "<integer>")
```

`WhichAxis`

Axis of the cylinder. Possible values are: `"X", "Y", "Z"`

Example: `"WhichAxis:=", "Z"`

CreateEllipse

Use: Creates an ellipse primitive.

Command: **Draw>Ellipse**

Syntax:
```
CreateEllipse <EllipseParametersArray>, <AttributesArray>
```

Return Value: The name of the newly created object.

Parameters:
```
<EllipseParametersArray>
   Array("NAME:EllipseParameters",
      "XCenter:=", <value>,
      "YCenter:=", <value>,
      "ZCenter:=", <value>,
      "MajRadius:=", <value>,
      "Ratio:=", <value>,
      "WhichAxis:=", <string>
      "NumSegments:=", "<integer>")
```

`WhichAxis`

Axis of normal vector to the ellipse. Possible values are: `"X", "Y", "Z"`

Example: `"WhichAxis:=", "Z"` means the ellipse will be drawn in the XY plane.

CreateEquationCurve

Use: Create an equation-based curve.

Command: **Draw>Equation Based Curve**

Syntax: `CreateEquationCurve Array <Parameters> Array<Attributes>`

Return Value: None

Parameters:

```
Array("NAME:EquationBasedCurveParameters",
                "XtFunction:=", "<value>",
                "YtFunction:=", "<value>",
                "ZtFunction:=", "<value>",
                "tStart:=", "<value>",
                "tEnd:=", "<value>",
                "NumOfPointsOnCurve:=", "<value>",
                "Version:=", <ID>),
Array("NAME:Attributes",
                "Name:=", "<textn>",
                "Flags:=", "",
                "Color:=", "(<int> <int> <int>)",
                "Transparency:=", <value>,
                "PartCoordinateSystem:=", "<id>",
                "UDMId:=", "",
                "MaterialValue:=", "" & Chr(34)
                & "vacuum"
                & Chr(34) & "",
                "SolveInside:=", <boolean>)
```

Example:

CreateEquationSurface

Use: Create an equation based surface.

Command: **Draw>Create Equation Based Surface**

Syntax: `CreateEquationSurface Array <parameters> Array <attributes>`

Return Value:

Parameters:

```
Array("NAME:EquationBasedSurfaceParameters",
                "XuvFunction:=", "1",
                "YuvFunction:=", "1",
                "ZuvFunction:=", "1",
                "uStart:=", "2",
                "uEnd:=", "2",
                "vStart:=", "2",
                "vEnd:=", "2",
                "Version:=", 1),
Array("NAME:Attributes", "Name:=", "EquationSurface1",
                "Flags:=", "",
                "Color:=", "(132 132 193)",
                "Transparency:=", 0,
                "PartCoordinateSystem:=", "Global",
                "UDMId:=", "",
                "MaterialValue:=", ""
                & Chr(34)
                & "vacuum"
                & Chr(34)
                & "", "SolveInside:=", true)
```

Example:

CreateHelix

Use: Creates a helix by sweeping the specified 2D objects.

Command: **Draw>Helix**

Syntax: `CreateHelix <SelectionsArray>, <HelixParametersArray>`

Return Value: The name of the newly created object.

Parameters:

```
<SelectionsArray>
   Array("NAME:Selections",
      "Selections:=", <string>)
```

```
Selections
```
Comma-separated list of parts to sweep.

Example: `"Selections:=", "Rect1, Rect2"`

```
<HelixParametersArray>
   Array("NAME:HelixParameters",
      "XCenter:=", <value>,
      "YCenter:=", <value>,
      "ZCenter:=", <value>,
      "XStartDir:=", <value>,
      "YStartDir:=", <value>,
      "ZStartDir:=", <value>,
      "Thread:=", <value>,
      "NumThread:=", <value>,
      "RightHand:=", <bool>)
```

CreatePoint

Use: Creates a point. Only the name and color attributes from `<AttributesArray>` are supported.

Command: **Draw>Point**

Syntax: `CreatePoint <PointParametersArray>, <AttributesArray>`

Return Value: The name of the newly created object.

Parameters:

```
<PointParametersArray>
   Array("NAME:PointParameters",
      "PointX:=", <value>,
      "PointY:=", <value>,
      "PointZ:=", <value>)
```

CreateUserDefinedPart

The documented command is applicable to Q3D Extractor.

Use: Creates a user-defined part.

Command: **Draw>User Defined Primitive**

Syntax: `CreateUserDefinedPart <UserDefinedParametersArray>, <AttributesArray>`

Return Value: The name of the newly created object.

Parameters:

```
<UserDefinedParametersArray>
     Array("NAME:UserDefinedPrimitiveParameters",
       "CoordinateSystemID:=", <value>,
       "DllName:=", <string>,
       "Library:=", <string>,
     Array("NAME:ParamVector", Array("NAME:Pair",
     "Name:=",
       <string>, "Value:=", <value>)…)
```

Example:

```
oEditor.CreateUserDefinedPart
Array("NAME:UserDefinedPrimitiveParameters",
"CoordinateSystemID:=",  _ -1, "DllName:=", "Examples/
RectangularSpiral", "NoOfParameters:=", 6, "Library:=",
_ "syslib", Array("NAME:ParamVector", Array("NAME:Pair",
"Name:=", "Xpos", "Value:=", "0mm"), Array("NAME:Pair",
"Name:=",  _ "Ypos", "Value:=", "0mm"),
Array("NAME:Pair", "Name:=", "TurnSep", "Value:=",
"5mm"), Array("NAME:Pair", "Name:=",  _ "Turns",
"Value:=", "2"), Array("NAME:Pair", "Name:=", "Width",
"Value:=", "2mm"), Array("NAME:Pair", "Name:=",  _
"Height", "Value:=", "2mm"))), Array("NAME:Attributes",
"Name:=",  _ "RectangularSpiral1", "Flags:=", "",
"Color:=", "(132 132 193)", "Transparency:=",  _ 0,
"PartCoordinateSystem:=", "Global", "MaterialName:=",
"copper", "SolveInside:=",  _ false)
```

CreatePolyline

Use: Creates a polyline primitive.

Command: **Draw>Polyline**

Syntax:

```
CreatePolyline <PolylineParametersArray>,
   <AttributesArray>
```

Return Value: The name of the newly created object.

Parameters:

```
<PolylineParametersArray>
   Array("NAME:PolylineParameters",
      "IsPolylineCovered:=", <bool>,
      "IsPolylineClosed:=", <bool>,
      <PolylinePointsArray>,
      <PolylineSegmentsArray>)
```

```
<PolylinePointsArray>
   Array("NAME:PolylinePoints", <OnePointArray>,
   <OnePointArray>, ...)

<OnePointArray>
   Array("NAME:PLPoint",
      "X:=", <value>,
      "Y:=", <value>,
      "Z:=", <value>))

<PolylineSegmentsArray>
      Array("NAME:PolylineSegments",
         <OneSegmentArray>, <OneSegmentArray>, ...)

<OneSegmentArray>
      Array("NAME:PLSegment",
         "SegmentType:=", <string>,
         "StartIndex:=", <value>,
         "NoOfPoints:=", <value>)
```

SegmentType

Can be "Line", "Arc", "Spline", or "AngularArc"

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 14.0.0
' 9:49:26 AM  Mar 09, 2011
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
```

```
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project58")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")

oEditor.CreatePolyline Array("NAME:PolylineParameters",
"IsPolylineCovered:=", true,
"IsPolylineClosed:=", false,
Array("NAME:PolylinePoints",
Array("NAME:PLPoint", "X:=", "1.4mm",
"Y:=", "-2mm",
"Z:=", "0mm"),
Array("NAME:PLPoint", "X:=", "0mm", "Y:=", "-2mm", "Z:=", "0mm"),
Array("NAME:PLPoint", "X:=", "-0.8mm", "Y:=", "-1.6mm",
"Z:=", "0mm"),
Array("NAME:PLPoint", "X:=", "-1.4mm", "Y:=", "-0.6mm",
"Z:=", "0mm")),
Array("NAME:PolylineSegments", Array("NAME:PLSegment",
"SegmentType:=", "Line",
"StartIndex:=", 0, "NoOfPoints:=", 2),
Array("NAME:PLSegment", "SegmentType:=", "Line",
"StartIndex:=", 1, "NoOfPoints:=", 2),
Array("NAME:PLSegment", "SegmentType:=", "Line", "StartIndex:=", 2,
"NoOfPoints:=", 2)),
Array("NAME:PolylineXSection", "XSectionType:=", "None",
"XSectionOrient:=", "Auto", "XSectionWidth:=", "0mm",
"XSectionTopWidth:=", "0mm", "XSectionHeight:=", "0mm",
"XSectionNumSegments:=", "0", "XSectionBendType:=", "Corner")),
Array("NAME:Attributes", "Name:=", "Polyline2", "Flags:=", "",
"Color:=", "(132 132 193)",
"Transparency:=", 0,
"PartCoordinateSystem:=", "Global",
"UDMId:=", "", "MaterialValue:=", ""
& Chr(34) & "vacuum" & Chr(34) & "", "SolveInside:=", true)
```

CreateRectangle

Use: Creates a rectangle primitive.

Command: **Draw>Rectangle**

Syntax:
```
CreateRectangle <RectangleParametersArray>,
   <AttributesArray>
```

Return Value: The name of the newly created object.

Parameters:
```
<RectangleParametersArray>
   Array("NAME:RectangleParameters",
      "XStart:=", <value>,
      "YStart:=", <value>,
      "ZStart:=", <value>,
      "Width:=", <value>,
      "Height:=", <value>,
      "WhichAxis:=", <string>)

WhichAxis
```
Axis of normal vector to the rectangle. Possible values are: `"X", "Y", "Z"`
Example: `"WhichAxis:=", "Z"` means the rectangle will be drawn in the XY plane.

CreateRectangle (2D Extractor)

Use: Creates a rectangle primitive.

Command: **Draw>Rectangle**

Syntax:
```
CreateRectangle <RectangleParametersArray>,
   <AttributesArray>
```

Return Value: The name of the newly created object.

Parameters:
```
<RectangleParametersArray>
   Array("NAME:RectangleParameters",
      "XStart:=", <value>,
      "YStart:=", <value>,
      "Width:=", <value>,
      "Height:=", <value>,
      "WhichAxis:=", <string>)
WhichAxis
```

```
Axis of the normal vector to the rectangle. Possible values are: "X",
"Y".
```

CreateRegion

Use: Defines a region containing the design.

Command: **Draw>Create Region**

Syntax: `CreateRegion <RegionParameters> <RegionAttributes>`

Return Value: The name of the newly created object.

Parameters:

```
<RegionParameters>
   Array("NAME:RegionParameters", _
      "+XPaddingType:=", <Offset_Type>,
      "+XPadding:=", "<X_value>", _
      "-XPaddingType:=", <Offset_Type>,
      "-XPadding:=", "<-X_value>", _
      "+YPaddingType:=", <Offset_Type>,
      "+YPadding:=", "<Y_value>",
      "-YPaddingType:=", <Offset_Type>,
      "-YPadding:=", "<-Y_value>", _
      "+ZPaddingType:=", <Offset_Type>,
      "+ZPadding:=", "<Z_value>", _
      "-ZPaddingType:=", <Offset_Type>,
      "-ZPadding:=", "<-Z_value>")
<Offset_Type>
   Type: String
   Can be one of:
   "Percentage Offset",
   "Absolute Offset",
   "Absolute Position",
<RegionAttributes>
   Array("NAME:Attributes",
   "Name:=", "Region", _
   "Flags:=", "Wireframe<# or >", _
   "Color:=", "(<red_int> <green_int> <blue_int>)", _
   "Transparency:=", <real>, _
   "PartCoordinateSystem:=", "<ID>", _
```

```
                "MaterialName:=", "<MaterialName>", _
                "SolveInside:=", <Boolean>)
```

Example:

```
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.CreateRegion Array("NAME:RegionParameters",
"+XPaddingType:=", "Absolute Offset", _
"+XPadding:=", "1um", _
"-XPaddingType:=", "Absolute Offset", _
"-XPadding:=", "1um", _
"+YPaddingType:=", "Absolute Offset", _
"+YPadding:=", "1um",
"-YPaddingType:=", "Absolute Offset", _
"-YPadding:=", "1um", _
"+ZPaddingType:=", "Absolute Offset", _
"+ZPadding:=", "1um", _
"-ZPaddingType:=", "Absolute Offset", _
"-ZPadding:=", "1um"),
Array("NAME:Attributes", "Name:=",  _
"Region", "Flags:=", "Wireframe#", _
"Color:=", "(255 0 0)", _
"Transparency:=", 0, _
"PartCoordinateSystem:=", "Global", _
"MaterialValue:=", "" & Chr(34) & "vacuum" & Chr(34) & "", _
"SolveInside:=", true)
```

For Q3D Extractor the CreateRegion command details are as follows:

Use: Defines a region containing the design.

Command: **Draw>Create Region**

Syntax: `CreateRegion <RegionParameters> <RegionAttributes>`

Return Value: The name of the newly created object.

Parameters:

```
<RegionParameters>
   Array("NAME:RegionParameters", _
      "CoordinateSystemID:=", <ID_number>_
      "+XPadding:=", "<X_value>", _
```

```
                    "-XPadding:=", "<-X_value>", _
                    "+YPadding:=", "<Y_value>",
                    "-YPadding:=", "<-Y_value>", _
                    "+ZPadding:=", "<Z_value>", _
                    "-ZPadding:=", "<-Z_value>")
                <RegionAttributes>
                  Array("NAME:Attributes",
                  "Name:=", "Region", _
                  "Flags:=", "Wireframe<# or >", _
                  "Color:=", "(<red_int> <green_int> <blue_int>)", _
                  "Transparency:=", <real>, _
                  "PartCoordinateSystem:=", "<ID>", _
                  "MaterialName:=", "<MaterialName>", _
                  "SolveInside:=", <Boolean>)
```

```
Set oEditor = oDesign.SetActiveEditor("3D Modeler") oEditor.CreateRe-
gion Array("NAME:RegionParameters", _ "CoordinateSystemID:=", -1, _
"+XPadding:=", "0", "-XPadding:=", "0", _ "+YPadding:=", "0", "-YPad-
ding:=", "0", _ "+ZPadding:=", "0", "-ZPadding:=", "0"), _
Array("NAME:Attributes", "Name:=", "Region", _ "Flags:=", "Wire-
frame#", _ "Color:=", "(255 0 0)", _ "Transparency:=",
0.400000005960464, _ "PartCoordinateSystem:=", "Global", _ "Material-
Name:=", "vacuum", _ "SolveInside:=", true)
```

CreateRegularPolyhedron

Use: Creates a regular polyhedron primitive.

Command: **Draw>Regular Polyhedron**

Syntax:
```
CreateRegularPolyhedron <PolyhedronParametersArray>,
  <AttributesArray>
```

Return Value: The name of the newly created object.

Parameters:
```
<PolyhedronParametersArray>
  Array("NAME:PolyhedronParameters",
    "XCenter:=", <value>,
    "YCenter:=", <value>,
    "ZCenter:=", <value>,
    "XStart:=",  <value>,
```

```
    "YStart:=",  <value>,
    "ZStart:=",  <value>,
    "Height:=",  <value>,
    "NumSides:=", <value>,
    "WhichAxis:=", <string>)
```

`NumSides:`

Specify a number greater than 2.

`WhichAxis`

Axis of the polyhedron. Possible values are: `"X", "Y", "Z"`

Example: `"WhichAxis:=", "Z"`

CreateRegularPolygon

Use: Creates a regular polygon primitive.

Command: **Draw>RegularPolygon**

Syntax:

```
CreateRegularPolygon <PolygonParametersArray>,
  <AttributesArray>
```

Return Value: The name of the newly created object.

Parameters:

```
<PolygonParametersArray>
  Array("NAME:RegularPolygonParameters",
    "XCenter:=",<value>,
    "YCenter:=",<value>,
    "ZCenter:=",<value>,
    "XStart:=", <value>,
    "YStart:=", <value>,
    "ZStart:=", <value>,
    "NumSides:=", "12",
    "WhichAxis:=", <string>)
```

`NumSides`

Specify a number greater than 2.

`WhichAxis`

Axis of normal vector to the polygon. Possible values are: `"X", "Y", "Z"`

Example: `"WhichAxis:=", "Z"` means the polygon will be drawn in the XY plane.

CreateRegularPolyhedron (2D Extractor)

Use: Creates a regular polyhedron primitive.

Command: **Draw>Regular Polyhedron**

Syntax:
```
CreateRegularPolyhedron <PolyhedronParametersArray>,
   <AttributesArray>
```

Return Value: The name of the newly created object.

Parameters:
```
<PolyhedronParametersArray>
   Array("NAME:PolyhedronParameters",
      "XCenter:=", <value>,
      "YCenter:=", <value>,
      "XStart:=",  <value>,
      "YStart:=",  <value>,
      "Height:=",  <value>,
      "NumSides:=", <value>,
      "WhichAxis:=", <string>)
NumSides:
   Specify a number greater than 2.
WhichAxis
   Axis of the polyhedron. Possible values are: "X", "Y"
```

CreateSphere

Use: Creates a sphere primitive.

Command: **Draw>Sphere**

Syntax:
```
CreateSphere <SphereParametersArray>, <AttributesArray>
```

Return Value: The name of the newly created object.

Parameters:
```
<SphereParametersArray>
   Array("NAME:SphereParameters",
      "XCenter:=", <value>,
      "YCenter:=", <value>,
      "ZCenter:=", <value>,
      "Radius:=", <value>)
```

CreateSpiral

Use: Creates a spiral by sweeping the specified 2D objects.

Command: **Draw>Spiral**

Syntax: `CreateSpiral <SelectionsArray>, <SpiralParametersArray>`

Return Value: The name of the newly created object.

Parameters:

```
<SelectionsArray>
   Array("NAME:Selections",
      "Selections:=", <string>)
```

`Selections`

Comma separated list of parts to sweep.

Example: `"Selections:=", "Rect1, Rect2"`

```
<SpiralParametersArray>
   Array("NAME:SpiralParameters",
      "XCenter:=", <value>,
      "YCenter:=", <value>,
      "ZCenter:=", <value>,
      "XStartDir:=", <value>,
      "YStartDir:=", <value>,
      "ZStartDir:=", <value>,
      "NumThread:=", <value>,
      "RightHand:=", <bool>,
      "RadiusIncrement:=", <value>)
```

CreateTorus

Use: Creates a torus primitive.

Command: **Draw>Torus**

Syntax: `CreateTorus <TorusParametersArray>, <AttributesArray>`

Return Value: The name of the newly created object.

Parameters:

```
<TorusParametersArray>
   Array("NAME:TorusParameters",
      "XCenter:=", <value>,
      "YCenter:=", <value>,
      "ZCenter:=", <value>,
      "MajorRadius:=", <value>,
```

```
"MinorRadius:=", <value>,
"WhichAxis:=", <string>)
```

`WhichAxis`

Axis of the torus. Possible values are: `"X", "Y", "Z"`

Example: `"WhichAxis:=", "Z"`

CreateUserDefinedPart

Use: Create User defined part.

Command: **Draw>UserDefinedPrimitive**

Syntax:
```
CreateUserDefinedPart <UserDefinedParametersArray>,
<AttributesArray>
```

Return Value: None

Parameters:
```
<UserDefinedParametersArray>
   Array("NAME:UserDefinedPrimitiveParameters",
    "CoordinateSystemID:=", <value>,
    "DllName:=", <string>,
    "Library:=", <string>,
     Array("NAME:ParamVector",
     Array("NAME:Pair", "Name:=", <string>, "Value:=",
     <value>)…)
```

Example:
```
oEditor.CreateUserDefinedPart
Array("NAME:UserDefinedPrimitiveParameters",
"CoordinateSystemID:=", -1,
"DllName:=", "Examples/RectangularSpiral",
"NoOfParameters:=", 6, "Library:=", "syslib",
Array("NAME:ParamVector",
Array("NAME:Pair", "Name:=", "Xpos", "Value:=", "0mm"),
Array("NAME:Pair", "Name:=", "Ypos", "Value:=", "0mm"),
Array("NAME:Pair", "Name:=", "TurnSep", "Value:=", "5mm"),
Array("NAME:Pair", "Name:=", "Turns", "Value:=", "2"),
Array("NAME:Pair", "Name:=", "Width", "Value:=", "2mm"),
Array("NAME:Pair", "Name:=", "Height", "Value:=", "2mm"))),
Array("NAME:Attributes", "Name:=", "RectangularSpiral1",
"Flags:=", "",
"Color:=", "(132 132 193)",
"Transparency:=", 0,
"PartCoordinateSystem:=", "Global",
```

```
"MaterialName:=", "copper",
"SolveInside:=",  _
              false)
```

Edit3DComponent

Use: Edit 3D Component

Command: None

Syntax: `Edit3DComponent <Component Name>`

Return Value: None

Parameters:

`<Component Name>`

Name of the 3D component

`<Data>`

Data of the 3D component

Example:

```
oEditor.Edit3DComponent "Connector1",
Array("NAME:EditComponentParametersData",
"NewComponentName:=",   " Connector2",
"GeometryParameters:=", "",
"MaterialParameters:=", "",
"DesignParameters:=", "",
Array("NAME:Component Meshing", "MeshAssembly:=", false),
Array("NAME:Excitations", "Suppressed:=", Array()))
```

EditPolyline

Use: Modifies a polyline primitive. Specify the name of the polyline to modify and the new set of data for the polyline.

Command: **Draw>Line Segment>Insert Segment Before>Straight**

Draw>Line Segment>Insert Segment Before>Spline

Draw>Line Segment>Insert Segment Before>3 Point Arc

Draw>Line Segment>Insert Segment Before>Center Point Arc

Draw>Line Segment>Insert Segment After>Straight

Draw>Line Segment>Insert Segment After>Spline

Draw>Line Segment>Insert Segment After>3 Point Arc

Draw>Line Segment>Insert Segment After>Center Point Arc

Edit>Delete Start Point

Edit>Delete End Point.

Syntax:

```
EditPolyline <SelectionsArray>,
  <PolylineParametersArray>,
```

Return Value: The name of the newly created object

Parameters:

```
<SelectionsArray>
   Array("NAME:Selections",
      "Selections:=", "string")
```

`Selections`

Name of the polyline to modify. The name should be formatted as `"<PolylineName>:CreatePolyline:1"`.

Example: `"Selections:=", "Polyline1:CreatePolyline:1"`

Get3DComponentParameters

Use: Get parameters of 3D components

Command: none

Syntax: `Get3DComponentParameters <Component Name>`

Return Value: An array of component parameters.

Parameters: `<Component Name>`

Name of the 3D component

Example:

```
Dim paras
paras = oEditor.Get3DComponentParameters("Connector")
```

Get3DComponentDefinitionNames

Use: Get names of 3D component definitions

Command: None

Syntax: `Get3DComponentDefinitionNames`

Return Value: An array of component definition names.

Parameters:

Example:

```
Dim defNames
defNames = oEditor.Get3DComponentDefinitionNames()
```

Get3DComponentInstanceNames

Use: Get instance names of 3D component definitions.

Command: None

Syntax: `An array of 3D component instance names.`

Return Value: An array of 3D component instance names.

Parameters: `<Definition Name>`

Name of the 3D component definition

Example:

```
Dim instNames
instNames = oEditor.Get3DComponentInstanceNames("Connector")
```

Get3DComponentMaterialNames

Use: Get material names of 3D component.

Command: None.

Syntax: `Get3DComponentMaterialNames <Component Instance Name>`

Return Value: An array of material names.

Parameters: `<Component Instance Name>`

Name of the component

Example:

```
Dim materialNames
materialNames = oEditor.Get3DComponentMaterialNames("Connector1")
```

Get3DComponentMaterialProperties

Use: Get material properties of 3D component material.

Command: None

Syntax: `Get3DComponentMaterialProperties <Complete Material Name>`

Return Value: An array of material properties.

Parameters: `<Complete Material Name>`

Name of the material with a complete format

Example:

```
Dim materialProperties
materialProperties = oEditor.Get3DComponentMaterialProperties("Con-
nector1:Material01")
```

Insert3DComponent

Use: Insert a 3D Component

Command: none

Syntax: `Insert3DComponent <Data>`

Return Value: None

Parameters: `<Data>`

Data of the 3D component

Example:

```
oEditor.Insert3DComponent Array("NAME:InsertComponent-
Data",
"Parameters:=", "",
"TargetCS:=",   "Global",
"ComponentFile:=", "C:\tmp\Connector.a3dcomp")
```

InsertPolylineSegment

Use: Inserts a polyline segment either before or after an existing segment of a polyline primitive.

Command: **Draw>Line Segment>Insert Segment Before>Straight**
Draw>Line Segment>Insert Segment Before>Spline
Draw>Line Segment>Insert Segment Before>3 Point Arc
Draw>Line Segment>Insert Segment Before>Center Point Arc
Draw>Line Segment>Insert Segment After>Straight
Draw>Line Segment>Insert Segment After>Spline
Draw>Line Segment>Insert Segment After>3 Point Arc
Draw>Line Segment>Insert Segment After>Center Point Arc

Syntax: `InsertPolylineSegment <InsertPolylineSegmentArray>`

Return Value: None

Parameters:

```
<InsertPolylineSegmentArray>
   Array("Name:Insert Polyline Segment",
      "Selections:=", <string>,
      "Segment Index:=", <value>,
      "At Start:=", <bool>,
      "SegmentType:=", <string>
      <PolylinePointsArray>)

<PolylinePointsArray>
   Array("Name:Polyline Points", <OnePointArray>,
      <OnePointArray>, ...)

<OnePointArray>
   Array("Name:PLPoint",
      "X:=", <value>,
      "Y:=", <value>,
      "Z:=", <value>)
```

```
Selections
```

Name of the polyline to modify. The name should be formatted as

```
"<PolylineName>:CreatePolyline:1".
Example: "Selections:=", "Polyline1:CreatePolyline:1"

SegmentType
  Can be "Line", "Arc", "Spline", or "AngularArc"
```

SweepAlongPath

Use: Sweeps the specified 1D or 2D parts along a path. The last 1D object specified is the path for the sweep.

Command: **Draw>Sweep>Along Path**

Syntax:

```
SweepAlongPath <SelectionsArray>,
  <PathSweepParametersArray>
```

Return Value: None

Parameters:

```
<PathSweepParametersArray>
  Array("NAME:PathSweepParameters",
    "DraftAngle:=", <value>,
    "DraftType:=", <string>,
    "TwistAngle:=", <value>)

DraftType
```

Possible values are `"Extended"`, `"Round"`, `"Natural"`

Example:

```
oEditor.SweepAlongPath _
  Array("NAME:Selections", "Selections:=",
    "Polygon1,Polyline1"),_
  Array("NAME:PathSweepParameters", _
    "DraftAngle:=", "0deg",_
    "DraftType:=", "Round",_
    "TwistAngle:=", "30deg")
```

SweepAlongVector

Use: Sweeps the specified 1D or 2D parts along a vector.

Command: **Draw>Sweep>Along Vector**

Syntax:
```
SweepAlongVector <SelectionsArray>,
   <VecSweepParametersArray>
```

Return Value: None

Parameters:
```
<VecSweepParametersArray>
   Array("NAME:VectorSweepParameters",
      "DraftAngle:=", <value>,
      "DraftType:=", <string>,
      "SweepVectorX:=", <value>, _
      "SweepVectorY:=", <value>,
      "SweepVectorZ:=", <value)

DraftType
```
Possible values are `"Extended"`, `"Round"`, `"Natural"`

SweepAroundAxis

Use: Sweeps the specified 1D or 2D parts around an axis.

Command: **Draw>Sweep>Around Axis**

Syntax:
```
SweepAroundAxis <SelectionsArray>,
   <AxisSweepParametersArray>
```

Return Value: None

Parameters:
```
<AxisSweepParametersArray>
   Array("NAME:AxisSweepParameters",
      "DraftAngle:=",  <value>,
      "DraftType:=", <string>,
      "SweepAxis:=", <string>,
      "SweepAngle:=", <value>)

DraftType
```
Possible values are `"Extended"`, `"Round"`, `"Natural"`

```
SweepAxis
```
Possible values are `"X"`, `"Y"`, `"Z"`

SweepFacesAlongNormal

Use: Sweep selected faces along normal to create new object.

Command: **Modeler>Surface>Sweep Faces Along Normal**

Syntax:
```
SweepFacesAlongNormal <SelectionsArray>,
<ParametersArray>
```

Return Value: None

Parameters:
```
<ParametersArray>
   Array("NAME:SweepFaceAlongNormalToParameters",
   "FacesToDetach:=", Array(<FaceID>),
   "LengthOfSweep:=", "<Value><units>"))
```

Example:
```
oEditor.SweepFacesAlongNormal Array("NAME:Selections",
"Selections:=", "Box1",
"NewPartsModelFlag:=",  "Model"),
Array("NAME:Parameters",
Array("NAME:SweepFaceAlongNormalToParameters",
"FacesToDetach:=", Array(  12),
"LengthOfSweep:=", "0.1mm"))
```

SweepFacesAlongNormalWithAttributes

Use: Sweep selected faces along normal to create new object, user can specify the attributes of the new object

Command: Recorded during PML creation

Syntax:
```
SweepFacesAlongNormalWidthAttributes <SelectionsArray>,
<ParametersArray>, <AttributeArray>
```

Return Value: None

Parameters:
```
<ParametersArray>
     Array("NAME:SweepFaceAlongNormalToParameters",
     "FacesToDetach:=", Array(12),
     "LengthOfSweep:=", "0.1mm"))
```

Example:

oEditor.SweepFacesAlongNormalWithAttributes Array("NAME:Selections",
"Selections:=","Box1",
"NewPartsModelFlag:=", "Model"),
Array("NAME:Parameters",
Array("NAME:SweepFaceAlongNormalToParameters", "FacesToDetach:=", Array(7),
"LengthOfSweep:=", "0.424865556413828mm")),
Array("NAME:Attributes", "Name:=", "PML_Box1_1", "Flags:=", "",
"Color:=", "(132 132 193)",
"Transparency:=", 0.899999976158142,
"PartCoordinateSystem:=", "FaceCS1",

```
"UDMId:=", "",
"MaterialValue:=",  "" & Chr(34) & "vacuum" & Chr(34) & "",
"SolveInside:=", true)
```

UpdateComponentDefinition

Use: Update component definition

Command: None

Syntax: `UpdateComponentDefinition <Data>`

Return Value: none

Parameters: `<Data> Component data`

Example:

```
oEditor.UpdateComponentDefinition
Array("NAME:UpdateDefinitionData",
"DefinitionNames:=", " Connector, Magic_Tee",
"Passwords:=", Array("", ""))
```

Edit Menu Commands

Copy
DeletePolyLinePoint
DuplicateAlongLine
DuplicateAroundAxis
DuplicateAroundAxis (2D Extractor)
DuplicateMirror
DuplicateMirror (2D Extractor)
Mirror
Mirror (2D Extractor)
Move
OffsetFaces
Paste
Rotate
Rotate (2DExtractor)
Scale
Scale (2D Extractor)

Copy

Use: Copies specified parts.

Command: **Edit>Copy**

Syntax: `Copy <SelectionsArray>`

Return Value: None

DeletePolylinePoint

Use: Deletes either a start or end point from an existing polyline segment.

Command: **Edit>Delete Start Point**
Edit>Delete End Point

Syntax:
```
DeletePolylinePoint <DeletePointArray>
```

Return Value: None

Parameters:
```
<DeletePointArray>
   Array("Name:Delete Point",
      "Selections:=", <string>,
      "Segment Index:=", <value>,
      "At Start:=", <bool>)
```

`Selections`

Name of the polyline to modify. The name should be formatted as `"<PolylineName>:CreatePolyline:1"`.

`Example: "Selections:=", "Polyline1:CreatePolyline:1"`

DuplicateAlongLine

Use: Duplicates specified parts along line.

Command: **Edit>Duplicate>Along Line**

Syntax:
```
DuplicateAlongLine <SelectionsArray>,
   <DupLineParametersArray> <DupBoundariesArray>
```

Return Value: None

Parameters:
```
<DupLineParametersArray>
   Array("NAME:DuplicateToAlongLineParameters",
      "XComponent:=", <value>,
      "YComponent:=", <value>,
      "ZComponent:=", <value>,
      "NumClones:=", <value>)
```

`NumClones`

Specify a number greater than 1.

`<DupBoundariesArray>`

```
Array("NAME:Options", "DuplicateBoundaries:=", <bool>)
```

Example:

```
oEditor.DuplicateAlongLine Array("NAME:Selections", _
"Selections:=", "Box1", _
"NewPartsModelFlag:=", "Model"), _
Array("NAME:DuplicateToAlongLineParameters", _
"CreateNewObjects:=", true, "XComponent:=", "0mm", _
"YComponent:=", "1.2mm", _
"ZComponent:=", "0mm", _
"NumClones:=", "2"), _
Array("NAME:Options", "DuplicateBoundaries:=", false)
```

DuplicateAroundAxis

Use: Duplicates specified parts around an axis.

Command: **Edit>Duplicate>Around Axis**

Syntax:

```
DuplicateAroundAxis <SelectionsArray>,
  <DupAxisParametersArray> <DupBoundariesArray>
```

Return Value: None

Parameters:

```
<DupAxisParametersArray>
  Array("NAME:DuplicateAroundAxisParameters",
    "WhichAxis:=", <string>,
    "AngleStr:=", <value>,
    "NumClones:=",<value>)
```

`WhichAxis`

Axis to duplicate around. Possible values are: `"X", "Y", "Z"`

Example: `"WhichAxis:=", "Z"`

`NumClones:`

Specify a number greater than 1.

```
<DupBoundariesArray>
Array("NAME:Options", "DuplicateBoundaries:=", <bool>)
```

Example:

```
oEditor.DuplicateAroundAxis Array("NAME:Selections", _
"Selections:=", "Box1", _
"NewPartsModelFlag:=", "Model"), _
Array("NAME:DuplicateAroundAxisParameters", _
"CreateNewObjects:=", false, _
"WhichAxis:=", "Z", _
"AngleStr:=", "90deg", _
"NumClones:=", "2"), _
Array("NAME:Options", "DuplicateBoundaries:=", false)
```

DuplicateAroundAxis (2D Extractor)

Use: Duplicates specified parts around an axis.

Command: **Edit>Duplicate>Around Axis**

Syntax:
```
DuplicateAroundAxis <SelectionsArray>,
   <DupAxisParametersArray>
```

Return Value: None

Parameters:
```
<DupAxisParametersArray>
   Array("NAME:DuplicateAroundAxisParameters",
      "WhichAxis:=", <string>,
      "AngleStr:=", <value>,
      "NumClones:=",<value>)
WhichAxis
   Axis to duplicate around. Possible values are: "X",
   "Y".
NumClones:
   Specify a number greater than 1.
```

DuplicateMirror

Use: Duplicate specified parts according to a mirror plane.

Command: **Edit>Duplicate>Mirror**

Syntax:
```
DuplicateMirror <SelectionsArray>,
   <DupMirrorParametersArray>
```

Return Value: None

Parameters:
```
<DupMirrorParametersArray>
   Array("NAME:DuplicateToMirrorParameters",
```

```
                    "DuplicateMirrorBaseX:=", <value>,
                    "DuplicateMirrorBaseY:=", <value>,
                    "DuplicateMirrorBaseZ:=", <value>,
                    "DuplicateMirrorNormalX:=", <value>,
                    "DuplicateMirrorNormalY:=", <value>,
                    "DuplicateMirrorNormalZ:=", <value>)

                <DupBoundariesArray>
                Array("NAME:Options", "DuplicateBoundaries:=", <bool>)
```

Example:

```
oEditor.DuplicateMirror Array("NAME:Selections", _
"Selections:=", "Box1", _
"NewPartsModelFlag:=", "Model"), _
Array("NAME:DuplicateToMirrorParameters", _
"DuplicateMirrorBaseX:=", "0mm", _
"DuplicateMirrorBaseY:=", "0mm", _
"DuplicateMirrorBaseZ:=", "0mm", _
"DuplicateMirrorNormalX:=", "0mm", _
"DuplicateMirrorNormalY:=", "-1mm", _
"DuplicateMirrorNormalZ:=", "0mm"), _
Array("NAME:Options", "DuplicateBoundaries:=", false)
```

DuplicateMirror (2D Extractor)

Use: Duplicates specified parts according to a mirror plane.

Command: **Edit>Duplicate>Mirror**

Syntax:

```
DuplicateMirror <SelectionsArray>,
   <DupMirrorParametersArray>
```

Return Value: None

Parameters:

```
<DupMirrorParametersArray>
   Array("NAME:DuplicateToMirrorParameters",
      "DuplicateMirrorBaseX:=", <value>,
      "DuplicateMirrorBaseY:=", <value>,
      "DuplicateMirrorNormalX:=", <value>,
```

```
"DuplicateMirrorNormalY:=", <value>,
```

Mirror

Use: Mirrors specified parts.

Command: **Edit>Arrange>Mirror**

Syntax: `Mirror <SelectionsArray>, <MirrorParametersArray>`

Return Value: None

Parameters:

```
<MirrorParametersArray>
   Array("NAME:MirrorParameters",
      "MirrorBaseX:=", <value>,
      "MirrorBaseY:=", <value>,
      "MirrorBaseZ:=", <value>,
      "MirrorNormalX:=", <value>,
      "MirrorNormalY:=", <value>,
      "MirrorNormalZ:=", <value>)
```

Example:

```
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.Mirror Array("NAME:Selections", "Selections:=", "Box1", _
"NewPartsModelFlag:=", "Model"), _
Array("NAME:MirrorParameters", _
"MirrorBaseX:=", "-0.8mm", _
"MirrorBaseY:=", "-1mm", _
"MirrorBaseZ:=", "0mm", _
"MirrorNormalX:=", "0.948683298050514mm", _
"MirrorNormalY:=", "-0.316227766016838mm", _
"MirrorNormalZ:=", "0mm")
```

Mirror (2D Extractor)

Use: Mirrors specified parts.

Command: **Edit>Arrange>Mirror**

Syntax: `Mirror <SelectionsArray>, <MirrorParametersArray>`

Return Value: None

Parameters:

```
<MirrorParametersArray>
   Array("NAME:MirrorParameters",
      "MirrorBaseX:=", <value>,
```

```
                    "MirrorBaseY:=", <value>,
                    "MirrorNormalX:=", <value>,
"MirrorNormalY:=", <value>,
```

Move

Use: Moves specified parts.

Command: **Edit>Arrange>Move**

Syntax:
```
Move <SelectionsArray>, <MoveParametersArray>
```

Return Value: None

Parameters:
```
<MoveParametersArray>
   Array("NAME:TranslateParameters",
      "TranslateVectorX:=", <value>,
      "TranslateVectorY:=", <value>,
      "TranslateVectorZ:=", <value>)
```

OffsetFaces

Use: Offsets faces of specified parts.

Command: **Edit>Arrange>Offset**

Syntax:
```
OffsetFaces <SelectionsArray>, <OffsetParametersArray>
```

Return Value: None

Parameters:
```
<OffsetParametersArray>
   Array("NAME:OffsetParameters",
      "OffsetDistance:=", <value>)
```

Paste [Model Editor]

Use: Pastes copied objects and returns an array of pasted objects from the 3D model editor.

Command: **Edit>Paste**

Syntax:
```
Paste
```

Return Value: One dimensional array of pasted object names. The order is not guarenteed to be alphabetical.

Parameters:
```
None.
```

Example:
```
arrayEntities = oEditor.Paste
```

Rotate

Use: Rotates specified parts.

Command: **Edit>Arrange>Rotate**

Syntax:
```
Rotate <SelectionsArray>, <RotateParametersArray>
```

Return Value: None

Parameters:
```
<RotateParametersArray>
   Array("NAME:RotateParameters",
      "RotateAxis:=", <string>
      "RotateAngle:=", <value>)

RotateAxis
```
Possible values are: `"X"`, `"Y"`, `"Z"`

Rotate (2D Extractor)

Use: Rotates specified parts.

Command: **Edit>Arrange>Rotate**

Syntax:
```
Rotate <SelectionsArray>, <RotateParametersArray>
```

Return Value: None

Parameters:
```
<RotateParametersArray>
   Array("NAME:RotateParameters",
      "RotateAxis:=", <string>
      "RotateAngle:=", <value>)
RotateAxis
   Possible values are: "X", "Y".
```

Scale

Use: Scales specified parts.

Command: **Edit>Scale**

Syntax:
```
Scale <SelectionsArray>, <ScaleParametersArray>
```

Return Value: None

Parameters:
```
<ScaleParametersArray>
   Array("NAME:ScaleParameters",
      "ScaleX:=", <value>,
      "ScaleY:=", <value>,
      "ScaleZ:=", <value>)
```

Scale (2D Extractor)

Use: Scales specified parts.

Command: **Edit>Scale**

Syntax: `Scale <SelectionsArray>, <ScaleParametersArray>`

Return Value: None

Parameters:

```
<ScaleParametersArray>
   Array("NAME:ScaleParameters",
      "ScaleX:=", <value>,
      "ScaleY:=", <value>,
```

Modeler Menu Commands

AssignMaterial
Chamfer
Connect
CoverLines
CoverSurfaces
CreateEntityList
CreateFaceCS
CreateFaceCS (2D Extractor)
CreateObjectCS
CreateObjectFromEdge
CreateObjectFromFaces
CreateRelativeCS
CreateRelativeCS(2D Extractor)
DeleteLastOperation
DetachFaces
EditEntityList
EditFaceCS
EditObjectCS
EditRelativeCS
EditRelativeCS (2DExtractor)
Export
Fillet
Generate History
GetActiveCoordinateSystem

AssignMaterial

Use: Assigns a material to the specified objects. Only the `MaterialName` and `SolveInside` parameters of `<AttributesArray>` are supported.

Command: **Modeler>Assign Material**

Syntax: `AssignMaterial <SelectionsArray>, <AttributesArray>`

Return Value: None

Example:

```
oEditor.AssignMaterial _
   Array("NAME:Selections", "Selections:=", "Polygon1"),
   Array("NAME:Attributes", _
      "MaterialName:=", "tungsten",_
      "SolveInside:=", false)
```

Chamfer

Use: Creates a chamfer.

Command: Modeler>Chamfer

Syntax: `Chamfer (<ObjectName> <ChamferParameters>)`

Return Value: None

Parameters:

```
<ObjectName>
   Array("NAME:Selections", _
      "Selections:=", <string>),
         <ChamferParameters>
      Array("NAME:Parameters", _
         Array("NAME:ChamferParameters", _
            "CoordinateSystemID:=", <value>,
            "Edges:=", <ArrayOfEdgeIDs>,
            "LeftRange:=", <value>))
```

Example:

```
oEditor.Chamfer Array("Name:Selections", _
"Selections:=", "Box1"), Array("NAME:Parameters", _
Array("NAME:ChamferParameters", _
"CoordinateSystemID:=", -1, _
"Edges:=", Array(13), "LeftRange:=", "1mm"))
```

Connect

Use: Connects specified 1D parts to form a sheet.

Command: **Modeler>Surface>Connect**

Syntax: `Connect <SelectionsArray>`

Return Value: None

CoverLines

Use: Covers the specified 1D objects to form a sheet.

Command: **Modeler>Surface>Cover Lines**

Syntax: `CoverLines <SelectionsArray>`

Return Value: None

CoverSurfaces

Use: Covers the specified objects to form a solid object.

Command: **Modeler>Surface>Cover Faces**

Syntax:
```
CoverSurfaces <SelectionsArray>
```

Return Value: None

CreateEntityList

Use: Creates a list of entities. The list can contain objects or faces, but not both. Only the `Name` attribute from `<AttributesArray>` is supported.

Command: **Modeler>List>Create>Object List**

Modeler>List>Create>Face List

Syntax:
```
CreateEntityList <EntityListParametersArray>,
  <AttributesArray>
```

Return Value: None

Parameters:
```
<EntityListParametersArray>
  Array("NAME:GeometryEntityListParameters",
    "EntityType:=", <string>,
    "EntityList:=", <array>
```

`EntityType`

Possible values are `"Object"`, `"Face"`

`EntityList`

Array of integers – the IDs of the objects or faces to put in the list. To get the IDs, use GetObjectIDByName

CreateFaceCS

Use: Creates a face coordinate system. Only the `Name` attribute of the `<AttributesArray>` parameter is supported.

Command: **Modeler>Coordinate System>Create>Face CS**

Syntax:
```
CreateFaceCS <FaceCSParametersArray>, <AttributesArray>
```

Return Value: None

Parameters:
```
<FaceCSParametersArray>
  Array("NAME:FaceCSParameters",
    "FaceID:=", <int>,
```

```
"PartID:=", <int>,
Array("NAME:OriginPosn",
   "IsAttachedToEntity:=", <bool>,
   "EntityID:=", <value>,
   "PositionType:=", <string>,
   "UParam:=", <value>,
   "VParam:=", <value>,
   "XPosition:=", <value>,
   "YPosition:=", <value>,
   "ZPosition:=", <value>)
Array("NAME:AxisPosn",
   "IsAttachedToEntity:=", <bool>
   "EntityID:=",  <value>
   "PositionType:=", <string>,
   "UParam:=", <value>,
   "VParam:=", <value>,
   "XPosition:=", <value>,
   "YPosition:=", <value>,
   "ZPosition:=", <value>)
   "WhichAxis:=", <string>)
```

`FaceID`
ID of the face on which to create the coordinate system.

`PartID`
ID of the object on which the face ID lies.

`IsAttachedToEntity`
Specifies whether the point is anchored (to a vertex, edge, or face).
If `IsAttachedToEntity` is true, provide the `UParam` and `VParam` parameters. Otherwise, provide the `XPosition`, `YPosition`, and `ZPosition` parameters.

`EntityID`
ID of the vertex, edge, or face to which the point is anchored.

`PositionType`

Place where the point is anchored.

Possible values are: `"FaceCenter"`, `"EdgeCenter"`, `"OnVertex"`, `"OnEdge"`, `"OnFace"`

`UParam, VParam`

Numbers between 0 and 1 representing the relative position of the point on the edge or face.

Example: `UParam = .5`, `VParam = .5` would be the center of a face.

`XPosition, YPosition, ZPosition`

Fixed position of the point.

`WhichAxis`

Possible values are `"X"`, `"Y"`, `"Z"`

CreateFaceCS (2D Extractor)

Use: Creates a face coordinate system. Only the `Name` attribute of the `<AttributesArray>` parameter is supported.

Command: **Modeler>Coordinate System>Create>Face CS**

Syntax: `CreateFaceCS <FaceCSParametersArray>, <AttributesArray>`

Return Value: None

Parameters:

```
<FaceCSParametersArray>
   Array("NAME:FaceCSParameters",
      "FaceID:=", <int>, "PartID:=", <int>,
      Array("NAME:OriginPosn",
         "IsAttachedToEntity:=", <bool>,
         "EntityID:=", <value>,
         "PositionType:=", <string>,
         "UParam:=", <value>,
         "VParam:=", <value>,
         "XPosition:=", <value>,
         "YPosition:=", <value>)
      Array("NAME:AxisPosn",
         "IsAttachedToEntity:=", <bool>
```

```
        "EntityID:=",  <value>
        "PositionType:=", <string>,
        "UParam:=", <value>,
        "VParam:=", <value>,
        "XPosition:=", <value>,
        "YPosition:=", <value>)
        "WhichAxis:=", <string>)
```

FaceID

ID of the face on which to create the coordinate system.

PartID

ID of the object on which the face ID lies.

IsAttachedToEntity

Specifies whether the point is anchored (to a vertex, edge, or face).

If IsAttachedToEntity is true, provide the UParam and VParam parameters. Otherwise, provide the XPosition and YPosition parameters.

EntityID

ID of the vertex, edge, or face to which the point is anchored.

PositionType

Place where the point is anchored.

Possible values are: "FaceCenter", "EdgeCenter", "OnVertex", "OnEdge", "OnFace"

UParam, VParam

Numbers between 0 and 1 representing the relative position of the point on the edge or face.

Example: UParam = .5, VParam = .5 would be the center of a face.

XPosition, YPosition

Fixed position of the point.

WhichAxis

Possible values are "X", "Y".

CreateObjectCS

Use: Creates an Object coordinate system.

Command: **Modeler>Coordinate System>Create>Object><Offset | Rotated | Both>**

Syntax: `CreateObjectCS <ParameterArrays>`

Return Value: None

Parameters:

```
Array("NAME:ObjectCSParameters",
    "PartID:=", <ID>,
    "ReverseXAxis:=", <Boolean>,
    "ReverseYAxis:=", <Boolean>,
  Array("NAME:Origin",
    "IsAttachedToEntity:=", <Boolean>,
    "EntityID:=", <ID>,
    "PositionType:=", "<OnPositionID>,
        String, one of OnVertex, FaceCenter, OnEdge, AbsolutePosition
    "UParam:=", <integer>,
    "VParam:=", <Integer>,
    "XPosition:=", "<Integer>",
    "YPosition:=", "<Integer>",
    "ZPosition:=", "<Integer>"),
  Array("NAME:xAxis",
    "DirectionType:=", "AbsoluteDirection",
    "EdgeID:=", <Int>,
    "FaceID:=", <Int>,
    "xDirection:=", "<int>",
    "yDirection:=", "<Int>",
    "zDirection:=", "<Int>",
    "UParam:=", <int>,
    "VParam:=", <Int>),
  Array("NAME:yAxis",
    "DirectionType:=", "AbsoluteDirection",
    "EdgeID:=", <Int>,
    "FaceID:=", <int>,
    "xDirection:=", "<int>",
    "yDirection:=", "<int>",
    "zDirection:=", "<int>",
    "UParam:=", <int>,
    "VParam:=", <int>)),
  Array("NAME:Attributes",
```

"Name:=", "<ObjectCSName>")

Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("coax_bend")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.CreateObjectCS Array("NAME:ObjectCSParameters",
"PartID:=", 24,
"ReverseXAxis:=", false,
"ReverseYAxis:=", false,
Array("NAME:Origin",
"IsAttachedToEntity:=",
true, "EntityID:=", 30,
"PositionType:=", "OnVertex",
"UParam:=", 0,
"VParam:=", 0,
"XPosition:=", "0",
"YPosition:=", "0",
"ZPosition:=", "0"),
Array("NAME:xAxis",
"DirectionType:=", "AbsoluteDirection",
"EdgeID:=", -1,
"FaceID:=", -1,
"xDirection:=", "1",
"yDirection:=", "0",
"zDirection:=", "0",
"UParam:=", 0,
"VParam:=", 0),
Array("NAME:yAxis",
"DirectionType:=", "AbsoluteDirection",
"EdgeID:=", -1,
"FaceID:=", -1,
```

```
    "xDirection:=", "0",
    "yDirection:=", "1",
    "zDirection:=", "0",
    "UParam:=", 0, "VParam:=", 0)),
    Array("NAME:Attributes",
    "Name:=", "ObjectCS1")
```

CreateObjectFromEdges

Use: Creates a polyline from the specified object edge.

Command: **Modeler>Create Object From Edge**

Syntax:

```
CreateObjectFromEdges <SelectionsArray>,
   <ObjFromEdgeParametersArray>
```

Return Value: None

Parameters:

```
<SelectionsArray>
   Array("NAME:Selections",
      "Selections:=" <ObjName>)

<ObjFromEdgeParametersArray>
   Array("NAME:Parameters",
      <EdgeParametersArray>)

<EdgeParametersArray>
   Array("Name:BodyFromEdgeToParameters",
      "CoordinateSystemID:=", <int>,
      "Edges:=", <EdgeIDarray>)
```

Example:

```
oEditor.CreateEdgeFromEdges _
   Array("NAME:Selections", "Selections:=", "Box1"),_
   Array("NAME:Parameters", _
      Array("NAME:BodyFromEdgeToParameters", _
         "CoordinateSystemID:=", -1, _
         "Edges:=", Array(13)))
```

CreateObjectFromFaces

Use: Creates 2D objects from the specified faces.

Command: **Modeler>Surface>Create Object From Face**

Syntax:

```
CreateObjectFromFaces <SelectionsArray>,
   <ObjFromFaceParametersArray>
```

Return Value: None

Parameters:

```
<ObjFromFaceParametersArray>
   Array("NAME:Parameters",
      <FacesOfOneObjToDetach>, <FacesOfOneObjToDetach>,
         ...)

<FacesOfOneObjToDetach>
   Array("Name:BodyFromFaceToParameters",
       "FacesToDetach:=", <array>)

FacesToDetach
```

Array of integers – the IDs of the faces to use to create objects.

Example:

```
oEditor.CreateObjectFromFaces _
   Array("NAME:Selections", "Selections:=", "Box1"),_
   Array("NAME:Parameters", _
      Array("NAME:BodyFromFaceToParameters", _
         "FacesToDetach:=", Array(185)))
```

CreateRelativeCS

Use: Creates a relative coordinate system. Only the `Name` attribute of the `<AttributesArray>` parameter is supported.

Command: **Modeler>Coordinate System>Create>Relative CS->Offset**
Modeler>Coordinate System>Create>Relative CS->Rotated
Modeler>Coordinate System>Create>Relative CS->Both

Syntax:

```
CreateRelativeCS <RelativeCSParametersArray>,
   <AttributesArray>
```

Return Value: None

Parameters:

```
<RelativeCSParametersArray>
   Array("NAME:RelativeCSParameters",
      "OriginX:=", <value>,
      "OriginY:=", <value>,
      "OriginZ:=", <value>,
```

```
    "XAxisXvec:=", <value>,
    "XAxisYvec:=", <value>,
    "XAxisZvec:=", <value>,
    "YAxisXvec:=", <value>,
    "YAxisYvec:=", <value>,
    "YAxisZvec:=", <value>)
```

CreateRelativeCS (2D Extractor)

Use: Creates a relative coordinate system. Only the `Name` attribute of the `<AttributesArray>` parameter is supported.

Command: **Modeler>Coordinate System>Create>Relative CS>Offset**

Modeler>Coordinate System>Create>Relative CS>Rotated

Modeler>Coordinate System>Create>Relative CS>Both

Syntax:

```
CreateRelativeCS <RelativeCSParametersArray>,
  <AttributesArray>
```

Return Value: None

Parameters:

```
<RelativeCSParametersArray>
  Array("NAME:RelativeCSParameters",
    "OriginX:=", <value>,
    "OriginY:=", <value>,
    "XAxisXvec:=", <value>,
    "XAxisYvec:=", <value>,
    "YAxisXvec:=", <value>,
      "YAxisYvec:=", <value>)
```

DeleteLastOperation

Use: Deletes the last operation for specified objects.

Command: **Modeler>Delete Last Operation**

Syntax:

```
DeleteLastOperation <SelectionsArray>
```

Return Value: None

DetachFaces

Use: Detaches the specified faces.

Command: **Modeler>Surface>Detach Faces**

Syntax:

```
DetachFaces <SelectionsArray>,
  <DetachFacesParametersArray>
```

Return Value: None

Parameters:

```
<DetachFacesParametersArray>
   Array("NAME:Parameters",
      <FacesOfOneObjToDetach>,
      <FacesOfOneObjToDetach>, ...)

<FacesOfOneObjToDetach>
   Array("Name:DetachFacesToParameters",
      "FacesToDetach:=", <array>)

FacesToDetach
```

An array of integers – the face IDs of the faces to detach.

Example:

```
oEditor.DetachFaces _
   Array("NAME:Selections", "Selections:=",_
      "Box5,Box4"),_
   Array("NAME:Parameters", _
      Array("NAME:DetachFacesToParameters", _
         "FacesToDetach:=", Array(123, 122)),
   Array("NAME:DetachFacesToParameters", _
      "FacesToDetach:=", Array(94)))
```

EditEntityList

Use: Modifies an entity list.

Command: **Modeler>List>Reassign**

Syntax:

```
EditEntityList <SelectionsArray>,
   <EntityListParametersArray>
```

Return Value: None

EditFaceCS

Use: Recreates an existing face coordinate system. The name of the coordinate system to modify should be specified in the `<AttributesArray>` parameter.

Command: **Modeler->Coordinate System->Edit**

Syntax:

```
EditFaceCS <FaceCSParametersArray>, <AttributesArray>
```

Return Value: None

EditObjectCS

Use: Edit an existing Object CS.

Command: **Modeler>Coordinate System>Edit**

Syntax: `EditObjectCS <Array>`

Return Value: None

Parameters:

```
Array("NAME:ObjectCSParameters",
    "PartID:=", <ID>,
    "ReverseXAxis:=", <Boolean>,
    "ReverseYAxis:=", <Boolean>,
  Array("NAME:Origin",
    "IsAttachedToEntity:=", <Boolean>,
    "EntityID:=", <ID>,
    "PositionType:=", "<OnPositionID>,
        String, one of OnVertex, FaceCenter, OnEdge, AbsolutePosition
    "UParam:=", <integer>,
    "VParam:=", <Integer>,
    "XPosition:=", "<Integer>",
    "YPosition:=", "<Integer>",
    "ZPosition:=", "<Integer>"),
  Array("NAME:xAxis",
    "DirectionType:=", "AbsoluteDirection",
    "EdgeID:=", <Int>,
    "FaceID:=", <Int>,
    "xDirection:=", "<int>",
    "yDirection:=", "<Int>",
    "zDirection:=", "<Int>",
    "UParam:=", <int>,
    "VParam:=", <Int>),
  Array("NAME:yAxis",
    "DirectionType:=", "AbsoluteDirection",
    "EdgeID:=", <Int>,
    "FaceID:=", <int>,
    "xDirection:=", "<int>",
```

```
            "yDirection:=", "<int>",
            "zDirection:=", "<int>",
            "UParam:=", <int>,
            "VParam:=", <int>)),
        Array("NAME:Attributes",
            "Name:=", "<ObjectCSName>")
```

Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project53")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.SetWCS Array("NAME:SetWCS Parameter",
"Working Coordinate System:=", "ObjectCS1")
oEditor.EditObjectCS Array("NAME:ObjectCSParameters",
"PartID:=", 6,
"ReverseXAxis:=", false,
"ReverseYAxis:=", false,
Array("NAME:Origin", "IsAttachedToEntity:=", false,
"EntityID:=", -1,
"PositionType:=", "AbsolutePosition",
"UParam:=", 0, "VParam:=", 0,
"XPosition:=", "0mm", "YPosition:=", "0mm",
"ZPosition:=", "0mm"),
Array("NAME:xAxisPos", "IsAttachedToEntity:=", true,
"EntityID:=", 13,
"PositionType:=", "OnEdge",
"UParam:=", 0.75, "VParam:=", 0,
```

```
"XPosition:=", "0", "YPosition:=", "0", "ZPosition:=", "0"),
Array("NAME:yAxisPos", "IsAttachedToEntity:=", true,
"EntityID:=", 7,
"PositionType:=", "FaceCenter",
"UParam:=", 0, "VParam:=", 0,
"XPosition:=", "0", "YPosition:=", "0", "ZPosition:=", "0")),
Array("NAME:Attributes", "Name:=", "ObjectCS1")
```

EditRelativeCS

Use: Modifies a relative coordinate system. Use `<AttributesArray>` to indicate the name of the coordinate system to modify.

Command: **Modeler>Coordinate System>Edit**

Syntax:
```
EditRelativeCS <RelativeCSParametersArray>,
   <AttributesArray>
```

Return Value: None

Parameters:
```
<ParametersArray>
   Array("NAME:RelativeCSParameters",
      "OriginX:=", <value>,
      "OriginY:=", <value>,
      "OriginZ:=", <value>,
      "XAxisXvec:=", <value>,
      "XAxisYvec:=", <value>,
      "XAxisZvec:=", <value>,
      "YAxisXvec:=", <value>,
      "YAxisYvec:=", <value>,
      "YAxisZvec:=", <value>)
```

EditRelativeCS (2D Extractor)

Use: Modifies a relative coordinate system. Use `<AttributesArray>` to indicate the name of the coordinate system to modify.

Command: **Modeler>Coordinate System>Edit**

Syntax:
```
EditRelativeCS <RelativeCSParametersArray>,
   <AttributesArray>
```

Return Value: None

Parameters:
```
<ParametersArray>
```

```
Array("NAME:RelativeCSParameters",
   "OriginX:=", <value>,
   "OriginY:=", <value>,
   "XAxisXvec:=", <value>,
   "XAxisYvec:=", <value>,
   "YAxisXvec:=", <value>,
   "YAxisYvec:=", <value>)
```

Export

Use: Exports the model to a file.

Command: **Modeler>Export**

Syntax:
```
Export <ExportParametersArray>
```

Return Value: None

Parameters:
```
<ExportParametersArray>
   Array("NAME:ExportParameters",
      "File Name:=", <string>,
      "Major Version:=", <int>,
      "Minor Version:=", <int>)
```

`Major Version`

Can be –1 or any ACIS major version supported by HFSS software.

`Minor Version`

Can be –1 or any ACIS minor version supported by HFSS software.

ExportModelImageToFile

Use: Export an image of the model to a file.

Command: **Modeler>Export...**

Syntax:
```
ExportModelImageToFile "<path>/
<imageName>.<formatsuffix>" 0,0, Array(<SaveImageParams>)
```

Return Value: None

Parameters:
```
<path>
   <imagename>
```

<formatsuffix>, You can export the following graphics formats:

Extension	Contents
.bmp	Bitmap files.
.gif	Graphics Interchange Format files.
.jpeg	Joint Photographics Experts Group files.
.tiff	Tagged Image File Format files.
.wrl	Virtual Reality Modeling Language (VRML) files.

```
0,0,
Array("NAME:SaveImageParams",
"ShowAxis:=","<string>",
"ShowGrid:=", "<string>",
"ShowRuler:=", "<string>")
```

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 15.0.0
' 2:28:32 PM  Jul 30, 2012
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("dra_diel")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.ExportModelImageToFile  _
```

```
    "C:/MyPath/Downloads/dra_example.jpg", 0, 0,
  Array("NAME:SaveImageParams", "ShowAxis:=",  _
    "Default", "ShowGrid:=", "Default", "ShowRuler:=", "Default")
```

Fillet

Use: Creates a fillet.

Command: Modeler>Fillet

Syntax: `Fillet(<ObjectName> <FilletParameters>)`

Return Value: None

Parameters:

```
<ObjectName>
   Array("NAME:Selections", _
      "Selections:=", <string>),
<FilletParameters>
   Array("NAME:Parameters", _
   Array("NAME:FilletParameters", _
      "CoordinateSystemID:=", <value>,
      "Edges:=", <ArrayOfEdgeIDs>,
      "Radius:=", <value>,
      "Setback:=", <value>))
```

Example:

```
oEditor.Fillet Array("Name:Selections", "Selections:=", _
"Box1"), Array("NAME:Parameters", Array("NAME:FilletParameters", _
"CoordinateSystemID:=", -1, "Edges:=", Array(13), "Radius:=", _
"1mm", "Setback:=", "0mm"))
```

GenerateHistory

Use: Generates the history for specified 1D objects.

Command: **Modeler>Generate History**

Syntax: `GenerateHistory <SelectionsArray>`

Return Value: None

HealObject

Use: To heal an imported object.

Command: **Modeler>Model Healing>Heal**

Syntax: `HealObject <parameters>`

Return Value: None

Parameters:

```
Array("NAME:Selections",
   "Selections:=", "<objectID>",
   "NewPartsModelFlag:=", ["Model" | "NonModel"]),
   Array("NAME:ObjectHealingParameters",
   "Version:=", 1,
   "AutoHeal:=", <boolean>,
   "TolerantStitch:=", <boolean>,
   "SimplifyGeom:=", <boolean>,
   "TightenGaps:=", <boolean>,
   "StopAfterFirstStitchError:=", <boolean>,
   "MaxStitchTol:=", <smallvalue>,
   "ExplodeAndStitch:=", <boolean>,
   "GeomSimplificationTol:=", <value>,
   "MaximumGeneratedRadiusForSimplification:=", <value>,
   "SimplifyType:=", 2,
   "TightenGapsWidth:=", <value>,
   "RemoveSliverFaces:=", <boolean>,
   "RemoveSmallEdges:=", <boolean>,
   "RemoveSmallFaces:=", <boolean>,
   "SliverFaceTol:=", <value>,
   "SmallEdgeTol:=", <value>,
   "SmallFaceAreaTol:=", <value>,
   "BoundingBoxScaleFactor:=", <value>,
   "RemoveHoles:=", <boolean>,
   "RemoveChamfers:=", <boolean>,
   "RemoveBlends:=", <boolean>,
   "HoleRadiusTol:=", <value>,
   "ChamferWidthTol:=", <value>,
   "BlendRadiusTol:=", <value>,
   "AllowableSurfaceAreaChange:=", <value>,
   "AllowableVolumeChange:=", <value>)
```

Example:

```
oEditor.HealObject Array("NAME:Selections",
"Selections:=", "Box1",
"NewPartsModelFlag:=", "Model"),
Array("NAME:ObjectHealingParameters",
"Version:=", 1,
"AutoHeal:=", false,
"TolerantStitch:=", true,
"SimplifyGeom:=", true,
"TightenGaps:=", true,
"StopAfterFirstStitchError:=", false,
"MaxStitchTol:=", 0.001,
"ExplodeAndStitch:=", true,
"GeomSimplificationTol:=", -1,
"MaximumGeneratedRadiusForSimplification:=", -1,
"SimplifyType:=", 2,
"TightenGapsWidth:=", 1E-006,
"RemoveSliverFaces:=", true,
"RemoveSmallEdges:=", true,
"RemoveSmallFaces:=", true,
"SliverFaceTol:=", 0,
"SmallEdgeTol:=", 0.1,
"SmallFaceAreaTol:=", 0.1,
"BoundingBoxScaleFactor:=", 1250,
"RemoveHoles:=", true,
"RemoveChamfers:=", true,
"RemoveBlends:=", true,
"HoleRadiusTol:=", 0.1,
"ChamferWidthTol:=", 0.1,
"BlendRadiusTol:=", 0.1,
"AllowableSurfaceAreaChange:=", 5,
"AllowableVolumeChange:=", 5)
```

GetActiveCoordinateSystem

Use: Get active coordinate system.

Command: None.

Syntax: `GetActiveCoordinateSystem`

Return Value: Active coordinate name.

Parameters: `None.`

Example:

```
Dim csName
csName = oEditor.GetActiveCoordinateSystem
```

GetCoordinateSystems

Use: Get coordinate system names.

Command: None

Syntax: `GetCoordinateSystems`

Return Value: An array of coordinate system names

Parameters: `None`

Example:

```
Dim csNames
csNames = oEditor.GetCoordinateSystems ()
```

Import

Use: Imports a 3D model file.

Command: **Modeler>Import**

Syntax: `Import <ImportParametersArray>`

Return Value: None

Parameters:

```
<ImportParametersArray>
   Array("NAME:NativeBodyParameters",
   "CoordinateSystemID:=", <ID>,
   "HealOption:=", <integer>,
   "CheckModel:=", <boolean>,
   "Options:=", "",
   "FileType:=", "UnRecognized",
   "MaxStitchTol:=", <value>,
   "SourceFile:=", "string")
```

For Q3D Extractor the Import command details are as follows:

Use: Imports a 3D model file.

Command: **Modeler>Import**

Syntax: `Import <ImportParametersArray>`

Return Value: None

Parameters:

```
<ImportParametersArray>
   Array("NAME:NativeBodyParameters",
      "AutoHeal:=", <bool>,
      "Options:=",  <string>,
```

Example:

```
"SourceFile:=", <string>)
```

ImportDXF

Use: Imports an AutoCAD model file and a tech file. The tech file is an ASCII file that lists the units, the layer name followed by tab delimited color, elevations and thickness.

Command: **Modeler>Import**

Syntax: `ImportDXF <ImportParametersArray>`

Return Value: None

Parameters:

```
<ImportParametersArray>
   Array("NAME:options",
      "FileName:=", <path>,
      "Scale:=", <real>,
      "UnionOverlapping:=", <boolean>,
      "AutoDetectClosed:=", <boolean>,
      "SelfStitch:=", <boolean>,
      "DefeatureGeometry:=", <boolean>,
      "DefeatureDistance:=", <real>,
      "RoundCoordinates:=", <boolean>,
      "RoundNumDigits:=", <integer>,
      "WritePolyWithWidthAsFilledPoly:=", <boolean>,
      "ImportMethod:=", <integer>,
      "2DSheetBodies:=", <boolean>,
   Array("NAME:LayerInfo",
      Array("NAME:<layerName>",
      "source:=", "<integer>",
      "display_source:=", "<integer>",
      "import:=", <boolean>,
```

```
        "dest:=", "<integer>",
        "dest_selected:=", <boolean>,
        "layer_type:=", "<string>",
        "paint:=", <boolean>),
      Array("NAME:TechFileLayers", "layer:=",
        Array("name:=", "ground",
        "color:=", "purple",
        "elev:=", 0,
        "thick:=", 0.0001),
        )))
      oEditor.Import Array("NAME:NativeBodyParameters",
        "CoordinateSystemID:=", -1,
        "HealOption:=", <integer>,
        "CheckModel:=", <boolean>,
        "Options:=", "",
        "FileType:=", "UnRecognized",
        "MaxStitchTol:=", <real>,
        "SourceFile:=", "<path>" & ".sm3")
```

Example:

```
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.ImportDXF Array("NAME:options", "FileName:=",  _
  "C:/mymodel.dxf", ...
Array("NAME:TechFileLayers", ...
"layer:=", Array("name:=", "BOTTOMLAYER",
"color:=", "purple",
"elev:=", 0,
"thick:=", 200),
)))
oEditor.Import Array("NAME:NativeBodyParameters",
"CoordinateSystemID:=", -1,
"HealOption:=", 0,
"CheckModel:=", true,
"Options:=", "",
```

```
"FileType:=", "UnRecognized",
"MaxStitchTol:=", 0.001,
"SourceFile:=", "C:design." & ".sm3")
```

ImportGDSII [Modeler Import]

Use: Imports a GDSII 3D model file.

Command: **Modeler>Import**

Syntax: `ImportGDSII <ImportParametersArray>`

Return Value: None

Parameters:

```
"FileName:=", "<string>",
"NodeConversionType:=", "<string>",
"MaxLayerNumber:=", <integer>,
"ApproxPolyToCircle:=", <boolean>,
"FlattenHierarchy:=", <boolean>,
"ImportMethod:=", <integer>,
Array("NAME:LayerMap",
   Array("NAME:LayerMapInfo",
   "LayerNum:=", <integer>,
   "Import:=", <booleaninteger>,
   "IsVIA:=", <booleaninteger>,
   "IsLayerNew:=", <booleanInteger>,
   "DestLayer:=", "<string>",
   "layer_type:=", "<string>",
   "IsPresentInLayout:=", <boolean>),

   "OrderMap:=", Array("entry:=",
   Array("order:=", <integer>,
   "layer:=", "<string>"),
   "entry:=", Array("order:=",
   <integer>, "layer:=", "Signal25"),
   "entry:=", Array("order:=", <integer+1>,
   "layer:=", "Signal30"),
   ...),
Array("NAME:Structs",
```

```
Array("NAME:GDSIIStruct", "ImportStruct:=", <boolean>,
   "CreateNewCell:=", <boolean>,
   "StructName:=", "string>",
   Array("NAME:Elements")),
   Array("NAME:GDSIIStruct", "ImportStruct:=",<boolean>,
   "CreateNewCell:=", <boolean>,
   "StructName:=", "<string>",
   Array("NAME:Elements"))))

oEditor.Import Array("NAME:NativeBodyParameters",
   "CoordinateSystemID:=", <ID>,
   "HealOption:=", <integer>,
   "CheckModel:=", <boolean>,
   "Options:=", "",
   "FileType:=", "UnRecognized",
   "MaxStitchTol:=", <value>,
   "SourceFile:=", "string")
```

Example:

```
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.ImportGDSII Array("NAME:options", ...
```

Intersect

Use: Intersects specified objects.

Command: **Modeler>Boolean>Instersect**

Syntax: `Intersect <SelectionsArray>, <IntersectParametersArray>`

Return Value: None

Parameters:

```
<IntersectParametersArray>
   Array("NAME:IntersectParameters",
      "KeepOriginals:=",  <bool>)
```

MoveCStoEnd

Use: Moves the named ObjectCS to the end of the History tree.

Command: **Modeler>Coordinate System>Move CS to End**

Syntax: `MoveCSToEnd Array("NAME:Selections", "Selections:=", "<ObjectCSName>")`

Return Value: None

Parameters: `<ObjectCSName>`

Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("coax_bend")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.MoveCSToEnd Array("NAME:Selections",
"Selections:=", "ObjectCS1")
```

MoveFaces

Use: Moves the specified faces along normal or along a vector.

Command: **Modeler>Surface>Move Faces>Along Normal**

Modeler>Surface>Move Faces>Along Vector

Syntax: `MoveFaces <SelectionsArray>, <MoveFacesParametersArray>`

Return Value: None

Parameters:

```
<MoveFacesParametersArray>
   Array("NAME:Parameters",
       <FacesOfOneObjToMove>, <FacesOfOneObjToMove>, ...)

<FacesOfOneObjToMove>
   Array("Name:MoveFacesParameters",
      "MoveAlongNormalFlag:=",  <bool>,
      "OffsetDistance:=", <value>,
      "MoveVectorX:=", <value>,
```

```
    "MoveVectorY:=", <value>,
    "MoveVectorZ:=", <value>,
    "FacesToMove:=", <array>)
```

`MoveAlongNormalFlag`

Specifies whether to move along the face normal or along a vector.

If false, provide the `MoveVectorX`, `MoveVectorY`, and `MoveVectorZ` parameters.

`FacesToMove`

Array of integers – the IDs of the faces to move

Example:

```
oEditor.MoveFaces _
  Array("NAME:Selections", "Selections:=", _
    "Box2,Box1"), _
  Array("NAME:Parameters", _
    Array("NAME:MoveFacesParameters", _
      "MoveAlongNormalFlag:=", true, _
      "OffsetDistance:=", "1mm", _
      "FacesToMove:=", Array(218)),
    Array("NAME:MoveFacesParameters", _
      "MoveAlongNormalFlag:=", false,_
      "OffsetDistance:=", "1mm", _
      "MoveVectorX:=", "1mm", _
      "MoveVectorY:=", "0mm", _
      "MoveVectorZ:=", "0mm", _
      "FacesToMove:=", Array(185)))
```

MoveFaces (2D Extractor)

Use: Moves the specified faces along normal or along a vector.

Command: **Modeler>Surface>Move Faces>Along Normal**

Modeler>Surface>Move Faces>Along Vector

Syntax: `MoveFaces <SelectionsArray>, <MoveFacesParametersArray>`

Return Value: None

Parameters: `<MoveFacesParametersArray>`

```
  Array("NAME:Parameters",
      <FacesOfOneObjToMove>, <FacesOfOneObjToMove>, ...)
<FacesOfOneObjToMove>
  Array("Name:MoveFacesParameters",
    "MoveAlongNormalFlag:=",  <bool>,
    "OffsetDistance:=", <value>,
    "MoveVectorX:=", <value>, "MoveVectorY:=", <value>,
    "FacesToMove:=", <array>)
MoveAlongNormalFlag
  Specifies whether to move along the face normal or
  along a vector.
  If false, provide the MoveVectorX and MoveVectorYparam-
  eters.
FacesToMove
  Array of integers - the IDs of the faces to move
```

Example:

```
oEditor.MoveFaces _
  Array("NAME:Selections", "Selections:=", _
    "Box2,Box1"), _
  Array("NAME:Parameters", _
    Array("NAME:MoveFacesParameters", _
      "MoveAlongNormalFlag:=", true, _
      "OffsetDistance:=", "1mm", _
      "FacesToMove:=", Array(218)),
    Array("NAME:MoveFacesParameters", _
      "MoveAlongNormalFlag:=", false,_
      "OffsetDistance:=", "1mm", _
      "MoveVectorX:=", "1mm", _"MoveVectorY:=", "0mm",
      __"FacesToMove:=", Array(185)))
```

ProjectSheet

Use: Project planar sheet object, typically for modeling thin conformal deposits. Typically followed by **Thicken Sheet**.

Command: **Modeler>Surface>Project Sheet**

Syntax:

```
ProjectSheet
(Array("NAME:Selections",Selections:="...,...,""),
Array("NAME:ProjectSheetParameters")
```

Return Value: None

Parameters: None

Example:

```
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.ProjectSheet Array("NAME:Selections", "Selections:=",
"Box1,Box2,Polyline1"),
array("NAME:ProjectSheetParameters")
```

PurgeHistory

Use: Purges the construction history of the selected object. For complex objects this simplifies the object and can improve modeler speed.

Command: **Modeler>Purge History**

Syntax: `PurgeHistory <PurgeHistoryArray>`

Return Value: None

Parameters:

```
<PurgeHistoryArray>
   Array("Name:Selections",
   "Selections:=", <string>,
   "NewPartsModelFlag:=", ["Model" | "NonModel"])
```

`Selections`

Name of the object to purge.

`NewPartsModelFlag`

Flag to indicate model properties, Model or NonModel.

Example:

```
oEditor.PurgeHistory Array("NAME:Selections", _
"Selections:=", "Polygon1", "NewPartsModelFlag:=", "Model")
```

Section

Use: Creates a 2D cross-section of the selection in the specified plane.

Command: **Modeler>Surface>Section**

Syntax: `Section <SelectionsArray>, <SectionParametersArray>`

Return Value: None

Parameters:

```
<SectionParametersArray>
   Array("NAME:SectionToParameters",
      "SectionPlane:=", <string>)
```

```
Section Plane
```

Possible values are `"XY", "YZ", "ZX"`

Section (2D Extractor)

Use: Creates a 2D cross-section of the selection in the specified plane.

Command: **Modeler>Surface>Section**

Syntax:
```
Section <SelectionsArray>, <SectionParametersArray>
```

Return Value: None

Parameters:
```
<SectionParametersArray>
  Array("NAME:SectionToParameters",
    "SectionPlane:=", <string>)
Section Plane
      Possible values are "XY".
```

SeparateBody

Use: Separates bodies of specified multi-lump objects.

Command: **Modeler>Boolean>Separate Bodies**

Syntax:
```
SeparateBody <SelectionsArray>
```

Return Value: None

Parameters:
```
<SelectionsArray>
  Selections:=",<objectNames>"
  "NewPartsModelFlag:=", ["Model" | "NonModel"]
```

Indicates the type of model properties in the geometry - Model or Non Model.

Example:

```
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.SeparateBody Array("NAME:Selections",
"Selections:=", "Rectangle1,Circle1",
"NewPartsModelFlag:=", "Model")
```

SetModelUnits

Use: Sets the model units.

Command: **Modeler>Units**

Syntax:
```
SetModelUnits <ModelUnitsParametersArray>
```

Return Value: None

Parameters:

```
<ModelUnitsParametersArray>
   Array("NAME:Units Parameter",
      "Units:=", <string>,
      "Rescale:=", <bool>)
```

`Units`

Possible values are: `"cm", "ft", "in", "meter", "mil", "mm", "nm", "uin", "um"`

SetWCS

Use: Sets the working coordinate system.

Command: **Modeler>Coordinate System>Set Working CS**

Syntax: `SetWCS <WCSParametersArray>`

Return Value: None

Parameters:

```
<WCSParametersArray>
   Array("NAME:SetWCS Parameter",
      "Working Coordinate System:=", <string>)
```

`Working Coordinate System`

Name of the coordinate system to set as the WCS.

ShowWindow

Use: Opens the selected 3D model editor window.

Syntax: `ShowWindow`

Return Value: None

Parameters: None

Example:

```
Set oDesign = oProject.GetActiveDesign
Set oModeler = oDesign.SetActiveEditor("3D Modeler")
oEditor.ShowWindow
```

Split

Use: Splits specified objects along a plane.

Command: **Modeler->Boolean->Split**

Syntax: `Split <SelectionsArray>, <SplitParametersArray>`

Return Value: None

Parameters:

```
<SplitParametersArray>
   Array("NAME:SplitToParameters",
      "SplitPlane:=", <string>,
      "WhichSide:=", <string>)
```

`SplitPlane`

Possible values are `"XY", "YZ", "ZX"`

`WhichSide`

Side to keep. Possible values are `"Both"`, `"PositiveOnly"`, `"NegativeOnly"`

Split (2D Extractor)

Use: Splits specified objects along a plane.

Command: **Modeler>Boolean>Split**

Syntax:

```
Split <SelectionsArray>, <SplitParametersArray>
```

Return Value: None

Parameters:

```
<SplitParametersArray>
   Array("NAME:SplitToParameters",
      "SplitPlane:=", <string>,
      "WhichSide:=", <string>)
```

`SplitPlane`

Possible values are `"XY"`

`WhichSide`

Side to keep. Possible values are `"Both"`, `"PositiveOnly"`, `"NegativeOnly"`

Subtract

Use: Subtracts specified objects.

Command: **Modeler->Boolean->Subtract**

Syntax:

```
Subtract <SubtractSelectionsArray>,
   <SubtractParametersArray>
```

Return Value: None

Parameters:

```
<SubtractSelectionsArray>
   Array("NAME:Selections",
      "Blank Parts:=", <string>,
      "Tool Parts:=", <string>)
```

```
Blank Parts
```
Comma-separated list of parts to use as the blank in the subtract operation.
Example: `"Blank Parts:=", "Box1, Box2"`

```
Tool Parts
```
Comma-separated list of parts to use as the tool in the subtract operation.
Example: `"Blank Parts:=", "Box3, Box4"`

```
<SubtractParametersArray>
  Array("NAME:SubtractParameters",
    "KeepOriginals:=",  <bool>)
```

Example:

```
oEditor.Subtract _
  Array("NAME:Selections", _
    "Blank Parts:=", "Polygon1",_
    "Tool Parts:=",  "Box1"),_
  Array("NAME:SubtractParameters", _
    "KeepOriginals:=", false)
```

SweepFacesAlongNormal

Use: Sweep a face along normal.

Command: **Modeler>Sweep Faces Along Normal**

Syntax: `SweepFacesAlongNormal <selection> <parameters>`

Return Value: None

Parameters:

```
Array("NAME:Selections",
  "Selections:=", "<faceID>",
  "NewPartsModelFlag:=", ["Model" | "NonModel"]),
  Array("NAME:Parameters",
  Array("NAME:SweepFaceAlongNormalToParameters",
  "FacesToDetach:=", Array( <faceID>),
  "LengthOfSweep:=", "<value><units>")
  )
```

Example:

```
oEditor.SweepFacesAlongNormal
```

```
Array("NAME:Selections",
"Selections:=", "Rectangle1",
"NewPartsModelFlag:=", "Model"),
Array("NAME:Parameters",
Array("NAME:SweepFaceAlongNormalToParameters",
"FacesToDetach:=",
Array( 57),
"LengthOfSweep:=", "0.5mm")
)
```

ThickenSheet

Use: Thicken a sheet object to convert it to a 3D object.

Command: **Modeler>Surface>Thicken Sheet**

Syntax:
```
Thicken Sheet <SelectionParameters>
<SheetThickenParameters>
```

Return Value: None

Parameters:
```
Array("NAME:Selections",
   "Selections:=", "<objectID>",
   "NewPartsModelFlag:=", ["Model" | "NonModel"]),
   Array("NAME:SheetThickenParameters",
   "Thickness:=", "<value><units>",
   "BothSides:=", <boolean>)
```

Example:
```
oEditor.ThickenSheet Array("NAME:Selections",
"Selections:=", "Rectangle1",
"NewPartsModelFlag:=", "Model"),
Array("NAME:SheetThickenParameters",
"Thickness:=", "1.01mm",
"BothSides:=", false)
```

UncoverFaces

Use: Uncovers specified faces.

Command: **Modeler>Surface>Uncover Faces**

Syntax:
```
UncoverFaces <SelectionsArray>, <UncoverParametersArray>
```

Return Value: None

Parameters:
```
<UncoverParametersArray>
   Array("NAME:Parameters",
```

```
      <FacesOfOneObjToUncover>,
        <FacesOfOneObjToUncover>,...)

<FacesOfOneObjToUncover>
   Array("Name:UncoverFacesParameters",
      "FacesToUncover:=", <array>)
```

FacesToUncover

An array of integers – the face IDs of the faces to uncover.

Example:

```
oEditor.UncoverFaces _
   Array("NAME:Selections", "Selections:=", _
      "Box3,Box2"),_
   Array("NAME:Parameters", _
      Array("NAME:UncoverFacesParameters", _
         "FacesToUncover:=", Array(69)),
      Array("NAME:UncoverFacesParameters", _
         "FacesToUncover:=", Array(36)))
```

Unite

Use: Unites the specified objects.

Command: **Modeler>Boolean>Unite**

Syntax: `Unite <SelectionsArray>, <UniteParametersArray>`

Return Value: None

Parameters:

```
<UniteParametersArray>
   Array("NAME:UniteParameters",
        "KeepOriginals:=", <bool>)
```

WrapSheet

Use: Wraps a sheet object to another object.

Command: Wrap Sheet

Syntax:

```
WrapSheet Array("NAME:Selections", "Selections:=",
"<SheetID>,<ObjectID>"),
Array("NAME:WrapSheetParameters", "Imprinted:=",
<Boolean>)
```

Return Value: None

Parameters:

`<SheetID>`

ID of sheet object.

`<ObjectID>`

ID of 3D object.

`<Boolean>`

true or false.

Example:

```
' ----------------------------------------------
' Script Recorded by ANSYS Electronics Desktop Version 2015.0.0
' 13:58:45  Nov 13, 2014
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project3")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.WrapSheet Array("NAME:Selections", "Selections:=", "Rectan-
gle1,Box1"), Array("NAME:WrapSheetParameters", "Imprinted:=",  _
  true)
```

Other oEditor Commands

ChangeProperty

Delete

GetBodyNamesByPosition

GetEdgeByPosition

GetEdgeByID (Q3D Extractor)

GetEdgeByPosition(2DExtractor)

GetEdgeIDsFromObject
GetEdgeIDsFromFace
GetFaceArea
GetFaceByPosition
GetFaceByPosition(2DExtractor)
GetFaceCenter
GetFaceIDs
GetModelBoundingBox
GetObjectIDByName
GetObjectName
GetObjectNameByFaceID
GetObjectsByMaterial
GetObjectsInGroup
GetMatchedObjectName
GetModelUnits
GetNumObjects
GetSelections
GetVertexIDsFromEdge
GetVertexIDsFromFace
GetVertexIDsFromObject
GetVertexPosition
GetUserPosition
PageSetup
RenamePart

ChangeProperty

Use: Change the properties used to create an object in the history tree

Command: Select a command in the History Tree and select Properties.

Syntax: `ChangeProperty <array>`

Return Value: None

Parameters: `Varies, depending on the properties associated with the select object command.`

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 14.0.0
' 9:49:26 AM  Mar 09, 2011
```

```
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project58")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.ChangeProperty Array("NAME:AllTabs", Array("NAME:Geometry3DC-
mdTab",
Array("NAME:PropServers", "Polyline2:CreatePolyline:1"),
Array("NAME:ChangedProps",
Array("NAME:Type", "Value:=", "Isosceles Trapezoid"),
Array("NAME:Width/Diameter", "Value:=", "0.1mm"),
Array("NAME:Top Width", "Value:=", "0.05mm"),
Array("NAME:Height", "Value:=", "0.02mm"))))
oEditor.ChangeProperty Array("NAME:AllTabs", Array("NAME:Geometry3DC-
mdTab", Array("NAME:PropServers", "Polyline2:CreatePolyline:1"),
Array("NAME:ChangedProps",
Array("NAME:Bend Type", "Value:=", "Corner"))))
```

Delete

Use: Deletes specified objects, coordinate systems, points, planes, etc.

Command: None

Syntax: `Delete <SelectionsArray>`

Return Value: None

GetBodyNamesByPosition

Use: Gets the IDs of objects that contact the given point.

Command: None.

Syntax: `GetBodyNamesByPosition(<positionParameters>).`

Return Value: Retuurns a list of IDs.

Parameters:

```
<PositionParameters>
   Array("NAME:Parameters", _
      "Xposition:=", <value>,
      "YPosition:=", <value>,
      "ZPosition:=", <value>)
```

Example:

```
bodyNames = oEditor.GetBodyNamesByPosition(Array("NAME:Parameters", _
      "XPosition:=", "a", _
      "YPosition:=", "0mm", _
      "ZPosition:=", "0mm"))
```

GetEdgeByPosition

Use: Gets the edge IDcorresponding to position input.

Syntax: `GetEdgeByPosition(<PositionParameters>)`

Return Value: Returns an integer edge ID.

Parameters:

```
<PositionParameters>
   Array("NAME:EdgeParameters", _
      "BodyName:=", <string>,
      "Xposition:=", <value>,
      "YPosition:=", <value>,
      "ZPosition:=", <value>)
```

Example:

```
edgeid = oEditor.GetEdgeByPosition(Array("NAME:EdgeParameters", _
"BodyName:=", "Box1", "XPosition:=", "3.4mm", _
"YPosition:=", "2.8mm", "ZPosition:=", "0.4mm"))
```

GetEdgeByPosition (2D Extractor)

Use: Gets the edge id corresponding to position input.

Syntax: `GetEdgeByPosition(<PositionParameters>)`

Return Value: Returns an integer edge id.

Parameters:

```
<PositionParameters>
   Array("NAME:EdgeParameters", _
      "BodyName:=", <string>,
      "Xposition:=", <value>, "YPosition:=", <value>)
```

```
edgeid = oEditor.GetEdgeByPosition(Array("NAME:EdgeParameters", _ "Body-
Name:=", "Box1", "XPosition:=", "3.4mm", _ "YPosition:=", "2.8mm")
```

GetEdgeByID (Q3D Extractor)

Use: Gets the edge by ID.

Command: None

Syntax: `GetEdgeByID(<ID>)`

Return Value: None

Parameters: `<ID>`

Type: string

GetEdgeIDsFromFace

Use: Get the edge IDs of given a face ID.

Command: None

Syntax: `GetEdgeIDsFromFace <Face ID>`

Return Value: An array of edge IDs

Parameters: <FaceID>

ID of the face

Example:

```
Dim oEdgeIDs
oEdgeIDs = Array()
oEdgeIDs = oEditor.GetEdgeIDsFromFace(10)
```

GetEdgeIDsFromObject

Use: Get the edge IDs of given an object name

Command: None

Syntax: `GetEdgeIDsFromObject <Object Name>`

Return Value: An array of edge IDs

Parameters: <ObjectName>

Name of the object

Example: `Example:`

```
Dim oEdgeIDs
oEdgeIDs = Array()
oEdgeIDs = oEditor.GetEdgeIDsFromObject("Box1")
```

GetFaceArea

Use: Get area of given face ID.

Command: None.

Syntax: `GetFaceArea <face ID>`

Return Value: Double value for face area.

Parameters: `<FaceID>`

Face ID

Example:

```
Dim oArea
oArea = oEditor.GetFaceArea(10)
```

GetFaceCenter

Use: Given a face ID, return the center position

Command: none

Syntax: `GetFaceCenter <FaceID>`

Return Value: An array containing face center position

Parameters: `<FaceID>`

Example:

```
Dim oFaceCenter
oFaceCenter = oEditor.GetFaceCenter(oFaceID)
```

GetFaceByPosition

Use: Gets the face id corresponding to position input.

Syntax: `GetFaceByPosition(<FaceByPositionParametersArray>)`

Return Value: Returns an integer face id

Parameters: `(<FaceByPositionParametersArray>)`

Array("NAME:FaceParameters",

"BodyName:=", <string>,

"XPosition:=", <value>,

"YPosition:=", <value>,

"ZPosition:=", <value>)

BodyName

Name of the body on which the point lies.

The point should be on exactly one face, not at a vertex or edge where two or more faces join. And the coordinates must be on the face.

Example:

```
Dim faceid
faceid = oEditor.GetFaceByPosition(Array("NAME:FaceParameters", _
"BodyName:=" "Box1", "XPosition:=", "3.4mm", "YPosition:=, _
"2.8mm", "ZPosition:=", "0.4mm"))
```

GetFaceByPosition (2D Extractor)

Use: Gets the ID of a face by position.

Command: None

Syntax: `GetFaceByPosition <FaceByPositionParametersArray>`

Return Value: An integer containing the face ID.

Parameters:

```
<FaceByPositionParametersArray>
   Array("NAME:FaceParameters",
      "BodyName:=", <string>,
      "XPosition:=", <value>,
      "YPosition:=", <value>)
BodyName
   Name of the body on which the point lies.
```

Example:

```
Dim oFaceID
FaceID = oEditor.GetFaceByPosition_
   (Array("NAME:Parameters",_
      "BodyName:=", "Box1",_
"XPosition:=", "0mm",_ "YPosition:=", "0mm"))
```

GetFaceIDs

Use: Get the face IDs of given an object name

Command: None.

Syntax: `GetFaceIDs <Object Name>`

Return Value: An array of face IDs

Parameters:

`<ObjectName>`

Name of the object

Example:

```
Dim oFaceIDs
oFaceIDs = Array()
oFaceIDs = oEditor.GetFaceIDs("Box1")
```

GetModelBoundingBox

Use: Gets the bounding box of the current model.

Syntax: GetModelBoundingBox()

Return Value: Returns the Xmin, Ymin, Zmin, Xmax, Ymax, Zmax values that define the bounding box.

Parameters: None

Example:

```
Dim oBoundingBox
oBoundingBox = oEditor.GetModelBoundingBox()
```

GetObjectIDByName

Use: Get Object IDs to provide for CreateEntityList.

Syntax: GetObjectIDByName("<objectName>")

Return Value: <Object ID>

Parameters: <objectName>

Type: <string>

Example:

```
oObjectID = oEditor.GetObjectIDByName("Box2")
```

GetObjectName

Use: Gets an object name corresponding to the 0 base index of the creation order.

Syntax: GetObjectName(<Index>)

Return Value: Returns the object name of the corresponding object.

Parameters: <Index>

Type: <string>

The 0 base index of the creation order

Example:

```
objectname = oEditor.GetObjectName(3)
```

GetObjectNameByFaceID

Use: Gets an object name corresponding to the input face id.

Syntax: GetObjectName(<FaceID>)

Return Value: Returns the name of the corresponding object.

Parameters: <FaceID>

Type: <string>

Example:

```
objectname = oEditor.GetObjectNameByFaceID(Face10)
```

GetObjectsByMaterial

Use: Get objects by material name.

Command: None.

Syntax: `GetObjectsByMaterial <Material Name>`

Return Value: An array of object names.

Parameters: `<Material Name>`

Type: <string>

Material name/

Example:

```
Dim objNames
objNames = oEditor.GetObjectsByMaterial("vacuum")
```

GetObjectsInGroup

Use: Returns the objects for the specified group.

Syntax: `GetObjectsInGroup(<GroupName>)`

Return Value: The objects in the group.

Parameters: `<groupName>`

Type: <string>

One of <materialName>, <assignmentName>, "Non Model", "Solids", "Unclassified", "Sheets", "Lines"

Example:

```
Set oEditor = oDesign.SetActiveEditor("3D Modeler")

Dim oObject
Dim oObjects
Set oObjects = oEditor.GetObjectsInGroup( "Sheets" )
For Each oObject in oObjects
            MsgBox oObject
Next
```

GetMatchedObjectName

Use: Gets all object names containing the input text string.

Syntax: `GetMatchedObjectName(<ObjectNameWildcardText>)`

Return Value: Array of object names containing wildcard text.

Parameters: `<ObjectNameWildcardText>`

Type: <string>

Text to be used for object name matching.

Example:

```
objectnames = oEditor.GetMatchedObjectName("Box*")
```

GetModelUnits

Use: Get the model units.

Command: None.

Syntax: `GetModelUnits`

Return Value: A string contains current model units.

Parameters: `None.`

Example:

```
Dim oUnit
oUnit = oEditor.GetModelUnits()
```

GetNumObjects

Use: Gets the number of objects in a design.

Syntax: `GetNumObjects`

Return Value: Returns the number of objects.

Type: <int>

Parameters: `None`

Example:

```
totalobjects = oEditor.GetNumObjects
```

GetSelections [Model Editor]

Use: Informational.

Syntax: `GetSelections`

Return Value: Array of IDs.

Parameters: `None`

Example:

```
Set oProject = oDesktop.SetActiveProject("Project6")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
Dim A
A = Array()
A = oEditor.GetSelections
Dim B
B = Join(A,",")
'Debug.Write "The Selections are " &B
MsgBox(B)
Dim C
C = Array("NAME:Selections", "Selections:=", B)
oEditor.Delete C
```

GetUserPosition

Use: Returns the coordinates of an interactive position input in the 3D model window.

Syntax: `GetUserPosition(<PositionInputPrompt>)`

Return Value: Array of coordinates

Parameters: `<PositionInputPrompt>`

Type: <string>

Example:

```
Dim position
Dim coord
position = oEditor.GetUserPosition("Enter a point")
For Each coord in position
   Msgbox(coord)
Next
```

GetVertexIDsFromEdge

Use: Get the vertex IDs of given a edge ID.

Command: None.

Syntax: `GetVertexIDsFromEdge <Edge ID>`

Return Value: An array of edge IDs.

Parameters: `<EdgeID>`

ID of the edge.

Example:

```
Dim oVertexIDs
oVertexIDs = Array()
oVertexIDs = oEditor.GetVertexIDsFromEdge(10)
```

GetVertexIDsFromFace

Use: Get the vertex IDs of given a face ID.

Command: None.

Syntax: `GetVertexIDsFromFace <Face ID>`

Return Value: An array of vertex IDs

Parameters: `<FaceID>`

ID of the face

Example:

```
Dim oVertexIDs
oVertexIDs = Array()
oVertexIDs = oEditor.GetVertexIDsFromFace(10)
```

GetVertexIDsFromObject

Use: Get the vertex IDs of given an object name

Command: None.

Syntax: `GetVertexIDsFromObject <Object Name>`

Return Value: An array of edge IDs

Parameters: `<ObjectName>`

Name of the object

Example:

```
Dim oVertexIDs
oVertexIDs = Array()
oVertexIDs = oEditor.GetVertexIDsFromObject("Box1")
```

GetVertexPosition

Use: Returns a vector of vertex coordinates.

Syntax: `GetVertexPosition(<VertexID>)`

Return Value: Vector of coordinates

Parameters: `<VertexID>`

Type: <vector>

Example:

```
position = oEditor.GetVertexPosition(VertexIDs(0))
```

PageSetup

Use: Specifies the page settings for printing.

Command: **File>Page Setup**

Syntax: `PageSetup <PageSetupParametersArray>`

Return Value: None

Parameters:

```
<PageSetupParametersArray>
   Array("NAME:PageSetupData",
      "margins:=",
        Array("left:=", <value>,
        "right:=", <value>,
        "top:=", <value>,
        "bottom:=", <value>))
```

RenamePart

Use: Renames an object.

Command: None

Syntax: `RenamePart <RenameParametersArray>`

Return Value: None

Parameters:

```
<RenameParametersArray>
   Array("NAME:Rename Data",
      "Old Name:=", <string>,
      "New Name:=", <string>)
```

12 Output Variable Script Commands

The Output variable commands should be executed by the "OutputVariable" module. First obtain the output variable module from oDesign and use it for output variable commands.

```
Set oModule = oDesign.GetModule("OutputVariable")
oModule.CommandName <args>
```

The old output variable commands are still supported but they are deprecated and produce a warning in the message window. The old Output variable commands were executed by the `oModule` object.

```
Set oDesign = oProject.GetDesign ("HfssDesign1")
Set oModule = oDesign.GetModule("OutputVariable")
```

For Q3D Extractor the old output variable commands were executed as follows.

```
Set oDesign = oProject.GetDesign ("Q3DDesign1" or "2DExtractorDesign1")
Set oModule = oDesign.GetModule("OutputVariable")
```

List of commands are as follows:

CreateOutputVariable

Different forms of this command are documented for HFSS, Q3D Extractor and Schematic and Layout Editors.

Command details for HFSS are as follows:

Use: Add a new output variable to the output variable list. Output variables are associated with a name and an expression. The name of an output variable is not permitted to collide with design variables or Sim values or with other output variable names. It cannot have spaces or any arithmetic or other operators in it. The definitions cannot be cyclic. For example, A = 2*B, B=3*A is not allowed.

Command: **HFSS>Results>Output Variable**

Syntax:
```
CreateOutputVariable <OutputVarName>, <Expression>,
<Solution Name>, <reportTypeName>, <ContextArray>
```

Return Value: None.

Parameters:

`<OutputVarName>`

Type: <string>

Name of the output variable

`<Expression>`

Type: <value>

Value to assign to the variable

`<SolutionName>`

Type: <string>

The name of the solution as seen in the output variable UI.

`<ReportTypeName >`

Type: <string>

The name of the report type as seen in the output variable UI.

`<ContextArray>`

Type: <variant>

Context for which the output variable expression is being evaluated.

Example:

```
Set oModule = oDesign.GetModule("OutputVariable")
oModule.CreateOutputVariable "test","mag(S(WavePort1,WavePort1))", _
    "Setup1 : LastAdaptive", "Modal Solution Data", _
    Array("Domain:=", "Sweep")
```

For Layout Editor the CreateOutputVariable syntax and other details are as follows:

Syntax: `CreateOutputVariable "<varName>", <Expression>, <SolutionName>, <SolutionType>, <Domain>`

Return Value: None

Parameters:

`<VarName>`

Type: <string>

Name of the output variable.

`<Expression>`

Type: <value>

Value to assign to the variable.

`<SolutionName>`

Type: <string>

Name of the solution as listed in the output variable UI.

For example: `"Setup1 : Last Adaptive"`

`<ReportTypeName>`

Type: <string>

The name of the report type as seen in the output variable UI.

`<SimValueContext>`

Type: <variant>

Context for which the output variable expression is being evaluated. For more information see SimValueContext.

Example:

```
Set oModule = oDesign.GetModule("OutputVariable")
    oModule.CreateOutputVariable "test","mag(S(WavePort1,WavePort1))",
    _
      "Setup1 : LastAdaptive ", "Modal Solution Data", Array("Domain:=", _
    "Sweep")

    oModule.CreateOutputVariable "Var_" & OutputQuantity,_
    OutputQuantity, Solution, "Far Fields", _
Array("Context:=", InfiniteSphere, "Domain:=", "Sweep")
```

For Q3D Extractor or a 2D Extractor the command and example are as follows:

Command: **Q3D Extractor** or **2D Extractor>Results>Output Variables**

Example:

```
Set oModule = oDesign.GetModule("OutputVariable")
oModule.CreateOutputVariable "char_Z", _
"sqrt(ACL(via:via_source,via:via_source)/
C(via,GroundPlane))", _ "Setup1 : LastAdaptive",
"Matrix", Array("Context:=", "Original")
```

DeleteOutputVariable

Use: Deletes an existing output variable. The variable can only be deleted if it is not in use by any traces.

Command: **HFSS>Output Variables**, dialog **Delete Button**

Syntax: `DeleteOutputVariable <OutputVarName>`

Return Value: None

Parameters: `<OutputVarName>`

Type: <string>

Name of the output variable.

Example:

```
Set oModule = oDesign.GetModule("OutputVariable")
oModule.DeleteOutputVariable “efield_online”
```

For Q3D Extractor, the DeleteOutputVariable command details are as follows.

Use: Deletes an existing output variable. The variable can only be deleted if it is not in use by any traces.

Command: **Delete** command in the **Output Variables** dialog box. Click **Q3D Extractor** or **2D Extractor>Results>Output Variables** to open the **Output Variables** dialog box.

Syntax: `DeleteOutputVariable <VarName>`

Return Value: None

Parameters: `<VarName>`

Type: <string>

Name of the output variable.

Example:

```
Set oModule = oDesign.GetModule("OutputVariable")
oModule.DeleteOutputVariable "Var_" & OutputQuantity
```

DoesOutputVariableExist

Use: Verify that a named output variable exists.

Command: None.

Syntax: `DoesOutputVariableExist(<outputVariableName>)`

Return Value: True if the variable exists. False otherwise.

Parameters: `<outputVariableName>`

Type: <string>

Text string of the putput variable name

Example:

```
oProject = oDesktop.GetActiveProject()
oDesign = oProject.GetActiveDesign()
oModule = oDesign.GetModule("OutputVariable")
oModule.DoesOutputVariableExist("MyTestVar")
```

EditOutputVariable

Use: Changes the name or expression of an existing output variable.

Syntax: `EditOutputVariable <OrigVarName>, <NewExpression>, <NewVarName>, <SolutionName>, <reportTypeName>, <ContextArray>`

Provide empty quotes "" as the NewVarName or NewExpression if it should not be changed.

Return Value: None

Parameters: `<OrigVarName>`

Type: <string>

Name of the original output variable.

`<NewExpression>`

Type: <string>

New value to assign to the variable.

`<NewVarName>`

Type: <string>

New name of the variable if any, else pass empty string.

`<SolutionName>`

Type: <string>

Name of the solution as seen in the output variable UI.

For example: `"Setup1 : Last Adaptive"`

`<ReportTypeName>`

Type: <string>

The name of the report type as seen in the output variable UI.

<ContextArray>

Type: <variant>

Context for which the output variable expression is being evaluated

Array("Context:=", <Context>)

Example:

```
Set oModule = oDesign.GetModule("OutputVariable")
oModule.EditOutputVariable "test", "dB(S(WavePort1,WavePort1)) ", _
"testNew", "Setup1 : LastAdaptive", "Modal Solution Data", _
Array("Domain:=", "Sweep")
```

For Q3D Extractor the EditOutputVariable command details are as follows.

Use: Changes the name or expression of an existing output variable.

Syntax:
```
EditOutputVariable <OrigVarName> <NewExpression>
<NewVarName> <SoultionName> <SolutionType> <ContextArray>
```

Provide empty quotes `""` as the `NewVarName` or `NewExpression` if it should not be changed.

Return Value: None

Parameters:

`<OrigVarName>`

Type: <string>

Original name of the variable.

`<NewExpression>`

Type: <value>

New value to assign to the variable.

`<NewVarName>`

Type: <string>

New name of the variable if any, or else pass an empty string.

`<SolutionName>`

Type: <string>

Name of the solution as seen in the output variable UI.

For example: `"Setup1 : Last Adaptive"`

`<SolutionType>`

Type: <string>

The name of the report type as seen in the output variable UI.

`<ContextArray>`

Type: <variant>

Context for which the output variable expression is being evaluated.

Example:
```
oModule.EditOutputVariable "char_Z", _
"sqrt(ACL(via:via_source,via:via_source)/C(via,via))",
"char_Z", _ "Setup1 : LastAdaptive", "Matrix",
Array("Context:=", "Original")
```

GetOutputVariableValue

Different forms of this command are documented for HFSS, Q3D Extractor as well Schematic and Layout Editors.

For HFSS, the command details are as follows:

Use: Gets the double value of an output variable. Only those expressions that return a double value are supported. The expression is evaluated only for a single point.

Syntax: `GetOutputVariableValue(<OutputVarName>, <IntrinsicVariation>, <SolutionName>, <ReportTypeName>, <ContextArray>)`

Return Value: Double value of the output variable.

Parameters:

`<OutputVarName>`

Type: <string>

Name of the output variable.

`<IntrinsicVariation>`

Type: <string>

A set of intrinsic variable value pairs to use when evaluating the output expression.

Example: "Freq='20GHz' Theta='20deg' Phi='30deg' in HFSS

"" in Q3D Extractor

`<SolutionName>`

Type: <string>

Name of the solution as listed in the output variable UI.

For example: `"Setup1 : Last Adaptive"`

`<ReportTypeName>`

Type: <string>

The name of the report type as seen in the output variable UI. Possible HFSS values are:

`"Modal Solution Data"` - Only for Driven Modal solution-type problems with ports.

`"Terminal S Parameters"` - Only for Driven Terminal solution-type problems with ports.

`"Eigenmode Parameters"` - Only for Eigenmode solution-type problems.

`"Fields"`

`"Far Fields"` - Only for problems with radiation or PML boundaries.

`"Near Fields"` - Only for problems with radiation or PML boundaries.

“Emission Test”

`<ContextArray>`

Type: Array

Context for which the output variable expression is being evaluated. This can be empty if there is no context (for example, for S- parameters). The Reporter uses interpolation to evaluate the values at the given special sweeps. If a requested special-

sweep value is outside the range the Reporter does not do extrapolation. Rather it issues an error message indicating values outside the available range for a primary sweep.

Example:

```
Array("Context:=","Infinite Sphere1")
or Array("Context:=","Polyline1")
or Array()
```

Example:

```
' ----------------------------------------------
' Script Recorded by ANSYS Electronics Desktop Version 2016.0.0
' MSG box displays value of the output variable
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Dim var1
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project1")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("OutputVariable")
var1 = oModule.GetOutputVariableValue("p", "freq='1GHz'", _
        "Setup1 : LastAdap-tive",Array())
msgbox (var1)
```

Example:

```
' ----------------------------------------------------------
' Sample script to get output variable values in 2.0 products
' ----------------------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim projects
```

```
Dim oProject
Dim oDesign
Dim oModule
Dim val
' ----------------------------------------------------------
' Get all of the VBS objects needed to talk to the product
' ----------------------------------------------------------
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
Set projects = oDesktop.GetProjects()
Set oProject = projects(0)
Set oDesign = oProject.GetDesign ("HfssDesign1")
Set oModule = oDesign.GetModule("OutputVariable")
' ----------------------------------------------------------
' fieldOV calculated at a point so we don't need distance
' ----------------------------------------------------------
val = oModule.GetOutputVariableValue ( "fieldOV", "Freq='1GHz'",_
    "Setup1 : LastAdaptive", "Fields", Array("Context:=", "Point1"))
' ----------------------------------------------------------
' SValue11 is a Hfss matrix parameter defined as
' S(WavePort1,WavePort1)
' it needs no context
' ----------------------------------------------------------
val = oModule.GetOutputVariableValue ( "SValue11", _
    "Setup1 : LastAdaptive", _
    "Freq='1GHz'",_
    "Modal Solution Data", _
Array())

' ----------------------------------------------------------
' Now, look at the original output variable in a different design
' variation
' ----------------------------------------------------------
val = oModule.GetOutputVariableValue ( "fieldOV", _
    "Distance='0' _
```

```
    "Setup1 : LastAdaptive", "Fields", _
    Freq='1GHz' xsize='0.4mm' ysize='4.1mm'", _
Array("Context:=", "Polyline1", "PointCount:=", 1 ) )
' ------------------------------------------------------------
' Look at the same variable at a position 1mm along the line
' ------------------------------------------------------------
val = oModule.GetOutputVariableValue ( "fieldOV", _
    "Distance='1mm'
    "Setup1 : LastAdaptive", _
    Freq='1GHz'", _
    "Fields", _
    Array("Context:=", "Polyline1", "PointCount:=", 3 ) _
        )
MsgBox( "2 val " & FormatNumber(val) )
```

For HFSS Layout and Schematic Editor the command details are as follows:

Syntax: `GetOutputVariableValue(<OutputVarName>, <VariationKey>, <SolutionName>, <ReportType>, <ContextArray>)`

Return Value: Double value of the output variable.

Parameters:

`<OutputVarName>`
Type: <string>
Name of the output variable.

`<VariationKey>`
Type: <string>
Example: " 'F='20GHz' x_size='1.0in' "

`<SolutionName>`
Type: <string>
Name of the solution as listed in the output variable UI.
For example: `"Setup1 : Last Adaptive"`

`<ReportType>`
Type: <string>

Possible values are:

"Standard" - For most plot types.

"Load Pull" - For load pull plots.

"Constellation" - For constellation plots.

"Data table" - For data tables.

"Eye Diagram" - For eye diagrams.

"Statistical" - For statistical plots.

<ContextArray>

Type: Array

Context for the output variable. For more information see SimValueContext.

Example:

```
' ----------------------------------------------
' Script used by Ansys Designer Version 4.1.0
----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsysDesigner.Design-
erScript")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("MyTransient-
Project")
Set oDesign = oProject.SetActiveDesign("Nexxim1")
Set oModule = oDesign.GetModule("OutputVariable")
val1=oModule.GetOutputVariableValue("vout", "MyTransient-
Setup", "time='3.9462ns'", "Standard", _
Array("NAME:Context", "SimValueContext:=", Array(1, 0, 2,
0, false, false, -1, 1, 0, 1, 1, "", 0, 0)))
MsgBox val1
```

Example:

```
' ----------------------------------------------
' Script used by Ansys Designer Version 4.1.0
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsysDesigner.Design-
erScript")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("MyLNAExample")
Set oDesign = oProject.SetActiveDesign("Nexxim1")
Set oModule = oDesign.GetModule("OutputVariable")
val1=oModule.GetOutputVariableValue("magS", _
   "LinearFrequency", "F='1GHz'", "Standard", _
   Array("NAME:Context", "SimValueContext:=", _
   Array(3, 0, 2, 0, false, false, -1, 1, 0, 1, 1, "", 0,
   0)))
MsgBox val1
```

For Q3D Extractor, the GetOutputVariableValue command details are as follows:

Use: Gets the double value of an output variable. Only those expressions that return a double value are supported. The expression is evaluated only for a single point.

Syntax: `GetOutputVariableValue(<OutputVarName>, <VariationKey>, <SolutionName>, <ReportType>, <ContextArray>)`

Return Value: Double value of the output variable.

Parameters: `<OutputVarName>`

Type: <string>

Name of the output variable.

<VariationKey>

Type: <string>

Example: “Freq=’20GHz’ x_size=’1.0in’"

<SolutionName>

Type: <string>

Name of the solution as listed in the output variable UI.

For example: "Setup1 : Last Adaptive"

<ReportType>

Type: <string>

Possible values are:

"Matrix"

"C Fields Report"

"DC R/L Fields Report"

"AC R/L Fields Report"

<ContextArray>

Type: Array

Context for which the output variable expression is being evaluated. This can be empty if there is no context.

Example:

```
Array("Context:=","Original")
or Array("Context:=","Polyline1")
or Array()
```

Example:

```
Set oModule = oDesign.GetModule("OutputVariable")
Dim Val
Val=oModule.GetOutputVariableValue("OutVarTest",_
Freq=’500MHz’ "Setup1 : LastAdaptive", "Matrix", _
Array ("Context:= “,Original" ))
```

SimValueContext

SimValueContext holds context information for a trace, and describes how data for a trace should be extracted from the simulation. SimValueContext contains a list of 14 required initial values:

```
SimValueContext (
Domain ID, Calculation Type, Number of Cycles, Rise Time,
```

```
Step, Impulse, Context ID, Window Width,
Window Type, TDR Kaiser Parameter, Hold Time, DeviceName,
TDR Step Time, DR Maximum Time )
```

For example, the following indicates a trace in the Time Domain, Standard Calculation with the number of cycles being 2:

```
"SimValueContext:=", Array(1, 0, 2, 0, false, false, -1, 1, 0, 1, 1,
"", 0, 0)
```

Additional, context-specific values may follow the required values, as described in subsection 15 below.

1. Domain ID

No Domain	0
Time Domain	1
Spectrum Domain	2
Sweep Domain	3
Device Domain	4
SinglePt Domain	5
LoadPull Domain	6
Transient Domain	7
Budget Domain	8
NetworkFunction Domain	9
Oscillator Domain	55802
Noise Domain	55803
Transfer Function Domain	55807
Time Frequency Domain	55808
Transient Time Domain	55809
Periodic AC Domain	55818
UI Domain	55819

Eye Measurement Domain	55823
Initial Response Domain	55824
Peak Distortion Domain	55825

2. Calculation Type

Standard Calculation	0
Device2_DCIV	1
Device3_DCIV_Output	2
Device3_DCIV_Input	3
Device3_DCIV_Transfer	4
Device3_DCIV_Reverse	5
Device2_ACLoad	6
Device3_ACLoad_Output	7
Device3_ACLoad_Input	8
Device3_ACLoad_Transfer	9
Device3_ACLoad_Reverse	10
Constellation	11
EyeDiagram	12
FreeX (Statistic Report)	13

3. **Number of Cycles** — Used in Time Domain in HarmonicBalance analysis.
4. **Rise Time** — Not used by Designer/Nexxim.

5. **Step** — Not used by Designer/Nexxim.
6. **Impulse** — Not used by Designer/Nexxim.
7. **Context ID** — Not used by Designer/Nexxim.
8. **Window Width** — Not used by Designer/Nexxim.
9. **Window Type** — Not used by Designer/Nexxim.
10. **TDR Kaiser Parameter** — Not used by Designer/Nexxim.
11. **Hold Time** — Not used by Designer/Nexxim.
12. **DeviceName** — Not used by Designer/Nexxim.
13. **TDR Step Time** — Not used by Designer/Nexxim.
14. **TDR Maximum Time** — Not used by Designer/Nexxim.
15. **Context-specific values** — Used in Time Domain in HarmonicBalance analysis.

Context-specific values are entered in the format "key, true/false, keyvalue", where:
— **"key"** is the name of the key being set.
— **"true/false"** indicates whether the key is a string value.
— **"keyvalue"** is the value of the key.
— The order of the context keys is not significant.
— Context keys have software defaults that will be used if not provided in the script.

Example:

```
"SimValueContext:=", Array(1, 0, 2, 0, false, false, -1,
1, 0, 1, 1, "", 0, 0,
"DE", false, "0",
"DP", false, "20000000",
"DT", false, "0.001",
"WE", false, "10ns",
"WM", false, "10ns",
"WN", false, "0ns",
"WS", false, "0ns"))
```

a. Plotting Range for Time domain in Transient and QuickEye analysis:

Description	Key Name	Is a string?	Key Value
Start Time	WS	False	0ns
Stop Time	WE	False	10ns
Minimum Time	WM	False	0ns

Maximum Time	WN	False	10ns
Is Thinning Enabled?	DE	False	0
Dy/dx Tolerance	DT	False	0.001
Number of points	DP	False	20000000

b.Transient report context for Spectral domain in Transient analysis:

Description	Key Name	Is a string?	Key Value
Start Time	TS	False	0ns
Stop Time	TE	False	10ns
Max Harmonics	MH	False	100
Window type	WT	False	0
Width Percentage	WW	False	100
Kaiser Parameter	KP	False	0
Adjust Coherent Gain	CG	False	0

Window Type	ID
Rectangular	0
Bartlett	1
Blackman	2
Hamming	3
Hanning	4
Kaiser	5
Welch	6
Weber	7
Lanzcos	8

c. Eyeprobe index context for UI domain, Time domain, Eye Measuremant domain in Verif-Eye and QuickEye analysis:

Description	Key Name	Is a string?	Key Value
Eyeprobe compinst ID	PCID	False	0

d. Eyesource index context for Initial Response domain and Peak Distortion domain in VerifEye and QuickEye analysis:

Description	Key Name	Is a string?	Key Value
Eyesource compinst ID	SCID	False	0

e. UI domain context in VerifEye and QuickEye analysis:

Description	Key Name	Is a string?	Key Value
Use midpoint?	MIDPOINT	False	0 - Don't use midpoint. 1 - Use midpoint of amplitude. 2 - Use midpoint of UI.
Minimum latch overdrive	MLO	False	0

f. Distribution Context for UI Domain in VerifEye and QuickEye analysis:

Description	Key Name	Is a string?	Key Value
Use distribution?	USE_DIST	False	0 - No 1 - Yes
Distribution type	DIST	False	0 - Receiver Jitter 1 - Receiver Noise 2 - User Defined

Receiver Jitter Parameters

Description	Key Name	Is a string?	Key Value
DLL standard deviation	DSD	False	0
Distribution type	DIST	False	0
DLL taps	DMN	False	0
Static Offset	SOFF	False	0
Number of Gaussian data sets	NUMG	False	0
Gaussian std deviation	GS0,GS1...	False	0
Offset mean	GM1,GM1...	False	0
Number of Uniform data sets	NUMU	False	0
Uniform width	UW0,UW1...	False	0
Uniform mean	UM1,UM1...	False	0

Receiver Noise Parameters

Description	Key Name	Is a string?	Key Value
Number of Gaussian data sets	NUMG	False	0
Gaussian std deviation	GS0,GS1...	False	0
Number of Uniform data sets	NUMU	False	0
Uniform width	UW0,UW1...	False	0

User Defined Parameters

Description	Key Name	Is a string?	Key Value
Number of XY data pairs	NUMG	False	0
X data	X0,X1,X2.. .	False	0
Y data	Y0,Y1,Y2.. .	False	0
Cutoff probability	CP	False	0

13 Reporter Editor Script Commands

Reporter commands should be executed by the `oDesign` object. One example of accessing this object is:

```
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("ReportSetup")
```

Another example of accessing this object on Q3D Extractor is:

```
Set oDesign = Project.SetActiveDesign("Q3DDesign1")
Set oModule = oDesign.GetModule("ReportSetup")
```

All Report and Trace properties can be edited using the **ChangeProperty** commands. This includes Title properties, General properties, and Background properties such as border color, fonts, X and Y axis scaling, and number display.

Note: HFSS version 11 and above supports Reporter scripting. When you execute **Tools>Record Script**, HFSS Operations performed in the Reporter are automatically recorded.

AddCartesianLimitLine
AddCartesianXMarker
AddQuickEyeAnalysis
AddDeltaMarker
AddMarker
AddNote
AddTraces
AddVerifEyeAnalysis
ClearAllMarkers
ClearAllMarkers(2DExtractor)
CopyTracesData
CopyReportData
CopyReportDefinitions
CopyTraceDefinitions
CreateReport
CreateReport(2D Extractor)
CreateReportFromTemplate

AddCartesianLimitLine

Use: Adds a limit line to a report on the X axis.

Command: **Report2D>Add Limit Line**

Syntax:

```
AddCartesianLimitLine <ReportName>,
Array("NAME:CartesianLimitLine",
Array("NAME:XValues",<XValues>), "XUnits:=", "<XUnits>",
Array("NAME:YValues", <YValues>), "YUnits:=", "<YUnits>",
"YAxis:=", "<YAxisName>")
```

Return Value: None

Parameters:

`<ReportName>`

Type: <string>

Name of Report.

`<XValues>`

Type: <real>

Array of X coordinate values.

`<XUnits>`

Type: <string>

Unit of measure for X values.

`<YValues>`

Type: <real>

Array of Y coordinate values.

`<YUnits>`

Type: <string>

Unit of measure for Y values.

`<YAxisName>`

Type: <string>

Name of the Y axis associated with the limit line.

Example:

```
oModule = oDesign.GetModule("ReportSetup")
oModule.AddCartesianLimitLine "Project Outputs",
Array("NAME:CartesianLimitLine", Array("NAME:XValues", 0, 2, 5, 7,
10, 15), "XUnits:=", "s", Array("NAME:YValues", 0.05, 0.3, 0.65,
0.825, 0.95, 1), "YUnits:=", "mV", "YAxis:=", "Y1")
```

AddCartesianXMarker

Use: Adds a marker to a report on the X axis.

Command: **Report2D>Marker>Add X Marker**

Syntax: `AddCartesianXMarker <ReportName>, <MarkerID>, <Xcoord>`

Return Value: None

Parameters:

`<ReportName>`

Type: <string>

Name of Report.

`<MarkerID>`

Type: <string>

ID of the marker, for example: “M1”.

<XCoord>

Type: <real>

X location for the marker.

Example:

```
oModule.AddCartesianXMarker "XY Plot1", "MX1", 0
```

AddDeltaMarker

Use: Add markers to calculate differences between two trace points on a plot.

Command: **Report2D>Marker>Add Delta Marker**

Syntax: `AddDeltaMarker <ReportName>, <MarkerID_1>, <TraceID_1>, <Xcoord_1>, <MarkerID_2>, <TraceID_2> <Xcoord_2>`

Return Value: None

Parameters:

<ReportName>

Type: <string>

Name of Report.

<MarkerID>

Type: <string>

ID for the markers.

<TraceID>

Type: <string>

Typically given by expression plus solution name plus coordinate system type.

<XCoord>

Type: <real>

X location for the marker.

Example:

```
oModule.AddDeltaMarker "XY Plot 1",
"m3", "dB(S(LumpPort1 LumpPort1)) : Setup1 : Sweep1 : Cartesian", _
"3.22GHz", _
"m4", "dB(S(LumpPort1 LumpPort1)) : Setup1 : Sweep1 : Cartesian",_
"3.93GHz"
```

Another example that is related to Q3D Extractor is as follows:

Example:

```
Set oProject = oDesktop.SetActiveProject("dra_antenna")
Set oDesign = oProject.SetActiveDesign("Q3DDesign1")
Set oModule = oDesign.GetModule("ReportSetup")
```

```
oModule.AddDeltaMarker "XY Plot 1",
"m3", "ACR(via:source1 via:source1) : Setup1 : Sweep1 : Cartesian", _
"3.22GHz", _
"m4", "ACR(via:source1 via:source1) : Setup1 : Sweep1 : Cartesian",_
"3.93GHz"
```

AddMarker

Use: Adds a marker to a trace on a report.

Command: **Report2D>Marker>Add Marker**

Syntax: `AddMarker <ReportName>, <MarkerID>, <TraceID>, <Xcoord>,`

Return Value: None

Parameters:

`<ReportName>`

Type: <string>

Name of Report.

`<MarkerID>`

Type: <string>

ID for the marker.

`<TraceID>`

Type: <string>

Typically given by expression plus solution name plus coordinate system type.

`<XCoord>`

Type: <real>

X location for the marker.

Example:

```
Set oModule = oDesign.GetModule("ReportSetup")
oModule.AddMarker "XY Plot1", "m1",  _
  "mag(S(Port1 Port1)) : Setup1 : LastAdaptive : Cartesian", "0.3in"
```

AddNote

Use: Adds a note at a specified location to a given report.

Command: Right-click on the plot and select **Add Note**

Syntax: `AddNote <ReportName> <NoteDataArray>)`

Return Value: None

Parameters:

`<ReportName>`

Type: <string>

Name of report.

<NoteDataArray>

Type: Array

Array("NAME:<NoteDataName>", <NoteArray>)

<NoteDataName>

Type: String

<NoteArray>

Array("NAME:<NoteDataSourceName>",
"SourceName:=", <SourceName>,
"HaveDefaultPos:=", <boolean>,
"DefaultXPos:=", <XPos>,
"DefaultYPos:=", <YPos>,
"String:=", <Note>))

Example:

```
Set oModule = oDesign.GetModule("ReportSetup")
oModule.AddNote "XY Plot1", Array("NAME:NoteDataSource",
Array("NAME:NoteDataSource", "SourceName:=", "Note1",
"HaveDefaultPos:=", true, "DefaultXPos:=", 1996, "DefaultYPos:=", _
3177, "String:=", "This is a note"))
```

AddTraces

Use: Creates a new trace and adds it to the specified report.

Command: **Modify Report>Add Trace**

Syntax: Add Traces <ReportName> <SolutionName> <ContextArray> <FamiliesArray> <ReportDataArray>

Return Value: None

Parameters:

<ReportName>

Type: <string>

Name of Report.

<SolutionName>

Type: <string>

Name of the solution as listed in the **Modify Report** dialog box.

For example: "Setup1 : Last Adaptive"

<ContextArray>

Type: Array of strings

Context for which the expression is being evaluated. This can be an empty string if there is no context.

Array("Domain:=", <DomainType>)

<DomainType>

ex. "Sweep" or "Time"

Array("Context:=", <GeometryType>)

<GeometryType>

ex. "Infinite Sphere*n*", "Sphere*n*", "Polyline*n*"

<FamiliesArray>

Type: Array of strings

Contains sweep definitions for the report.

Array("<VariableName>:= ", <ValueArray>)

<ValueArray>

Array("All") or Array("Value1", "Value2", ..."Value*n*")

examples of <VariableName>

"Freq", "Theta", "Distance"

<ReportDataArray>

Type: Array of strings

This array contains the report quantity and X, Y, and (Z) axis definitions.

Array("X Component:=", <VariableName>, "Y Component:=", <VariableName> | <ReportQuantityArray>)

<ReportQuantityArray>

ex. Array("dB(S(Port1, Port1))")

Example:

```
oModule.AddTraces "XY Plot1", "Setup1 : Sweep1", _
Array("Domain:=", "Time", "HoldTime:=", 1, "RiseTime:=", 0, _
"StepTime:=", 6.24999373748E-012, "Step:=", false, _
"WindowWidth:=", 1, _
```

```
"WindowType:=", 0, "KaiserParameter:=", 1, _
"MaximumTime:=", 6.2437437437444E-009), _
Array("Time:=", Array("All"), "OverridingValues:=", Array("0s", _
"6.24999373748188e-012s", ... )),
Array("X Component:=", "Time", _
"Y Component:=", Array("TDRZ(WavePort1)")), _
Array()
```

An example for Q3D Extractor is as follows:

Example:

```
oModule.AddTraces "XY Plot 1", "Setup1 : AdaptivePass",
Array("Context:=",  _ "Original"), Array("Pass:=",
Array("All"), "Freq:=", Array("0.5GHz"), "viarad:=",
Array( _ "Nominal"), "padrad:=", Array("Nominal")),
Array("X Component:=", "Pass", "Y Component:=", Array(
_"DCL(via:via_source,via:via_source)")), Array()
```

For Designer, the AddTraces command has the following details.

Use: Creates a new trace and adds it to the specified report.

Command: Modify Report>Add Trace

Syntax: `Add Traces <ReportName> <SolutionName> <ContextArray <FamiliesArray> <ReportDataArray>`

Return Value: None

Parameters:

`<ReportName>`

Type: <string>

Name of Report.

`<SolutionName>`

Type: <string>

Name of the solution as listed in the **Modify Report** dialog box.

For example: `"Setup1 : Last Adaptive"`

`<ContextArray>`

Type: Array of strings

Context for which the expression is being evaluated. This can be an empty string if there is no context.

Array("Domain:=", <DomainType>)

<DomainType>
ex. “Sweep” or “Time”

Array(“Context:=”, <SimValueContext>)
Context for the trace. For more information see SimValueContext.

<FamiliesArray>
Type: Array of strings
Contains sweep definitions for the report.
Array(“<VariableName>:= “, <ValueArray>)

<ValueArray>
Array(“All”) or Array(“Value1”, “Value2”, …”Value*n*”)
examples of <VariableName>
“Freq”, “Theta”, “Distance”

<ReportDataArray>
Type: Array of strings
This array contains the report quantity and X, Y, and (Z) axis definitions.
Array(“X Component:=”, <VariableName>, “Y Component:=”, <Variable-Name> | <ReportQuantityArray>)

<ReportQuantityArray>
ex. Array(“dB(S(Port1, Port1))”)

Example:

```
oModule.AddTraces "XY Plot1", "Setup1 : Sweep1", _
Array("Domain:=", "Time", "HoldTime:=", 1, "RiseTime:=",
0, _
"StepTime:=", 6.24999373748E-012, "Step:=", false, _
"WindowWidth:=", 1, _
"WindowType:=", 0, "KaiserParameter:=", 1, _
"MaximumTime:=", 6.2437437437444E-009), _
Array("Time:=", Array("All"), "OverridingValues:=", _
Array("0s","6.24999373748188e-012s", ... )), _
Array("X Component:=", "Time", _
"Y Component:=", Array("TDRZ(WavePort1)")), _
```

```
Array()
```

AddQuickEyeAnalysis

Use: Adds a QuickEye analysis to the design.

Command: Context menu of Analysis icon in project tree -> AddSolution Setup... -> QuickEye Analysis

Syntax:

```
AddQuickEyeAnalysis (Array("NAME:SimSetupName",
"DataBlockID:=", <int>,
"SimSetupID:=", <int>,
"OptionName:=", <string>,
"AdditionalOptions:=", <string>,
"AlterBlockName:=", <string>,
"FilterText:=",  <string>,
"AnalysisEnabled:=", <int>,    // 1 if enabled, 0 if dis-
abled
Array("NAME:OutputQuantities",  <QuantityArray>,  <Quan-
tityArray>...)
Array("NAME:NoiseOutputQuantities", <QuantityArray>,
<QuantityArray>...) ),
"Name:=",  <string>,     // name for new analysis
"QuickEyeAnalysis:=", Array(<string>,       // Input rise
time
<string>,         // Input low voltage
<string>,       // Input high voltage
<string>,       // bits per second
<string>,       // number of FFE taps
<string>,       // Random Jitter Standard Deviation
<string>,       // Delay
<string>,       // Duty cycle distortion
<bool>,       // true if unit interval, false if bits per
second
<string>,       // number of DFE taps
<bool>,        // true to calculate FFE, false if weights
are specified
<bool>,        // true to calculate DFE, false if weights
are specified
```

```
<string>,         // DFE decision threshold
<string>,        // DFE decision high
<string>,        // DFE decision low
<bool>),        // true if using specified equalization,
false if disabled
Array("NAME:SweepDefinition",
"Variable:=", <string>,
"Data:=", <string>,   // sweep
"OffsetF1:=", <bool>,
"Synchronize:=",   <int>),  // 1 to Synchronize, 0 other-
wise
"FFEWts:=", Array(".5", "-2"),     // optional, specified
weights
"DFEWts:=", Array("2")))             // optional, speci-
fied weights
```

Return Value: <string> – // Name of the analysis that was added

// If the name requested conflicts with the name of an existing
// analysis, the requested name is altered to be unique.
// The name returned reflects any change made to be unique.

Parameters: Parameters: <QuantityArray>:

Array("NAME:Quantity",
"NodeType:=", <string>, // CompInst, Variable, Net, Harmonics, or Custom
"CompID:=", <string>,
"CompName:=", <string>,
"QuantityName:=", <string>,
"Selected:=", <bool>,
"UnitType:=", <string>,
"DataType:=", <string>, // Real, Complex, Integer, Enum, Char, Free, Array, Record
"CircuitInstanceID:=", <string)

Example:

```
dim name
name = oModule.AddQuickEyeAnalysis (Array("NAME:Sim-
Setup", "DataBlockID:=", 28, "SimSetupID:=",  _
3, "OptionName:=", "(Default Options)", "AdditionalOp-
tions:=", "", "AlterBlockName:=",  _
```

```
"", "FilterText:=", "", "AnalysisEnabled:=", 1,
Array("NAME:OutputQuantities", Array("NAME:Quantity",
"NodeType:=", "CompInst", "CompID:=",  _
"5", "CompName:=", "Level01_NPN_Model_5", "Quanti-
tyName:=", "I", "Selected:=",  _
true, "UnitType:=", "NoUnit", "DataType:=", "Real", "Cir-
cuitInstanceID:=", ""), Array("NAME:Quantity", "Node-
Type:=",  _
"CompInst", "CompID:=", "10", "CompName:=", "RES__10",
"QuantityName:=", "I", "Selected:=",  _
true, "UnitType:=", "NoUnit", "DataType:=", "Real", "Cir-
cuitInstanceID:=", ""), Array("NAME:Quantity", "Node-
Type:=",  _
"Net", "CompID:=", "", "CompName:=", "net_47", "Quanti-
tyName:=", "V", "Selected:=",  _
true, "UnitType:=", "NoUnit", "DataType:=", "Real", "Cir-
cuitInstanceID:=", "")), Array("NAME:NoiseOutputQuanti-
ties"), "Name:=",  _
"MyQuickEyeAnalysis", "QuickEyeAnalysis:=", Array("5e-
10", "0", "1", "1e-9", "2",  _
"0", "1e-9", "0", true, "1", false, false, "0", "1", "-
1", true), Array("NAME:SweepDefinition", "Variable:=",  _
"Temp", "Data:=", "LINC 1cel 10cel 10", "OffsetF1:=",
false, "Synchronize:=",  _
0), "FFEWts:=", Array(".5", "-2"), "DFEWts:=",
Array("2")))
```

AddVerifEyeAnalysis

Use: Adds a VerifEye analysis to the design.

Command: Context menu of Analysis icon in project tree -> AddSolution Setup... -> VerifEye (Statistical Eye) Analysis

Syntax: AddVerifEyeAnalysis (Array("NAME:SimSetupName",

"DataBlockID:=", <int>,

"SimSetupID:=", <int>,

"OptionName:=", <string>,

"AdditionalOptions:=", <string>,

"AlterBlockName:=", <string>,

"FilterText:=", <string>,

"AnalysisEnabled:=", <int>, // 1 if enabled, 0 if disabled
Array("NAME:OutputQuantities", <QuantityArray>, <QuantityArray>...)
Array("NAME:NoiseOutputQuantities", <QuantityArray>, <QuantityArray>...)
),
"Name:=", <string>, // name for new analysis
"VerifEyeAnalysis:=", Array(<string>, // Input rise time
<string>, // Input low voltage
<string>, // Input high voltage
<string>, // bits per second
<string>, // number of FFE taps
<string>, // Random Jitter Standard Deviation
<string>, // Delay
<string>, // Duty cycle distortion
<bool>, // true if unit interval, false if bits per second
<string>, // number of DFE taps
<bool>, // true to calculate FFE, false if weights are specified
<bool>, // true to calculate DFE, false if weights are specified
<string>, // DFE decision threshold
<string>, // DFE decision high
<string>, // DFE decision low
<bool>), // true if using specified equalization, false if disabled
Array("NAME:SweepDefinition",
"Variable:=", <string>,
"Data:=", <string>, // sweep
"OffsetF1:=", <bool>,
"Synchronize:=", <int>), // 1 to Synchronize, 0 otherwise
"FFEWts:=", Array(".5", "-2"), // optional, specified weights
"DFEWts:=", Array("2"))) // optional, specified weights

Return Value: <string> – // Name of the analysis that was added

// If the name requested conflicts with the name of an existing
// analysis, the requested name is altered to be unique.
// The name returned reflects any change made to be unique.

Parameters: <QuantityArray>:

Array("NAME:Quantity",
"NodeType:=", <string>, // CompInst, Variable, Net, Harmonics, or Custom

"CompID:=", <string>,

"CompName:=", <string>,

"QuantityName:=", <string>,

"Selected:=", <bool>,

"UnitType:=", <string>,

"DataType:=", <string>, // Real, Complex, Integer, Enum, Char, Free, Array, Record

"CircuitInstanceID:=", <string)

Example:

```
dim name
name = oModule.AddVerifEyeAnalysis (Array("NAME:Sim-
Setup", "DataBlockID:=", 27, "SimSetupID:=", _
1, "OptionName:=", "Options", "AdditionalOptions:=", "",
"AlterBlockName:=", "", "FilterText:=", _
"", "AnalysisEnabled:=", 1, Array("NAME:OutputQuanti-
ties", Array("NAME:Quantity", "NodeType:=", "CompInst",
"CompID:=", _
"5", "CompName:=", "Level01_NPN_Model_5", "Quanti-
tyName:=", "I", "Selected:=", _
true, "UnitType:=", "NoUnit", "DataType:=", "Real", "Cir-
cuitInstanceID:=", ""), Array("NAME:Quantity", "Node-
Type:=", _
"CompInst", "CompID:=", "10", "CompName:=", "RES__10",
"QuantityName:=", "I", "Selected:=", _
true, "UnitType:=", "NoUnit", "DataType:=", "Real", "Cir-
cuitInstanceID:=", ""), Array("NAME:Quantity", "Node-
Type:=", _
"Net", "CompID:=", "", "CompName:=", "net_47", "Quanti-
tyName:=", "V", "Selected:=", _
true, "UnitType:=", "NoUnit", "DataType:=", "Real", "Cir-
cuitInstanceID:=", "")), Array("NAME:NoiseOutputQuanti-
ties"), "Name:=", _
"MyVerifEyeAnalysis", "VerifEyeAnalysis:=", Array("5e-
10", "0", "1", "1e-9", "2", _
"0", "1e-9", "0", true, "1", false, false, "0", "1", "-
1", true), Array("NAME:SweepDefinition", "Variable:=", _
"Temp", "Data:=", "LINC 1cel 10cel 10", "OffsetF1:=",
false, "Synchronize:=", _
```

```
0), "FFEWts:=", Array(".5", "-2"), "DFEWts:=",
Array("2")))
```

ClearAllMarkers

Use: Clears all markers from a report.

Command: Report2d>Markers>ClearAllMarkers

Syntax: `ClearAllMarkers "<ReportName>"`

Return Value: None

Parameters: `<ReportName>`

Type: <string>

Name of Report.

Example:

```
Set oProject = oDesktop.SetActiveProject("dra_antenna")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("ReportSetup")
oModule.ClearAllMarkers "XY Plot 1"
```

For Q3D Extractor the ClearAllMarkers command has the following details.

Use: Clears all markers.

Command: Q3DExtractor>Fields>Fields>Marker>Clear All

Syntax: `ClearAllMarkers`

Return Value: none

Parameters: `none`

Example:

```
Dim oAnsoftApp
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftQ3D.Q3DScriptInter-
face")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Solenoid")
```

```
Set oDesign = oProject.SetActiveDesign("Solenoid")
Set oModule = oDesign.GetModule("FieldsReporter")
oModule.ClearAllMarkers
```

ClearAllMarkers (2D Extractor)

Use: Clears all markers from a report.

Command: Report2d>Markers>ClearAllMarkers

Syntax: `ClearAllMarkers "<ReportName>"`

Return Value: None

Parameters: `<ReportName>`

Type: <string>

Name of Report.

Example:

```
Set oProject = oDesktop.SetActiveProject("dra_antenna")
Set oDesign = oProject.SetActiveDesign("Q3DDesign1")
Set oModule = oDesign.GetModule("ReportSetup")
oModule.ClearAllMarkers "XY Plot 1"
```

CopyTracesData

Use: Copy trace data for a paste operation.

Command: Select a trace in the Project tree, right-click and select **Copy Data**

Syntax: `CopyTracesData <ReportName> <TracesArray>)`

Return Value: None

Parameters: `<ReportName>`

Type: <string>

Name of Report.

`<TracesArray>`

Type: Array of Strings

Trace definitions from which to copy corresponding data.

Example:

```
oModule.CopyTracesData "Transmission", Array("mag(S(Port1,Port2))")
```

An example related to Q3D Extractor is as follows:

Example:

```
oModule.CopyTracesData "C11", Array("C (Box1,Box1))")
```

CopyReportData

Use: Copy all data corresponding to the specified reports.

Command: Select a report in the Project tree, right-click and select **Copy Data**

Syntax: `CopyReportData <ReportsArray>`

Return Value: None

Parameters: `<ReportsArray>`

Type: Array of strings

Names of reports from which to copy data.

Example:

```
oModule.CopyReportData Array("Transmission", "Reflection")
```

Example:

```
oModule.CopyReportData Array("L11", "Zo")
```

CopyReportDefinitions

Use: Copy the definition of a report for paste operations.

Command: Select a report in the Project tree, right-click and select **Copy Definition**

Syntax: `CopyReportDefinitions <ReportsArray>`

Return Value: None

Parameters: `<ReportsArray>`

Type: Array of strings

Names of reports from which to copy the definitions.

Example:

```
oModule.CopyReportDefinitions Array("Transmission", "Reflection")
```

CopyTraceDefinitions

Use: Copy trace definitions for a paste operation.

Command: Select a trace in the Project tree, right-click and select **Copy Definition**

Syntax: `CopyTraceDefinitions <ReportName> <TracesArray>`

Return Value: None

Parameters: `<ReportName>`

Type: <string>

Name of Report.

`<TracesArray>`

Type: Array of strings.

Trace definitions to copy.

Example:

```
oModule.CopyTraceDefinitions "Transmission",
Array("mag(S(Port1,Port2))")
```

The following example is applicable for Q3D Extractor.

Example:

```
oModule.CopyTraceDefinitions "R11",
Array("R(via:source1, via:source1)")
```

CreateReport

Use: Creates a new report with a single trace and adds it to the **Results** branch in the project tree.

Command: **HFSS>Results>Create <type> Report**

Syntax:

```
CreateReport <ReportName> <ReportType> <DisplayType>
<SolutionName> <ContextArray> <FamiliesArray>
<ReportDataArray>
```

Return Value: None

Parameters:

`<ReportName>`

Type: <string>

Name of Report.

`<ReportType>`

Type: <string>

Possible values are:

`"Modal S Parameters"` - Only for Driven Modal solution-type problems with ports.

`"Terminal S Parameters"` - Only for Driven Terminal solution-type problems with ports.

`"Eigenmode Parameters"` - Only for Eigenmode solution-type problems.

`"Fields"`

`"Far Fields"` - Only for problems with radiation or PML boundaries.

`"Near Fields"` - Only for problems with radiation or PML boundaries.

“Emission Test”

`<DisplayType>`

Type: <string>

If `ReportType` is `"Modal S Parameters"`, `"Terminal S Parameters"`, or `"Eigenmode Parameters"`, then set to one of the following:

`"Rectangular Plot"`, `"Polar Plot"`, `"Radiation Pattern"`, `"Smith Chart"`, `"Data Table"`, `"3D Rectangular Plot"`, or `"3D Polar Plot"`.

If <ReportType> is "Fields", then set to one of the following:

"Rectangular Plot", "Polar Plot", "Radiation Pattern", "Data Table", or "3D Rectangular Plot".

If <ReportType> is "Far Fields" or "Near Fields", then set to one of the following:

"Rectangular Plot", "Radiation Pattern", "Data Table", "3D Rectangular Plot", or "3D Polar Plot"

If <ReportType> is "Emission Test", then set to one of the following:

"Rectangular Plot" or "Data Table"

<SolutionName>

Type: <string>

Name of the solution as listed in the **Modify Report** dialog box.

For example: "Setup1 : Last Adaptive"

<Context*Array*>

Type: Array of strings

Context for which the expression is being evaluated. This can be an empty string if there is no context.

Array("Domain:=", <DomainType>)

<DomainType>

ex. "Sweep" or "Time"

Array("Context:=", <GeometryType>)

<GeometryType>

ex. "Infinite Sphere*n*", "Sphere*n*", "Polyline*n*"

<FamiliesArray>

Type: Array of strings

Contains sweep definitions for the report.

Array("<VariableName>:= ", <ValueArray>)

<ValueArray>

Array("All") or Array("Value1", "Value2", ..."Value*n*")

examples of <VariableName>

"Freq", "Theta", "Distance"

<ReportDataArray>

Type: Array of strings

This array contains the report quantity and X, Y, and (Z) axis definitions.

Array("X Component:=", <VariableName>, "Y Component:=", <VariableName> | <ReportQuantityArray>)

<ReportQuantityArray>

ex. Array("dB(S(Port1, Port1))")

Example:

```
oModule.CreateReport "Rept2DRectFreq",_
   "Modal Solution Data", "XY Plot", _
        "Setup1 : Sweep1",_
        Array("Domain:=", "Sweep"), _
        Array("Freq:=", Array("All")), _
        Array("X Component:=", "Freq",
        "Y Component:=", _ Array("dB(S(LumpPort1,Lump-
        Port1))")), _
        Array()
```

Example:

```
Set oModule = oDesign.GetModule("ReportSetup")
oModule.CreateReport "3D Cartesian Plot1", "Far Fields",_
"3D Cartesian Plot", "Setup1 : LastAdaptive", _
Array("Context:=", "Infinite Sphere1", "Domain:=", "Sweep"),
Array("Theta:=", Array("All"), "Phi:=", Array("All"), _
"Freq:=", Array("10GHz")), _
Array("X Component:=", "Theta", _
"Y Component:=", "Phi", _
"Z Component:=", Array("rETotal")), _
Array()
```

Example:

```
oModule.CreateReport "ReptSmithFreq",_
"Modal Solution Data", "Smith Plot", "Setup1 : Sweep1", _
Array("Domain:=", "Sweep"), _
Array("Freq:=", Array("All")),_
Array("Polar Component:=", _
Array("ln(Y(LumpPort1,LumpPort1))")), _
Array()
```

For Designer the CreateReport command details are as follows.

CreateReport [Designer]

Use: Creates a new report with a single trace and adds it to the **Results** branch in the project tree.

Command: Product Menu>Results>Create <type> Report

Syntax: `CreateReport <ReportName> <ReportType> <DisplayType> <SolutionName> <ContextArray> <FamilesArray> <ReportDataArray>`

Return Value: None

Parameters:

`<ReportName>`

Type: <string>

Name of Report.

<ReportType>

Type: <string>

Possible values are:

`"Standard"` - For most plot types.

`"Load Pull"` - For load pull plots.

`"Constellation"` - For constellation plots.

`"Data table"` - For data tables.

`"Eye Diagram"` - For eye diagrams.

`"Statistical"` - For statistical plots.

`<DisplayType>`

Type: <string>

Possible values are:

"Rectangular Plot", "Polar Plot", "Radiation Pattern", "Smith Chart", "Data Table", "3D Rectangular Plot", "3D Polar Plot", or “Rectangular

Stacked Plot".

`<SolutionName>`

Type: <string>

Name of the solution as listed in the **Modify Report** dialog box.

For example: `"Setup1 : Last Adaptive"`

`<ContextArray>`

Type: Array of strings

Context for which the expression is being evaluated. This can be an empty string if there is no context.

Array("Domain:=", <DomainType>)

`<DomainType>`

ex. "Sweep" or "Time"

`Array("Context:=", <SimValueContext>)`

Context for the trace. For more information see SimValueContext.

`<FamiliesArray>`

Type: Array of strings

Contains sweep definitions for the report.

Array("<VariableName>:= ", <ValueArray>)

`<ValueArray>`

Array("All") or Array("Value1", "Value2", ..."Value*n*")

examples of <VariableName>

"Freq", "Theta", "Distance"

`<ReportDataArray>`

Type: Array of strings

This array contains the report quantity and X, Y, and (Z) axis definitions.

Array("X Component:=", <VariableName>, "Y Component:=", <Variable-Name> | <ReportQuantityArray>)

`<ReportQuantityArray>`

ex. Array("dB(S(Port1, Port1))")

Example:

```
oModule.CreateReport "XY Stacked Plot 1", "Standard",
"Rectangular Stacked Plot",  _
"LinearFrequency", Array("NAME:Context", "SimValueCon-
text:=", Array(3, 0, 2, 0,  _
false, false, -1, 1, 0, 1, 1, "", 0, 0)), Array("F:=",
Array("All")), Array("X Component:=",  _
"F", "Y Component:=", Array("dB(S(Port1,Port1))",
"dB(S(Port1,Port2))",  _
"dB(S(Port2,Port1))", "dB(S(Port2,Port2))")), Array()
```

Example:

```
oModule.CreateReport "Data Table 1", "Standard", "Data
Table", "LinearFrequency",_
Array("NAME:Context", "SimValueContext:=", Array( _
3, 0, 2, 0, false, false, -1, 1, 0, 1, 1, "", 0, 0)),
Array("F:=", Array("All")), Array("X Component:=",  _
"F", "Y Component:=", Array("dB(S(Port1,Port1))")),
Array()
```

Example:

```
oModule.CreateReport "3D Rectangular Plot 1", "Standard",
"3D Rectangular Plot",  _
"LinearFrequency", Array("NAME:Context", "SimValueCon-
text:=", Array(3, 0, 2, 0,  _
false, false, -1, 1, 0, 1, 1, "", 0, 0)), Array("F:=",
Array("All")), Array("X Component:=",  _
"F", "Y Component:=", "F", "Z Component:=",
Array("dB(S(Port1,Port1))")), Array()
```

Example:

```
oModule.CreateReport "3D Rectangular Plot 2", "Standard",
"3D Rectangular Plot",  _
"LinearFrequency", Array("NAME:Context", "SimValueCon-
text:=", Array(3, 0, 2, 0,  _
```

```
false, false, -1, 1, 0, 1, 1, "", 0, 0)), Array("F:=",
Array("All")), Array("X Component:=",  _
"F", "Y Component:=", "F", "Z Component:=",
Array("dB(S(Port1,Port1))")), Array()
```

For Q3D Extractor the CreateReport command details are as follows.

CreateReport (Q3D Extractor)

Use: Creates a new report with a single trace and adds it to the **Results** branch in the project tree. To add more traces, use the **AddTraces** command. To edit the display properties, use the **ChangeProperty** Script command.

Command: **Q3D Extractor>Results>Create <type> Report**

Syntax:
```
CreateReport <ReportName> <ReportType> <DisplayType>
<SolutionName> <Context> <ReportDataArray>
```

Return Value: None

Parameters:

`<ReportName>`

Type: <string>

Name of Report.

`ReportType`

Possible values are:

`"Matrix".`

`"DC R/L Fields".`

`"AC R/L Fields".`

`"C Fields".`

`DisplayType`

`"Rectangular Plot", "Data Table", or "3D Rectangular Plot".`

`<TraceArray>`

```
Array("NAME:Traces",
   <OneTraceArray>, <OneTraceArray>,...)
```

`<OneTraceArray>`

```
Array("NAME:<TraceName>,
   "SolutionName:=","string",
   "Context:=","string",
   <DisplayTypeDependentData>)
```

`<SolutionName>`

Name of the solution as listed in the **Traces** dialog box.

```
   For example: "Setup1 : Last Adaptive"
<Context>
   Context for which the output variable expression is
   being evaluated.  This can be an empty string if there
   is no context.
   Example: "Line1" or ""
   Field reports usually require a polyline (e.g. "Line1")
   unless they are integrations. Q3D Extractor matrix data
   requires a Reduce Matrix operation(e.g. "Original").
<DisplayTypeDependentData>
   This data varies according to the display type. See the
   examples below.
```

Example:

```
oDesign.CreateReport Array("NAME:Rept2DRectTime",_
   "ReportType:=","Matrix",_
   "DisplayType:=","Rectangular Plot", _
   Array("NAME:Traces", _
      Array("NAME:Trace1", _
         "SolutionName:=", _
            "Setup1 : Adaptive_2", _
         "Context:=","Original", _
         "XComponent:=", "Pass", _
         "YComponent:=", "C(Box1, Box1)",_
         "YAxis:=", 1)))
```

CreateReport (2D Extractor)

Use: Creates a new report and adds it to the **Results** branch in the project tree.

Command: **2D Extractor>Results>Create <type> Report**

Syntax:

```
CreateReport <ReportDataArray>
```

Return Value: None

Parameters:

```
<ReportDataArray>
   Array("NAME:<ReportName>",
      "ReportType:=", <string>,
      "DisplayType:=", <string>,
      <TraceArray>)
ReportType
```

Possible values are:

"Matrix".

"RL Fields".

"CG Fields".

DisplayType

"Rectangular Plot", "Data Table", or "3D Rectangular Plot".

<TraceArray>

Array("NAME:Traces",

<OneTraceArray>, <OneTraceArray>,...)

<OneTraceArray>

Array("NAME:<TraceName>,

"SolutionName:=","string",

"Context:=","string",

<DisplayTypeDependentData>)

<SolutionName>

Name of the solution as listed in the **Traces** dialog box.

For example: "Setup1 : Last Adaptive"

<Context>

Context for which the output variable expression is being evaluated. This can be an empty string if there is no context.

Example: "Line1" or ""

Field reports usually require a polyline (e.g. "Line1") unless they are integrations. 2D Extractor matrix data requires a Reduce Matrix operation(e.g. "Original").

<DisplayTypeDependentData>

This data varies according to the display type. See the examples below.

Example:

```
oDesign.CreateReport Array("NAME:Rept2DRectTime",_
  "ReportType:=","Matrix",_
  "DisplayType:=","Rectangular Plot", _
  Array("NAME:Traces", _
    Array("NAME:Trace1", _
      "SolutionName:=", _
        "Setup1 : Adaptive_2", _
      "Context:=","Original", _
```

```
                    "XComponent:=", "Pass", _
                    "YComponent:=", "C(Box1, Box1)",_
                    "YAxis:=", 1)))
```

CreateReportFromTemplate

Use: Create a report from a saved template.

Command: **HFSS>Results>PersonalLib>**<TemplateName>

Syntax: `CreateReportFromTemplate "<TemplatePath>"`

Return Value: A new report.

Parameters: `<TemplatePath>`

Type: <string>

Path to report template.

Example:

```
Set oProject = oDesktop.SetActiveProject("wg_combiner")
Set oDesign = oProject.SetActiveDesign("wg_combiner")
Set oModule = oDesign.GetModule("ReportSetup")
oModule.CreateReportFromTemplate  _
  "C:\MyHFSS11Projects\PersonalLib\" & _
  "ReportTemplates\TestTemplate.rpt"
```

For Q3D Extractor the command CreateReportFromTemplate details are as follows:

Use: Create a report from a saved template.

Command: **Q3D Extractor>Results>PersonalLib>**<TemplateName>

Syntax: `CreateReportFromTemplate "<TemplatePath>"`

Return Value: A new report.

Parameters: `<TemplatePath>`

Type: <string>

Path to report template.

Example:

```
Set oProject = oDesktop.SetActiveProject("connector")
Set oDesign = oProject.SetActiveDesign("connector")
Set oModule = oDesign.GetModule("ReportSetup")
oModule.CreateReportFromTemplate  _
  "C:\MyQ3DProjects\PersonalLib\" & _
"ReportTemplates\TestTemplate.rpt"
```

CreateReportOfAllQuantities

Use: Create a report including all quantities in a category. Cannot create a report with expressions.

Command: None.

Syntax:
```
CreateReportOfAllQuantities(reportNameArg, reportTypeArg,
displayTypeArg, solutionNameArg, simValueCtxtArg,
categoryNameArg, pointSetArg,
commonComponentsOfTracesArg, extTraceInfoArg);
```

Return Value: Report of all quantities in a category.

Parameters:

`reportTypeArg`

Report type name as input parameter.

`displayTypeArg`

Display type name as input parameter.

`solutionNameArg`

Solution name as input parameter.

`simValueCtxtArg`

a context name, or array of string that encoded the context(I).

`categoryNameArg`

a category name as input parameter.

`pointSetArg`

Array of strings(II).

`commonComponentsOfTracesArg`

Array of strings (III)

`extTraceInfoArg`

Array of strings(IV)

Example:

```
solutions= oModule .CreateReportOfAllQuantities("Smith Chart all",
"Modal Solution Data", "Smith Chart", "Setup1 : LastAdaptive", [],"S
Parameter",["Freq:=", ["All"], "offset:=", ["All"],"a:=", ["Nomi-
nal"],"b:=", ["Nominal"]],[], [])
```

DeleteMarker

Use: Deletes the specified marker.

Command: **Q3DExtractor>Fields>Fields>Marker>Delete Marker**

Syntax:
```
DeleteMarker <MarkerName>
```

Return Value: None

Parameters:

```
<MarkerName>
Type: <string>
Name of the marker.
```

Example:

```
Dim oAnsoftApp
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftQ3D.Q3DScriptInter-
face")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Solenoid")
Set oDesign = oProject.SetActiveDesign("Solenoid")
Set oModule = oDesign.GetModule("FieldsReporter")
oModule.DeleteMarker "m1"
```

DeleteAllReports

Use: Deletes all existing reports.

Command: Right-click the report to delete in the project tree, and then click **Delete All Reports** on the shortcut menu.

Syntax: `DeleteAllReports`

Return Value: None

Example:

```
oModule.DeleteAllReports
```

DeleteReports

Use: Deletes an existing report or reports.

Command: Right-click the report to delete in the project tree, and then click **Delete** on the shortcut menu.

Syntax: `DeleteReports(<ReportNameArray>)`

Return Value: None

Parameters:

```
<ReportNameArray>
```

Type: Array of strings

Example:

```
oModule.DeleteReports Array("Rept2DRectFreq")
```

DeleteTraces

Use: Deletes an existing traces or traces.

Command: Right-click the report to delete in the project tree, and then click **Delete** on the shortcut menu.

Syntax: `DeleteTraces(<TraceSelectionArray>)`

Return Value: None

Parameters:

`<TraceSelectionArray>`

Type: Array of strings

Array("<ReportName>:=", <TracesArray>, <TracesArray>,...)

`<ReportName>`

Type: <string>

Name of Report.

`<TracesArray>`

Type: Array of strings

This array contains the traces to delete within a report.

Array(<Trace>, <Trace>, ...)

`<Trace>`

Type: string

Example:

```
oModule.DeleteTraces Array("XY Plot 1:=", Array("dB(S(LumpPort1,Lump-
Port1))"), "XY Plot 2:=", Array("Mag_E"))
```

An example applicable for Q3D Extractor is as follows.

Example:

```
oModule.DeleteTraces Array("XY Plot 1:=", Array("R(via:source1,
via:source1)"), "XY Plot 2:=", Array("char_imped"))
```

EditQuickEyeAnalysis

Use: Modifies an existing QuickEye analysis.

Command: Double-click on the analysis in the project tree.

Syntax:

```
EditQuickEyeAnalysis (<string>,      // name of analysis
to edit
```

Array("NAME:SimSetupName",
"DataBlockID:=", <int>,
"SimSetupID:=", <int>,
"OptionName:=", <string>,
"AdditionalOptions:=", <string>,
"AlterBlockName:=", <string>,
"FilterText:=", <string>,
"AnalysisEnabled:=", <int>, // 1 if enabled, 0 if disabled
Array("NAME:OutputQuantities", <QuantityArray>, <QuantityArray>...)
Array("NAME:NoiseOutputQuantities", <QuantityArray>, <QuantityArray>...)),
"Name:=", <string>, // name for new analysis
"QuickEyeAnalysis:=", Array(<string>, // Input rise time
<string>, // Input low voltage
<string>, // Input high voltage
<string>, // bits per second
<string>, // number of FFE taps
<string>, // Random Jitter Standard Deviation
<string>, // Delay
<string>, // Duty cycle distortion
<bool>, // true if unit interval, false if bits per second
<string>, // number of DFE taps
<bool>, // true to calculate FFE, false if weights are specified
<bool>, // true to calculate DFE, false if weights are specified
<string>, // DFE decision threshold
<string>, // DFE decision high
<string>, // DFE decision low
<bool>), // true if using specified equalization, false if disabled
Array("NAME:SweepDefinition",
"Variable:=", <string>,
"Data:=", <string>, // sweep
"OffsetF1:=", <bool>,
"Synchronize:=", <int>), // 1 to Synchronize, 0 otherwise
"FFEWts:=", Array(".5", "-2"), // optional, specified weights
"DFEWts:=", Array("2"))) // optional, specified weights

Return Value: <string> – // Name of the analysis after being modified

// If the name requested conflicts with the name of an existing
// analysis, the requested name is altered to be unique.
// The name returned reflects any change made to be unique.

Parameters:

<QuantityArray>:
Array("NAME:Quantity",
"NodeType:=", <string>, // CompInst, Variable, Net, Harmonics, or Custom
"CompID:=", <string>,
"CompName:=", <string>,
"QuantityName:=", <string>,
"Selected:=", <bool>,
"UnitType:=", <string>,
"DataType:=", <string>, // Real, Complex, Integer, Enum, Char, Free, Array, Record
"CircuitInstanceID:=", <string)

Example:

```
dim name
name = oModule.EditQuickEyeAnalysis ("MyQuickEyeAnalysis",
Array("NAME:SimSetup", "DataBlockID:=", 28, "SimSetupID:=",  _
3, "OptionName:=", "(Default Options)", "AdditionalOptions:=", "",
"AlterBlockName:=",  _
"", "FilterText:=", "", "AnalysisEnabled:=", 1, Array("NAME:Out-
putQuantities", Array("NAME:Quantity", "NodeType:=", "CompInst",
"CompID:=",  _
"5", "CompName:=", "Level01_NPN_Model_5", "QuantityName:=", "I",
"Selected:=",  _
true, "UnitType:=", "NoUnit", "DataType:=", "Real", "CircuitInstan-
ceID:=", ""), Array("NAME:Quantity", "NodeType:=",  _
"CompInst", "CompID:=", "10", "CompName:=", "RES__10", "Quanti-
tyName:=", "I", "Selected:=",  _
true, "UnitType:=", "NoUnit", "DataType:=", "Real", "CircuitInstan-
ceID:=", ""), Array("NAME:Quantity", "NodeType:=",  _
"Net", "CompID:=", "", "CompName:=", "net_47", "QuantityName:=", "V",
"Selected:=",  _
true, "UnitType:=", "NoUnit", "DataType:=", "Real", "CircuitInstan-
ceID:=", "")), Array("NAME:NoiseOutputQuantities"), "Name:=",  _
"MyQuickEyeAnalysis", "QuickEyeAnalysis:=", Array("5e-10", "0", "1",
"1e-9", "2",  _
"0", "1e-9", "0", true, "1", false, false, "0", "1", "-1", true),
Array("NAME:SweepDefinition", "Variable:=",  _
```

```
"Temp", "Data:=", "LINC 1cel 10cel 10", "OffsetF1:=", false, "Syn-
chronize:=",  _
0), "FFEWts:=", Array(".5", "-2"), "DFEWts:=", Array("2")))
```

EditVerifEyeAnalysis

Use: Edits an existing VerifEye analysis.

Command: Double-click on the analysis in the project tree.

Syntax:

EditVerifEyeAnalysis (<string>, // name of analysis to edit
Array("NAME:SimSetupName",
"DataBlockID:=", <int>,
"SimSetupID:=", <int>,
"OptionName:=", <string>,
"AdditionalOptions:=", <string>,
"AlterBlockName:=", <string>,
"FilterText:=", <string>,
"AnalysisEnabled:=", <int>, // 1 if enabled, 0 if disabled
Array("NAME:OutputQuantities", <QuantityArray>, <QuantityArray>...)
Array("NAME:NoiseOutputQuantities", <QuantityArray>, <QuantityArray>...)),
"Name:=", <string>, // name for modified analysis
"VerifEyeAnalysis:=", Array(<string>, // Input rise time
<string>, // Input low voltage
<string>, // Input high voltage
<string>, // bits per second
<string>, // number of FFE taps
<string>, // Random Jitter Standard Deviation
<string>, // Delay
<string>, // Duty cycle distortion
<bool>, // true if unit interval, false if bits per second
<string>, // number of DFE taps
<bool>, // true to calculate FFE, false if weights are specified
<bool>, // true to calculate DFE, false if weights are specified
<string>, // DFE decision threshold
<string>, // DFE decision high
<string>, // DFE decision low
<bool>), // true if using specified equalization, false if disabled

Array("NAME:SweepDefinition",
"Variable:=", <string>,
"Data:=", <string>, // sweep
"OffsetF1:=", <bool>,
"Synchronize:=", <int>), // 1 to Synchronize, 0 otherwise
"FFEWts:=", Array(".5", "-2"), // optional, specified weights
"DFEWts:=", Array("2"))) // optional, specified weights

Return Value: <string> – // Name of the analysis after being modified
// If the name requested conflicts with the name of an existing
// analysis, the requested name is altered to be unique.
// The name returned reflects any change made to be unique.

Parameters: <QuantityArray>:
Array("NAME:Quantity",
"NodeType:=", <string>, // CompInst, Variable, Net, Harmonics, or Custom
"CompID:=", <string>,
"CompName:=", <string>,
"QuantityName:=", <string>,
"Selected:=", <bool>,
"UnitType:=", <string>,
"DataType:=", <string>, // Real, Complex, Integer, Enum, Char, Free, Array, Record
"CircuitInstanceID:=", <string)

Example:

```
dim name
name = oModule.EditVerifEyeAnalysis ("MyVerifEyeAnalysis",
Array("NAME:SimSetup", "DataBlockID:=", 27, "SimSetupID:=", _
1, "OptionName:=", "Options", "AdditionalOptions:=", "", "AlterBlock-
Name:=", "", "FilterText:=", _
"", "AnalysisEnabled:=", 1, Array("NAME:OutputQuantities",
Array("NAME:Quantity", "NodeType:=", "CompInst", "CompID:=", _
"5", "CompName:=", "Level01_NPN_Model_5", "QuantityName:=", "I",
"Selected:=", _
true, "UnitType:=", "NoUnit", "DataType:=", "Real", "CircuitInstan-
ceID:=", ""), Array("NAME:Quantity", "NodeType:=", _
"CompInst", "CompID:=", "10", "CompName:=", "RES__10", "Quanti-
tyName:=", "I", "Selected:=", _
true, "UnitType:=", "NoUnit", "DataType:=", "Real", "CircuitInstan-
ceID:=", ""), Array("NAME:Quantity", "NodeType:=", _
```

```
"Net", "CompID:=", "", "CompName:=", "net_47", "QuantityName:=", "V",
"Selected:=",  _
true, "UnitType:=", "NoUnit", "DataType:=", "Real", "CircuitInstan-
ceID:=", ""), Array("NAME:NoiseOutputQuantities"), "Name:=",  _
"MyVerifEyeAnalysis", "VerifEyeAnalysis:=", Array("5e-10", "0", "1",
"1e-9", "2",  _
"0", "1e-9", "0", true, "1", false, false, "0", "1", "-1", true),
Array("NAME:SweepDefinition", "Variable:=",  _
"Temp", "Data:=", "LINC 1cel 10cel 10", "OffsetF1:=", false, "Syn-
chronize:=",  _
0), "FFEWts:=", Array(".5", "-2"), "DFEWts:=", Array("2")))
```

ExportPlotImageToFile [Reporter]

Use: Create field plot exports of existing field plots from a given view points, and with the model being auto-sized automatically for each view.

Command: None.

Syntax: `ExportPlotImageToFile(<FileName>, "", <plotItemName>, <setViewTopDownDirectionByRelativeCS>)`

Return Value: An image file.

Parameters:

`<FileName>`

Type: <string>

Full path plus file name.

`""`

Type: <EmptyString>

`<PlotItemName>`

Type: <string>

Name of fields to plot.

`<SetViewTopDownDirectionByRelativeCS>`

Type: <string>

Name of relative coordinate system to use for the field plot.

Example:

```
'This example demonstrates the creation of E-field plots 'of three
different orientations:
'Mag_E1 created in-plane with the XY-plane of the Global 'coordinate
system
```

```
'Mag_E2 created in-plane with the XY-plane of the RelativeCS1 coordi-
nate system
'Mag_E3 created in-plane with the XY-plane of the RelativeCS2 coordi-
nate system
Dim oAnsoftApp, oDesktop, oProject, oDesign, oEditor, oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.GetActiveProject()
Set oDesign = oProject.GetActiveDesign()

Set oModule = oDesign.GetModule("FieldsReporter")
'First set the module "FieldsReporter"

oModule.ExportPlotImageToFile "C:\TestEPITF1.jpg", "", "Mag_E1",
"Global"
oModule.ExportPlotImageToFile "C:\TestEPITF2.jpg", "", "Mag_E2",
"RelativeCS1"
oModule.ExportPlotImageToFile "C:\TestEPITF3.jpg", "", "Mag_E3",
"RelativeCS2"
```

ExportReport

Note The ExportReport script command has been replaced by the script command ExportToFile. ExportReport remains in order to retain backward compatibility for existing scripts, but it is strongly recommended that you now use ExportToFile.

Use:	Export a report to a data file.
Command:	None
Syntax:	`ExportReport <ReportName>, <FileName>, <FileExtension>`
Return Value:	None
Parameters:	`<ReportName>` Type: string <Filename> Type: string <FileExtension> Type: string

Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule

Set oAnsoftApp = CreateObject("AnsysDesigner.Design-
erScript")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow

Set oProject = oDesktop.SetActiveProject("BJTinverter")
Set oDesign = oProject.SetActiveDesign("Nexxim1")
oDesign.ExportReport "Data Table 1", "table_test", "csv"
```

ExportToFile

Note The ExportToFile script command has replaced the script command ExportReport. ExportReport remains in order to retain backward compatibility for existing scripts, but it is strongly recommended that you now use ExportToFile.

Use: From a data table or plot, generates text format, comma delimited, tab delimited, or .dat type output files.

Command: Right-click on report name in the Project tree and select Export Data.

Syntax: `ExportToFile <ReportName>, <FileName>`

Return Value: None

Parameters: *<ReportName>*

Type: string

<Filename>

Type: string

.txt	Post processor format file
.csv	Comma-delimited data file
.tab	Tab-separated file
.dat	ANSYS plot data file

Example:

```
Set oModule = oDesign.GetModule('ReportSetup')
oModule.ExportToFile('Plot 1','c:\report1.dat')
```

ExportToFile [Reporter]

Use: From a data table or plot, generates text format, comma delimited, tab delimited, or .dat type output files.

Command: Right-click on report name in the Project tree and select **Export Data**.

Syntax:

```
ExportToFile <ReportName>, <FileName>
```

Return Value: None

Parameters:

<ReportName>

Type: string

<FileName>

Type: string

Path and file name.

.txt	Post processor format file
.csv	Comma-delimited data file
.tab	Tab-separated file
.dat	Ansoft plot data file

Example:

```
oModule = oDesign.GetModule('ReportSetup')
oModule.ExportToFile('Plot 1','c:\report1.dat')
```

ExportMarkerTable

The documented command is applicable for Q3D Extractor.

Use: Exports the marker table to a .csv or .tab file.

Command: **Q3DExtractor>Fields>Fields>Marker>ExportMarkerTable**

Syntax:

```
ExportMarkerTable <pathandfilename>
```

Return Value: none

Parameters:

```
<pathandfilename>
```

Example:

```
Dim oAnsoftApp
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftQ3D.Q3DScriptInter-
face")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Solenoid")
Set oDesign = oProject.SetActiveDesign("Solenoid")
Set oModule = oDesign.GetModule("FieldsReporter")
oModule.ExportMarkerTable "C:/work/FieldMarkerTable.csv"
```

FFTOnReport

Use: Perform an FFT on a selected report.

Command: **HFSS>Results>Perform FFT on Report**

Syntax: `FFTOnReport "<plotName>", <FFTWindowType>, "<function>"`

Return Value: Creates a plot named FFT "PlotName"

Parameters:

<PlotName>

Type: string

<FFTWindowType>

Type: string

Rectangular, Tri, Van Hann, Hamming, Blackman,Lanczos, Weber, Welch.

<function>

Type: string

<none>, ang_deg, ang_rad, arg, cang_deg, cang_rad, dB, dB1 normalize, dB20normalize, dBc, im, mag, normalize, re.

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 14.0.0
' 3:39:35 PM  Mar 16, 2011
```

```
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("OptimTee")
Set oDesign = oProject.SetActiveDesign("TeeModel")
Set oModule = oDesign.GetModule("Solutions")
oModule.FFTOnReport "XY Plot 1", "Rectangular", "dB"
```

For Q3D Extractor the details for FFTOnReports are as follows:

Use: Perform an FFT on a selected report.

Command: **Q3D Extractor** or **2D Extractor>Results>Perform FFT on Report**

Syntax: `"<plotName>", <FFTWindowType>, "<function>"`

Return Value: Creates a plot named FFT "Plot Name".

Parameters:

<PlotName>

Type: string

<FFTWindowType>

Type: string

Rectangular, Tri, Van Hann, Hamming, Blackman,Lanczos, Weber, Welch.

<Function>

Type: string

<none>, ang_deg, ang_rad, arg, cang_deg, cang_rad, dB, dB1 normalize, dB20normalize, dBc, im, mag, normalize, re.

Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
```

```
Set oAnsoftApp = CreateObject("AnsoftQ3D.Q3DScriptInter-
face")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("OptimTee")
Set oDesign = oProject.SetActiveDesign("TeeModel")
Set oModule = oDesign.GetModule("Solutions")
oModule.FFTOnReport "XY Plot 1", "Rectangular", "dB"
```

GetAllReportNames

Use: Gets the names of existing reports in a design.

Syntax: `GetAllReportNames()`

Return Value: Array of report names.

Parameters: `None`

Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHFSS.HFSSScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.GetActiveProject
Set oDesign = oProject.GetActiveDesign
Set oReportModule = oDesign.GetModule("ReportSetup")
Dim names
names = oReportModule.GetAllReportNames
For index = 0 to UBound(names)
  MsgBox(names(index))
Next
Set oFieldReportModule = oDesign.GetModule("FieldsReporter")
Set collection = oFieldReportModule.GetFieldPlotNames
For index = 0 to collection.Count-1
```

```
    MsgBox(collection.Item(index))
  Next
```

GetAllCategories

Use: Get all available category names (not including variable and output-variables) in a solution for a report type and display type, returned as an array of text strings.

Command: None

Syntax: `GetAllCategories(reportTypeArg,displayTypeArg, solutionNameArg, simValueCtxtArg, categoryName_array)`

Return Value: Array of text strings.

Parameters:

`reportTypeArg`

Report type name as input parameter.

`displayTypeArg`

display type name as input parameter.

`solutionNameArg`

Solution name as input parameter.

`simValueCtxtArg`

Acontext name, or array of strings that encode the contexts(I).

`categoryName_array`

Output parameter for returning category names.

Example:

```
categories= oModule.GetAllCategories("Far Fields", "Rectangular
Plot", "Setup1 : LastAdaptive", "Infinite Sphere1")
```

GetAllQuantities

Use: Gets all available quantity names in category, returned as an array of text strings.

Command: None.

Syntax: `GetAllQuantities(reportTypeArg, displayTypeArg, solutionNameArg, simValueCtxtArg, categoryNameArg, quantityName_array);`

Return Value: Array of text strings.

Parameters:

`reportTypeArg`

Report type name as input parameter.

`displayTypeArg`

Display type name as input parameter.

`solutionNameArg`

Solution name as input parameter.

`simValueCtxtArg`

A context name, or array of string that encoded the contexts(I).

`categoryNameArg`

A category name as input parameter. a category name returned in GetAllCategories() or "Variables", or "Output Variables"

`quantityName_array`

Output parameter for returning quantity names found in a category.

Example:

```
quantities= oModule.GetAllQuantities("Far Fields", "Rectangular
Plot", "Setup1 : LastAdaptive", "Infinite Sphere1", "Gain")
```

GetAvailableDisplayTypes

Use: All supported display types in report type as an array of text strings.

Command: None

Syntax: `GetAvailableDisplayTypes(reportTypeArg,displayType_array);`

Return Value: Array of text strings

Parameters: `reportTypeArg`

report type name as input parameter

`displayType_array`

output parameter for returning display types

Example:

```
displayTypes = oModule .GetAvailableDisplayTypes("Far Fields")
```

GetAvailableReportTypes

Use: Get all available report types in the current Design as an array of text string.

Command: None.

Syntax: `GetAvailableReportTypes([out, retval] VARIANT* reportType_array)`

Return Value: array of text string

Parameters: `reportType_array`

output parameter for returning report types

Example:

```
reportTypes = oModule .GetAvailableReportTypes()
```

GetAvailableSolutions

Use: Get all available solutions in report type as an array of text strings.

Command: None

Syntax: `GetAvailableSolutions(reportTypeArg,solution_array)`

Return Value: Array of text strings.

Parameters: `reportTypeArg`

Report type name as input parameter.

`solution_array`

Output parameter for returning solution names.

Example:

```
solutions= oModule .GetAvailableSolutions("Far Fields")
```

GetDisplayType

Use: Get the display type of a report.

Command: None

Syntax: `GetDisplayType "<reportName>"`

Return Value: Report <displaytype> of a report.

`<DisplayType>`

Type: <string>

If `ReportType` is `"Modal S Parameters"`, `"Terminal S Parameters"`, or `"Eigenmode Parameters"`, then returns one of the following:

`"Rectangular Plot"`, `"Polar Plot"`, `"Radiation Pattern"`, `"Smith Chart"`, `"Data Table"`, `"3D Rectangular Plot"`, or `"3D Polar Plot"`.

If `<ReportType>` is `"Fields"`, then returns one of the following:

`"Rectangular Plot"`, `"Polar Plot"`, `"Radiation Pattern"`, `"Data Table"`, or `"3D Rectangular Plot"`.

If `<ReportType>` is `"Far Fields"` or `"Near Fields"`, then returns one of the following:

"Rectangular Plot", "Radiation Pattern", "Data Table", "3D Rectangular Plot", or "3D Polar Plot"

If <ReportType> is "Emission Test", then returns one of the following:
"Rectangular Plot" or "Data Table"

Parameters:

<ReportName>
Type: <string>
Report name.

Example:

```
Set oDesign = oProject.SetActiveDesign("wg_combiner")
Set oModule = oDesign.GetModule("ReportSetup")
MyPlotDisplayType = oModule.GetDisplayType "XY Plot1"
```

For Q3D Extractor the GetDisplayType command has the following details.

Use: Get the display type of a report.

Command: None

Syntax: GetDisplayType "<reportName>"

Return Value: Report <displaytype> of a report.

<DisplayType>
Type: <string>
If ReportType is "Matrix Report", then returns one of the following:
"Rectangular Plot", "Data Table", or "3D Rectangular Plot".
If <ReportType> is "CG Fields", ""DC R/L Fields", or "AC R/L Fields", then returns one of the following:
"Rectangular Plot", "Data Table", or "3D Rectangular Plot".

Parameters:

<ReportName>
Type: <string>
Report name.

Example:

```
Set oDesign = oProject.SetActiveDesign("connector")
Set oModule = oDesign.GetModule("ReportSetup")
MyPlotDisplayType = oModule.GetDisplayType "XY Plot1"
```

GetSolutionContexts

Use: Get all available solution context names in a solution as an array of text strings.

Command: None.

Syntax: `GetSolutionContexts(reportTypeArg,displayTypeArg, solutionNameArg, contextName_array);`

Return Value: Array of text strings.

Parameters:

`reportTypeArg`

Report type name as input parameter.

`displayTypeArg`

Display type name as input parameter.

`solutionNameArg`

Solution name as input parameter.

`contextName_array`

Output parameter for returning context names.

Example:

```
contexts= oModule .GetSolutionContexts("Far Fields", "Rectangular
Plot", "Setup1 : LastAdaptive")
```

ImportIntoReport

Use: Imports .tab, .csv, and .dat format files into a report.

Command: Right-click on report name in the Project tree and select **Export Data**.

Syntax: `ImportIntoReport(<ReportName>, <FileName>)`

Return Value: None

Parameters:

<ReportName>

Type: string

<FileName>

Type: string

Path and file name.

.csv	Comma-delimited data file
.tab	Tab-separated file
.dat	Ansoft plot data file

Example:

```
oModule = oDesign.GetModule('ReportSetup')
oDesign.ImportIntoReport('Plot 1','c:\report1.dat')
```

PasteReports

Use: Paste copied reports to results in the current project.

Command: Paste

Syntax: `PasteReports`

Return Value: None

Parameters: `None`

Example:

```
oModule.PasteReports
```

PasteTraces

Use: To paste copied traces to a named plot.

Command: Paste

Syntax: `PasteTraces('<plotName>')`

Return Value: None

Parameters: `<plotName>`

Type: <string>

Name of plot.

Example:

```
oModule.PasteTraces "XY Plot1"
```

RenameReport

Use: Renames an existing report.

Command: Select a report on the Project tree, right-click and select Rename

Syntax: `RenameReport <OldReportName>, <NewReportName>`

Return Value: None

Parameters: *`<OldReportName>`*

Type: string

`<NewReportName>`

Type: string

Example:

```
oModule.RenameReport "XY Plot1", "Reflection"
```

An example related to Q3D Extractor is as follows.

Example:

```
oModule.RenameReport "XY Plot1", "Reflection"
```

RenameTrace

Use: To rename a trace in a plot

Command: None

Syntax: `RenameTrace "<plotName>" "<traceID>" "<newName>"`

Return Value: None

Parameters:

`<plotName>`

Type: <string>

Name of plot.

`<traceID>`

Type: <string>

Name of trace.

`<newName>`

Type: <string>

New trace name.

Example:

```
oModule.RenameTrace "XY Plot1", "dB(S(WavePort1,WavePort1))1",_
"Port1dbS"
```

An example related to Q3D Extractor is as follows:

Example:

```
oModule.RenameTrace "XY Plot1", "ACR(trace:src1,
trace:src1)", "TraceRes"
```

UpdateAllReports

Use: Updates the specified reports in the Results branch in the project tree.

Command: **HFSS>Results>Update All Reports**

Syntax: `UpdateReports Array("<plotname>")`

Return Value: None.

Parameters:

`"<plotname>"`

Type: <string>

Name of plot.

Example:

```
Set oModule = oDesign.GetModule("ReportSetup")
```

```
oModule.UpdateAllReports
```

UpdateReports

Use: Updates the specified reports in the Results branch in the project tree.

Command: **Update Report**

Syntax: `UpdateReports Array("<plotname>")`

Return Value: None.

Parameters: `"<plotname>"`

Type: <string>

Name of plot.

Example:

```
Set oModule = oDesign.GetModule("ReportSetup")
oModule.UpdateReports Array("XY Plot 1")
```

UpdateTraces

Use: Update the traces in a report for which traces are not automatically updated by the Report Traces dialog, Update Report, Real Time selection.

Command: Report dialogue, Apply Traces button

Syntax: `UpdateTraces "<plotName>" Array("<TraceDef>") Array()`

Return Value:

Parameters: `<ReportName>`

Type: <string>

Name of Report.

`<SolutionName>`

Type: <string>

Name of the solution as listed in the **Modify Report** dialog box.

For example: `"Setup1 : Last Adaptive"`

`<Context`*`Array`*`>`

Type: Array of strings

Context for which the expression is being evaluated. This can be an empty string if there is no context.

Array(“Domain:=”, <DomainType>)

<DomainType>

ex. “Sweep” or “Time”

Array(“Context:=”, <GeometryType>)

<GeometryType>

ex. “Infinite Sphere*n*”, “Sphere*n*”, “Polyline*n*”

`<FamiliesArray>`

Type: Array of strings

Contains sweep definitions for the report.

Array(“<VariableName>:= “, <ValueArray>)

<ValueArray>

Array(“All”) or Array(“Value1”, “Value2”, ...”Value*n*”)

examples of <VariableName>

“Freq”, “Theta”, “Distance”

`<ReportDataArray>`

Type: Array of strings

This array contains the report quantity and X, Y, and (Z) axis definitions.

Array(“X Component:=”, <VariableName>, “Y Component:=”, <VariableName> | <ReportQuantityArray>)

<ReportQuantityArray>

ex. Array(“dB(S(Port1, Port1))”)

`Array()`

Type: Empty array.

Denotes the end of the UpdateTraces command.

Example:

```
Set oModule = oDesign.GetModule("ReportSetup")
oModule.UpdateTraces "XY Plot1", _ Array("dB(S(WavePort1,Wave-
Port1))"), _
  "Setup1 : Sweep1", _
Array("Domain:=", "Sweep"), _
Array("Freq:=", Array("All")), _
Array("X Component:=", "Freq", _
"Y Component:=", Array("dB(S(WavePort1,WavePort1))")), _
```

```
Array()
```

Example:

```
oModule.UpdateTraces "XY Plot 1", Array("dB(S(WavePort1,Wave-
Port1))"),  _
  "Setup1 : Sweep1", _
Array("Domain:=", "Time", "HoldTime:=", 1, _
"RiseTime:=", 0, "StepTime:=", 0, "Step:=", false, _
"WindowWidth:=", 1, _
"WindowType:=", 0, "KaiserParameter:=", 1, _
"MaximumTime:=", 0), _
Array("Time:=", Array("All")), _
Array("X Component:=", "Time", _
"Y Component:=", Array("dB(S(WavePort1,WavePort1))")), _
Array()
```

For Designer the **Update** command details are as follows:

Use: Update the traces in a report for which traces are not automatically updated by the Report Traces dialog, Update Report, Real Time selection.

Command: Report dialogue, Apply Traces button

Syntax: `UpdateTraces "<plotName>" Array("<TraceDef>") Array()`

Return Value: None

Parameters:

`<ReportName>`

Type: <string>

Name of Report.

<SolutionName>

Type: <string>

Name of the solution as listed in the **Modify Report** dialog box.

For example: `"Setup1 : Last Adaptive"`

<ContextArray>

Type: Array of strings

Context for which the expression is being evaluated. This can be an empty string if there is no context.

Array("Domain:=", <DomainType>)

<DomainType>
ex. “Sweep” or “Time”

Array(“Context:=”, <SimValueContext>)
Context for the trace. For more information see SimValueContext.

<FamiliesArray>
Type: Array of strings
Contains sweep definitions for the report.
Array(“<VariableName>:= “, <ValueArray>)

<ValueArray>
Array(“All”) or Array(“Value1”, “Value2”, ...”Value*n*”)
examples of <VariableName>
“Freq”, “Theta”, “Distance”

<ReportDataArray>
Type: Array of strings
This array contains the report quantity and X, Y, and (Z) axis definitions.
Array(“X Component:=”, <VariableName>, “Y Component:=”, <VariableName> | <ReportQuantityArray>)

<ReportQuantityArray>
ex. Array(“dB(S(Port1, Port1))”)
Array()
Type: Empty array.
Denotes the end of the UpdateTraces command.

Example:

```
Set oModule = oDesign.GetModule("ReportSetup")
oModule.UpdateTraces "XY Plot1", _ Array("dB(S(Wave-
Port1,WavePort1))"), _
  "Setup1 : Sweep1", _
Array("Domain:=", "Sweep"), _
Array("Freq:=", Array("All")), _
Array("X Component:=", "Freq", _
```

```
"Y Component:=", Array("dB(S(WavePort1,WavePort1))")), _
Array()
```

Example:

```
oModule.UpdateTraces "XY Plot 1", Array("dB(S(Wave-
Port1,WavePort1))"),  _
  "Setup1 : Sweep1", _
Array("Domain:=", "Time", "HoldTime:=", 1, _
"RiseTime:=", 0, "StepTime:=", 0, "Step:=", false, _
"WindowWidth:=", 1, _
"WindowType:=", 0, "KaiserParameter:=", 1, _
"MaximumTime:=", 0), _
Array("Time:=", Array("All")), _
Array("X Component:=", "Time", _
"Y Component:=", Array("dB(S(WavePort1,WavePort1))")), _
Array()
```

For Q3D Extractor, the command details are as follows:

Use: Update the traces in a report for which traces are not automatically updated by the **Report Traces** dialog box, Update Report, Real Time selection.

Command: Report dialogue, Apply Traces button

Syntax: `UpdateTraces "<plotName>" Array("<TraceDef>") Array()`

Return Value:

Parameters:

`<ReportName>`

Type: <string>

Name of Report.

`<SolutionName>`

Type: <string>

Name of the solution as listed in the **Modify Report** dialog box.

For example: `"Setup1 : Last Adaptive"`

`<ContextArray>`

Type: Array of strings

Context for which the expression is being evaluated. This can be an empty string if there is no context.

Array("Context:=", <DomainType>)

<DomainType>

ex. "Original" or "RM1"

Array("Context:=", <GeometryType>)

<GeometryType>

ex. "Spher*e*n", "Polylin*e*n"

`<FamiliesArray>`

Type: Array of strings

Contains sweep definitions for the report.

Array("<VariableName>:= ", <ValueArray>)

<ValueArray>

Array("All") or Array("Value1", "Value2", ..."Valu*e*n")

examples of <VariableName>

"Freq"

`<ReportDataArray>`

Type: Array of strings

This array contains the report quantity and X, Y, and (Z) axis definitions.

Array("X Component:=", <VariableName>, "Y Component:=", <VariableName> | <ReportQuantityArray>)

<ReportQuantityArray>

ex. Array("ACR(trace:src1, trace:src1)")

`Array()`

Type: Empty array.

Denotes the end of the UpdateTraces command.

Example:

```
Set oModule = oDesign.GetModule("ReportSetup")
oModule.UpdateTraces "XY Plot1", _ Array("ACR(trace:src1,
trace:src1)"), _ "Setup1 : Sweep1", _
Array("Context:=", "Original"), _
Array("Freq:=", Array("All")), _
Array("X Component:=", "Freq", _
"Y Component:=", Array("ACR(trace:src1, trace:src1)"), _
Array()
```

UpdateTracesContextandSweeps

Use: Use this command to edit sweeps and context of multiple traces without affecting their component expressions.

Command: **Modify Report** with multiple traces selected.

Syntax: `UpdateTracesContextandSweeps`

Return Value: None.

Parameters: `<ReportName>`

Type: <string>

Name of Report.

`Array(<traceIDs>)`

`<traceID>`

Type: <string>

Name of trace.

`<SolutionName>`

Type: <string>

Name of the solution as listed in the **Modify Report** dialog box.

For example: `"Setup1 : Last Adaptive"`

`<Context`*`Array`*`>`

Type: string.

Context for which the expression is being evaluated. This can be an empty string if there is no context.

ex. “Sweep” or “Time”

`Array<pointSet>`

Type: <string>

Point set for the selected traces, for example, X and Y values for the plot.

Example:

```
Set oProject = oDesktop.SetActiveProject("Tee")
Set oDesign = oProject.SetActiveDesign("TeeModel")
Set oModule = oDesign.GetModule("ReportSetup")
oModule.UpdateTracesContextAndSweeps _
"Active S Parameter Quick Report", _
Array( _
  "dB(ActiveS(Port1:1))", "dB(ActiveS(Port2:1))"), _
"Setup1 : Sweep1", Array(), _
Array("Freq:=", _
```

```
Array( _
  "9GHz", "9.05GHz", "9.1GHz", "9.15GHz", "9.2GHz", _
"9.25GHz", "9.3GHz", "9.35GHz",  _
  "9.4GHz", "9.45GHz", "9.5GHz", "9.55GHz", _
"9.6GHz", "9.65GHz", "9.7GHz",  _
  "9.75GHz", "9.8GHz", "9.85GHz", "9.9GHz", "9.95GHz", "10GHz"),_
"offset:=", Array("All"))
```

An example related to Q3D Extractor is as follows:

Example:

```
Set oProject = oDesktop.SetActiveProject("Via")
Set oDesign = oProject.SetActiveDesign("ViaModel")
Set oModule = oDesign.GetModule("ReportSetup")
oModule.UpdateTracesContextAndSweeps _
"C Matrix Quick Report", _
Array( _
  "C(trace, trace)", "C(gnd, gnd)"), _
"Setup1 : Sweep1", Array ("Context:=", "Original")
Array("Freq:=", _
Array("All"))
```

14 Boundary and Excitation Module Script Commands

Boundary and excitation commands should be executed by the `"BoundarySetup"` module.

```
Set oModule = oDesign.GetModule("BoundarySetup")
```

Conventions Used in this Chapter

`<BoundName>`

Type: string.

Name of a boundary.

`<AssignmentObjects>`

Type: Array of strings.

An array of object names.

`<AssignmentFaces>`

Type: Array of integers.

An array of face IDs. The ID of a face can be determined through the user interface using the **3D Modeler>Measure>Area** command. The face ID is given in the **Measure Information** dialog box.

```
<LineEndPoint>
   Array(<double>, <double>, <double>)
```

Legal Names for Boundaries in HFSS Scripts

Perfect E	Radiation
Perfect H	Symmetry
Finite Conductivity	Master
Impedance	Slave
Layered Impedance	Lumped RLC

Legal Names for Excitations in HFSS Scripts

Wave Port	Hertizian-Dipole Incident Wave
Lumped Port	Cylindrical Incident Wave
Voltage	Gaussian Beam
Current	Linear Antenna Incident Wave
Magnetic Bias	Far Field Incident Wave
Plane Incident Wave	Near Field Incident Wave

General Commands Recognized by the Boundary/Excitations Module

GetExcitation
GetExcitationAssignment(2DExtractor)
GetExcitationsOfType
GetNumBoundaries
GetNumBoundariesOfType
GetNumExcitations
GetNumExcitationsOfType
GetPortExcitationsCount
ReassignBoundary
RenameBoundary
ReprioritizeBoundaries
SetDefaultBaseName

AutoIdentifyPorts

Use: Automatically assign ports and terminals in a terminal design.

Command: **HFSS>Excitations>Assign>Wave Port|Lumped Port**

Syntax:
```
AutoIdentifyPorts <FaceIDArray> <IsWavePort>,
<ReferenceConductorsArray> <BaseNameforCreatedPorts>
<UseConductorNamesAsBaseNameforTerminals>
```

Return Value: None.

Parameters:

`<FaceIDArray>`

Array("NAME:Faces", <FaceID>, <FaceID>, ...)

`<IsWavePort>`

Type: Boolean

true = waveport, false = lumped port

`<ReferenceConductorsArray>`

Array("NAME:ReferenceConductors", <ConductorName>, <ConductorName>, ...)

`<BaseNameforCreatedPorts>`

Type: <string>

<empty string> = default name for the wave or lumped port, <string> = base name to use for created ports

`<UseConductorNamesAsBaseNameforTerminals>`

Type: Boolean

true = use conductor names, false = use port object name as base name

Example:

```
Set oModule = oDesign.GetModule("BoundarySetup"
```

```
oModule.AutoIdentifyPorts Array("NAME:Faces", 52), true, _
Array("NAME:ReferenceConductors", "Conductor1") true
```

AutoIdentifyTerminals

Use: Automatically identify the terminals within the given ports.

Command: **HFSS>Excitations>Assign>Auto Assign Terminals**

Syntax: `AutoIdentifyTerminals <ReferenceConductorsArray>, <PortNames> <UseConductorNamesAsBaseNameforTerminals>`

Return Value: None

Parameters:

`<ReferenceConductors>`

Array("NAME:ReferenceConductors", <ConductorName>, <ConductorName>, ...)

`<portNames>`

List of names.

`<UseConductorNamesAsBaseNameforTerminals>`

Type: Boolean

true = use conductor names, false = use port object name as base name

Example:

```
Set oModule = oDesign.GetModule("BoundarySetup"
oModule.AutoIdentifyTerminals Array("NAME:ReferenceConductors", "Con-
ductor1"), "WavePort1" true
```

ChangeImpedanceMult

Use: Modifies the port impedance multiplier.

Command: **HFSS>Excitations>Edit Impedance Mult**

Syntax: `ChangeImpedanceMult <MultVal>`

Return Value: None

Parameters:

`<MultVal>`

Type: <value>

New value for the impedance multiplier.

Example:

```
oModule.ChangeImpedanceMult 0.5
```

DeleteAllBoundaries

Use: Deletes all boundaries.

Command: **HFSS>Boundaries>Delete All**

Syntax: `DeleteAllBoundaries`

Return Value: None

Example: `oModule.DeleteAllBoundaries`

For Q3D Extractor or 2D Extractor, the DeleteAllBoundaries command has the following details.

Use: Delete all the boundaries

Command: **Q3D Extractor>Boundary>Delete All** or **2D Extractor>Boundary>Delete All**

Syntax: `DeleteAllBoundaries`

Return Value: None

Example: `oModule.DeleteAllBoundaries`

DeleteAllExcitations

Use: Deletes all excitations.

Command: **HFSS>Excitations>Delete All**

Syntax: `DeleteAllExcitations`

Return Value: None

Example:

```
oModule.DeleteAllExcitations
```

For Q3D Extractor or 2D Extractor, the DeleteAllExcitations command has the following details.

Use: Deletes all excitations.

Command: **Q3D Extractor>Nets>Delete All** or **2D Extractor>Conductor>Delete All**

Syntax: `DeleteAllExcitations`

Return Value: None

Example: `oModule.DeleteAllExcitations`

DeleteBoundaries

Use: Deletes the specified boundaries and excitations.

Command: **Delete** command in the **List** dialog box. Click **HFSS>List** to open the **List** dialog box.

Syntax: `DeleteBoundaries <NameArray>`

Return Value: None

Parameters: `<NameArray>`

Type: Array of strings

An array of boundary names.

Example:

```
oModule.DeleteBoundaries Array("PerfE1", "WavePort1")
```

For Q3D Extractor, the DeleteBoundaries command has the following details.

Use: Deletes the specified boundaries and excitations.

Command: **Delete** command in the **List** dialog box. Click **Q3D Extractor** or **2D Extractor>List** to open the **List** dialog box. Or, use the **2D Extractor>Conductor>Delete** command.

Syntax: `DeleteBoundaries <NameArray>`

None

GetBoundaryAssignment

Use: Gets a list of face IDs associated with the given boundary or excitation assignment.

Syntax: `GetBoundaryAssignment(<BoundaryName>)`

Return Value: Returns integer array of face IDs.

Parameters: `<BoundaryName>`

Type: <string>

Previously defined boundary or excitation name.

Example:

```
list = oModule.GetBoundaryAssignment("Rad1")
```

GetBoundaries

Use: Gets boundary names for a project.

Syntax: `GetBoundaries()`

Return Value: Array of boundary names.

Parameters: `None`

Example:

```
bndinfo_array = oModule.GetBoundaries()
```

GetBoundariesOfType

Use: Gets boundary names of the given type.

Syntax: `GetBoundariesOfType(<BoundaryType>)`

Return Value: Array of boundary names of the given type.

Parameters: `<BoundaryType>`

Type:<string>

Name of legal boundary type.

For example: "Radiation".

Example:

```
bndname_array = oModule.GetBoundariesOfType("Perfect E")
```

GetDefaultBaseName

Use: Gets the default base name for boundaries for a project.

Syntax: `GetDefaultBaseName <BoundaryType>`

Return Value: String of boundary default base name.

Parameters: `<BoundaryType>`

Type:<string>

Name of legal boundary type.

For example: "Radiation".

Example:

```
bnddefault_BaseName = oModule.GetDefaultBaseName "Radiation"
```

GetExcitations

Use: Gets excitation port and terminal names for a model.

Syntax: `GetExcitations()`

Return Value: Pairs of strings. The first is the name of the excitation (e.g. "port1:1") and the second is its type ("Wave Port")

Parameters: `None`

Example:

```
excite_name_array = oModule.GetExcitations()
```

For Q3D Extractor the command details are similar:

Use: Return a listing of excitations

Command: none

Syntax: `GetExcitations`

Return Value: Variant array, excitation name paired with excitation type.

Example: `oModule.GetExcitations`

GetExcitationsOfType

Use: Gets excitation names of the given type.

Syntax: `GetExcitationsOfType(<ExcitationType>)`

Return Value: Array of excitation names of the given type.

Parameters: `<ExcitationType>`

Type: <string>

Name of legal excitation type.

For example: "Plane Incident Wave.

Example:

```
excite_name_array = _
oModule.GetExcitationsOfType("Wave Port")
```

GetNumBoundaries

Use: Gets the number of boundaries in a design.

Syntax: `GetNumBoundaries()`

Return Value: Integer count

Parameters: `None`

Example:

```
numbound = oModule.GetNumBoundaries()
```

GetNumBoundariesOfType

Use: Gets the number of boundaries of the given type.

Syntax: `GetNumBoundariesOfType(<BoundaryType>)`

Return Value: Integer count

Parameters: `<BoundaryType>`

Type: <string>

Example:

```
numbound = oModule.GetNumBoundariesOfType("Perfect E")
```

GetNumExcitations

Use: Gets the number of excitations in a design, including all defined modes and terminals of ports.

Syntax: `GetNumExcitations()`

Return Value: Integer count

Parameters: `None`

Example:

```
numexcite = oModule.GetNumExcitations()
```

For 2D Extractor the command details are as follows:

Use: Gets the number of excitations in a design, including all defined signal lines, non-ideal grounds, floating lines, reference ground and surface ground.

Syntax: `GetNumExcitations()`

Return Value: Integer count

Parameters: `None`

Example:

```
numexcite = oModule.GetNumExcitations()
```

GetExcitationAssignment (2D Extractor)

Use: Return the geometry assignment of an excitation.

Command: None.

Syntax: `GetExcitationAssignment "<name>"`

Return Value: `Variant array of the geometry`

Parameters: `<name>`

Type: String

The name of the excitation

Example:

```
oModule.GetExcitationAssignment "Net1"
```

GetNumExcitationsOfType

Use: Gets the number of excitations of the given type, including all defined modes and terminals of ports.

Syntax: `GetNumExcitationsOfType(<ExcitationType>)`

Return Value: Integer count

Parameters: `<ExcitationType>`

Type: <string>

Example:

```
numexcite = oModule.GetNumExcitationsOfType("Voltage")
```

GetPortExcitationCounts

Use: Gets all port names and corresponding number of modes/terminals for each port excitation.

Syntax: `GetPortExcitationCounts()`

Return Value: Array of port names (Type: <string>) and corresponding mode/terminal counts (Type: <integer>).

Parameters: `None`

Example:

```
portinfo = oModule.GetPortExcitationCounts()
```

ReassignBoundary

Use: Specifies a new geometry assignment for a boundary.

Command: **HFSS>Boundaries>Reassign** or **HFSS>Excitations>Reassign**

Syntax:
```
ReassignBoundary Array("Name:<BoundName>",
   "Objects:=", <AssignmentObjects>,
   "Faces:=", <AssignmentFaces>)
```

Return Value: None

Example:
```
oModule.ReassignBoundary Array("NAME:PerfE1",_
"Objects:=", Array("Box2", "Box3"),_
"Faces:=", Array(12, 11))
```

For Q3D Extractor, the command details are as follows:

Use: Specifies a new geometry assignment for a net/terminal.

Command: **Q3D Extractor>Nets>Reassign>Net**

Syntax:
```
ReassignBoundary Array("Name:<BoundName>",
   "Objects:=", <AssignmentObjects>,
   "Faces:=", <AssignmentFaces>)
```

Return Value: None

Example:
```
oModule.ReassignBoundary Array("NAME:Net1",_
   "Objects:=", Array("Box2", "Box3"),_
"Objects:=", Array("Box3", "Box4"))
```

For 2D Extractor, the command details are as follows:

Use: Specifies a new geometry assignment for a net/terminal.

Command: **2D Extractor>Conductor>Reassign**

Syntax:
```
ReassignBoundary Array("Name:<BoundName>",
   "Objects:=", <AssignmentObjects>,
   "Faces:=", <AssignmentFaces>)
```

Return Value: None

Example:
```
oModule.ReassignBoundary Array("NAME:Rectangle2",
   "Objects:=", Array("Rectangle3"))
```

RenameBoundary

Use: Renames a boundary or excitation.

Command: Right-click a boundary in the project tree, and then click **Rename** on the shortcut menu.

Syntax: `RenameBoundary <OldName>, <NewName>`

Return Value: None

Parameters: `<OldName>`

Type: <string>

`<NewName>`

Type: <string>

Example:

```
oModule.RenameBoundary "PerfE1"  "PerfE"
```

For Q3D Extractor, the RenameBoundary command details are as follows:

Use: Renames an excitation.

Command: Right-click a net/terminal in the project tree, and then click **Rename** on the shortcut menu.

Syntax: `RenameBoundary <OldName>, <NewName>`

Return Value: None

Parameters: `<OldName>`

Type: <string>

`<NewName>`

Type: <string>

Example: `oModule.RenameBoundary "Net1"  "Net2"`

Example: `oModule.RenameBoundary "Rectangle4"  "VCC"`

ReprioritizeBoundaries

Use: Specifies the order in which the boundaries and excitations are recognized by the solver. The first boundary in the list has the highest priority. Note: this command is only valid if all defined boundaries and excitations appear in the list. All ports must be listed before any other boundary type.

Command: **HFSS>Boundaries>Reprioritize**

Syntax: `ReprioritizeBoundaries <NewOrderArray>`

Return Value: None

Parameters: `<NewOrderArray>`

```
Array("NAME:NewOrder", <BoundName>, <BoundName>, ...)
```

Example:

```
oModule.RepriorititzeBoundaries Array("NAME:NewOrder", _
"Imped1", "PerfE1", "PerfH1")
```

SetDefaultBaseName

Use: Sets the default base name for boundaries for a project.

Syntax: `SetDefaultBaseName <BoundaryType>, <DefaultName>`

Return Value: String of boundary default base name.

Parameters:

`<BoundaryType>`

Type:<string>

Name of legal boundary type.

For example: "Radiation".

`<NewName>`

Type: <string>

Example:

```
bnddefault_BaseName = oModule.SetDefaultBaseName "Radiation",_
"RadBnd"
```

Script Commands for Creating and Modifying Boundaries

Following are script commands for creating and modifying boundaries that are recognized by the `"BoundarySetup"` module. In the following commands, all named data can be included or excluded as desired and may appear in any order.

AssignCurrent
AssignFiniteCond
AssignFloquet
AssignHalfSpace
AssignIERegion
AssignImpedance
AssignIncidentWave
AssignLayeredImp
AssignLumpedPort
AssignLumpedRLC
AssignMagneticBias
AssignMaster
AssignPerfectE
AssignPerfectH
AssignRadiation
AssignRadiation
AssignScreeningImpedance
AssignSymmetry
AssignTerminal
AssignVoltage
AssignWavePort
EditCurrent
EditDiffPairs
EditFiniteCond
EditHalfSpace
EditImpedance
EditIncidentWave
EditLayeredImpedance
EditMaster
EditPerfectE
EditPerfectH

EditLumpedPort
EditLumpedRLC
EditMagneticBias
EditRadiation
EditSlave
EditSymmetry
EditTerminal
EditVoltage
EditWavePort
SetTerminalReferenceImpedances
UnassignIERegions

AssignCurent

Use: Creates a current source.

Command: **HFSS>Excitations>Assign>Current**

Syntax:
```
AssignCurrent <CurrentArray>
```

Return Value: None

Parameters:
```
<CurrentArray>
   Array("NAME:<BoundName>",
      "Objects:=", <AssignmentObjects>,
      "Current:=", <value>,
      <DirectionArray>,
      "Faces:=", <AssignmentFaces>)

<DirectionArray>
   Array("NAME:Direction",
      "Start:=", <LineEndPoint>,
      "End:=", <LineEndPoint>)
```

Example:
```
oModule.AssignCurrent Array("NAME:Current1",_
   "Current:=", "1000mA",_
   Array("NAME:Direction",_
      "Start:=", Array(-0.4, 0.4, -1.6),_
      "End:=", Array(-0.4, 0.4, 0)), _
   "Faces:=", Array(12))
```

AssignFiniteCond

Use: Creates a finite conductivity boundary.

Command: **HFSS>Boundaries>Assign>Finite Conductivity**

Syntax: `AssignFiniteCond <FiniteCondArray>`

Return Value: None

Parameters:

```
<FiniteCondArray>
   Array("NAME:<BoundName>",
      "UseMaterial:=",<bool>,
      "Material:=", <string>,
      "Conductivity:=", <value>,
      "Permeability:=", <value>,
      "Roughness:=" <value>,
      "InfGroundPlane:=", <bool>,
      "Objects:=", <AssignmentObjects>,
      "Faces:=", <AssignmentFaces>
      Radius:=", "<value>", "Ratio:=", "<value>"))
```

`UseMaterial`

If `True`, provide `Material` parameter.

If `False`, provide `Conductivity` and `Permeability` parameters.

For Huray Roughness, use Radius and Ratio. For Groisse roughness model, use Roughness.

Example:

```
oModule.AssignFiniteCond Array("NAME:FiniteCond1",_
   "UseMaterial:=", false,_
   "Conductivity:=", "58000000",_
   "Permeability:=", "1",_
   "InfGroundPlane:=", false,_
   "Faces:=", Array(12))
```

Example:

```
oModule.AssignFiniteCond Array("NAME:FiniteCond1",_
   "UseMaterial:=", true, _
   "Material:=", "copper",_
   "InfGroundPlane:=", false,_
```

```
"Faces:=", Array(12)
Radius:=", "1um", "Ratio:=", "1"))
```

AssignFloquet

Use: Create a Floquet port.

Command: **HFSS>Excitations>Assign>Floquet**

Syntax: `AssignFloquetPort <FloquetPortArray>`

Return Value: None.

Parameters:

```
<FloquetPortArray>
  Array("NAME:<BoundName>",
    "Faces:=", <FaceIDArray>,
    <ModesArray>,
    "NumModes:=", <Int>,
    "RenormalizeAllTerminals:=", <Boolean>,
    "DoDeembed:=", <Boolean>,
    Array("NAME:Modes", Array("NAME:Mode1",
    "ModeNum:=", <Int>,
    "UseIntLine:=", <Boolean>),
      Array("NAME:Mode2", "ModeNum:=", <Int>,
      "UseIntLine:=", <Boolean>)),
    "ShowReporterFilter:=", <Boolean>,
    "UseScanAngles:=", <Boolean>,
    "Phi:=", "<numdeg>",
    "Theta:=", "<numdeg>",
    Array("NAME:LatticeAVector",
      "Start:=", Array("<num><units>", "num><units>",
      "<num><units>"),
      "End:=", Array("<num><units>", "num><units>",
      "<num><units>")),
    Array("NAME:LatticeBVector",
      "Start:=", Array("<num><units>", "num><units>",
      "<num><units>"),
      "End:=", Array("<num><units>", "num><units>",
      "<num><units>")),
    Array("NAME:ModesCalculator",
      "Frequency:=", "<Value>GHz",
      "FrequencyChanged:=", <Boolean>,
```

```
            "PhiStart:=", "<num>deg",
            "PhiStop:=", "<num>deg",
            "PhiStep:=", "<num>deg",
            "ThetaStart:=", "<num>deg",
            "ThetaStop:=", "<num>deg",
            "ThetaStep:=", "<num>deg"),
         Array("NAME:ModesList",
            Array("NAME:Mode",
            "ModeNumber:=", <ModeID>,
            "IndexM:=", <Index>,
            "IndexN:=", <Index>,
            "KC2:=", <value>,
            "PropagationState:=", "Propagating",
            "Attenuation:=", 0,
            "PolarizationState:=", "TE",
            "AffectsRefinement:=", <Boolean>),
         Array("NAME:Mode",
            "ModeNumber:=", <ModeID>,
            "IndexM:=", <Index>,
            "IndexN:=", <Index>,
            "KC2:=", <value>,
            "PropagationState:=", "<Propagating>",
            "Attenuation:=", <value>,
            "PolarizationState:=", "<TE or TM>",
            "AffectsRefinement:=", <Boolean>)))
```

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 13.0.0
' 1:54:11 PM  Jun 15, 2010
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
```

```
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project44")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AssignFloquetPort Array("NAME:FloquetPort1",
"Faces:=", Array(7),
"NumModes:=", 2,
"RenormalizeAllTerminals:=", true,
"DoDeembed:=", false,
Array("NAME:Modes", Array("NAME:Mode1", "ModeNum:=", 1,
"UseIntLine:=", false),
Array("NAME:Mode2", "ModeNum:=", 2, "UseIntLine:=", false)),
"ShowReporterFilter:=", false,
"UseScanAngles:=", true, "Phi:=", "0deg", "Theta:=", "0deg",
Array("NAME:LatticeAVector", "Start:=", Array("0mm", "0mm", "0.8mm"),
"End:=", Array( "0mm", "0.6mm", "0.8mm")),
Array("NAME:LatticeBVector", "Start:=", Array("0mm", "0mm", "0.8mm"),
"End:=", Array("0.8mm", "0mm", "0.8mm")),
Array("NAME:ModesCalculator", "Frequency:=", "1GHz",
"FrequencyChanged:=", false,
"PhiStart:=", "0deg", "PhiStop:=", "0deg", "PhiStep:=", "0deg", "The-
taStart:=", "0deg", "ThetaStop:=", "0deg", "ThetaStep:=", "0deg"),
Array("NAME:ModesList", Array("NAME:Mode", "ModeNumber:=", 1,
"IndexM:=", 0, "IndexN:=", 0, "KC2:=", 0,
"PropagationState:=", "Propagating",
"Attenuation:=", 0,
"PolarizationState:=", "TE",
"AffectsRefinement:=", false),
Array("NAME:Mode", "ModeNumber:=", 2,
"IndexM:=", 0, "IndexN:=", 0,
"KC2:=", 0,
"PropagationState:=", "Propagating",
```

```
"Attenuation:=", 0,
"PolarizationState:=", "TM", "AffectsRefinement:=", false)))
```

AssignHalfSpace

Use: Assign a Half Space boundary, dividing the background material at a specified Z axis point. You also assign a material, typically to the lower half.

Command: **Assign Half Space**

Syntax: `AssignHalfSpace Array("NAME:HalfSpacen", "ZLocation:=", "<intUnits>", "Material:=", "<string>")`

Return Value: None

Parameters: `"NAME:<stringN>"`

String

ZLocation

Z value and Units

Materials

<string> defining the material.

Example:

```
oModule.AssignHalfSpace Array("NAME:HalfSpace1", "ZLocation:=",
"2mm", "Material:=",  "water_sea")
```

AssignIERegion

Use: Assign an IE Region to a conductor contained within a FEBI Radiation boundary.

Command: **Assign IE Region**

Syntax: `AssignIERegion <"geometryName">`

Return Value: None

Parameters: `<GeometryName>`

Type: String

Name of the geometry assigned as an IE Region.

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 14.0.0
' 2:26:27 PM  Mar 07, 2011
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
```

```
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project58")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AssignIERegion "Box1"
```

AssignImpedance

Use: Creates an impedance boundary for an HFSS design.

Command: **HFSS>Boundaries>Assign>Impedance**

Syntax: `AssignImpedance <ImpedanceArray>`

Return Value: None

Parameters:

```
<ImpedanceArray>
   Array("NAME:<BoundName>",
      "Resistance:=", <value>,
      "Reactance:=", <value>,
      "InfGroundPlane:=", <bool>,
      "Objects:=", <AssignmentObjects>,
      "Faces:=", <AssignmentFaces>)
```

Example:

```
oModule.AssignImpedance Array("NAME:Imped1",_
   "Resistance:=", "50",_
   "Reactance:=", "50",_
   "InfGroundPlane:=", false,_
   "Faces:=", Array(12))
```

AssignIncidentWave

Use: Creates an incident wave excitation.

Command: **HFSS>Excitations>Assign>IncidentWave**

Syntax: `AssignIncidentWave <IncidentWaveArray>`

Return Value: None

Parameters:

```
<IncidentWaveArray>
   Array("NAME:<BoundName>",
      "IsCartesian:=",<bool>
      "EoX:=", <value>,
      "EoY:=", <value>,
      "EoZ:=", <value>,
      "kX:=", <value>,
      "kY:=", <value>,
      "kZ:=", <value>
      "PhiStart:=",<value>,
      "PhiStop:=", <value>,
      "PhiPoints:=", <int>,
      "ThetaStart:=", <value>,
      "ThetaStop:=", <value>,
      "ThetaPoints:=", <int>,
      "EoPhi:=", <value>,
      "EoTheta:=", <value>)
```

`IsCartesian`

If true, provide the `EoX`, `EoY`, `EoZ`, `kX`, `kY`, `kZ` parameters.

If false, provide the `PhiStart`, `PhiStop`, `PhiPoints`, `ThetaStart`, `ThetStop`, `ThetaPoints`, `EoPhi`, `EoTheta` parameters.

Example:

```
oModule.AssignIncidentWave Array("NAME:IncWave1",_
   "IsCartesian:=", true,_
   "EoX:=", "1", "EoY:=", "0", "EoZ:=", "0",_
   "kX:=", "0", "kY:=", "0", "kZ:=", "1")
```

Example:

```
oModule.AssignIncidentWave Array("NAME:IncWave2",_
   "IsCartesian:=", false,_
   "PhiStart:=","0deg",_
   "PhiStop:=", "90deg",_
   "PhiPoints:=", 2,_
   "ThetaStart:=", "0deg",_
   "ThetaStop:=", "180deg",_
```

```
   "ThetaPoints:=", 3, _
   "EoPhi:=", "1", "EoTheta:=", "0")
```

AssignLayeredImp

Use: Creates a layered impedance boundary.

Command: **HFSS>Boundaries>Assign>Layered Impedance**

Syntax: `AssignLayeredImp <LayeredImpArray>`

Return Value: None

Parameters:

```
<LayeredImpArray>
   Array("NAME:<BoundName>",
      "Frequency:=", <value>,
      "Roughness:=", <value>,
      "IsInternal:=", <bool>,
      <LayersArray>,
      "Objects:=", <AssignmentObjects>,
      "Faces:=", <AssignmentFaces>)

<LayersArray>
   Array("NAME:Layers",
      <OneLayerArray>, <OneLayerArray>, ...)

<OneLayerArray>
   Array("NAME:<LayerName>",
      "LayerType:=", <LayerType>,
      "Thickness:=",<value>,
      "Material:=", <string>)
```

`<LayerName>`
Type: <string>
Specifies the layer number, such as `"Layer1"` or `"Layer2"`

`<LayerType>`
Type: <string>
Should be specified for the last layer only.
Possible values: `"Infinite"`, `"PerfectE"`, or `"PerfectH"`

Thickness

Thickness of the layer. Should be specified for all layers except the last layer.

Material

Material assigned on the layer. For the last layer, do not specify a material if the LayerType is "PerfectE" or "PerfectH".

InfGroundPlane <boolean>

For HFSS designs, you can specify whether one layer is an infinite ground plane.

Example:

```
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AssignLayeredImp Array("NAME:Layered1",
    "Objects:=", Array("Rectangle1"),
                "Frequency:=", "10GHz",
                "Roughness:=", "0um",
                "IsInternal:=", false,
               Array("NAME:Layers",
               Array("NAME:Layer1",
                "Thickness:=",   "1um",
                "Material:=", "vacuum"),
               Array("NAME:Layer2",
                "LayerType:=", "Infinite",
                "Thickness:=",  "1um",
                "Material:=", "vacuum")),
    "InfGroundPlane:=", true)
```

AssignLumpedPort

Use: Creates a lumped port.

Command: **HFSS>Excitations>Assign>Lumped Port**

Syntax: AssignLumpedPort <LumpedPortArray>

Return Value: None

Parameters:

```
<LumpedPortArray>
   Array("NAME:<BoundName>",
      "Faces:=", <FaceIDArray>,
      "RenormalizeAllTerminals:=", <boolean>
```

```
                    "DoDeembed:="' <boolean"
                    <ModesArray>,
                    "TerminalIDList:=", <TerminalsArray>,
                    "FullResistance:=", <value>,
                    "FullReactance:=", <value>,
                    )
```

Example:

```
            oModule.AssignLumpedPort Array("NAME:LumpPort1",_
              Array("NAME:Modes",_
                 "Resistance:=", "50Ohm",_
                 "Reactance:=","0Ohm",_
                 Array("NAME:Mode1",_
                    "ModeNum:=",1,_
                    "UseIntLine:=", true,_
                    Array("NAME:IntLine",_
                       "Start:=", Array(-0.4, 0.4, -1.6),_
                       "End:=", Array(-0.4, 0.4, 0)),_
                       "CharImp:=", "Zpv")),_
                 "Faces:=", Array(11))
```

Example:

```
oModule.AssignLumpedPort Array("NAME:LumpPort1", _
"Faces:=", Array(52), "TerminalIDList:=", Array(), _
"FullResistance:=", "50ohm", "FullReactance:=", "0ohm")
```

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 14.0.0
' 2:18:20 PM  May 20, 2011
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
```

```
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("calib_modal_test")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AssignLumpedPort "lp2", Array("NAME:lp2",
"RenormalizeAllTerminals:=", true,
"DoDeembed:=", true,
Array("NAME:Modes", Array("NAME:Mode1",
"ModeNum:=", 1,
"UseIntLine:=", true,
Array("NAME:IntLine",
"Start:=", Array( "120mm", "50mm", "40mm"),
"End:=", Array("120mm", "50mm", "120mm")),
"CharImp:=", "Zpi")),
"ShowReporterFilter:=", false,
"ReporterFilter:=", Array(true), "FullResistance:=", "50ohm",
"FullReactance:=", "0ohm")
```

AssignLumpedRLC

Use: Creates a lumped RLC boundary.

Command: **HFSS>Boundaries>Assign>Lumped RLC**

Syntax: `AssignLumpedRLC <LumpedRLCArray>`

Return Value: None

Parameters:

```
<LumpedRLCArray>
   Array("NAME:<BoundName>",
      "UseResist:=",<bool>,
      "Resistance:=", <value>,
      "UseInduct:=", <bool>,
      "Inductance:=", <value>,
      "UseCap:=", <bool>,
      "Capacitance:=", <value>,
      <CurrentLineArray>,
      "Objects:=", <AssignmentObjects>,
      "Faces:=", <AssignmentFaces>)

<CurrentLineArray>
```

```
   Array("NAME:CurrentLine", _
      "Start:=", <LineEndPoint>,
      "End:=", <LineEndPoint>)
```

Example:

```
oModule.AssignLumpedRLC Array("NAME:LumpRLC1",_
   "UseResist:=", true,_
   "Resistance:=", "10Ohm",_
   "UseInduct:=", true,_
   "Inductance:=", "10nH",_
   "UseCap:=", true,_
   "Capacitance:=","10pF",_
   Array("NAME:CurrentLine", _
      "Start:=", Array(-0.4, -1.2, -1.6),_
      "End:=", Array(-0.4, -1.2, 0)),
   "Faces:=", Array(12))
```

AssignMagneticBias

Use: Creates a magnetic bias source.

Command: **HFSS>Excitations>Assign>Magnetic Bias**

Syntax: `AssignMagneticBias <MagneticBiasArray>`

Return Value: None

Parameters:

```
<MagneticBiasArray>
   Array("NAME:<BoundName>",
      "IsUniformBias:=", <bool>,
      "Bias:=", <value>,
      "XAngle:=", <value>,
      "YAngle:=", <value>,
      "ZAngle:=", <value>,
      "Project:=",<string>,
      "Objects:=", <AssignmentObjects>)
```

`IsUniformBias`

If true, supply the `Bias`, `XAngle`, `YAngle`, and `ZAngle` parameters.

If false, supply the `Project` parameter.

Example:

```
oModule.AssignMagneticBias Array("NAME:MagBias1",_
```

```
    "IsUniformBias:=", true,_
    "Bias:=", "1",_
    "XAngle:=", "10deg",_
    "YAngle:=", "10deg",_
    "ZAngle:=", "10deg",_
    "Objects:=", Array("Box2"))
```

Example:

```
oModule.AssignMagneticBias Array("NAME:MagBias2",_
    "IsUniformBias:=", false,_
    "Project:=","D:/Maxwell/testing/m3dfs.pjt",_
    "Objects:=", Array("Box2"))
```

AssignMaster

Use: Creates a master boundary.

Command: **HFSS>Boundaries>Assign>Master**

Syntax: `AssignMaster <MasterArray>`

Return Value: None

Parameters:

```
<MasterArray>
   Array("NAME:<BoundName>",
      <CoordSysArray>,
      "ReverseV:=", <bool>,
      "Faces:=", <AssignmentFaces>)

<CoordSysArray>
   Array("NAME:CoordSysVector",
      "Origin:=", <CoordSysPoint>,
      "UPos:=", <LineEndPoint>)
```

Example:

```
oModule.AssignMaster Array("NAME:Master1",_
   Array("NAME:CoordSysVector",_
      "Origin:=", Array(-1.4, -1.4, -0.8),_
      "UPos:=", Array(-1.4, -1.4, 0)),_
   "ReverseV:=", false,_
   "Faces:=", Array(12))
```

AssignPerfectE

Use: Creates a perfect E boundary.

Command: **HFSS>Boundaries>Assign>Perfect E**

Syntax:
```
AssignPerfectE <PerfectEArray>
```

Return Value: None

Parameters:
```
<PerfectEArray>
   Array("NAME:<BoundName>",
      "InfGroundPlane:=", <bool>,
      "Objects:=", <AssignmentObjects>,
      "Faces:=", <AssignmentFaces>)
```

Example:
```
oModule.AssignPerfectE Array("NAME:PerfE1",_
   "InfGroundPlane:=", false,_
   "Faces:=", Array(12))
```

AssignPerfectH

Use: Creates a perfect H boundary.

Command: **HFSS>Boundaries>Assign>PerfectH**

Syntax:
```
AssignPerfectH <PerfectHArray>
```

Return Value: None

Parameters:
```
<PerfectHArray>
   Array("Name:<BoundName>",
      "Objects:=", <AssignmentObjects>,
      "Faces:=", <AssignmentFaces>)
```

Example:
```
oModule.AssignPerfectH Array("NAME:PerfH1",_
   "Faces:=", Array(12))
```

AssignRadiation

Use: Creates a radiation boundary.

Command: **HFSS>Boundaries>Assign>Radiation**

Syntax:
```
AssignRadiation <RadiationArray>
```

Return Value: None

Parameters:
```
<RadiationArray>
   Array("NAME:<BoundName>",
```

```
                    "Objects:=", <AssignmentObjects>,
                    "Faces:=", <AssignmentFaces>
                    "IsIncidentField:=", <boolean>, _
                    "IsEnforcedHField:=", <boolean>, _
                    "IsEnforcedEField:=", <boolean>, _
                    "IsFssReference:=", <boolean>, _
                    "IsForPML:=", <boolean>, _
                    "UseAdaptiveIE:=", <boolean>, _
                    "IncludeInPostproc:=", <boolean>))
```

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 14.0.0
' 2:27:20 PM  Sep 13, 2011
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project59")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AssignRadiation Array("NAME:Rad1", _
"Objects:=", Array("Box1"), _
"IsIncidentField:=", false, _
"IsEnforcedField:=", false, _
"IsFssReference:=", false, _
"IsForPML:=", false, _
"UseAdaptiveIE:=", true, _
"IncludeInPostproc:=", true)
```

AssignScreeningImpedance

Use: Creates a screening impedance boundary.

Command: **HFSS>Boundaries>Assign>Screening Impedance**

Syntax:
```
AssignScreeningImpedance <ScreeningArray>
```

Return Value: None.

Parameters:
```
<ScreeningArray>
Array("NAME:<name>",
"Objects:=", Array( "<name>"),
   "IsAnisotropic:=", <Boolean>,
```
If true, you need to specify the coordinate system
```
   "CoordSystem:=", <integer or name>,
   "HasExternalLink:=", <Boolean>,
```
true or false. If False, specify XResistence and XReactance values. Also see the first example.
```
"XResistance:=", "<value>",
"XReactance:=", "<value>"
```
If true, then specify the external link array with the project and solution to use. Also see the second example.
```
Array("NAME:XLink",
   "Project:=", "<projectName>.aedt",
   "Design:=", "<DesignName>",
   "Soln:=", "Setup1 : LastAdaptive",
   Array("NAME:Params", "<variable>:=", "<value>"),
   "ForceSourceToSolve:=", <Boolean>,
   "PreservePartnerSoln:=", <Boolean>,
   "PathRelativeTo:=", "TargetProject"),
Array("NAME:YLink",
   "Project:=", "<projectName>.aedt",
   "Design:=", "HFSSDesign1",
   "Soln:=", "Setup1 : LastAdaptive",
   Array("NAME:Params", "<variable>:=", "<value>"),
   "ForceSourceToSolve:=", <Boolean>
   "PreservePartnerSoln:=", <Boolean>,
   "PathRelativeTo:=", "TargetProject"))
```

Example:
```
Dim oAnsoftApp
```

```
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project53")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AssignScreeningImpedance Array("NAME:Screening1",
"Faces:=", Array(12),
"IsAnisotropic:=", false,
"HasExternalLink:=", false,
"XResistance:=", "377", "XReactance:=", "0")
```

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 13.0.0
' 4:17:23 PM  Oct 29, 2010
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project53")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AssignScreeningImpedance Array("NAME:Screening1",
"Objects:=", Array( "Rectangle1"),
```

```
"IsAnisotropic:=", true,
"CoordSystem:=", 1,
"HasExternalLink:=", true,
Array("NAME:XLink",
"Project:=", "mydesign.aedt",
"Design:=", "HFSSDesign1",
"Soln:=", "Setup1 : LastAdaptive",
Array("NAME:Params", "bend_angle:=", "50deg"),
"ForceSourceToSolve:=", false,
"PreservePartnerSoln:=", false,
"PathRelativeTo:=", "TargetProject"),
Array("NAME:YLink", "Project:=", "mydesign.aedt",
"Design:=", "HFSSDesign1",
"Soln:=", "Setup1 : LastAdaptive",
Array("NAME:Params", "bend_angle:=", "50deg"),
"ForceSourceToSolve:=", true,
"PreservePartnerSoln:=", true,
"PathRelativeTo:=", "TargetProject"))
```

AssignSlave

Use: Creates a slave boundary.

Command: **HFSS>Boundaries>Assign>Slave**

Syntax: `AssignSlave <SlaveArray>`

Return Value: None

Parameters:

```
<SlaveArray>
   Array("NAME:<BoundName>",
      <CoordSysArray>,
      "ReverseV:=", <bool>,
      "Master:=", <string>,
      "UseScanAngles:=", <bool>,
      "Phi:=", <value>,
      "Theta:=", <value>,
      "Phase:=", <value>,
      "Objects:=", <AssignmentObjects>,
      "Faces:=", <AssignmentFaces>)

<UseScanAngles>
```

If `UseScanAngles` is `True`, then `Phi` and `Theta` should be specified.

If it is `False`, then `Phase` should be specified.

Example:

```
oModule.AssignSlave Array("NAME:Slave1",_
  Array("NAME:CoordSysVector", _
    "Origin:=", Array(-1, 0, 0.2),_
    "UPos:=", Array(-1, 0, 0)),_
  "ReverseV:=", false,_
  "Master:=", "Master1",_
  "UseScanAngles:=", true,_
  "Phi:=", "10deg",_
  "Theta:=", "0deg",_
  "Faces:=", Array(12))
```

Example:

```
oModule.AssignSlave Array("NAME:Slave2",_
  Array("NAME:CoordSysVector",_
    "Origin:=", Array(-1, 0, 0.2),_
    "UPos:=", Array(-2, 0, 0.2)),_
  "ReverseV:=", false,_
  "Master:=", "Master1",_
  "UseScanAngles:=",false,_
  "Phase:=", "10deg",_
  "Faces:=", Array(11))
```

AssignSymmetry

Use: Creates a symmetry boundary.

Command: **HFSS>Boundaries>Assign>Symmetry**

Syntax: `AssignSymmetry <SymmetryArray>`

Return Value: None

Parameters:

```
<SymmetryArray>
  Array("NAME:<BoundName>",
    "IsPerfectE:=", <bool>
    "Objects:=", <AssignmentObjects>,
    "Faces:=", <AssignmentFaces>)
```

Example:

```
oModule.AssignSymmetry Array("NAME:Sym1",_
  "IsPerfectE:=", true,_
```

```
"Faces:=", Array(12))
```

AssignTerminal

Use: Assigning terminals to a port.

Command: **HFSS>Excitations>Assign>Terminal**

Syntax: `AssignTerminal <TerminalArray>`

Return Value: None

Parameters:

`<TerminalArray>`

Array("NAME: <TerminalName>, "Edges:", <EdgeIDArray>, "ParentBndID":= , "<PortName>", "TeminalResistance:=", <value>)

`<TerminalName>`

Type: String

`<EdgeIDArray>`

Type: Array of strings

`<PortName>`

Type: String

Name of Port.

`<value>`

Type: string

Value and units for the resistance.

Example:

```
oModule.AssignTerminal Array("NAME:Rectangle1_T1", _
"Edges:=", Array(36), "ParentBndID:=",  _
  "WavePort1", "TerminalResistance:=", "50ohm")
```

For Q3D Extractor the command details are as follows:

Use: Assigns 1D and 2D terminals.

Command: **Q3D Extractor>Nets>AssignTerminals**

Syntax:

```
AssignTerminals <TerminalArray>
<TerminalArray>
   Array("NAME:AssignTerminals",
   Array("Name:SourceList", Array("Name:<SourceName>",
   "Net:=", <NetObject>, "Objects:=", <Assignment 2D/
   1D>)),
   Array("Name:SinkList", Array("Name:<SinkName>",
   "Net:=", <NetObject>, "Objects:=", <Assignment 2D/
```

```
    1D>)),
    "Name:DeleteList", <Name Array>)
```

Return Value: None

Example:

```
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AssignTerminals Array("NAME:AssignTerminals",
Array("NAME:SourceList", Array("NAME:Polyline2", "Objects:=", Array(
"Polyline2"), "ParentBndID:=", "Box1", "Net:=", "Box1"),
Array("NAME:Rectangle1", "Objects:=", Array(   "Rectangle1"),
"ParentBndID:=", "Box1", "Net:=", "Box1")), Array("NAME:SinkList"),
"DeleteList:=",  _ "")
```

AssignVoltage

Use: Creates a voltage source.

Command: **HFSS>Excitations>Assign>Voltage**

Syntax:

```
AssignVoltage <VoltageArray>
```

Return Value: None

Parameters:

```
<VoltageArray>
   Array("NAME:<BoundName>",
      "Voltage:=", <value>,
      <DirectionArray>,
      "Objects:=", <AssignmentObjects>,
      "Faces:=", <AssignmentFaces>)

<DirectionArray>
   Array("NAME:Direction",_
      "Start:=",<LineEndPoint>,
      "End:=", <LineEndPoint>)
```

Example:

```
oModule.AssignVoltage Array("NAME:Voltage1",_
   "Voltage:=", "1000mV",_
   Array("NAME:Direction",_
      "Start:=", Array(-0.4, -1.2, 0),_
      "End:=", Array(-1.4, -1.2, 0)),_
   "Faces:=", Array(7))
```

AssignWavePort

Use: Creates a wave port.

Command: **HFSS>Excitations>Assign>Wave Port**

Syntax: `AssignWavePort <WavePortArray>`

Return Value: None

Parameters:

```
<WavePortArray>
   Array("NAME:<BoundName>",
      "Faces:=", <FaceIDArray>,
      "NumModes:=", <int>,
      "PolarizeEField:=",<bool>,
      "DoDeembed:=", <bool>,
      "DeembedDist:=", <value>,
      "DoRenorm:=", <bool>,
      "RenormValue:=",<value>,
      <ModesArray>,
      "TerminalIDList:=", <TerminalsArray>
      )
```

`NumModes`

Number of modes for modal problems.

Number of terminals for terminal problems.

`<ModesArray>`

Specify for modal problems.

```
   Array("NAME:Modes",
      <OneModeArray>, <OneModeArray>, ...)
```

```
<OneModeArray>
   Array("NAME:<ModeName>",
      "ModeNum:=", <int>,
      "UseIntLine:=", <bool>,
      <IntLineArray>)
```

`<ModeName>`

Type: <string>

Name of the mode. Format is `"Mode<int>"`. For example `"Mode1"`.

```
<IntLineArray>
   Array("NAME:IntLine",
      "Start:=", <LineEndPoint>,
      "End:=", <LineEndPoint>,
      "CharImp:=", <string>)
```

`CharImp`

Characteristic impedance of the mode. Possible values are `"Zpi"`, `"Zpv"`, or `"Zvi"`

Example: Modal problem:

```
oModule.AssignWavePort Array("NAME:WavePort1",_
   "NumModes:=", 2,_
   "PolarizeEField:=",false,_
   "DoDeembed:=", true,_
   "DeembedDist:=", "10mil",_
   "DoRenorm:=", true,_
   "RenormValue:=","50Ohm",
   Array("NAME:Modes",_
      Array("NAME:Mode1",_
         "ModeNum:=", 1,_
         "UseIntLine:=", true,_
         Array("NAME:IntLine",_
            "Start:=", Array(-0.4, -1.2, 0),_
            "End:=", Array(-1.4, 0.4, 0)),_
            "CharImp:=", "Zpi"), _
      Array("NAME:Mode2",_
         "ModeNum:=", 2,_
         "UseIntLine:=", false)),_
   "Faces:=", Array(7))
```

Example: Terminal problem:

```
oModule.AssignWavePort Array("NAME:WavePort1",_
   "Faces:=", Array(11)
   "NumModes:=", 2,_
```

```
"PolarizeEField:=", false,_
"DoDeembed:=", false,
"TeminalIDList:=", Array()
)
```

EditCurrent

Use: Modifies a current source.

Command: Double-click the excitation in the project tree to modify its settings.

Syntax: `EditCurrent <BoundName> <CurrentArray>`

Return Value: None

EditDiffPairs

Use: Edits the properties of differential pairs defined from terminal excitations on wave ports.

Command: **HFSS>Excitations>Differential Pairs**

Syntax: `EditDiffPairs <DifferentialPairsArray>`

Return Value: None

Parameters:

```
<DifferentialPairsArray>
   Array("NAME:EditDiffPairs",
      <OneDiffPairArray>, <OneDiffPairArray>,...)

<OneDiffPairArray>
   Array("NAME:Pair1",_
      "PosBoundary:=", <string>,
      "NegBoundary:=", <string>,
      "CommonName:=", <string>,
      "CommonRefZ:=", <value>,
      "DiffName:=", <string>,
      "DiffRefZ:=", <value>,
      "IsActive:=", <boolean>)
```

`PosBoundary`

Name of the terminal to use as the positive terminal.

`NegBoundary`

Name of the terminal to use as the negative terminal.

`CommonName`

Name for the common mode.

`CommonRefZ`

Reference impedance for the common mode.

`DiffName`

Name for the differential mode.

`DiffRefZ`

Reference impedance for the differential mode.

Example:

```
oModule.EditDiffPairs Array("NAME:EditDiffPairs", Array("NAME:Pair1",
"PosBoundary:=",  _
  "Rectangle1_T1", "NegBoundary:=", "Rectangle2_T1", _
"CommonName:=", "Comm1", "CommonRefZ:=", "25ohm", _
"DiffName:=", "Diff1", "DiffRefZ:=", "100ohm", "IsActive:=", true))
```

EditFiniteCond

Use: Modifies a finite conductivity boundary.

Command: Double-click the boundary in the project tree to modify its settings.

Syntax: `EditFiniteCond <BoundName> <FiniteCondArray>`

Return Value: None

Parameters:

```
<FiniteCondArray>
   Array("NAME:<BoundName>",
      "UseMaterial:=",<bool>,
      "Material:=", <string>,
      "Conductivity:=", <value>,
      "Permeability:=", <value>,
      "Roughness:=" <value>,
      "InfGroundPlane:=", <bool>,
      "Objects:=", <AssignmentObjects>,
      "Faces:=", <AssignmentFaces>
```

```
Radius:=", "<value>", "Ratio:=", "<value>"))
```

`UseMaterial`

If `True`, provide `Material` parameter.

If `False`, provide `Conductivity` and `Permeability` parameters.

For Huray Roughness, use Radius and Ratio. For Groisse roughness model, use Roughness.

Parameters:

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 14.0.0
' 2:12:43 PM  May 20, 2011
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project56")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.EditFiniteCond "FiniteCond1", Array("NAME:FiniteCond1",
"UseMaterial:=", true,
"Material:=", "copper",
"UseThickness:=", false,
"Radius:=", "0.4um",
"Ratio:=", "2.9",
"InfGroundPlane:=", true)
```

EditHalfSpace

Use: Edit a Half Space boundary name, Z location, and or materials.

Command: **Edit Properties**

Syntax: `EditHalfSpace Array("NAME:HalfSpace`*n*`", "ZLocation:=", "<intUnits>", "Material:=", "<string>")`

Return Value: None

Parameters: `"NAME:<stringN>"`

String

ZLocation

Z value and Units

Materials

<string> defining the material.

Example:

```
oModule.EditHalfSpace "HalfSpace1", Array("NAME:HalfSpace1", "ZLoca-
tion:=", "3mm", "Material:=",  "tungsten")
```

EditImpedance

Use: Modifies an impedance boundary.

Command: Double-click the boundary in the project tree to modify its settings.

Syntax: `EditImpedance <BoundName> <ImpedanceArray>`

Return Value: None

EditIncidentWave

Use: Modifies an incident wave excitation.

Command: Double-click the excitation in the project tree to modify its settings.

Syntax: `EditIncidentWave <BoundName> <IncidentWaveArray>`

Return Value: None

Parameters:

Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Cube_RCS_00a")
```

```
Set oDesign = oProject.SetActiveDesign("PEC_d1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.EditIncidentWave "IncPWave1",
Array("NAME:IncPWave1", "IsCartesian:=", false,
"PhiStart:=", "$phi_inc",
"PhiStop:=", "$phi_inc",
"PhiPoints:=", 1,
"ThetaStart:=", "$theta_inc",
"ThetaStop:=", "$theta_inc",
"ThetaPoints:=", 1, "EoPhi:=", "$phi_pol",
"EoTheta:=", "$theta_pol",
"OriginX:=", "0mm",
"OriginY:=", "0mm",
"OriginZ:=", "0mm",
"TransientActive:=", 1,
"TimeProfile:=", "Broadband Pulse",
"HfssFrequency:=", "1GHz",
"MinFreq:=", "100MHz",
"MaxFreq:=", "1GHz",
"Delay:=", "0.2s",
"NumFreqsExtracted:=", 401,
"SweepMinFreq:=", "100MHz",
"SweepMaxFreq:=", "1GHz",
"IsPropagating:=", true,
"IsEvanescent:=", false, "IsEllipticallyPolarized:=", false)
```

EditLayeredImpedance

Use: Modifies a layered impedance boundary.

Command: Double-click the boundary in the project tree to modify its settings.

Syntax: `EditLayeredImp <BoundName> <LayeredImpArray>`

Return Value: None

EditMaster

Use: Modifies a master boundary.

Command: Double-click the boundary in the project tree to modify its settings.

Syntax: `Edit <BoundName> <MasterArray>`

Return Value: None

EditPerfectE

Use: Modifies a perfect E boundary.

Command: Double-click the boundary in the project tree to modify its settings.

Syntax: `EditPerfectE <BoundName>, <PerfectEArray>`

Return Value: None

EditPerfectH

Use: Modifies a perfect H boundary.

Command: Double-click the boundary in the project tree to modify its settings.

Syntax: `EditPerfectH <BoundName> <PerfectHArray>`

Return Value: None

EditLumpedPort

Use: Modifies a lumped port.

Command: Double-click the excitation in the project tree to modify its settings.

Syntax: `EditLumpedPort <BoundName> <LumpedPortArray>`

Parameters:

```
<LumpedPortArray>
   Array("NAME:<BoundName>",
      "Faces:=", <FaceIDArray>,
      "RenormalizeAllTerminals:=", <boolean>
      "DoDeembed:="’ <boolean"
      <ModesArray>,
      “TerminalIDList:=”, <TerminalsArray>,
      "FullResistance:=", <value>,
      "FullReactance:=", <value>,
      )
```

Return Value: None

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 14.0.0
' 2:18:20 PM  May 20, 2011
```

```
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("calib_modal_test")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.EditLumpedPort "lp2", Array("NAME:lp2", "RenormalizeAllTerminals:=", true, "DoDeem-
bed:=", _
  true, Array("NAME:Modes", Array("NAME:Mode1", "ModeNum:=", 1, "UseIntLine:=", true,
Array("NAME:IntLine", "Start:=", Array( _
  "120mm", "50mm", "40mm"), "End:=", Array("120mm", "50mm", "120mm")), "CharImp:=", _
  "Zpi")), "ShowReporterFilter:=", false, "ReporterFilter:=", Array(true), "FullResistance:=", _
  "50ohm", "FullReactance:=", "0ohm")
```

EditLumpedRLC

Use: Modifies a lumped RLC boundary.

Command: Double-click the boundary in the project tree to modify its settings.

Syntax: `EditLumpedRLC <BoundName> <LumpedRLCArray>`

Return Value: None

EditMagneticBias

Use: Modifies a magnetic bias excitation.

Command: Double-click the excitation in the project tree to modify its settings.

Syntax: `EditMagneticBias <BoundName> <MagneticBiasArray>`

Return Value: None

Parameters:

EditRadiation

Use: Modifies a radiation boundary.

Command: Double-click the boundary in the project tree to modify its settings.

Syntax: `EditRadiation <BoundName> <RadiationArray>`

Return Value: None

Parameters:

```
<RadiationArray>
   Array("NAME:<BoundName>",
      "Objects:=", <AssignmentObjects>,
      "Faces:=", <AssignmentFaces>
      "IsIncidentField:=", <boolean>, _
      "IsEnforcedHField:=", <boolean>, _
      "IsEnforcedEField:=", <boolean>, _
      "IsFssReference:=", <boolean>, _
      "IsForPML:=", <boolean>, _
      "UseAdaptiveIE:=", <boolean>, _
      "IncludeInPostproc:=", <boolean>))
```

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 14.0.0
' 2:34:08 PM  Sep 13, 2011
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project59")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.EditRadiation "Rad1", _
Array("NAME:Rad1", _
"IsIncidentField:=", true, _
"IsEnforcedField:=", false, _
```

```
"IsFssReference:=", false, _
"IsForPML:=", false, _
"UseAdaptiveIE:=", false, _
"IncludeInPostproc:=", true)
```

EditSlave

Use: Modifies a slave boundary.

Command: Double-click the boundary in the project tree to modify its settings.

Syntax: `EditSlave <BoundName> <SlaveArray>`

Return Value: None

EditSymmetry

Use: Modifies a symmetry boundary.

Command: Double-click the boundary in the project tree to modify its settings.

Syntax: `EditSymmetry <BoundName> <SymmetryArray>`

Return Value: None

EditTerminal

Use: Modifies properties of a terminal

Command: Edit Properties for a selected terminal

Syntax: `EditTerminal <TerminalArray>)`

Return Value: None

Parameters:

`<TerminalArray>`

Array("NAME: <TerminalName>", "ParentBndID:=", "<PortName>", "Terminal-Resistance:=," <value>")

`<TerminalName>`

Type:String

`<PortName>`

Type: String

`<value>`

Type: <string>

Value and units of resistance.

Example:

```
Set oModule = oDesign.GetModule("BoundarySetup")
```

```
oModule.EditTerminal "Rectangle2_T1", Array("NAME:Rectangle2_T1", _
"ParentBndID:=", "WavePort1", "TerminalResistance:=", "75ohm")
```

EditVoltage

Use: Modifies a voltage source.

Command: Double-click the excitation in the project tree to modify its settings.

Syntax: `EditVoltage <BoundName> <VoltageArray>`

Return Value: None

EditWavePort

Use: Modifies a wave port.

Command: Double-click the excitation in the project tree to modify its settings.

Syntax: `EditWavePort <BoundName> <WavePortArray>`

Return Value: None

Example:

SetTerminalReferenceImpedances

Use: To set the reference impedance for all terminals within a specified port.

Command: **HFSS>Excitations>Set Terminal Reference Impedances** or **HFSS-IE>Excitations>Set Terminal Reference Impedances**

Syntax: `SetTerminalReferenceImpedances <value>, <PortName>`

Return Value: None

Parameters:

`<value>`

Type: <string>

The value and units for the the impedance

`<PortName>`

Type: <string>

The name of the port.

Example:

```
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.SetTerminalReferenceImpedances "75ohm", "WavePort1"
```

UnassignIERegions

Use: Unassign one or more IE Regions assigned to conducting objects.

Command: **Unassign IE Regions**

Syntax: `UnassignIERegion Array (<"geometryName">)`

Return Value: None

Parameters: `<GeometryName>`

Type: String

Name of one or more geometries assigned as an IE Region.

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 14.0.0
' 2:51:43 PM  Mar 07, 2011
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project58")
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.UnassignIERegions Array("Box1")
```

Script Commands for Creating and Modifying Boundaries in HFSS-IE

AssignAperature [HFSS-IE]

Use: Creates an aperature boundary on a sheet in an HFIE design.

Command: **HFSS-IE>Boundary>Assign>Aperature...**

Syntax:
```
AssignAperture Array("NAME:<boundName>", "Objects:=",
Array("<sheetName>"))
```

Parameters:

`<boundName>`

Type: <string>

Boundary name.

`<sheetName>`

Type: <string>

Name of the sheet object to which you assign the boundary

Return Value: None

Example:

```
Set oProject = oDesktop.SetActiveProject("Project15")
Set oDesign = oProject.SetActiveDesign("HFIEDesign1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AssignAperture Array("NAME:Aperture1", _
"Objects:=", Array("Rectangle1"))
```

AssignFiniteCond [HFSS-IE]

Use: Creates a finite conductivity boundary.

Command: **HFSS-IE>Boundaries>Assign>Finite Conductivity**

Syntax:
```
AssignFiniteCond <FiniteCondArray>
```

Return Value: None

Parameters:
```
<FiniteCondArray>
   Array("NAME:<BoundName>",
      "Objects:=", <AssignmentObjects>,
      "UseMaterial:=",<bool>,
```

```
"Material:=", <string>,
"Conductivity:=", <value>,
"Permeability:=", <value>,
"InfGroundPlane:=", <bool>,
"Faces:=", <AssignmentFaces>)
```

`UseMaterial`

If `True`, provide `Material` parameter.

If `False`, provide `Conductivity` and `Permeability` parameters.

Example:

```
oModule.AssignFiniteCond Array("NAME:FiniteCond1",_
  "Objects:=", Array("Rectangle2"), _
  "UseMaterial:=", false, _
  "Conductivity:=", "58000000", _
  "Permeability:=", "1", _
  "Roughness:=", "0um", _
  "UseThickness:=", false)
```

AssignHalfSpace [HFSS-IE]

Use: Creates a Half Space boundary for an HFSS-IE design.

Command: **HFSS-IE>Boundaries>Assign>Half Space**

Syntax: `AssignHalfSpace Array("NAME:HalfSpacen", "ZLocation:=, <Value><Units>", "Material:=", <string>")`

Return Value: None

Parameters: `Zlocation`

Z Coordinate and units for the definition of the half space boundary.

`Material`

String identifying the material for the lower half of the space.

Example:

```
' ----------------------------------------------
' Script Recorded by ANSYS Electronics Desktop Version 2015.0.0
' 14:20:17  Aug 25, 2014
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
```

```
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project2")
Set oDesign = oProject.SetActiveDesign("IEDesign1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AssignHalfSpace Array("NAME:HalfSpace1", "ZLocation:=",
"0mm", "Material:=", "water_sea")
```

AssignImpedance [HFSS-IE]

Use: Creates an impedance boundary for an HFSS-IE design.

Command: **HFSS-IE>Boundaries>Assign>Impedance**

Syntax: `AssignImpedance <ImpedanceArray>`

Return Value: None

Parameters:

```
<ImpedanceArray>
   Array("NAME:<BoundName>",
      "Resistance:=", <value>,
      "Reactance:=", <value>,
      "Objects:=", <AssignmentObjects>,
      "Faces:=", <AssignmentFaces>)
```

Example:

```
oModule.AssignImpedance Array("NAME:Imped1",_
   "Resistance:=", "50",_
   "Reactance:=", "50",_
   "Faces:=", Array(12))
```

AssignInfiniteGroundPlane [HFSS-IE]

Use: Creates an infinite ground plane in HFIE.

Command: **HFSS-IE>Boundaries>Assign>Infinite Ground Plane**

Syntax: `AssignInfGndPlane <Array>`

Return Value: None

Parameters:

`<Array>`

"NAME:"<InfGndPlaneName>

"ZLocation:=", <value>
"Roughness:=", <value>
"Material:=", "<name>"

Example:

```
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AssignInfGndPlane Array("NAME:InfGndPlane1", _
"ZLocation:=", "0mm", _
"Roughness:=", "0mm", _
"Material:=", "Copper")
```

AssignLumpedPort [HFSS-IE]

Use: Creates a lumped port.

Command: **HFSS>Excitations>Assign>Lumped Port**

Syntax: `AutoIdentifyPorts <LumpedPortArray>`

Return Value: None

Parameters: `Array`

Example:

```
Set oDesign = oProject.SetActiveDesign("HFSSIEDesign1")
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AutoIdentifyPorts Array("NAME:Faces", 12), _
Array("NAME:ReferenceConductors", "Box1"), _
"LumpPort1", true
```

Script Commands for Creating and Modifying PMLs

Following are script commands for creating and modifiying PMLs that are recognized by the `"BoundarySetup"` module.

The **PML Setup** wizard allows you to set up one or more PMLs in the model. There is not a single 'Create PML' or 'Edit PML' command that represents the work performed by the **PML Setup** wizard. Instead, a series of geometry and material commands are executed. As a result, when a script is being recorded, a series of geometry and material creation commands is what is actually recorded in the script for a PML setup. This is followed by a script command stating that PMLs have been set up or modified.

CreatePML

ModifyPMLGroup

PMLGroupCreated

PMLGroupModified

RecalculatePMLMaterial

CreatePML

Use: Command to create a new PML group from the script. This is equivalent to creating a new PML group in the user interface.

Command: None

Syntax: For manually created PMLs:

```
CreatePML Array("UserDrawnGroup:=", true,
   "PMLObj:=", <string>,
   "BaseObj:=", <string>,
   "Thickness:=", <value>,
   "Orientation:=", <string>,
   "RadDist:=", <value>,
   "UseFreq:=", <bool>,
   "MinFreq:=", <value>,
   "MinBeta:=", <double>)
   "RadIncidentField:=", <bool>
   "RadFssReference:=", <bool>
```

For automatically created PMLs:

```
CreatePML Array("UserDrawnGroup:=", false,
   "PMLFaces:=", <AssignmentFaces>,
   "CreateJoiningObjs:=", <bool>,
   "Thickness:=", <value>,
```

```
"RadDist:=", <value>,
"UseFreq:=", <bool>,
"MinFreq:=", <value>,
"MinBeta:=", <double>)
"RadIncidentField:=", <bool>
"RadFssReference:=", <bool>
```

Return Value: None

Parameters:

`PMLObj`

Name of the object to use as the PML cover.

`BaseObj`

Name of the base object touching the PML cover object.

`Orientation`

String representing the orientation of the PML.

Possible values are: `"XAxis"`, `"YAxis"`, and `"ZAxis"`

`UseFreq`

If true, provide the `MinFreq` parameter.

If false, provide the `MinBeta` parameter.

Example:

```
oModule.CreatePML Array("UserDrawnGroup:=", false,_
  "PMLFaces:=", Array(120), "CreateJoiningObjs:=",_
  true,_
  "Thickness:=", "0.33mm", "RadDist:=", "1.6mm",_
  "UseFreq:=", true, "MinFreq:=", "1GHz")
```

Example:

```
oModule.CreatePML Array("UserDrawnGroup:=", true,_
  "PMLObj:=", "Box1", "BaseObj:=", "Box2", _
  "Thickness:=", "0.3mm", "Orientation:=", "ZAxis", _
  "RadDist:=", "1.6mm", "UseFreq:=", false, _
  "MinBeta:=", "2")
```

ModifyPMLGroup

Use: Command to modify a PML group. Note: This is the scripting equivalent to clicking **Update** in the **PML Setup** wizard. This does not actually modify the materials. It only modifies the data stored by the **PML Setup** wizard.

Command: None

Syntax:

```
ModifyPMLGroup Array("NAME:<GroupName>",
   "RadDist:=", <value>,
   "UseFreq:=", <bool>,
   "MinFreq:=", <value>,
   "MinBeta:=", <double>)
```

Return Value: None

Parameters:

`<GroupName>`

Name of the PML group to modify.

`UseFreq`

If true, provide the `MinFreq` argument.

If false, provide the `MinBeta` argument.

Example:

```
oModule.ModifyPMLGroup Array("NAME:PMLGroup1",
   "RadDist:=", "1.166666667mm",
   "UseFreq:=", false, "MinBeta:=", 2)
```

PMLGroupCreated

Use: Command added by HFSS after a PML has been created. It is not responsible for creating the PML objects and materials. It just contains the information needed by the **PML Setup** wizard for future modification of the PML. This script command is not intended to be modified by you. Removing this command from the script will prevent future modification of the PML through the user interface after the script is played back.

Command: **HFSS>Boundaries>Assign>PML Setup Wizard**

Syntax: `PMLGroupCreated <args>`

Return Value: None

PMLGroupModified

Use: Command added by HFSS after a PML's parameters are modified. This updates the **PML Setup** wizard's data. This script command is not intended to be modified by you. Removing this command from the script will prevent future modification of the PML through the user interface after the script is played back.

Command: Modify existing PML in the **PML Setup** wizard.

Syntax: `PMLGroupModified <args>`

Return Value: None

RecalculatePMLMaterials

Use: Scripting equivalent to clicking **Recalculate Materials** in the **PML Setup** wizard. This will update the PML materials to match the current state of the **PML Setup** wizard data.

Command: None

Syntax: `RecalculatePMLMaterials`

Return Value: None

Example:

```
oModule.RecalculatePMLMaterials
```

Script Commands for Creating and Modifying Boundaries in 2D Extractor

Following are script commands for creating and modifying boundaries that are recognized by the `"BoundarySetup"` module. In the following commands, all named data can be included/excluded as desired and may appear in any order.

AutoAssignSignals
AssignSingleSignalLine
AssignSingleNonIdealGround
AssignSingleReferenceGround
AssignSingleSurfaceGround
AssignSingleFloatingLine
AssignMultiFloatingLine
AssignMultiSignalLine
AssignMultiNonIdealGround

AutoAssignSignals

Use: Automatically assigns conducting objects to be "Signal" conductor.

Command: **2D Extractor>Conductor>Auto Assign Signals**

Syntax: `AutoAssignSignals`

Return Value: None

Command: oModule.AutoAssignSignals

AssignSingleSignalLine

Use: Assigns a single "Signal Line" on an object.

Command: **2D Extractor>Conductor>Assign>Signal Line**

Syntax: `AssignSingleSignalLine <AssignmentParameters>`

Return Value: None

Parameters:

```
<AssignmentParameters>
   Array("NAME:<SourceName>", "Objects:=", <AssignmentObjects>)
   "SolveOptions:=<SolveOptions>", "Thickness:=<Thickness>")
   <SourceName>
      Type:<String>
      Name of the conductor
   <AssignmentObjects>
      Type:<String>
```

```
    Name of the object or body in geometry.
  <SolveOptions>
    Type: <String>
    Value: one of the following - "SolveInside", "Solve-
    OnBoundary", "Automatic".
  <Thickness>
    Type: String
    Value: a double value with units of length.
```

Example:

```
oModule.AssignSingleSignalLine Array("NAME:Rectangle1",
"Objects:=", Array( _ "Rectangle1"), "SolveOption:=",
"Boundary", "Thickness:=", "0.171428571428571mm"))
```

AssignSingleNonIdealGround

Use: Assigns a single "NonIdealGround" on an object.

Command: **2D Extractor>Conductor>Assign>Non Ideal Ground**

Syntax:

```
AssignSingleNonIdealGround <AssignmentParameters>
```

Return Value: None

Parameters:

```
<AssignmentParameters>
  Array("NAME:<SourceName>", "Objects:=", <AssignmentOb-
  jects>)
  "SolveOptions:=<SolveOptions>", "Thickness:=<Thick-
  ness>")
  <SourceName>
    Type:<String>
    Name of the conductor
  <AssignmentObjects>
    Type:<String>
    Name of the object or body in geometry.
  <SolveOptions>
    Type: <String>
    Value: one of the following - "SolveInside", "Solve-
    OnBoundary", "Automatic".
  <Thickness>
    Type: String
    Value: a double value with units of length.
```

Example:

```
oModule.AssignSingleNonIdealGround
Array("NAME:Rectangle1", "Objects:=", Array( _
"Rectangle1"), "SolveOption:=", "Boundary",
"Thickness:=", "0.171428571428571mm")
```

AssignSingleReferenceGround

Use: Assigns a single "Reference Ground" on an object.

Command: **2D Extractor>Conductor>Assign>Reference Ground**

Syntax: `AssignSingleReferenceGround <AssignmentParameters>`

Return Value: None

Parameters:

```
<AssignmentParameters>
  Array("NAME:<SourceName>", "Objects:=", <AssignmentObjects>)
  "SolveOptions:=<SolveOptions>", "Thickness:=<Thickness>")
  <SourceName>
    Type:<String>
    Name of the conductor
  <AssignmentObjects>
    Type:<String>
    Name of the object or body in geometry.
  <SolveOptions>
    Type: <String>
    Value: one of the following - "SolveInside", "SolveOnBoundary", "Automatic".
  <Thickness>
    Type: String
    Value: a double value with units of length.
```

Example:

```
oModule.AssignSingleReferenceGround
Array("NAME:Rectangle1", "Objects:=", Array( _
"Rectangle1"), "SolveOption:=", "Boundary",
"Thickness:=", "0.171428571428571mm")
```

AssignSingleSurfaceGround

Use: Assigns a single "SurfaceGround" on an object.

Command: **2D Extractor>Conductor>Assign>Surface Ground**

Syntax: `AssignSingleSurfaceGround <AssignmentParameters>`

Return Value: None

Parameters:

```
<AssignmentParameters>
  Array("NAME:<SourceName>", "Objects:=", <AssignmentObjects>)
  "SolveOptions:=<SolveOptions>", "Thickness:=<Thickness>")
  <SourceName>
    Type:<String>
    Name of the conductor
  <AssignmentObjects>
    Type:<String>
    Name of the object or body in geometry.
  <SolveOptions>
    Type: <String>
    Value: one of the following - "SolveInside", "SolveOnBoundary", "Automatic".
  <Thickness>
    Type: String
    Value: a double value with units of length.
```

Example:

```
oModule.AssignSingleSurfaceGround
Array("NAME:Rectangle1", "Objects:=", Array( _
"Rectangle1"), "SolveOption:=", "Boundary",
"Thickness:=", "0.171428571428571mm")
```

AssignSingleFloatingLine

Use: Assigns a single "FloatingLine" on an object.

Command: **2D Extractor>Conductor>Assign>Floating Line**

Syntax: `AssignSingleFloatingLine <AssignmentParameters>`

Return Value: None

Parameters:

```
<AssignmentParameters>
  Array("NAME:<SourceName>", "Objects:=", <AssignmentObjects>)
  "SolveOptions:=<SolveOptions>", "Thickness:=<Thickness>")
  <SourceName>
    Type:<String>
```

```
    Name of the conductor
  <AssignmentObjects>
    Type:<String>
    Name of the object or body in geometry.
  <SolveOptions>
    Type: <String>
    Value: one of the following - "SolveInside", "Solve-
    OnBoundary", "Automatic".
  <Thickness>
    Type: String
    Value: a double value with units of length.
```

Example:

```
oModule.AssignSingleFloatingLine Array("NAME:Rectangle1",
"Objects:=", Array( _ "Rectangle1"), "SolveOption:=",
"Boundary", "Thickness:=", "0.171428571428571mm")
```

AssignMultiFloatingLine

Use: Assigns multiple "Floating Line" on selected objects (each per object).

Command: **2D Extractor>Conductor>Assign>Floating Line**

Syntax:

```
AssignMultiFloatingLine <MultiAssignmentParameters>
```

Return Value: None

Parameters:

```
<MultiAssignmentParameters>
     Array(<ConductorName>,<ConductorName>..), <Assign-
     mentParameters>
<ConductorName>
  Type:<String>
  Name of the assigned conductors.
<AssignmentParameters>
  Array("NAME:<SourceName>", "Objects:=", <AssignmentOb-
  jects>)
  "SolveOptions:=<SolveOptions>", "Thickness:=<Thick-
  ness>")
  <SourceName>
    Type:<String>
    Name of the conductor
  <AssignmentObjects>
    Type:<String>
```

```
    Name of the object or body in geometry.
  <SolveOptions>
    Type: <String>
    Value: one of the following - "SolveInside", "Solve-
    OnBoundary", "Automatic".
  <Thickness>
    Type: String
    Value: a double value with units of length.
```

Example:

```
oModule.AssignMultiSignalLine Array("Rectangle3",
"Rectangle4"), Array("NAME:Rectangle1", "Objects:=",
Array( _ "Rectangle1"), "SolveOption:=", "Boundary",
"Thickness:=", "0.171428571428571mm")
```

AssignMultiSignalLine

Use: Assigns multiple "Signal Line" on selected objects (each per object).

Command: **2D Extractor>Conductor>Assign>Signal Line**

Syntax: `AssignMultiSignalLine <MultiAssignmentParameters>`

Return Value: None

Parameters:

```
<MultiAssignmentParameters>
    Array(<ConductorName>,<ConductorName>..), <Assign-
    mentParameters>
<ConductorName>
  Type:<String>
  Name of the assigned conductors.
<AssignmentParameters>
  Array("NAME:<SourceName>", "Objects:=", <AssignmentOb-
  jects>)
  "SolveOptions:=<SolveOptions>", "Thickness:=<Thick-
  ness>")
  <SourceName>
    Type:<String>
    Name of the conductor
  <AssignmentObjects>
    Type:<String>
    Name of the object or body in geometry.
  <SolveOptions>
```

```
    Type: <String>
    Value: one of the following - "SolveInside", "Solve-
    OnBoundary", "Automatic".
  <Thickness>
    Type: String
    Value: a double value with units of length.
```

Example:

```
oModule.AssignMultiFloatingLine Array("Rectangle3",
"Rectangle4"), Array("NAME:Rectangle1", "Objects:=",
Array( _ "Rectangle1"), "SolveOption:=", "Boundary",
"Thickness:=", "0.171428571428571mm")
```

AssignMultiNonIdealGround

Use: Assigns multiple "Non Ideal Ground" on selected objects (each per object).

Command: **2D Extractor>Conductor>Assign>Non Ideal Ground**

Syntax:

```
AssignMultiNonIdealGround <MultiAssignmentParameters>
```

Return Value: None

Parameters:

```
<MultiAssignmentParameters>
    Array(<ConductorName>,<ConductorName>..), <Assign-
    mentParameters>
<ConductorName>
  Type:<String>
  Name of the assigned conductors.
<AssignmentParameters>
  Array("NAME:<SourceName>", "Objects:=", <AssignmentOb-
  jects>)
  "SolveOptions:=<SolveOptions>", "Thickness:=<Thick-
  ness>")
  <SourceName>
    Type:<String>
    Name of the conductor
  <AssignmentObjects>
    Type:<String>
    Name of the object or body in geometry.
  <SolveOptions>
    Type: <String>
    Value: one of the following - "SolveInside", "Solve-
    OnBoundary", "Automatic".
```

```
<Thickness>
  Type: String
  Value: a double value with units of length.
```

Example:

```
oModule.AssignMultiNonIdealGround Array("Rectangle3",
"Rectangle4"), Array("NAME:Rectangle1", "Objects:=",
Array( _ "Rectangle1"), "SolveOption:=", "Boundary",
"Thickness:=", "0.171428571428571mm")
```

ToggleConductor

Use: Toggles between one conductor to other.

Command: Click a conductor in the project tree, and then click **Toggle** on the shortcut menu.

Syntax:

```
ToggleConductor <TerminalName>, <NewType>
```

Return Value: None

Parameters:

```
<ConductorName>
  Type: <string>
  Conductor name that is selected.
<NewType>
  Type: <String>
  New type of conductor to be toggled.
```

Example:

```
oModule.ToggleConductor "Rectangle5", "ReferenceGround"
```

EditSignalLine

Use: Edits the properties of a signal line conductor.

Command: Double-click a conductor.

Syntax:

```
EditSignalLine <ConductorName> , <ConductorParameters>
```

Return Value: None

Parameters:

```
<ConductorName>
  Type: <string>
  Conductor name that is selected.
<ConductorParameters>
  Array(("Name:=<NewName>", "SolveOptions:=<SolveOp-
  tions>", "Thickness:=<Thickness>"))
  <SolveOptions>
    Type: <String>
    Value: one of the following, "SolveInside", "SolveO-
```

```
    nBoundary", "Automatic"
  <Thickness>
    Type: String
    Value: a double value with units of length.
  <NewName>
    Type: <String>
    New name of conductor to be toggled.
```

Example:

```
oModule.EditSignalLine "Rectangle6",
Array("NAME:Rectangle6_1", "SolveOption:=", "Automatic",
"Thickness:=", "0.179464285714286mm")
```

EditNonIdealGround

Use: Edits the properties of a non ideal ground conductor.

Command: Double-click a conductor.

Syntax:

```
EditNonIdealGround <ConductorName> ,
<ConductorParameters>
```

Return Value: None

Parameters:

```
<ConductorName>
  Type: <string>
  Conductor name that is selected.
<ConductorParameters>
  Array(("Name:=<NewName>", "SolveOptions:=<SolveOp-
  tions>", "Thickness:=<Thickness>"))
  <SolveOptions>
    Type: <String>
    Value: one of the following, "SolveInside", "SolveO-
    nBoundary", "Automatic"
  <Thickness>
    Type: String
    Value: a double value with units of length.
  <NewName>
    Type: <String>
    New name of conductor to be toggled.
```

Example:

```
oModule.EditNonIdealGround "Rectangle6",
Array("NAME:Rectangle6_1", "SolveOption:=", "Automatic",
"Thickness:=", "0.179464285714286mm")
```

EditReferenceGround

Use: Edits the properties of a reference ground conductor.

Command: Double-click a conductor.

Syntax:
```
EditReferenceGround <ConductorName> ,
<ConductorParameters>
```

Return Value: None

Parameters:
```
<ConductorName>
   Type: <string>
   Conductor name that is selected.
<ConductorParameters>
   Array(("Name:=<NewName>", "SolveOptions:=<SolveOp-
   tions>", "Thickness:=<Thickness>"))
   <SolveOptions>
      Type: <String>
      Value: one of the following, "SolveInside", "SolveO-
      nBoundary", "Automatic"
   <Thickness>
      Type: String
      Value: a double value with units of length.
   <NewName>
      Type: <String>
      New name of conductor to be toggled.
```

Example:
```
oModule.EditReferenceGround "Rectangle6",
Array("NAME:Rectangle6_1", "SolveOption:=", "Automatic",
"Thickness:=", "0.179464285714286mm")
```

EditSurfaceGround

Use: Edits the properties of a surface ground conductor.

Command: Double-click a conductor.

Syntax:
```
EditSurfaceGround <ConductorName> , <ConductorParameters>
```

Return Value: None

Parameters:
```
<ConductorName>
   Type: <string>
   Conductor name that is selected.
<ConductorParameters>
```

```
Array(("Name:=<NewName>", "SolveOptions:=<SolveOp-
tions>", "Thickness:=<Thickness>"))
<SolveOptions>
  Type: <String>
  Value: one of the following, "SolveInside", "SolveO-
  nBoundary", "Automatic"
<Thickness>
  Type: String
  Value: a double value with units of length.
<NewName>
  Type: <String>
  New name of conductor to be toggled.
```

Example:

```
oModule.EditSurfaceGround "Rectangle6",
Array("NAME:Rectangle6_1", "SolveOption:=", "Automatic",
"Thickness:=", "0.179464285714286mm")
```

EditFloatingLine

Use: Edits the properties of a floating line conductor.

Command: Double-click a conductor.

Syntax:

```
EditFloatingLine <ConductorName> , <ConductorParameters>
```

Return Value: None

Parameters:

```
<ConductorName>
  Type: <string>
  Conductor name that is selected.
<ConductorParameters>
  Array(("Name:=<NewName>", "SolveOptions:=<SolveOp-
  tions>", "Thickness:=<Thickness>"))
  <SolveOptions>
    Type: <String>
    Value: one of the following, "SolveInside", "SolveO-
    nBoundary", "Automatic"
  <Thickness>
    Type: String
    Value: a double value with units of length.
  <NewName>
    Type: <String>
```

```
                    New name of conductor to be toggled.
```

Example:

```
oModule.EditFloatingLine "Rectangle6",
Array("NAME:Rectangle6_1") "SolveOption:=", "Automatic",
"Thickness:=", "0.179464285714286mm")
```

GetNumExcitations (2D Extractor)

Use: Gets the number of excitations in a design, including all defined signal lines, non-ideal grounds, floating lines, reference ground and surface ground.

Syntax:

```
GetNumExcitations()
```

Return Value: Integer count

Parameters:

```
None
```

Example:

```
numexcite = oModule.GetNumExcitations()
```

GetExcitationAssignment (2D Extractor)

Use: Return the geometry assignment of an excitation.

Command: None.

Syntax:

```
GetExcitationAssignment "<name>"
```

Return Value:

```
Variant array of the geometry
```

Parameters:

```
<name>
```

Type: String

The name of the excitation

Example:

```
oModule.GetExcitationAssignment "Net1"
```

GetExcitations (2D Extractor)

Use: Return a listing of excitations

Command: none

Syntax:

```
GetExcitations
```

Return Value: Variant array, excitation name paired with excitation type.

Example:

```
oModule.GetExcitations
```

SetConductivityThreshold (2D Extractor)

Use: Sets the material thresholds for conductivity.

Command: **2D Extractor>Conductor>Set Material Thresholds**

Syntax:

```
SetConductivityThresholds
```

Return Value: None

Parameters:

```
SetConductivityThreshold <InsulatorThreshold>,
<PerfectConductorThreshold>
<InsulatorThreshold>
   Type:<double>
   Threshold for insulator/conductor.
<PerfectConductorThreshold>
   Type:<double>
   Threshold that decides whether a conductor is perfectly
   conducting. It must be higher than insulator threshold.
```

Example:

```
oModule.SetConductivityThreshold 700, 1E+030
```

AssignFiniteCond [2D Extractor]

Use: Assign a single finite conductivity boundary on selected edges.

Command: **2D Extractor>Boundary>Assign>Finite Conductivity**

Syntax:
```
AssignFiniteCond Array("NAME:<Name>", "Edges:=",
Array(<edge ids>), <FiniteCondParameters>)
```

Return Value: none

Parameters:
```
<Name>
   Type:<String>
   Name of the boundary
<edge ids>
   Type:<integer list>
   List of edge ids, separated by comma.
<FiniteCondParameters>
Parameters:
   "Roughness:=",  "<Roughness>",  "UseCoating:=", <Use-
   Coating>, "LayerThickness:=", "<Thickness>", "UseMate-
   rial:=",  <UseMaterial>, "Material:=", "<MaterialName>"
   <Roughness>
      Type: <String>
      Value: double with units of length
   <UseCoating>
      Type: <Boolean>
      Value: true or false
   <Thickness>
      Type: String
      Value: double with units of length
   <UseMaterial>
      Type:<Boolean>
      Value: true or false
   <MaterialName>
      Type: String
      Value: specify material name for coating.
   <Radius>
      Type: String
      Value: double with units of length.
   <Ratio>
```

```
    Type: String
    Value: double.
```

Example: This example is for the Hammerstad-Jensen surface roughness model.

```
oModule.AssignFiniteCond Array("NAME:FiniteCond1",
"Edges:=", Array(7, 9), "Roughness:=",  _ "2um",
"UseCoating:=", true, "LayerThickness:=", "1.2um",
"UseMaterial:=",  _ true, "Material:=", "Copper")
```

Example: This example is for the Hurray surface roughness model.

```
oModule.AssignFiniteCond Array("NAME:FiniteCond1",
"Edges:=", Array(7), "UseCoating:=",  _ false,
"Radius:=", "0.5um", "Ratio:=", "2.9")
```

EditFiniteCond [2D Extractor]

Use: Edit parameters of single finite conductivity boundary.

Command: Double-click finite conductivity boundary in project tree.

Syntax: `EditFiniteCond <BoundaryName>, Array("NAME:<Name>", <FiniteCondParameters>)`

Return Value: none

Parameters:

```
<BoundaryName>
   Type: <String>
   Name of the boundary to edit.
<Name>
   Type:<String>
   Name of the boundary
<FiniteCondParameters>
Parameters:
   "Roughness:=",  "<Roughness>",  "UseCoating:=", <Use-
   Coating>, "LayerThickness:=", "<Thickness>", "UseMate-
   rial:=",  <UseMaterial>, "Material:=", "<MaterialName>"
   <Roughness>
      Type: <String>
      Value: double with units of length
   <UseCoating>
```

```
        Type: <Boolean>
        Value: true or false
      <Thickness>
        Type: String
        Value: double with units of length
      <UseMaterial>
        Type:<Boolean>
        Value: true or false
      <MaterialName>
        Type: String
        Value: specify material name for coating.
```

Example:

```
oModule.EditFiniteCond "FiniteCond1",
Array("NAME:FiniteCond1", "Roughness:=", "2um",
"UseCoating:=",  _ false)
```

Script Commands for Creating and Modifying Boundaries in Q3D Extractor

Following are script commands for creating and modifying boundaries that are recognized by the `"BoundarySetup"` module. In the following commands, all named data can be included/excluded as desired and may appear in any order.

Assign2DTerminal

Use: Assigns 2D terminals

Command: **Q3D Extractor>Nets>Assign 2D Terminals**

Syntax: `Assign2DTerminal <TerminalArray>`

Parameters:

```
<TerminalArray>
   Array("NAME:AssignTerminals",
      Array("Name:SourceList", Array("Name:<SourceName>",
   "Net:=", <NetObject>, "Objects:=", <Assignment 2D>)),
      Array("Name:SinkList", Array("Name:<SinkName>",
   "Net:=", <NetObject>, "Objects:=", <Assignment 2D>)),
      "Name:DeleteList", <Name Array>)
```

Return Value: None

Example:

```
OModule.Assign2DTerminals Array("NAME:AssignTerminals",
Array("NAME:SourceList", Array("NAME:Source2", "Net:=",_
   "Box2", "Objects:=", Array("Rectangle1"))),
   Array("NAME:SinkList", Array("NAME:Sink1", "Net:=",  _
```

```
    "Box1", "Objects:=", Array("Rectangle2"))),
    "DeleteList:=", "")
oModule.AssignPerfectH Array("NAME:PerfH1",_ "Faces:=",
Array(12))
```

AssignNet [Q3D Extractor]

Use: Creates a net.

Command: **Q3D Extractor>Nets>Assign>Net**

Syntax:
```
AssignNet <NetArray>
```

Return Value: None

Parameters:
```
<NetArray>
   Array("NAME:<NetName>", "Objects:=", <AssignmentOb-
   jects>)
```

Example:
```
oModule.AssignNet Array("NAME:Net1",_
   "Objects:=", Array(12))
```

AssignSource [Q3D Extractor]

Use: Creates a current source.

Command: **Q3D Extractor>Nets>Assign>Source**

Syntax:
```
AssignSource <SourceArray>
```

Return Value: None

Parameters:
```
<SourceArray>
   Array("NAME:<SourceName>", "Net:=", <NetName>,
      "Faces:=", <AssignmentFaces>)
```

Example:
```
oModule.AssignSource Array("NAME:Source1",_ "Net:=",
"Net1", "Faces:=", Array(12))
```

AssignSink

Use: Creates a sink.

Command: **Q3D Extractor>Nets>Assign>Sink**

Syntax:
```
AssignSink <SinkArray>
```

Return Value: None

Parameters:
```
<SinkArray>
   Array("NAME:<SinkName>", "Net:=", <NetName>,
      "Faces:=", <AssignmentFaces>)
```

Example:

```
oModule.AssignSink Array("NAME:Sink1",_ "Net:=", "Net1",
   "Faces:=", Array(12))
```

AssignTerminals [Q3D Extractor]

Use: Assigns 1D and 2D terminals.

Command: **Q3D Extractor>Nets>AssignTerminals**

Syntax:

```
AssignTerminals <TerminalArray>
<TerminalArray>
   Array("NAME:AssignTerminals",
   Array("Name:SourceList", Array("Name:<SourceName>",
   "Net:=", <NetObject>, "Objects:=", <Assignment 2D/
   1D>)),
   Array("Name:SinkList", Array("Name:<SinkName>",
   "Net:=", <NetObject>, "Objects:=", <Assignment 2D/
   1D>)),
   "Name:DeleteList", <Name Array>)
```

Return Value: None

Example:

```
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AssignTerminals Array("NAME:AssignTerminals",
Array("NAME:SourceList", Array("NAME:Polyline2",
"Objects:=", Array(  "Polyline2"), "ParentBndID:=",
"Box1", "Net:=", "Box1"), Array("NAME:Rectangle1",
"Objects:=", Array(   "Rectangle1"), "ParentBndID:=",
"Box1", "Net:=", "Box1")), Array("NAME:SinkList"),
"DeleteList:=",  _ "")
```

AssignThinConductor

Use: Assign a thin conductor boundary on a 2D object.

Command: **Q3D Extractor>Boundary>Assign>Thin Conductor**

Syntax:

```
AssignThinConductor <Boundary Name>,
Array("NAME:<Boundary Name>,"Material:=", <Material
Name>, "Thickness:=", <thickness string", "Direction:=",
<direction string>)
```

Return Value: none

Parameters:

```
<Name>
     Type:<String>
     Name of the boundary
```

```
<sheet ids>
     Type:<String>
     List of 2D object names, separated by comma.
<ThinCondParameters>
Parameters:
     "Material:=",  "<Material Name>",  "Thickness:=",
     <Thickness>
<Material Name>
     Type: <String>
     Value: material name, default is "Copper"
<Thickness>
     Type: String
     Value: double with units of length
<Direction>
     Type: String
     Value: can be "Positive" or "Negative"
```

Example:

```
Set oModule = oDesign.GetModule("BoundarySetup")
oModule.AssignThinConductor  "ThinCond1",
Array("NAME:ThinCond1", "Material:=", "Copper",
"Thickness:=", _ "2mm", "Direction:=", "Positive")
```

EditThinConductor

Use: Edit parameters of single thin conductor boundary.

Command: Double-click thin conductor boundary in the project tree.

Syntax: `EditThinConductor <BoundaryName>, Array("NAME:<Name>", <ThinCondParameters>`

Return Value: none

Parameters:

```
BoundaryName>
     Type: <String>
     Name of the boundary to edit.
<Name>
     Type:<String>
     Name of the boundary
```

Example:
```
oModule.EditThinConductor "ThinCond1",
Array("NAME:ThinCond1", "Material:=", "Copper",
"Thickness:=",  "3mm")
```

EditNet [Q3D Extractor]

Use: Edits a net.

Command: Double-click the net in the project tree to modify its settings.

Syntax: `EditNet <NetName>, <NetArray>`

Return Value: None

Example: `oModule.EditNet "Net1", Array ("Name:Net2")`

GetExcitationAssignment [Q3D Extractor]

Use: Return the geometry assignment of an excitation.

Command: None.

Syntax: `GetExcitationAssignment "<name>"`

Return Value: `VARIANT array of the geometry`

Parameters: `<name>`

Type: String

The name of the excitation

Example: `oModule.GetExcitationAssignment "Net1"`

GetExcitations [Q3D Extractor]

Use: Return a listing of excitations

Command: None.

Syntax: `GetExcitations`

Return Value: Variant array, excitation name paired with excitation type.

Example: `oModule.GetExcitations]`

GetNumExcitations [Q3D Extractor]

Use: Gets the number of excitations in a design, including all defined sources, sinks, and nets.

Command: None.

Syntax: `GetNumExcitations "<Type>"`

Return Value: Long

Parameters: `<Type>`

```
   Type: <string>
   Comma separated string. Choices for "string" are
   "Sink", "Source", "Net"
```

Example:

```
Dim num
num = oModule.GetNumExcitations "Net, Sink"
```

ToggleTerminal

Use: Toggles between source and sink.

Command: Click a terminal in the project tree, and then click **Toggle** on the shortcut menu.

Syntax:

```
ToggleTerminal <TerminalName>
```

Return Value: None

SetMaterialThresholds [Q3D Extractor]

Use: Sets conductivity threshold.

Command: **Q3D Extractor>Nets>Set Material Thresholds**

Syntax:

```
SetMaterialThresholds
```

Return Value: None

Parameters:

```
SetMaterialThresholds <InsulatorThreshold>,
<PerfectConductorThreshold>, <Magnetic Threshold>
<InsulatorThreshold>
   Type:<double>
   Threshold for insulator/conductor.
<PerfectConductorThreshold>
   Type:<double>
   Threshold that decides whether a conductor is perfectly
   conducting. It must be higher than insulator threshold.
<Magnetic Threshold>
   Type:<double>
   Threshold that decides whether a material  is magnetic.
   You must specify a threshold for permeability.
   Default Value: 1.01
```

Example:

```
oModule.SetMaterialThresholds 700, 1E+030, 1.01
```

15 Mesh Operations Module Script Commands

Commands for mesh setup and operations should be executed by the "MeshSetup" module.

```
Set oModule = oDesign.GetModule("MeshSetup")
oModule.CommandName <args>
```

Conventions Used in this Chapter

```
<OpName>
   Type: <string>
   Name of a mesh operation.
<AssignmentObjects>
   Type: Array of strings
   An array of object names.
<AssignmentFaces>
   Type: Array of integers.
```

An array of face IDs. The ID of a face can be determined through the user interface using the **3DModeler>Measure>Area** command. The face ID is given in the **Measure Information** dialog box.

General Commands Recognized by the Mesh Operations Module

General commands recognized by the Mesh Operations Module:

DeleteOp

GetOperationName

RenameOp

DeleteOp

Use: Deletes the specified mesh operations.

Command: **Delete** command in the **List** dialog box. Click **HFSS or Q3D Extractor or 2D Extractor >List** to access the **List** dialog box.

Syntax:
```
DeleteOp <NameArray>
```

Return Value: None

Parameters:

`<NameArray>`

Type: Array of strings.

An array of mesh operation names.

Example:
```
oModule.DeleteOp Array("Length1", "SkinDepth1",_
   "Length2")
```

GetOperationNames

Use: Gets the names of mesh operations defined in a design.

Syntax:
```
GetOperationNames(<OperationType>)
```

Return Value: Array of mesh operation names.

Parameters:

`<OperationType>`

`Type: <string>`

For example: "Skin Depth Based"

Example:
```
Set opnames = oModule.GetOperationNames("Length Based")
For Each name in opnames
     Msgbox name
Next
```

RenameOp

Use: Renames a mesh operation.

Command: Right-click the mesh operation in the project tree, and then click **Rename** on the shortcut menu.

Syntax: `RenameOp <OldName>, <NewName>`

Return Value: None

Parameters: `<OldName>`

Type: <string>

Old name for the mesh operation.

`<NewName>`

Type: <string>

New name for the mesh operation.

Example:

```
oModule.RenameOp "SkinDepth1", "NewName"
```

Script Commands for Creating and Modifying Mesh Operations

Script commands for creating and modifying mesh operations are as follows:

AssignLengthOp

AssignModelResolutionOp

AssignSkinDepthOp

AssignTrueSurfOp

EditLengthOp

EditModelResolutionOp

EditSkinDepthOp

EditTrueSurfOp

AssignLengthOp

Use: Assigns length-based operations to the selection.

Command: **HFSS>Mesh Operations>Assign>On Selection** or **HFSS>Mesh Operations>Assign>Inside Selection>Length Based**.

Syntax: `AssignLengthOp <LengthOpParams>`

Return Value: None

Parameters:

```
<LengthOpParams>
   Array("NAME:<OpName>",
      "RefineInside:=", <bool>,
      "Objects:=", <AssignmentObjects>,
      "Faces:=", <AssignmentFaces>,
      "RestrictElem:=", <bool>
      "NumMaxElem:=", <integer>
      "RestrictLength:=", <bool>
      "MaxLength:=", <value>)
```

`RefineInside`

If true, `Objects` should be specified. Implies apply restrictions to tetrahedra inside the object.

If false, `Faces` and/or `Objects` can be specified. Implies apply restrictions to triangles on the surface of the face or object.

`RestrictElem`

If true, `NumMaxElem` should be specified.

RestrictLength

If true, MaxLength should be specified.

Example: Assigning length-based operations to the inside tetrahedra of an object:

```
oModule.AssignLengthOp Array("NAME:Length1", _
   "RefineInside:=", true, _
   "Objects:=", Array("Box1"), _
   "RestrictElem:=", true, _
   "NumMaxElem:=", 1000, _
   "RestrictLength:=",  true, _
   "MaxLength:=", "1mm")
```

For Q3D Extractor, the equivalent command is as follows:

Command: **Q3D Extractor** or **2D Extractor>Mesh Operations>Assign>On Selection** or **Q3D Extractor>Mesh Operations>Assign>Inside Selection>Length Based**

AssignModelResolutionOp

Use: Assigns a model resolution name, value and unit for mesh operations, or specify to UseAutoFeaturelength. If UseAutoFeature length is true, the Defeature length is not used.

Command: **HFFS or Q3D Extractor or 2D Extractor >Mesh Operations>Assign>Model Resolution**

Syntax: `AssignModelResolutionOp Array(<ModelResParams>)`

Return Value: None

Parameters:

```
Array("NAME:<string>",
   "Objects:=", Array( "<modelname>"), _
   "UseAutoLength:=", <Boolean>, _
   "DefeatureLength:=", "<value><units>")
```

Example:

```
Set oDesign = oProject.SetActiveDesign("wg_combiner")
Set oModule = oDesign.GetModule("MeshSetup")
oModule.AssignModelResolutionOp Array("NAME:ModelResolution1",
"Objects:=", Array( "waveguide"), _
"UseAutoLength:=", true, _
"DefeatureLength:=", "71.5891053163818mil")
```

AssignSkinDepthOp

Use: Assigns a skin-depth based operations to the selection.

Command: **HFSS or 2D Extractor >Mesh Operations>Assign>On Selection>Skin Depth Based**

Syntax: `AssignSkinDepthOp <SkinDepthOpParams>`

Return Value: None

Parameters:

```
<SkinDepthOpParams>
   Array("NAME:<OpName>",
      "Faces:=", <AssignmentFaces>,
      "RestrictElem:=", <bool>,
      "NumMaxElem:=", <int>,
      "SkinDepth:=", <value>,
      "SurfTriMaxLength:=", <value>,
      "NumLayers:=", <int>)
```

`RestrictElem`

If true, `NumMaxElem` should be specified.

Example:

```
oModule.AssignSkinDepthOp Array("NAME:SkinDepth1", _
   "Faces:=", Array(7), _
   "RestrictElem:=", true, _
   "NumMaxElem:=", 1000, _
   "SkinDepth:=", "1mm", _
   "SurfTriMaxLength:=", "1mm", _
   "NumLayers:=", 2)
```

AssignTrueSurfOp

Use: Assigns a true surface-based mesh operation on the selection.

Command: **HFSS or Q3D Extractor or 2D Extractor>Mesh Operations>Assign>Surface Approximation**

Syntax: `AssignTrueSurfOp <TrueSurfOpParams>`

Return Value: None

Parameters:

```
<TrueSurfOpParams>
   Array("NAME:<OpName>",
      "Faces:=", <AssignmentFaces>,
      "SurfDevChoice:=",  <RadioOption>,
      "SurfDev:=", <value>,
```

```
    "NormalDevChoice:=", <RadioOption>,
    "NormalDev:=", <value>,
    "AspectRatioChoice:=", <RadioOption>,
    "AspectRatio:=", <double>)
```

<RadioOption>
Type: <int>
0: Ignore
1: Use defaults
2: Specify the value

Example:

```
oModule.AssignTrueSurfOp Array("NAME:TrueSurf1",
   "Faces:=", Array(9), _
   "SurfDevChoice:=",  2, _
   "SurfDev:=", "0.04123105626mm", _
   "NormalDevChoice:=", 2, _
   "NormalDev:=", "15deg", _
   "AspectRatioChoice:=", 1)
```

EditLengthOp

Use: Edits an existing length-based operation. This cannot be used to modify assignments. Instead, the mesh operation should be deleted and a new one created.

Command: Double-click the operation in the project tree to modify its settings.

Syntax: `EditLengthOp <OpName>, <LengthOpParams>`

Return Value: None

Example:

```
oModule.EditLengthOpK "Length1", Array("NAME:Length1", _
   "RefineInside:=", false, _
   "RestrictElem:=", false, _
   "RestrictLength:=", true, _
   "MaxLength:=", "2mm")
```

EditModelResolutionOp

Use: Assigns a model resolution name, value and unit for mesh operations. If UseAutoLength is true, the Defeature length is not used.

Command: Double-click the operation in the project tree to modify its settings.

Syntax:
```
EditModelResolutionOp Array(<ModelResParams>)
```

Return Value:

Parameters:
```
Array("NAME:<string>",
   "Objects:=", Array( "<modelname>"), _
   "UseAutoLength:=", <Boolean>, _
   "DefeatureLength:=", "<value><units>")
```

Example:
```
Set oDesign = oProject.SetActiveDesign("wg_combiner")
Set oModule = oDesign.GetModule("MeshSetup")
oModule.EditModelResolutionOp "ModelResolution1", _ Array("NAME:Mod-
elResolution1", "UseAutoLength:=", false, _
"DefeatureLength:=", "71.5891053163818mil")
```

EditSkinDepthOp

Use: Modifies an existing skin-depth based mesh operation. Assignments cannot be changed using this command. To change the assignment, you must delete operation and create it using a new assignment.

Command: Double-click the operation in the project tree to modify its settings.

Syntax:
```
EditSkinDepthOp <OpName>, <SkinDepthOpParams>
```

Return Value: None

Example:
```
oModule.EditSkinDepthOp "SkinDepth1",
   Array("NAME:SkinD",_
      "RestrictElem:=", false, _
      "SkinDepth:=", "2mm", _
      "SurfTriMaxLength:=", "1mm", _
      "NumLayers:=", 2)
```

EditTrueSurfOp

Use: Modifies an existing true surface approximation-based mesh operation. Assignments cannot be changed using this command. To change the assignment, delete this operation and create it using a new assignment.

Command: Double-click the operation in the project tree to modify its settings.

Syntax: `EditTrueSurfOp <OpName>, <TrueSurfOpParams>`

Return Value: None

Example:

```
oModule.EditTrueSurfOp "TrueSurf2",
   Array("NAME:trusurf", _
      "SurfDevChoice:=", 2, _
      "SurfDev:=","0.03mm", _
      "NormalDevChoice:=", 1, _
      "AspectRatioChoice:=", 2, _
      "AspectRatio:=", 10)
```

16 Analysis Setup Module Script Commands

HFSS analysis setup commands should be executed by the Analysis module, referred to in HFSS scripts as the `"AnalysisSetup"` module.

```
Set oModule = oDesign.GetModule("AnalysisSetup")
```

CopySetup

The documented command is applicable for Q3D Extractor.

Use: Copy a solve setup.

Syntax: `CopySetup <SetupName>`

Return Value: None

Parameters:

```
<SetupName>
   Type: <String>
   Name of solve setup to be copied.
oModule.CopySetup "Setup6"
```

CopySweep

The documented command is applicable for Q3D Extractor.

Use: Copy a sweep.

Syntax: `CopySweep <SetupName>, <SweepName>`

Return Value: None

Parameters:

```
<SetupName>
   Type: <String>
   Name of solve setup to which the sweep belongs.
<SweepName>
   Type:<String>
   Name of sweep copied.
oModule.CopySweep "Setup6", "Sweep1"
```

DeleteDrivenSweep

Use: Deletes a frequency sweep. For HFSS-IE use DeleteSweep.

Command: Right-click a frequency sweep in the project tree, and then click **Rename** on the shortcut menu.

Syntax:
```
DeleteDrivenSweep <SetupName>, <SweepName>
```

Return Value: None

DeleteSetups

Use: Deletes one or more solution setups, which are specified by an array of solution setup names.

Command: Right-click a solution setup in the project tree, and then click **Delete** on the shortcut menu, or delete selected solution setups in the **List** dialog box.

Syntax:
```
DeleteSetups <SetupArray>
```

Return Value: None

Parameters:
```
<SetupArray>
   Array(<name1>, <name2>, ...)
```

Example:
```
oModule.DeleteSetups Array("Setup1", "Setup2")
```

DeleteSweep [HFSS-IE]

Use: Deletes a frequency sweep.

Command: Right-click a frequency sweep in the project tree, and then click **Rename** on the shortcut menu.

Syntax:
```
DeleteSweep <SetupName>, <SweepName>
```

Return Value: None

For Q3D Extractor, the command details are as follows:

Use: Deletes a sweep.

Syntax:
```
DeleteSweep <SetupName>, <SweepName>
```

Return Value: None

Parameters:
```
<SetupName>
   Type: <String>
   Name of solve setup.
<SweepName>
   Type: <String>
   Name of sweep to be deleted.
```

```
oModule.DeleteSweep "Setup6", "Sweep1"
```

EditFrequencySweep

Use: Modifies an existing frequency sweep. For HFSS-IE use EditSweep [HFSS-IE]

Command: Double-click a frequency sweep in the project tree to modify its settings.

Syntax:
```
EditFrequencySweep <SetupName>, <SweepName>,
<AttributesArray>
```

Return Value: None

Parameters:

`<SetupName>`

Type: <string>

Name of the solution setup containing the sweep to be edited.

`<SweepName>`

Type: <string>

Name of the sweep to be edited.

`<Attributes Array>`

```
Array("NAME:<SweepName>",
   "IsEnabled:=", <boolean>,
   "SetupType:=", <SetupType>,
   <FrequencyInformation>,
   "Type:=", <SweepType>,
   <SaveFieldsList>
   <DCExtrapInfo>)
```

See the `InsertFrequencySweep` command for details.

Example:

```
oModule.EditFrequencySweep "Setup1", "Sweep3", _
Array("NAME:Sweep3", "IsEnabled:=", true, _
"SetupType:=", "SinglePoints", _
"ValueList:=", Array("1GHz", "2GHz", "3GHz"), _
"Type:=", "Discrete", _
"SaveFieldsList:=", Array(false, false, false), _
"ExtrapToDC:=", false)
```

EditSetup

Use: Modifies an existing solution setup.

Command: Double-click a solution setup in the project tree to modify its settings.

Syntax: `EditSetup <SetupName>, <AttributesArray>`

Return Value: None

Parameters:

`<SetupName>`

Type: <string>

Name of the solution setup being edited.

`<AttributesArray>`

`Array("NAME:<NewSetupName>", <NamedParameters>)`

See the `InsertSetup` command for details and examples.

For Q3D Extractor, the EditSetup command has the following details.

Use: Edit an existing solution setup.

Command: Double-click a setup in the project tree to edit it.

Syntax:

```
EditSetup <SetupName>, Array("NAME:<NewSetupName>,
"Enabled:=",
<Enabled>,"AdaptiveFreq:=",<AdaptFreq>,<SolverParam>)
```

Return Value: None

Parameters:

```
<SetupName>
   Type: <string>
   Name of the solve setup being edited.
<NewSetupName>
   Type: <string>
   New name of the solve setup being edited
<SaveFields>
   Type: <string>
   Flag to indicate whether the fields are saved for last
   adaptive solution. (True or False).
<Enabled>
   Type: <bool>
   Flag to indicate whether the setup is enabled.
<AdaptFreq>
   Type: <String>
   Frequency with units.
<SolverParam>
   See InsertSetup for details.
```

```
oModule.EditSetup "Setup1", Array("NAME:Setup1", "AdaptiveFreq:=",
"1GHz", "EnableDistribProbTypeOption:=",  _ false, "SaveFields:=",
"true", "Enabled:=", true, Array("NAME:Cap", "MaxPass:=", 10, "Min-
Pass:=", 1, "MinConvPass:=",  _ 2, "PerError:=", 1, "PerRefine:=",
30, "AutoIncreaseSolutionOrder:=", false, "SolutionOrder:=",  _ "Nor-
mal"), Array("NAME:DC", "Residual:=", 1E-005, "SolveResOnly:=",  _
false, Array("NAME:Cond", "MaxPass:=", 10, "MinPass:=", 1, "MinConv-
Pass:=", 1, "PerError:=",  _ 1, "PerRefine:=", 30),
Array("NAME:Mult", "MaxPass:=", 1, "MinPass:=", 1, "MinConvPass:=",
_ 1, "PerError:=", 1, "PerRefine:=", 30)), Array("NAME:AC", "Max-
Pass:=", 10, "MinPass:=",  _ 1, "MinConvPass:=", 2, "PerError:=", 1,
"PerRefine:=", 30))
```

EditSetup (2D Extractor)

Use: Modifies an existing solution setup.

Command: Double-click a solution setup in the project tree to modify its settings.

Syntax:
```
EditSetup <SetupName>,
Array("NAME:<NewSetupName>,<SolverParam>)
```

Return Value: None

Parameters:
```
<SetupName>
    Type: <string>
    Name of the solution setup being edited.
<NewSetupName>
    Type: <string>
    Name of the solution setup being edited.
<SolverParam>
    See the InsertSetup command for details.
```

```
oModule.EditSetup "Setup1", Array("NAME:Setup1", "AdaptiveFreq:=",
"1GHz", "Enabled:=",  _ true, Array("NAME:CGDataBlock", "MaxPass:=",
7, "MinPass:=", 1, "MinConvPass:=", 1, "PerError:=",  _ 1, "PerRe-
fine:=", 30, "DataType:=",  _ "CG", "Included:=", true, "UseParam-
Conv:=", false, "UseLossyParamConv:=",  _ false,
"PerErrorParamConv:=", 1, "UseLossConv:=",  false ),
Array("NAME:RLDataBlock", "MaxPass:=", 10, "MinPass:=",  _ 1, "Min-
ConvPass:=", 1, "PerError:=", 1, "PerRefine:=", 30, "DataType:=",
"RL", "Included:=", true, "UseParamConv:=",  _ false, "UseLossyParam-
Conv:=", false, "PerErrorParamConv:=", 1, "UseLossConv:=",  false))
```

EditSweep [HFSS-IE]

Use: Modifies an existing sweep in HFSS-IE.

Command: Double-click a frequency sweep in the project tree to modify its settings.

Syntax: `EditSweep <SetupName>, <SweepName>, <AttributesArray>`

Return Value: None

Parameters: `<SetupName>`

Type: <string>

Name of the solution setup containing the sweep to be edited.

`<SweepName>`

Type: <string>

Name of the sweep to be edited.

```
<Attributes Array>
      Array("NAME:<SweepName>",
         "IsEnabled:=", <boolean>,
         "SetupType:=", <SetupType>,
         <FrequencyInformation>,
         "Type:=", <SweepType>,
         <SaveFieldsList>
         <DCExtrapInfo>)
<additionalRanges>
   Array( "SetupType:=", "<SetupType>",
   "StartValue:=", "<ValueUnits>",
   "StopValue:=", "<ValueUnits", "Count:=", <value>, ...)
```

See the `InsertFrequencySweep` command for details.

Example:

```
oModule.EditSweep "Setup1", "Sweep3", _
Array("NAME:Sweep3", "IsEnabled:=", true, _
"SetupType:=", "SinglePoints", _
"ValueList:=", Array("1GHz", "2GHz", "3GHz"), _
"Type:=", "Discrete", _
"SaveFieldsList:=", Array(false, false, false), _
"ExtrapToDC:=", false)
```

EditCircuitSettings

Use: Exports equivalent circuit data.

Command: Right-click the **Analysis** folder, and then choose **Edit Circuit Settings.**

Syntax:
```
EditCircuitSettings <ExportSettings>
```

Return Value: None

Parameters:
```
<ExportSettings>
```
See ExportCircuit for details.

Example:
```
oModule.EditCircuitSettings Array("NAME:CircuitData",
"MatrixName:=", "Original", "NumberOfCells:=",  _ "1",
"UserHasChangedSettings:=", true, "IncludeCap:=", true,
"IncludeCond:=",  _ true, Array("NAME:CouplingLimits",
"CouplingLimitType:=", "By Fraction", "CapFraction:=",  _
0.01, "IndFraction:=", 0.01, "ResFraction:=", 0.01,
"CondFraction:=", 0.01), "IncludeDCR:=",  _ false,
"IncudeDCL:=", false, "IncludeACR:=", false,
"IncludeACL:=", false, "ADDResistance:=",  _ true)
```

Example:
```
oModule.EditCircuitSettings Array("NAME:CircuitData",
"MatrixName:=", "Original", "NumberOfCells:=",  _ "1",
"UserHasChangedSettings:=", true, "IncludeCap:=", true,
"IncludeCond:=",  _ true, Array("NAME:CouplingLimits",
"CouplingLimitType:=", "By Fraction", "CapFraction:=",  _
0.01, "IndFraction:=", 0.01, "ResFraction:=", 0.01,
"CondFraction:=", 0.01), "IncludeR:=",  _ false,
"IncludeL:=", false, "ExportDistributed:=", true,
"LumpedLength:=",  _ "7meter", "RiseTime:=", "1s")
```

ExportCircuit

Use: Export equivalent circuit data.

Command: Right-click a setup in the project tree or the **Analysis** folder and choose **Export Circuit.**

Syntax:
```
ExportCircuit <Solution>, <Variation>, <FileName>,
<ExportSettings>, <ModelName>, <Freq>
```

Return Value: none

Parameters:
```
<Solution>
   <SetupName>:<SolutionName>
   <SetupName>
      Type: <string>
      Name of the setup where circuit is being exported
```

<SolutionName>
Type: <String>
Name of the solution.

<Variation>
Type: <string>
The variation where circuit is being exported

<FileName>
Type: <string>
The name of the file where circuit is being exported

<ModelName>
Type: <String>
Model name or name of the sub circuit (Optional). If not specified then <FileName> is considered as model name.

<Freq>
Type: <double>
Sweep frequecny in hz.

<ExportSettings>
Array("NAME:CircuitData", "MatrixName:=", _
<ReduceMatrix>, "NumberOfCells:=", <NumCell>, "UserHasChangedSettings:=", <UserChangedSettings>, "IncludeCap:=", <IncludeCap>, "IncludeCond:=", <IncludeCond>, Array("NAME:CouplingLimits", <CouplingLimitsArray>, "IncludeDCR:=", <IncludeDCR>, "IncludeDCL:=", <IncludeDCL>, "IncludeACR:=", <IncludeACR>, "IncludeACL:=", <IncludeACL>, "ADDResistance:=", <AddResistance>)

Parameters:

<ReduceMatrix>
Type: <string>
One of the reduced matrix setup or "Original"

<NumCell>
Type: <string>
Number of cells in export. Can be a variable.

<UserChangedSettings>
Type: <bool>
Whether user changed settings or use default set-

```
    tings.
<IncludeCap>
    Type: <bool>
    Flag indicates whether to export Capacitance matrix.
<IncludeCond>
    Type: <bool>
    Flag indicates whether to export Conductance matrix.
<IncludeDCR>
    Type: <bool>
    Flag indicates whether to export DC resistance
    matrix.
<IncludeDCL>
    Type: <bool>
    Flag indicates whether to export DC Inductance
    matrix.
<IncludeACR>
    Type: <bool>
    Flag indicates whether to export AC resistance
    matrix.
<IncludeACL>
    Type: <bool>
    Flag indicates whether to export AC inductance
    matrix.
<AddResistance>
    Type: <bool>
    Adds the DC and AC resistance.
```

Note You cannot export both AC and DC matrices unless **AddResistance** is selected.

```
<CouplingLimitsArray>
    Array("NAME:CouplingLimits", "CouplingLimit-
    Type:=",  _ <CouplingLimitType>, <CouplingLimitsPa-
    rameters>, 0.01, "CondFraction:=", 0.01)
    <CouplingLimitType> = "None"
      Argument not needed
    <CouplingLimitType> = "ByFraction"
      <CouplingLimitsParameters>
```

```
"CapFraction:=", <Fraction>, "IndFraction:=",
<Fraction>,
"ResFraction:=",  <Fraction>,"CondFrac-
tion:=",<Fraction>,
Parameters:
<Fraction>
  Type: <double>
  Fraction of the self term
<CouplingLimitType> = "ByValue"
  <CouplingLimitsParameters>
  "CapLimit:=", <Limit>, "IndLimit:=", <Limit>,
  "ResLimit:=", <Limit>,
  "CondLimit:=", <Limit>,
  Parameters:
    <Limit>
    Type: <string>
    Value of the limit.
```

Example:

```
oModule.ExportCircuit "Setup1 : LastAdaptive", "", "C:/
Project/Q3D/FourNets.cir", Array("NAME:CircuitData",
"MatrixName:=",  _ "Original", "NumberOfCells:=", "1",
"UserHasChangedSettings:=", true, "IncludeCap:=",  _
true, "IncludeCond:=", true, Array("NAME:CouplingLimits",
"CouplingLimitType:=",  _ "By Fraction", "CapFraction:=",
0.01, "IndFraction:=", 0.01, "ResFraction:=",  _ 0.01,
"CondFraction:=", 0.01), "IncludeDCR:=", false,
IncudeDCL:=", false, "IncludeACR:=",  _ false,
"IncludeACL:=", false, "ADDResistance:=", true), "",
2000000000000
```

ExportCircuit (2D Extractor)

Use: Exports equivalent circuit data.

Command: Right-click a setup in the project tree or in the **Analysis** folder, and then choose **Export Circuit**.

Syntax:

```
ExportCircuit <SolutionName>, <Variation>, <FileName>,
<ExportSettings>, <ModelName>, <FileType>, <Freq>
```

Return Value: None

Parameters:

```
<SolutionName>
  Type: <string>
```

Name of the setup where the circuit is being exported.

<Variation>

Type: <string>

The variation where circuit is being exported.

<FileName>

Type: <string>

The name of the file where circuit is being exported.

<ExportSettings>

```
Array("NAME:CircuitData", "MatrixName:=",  _
  <MatrixName>, "NumberOfCells:=", <NumberOfCells>,
"UserHasChangedSettings:=",<UserHasChangedSettings >,
"IncludeCap:=",  <IncludeCap>, "IncludeCond:=", <Inclu-
deCond>, <CouplingLimitsArray>, "IncludeR:=",
<IncludeR>, "IncludeL:=", <IncludeL>, "ExportDistrib-
uted:=", <ExportDistributed>,
"LumpedLength:=",<LumpedLength>, "RiseTime:=", <Rise-
Time>)
```

Parameters:

<MatrixName>

Type: <string>

One of the reduced matrix setup or "Original"

<NumberOfCells>

Type: <string>

Number of cells in export. Can be a variable.

<UserHasChangedSettings>

Type: <bool>

Whether user has changed settings or not, defaulted to false.

We set the default values for circuit data is user has changed the settings.

<IncludeCap>

Type:<bool>

Flag to tell whether to include capacitance matrix when exporting.

<IncludeCond>

Type:<bool>

Flag to tell whether to include conductance matrix

```
   when exporting.
<IncludeR>
   Type:<bool>
   Flag to tell whether to include resistance matrix
   when exporting.
<IncludeL>
   Type:<bool>
   Flag to tell whether to include inductance matrix
   when exporting.
<ExportDistributed>
   Type: <bool>
   Flag to tell whether to export in distributed mode or
   Lumped mode.
<LumpedLength>
   Type: <String>
   Length of the design.
<RiseTime>
   Type:<String>
   Rise time to calculate the number of cells.
<CouplingLimitsArray>
   Array("NAME:CouplingLimits", "CouplingLimit-
   Type:=",  _ <CouplingLimitType>, <CouplingLimitsPa-
   rameters>, 0.01, "CondFraction:=", 0.01)
   <CouplingLimitType> = "None"
      Argument not needed
   <CouplingLimitType> = "ByFraction"
      <CouplingLimitsParameters>
      "CapFraction:=", <Fraction>, "IndFraction:=",
      <Fraction>,
      "ResFraction:=",  <Fraction>,"CondFrac-
      tion:=",<Fraction>,
      Parameters:
      <Fraction>
         Type: <double>
         Fraction of the self term
   <CouplingLimitType> = "ByValue"
      <CouplingLimitsParameters>
```

"CapLimit:=", <Limit>, "IndLimit:=", <Limit>,
"ResLimit:=", <Limit>,

"CondLimit:=", <Limit>,

Parameters:

<Limit>

Type: <string>

Value of the limit.

<FileType>

Type: <string>

The type of file format, right now used to specify the type of "HSPICE" file format (because all HSPICE file formats have same extension *.sp).

Values:

"Hspice": simple HSPICE file format.

"Welement": Nexxim/HSPICE W Element file format

"RLGC": Nexxim/HSPICE RLGC W Element file format

<ModelName>

Type: <String>

Model name or name of the sub circuit (Optional). If not specified then <FileName> is considered as model name.

<Freq>

Type: <double>

Sweep frequency in hz.

Example:

```
oModule.ExportCircuit "Setup6 : LastAdaptive", "",  _
"C:/Project/Q3D/2DExtractor_Projects/tline.bsp",
Array("NAME:CircuitData", "MatrixName:=",  _ "Original",
"NumberOfCells:=", "1", "UserHasChangedSettings:=", true,
"IncludeCap:=",  _ true, "IncludeCond:=", true,
Array("NAME:CouplingLimits", "CouplingLimitType:=",  _
"By Fraction", "CapFraction:=", 0.01, "IndFraction:=",
0.01, "ResFraction:=",  _ 0.01, "CondFraction:=", 0.01),
"IncludeR:=", true, "IncludeL:=", true,
"ExportDistributed:=",  _ true, "LumpedLength:=",
"7meter", "RiseTime:=", "1s"), "HSpice", 2000000000
```

GetSetupCount

Use: Gets the number of analysis setups in a design.

Syntax: `GetSetups()`

Return Value: Number of setups.

Parameters: `None`

Example:

```
setupcount = oModule.GetSetupCount()
```

GetSetups

Use: Gets the names of analysis setups in a design.

Syntax: `GetSetups()`

Return Value: Array of analysis setup names.

Parameters: `None`

Example:

```
setupnames = oModule.GetSetups()
```

GetSetupNames

The documented command is valid for Q3D Extractor.

Use: Get the names of Q3D setups.

Command: None

Syntax: `GetSetupNames()`

Return Value: An array of strings. The setup names.

Parameters: `None.`

Example:

```
Set setups = oModule.GetSetupNames()
   numsetups = setups.Count
   for i=0 to numsetups-1
       setup = setups.Item(i)
       MsgBox "Setup Name = " & setup
   Next
```

GetSweepCount

Use: Gets the number of sweeps in a given analysis setup.

Syntax: `GetSweepCount(<SetupName>)`

Return Value: Number of sweeos for the named setup.

Parameters: `<SetupName>`

Type: <string>

Name of the setup.

Example:

```
sweepcount = oModule.GetSweepCount("Setup1")
```

GetSweeps

Use: Gets the names of all sweeps in a given analysis setup.

Syntax: `GetSweeps(<SetupName>)`

Return Value: Array of sweep names.

Parameters: `<SetupName>`

Type: <string>

Name of the setup.

Example:

```
sweepnames = oModule.GetSweeps("Setup1")
```

InsertFrequencySweep

Use: Adds a frequency sweep to a Driven solution-type setup.

Command: **HFSS>Analysis Setup>Add Sweep**

Syntax: `InsertFrequencySweep <SetupName>, <AttributesArray> [<additionalRanges>]`

Return Value: None

Parameters: `<SetupName>`

Type: <string>

Name of the solution setup into which the sweep will be inserted.

```
<Attributes Array>
     Array("NAME:<SweepName>",
        "IsEnabled:=", true,
        "SetupType:=", <SetupType>,
        "Type:=", <SweepType>,
        <FrequencyInformation>,
        <SaveFieldsList>
        <DCExtrapInfo>)
```

<SweepType>
Type: <string>
Ex. "Discrete", "Fast", or "Interpolating".

<SetupType>
Type: <string>
Ex. "LinearSetup", "LinearCount", or "SinglePoints".

<FrequencyInformation>
This will vary based on the sweep and solution type. See the examples below.

<DCExtrapInfo>
Information about whether and how to perform DC extrapolation. This parameter is not used for Discrete sweeps. See the examples below.

Example: Discrete Sweep

```
oModule.InsertFrequencySweep "Setup1", Array("NAME:Sweep2", _
"IsEnabled:=", true,
"SetupType:=", "LinearStep", _
"StartValue:=", "19.5GHz", _
"StopValue:=", "20.4GHz", _
"StepSize:=", "0.1GHz", _
"Type:=", "Discrete", _
"SaveFields:=", false, "ExtrapToDC:=", false)
```

Example: Fast Sweep

```
oModule.InsertFrequencySweep "Setup1", Array("NAME:Sweep4", _
"IsEnabled:=", true,
"SetupType:=", "LinearStep", _
"StartValue:=", "0GHz", _
"StopValue:=", "20.4GHz", _
"StepSize:=", "0.1GHz", _
"Type:=", "Fast", "SaveFields:=", true, _
"ExtrapToDC:=", true, _
```

```
"MinSolvedFreq:=", "0.1GHz")
```

Example: Interpolating Sweep with additional setups

```
oModule.InsertFrequencySweep "Setup1", Array("NAME:Sweep3", _
"IsEnabled:=", true, "SetupType:=",  _
  "LinearStep", "StartValue:=", "0GHz", _
"StopValue:=", "2.5GHz", "StepSize:=", "0.005GHz",
"Type:=", "Interpolating", _
"SaveFields:=", false, _
"InterpTolerance:=", 0.5, _
"InterpMaxSolns:=", 50, "InterpMinSolns:=", 0, _
"InterpMinSubranges:=", 1, _
"ExtrapToDC:=", true, "MinSolvedFreq:=", "0.005GHz", _
"InterpUseS:=", true, _
"InterpUseT:=", false, "InterpUsePortImped:=", false, _
"InterpUsePropConst:=", true, "UseFullBasis:=", true)
Array( "SetupType:=", "LogScale", "StartValue:=", "11GHz",
"StopValue:=",  _
              "12GHz", "Count:=", 91, _
         "SetupType:=", "LinearCount", "StartValue:=", "13GHz", _
              "StopValue:=", "100GHz", "Count:=", 91)
```

Example: Discrete sweeps with linear step and log scale:

```
oModule.InsertFrequencySweep "Setup1", Array("NAME:Sweep2", _
"IsEnabled:=", true,
"SetupType:=", "LinearStep", _
"StartValue:=", "0.005GHz", _
"StopValue:=", "2.5GHz", _
"StepSize:=", "0.005GHz", _
"Type:=", "Discrete", "SaveFields:=", false, _
"ExtrapToDC:=", false)

oModule.InsertFrequencySweep "Setup1", Array("NAME:Sweep3", _
```

```
"IsEnabled:=", true, "SetupType:=", "LogScale", _
"StartValue:=", "1GHz", _
"StopValue:=", "10GHz", _
"SamplesPerDecade:=", 4, _
"Type:=", "Discrete", _
"SaveFields:=", false, "ExtrapToDC:=", false)
```

Example: A Fast sweep, specified using the starting and stopping frequencies and the step count:

```
oModule.InsertFrequencySweep "Setup1", Array("NAME:Sweep4", _
"IsEnabled:=", true, "SetupType:=", "LinearCount", _
"StartValue:=", "1GHz", _
"StopValue:=", "10GHz", _
"Count:=", 3, _
"Type:=", "Fast", _
"SaveFields:=", true, "ExtrapToDC:=", false)
```

For Q3D Extractor the command details are as follows:

Use: Inserts a sweep in the selected solve setup.

Syntax: `InsertSweep`

Return Value: None

Parameters:

```
oModule.InsertSweep <SetupName>, <SweepDataArray>
<SetupName>
   Type: <String>
   Name of the solve setup in which the sweep is added.
<SweepDataArray>
   Array("NAME:<SweepName>", "IsEnabled:=", <EnableF-
   lag>, "SetupType:=",  _
     <SetupType>", <SetupTypeParameters>,   "Type:=",
   <SweepType>, "SaveFields:=", <SaveFields>, <Interp-
   SweepParameters>)
   Array("NAME:<SweepName>", "IsEnabled:=", <EnableF-
   lag>, "SetupType:=",  _
     <SetupType>", <SetupTypeParameters>,   "Type:=",
   <SweepType>, "SaveFieldsList:=", <SaveFieldsListParam-
   eters>, <InterpSweepParameter>)
```

Example:

```
Array("NAME:Sweep1", "IsEnabled:=", true, "SetupType:=",
_ "LinearStep", "StartValue:=", "1GHz", "StopValue:=",
"10GHz", "StepSize:=",  _ "1GHz", "Type:=", "Discrete",
"SaveFields:=", true)
```

Example:

```
Array("NAME:Sweep1", "IsEnabled:=", true, "SetupType:=",
_ "LinearCount", "StartValue:=", "1GHz", "StopValue:=",
"100GHz", "Count:=", 100, "Type:=",  _  "Discrete",
"SaveFields:=", true)
```

Example:

```
Array("NAME:Sweep1", "IsEnabled:=", true, "SetupType:=",
_ "LinearCount", "StartValue:=", "1GHz", "StopValue:=",
"100GHz", "Count:=", 100, "Type:=",  _  "Discrete",
"SaveFields:=", true)
```

Example:

```
Array("NAME:Sweep1", "IsEnabled:=", true, "SetupType:=",
_ "SinglePoints", "ValueList:=", Array("1GHz", "2GHz",
"5GHz"), "Type:=",  _ "Discrete", "SaveFieldsList:=",
Array(false, false, false)
```

Example:

```
Array("NAME:Sweep2", "IsEnabled:=", true, "SetupType:=",
_ "LinearStep", "StartValue:=", "1GHz", "StopValue:=",
"10GHz", "StepSize:=",  _ "1GHz", "Type:=",
"Interpolating", "SaveFields:=", false,
"InterpTolerance:=",  _ 0.01, "InterpMaxSolns:=", 9,
"InterpMinSolns:=", 4)
```

Parameters:

<SweepName>

Type: <String>

Sweep name

<EnableFlag>

Type:<bool>

Flag to tell whether sweep is enabled.

<SetupType>

Type:String>

Values: "LinearStep", "LinearCount", "LogScale", "SinglePoints"

<SetupTypeParameters>

Base on the value of setup type, setup type parameters varies.

<LinearStepParameters>

StartValue:=", <StartValue>, "StopValue:=",

```
    <StopValue>, "StepSize:=",   <StepSize>
    <StartValue>
      Type:<String>
      Start frequency.
    <StopValue>
      Type: <String>
      Stop Frequency
    <StepSize>
      Type:<String>
      Step frequency.
  <LinearCountParameters>
    StartValue:=", <StartValue>, "StopValue:=",
    <StopValue>, "Count:=",   <Count>
    <StartValue>
      Type:<String>
      Start frequency.
    <StopValue>
      Type: <String>
      Stop Frequency
    <Count>
      Type:<integer>
      Number of frequency points in the given range.
  <LogScaleParameters>
    "StartValue:=", <Start-
    Value>,"StopValue:=",<StopValue>, "SamplesPerDe-
    cade:=",   <Samples>
    <StartValue>
      Type:<String>
      Start frequency.
    <StopValue>
      Type: <String>
      Stop Frequency
    <Samples>
      Type:<integer>
      Number of samples per decade.
    <SinglePointsParameters>
```

```
                "ValueList:=", Array(<FrequencyPoint>,<Frequen-
                cyPoint> ..),
                <FrequencyPoint>
                Type: <String>
                Single frequency point.
        <SweepType>
          Type:<String>
          Values: "Discrete", "Interpolating"
          Type of sweep.
        <SaveFields>
          Type:<bool>
          Flag, whether to save fields for the frequency
          points.
        <SaveFieldsListParameters>
          "SaveFieldsList:=", Array(<SaveFields>,<Save-
          Fields>,…)
          Type:Array of flags.
        <InterpSweepParameter>
```

Note This parameter applies only to interpolating sweeps.

```
          "InterpTolerance:=",  <Tolerance>, "InterpMax-
          Solns:=",<MaxSolutions>, "InterpMinSolns:=",<MinSo-
          lutions>,
          <Tolerance>
            Type: <double>
            Tolerance for interpolating sweep.
          <MaxSolutions>
            Type:<Integer>
            Maximum basis points for interpolating sweep.
          <MinSolutions>
            Type:<Integer>
            Minimum basis points for interpolating sweep.
        <InterpSweepParameter> (For 2D Extractor)
          "InterpTolerance:=",  <Tolerance>, "InterpMax-
          Solns:=",<MaxSolutions>, "InterpMinSolns:=",<MinSo-
          lutions>, "InterpMinSubranges:=",
          <InterpMinSubRanges>
```

```
                    <Tolerance>
                      Type: <double>
                      Tolerance for interpolating sweep.
                    <MaxSolutions>
                      Type:<Integer>
                      Maximum basis points for interpolating sweep.
                    <MinSolutions>
                      Type:<Integer>
                      Minimum basis points for interpolating sweep.
                    <InterpMinSubRanges>
                      Type:<Integer>
                      Minimum number of subranges in the interpolator.
```

Example:

```
oModule.InsertSweep "Setup4", Array("NAME:Sweep2",
"IsEnabled:=", true, "SetupType:=",  _ "LinearStep",
"StartValue:=", "1GHz", "StopValue:=", "10GHz",
"StepSize:=",  _ "1GHz", "Type:=", "Interpolating",
"SaveFields:=", false, "InterpTolerance:=",  _ 0.01,
"InterpMaxSolns:=", 9, "InterpMinSolns:=", 4,
InterpMinSubranges:=", 2
```

Example:

```
oModule.InsertSweep "Setup4", Array("NAME:Sweep1",
"IsEnabled:=", true, "SetupType:=",  _
  "LinearStep", "StartValue:=", "1GHz", "StopValue:=",
"10GHz", "StepSize:=",  _
  "1GHz", "Type:=", "Discrete", "SaveFields:=", true)
```

Example:

```
oModule.InsertSweep "Setup4", Array("NAME:Sweep2",
"IsEnabled:=", true, "SetupType:=",  _
  "LinearStep", "StartValue:=", "1GHz", "StopValue:=",
"10GHz", "StepSize:=",  _
  "1GHz", "Type:=", "Interpolating", "SaveFields:=",
false, "InterpTolerance:=",  _
0.01, "InterpMaxSolns:=", 9, "InterpMinSolns:=", 4)
```

InsertSetup

Use: Adds a new solution setup.

Command: **HFSS>Analysis Setup>Add Solution Setup**

Syntax: `InsertSetup <SetupType>, <AttributesArray>`

Return Value: None

Parameters:

`<SetupType>`

Type: <string>

`"HfssDriven"` or `"HfssEigen"`. Must match the HFSS solution type.

`<AttributesArray>`

`Array("NAME:<SetupName>", <Named Parameters>)`

`<Named Parameters>`

The named parameters will vary according to the solution type. To see the required parameters for a specific set of parameters and their format, use the record script function, and view the resulting script in a text edtor. See the examples below.

Example: A Driven solution type with a mesh link. References to dependent solve in old scripts are converted to mesh link form.

```
Set oModule = oDesign.GetModule("AnalysisSetup")
oModule.InsertSetup "HfssDriven",
Array("NAME:Setup1",
"Frequency:=", "1GHz",
"MaxDeltaE:=", 0.1,
"MaximumPasses:=", 6,
"MinimumPasses:=", 1,
"MinimumConvergedPasses:=", 1,
"PercentRefinement:=", 30,
"IsEnabled:=", true,
Array("NAME:MeshLink",
"Project:=", "Tee.aedt",
"Design:=", "TeeModel",
"Soln:=", "Setup1 : LastAdaptive",
Array("NAME:Params", "offset:=", "0in"),
"ForceSourceToSolve:=", false,
"PreservePartnerSoln:=", false,
"PathRelativeTo:=", "SourceProduct",
"ApplyMeshOp:=", true),
"BasisOrder:=", 1,
"UseIterativeSolver:=", false,
"DoLambdaRefine:=", false,
"DoMaterialLambda:=", true,
```

```
"SetLambdaTarget:=", false,
"Target:=", 0.3333,
"UseMaxTetIncrease:=", false,
"MaxTetIncrease:=", 1000000,
"EnableSolverDomains:=", false,
"ThermalFeedback:=", false,
"UsingConstantDelta:=", 0,
"ConstantDelta:=", "0s",
"NumberSolveSteps:=", 1)
```

Example: A Driven solution type with ports:

```
Set oModule = oDesign.GetModule("AnalysisSetup")
oModule.InsertSetup "HfssDriven",
Array("NAME:Setup2",
"Frequency:=", "1GHz",
"PortsOnly:=", false,
"MaxDeltaS:=", 0.02,
"UseMatrixConv:=", false,
"MaximumPasses:=", 6,
"MinimumPasses:=", 1,
"MinimumConvergedPasses:=", 1,
"PercentRefinement:=", 30,
"IsEnabled:=", true,
"BasisOrder:=", 1,
"UseIterativeSolver:=", true,
"IterativeResidual:=", 0.0001,
"DoLambdaRefine:=", true,
"DoMaterialLambda:=", false,
"SetLambdaTarget:=", false,
"Target:=", 0.3333,
"UseMaxTetIncrease:=", false,
"MaxTetIncrease:=", 1000000,
"PortAccuracy:=", 2,
"UseABCOnPort:=", true,
"SetPortMinMaxTri:=", false,
```

```
    "EnableSolverDomains:=", false,
    "ThermalFeedback:=", false,
    "UsingConstantDelta:=", 0,
    "ConstantDelta:=", "0s",
    "NumberSolveSteps:=", 1)
```

Example: An Eigenmode solution type:

```
Set oProject = oDesktop.SetActiveProject("cavity")
Set oDesign = oProject.SetActiveDesign("HFSSModel1")
Set oModule = oDesign.GetModule("AnalysisSetup")
oModule.InsertSetup "HfssEigen",
Array("NAME:Setup2",
"MinimumFrequency:=", "1.77347GHz",
"NumModes:=", 1,
"MaxDeltaFreq:=", 10,
"ConvergeOnRealFreq:=", true,
"MaximumPasses:=", 3,
"MinimumPasses:=", 1,
"MinimumConvergedPasses:=", 1,
"PercentRefinement:=", 30,
"IsEnabled:=", true,
"BasisOrder:=", 1,
"UseIterativeSolver:=", false,
"DoLambdaRefine:=", true,
"DoMaterialLambda:=", true,
"SetLambdaTarget:=", false,
"Target:=", 0.2,
"UseMaxTetIncrease:=", false,
"MaxTetIncrease:=", 1000000,
"UsingConstantDelta:=", 0,
"ConstantDelta:=", "0s",
"NumberSolveSteps:=", 1)
```

Example:

A Driven solution type with ports and matrix convergence:

```
Set oDesign = oProject.SetActiveDesign("packagehfss")
```

```
Set oModule = oDesign.GetModule("AnalysisSetup")
oModule.InsertSetup "HfssDriven",
Array("NAME:Setup2",
"Frequency:=", "1GHz",
"PortsOnly:=", false,
"MaxDeltaS:=", 0.02,
"UseMatrixConv:=", true,
Array("NAME:ConvergenceMatrix",
"AllDiagEntries:=", true,
"MagMinThreshold:=", 0.01,
"Entry:=", Array("Port1:=", "abc", "ModeNum1:=", 1)),
"MaximumPasses:=", 6,
"MinimumPasses:=", 1,
"MinimumConvergedPasses:=", 1,
"PercentRefinement:=", 30,
"IsEnabled:=", true,
"BasisOrder:=", 1,
"UseIterativeSolver:=", false,
"DoLambdaRefine:=", true,
"DoMaterialLambda:=", true,
"SetLambdaTarget:=", false,
"Target:=", 0.3333,
"UseMaxTetIncrease:=", false,
"MaxTetIncrease:=", 1000000,
"PortAccuracy:=", 2,
"UseABCOnPort:=", true,
"SetPortMinMaxTri:=", false,
"EnableSolverDomains:=", false,
"ThermalFeedback:=", false,
"UsingConstantDelta:=", 0,
"ConstantDelta:=", "0s",
"NumberSolveSteps:=", 1)
```

For Q3D Extractor, the command details are as follows:

Use: Adds a new solution setup.

Command: **Q3D Extractor>Analysis Setup>Add Solution Setup**

Syntax:

```
InsertSetup "Matrix", Array ("NAME:<SetupName>",
"Enabled:=", <Enabled>, "AdaptiveFreq:=", <AdaptFreq>,
<SolveParam>)
```

Return Value: None

Parameters:

<SetupName>

Type: <string>

Name of the setup.

<SaveFields>

Type: <string>

Flag to indicate whether the fields are saved for last adaptive solution. (True or False).

<Enabled>

Type: <bool>

Flag to indicate whether the setup is enabled.

<AdaptFreq>

Type: <String>

Frequency with units.

<SolverParam>

Array("NAME:Cap", "Residual:=", <Residual>, <AdaptParam>),

Array("NAME:DC", "Residual:=", <Residual>, "SolveResOnly:=", <SolveResOnly>, Array("NAME:Cond", <AdaptParam>), Array("NAME:Mult", <AdaptParam>)),

Array("NAME:AC", "Residual:=", 1E-005, Array("NAME:Cond", <Adapt-Param>), Array("NAME:Mult", <AdaptParam>))

Parameters:

<ACFreq>

Type: <string>

Frequency of the AC solution.

<Residual>

Type: <double>

Solver residual.

<SolveResOnly>

Type: <bool>

Solve resistance only for a DC solution.

<AdaptParam>

"MaxPass:=", <MaxPass>, "PerError:=", <PerError>,

"PerRefine:=", <PerRefine>

Parameters:

<MaxPass>

Type: <int>

Maximum pass for a solution

<PerError>

Type: <double>

Percentage error as stopping criteria.

<PerRefine>

Type: <double>

Mesh refinement at each pass in percentage.

Example:

```
oModule.InsertSetup "Matrix", Array("NAME:Setup1", "AdaptiveFreq:=",
"1GHz", "EnableDistribProbTypeOption:=",  _ false, "SaveFields:=",
"true", "Enabled:=", true, Array("NAME:Cap", "MaxPass:=", 10, "Min-
Pass:=", 1, "MinConvPass:=",  _ 1, "PerError:=", 1, "PerRefine:=",
30, "AutoIncreaseSolutionOrder:=", false, "SolutionOrder:=",  _ "Nor-
mal"), Array("NAME:DC", "Residual:=", 1E-005, "SolveResOnly:=",  _
false, Array("NAME:Cond", "MaxPass:=", 10, "MinPass:=", 1, "MinConv-
Pass:=", 1, "PerError:=",  _ 1, "PerRefine:=", 30),
Array("NAME:Mult", "MaxPass:=", 1, "MinPass:=", 1, "MinConvPass:=",
_ 1, "PerError:=", 1, "PerRefine:=", 30)), Array("NAME:AC", "Max-
Pass:=", 10, "MinPass:=",  _ 1, "MinConvPass:=", 1, "PerError:=", 1,
"PerRefine:=", 30))
```

InsertSetup (2D Extractor)

Use: Adds a new solution setup.

Command: **2D Extractor>Analysis Setup>Add Solution Setup**

Syntax: `InsertSetup "2DMatrix", Array ("NAME:<SetupName>", <SolveParam>`

Return Value: None

Parameters:

`<SetupName>`

`Type: <string>`

`Name of the setup.`

<SolverParam>

`<AdaptiveFreq>`

`Type:<String>, String consist of double and units of frequency. Example "1GHz". Adaptive frequency for CG`

```
    and RL.
<Enabled>
    Type: <bool>
Array("NAME:CGDataBlock", <AdaptParam>),
Array("NAME:RLDataBlock", ", <AdaptParam>),
Parameters:
<AdaptParam>
    "MaxPass:=",10, "MinPass:=",  _
      1, "MinConvPass:=", 1, "PerError:=", 1, "PerRe-
    fine:=", 30, "SolverResidual:=",  _
      1E-005, "SolverChoice:=", "Auto", "DataType:=",
    "RL", "Included:=", true, "UseParamConv:=",  _
      false, "UseLossyParamConv:=", false, "PerError-
    ParamConv:=", 1
    Parameters:
      <MaxPass>
        Type: <int>
        Maximum pass for a solution
      <MinPass>
        Type: <int>
        Minimum pass for a solution
      <MinConvPass>
        Type: <int>
        Minimum converged  pass for a solution
      <PerError>
        Type: <double>
        Percentage error as stopping criteria
      <PerRefine>
        Type: <double>
        Mesh refinement at each pass in percentage
      <DataType>
        Type: <String> , values are "RL" and "CG"
        Type of data.
      <Included>
        Type: <bool>
        Flag that tells whether to extract a particular
```

data or not.

<UserParamConv>

<Type>: <bool>

Flag to tell whether to include the parameter convergence.

<UseLossyParamConv>

Type: <bool>

Flag, to include lossy parameters (G and R) while calculating convergence.

<PerErrorParamConv>

Type:<double>

Percent error for parameter convergence.

<UseLossConv>

Type: <bool>

Flag, to include loss in convergence criterion.

Example:

```
oModule.InsertSetup "Matrix", Array("NAME:Setup1",
"AdaptiveFreq:=", "1GHz", "Enabled:=",  _ true,
Array("NAME:CGDataBlock", "MaxPass:=", 5, "MinPass:=", 1,
"MinConvPass:=", 1, "PerError:=",  _ 1, "PerRefine:=",
30"DataType:=",  _ "CG", "Included:=", true,
"UseParamConv:=", false, "UseLossyParamConv:=",  _ false,
"PerErrorParamConv:=", 1, "UseLossConv:=",  false),
Array("NAME:RLDataBlock", "MaxPass:=", 10, "MinPass:=",
_ 1, "MinConvPass:=", 1, "PerError:=", 1, "PerRefine:=",
30, "DataType:=", "RL", "Included:=", true,
"UseParamConv:=",  _ false, "UseLossyParamConv:=", false,
"PerErrorParamConv:=", 1, "UseLossConv:=",  false))
```

InsertSetup [HFSS-IE]

Use: Adds a new HFSS-IE solution setup.

Command: **HFSS>Analysis Setup>Add Solution Setup**

Syntax:

```
InsertSetup <SetupType>, <AttributesArray>
[<AdditionalSetups>]
```

Return Value: None

Parameters:

<SetupType>

Type: <string>

"HFIESetup". Must match the HFSS solution type.

```
<AttributesArray>
Array("NAME:<SetupName>", <Named Parameters>)

<Named Parameters>
```

The named parameters will vary according to the solution type. To see the required parameters for a specific set of parameters and their format, use the record script function, and view the resulting script in a text edtor. See the HFSS-IE example below.

```
<additionalRanges>
Array( "SetupType:=", "<SetupType>",
"StartValue:=", "<ValueUnits>",
"StopValue:=", "<ValueUnits", "Count:=", <value>, ...)
```

Example:

```
oModule.InsertSetup "HFIESetup", Array("NAME:Setup3",
"MaximumPasses:=", 6,
"MinimumPasses:=", 1,
"MinimumConvergedPasses:=", 1,
"PercentRefinement:=", 30,
"Enabled:=", true,
"AdaptiveFreq:=", "1GHz",
"DoLambdaRefine:=", true,
"UseDefaultLambdaTarget:=", true,
"Target:=", 0.25,
"DoMaterialLambda:=", true,
"MaxDeltaS:=", 0.02,
"MaxDeltaE:=", 0.1,
"UsePOSolver:=", true)
```

InsertSetup [Transient]

Use: Add a new solution setup to a Transient design

Command: **FSS>Analysis Setup>Add Solution Setup**

Syntax: `InsertSetup <SetupType>, <AttributesArray>`

Return Value: None

Parameters: `<SetupType>`

Type: <string>

`"HfssTransient"`. Must match the HFSS solution type.

```
<AttributesArray>
Array("NAME:<SetupName>", <Named Parameters>)

<Named Parameters>
```

The named parameters will vary according to the solution type. To see the required parameters for a specific set of parameters and their format, use the record script function, and view the resulting script in a text edtor. See the examples below.

```
<additionalRanges>
   Array( "SetupType:=", "<SetupType>",
   "StartValue:=", "<ValueUnits>",
   "StopValue:=", "<ValueUnits", "Count:=", <value>, ...)
```

Example: Transient Solution Type

```
Set oModule = oDesign.GetModule("AnalysisSetup")
oModule.InsertSetup "HfssTransient",
Array("NAME:Setup1",
"Frequency:=", "1GHz",
"MaxDeltaE:=", 0.1,
"MaximumPasses:=", 20,
"IsEnabled:=", true,
"BasisOrder:=", -1,
"NoAdditionalRefinementOnImport:=", true,
Array("NAME:Transient",
"UseAutoTermination:=", 1,
"SteadyStateCriteria:=", 0.001,
"UseMinimumDuration:=", 0,
"TerminateOnMaximum:=", 1,
"UseMaxTime:=", 1,
"MaxTime:=", "20000ps"))
```

Example: Transient Network Solution Type

```
Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
Set oModule = oDesign.GetModule("AnalysisSetup")
oModule.InsertSetup "HfssTransient",
Array("NAME:Setup1",
"Frequency:=", "0.55GHz",
```

```
"MaxDeltaE:=", 0.1,
"MaximumPasses:=", 20,
"IsEnabled:=", true,
"BasisOrder:=", -1,
"NoAdditionalRefinementOnImport:=", true,
Array("NAME:Transient", "TimeProfile:=", "TDR",
"HfssFrequency:=", "0.55GHz",
"MinFreq:=", "0Hz",
"MaxFreq:=", "25028.5714285714MHz",
"NumFreqsExtracted:=", 401,
"RiseTime:=", "35ps",
"SyncTDRMidpoint:=", 1,
"TDRMidpoint:=", "17.5ps",
"UseAutoTermination:=", 0,
"TerminateOnMaximum:=", 1,
"UseMaxTime:=", 1,
"MaxTime:=", "20000ps"))
```

InsertSweep [HFSS-IE]

Use: Adds a frequency sweep to a Driven solution-type setup in HFSS-IE.

Command: **HFSS-IE>Analysis Setup>Add Sweep**

Syntax: `InsertSweep <SetupName>, <AttributesArray>`

Return Value: None

Parameters: `<SetupName>`

Type: <string>

Name of the solution setup into which the sweep will be inserted.

```
<Attributes Array>
     Array("NAME:<SweepName>",
        "IsEnabled:=", true,
        "SetupType:=", <SetupType>,
        "Type:=", <SweepType>,
        <FrequencyInformation>,
        <SaveFieldsList>
        <DCExtrapInfo>)
```

<SweepType>
Type: <string>
Ex. "Discrete", "Fast", or "Interpolating".

<SetupType>
Type: <string>
Ex. "LinearSetup", "LinearCount", or "SinglePoints".

<FrequencyInformation>
This will vary based on the sweep and solution type. See the examples below.

<DCExtrapInfo>
Information about whether and how to perform DC extrapolation. This parameter is not used for Discrete sweeps. See the examples below.

<additionalRanges>
Array("SetupType:=", "<SetupType>",
"StartValue:=", "<ValueUnits>",
"StopValue:=", "<ValueUnits", "Count:=", <value>, ...)

Example: Discrete Sweep

```
oModule.InsertSweep "Setup1", Array("NAME:Sweep2", _
"IsEnabled:=", true,
"SetupType:=", "LinearStep", _
"StartValue:=", "19.5GHz", _
"StopValue:=", "20.4GHz", _
"StepSize:=", "0.1GHz", _
"Type:=", "Discrete", _
"SaveFields:=", false, "ExtrapToDC:=", false)
```

Example: Fast Sweep

```
oModule.InsertSweep "Setup1", Array("NAME:Sweep4", _
"IsEnabled:=", true,
"SetupType:=", "LinearStep", _
```

```
"StartValue:=", "0GHz", _
"StopValue:=", "20.4GHz", _
"StepSize:=", "0.1GHz", _
"Type:=", "Fast", "SaveFields:=", true, _
"ExtrapToDC:=", true, _
"MinSolvedFreq:=", "0.1GHz")
```

Example: Interpolating Sweep

```
oModule.InsertSweep "Setup1", Array("NAME:Sweep3", _
"IsEnabled:=", true, "SetupType:=",  _
  "LinearStep", "StartValue:=", "0GHz", _
"StopValue:=", "2.5GHz", "StepSize:=", "0.005GHz",
"Type:=", "Interpolating", _
"SaveFields:=", false, _
"InterpTolerance:=", 0.5, _
"InterpMaxSolns:=", 50, "InterpMinSolns:=", 0, _
"InterpMinSubranges:=", 1, _
"ExtrapToDC:=", true, "MinSolvedFreq:=", "0.005GHz", _
"InterpUseS:=", true, _
"InterpUseT:=", false, "InterpUsePortImped:=", false, _
"InterpUsePropConst:=", true, "UseFullBasis:=", true)
```

Example: Discrete sweeps with linear step and log scale:

```
oModule.InsertSweep "Setup1", Array("NAME:Sweep2", _
"IsEnabled:=", true,
"SetupType:=", "LinearStep", _
"StartValue:=", "0.005GHz", _
"StopValue:=", "2.5GHz", _
"StepSize:=", "0.005GHz", _
"Type:=", "Discrete", "SaveFields:=", false, _
"ExtrapToDC:=", false)

oModule.InsertSweep "Setup1", Array("NAME:Sweep3", _
```

```
"IsEnabled:=", true, "SetupType:=", "LogScale", _
"StartValue:=", "1GHz", _
"StopValue:=", "10GHz", _
"SamplesPerDecade:=", 4, _
"Type:=", "Discrete", _
"SaveFields:=", false, "ExtrapToDC:=", false)
```

Example: A Fast sweep, specified using the starting and stopping frequencies and the step count:

```
oModule.InsertSweep "Setup1", Array("NAME:Sweep4", _
"IsEnabled:=", true, "SetupType:=", "LinearCount", _
"StartValue:=", "1GHz", _
"StopValue:=", "10GHz", _
"Count:=", 3, _
"Type:=", "Fast", _
"SaveFields:=", true, "ExtrapToDC:=", false)
```

PasteSetup

Use: Paste a solve setup.

Syntax: `PasteSetup`

Return Value: None

Example:

```
oModule.PasteSetup
```

PasteSweep

Use: Paste a sweep.

Syntax: `PasteSweep <SetupName>`

Return Value: None

Parameters:

```
<SetupName>
   Type: <String>
   Name of solve setup where the copied sweep is pasted.
```

Example:

```
oModule.PasteSweep "Setup6"
```

RenameDrivenSweep

Use: Renames an existing frequency sweep in HFSS. For HFSS-IE use RenameSweep.

Command: Right-click a frequency sweep in the project tree, and then click **Rename** on the shortcut menu.

Syntax:
```
RenameDrivenSweep <SetupName>, <OldSweepName>,
   <NewSweepName>
```

Return Value: None

Example:
```
oModule.RenameDrivenSweep "Setup1", "Sweep1", _
   "MySweep"
```

RenameSetup

Use: Renames an existing solution setup.

Command: Right-click a solution setup in the project tree, and then click **Rename** on the shortcut menu.

Syntax:
```
RenameSetup <OldName>, <NewName>
```

Return Value: None

Parameters:

`<OldName>`

Type: <string>

Name of the solution setup being renamed.

`<NewName>`

Type: <string>

New name for the solution setup.

Example:
```
oModule.RenameSetup "Setup1", "Setup2"
```

RenameSweep [HFSS-IE]

Use: Renames an existing frequency sweep in HFSS-IE.

Command: Right-click a frequency sweep in the project tree, and then click **Rename** on the shortcut menu.

Syntax:
```
RenameSweep <SetupName>, <OldSweepName>,
   <NewSweepName>
```

Return Value: None

Example:
```
oModule.RenameSweep "Setup1", "Sweep1", _
   "MySweep"
```

For Q3D Extractor, the command details are as follows:

Use: Rename an existing sweep.

Command: Right-click a sweep in the project tree, and then click **Rename** on the shortcut menu.

Syntax:
```
RenameSweep <SetupName>, <OldSweepName>, <NewSweepName>
```

Return Value: None

Parameters:
```
<SetupName>
   Type: <string>
   Name of the setup.
<OldSweepName>
   Type: <String>
   Name of sweep to be renamed.
<NewSweepName>
   Type: <string>
   New name for the sweep.
```

Example:
```
oModule.RenameSweep "Setup6", "Sweep3", "NewSweep"
```

RevertAllToInitial

Use: Marks the current mesh for all solution setups as invalid. This will force the next simulation to begin with the initial mesh.

Command: **HFSS>Analysis Setup>Revert to Initial Mesh**

Syntax:
```
RevertAllToInitial
```

Return Value: None

For Q3D Extractor and 2D Extractor, the command details are as follows:

Use: Marks the current mesh for all solution setups as invalid. This will force the next simulation to begin with the initial mesh.

Command: **Q3D Extractor** or **2D Extractor>Analysis Setup>Revert to Initial Mesh**

Syntax:
```
RevertAllToInitial
```

Return Value: None

Example:
```
oModule.RevertAllToInitial
```

RevertSetupToInitial

Use: Marks the current mesh for a solution setup as invalid. This will force the next simulation to begin with the initial mesh.

Command: Right-click a setup in the project tree, and then click **Revert to Initial Mesh** on the shortcut menu.

Syntax: `RevertSetupToInitial <SetupName>`

Return Value: None

Parameters:

```
<SetupName>
   The name of the solution setup you want to revert to.
```

Example:

```
oModule.RevertSetupToInitial "Setup1"
```

SetMPIVendor

The documented command is applicable for Q3D Extractor.

Use: Specify vendor name.

Syntax: **`SetMPIVendor`** `<vendorname>, <product name>`

Return Value: None

Parameters:

```
<vendorname>
   Type: <String>
   Name of MPI vendor. Possible values are "Intel", "Plat-
   formComputing"
<product name>
   Type: <String>
   Valid value is "Q3D Extractor". It is optional input,
   if not specified the default value is "Q3D Extractor"
```

```
oAnsoftApp.SetMPIVendor "Intel", "Q3D Extractor"
```

17 Optimetrics Module Script Commands

Optimetrics script commands should be executed by the `"Optimetrics"` module.

```
Set oModule = oDesign.GetModule("Optimetrics")
oModule.CommandName <args>
```


Conventions Used in this Chapter

`<VarName>`
Type: <string>
Name of a variable.

`<VarValue>`
Type: <string>
Value with unit (i.e., <value>, but cannot be an expression).

`<StartV>`
Type: <VarValue>
The starting value of a variable.

<StopV>
Type: <VarValue>
The stopping value of a variable.

<MinV>
Type: <VarValue>
The minimum value of a variable.

<MaxV>
Type: <VarValue>
The maximum value of a variable.

<IncludeVar>
Type: <bool>
Specifies whether the variable is included in the analysis.

```
<StartingPoint>
   Array("NAME:StartingPoint",  "<VarName>:=",
      <VarValue>, .... "<VarName>:=", <VarValue>)
```

<SaveField>
Type: <bool>
Specifies whether HFSS will remove the non-nominal field solution.

<MaxIter>
Type: <int>
Maximum iteration allowed in an analysis.

<PriorSetup>
Type: <string>
The name of the embedded parametric setup.

<Precede>
Type: <bool>

If true, the embedded parametric setup will be solved before the analysis begins.
If false, the embedded parametric setup will be solved during each iteration of the analysis.

```
<Constraint>
   Array("NAME:LCS",
      "lc:=", Array("<VarName>:=",
        <Coeff>, …"<VarName>:=", <Coeff>,  "rel:=",
        <Cond>, "rhs:=", <Rhs>), ...
      "lc:=", Array("<VarName>:=", <Coeff>, ..."
        <VarName>:=", <Coeff>, "rel:=", <Cond>, "rhs:=",
        <Rhs>))
```

<Coeff>
Type: <double>
Coefficient for a variable in the linear constraint.

<Cond>
Type: <string>
Inequality condition.

<Rhs>
Type: <double>
Inequality value.

```
<OptiGoalSpec>
   "Solution:=", <Soln>, "Calculation:=", <Calc>,
   "Context:=, <Geometry>
   Array("NAME:Ranges",
      "Range:", Array("Var:=",
        <VarName>, "Type:=", <RangeType>, "Start:=",
        <StartV>, "Stop:=", <StopV>), ...
      "Range:", Array("Var:=", <VarName>, "Type:=",
        <RangeType>, "Start:=", <StartV>, "Stop:=",
```

<StopV>))

<Soln>
Type: <string>
Name of the HFSS solution.

<Calc>
Type: <string>
An expression that is composed of a basic solution quantity and an output variable.

<ContextName>
Type: <string>
Name of context needed in the evaluation of <Calc>.

<Geometry>
Type: <string>
Name of geometry needed in the evaluation of <Calc>.

<RangeType>
Type: <string>
if "r", start and stop values specify a range for the variable.
if "s", start values specify the single value for the variable.

General Commands Recognized by the Optimetrics Module

Following are general script commands recognized by the **Optimetrics** module:

CopySetup
DeleteSetups (Optimetrics)
DistrbutedAnalyzeSetup
ExportDXConfigFile
ExportOptimetricsProfile
ExportOptimetricsResult
ExportParametricResults
GetSetupNames (Optimetrics)
GetSetupNamesByType (Optimetrics)
ImportSetup
PasteSetup
RenameSetup (Optimetrics)
SolveSetup (Optimetrics)
SolveAllSetup

CopySetup

Use:	Copy the specified Optimetrics setup.
Command:	None
Syntax:	`CopySetup <SetupName>`
Return Value:	None
Parameters:	`SetupName` Type: <string>
Example:	`oModule.CopySetup "OptimizationSetup1"`

DeleteSetups [Optimetrics]

Use:	Deletes the specified Optimetrics setups.
Command:	Right-click the setup in the project tree, and then click **Delete** on the shortcut menu.
Syntax:	`DeleteSetups <NameArray>`
Return Value:	None
Parameters:	`<NameArray>` Type: Array of strings. An array of setup names.
Example:	

```
oModule.DeleteSetups Array("OptimizationSetup1")
```

DistributedAnalyzeSetup

Use: Distributes all variable value instances within a parametric sweep to different machines already specified from within the user interface

Command: Right-click the parametric setup name in the project tree and select Distribute Analysis.

Syntax: `DistributedAnalyzeSetup <ParametricSetupName>`

Return Value: None

Parameters: <ParametricSetupName>

Type: <string>

Example:

```
oModule.DistributedAnalyzeSetup "ParametricSetup1"
```

ExportDXConfigFile

Use: Create an xml file with the setup information for Design Xplorer.

Command: Right click on the Design Xplorer setup in the project tree and choose **Export External Connector Addin Configuration…**

Syntax: `ExportDXConfigFile <SetupName>, <FileName>`

Return Value: None

Parameters: `<SetupName>`

Type: <string>

Must be one of existing DesignExplorer setup names.

`<FileName>`

Type : <string: file path>

Must be a valid file path and name.

Example:

```
oModule.ExportDXConfigFile "DesignXplorerSetup1",
"c:/exportdir/DXSetup1.xml")
```

ExportOptimetricsProfile

Use: Export Optimetrics profile data.

Command: Right click on the Optimetrics setup in the project tree and choose **View Analysis Result…** On the **Post Analysis Display** dialog, click the **Profile** tab and click on the **Export** button.

Syntax: `ExportOptimetricsProfile <SetupName>, <FileName>, [profileNum]`

Return Value: None

Parameters:

`<SetupName>`

Type: <string>

Must be one of the existing Parametric, Optimization, Sensitivity, Statistical or DesignXplorer setup names.

`<FileName>`

Type : <string: file path>

Must be a valid file path and name with extension of csv, tab, dat, or txt..

`[profileNum]`

Type : <string>

Must be a numeric string. Optional: defaulted to last profile number. It should be a zero indexed profile number.

Example:

```
oModule.ExportOptimetricsProfile "StatisticalSetup1",
"c:/exportdir/test.csv"
```

ExportOptimetricsResult

Use: Export an existing Optimization, Sensitivity, Statistical or DesignXplorer result. (Does not export Parametric results.)

Command: Right click on the desired Optimetrics setup in the project tree and choose **View Analysis Result...** On the **Post Analysis Display** dialog, click the **Result** tab, then select **Table** view, and click on the **Export** button.

Syntax:

```
ExportOptimetricsProfile <SetupName>, <FileName>,
[useFullOutputName]
```

Return Value: None

Parameters:

`<SetupName>`

Type: <string>

Must be one of the existing Optimization, Sensitivity, Statistical, or DesignXplorer setup names.

`<FileName>`

Type : <string: file path>

Must be a valid file path and name with extension of csv, tab, dat, or txt..

`[useFullOutputName]`

Type : <boolean>

Optional: defaulted to false. If set to true values will be printed with units. This parameter is ignored for Optimization and Statistical results.

Example:

```
oModule. ExportOptimetricsResult "StatisticalSetup1",
"c:/exportdir/test.csv", false
```

ExportParametricResults

Use: Export existing Parametric results.

Command: Right click on the desired Parametric setup in the project tree and choose **View Analysis Result...** On the **Post Analysis Display** dialog, click the **Result** tab, then select **Table** view, and click on the **Export** button.

Syntax:
```
ExportParametricResults <SetupName>, <FileName>,
<bOutputUnits>
```

Return Value: None

Parameters:

`<SetupName>`

Type: <string>

Must be one of the existing Parametric setup names.

`<FileName>`

Type : <string: file path>

Must be a valid file path and name with extension of csv, tab, dat, or txt..

`<bOutputUnits>`

Type : <boolean>

If set to true values will be printed with units.

Example:
```
oModule. ExportParametricResults "ParametricSetup1", "c:/
exportdir/test.csv", false
```

For Designer, the command details are as follows:

Use: Exports the results of a parametric sweep to a file.

Command: Right-click the setup in the project tree, and then click **Export Parametric Results** on the shortcut menu.

Syntax: ExportParametricResults ([in] BSTR theName, [in] BSTR filename, [in] BOOL bOutputUnits);

Return Value: None

Parameters:

`BSTR theName:`

Name of the parametric setup (a.k.a. ParametricSetup1)

`BSTR filename:`

the file the data will be exported to

`BOOL bOutputUnits:`

if set to TRUE, data will be exported formatted with units (a.k.a. 1pF).

If FALSE, data will be raw numbers (a.k.a. 1e-12).

Example:
```
Set oModule = oDesign.GetModule("Optimetrics")
```

```
oModule.ExportParametricResults "Table1", "C:\Proj-
ects\Temp\testexport_NoUnits.csv", FALSE
oModule.ExportParametricResults "Table1", "C:\Proj-
ects\Temp\testexport.csv", TRUE
```

GetSetupNames [Optimetrics]

Use: Gets a list of Optimetrics setup names.

Syntax: `GetSetupNames()`

Return Value: Collection of Optimetrics setup names

Parameters: `None`

Example:

```
For each name in
oModule.GetSetupNames()
Msgbox name
Next
```

Example:

```
oModule = oDesign.GetModule("Optimetrics")
set projects = oModule.GetSetupNames()
numprojects = projects.Count
```

GetSetupNamesByType [Optimetrics]

Use: Gets a list of Optimetrics setup names by type.

Syntax: `GetSetupNamesByType(<Optimetrics type>)`

Return Value: Collection of Optimetrics setup names of the given type.

Parameters: `<Optimetrics type>`

Type: String

Examples: parametric, optimization, statistical, sensitivity

Example:

```
For each name in
oModule.GetSetupNamesByType("optimization")
Msgbox name
Next
```

ImportSetup

Use: Import an Optimetric setup from a file.

Command: None

Syntax: `ImportSetup <SetupTypeName>, <SetupInfo>`

Return Value: None

Parameters: `SetupTypeName`

Type: <string>

Must be one of "OptiParametric" , "OptiOptimization", "OptiSensitivity", "OptiStatistical", or "OptiDesignExplorer".

```
<SetupInfo>
   Array("NAME:<SetupName>", "FilePath")
   <SetupName>
```

Type: <string>

Name of the setup.

```
   FilePath
```

Type : <string: file path>

Must be a valid file path and name.

Example:

```
oModule.ImportSetup "OptiStatistical",
Array("NAME:StatisticalSetup1",
"c:/importdir/mySetupInfoFile")
```

PasteSetup [Optimetrics]

The documented command is applicable for Designer.

Use: Pastes the specified Optimetrics setup.

Command: None

Syntax: SolveSetup <SetupName>

Return Value: None

Parameters: SetupName

Type: <string>

Example:

```
oModule.PasteSetup "OptimizationSetup1" "MyOptimization"
```

RenameSetup [Optimetrics]

Use: Renames the specified Optimetrics setup.

Command: Right-click the setup in the project tree, and then click **Rename** on the shortcut menu.

Syntax:

```
RenameSetup <OldName> <NewName>
```

Return Value: None

Parameters:

```
<OldName>
```

Type: <string>

```
<NewName>
```

Type: <string>

Example:

```
oModule.RenameSetup "OptimizationSetup1" "MyOptimization"
```

SolveSetup [Optimetrics]

Use: Solves the specified Optimetrics setup.

Command: Right-click the setup in the project tree, and then click **Analyze** on the shortcut menu.

Syntax: `SolveSetup <SetupName>`

Return Value: None

Parameters: `oModule.SolveSetup "OptimizationSetup1"`

SolveAllSetup

Use: Solves all Optimetrics setups.

Command: None

Syntax: SolveAllSetup <SetupName>

Return Value: None

Parameters: SetupName

Type: <string>

Example: `oModule.SolveAllSetup "OptimizationSetup1"`

Parametric Script Commands

EditSetup [Parametric]

Use: Modifies an existing parametric setup.

Command: Right-click the setup in the project tree, and then click **Properties** on the shortcut menu.

Syntax: `EditSetup <SetupName>, <ParametricParams>`

Return Value: None

GenerateVariationData (Parametric)

Use: Generate variation data before parametric solve for CAD integrated project

Command: Right click on the parametric setup in the project tree and choose "Generate Variation Data"

Syntax: `GenerateVariationData <SetupName>`

Return Value: None

Parameters: `<SetupName>`

Name of th esetup.

Example:

```
oModule.GenerateVariationData "ParametricSetup1"
```

InsertSetup [Parametric]

Use: Inserts a new parametric setup.

Command: Right-click the **Optimetrics** folder in the project tree, and then click **Add> Parametric** on the shortcut menu.

Syntax: `InsertSetup "OptiParametric", <ParametricParams>`

Return Value: None

Parameters:

```
<Parametric Params>
   Array("NAME:<SetupName>", "SaveFields:=",
   <SaveField>, <StartingPoint>, "Sim. Setups:=",
   <SimSetups>,
      <SweepDefs>, <SweepOps>,
      Array("NAME:Goals", Array("NAME:Goal",
```

```
      <OptiGoalSpec>), ... Array("NAME:Goal",
      <OptiGoalSpec>))
```

<SetupName>
Type: <string>
Name of the parametric setup.

<SimSetups>
Type: Array of strings.
An array of HFSS or Q3D Extractor or Designer solution setup names.

```
<SweepDefs>
   Array("NAME:Sweeps",
      Array("NAME:SweepDefinition", "Variable:=",
         <VarName>, "Data:=", <SweepData>,
         "Synchronize:=", <SyncNum>), ...
      Array("NAME:SweepDefinition", "Variable:=",
         <VarName>, "Data:=", <SweepData>,
         "Synchronize:=", <SyncNum>))
```

```
<SweepData>
   "<SweepType>, <StartV>, <StopV>, <StepV>"
```

<SweepType>
Type: <string>
The type of sweep data.

<SyncNum>
Type: <int>
SweepDatas with the same value are synchronized.

```
<SweepOps>
   Array("NAME:Sweep Operations", "<OpType>:=,
      Array(<VarValue>, …, <VarValue>), ...
      <OpType>:=, Array(<VarValue>, …, <VarValue>))
```

<OpType>
Type: <string>
The sweep operation type.

Example:

```
oModule.InsertSetup "OptiParametric",
   Array("NAME:ParametricSetup1", _
      "SaveFields:=", true,  _
   Array("NAME:StartingPoint"), _
      "Sim. Setups:=", Array("Setup1"),_
   Array("NAME:Sweeps", _
      Array("NAME:SweepDefinition", _
         "Variable:=", "$width", _
         "Data:=", "LIN 12mm 17mm 2.5mm", _
         "OffsetF1:=", false, _
         "Synchronize:=",  0),
      Array("NAME:SweepDefinition", _
         "Variable:=", "$length", _
         "Data:=",  "LIN 8mm 12mm 2mm", _
         "OffsetF1:=", false, _
         "Synchronize:=", 0)),
      Array("NAME:Sweep Operations"), _
   Array("NAME:Goals", _
      Array("NAME:Goal", _
         "Solution:=",  "Setup1 : LastAdaptive", _
         "Calculation:=", "returnloss", _
         "Context:=", "",_
       Array("NAME:Ranges", _
        "Range:=", Array("Var:=", "Freq", "Type:=", "s", _
          "Start:=", "8GHz", "Stop:=", "8GHz"))), _
      Array("NAME:Goal", _
         "Solution:=",  "Setup1 : LastAdaptive",_
         "Calculation:=", "reflect", _
         "Context:=", "", _
         Array("NAME:Ranges", _
```

```
        "Range:=", Array("Var:=", "Freq", "Type:=", "s", _
          "Start:=", "8GHz", "Stop:=", "8GHz")))))
```

Optimization Script Commands

EditSetup [Optimization]

Use: Modifies an existing optimization setup.

Command: Right-click the setup in the project tree, and then click **Properties** on the shortcut menu.

Syntax:
```
EditSetup <SetupName>, <OptimizationParams>
```

Return Value: None

InsertSetup [Optimization]

Use: Inserts a new optimization setup.

Command: Right-click the **Optimetrics** folder in the project tree, and then click **Add>Optimization** on the shortcut menu.

Syntax:
```
InsertSetup "OptiOptimization", <OptimizationParams>
```

Return Value: None

Parameters:
```
<OptimizationParams>
   Array("NAME:<SetupName>", "SaveFields:=",
   <SaveField>,  <StartingPoint>, "Optimizer:=",
   <Optimizer>,
   "MaxIterations:=", <MaxIter>, "PriorPSetup:=",
   <PriorSetup>, "PreSolvePSetup:=", <Preceed>,
   <OptimizationVars>, <Constraint>,
   Array("NAME:Goals", Array("NAME:Goal",
      <OptiGoalSpec>, <OptimizationGoalSpec>), ...
      Array("NAME:Goal", <OptiGoalSpec>,
      <OptimizationGoalSpec>)),
   "Acceptable_Cost:=", <AcceptableCost>, "Noise:=",
   <Noise>, "UpdateDesignWhenDone:=", <UpdateDesign"

<OptimizationVars>
   Array("NAME:Variables", "VarName:=", Array("i:=",
      <IncludeVar>, "Min:=", <MinV>, "Max:=", <MaxV>,
      "MinStep:=", <MinStepV>, "MaxStep:=", <MaxStepV>),
```

```
    ……. "VarName:=", Array("i:=", <IncludeVar>,
    "Min:=", <MinV>, "Max:=", <MaxV>,
  "MinStep:=", <MinStepV>, "MaxStep:=", <MaxStepV>))
```

<MinStepV>

Type : <VarValue>

The minimum step of the variable.

<MaxStepV>

Type: <VarValue>

The maximum step of the variable.

<AcceptableCost>

Type: <double>

The acceptable cost value for the optimizer to stop.

<Noise>

Type: <double>

The noise of the design.

<UpdateDesign>

Type: <bool>

Specifies whether or not to apply the optimal variation to the design after the optimization is done.

```
<OptimizationGoalSpec>
   "Condition:=", <OptimizationCond>,
   Array("NAME:GoalValue", "GoalValueType:=",
   <GoalValueType>,
   "Format:=", <GoalValueFormat>, "bG:=",
   Array("v:=", <GoalValue>)), "Weight:=", <Weight>)
```

<OptimizationCond>

Type: <string>

Either "<=", "==", or ">="

<GoalValueType>
Type: <string>
Either "Independent" or "Dependent"

<GoalValueFormat>
Type:<string>
Either "Real/Imag" or "Mag/Ang".

<GoalValue>
Type: <string>
Value in string. Value can be a real number, complex number, or expression.

Example:

```
oModule.InsertSetup "OptiOptimization",_
  Array("NAME:OptimizationSetup1", _
    "SaveFields:=",  false, _
    Array("NAME:StartingPoint", "$length:=", "8mm", _
    "$width:=", "14.5mm"), _
    "Optimizer:=",  "Quasi Newton", _
    "MaxIterations:=", 100, _
    "PriorPSetup:=", "ParametricSetup1", _
    "PreSolvePSetup:=",  true, _
    Array("NAME:Variables", _
      "$length:=", Array("i:=", true, "Min:=", "6mm", _
        "Max:=", "18mm", _
        "MinStep:=", "0.001mm", "MaxStep:=", _
        "1.2mm"), _
      "$width:=", Array("i:=",  true, "Min:=", _
        "6.5mm", "Max:=", "19.5mm", _
        "MinStep:=", "0.001mm", "MaxStep:=", _
        "1.3mm")), _
    Array("NAME:LCS"), _
    Array("NAME:Goals", _
      Array("NAME:Goal", _
        "Solution:=",  "Setup1 : LastAdaptive", _
        "Calculation:=", "reflect", _
```

```
        "Context:=", "", _
        Array("NAME:Ranges", _
        "Range:=", Array("Var:=",  "Freq", _
        "Type:=", "s", _
        "Start:=", "8GHz", "Stop:=", "8GHz")), _
        "Condition:=",  "<=", _
        Array("NAME:GoalValue", _
        "GoalValueType:=", "Independent", _
        "Format:=",  "Real/Imag", _
        "bG:=", Array("v:=", "[0.0001]")), _
        "Weight:=", "[1]")),
    "Acceptable_Cost:=",  0.0002, _
    "Noise:=", 0.0001, _
    "UpdateDesign:=", true, _
    "UpdateIteration:=", 5, _
    "KeepReportAxis:=",  true, _
    "UpdateDesignWhenDone:=", true)
```

Sensitivity Script Commands

EditSetup [Sensitivity]

Use: Modifies an existing sensitivity setup.

Command: Right-click the setup in the project tree, and then click **Properties** on the shortcut menu.

Syntax:
```
EditSetup <SetupName>, <SensitivityParams>
```

Return Value: None

InsertSetup [Sensitivity]

Use: Inserts a new sensitivity setup.

Command: Right-click **Optimetrics** in the project tree, and then click **Add>Sensitivity** on the shortcut menu.

Syntax:
```
InsertSetup "OptiSensitivity", <SensitivityParams>
```

Return Value: None

Parameters:
```
<SensitivityParams>
   Array("NAME:<SetupName>",
   "SaveFields:=", <SaveField>,
   <StartingPoint>,
   "MaxIterations:=", <MaxIter>,
   "PriorPSetup:=", <PriorSetup>,
   "PreSolvePSetup:=", <Preceed>,  <SensitivityVars>,
   <Constraint>,

   Array("NAME:Goals",
   Array("NAME:Goal", <OptiGoalSpec>), ...,
   Array("NAME:Goal", <OptiGoalSpec>)),
   "Master Goal:=". <MasterGoalID>,
   "MasterError:=", <MasterError>)

<SensitivityVars>
   Array("NAME:Variables",
      "VarName:=", Array("i:=", <IncludeVar>,
         "Min:=", <MinV>, "Max:=", <MaxV>,
```

```
        "IDisp:=", <InitialDisp>),...
      "VarName:=", Array("i:=", <IncludeVar>,
        "Min:=", <MinV>, "Max:=", <MaxV>,
        "IDisp:=", <InitialDisp>))
```

<InitialDisp>
Type : <VarValue>
The initial displacement of the variable.

<MasterGoalID>
Type: <int>
Index of the master goal. Index starts from zero.

<MasterError>
Type: <double>
Error associated with the master goal.

Example:

```
oModule.InsertSetup "OptiSensitivity", _
  Array("NAME:SensitivitySetup1",_
    "SaveFields:=", true,_
    Array("NAME:StartingPoint"), _
    "MaxIterations:=", 20,_
    "PriorPSetup:=", "",_
    "PreSolvePSetup:=",  true, _
    Array("NAME:Variables"), _
    Array("NAME:LCS"),_
    Array("NAME:Goals",_
      Array("NAME:Goal", _
        "Solution:=", "Setup1 : LastAdaptive",_
        "Calculation:=", "returnloss",_
        "Context:=", "",_
        Array("NAME:Ranges",_
        "Range:=", Array("Var:=", "Freq", "_
        Type:=", "s",_
        "Start:=", "8GHz", "Stop:=", "8GHz"))),_
```

```
            Array("NAME:Goal",_
               "Solution:=", "Setup1 : LastAdaptive",_
               "Calculation:=", "reflect",_
               "Context:=", "",_
               Array("NAME:Ranges",_
               "Range:=", Array("Var:=","Freq",_
               "Type:=", "s",_
               "Start:=", "8GHz", "Stop:=", "8GHz")))),_
         "Master Goal:=",  1,_
         "MasterError:=", 0.001)
```

Statistical Script Commands

EditSetup [Statistical]

InsertSetup Statistical

EditSetup [Statistical]

Use: Modifies an existing statistical setup.

Command: Right-click the setup in the project tree, and click **Properties** on the shortcut menu.

Syntax: `EditSetup <SetupName>, <StatisticalParams>`

Return Value: None

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 13.0.0
' 3:04:05 PM  May 18, 2010
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("optiguides")
```

```
Set oDesign = oProject.SetActiveDesign("HFSSModel1")
Set oModule = oDesign.GetModule("Optimetrics")
oModule.EditSetup "StatisticalSetup1", Array("NAME:StatisticalSet-
up1",
Array("NAME:ProdOptiSetupData",
"SaveFields:=", true, "CopyMesh:=", false),
Array("NAME:StartingPoint", "$length:=", "7.824547736mm",
"$width:=", "14.8570192mm"),
"MaxIterations:=", 50,
"PriorPSetup:=", "",

Array("NAME:Variables",
"$length:=", Array("i:=", true,
"int:=", false,
"Dist:=", "Uniform",
"Tol:=", "10%", "StdD:=", "0.2mm", "Min:=", "-3",
"Max:=", "3", "Shape:=", "1", "Scale:=", "0.04mm",
"Location:=", "0.4mm",
"Dataset:=", "", "LatinHypercube:=", "true", "VarMin:=", "0.2mm",
"VarMax:=", "0.6mm", "Prob:=", "0.01",
"Mean:=", "0.4mm"),

"$width:=", Array("i:=", true,
"int:=", false,
"Dist:=", "Gaussian",
"Tol:=", "10%",
"StdD:=", "0.2mm",
"Min:=", "-3", "Max:=", "3",
"Shape:=", "1",
"Scale:=", "0.04mm",
"Location:=", "0.4mm",
"Dataset:=", "",
"LatinHypercube:=", "true",
"VarMin:=", "0.2mm", "VarMax:=", "0.6mm",
"Prob:=", "0.02",
"Mean:=", "0.4mm")),

Array("NAME:Goals", Array("NAME:Goal",
"ReportType:=", "Modal Solution Data",
"Solution:=", "Setup1 : PortOnly",
Array("NAME:SimValueContext", "Domain:=", "Sweep"),
"Calculation:=", "returnloss",
```

```
"Name:=", "returnloss",

Array("NAME:Ranges",
"Range:=", Array("Var:=", "Freq",
"Type:=", "s",
"Start:=", "8.2GHz", "Stop:=", "0"))),

Array("NAME:Goal",
"ReportType:=", "Modal Solution Data",
"Solution:=", "Setup1 : PortOnly",
Array("NAME:SimValueContext",
"Domain:=", "Sweep"),
"Calculation:=", "reflect",
"Name:=", "reflect",
Array("NAME:Ranges",
"Range:=", Array("Var:=", "Freq",
"Type:=", "s",
"Start:=", "8.2GHz", "Stop:=", "0")))))
```

InsertSetup [Statistical]

Use: Inserts a new statistical setup.

Command: Right-click **Optimetrics** in the project tree, and then click **Add>Statistical** on the shortcut menu.

Syntax:
```
InsertSetup "OptiStatistical", <StatisticalParams>
```

Return Value: None

Parameters:
```
<StatisticalParams>
   Array("NAME:<SetupName>",
   "SaveFields:=", <SaveField>,
   <StartingPoint>,
   "MaxIterations:=",
   <MaxIter>,
   "PriorPSetup:=", <PriorSetup>,
   "PreSolvePSetup:=", <Preceed>,
   <StatisticalVars>,
   Array("NAME:Goals",
   Array("NAME:Goal", <OptiGoalSpec>), …,
   Array("NAME:Goal", <OptiGoalSpec>))),


<StatisticalVars>
   Array("NAME:Variables",
```

```
"VarName:=", Array("i:=", <Boolean>,
"int:=", <Boolean>,
"Dist:=", <DistType>,
"Tol:=", <Tolerance>,
   "StdD:=", <StdD>,
   "Min:=", <MinCutoff>,
   "Max:=", <MaxCutoff>, ...
   "Shape:=", "<VarParameter>",
   "Scale:=", "<VariableParameter>",
   "Location:=", "<VariableParameter>",
   "Dataset:=", "",
   "LatinHypercube:=", "<Boolean>",
   "VarMin:=", "<VariableValue>",
   "VarMax:=", "<VariableValue>",
   "Prob:=", "<Probability>",
   "Mean:=", "<Mean>")
)
```

`<DistType>`

Type : <string>

Distrbution can be `"Gaussian"`,`"Uniform"`, “LogNormal” or “UserDefined.”

`<Tolerance>`

Type: <VarValue>

The tolerance for the variable when distribution is `Uniform`.

`<StdD>`

Type: <VarValue>

The standard deviation for the variable when distribution is `Gaussian`.

`<MinCutoff>`

Type: <double>

Formerly the minimum cut-off for the variable when distribution is `Gaussian`. Legacy.

`<MaxCutoff>`

Type: <double>

Formerly the maximum cut-off for the variable when distribution is `Gaussian`. Leg-

acy.

<Prob>

Type: <double>

Probability for Gaussian distribution. A value >= 0 and < 0.1.

<Mean>

Type: <double>

Mean for Gaussian distribution.

Example:

```
' ----------------------------------------------
' Script Recorded by Ansoft HFSS Version 13.0.0
' 3:29:39 PM  May 18, 2010
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("OptimTee")
Set oDesign = oProject.SetActiveDesign("TeeModel")
oDesign.ChangeProperty Array("NAME:AllTabs",
Array("NAME:LocalVariableTab",
Array("NAME:PropServers", "LocalVariables"),
Array("NAME:ChangedProps",
Array("NAME:offset",
Array("NAME:Statistical", "Included:=", true)))))
Set oModule = oDesign.GetModule("Optimetrics")
oModule.InsertSetup "OptiStatistical", Array("NAME:StatisticalSet-
up1",
Array("NAME:ProdOptiSetupData",
"SaveFields:=", false, "CopyMesh:=", false),
Array("NAME:StartingPoint", "offset:=", "0in"),
```

```
"MaxIterations:=", 50, "PriorPSetup:=", "",

Array("NAME:Variables",

"offset:=", Array("i:=", true,
"int:=", false,
"Dist:=", "Gaussian",
"Tol:=", "10%",
"StdD:=", ".5in",
"Min:=", "-3",
"Max:=", "3",
"Shape:=", "1",
"Scale:=", "0in",
"Location:=", "0in",
"Dataset:=", "",
"LatinHypercube:=", "true",
"VarMin:=", "-1in",
"VarMax:=", "1in",
"Prob:=", "0.01",
"Mean:=", "0in")),

Array("NAME:Goals", Array("NAME:Goal",
"ReportType:=", "Modal Solution Data",
"Solution:=", "Setup1 : LastAdaptive", Array("NAME:SimValueContext"),
"Calculation:=", "Power11",
"Name:=", "Power11",
Array("NAME:Ranges",
"Range:=", Array("Var:=", "Freq", "Type:=", "d",
"DiscreteValues:=", "10GHz")))))
```

For Q3D Extractor and Designer the command details are as follows:

Use: Inserts a new statistical setup.

Command: Right-click **Optimetrics** in the project tree, and then click **Add>Statistical** on the shortcut menu.

Syntax: `InsertSetup "OptiStatistical", <StatisticalParams>`

Return Value: None

Parameters:

```
<StatisticalParams>
   Array("NAME:<SetupName>", "SaveFields:=",
   <SaveField>,  <StartingPoint>, "MaxIterations:=",
   <MaxIter>, "PriorPSetup:=", <PriorSetup>,
```

```
    "PreSolvePSetup:=", <Preceed>,  <StatisticalVars>,
    Array("NAME:Goals", Array("NAME:Goal",
    <OptiGoalSpec>), …, Array("NAME:Goal",
    <OptiGoalSpec>))),
  <StatisticalVars>
    Array("NAME:Variables",
      "VarName:=", Array("i:=", <IncludeVar>, "Dist:=",
        <DistType>, "Tol:=", <Tolerance>,
        "StdD:=", <StdD>, "Min:=", <MinCutoff>, "Max:=",
        <MaxCutoff>, ...
      "VarName:=", Array("i:=", <IncludeVar>, "Dist:=",
        <DistType>, "Tol:=", <Tolerance>, "StdD:=",
        <StdD>, "Min:=", <MinCutoff>, "Max:=",
        <MaxCutoff>))
    Parameters:
    <DistType>
      Type : <string>
      Distrbution can be "Gaussian" or "Uniform".
    <Tolerance>
      Type: <VarValue>
      The tolerance for the variable when distribution is
      Uniform.
    <StdD>
      Type: <VarValue>
      The standard deviation for the variable when distri-
      bution is Gaussian.
    <MinCutoff>
      Type: <double>
      The minimum cut-off for the variable when distribu-
      tion is Gaussian.
    <MaxCutoff>
      Type: <double>
      The maximum cut-off for the variable when distribu-
      tion is Gaussian.
```

Example:

```
oModule.InsertSetup "OptiStatistical", _
  Array("NAME:StatisticalSetup1", _
```

```
"SaveFields:=",  true, _
Array("NAME:StartingPoint"),_
"MaxIterations:=", 50,_
"PriorPSetup:=", "", _
Array("NAME:Variables"), _
Array("NAME:Goals", _
  Array("NAME:Goal", _
    "Solution:=", "Setup1 : LastAdaptive", _
    "Calculation:=", "returnloss", _
    "Context:=", "", _
    Array("NAME:Ranges", _
    "Range:=", Array("Var:=", "Freq", _
    "Type:=", "s",_
    "Start:=", "8GHz", "Stop:=", "8GHz"))),_
  Array("NAME:Goal",_
    "Solution:=", "Setup1 : LastAdaptive",_
    "Calculation:=", "reflect",_
    "Context:=", "", _
    Array("NAME:Ranges",_
    "Range:=", Array("Var:=", "Freq", "Type:=", _
    "s", "Start:=", "8GHz", "Stop:=", "8GHz"))))
```

18 Solutions Module Script Commands

Solutions commands should be executed by the `"Solutions"` module.

```
Set oModule = oDesign.GetModule("Solutions")
oModule.CommandName <args>
```

DeleteImportData
EditSources
DeleteSolutionVariation
ExportForSpice
ExportEigenmodes
ExportForHSpice
ExportNetworkData
ExportNMFData
GetAdaptiveFreq
GetAvailableVariations
GetExcitationScaling
GetISolutionVersionID
GetSolveRangeInfo
GetValidSolutionList
HasFields
HasMatrixData
HasMesh
ImportSolution
ImportTable

IsFieldAvailableAt
ListMatchingVariations
ListValuesOfVariables
ListVariations

DeleteImportData

Use: Deletes imported solution or table data in HFSS. Not in HFSS-IE

Command: **HFSS>Results>Import Solutions**

Syntax:
```
DeleteImportData <ImportSpecArray>
```

Return Value: None

Parameters:
```
<ImportSpecArray>
   Array(<ImportSpec>, ...)

<ImportSpec>
```
Type: <string>

Format of string is `"importname:solnnameORtablename"`.

Example:
```
oModule.DeleteImportData _
   Array("Import1:Adaptive_1", "Import2:DataTable")
```

EditSources

Use: Indicates which source excitations should be used for fields post processing.

Command: **HFSS>Fields>Edit Sources**

Syntax:
```
EditSources<FieldType>, <SourceArray>,
   <MultiplicityArray>, <MagnitudeArray>,
   <PhaseArray>, <TerminatedArray>,
   <ImpedanceArray>, <IncludePostProcessing>
```

Return Value: None

Parameters:
```
<FieldType>
```
Type: <string>

Possible values are:

`"NoIncidentWave", "ScatteredFields", "TotalFields", "IncidentFields", "EigenStoredEnergy", or "EigenPeakElectricField"`

```
<SourceArray>
   Array("NAME:SourceNames", <Source1Name>,
```

```
   <Source2Name>, ...)
```

A source name is typically the name of the associated excitation.

```
<MultiplicityArray>
   Array("NAME:Modes", <port1NumModes>, <port2NumModes>,
   ...)
```

or

```
   Array("NAME:Terminals", <port1NumTerminals>, ...)
```

A non-port source should indicate multiplicity of 1.

```
<MagnitudeArray>
   Array("NAME:Magnitudes", <Source1Mag>, <Source2Mag>,
   ...)
```

This gives the Mag of the complex excitation for each source.

```
<PhaseArray>
   Array("NAME:Phases", <Source1Phase>, <Source2Phase>,
   ...)
```

This gives the Phase in degrees of the complex excitation for each source. For Eigen-moded stored energy problems, phase array is not recorded by **Tools>Record Script to File**, , and is not needed when running scripts for such problems.

```
<TerminatedArray>
   Array("NAME:Terminated", <IsSource1Terminated>, ...)
```

Only used for a Driven Terminal solution-type problem.

If it is Driven Terminal, then each source must have an entry, but

entries for non port sources are ignored.

```
<ImpedanceArray>
   Array("NAME:Impedances", <Source1ComplexImped>, ...)
```

Only for a Driven Terminal solution-type problem.

If it is Driven Terminal, there must be an entry for each terminated

source. Complex format is a string representation as `"re + im j"`.

`<IncludePostProcessing>`

Type: <bool>

Specifies whether to include post processing effects.

<useIncidentVoltage>

Type: <bool>

For driven terminal projects, this specifies whether to use incident voltage. The default for this argument if missing is "false", so that legacy driven terminal projects behave as if using total voltage.

Example:

```
oModule.EditSources "NoIncidentWave", _
  Array("NAME:SourceNames", "WavePort1", _
  "WavePort2"), Array("NAME:Terminals", 2, 2), _
  Array("NAME:Magnitudes", 1, 0), _
  Array("NAME:Phases",  0, 0),_
  Array("NAME:Terminated", false, true, true, false), _
  Array("NAME:Impedances",  "50 + 80 j", "50 + 90 j")
```

Example:

```
oModule.EditSources "NoIncidentWave", _
  Array("NAME:SourceNames", "EigenMode"), _
  Array("NAME:Modes",  2), Array("NAME:Magnitudes", _
  0, 1), Array("NAME:Phases", 0, 45), _
  Array("NAME:Terminated"), Array("NAME:Impedances")
```

Example:

```
oModule.EditSources "TotalFields", _
  Array("NAME:SourceNames", "WavePort1", _
  "LumpPort1", "IncWave1", "Voltage1", "Current1"),_
  Array("NAME:Modes", 1, 1, 6, 1, 1), _
  Array("NAME:Magnitudes",  _
  17, 19, 1, 3, 5, 7, 9, 11, 13, 15), _
  Array("NAME:Phases", 0, 20, 2, 4, _
  6, 8, 10, 12, 14, 16), Array("NAME:Terminated"),_
  Array("NAME:Impedances")
```

Example:

```
oModule.EditSources "NoIncidentWave",
Array("NAME:Names", "P01__NET179_", _
"P02__NET178_", "P03__NET178_", "P04__NET179_"),
Array("NAME:Terminals", 1, 1, 1, 1),
Array("NAME:Magnitudes", "1", "0", "0", "0"),
```

```
Array("NAME:Phases", "0deg", "0deg", "0deg", "0deg"),
Array("NAME:Terminated", false, false, false, false),
Array("NAME:Impedances"), true, true
```

Example:

```
Set oDesign = oProject.SetActiveDesign("PEC_d1")
oDesign.SetSolutionType "Transient Network"
Set oModule = oDesign.GetModule("Solutions")
oModule.EditSources "ScatteredFields",
Array("NAME:Names", "IncPWave1"),
Array("NAME:Terminals", 1),
Array("NAME:Magnitudes", "1"),
Array("NAME:Phases", "0deg")
oModule.EditSources "ScatteredFields",
Array("NAME:Names", "IncPWave1"), Array("NAME:Terminals", 1),
Array("NAME:TransientMagnitudes", "1"),
Array("NAME:Delays", "0.2s")
```

For Q3D Extractor, the documented command details are as follows:

Use: Indicates which source excitations should be used for fields post processing.

Command: **Q3D Extractor** or **2D Extractor>Fields>Edit Sources**

Syntax:

```
EditSources <FieldType>, <SourceArray>,
   <MultiplicityArray>, <MagnitudeArray>,
   <PhaseArray>, <TerminatedArray>, <ImpedanceArray>
```

Return Value: None

Parameters:

```
<SourceSettings>
   "Value Type:=", <UnitType>, <SourceArray>, <ValueArray>
   Parameters:
   <UnitType>
      Type: <string>
      In Q3D Extractor, possible types for DC and AC are
      "A" and "V". The only type for Cap is "V".
      In 2D Extractor, the CG solution uses only "V" and
      the RL solution uses only "A".
   <SourceArray>
      Array("NAME:SourceNames", <SourceName>, <Source-
```

```
                    Name>, …)
                    Parameters:
                    <SourceName>
                      Type: <string>
                      Net name for capacitance. Source name for AC and
                      DC
                  <ValueArray>
                    Array("NAME:Source Values", <SourceValue>, <SourceV-
                    alue>, …)

                    Parameters:
                    <SourceValue>
                      Type: <double>
                      The value associated with a source in
                      <SourceArray>.
oModule.EditSources Array("NAME:AC", "Value Type:=", "A",
Array("NAME:Source Names", _ "TheVia:Source1"), Array("NAME:Source
Values", 1)),
Array("NAME:Cap", "Value Type:=", "N", _ Array("NAME:Source Names",
"GroundPlane", "TheVia"), Array("NAME:Source Values", 0, 1)), _
Array("NAME:DC", "Value Type:=",  "V", Array("NAME:Source Names",
"TheVia:Source1"),
Array("NAME:Source Values", 1))
```

DeleteSolutionVariation

Use: Deletes all solution data for specific solutions and design variations. This is obsolete and is supported only for backward compatibility. You should use DeleteFullVariation, DeleteFieldVariation, and DeleteLinked Variation.

Command: **HFSS>Results>Clean Up Solutions**

Syntax:

```
DeleteSolutionVariation
  Array(<DataSpecifierArray>, ...)
```

Return Value: None

Parameters:

```
<DataSpecifierArray>
  Array(<DesignVariationKey>, <SetupName>, <SolnName>)

<DesignVariationKey>
```

Type: <string>

Design variation string.

<SetupName>
Type: <string>
Name of the solution setup.

<SolnName>
Type: <string>
Name of the solutions within the solution setup.

Example:

```
oModule.DeleteSolutionVariation Array( _
   Array("width='2in'", "Setup1", "Adaptive_1") _
   Array("width='2in'", "Setup1", "Sweep1") )
```

For Q3D Extractor, the command details are as follows:

Use: Deletes matrix solution data for specific solutions and design variations.

Command: **Q3D Extractor** or **2D Extractor>Results>Clean Up Solutions**

Syntax: `DeleteSolutionVariation Array(<DataSpecifierArray>, …)`

Return Value: None

Parameters:

```
<DataSpecifierArray>
   Array(<DesignVariationKey>, <SetupName>, <SolnName>)
   Parameters:
   <DesignVariationKey>
      Type: <string>
      Design variation string.

   <SetupName>
      Type: <string>
      Name of the solve setup.
   <SolnName>
      Type: <string>
      Name of the solutions within the solve setup.
```

Example:

```
oModule.DeleteSolutionVariation
Array(Array("width='2in'", "Setup1", "Adaptive_1") _
   Array("width='2in'", "Setup1", "LastAdaptive") )
```

DeleteVariation [HFSS]

Use: Obsolete. Replaced by DeleteFullVariation, DeleteFieldVariation, and DeleteLinked Variation.

ExportForSpice

Use: Exports matrix solution data to a file in a format suitable for Spice analysis. Available only for Driven Terminal solution types with ports. Output in an appropriate format will be generated for each of the non-empty file names provided.

Command: None

Syntax:

```
ExportForSpice
<DesignVariationKey>, <SolnSelectionArray>, <SpiceType>,
<BandWidth>, <FWSFile>, <LumpedElementFile>,
<PoleZeroSpiceFile>, <PoleZeroMatlabFile>,
<PartialFractionFile>, <FittingErrorInPercent>,
<MaximumOrder>,
<UseCommonGround>,
<EnforcePassivity>
```

Return Value: None

Parameters:

`<SpiceType>`
Type: <int>
Possible values are:
0: PSpice
2: Maxwell Spice

`<BandWidth>`
Type: <int>
Possible values are:
0: Low (narrow) band width

`<FWSFile>`
Type: <string>

`<LumpedElementFile>`
Type: <string>

<PoleZeroSpiceFile>
Type: <string>

<PoleZeroMatlabFile>
Type: <string>

<PartialFractionFile>
Type: <string>
<FittingErrorInPercent>
Type: <Float>
<MaximumOrder>
Type: <Integer>
<UseCommonground>
Type: 0/1
<EnforcePassivity>
Type: 0/1

Example:

```
oModule.ExportForSpice "width='2in'", _
  Array("Setup1:Sweep1"), 2, 0, _
  "c:\mydir\Sweep1.fws", "", "", "", ""
```

ExportEigenmodes

Use: Exports a tab delimited table of Eigenmodes in HFSS. Not in HFSS-IE.

Command: None

Syntax:
```
ExportEigenmodes
<setupName> <solutionName> <DesignVariationKey>
<filename>
```

Return Value: None

Parameters:

<SolutionName>
Type: <string>
Name of the solutions within the solution setup.

<DesignVariationKey>

Type: <string>

Design variation string.

Example:

```
Set oModule = oDesign.GetModule("Solutions")
oModule.ExportEigenmodes "Setup1 : LastAdaptive", "",  _
  "C:\mydir\myeigenmode" & _
  ".eig"
```

ExportForHSpice

Use: Exports matrix solution data to a file in a format suitable for HSpice analysis. Available only for Driven Terminal solution types with ports. Output in an appropriate format will be generated for each of the non-empty file names provided.

Command: None

Syntax:

```
ExportForHSpice
<DesignVariationKey>, <SolnSelectionArray>, <SpiceType>,
<BandWidth>, <FWSFile>, <LumpedElementFile>,
<PoleZeroSpiceFile>, <PoleZeroMatlabFile>,
<PartialFractionFile>,
<FittingErrorInPercent>, <MaximumOrder>, <MinimumOrder>,
<EnforcePassivity>
```

Return Value: None

Parameters:

<SpiceType>

Type: <int>

Possible value is:

1: HSpice

<BandWidth>

Type: <int>

Possible value is:

0: Low (narrow) band width

<FWSFile>

Type: <string>

<LumpedElementFile>
Type: <string>

<PoleZeroSpiceFile>
Type: <string>

<PoleZeroMatlabFile>
Type: <string>

<PartialFractionFile>
Type: <string>

<FittingErrorInPercent>
Type: <float>
The accuracy to use in fitting the pole zero model.

<MaximumOrder>
Type: <int>
Maximum number of poles in rational function expansion.
<UseCommonGround>
Type:<init>
0/1
<EnforcePassivity>
Type: : <int>
0/1

Example:

```
oModule.ExportForHSpice "width='2in'", _
  Array("Setup1:Sweep1"), 1, 0, _
  "c:\mydir\Sweep1.fws", "", "", "", "", _
  .005, 20, 200
```

ExportNetworkData

Use: Exports matrix solution data to a file. Available only for Driven solution types with ports.

Command: None

Syntax: ExportNetworkData
<DesignVariationKey>, <SolnSelectionArray>, <FileFormat>,

`<OutFile>,<FreqsArray>, <DoRenorm>, <RenormImped>`
`<dataType> <pass> <complexFormat>`

Return Value: None

Parameters:

`<SolnSelectionArray>`
`Array(<SolnSelector>, <SolnSelector>, ...)`
If more than one array entry, this indicates a combined Interpolating sweep.
`<SolnSelector>`
Type: <string>
Gives solution setup name and solution name, separated by a colon.
`<FileFormat>`
Type: <int>
Possible values are:
2 : Tab delimited spreadsheet format (.tab)
3 : Touchstone (.sNp)
4 : CitiFile (.cit)
7 : Matlab (.m)
8 : Terminal Z0 spreadsheet

`<OutFile>`
Type: <string>
Full path to the file to write out.

`<FreqsArray>`
Type: Array of doubles.
The frequencies to export. To export all frequencies, use `Array("all")`.

`<DoRenorm>`
Type: <bool>
Specifies whether to renormalize the data before export.

`<RenormImped>`
Type: <double>
Real impedance value in ohms, for renormalization. Required in syntax, but ignored if `DoRenorm` is false.
`<DataType>`

Type: "S", "Y", or "Z"

The matrix to export.

<Pass>

Type: string, from 1 to N.

The pass to export. This is ignored if the sourceName is a frequency sweep. Leaving out this value or specifying -1 gets all passes.

<ComplexFormat>

Type: "0", "1", or "2"

The format to use for the exported data.

0 = Magnitude/Phase.

1= Real/Imaginary.

2= db/Phase.

<TouchstonePrecision>

Type: <int>

Default if not specified is 15.

Example: Export all frequencies:

```
oModule.ExportNetworkData "width='2in'", _
  Array("Setup1:Sweep1"),2, "c:\mydir\out.tab", _
  Array("all"), false, 0
```

Example: Export specific frequencies:

```
oModule.ExportNetworkData "width='2in'", _
  Array("Setup1:Sweep1", "Setup1:Sweep2"), 3, _
  "c:\mydir\out.s2p", Array(1.0e9, 1.5e9, 2.0e9), _
  true, 50.0
```

Example: Specify Precision for Touchstone output.

```
oModule.ExportNetworkData "", Array("Setup1:Sweep1"), 3,  _
  "C:/Ring_hybrid HFSSDesign.s4p", _
Array( "All"), true, 50, "S", -1, 0, 15
```

ExportNMFData [HFSS]

Use: Exports matrix solution data to a file in neutral model format. Available only for Driven solution types with ports. Variables can be held constant by setting their values in the variation field. For example: "length='50mm' width='30mm'". All other independent variables will be treated as NMF parameters.

Command: None

Syntax: ExportNMFData
<SolnSelectionArray>, <OutFile>, <FreqsArray>,
<DesignVariationKey>, <DoRenorm>, <RenormImped>

Return Value: None

Example:

```
oModule.ExportNMFData Array("Setup1:Sweep1"), _
     "c:\mydir\out.nmf", Array("all"), "", FALSE, 0
```

GetAdaptiveFreq

Use: To obtain an adaptive frequency for a specified setup.

Syntax: GetAdaptiveFreq(<SetupName>)

Return Value: Returns a frequency value.
Type: <double>
Example: "15500000000.0"

Parameters: <SetupName>
Type: <string>

Example:

```
   set oModule = oDesign.GetModule("Solutions")
adaptfreq = oModule.GetAdaptiveFreq("Setup1")
```

GetAvailableVariations

The documented command is applicable for Q3D Extractor.

Use: Returns the available variation for a solution.

Command: None

Syntax: GetAvailableVariations("<SetupName>:<SolnName>)

Return Value: An array of string object.

Parameters: <SetupName>
Type: <string>
The solve setup name.
< >
Type: <string>
The solution name ("Adaptive_1", … "Adaptive_N", "Last-Adaptive").

Example:

```
Dim oVarArray
   oVarArray = oModule.GetAvailableVariations
```

```
("Setup1 : LastAdaptive")
```

GetExcitationScaling

Use: Get source scaling parameters.

Syntax: `GetExcitationScaling("<port name>", <mode/terminal/eigenmode index>)`

Return Value: Returns a vector of strings representing source scaling parameters

Parameters:

`<portName>`

Type: <string>

`<mode/terminal/eigenmode index>`

Type: <Int>

Example:

First terminal scaling from port "1":

```
data = oModule.GetExcitationScaling("1")
or data = oModule.GetExcitationScaling("1", 1)
```

First mode scaling from port "1":

```
data = oModule.GetExcitationScaling("1", 1)
```

Second eigenmode scaling (name is irrelevant):

```
data = oModule.GetExcitationScaling("unused", 2)
```

GetISolutionVersionID

Use: To obtain the solution ID to help track solution validity.

Syntax: `GetISolutionVersionID(BSTR fullSolutionName)`

Return Value: Returns a solution ID.

Parameters: `None`

Example:

```
versionID = oModule.GetISolutionVersionID(BSTR fullSolutionName)
```

GetSolveRangeInfo

Use: To determine the frequency range of a particular simulation setup. For fast sweeps and interpolating sweeps this command returns the start and stop frequencies. For discrete sweeps, it returns a list of frequencies. For an adaptive solution, it returns the adaptive frequency.

Syntax: `GetSolveRangeInfo(<SolutionName>)`

Return Value: `An array of frequencies.`

Parameters: `<SolutionName>`

Type: `<string>`

Example:

```
set oModule = oDesign.GetModule("Solutions")
freqrange = oModule.GetSolveRangeInfo("Setup1:Sweep1")
```

GetValidISolutionList

Use: Gets all available solution names that exist in a design.

Syntax: `GetValidISolutionList(<IncludeImportedSolutions>)`

Return Value: Array of names

Parameters: `<IncludeImportedSolutions>`

`Type: <Boolean>`

`If no parameter is given the default is False.`

Example:

```
solution = oModule.GetValidISolutionList(True)
```

For Q3D Extractor, the command details are as follows:

Use: Returns a list of valid solutions for a design.

Command: None

Syntax: `GetValidISolutionList`

Return Value: An array of string object in the <SetupName>:<SolnName> format.

Example:

```
Dim oVarArray
   oVarArray = oModule. GetValidISolutionList
```

HasFields

Use: To determine if fields exist for a particular solution.

Syntax: `HasFields(<SolutionName>, <DesignVariation>)`

Return Value: Returns 1 or 0 (1= true, 0 = false)

Type: Boolean

Parameters: `<SolutionName>`

Type: `<string>`

Example: `"Setup1:LastAdaptive"`

`<DesignVariation>`

Type: `<string>`

Example: "x_size = 2mm"

Example:

```
set oModule = oDesign.GetModule("Solutions")
fieldsExist = oModule.HasFields("Setup1:Sweep1", _
"x_size=2mm")
```

HasMatrixData

Use: To determine if matrix data exists for a particular solution.

Syntax: HasMatrixData(<SolutionName>, <DesignVariation>, <matrixType>)

Return Value: Returns 1 or 0 (1= true, 0 = false)

Type: Boolean

Parameters:

<SolutionName>

Type: <string>

Example: "Setup1:LastAdpative"

<DesignVariation>

Type: <string>

Example: "radius = 4in"

<Matrix type>

Type: <string>

Example: "C, CD RL"

Example:

```
set oModule = oDesign.GetModule("Solutions")
matrixExist = 0Module.HasMatrixData("Setup1:Adaptive_1",_
"radius = 4in", "C, CD RL")
```

HasMesh

Use: To determine if a current mesh exists for a particular simulation setup, not including the initial mesh.

Syntax: HasMesh(<SetupName>, <DesignVariation>)

Return Value: Returns 1 or 0 (1= true, 0 = false)

Type: Boolean

Parameters:

<SetupName>

Type: <string>

<DesignVariation>

Type: <string>

Example:

```
set oModule = oDesign.GetModule("Solutions")
meshexist = oModule.HasMesh("Setup1", "x_size = 2in _
y_size = 1in")
```

ImportSolution

Use: Imports a matrix solution in HFSS, which can then be used in creating reports or in the display of matrix data. The imported solution need not have the same characteristics as the current design. Imported terminal data that meets the required criteria can be used for full-wave Spice export. Not used in HFSS-IE.

Command: **HFSS>Results>Import Solutions**

Syntax: `ImportSolution <FileName>, <ImportName>, <SolnArray>`

Return Value: None

Parameters:

`<FileName>`

Type: <string>

Location of the source data. The type of the data file will be determined strictly by its file extension. Supported types are Touchstone (.sNp or .yNp or .zNp or .tou), and Ansoft Designer (.flp).

`<ImportName>`

Type: <string>

Identifying name to use for the import, analogous to solution setup name.

`<SolnArray>`

Type: Array of strings

The names of the solutions selected for import from the file. The only import format supporting multiple solutions in one file is HFSS8.x format.

Example:

```
oModule.ImportSolution "c:\mydir\in.s2p", _
   "MeasuredData", Array("Sweep1")
```

ImportTable

Use: Imports a data table for use in plotting reports in HFSS. Not used in HFSS-IE. The table can have multiple independent real-valued columns of data, and multiple dependent real- or complex-valued columns of data. The data

supported imports are either tab delimited format (.tab) or comma delimited format (.csv). The first row may contain column names. Complex data columns are inferred from the column data format. In tab delimited format, `"(double, double)"` denotes a complex number. In comma delimited format, `"(double, double)"` denotes a complex number.

Command: **HFSS>Results>Import Solutions**

Syntax:

```
ImportTable
<FileName>, <ImportName>, <TableName>,
<ComplexIsRealImag>, <IsMatrixData>,
<ColNames>, <ColIndependentFlags>
```

Return Value: None

Parameters:

`<FileName>`

Type: <string>

Location of the source data.

`<ImportName>`

Type: <string>

Identifying name to use for the import, analogous to solution setup name.

`<TableName>`

Type: <string>

Identifying name for the table, analogous to solution name.

`<ComplexIsRealImag>`

Type: <bool>

Whether to use real/imag to interpret data for any complex column.

If false, then use mag/phase(degrees).

`<IsMatrixData>`

Type: <bool>

Controls whether the table data can be used in matrix data reports or in field data reports.

`<ColNames>`

`Array("ColName1", ...)`

Non-empty array used only if you want to override the column names obtained from

the table data file, in which case all column names are required.

`<ColIndependentFlags>`

`Array(<bool>, ...)`

Indicates which columns are independent. If this is the empty array, the default is that only the first column is independent. If this is the non- empty array, a flag must be present for every column.

Example:

```
oModule.ImportTable "c:\mydir\mytable.tab", _
   "ImportData", "Measurements", TRUE, TRUE, _
   Array(), Array(TRUE, TRUE, FALSE, FALSE, FALSE)
```

IsFieldAvailableAt

Use: To determine if a field solution exists for a particular frequency in a simulation.

Syntax: `IsFieldAvailableAt(<SolutionName>, <DesignVariation>, <Freq>)`

Return Value: Returns 1 or 0 (1= true, 0 = false)

Type: Boolean

Parameters: `<SolutionName>`

Type: `<string>`

`<DesignVariation>`

Type: `<string>`

Example: `"y_start = 3mm"`

`<Freq>`

Type: <double>

Example:

```
   set oModule = oDesign.GetModule("Solutions")
fieldsExist = oModule.IsFieldAvailableAt _
("Setup1:Sweep1", " ", "9000000000.0")
```

ListMatchingVariations

Use: Gets a list of solved variations that include the specified variable values.

Command: None

Syntax: `ListMatchingVariations`

(<FullSolutionName>, <ArrayOfMatchingVariableNames>, <ArrayOfMatchingVariableValueStringsIncludingUits>)

Return Value: An array of strings corresponding to solved variations. The match variables may be a partial set of design variables and the match values are one per variable in the same order as the variables.

Parameters: <FullSolutionName>

Type: String

<ArrayOfMatchingVariableNames>

Type: String

<ArrayOfMatchingVariableValueStringsIncludingUnits>

Type: String

Example:

```
list = oModule.ListMatchingVariations("Setup1 : LastAdaptive",_
Array("x_size", "y_size"), Array("2mm", "1mm"))
```

ListValuesOfVariable

Use: Gets the values of a specified variable corresponding to the solved variations.

Command: None

Syntax: ListValuesOfVariable(<FullSolutionName>, <VariableName>)

Return Value: An array of double precision values in SI units interpreted as the specified variable corresponding to the solved variations.

Parameters: <FullSolutionVariableName>

Type: String

<VariableName>

Type: String

Example:

```
list = oModule.ListValuesOfVariable("Setup1 : _LastAdap-
tive", "x_size")
```

ListVariations

Use: Get a list of solved variations.

Command: None

Syntax: ListVariations(<FullSolutionName>)

Return Value: An array of strings corresponding to solved variations.

Parameters: <FullSolutionName>

Type: String

Example:

```
list = oModule.ListVariations("Setup1 : LastAdaptive")
```

19 Reduce Matrix Module Script Commands

Reduce matrix commands should be executed by the `"ReduceMatrix"` module.

```
Set oDesign = oProject.SetActiveDesign ("Design1")
Set oModule = oDesign.GetModule("ReduceMatrix")
```

The following reduce matrix commands are available for Q3D Extractor:

- InsertRM
- EditRM
- RMAddOp
- DupRMAddOp
- RenameRM
- RenameRMO
- DeleteRM
- DeleteRMO
- DupRM

InsertRM

Use: Adds a new reduce matrix.

Command: **Q3D Extractor>Reduce Matrix>Move Sink**

Q3D Extractor> ReduceMatrix>Add Sink

Q3D Extractor> ReduceMatrix>Join In Series

Q3D Extractor> ReduceMatrix>Join In Parallel

Q3D Extractor> ReduceMatrix>Float Net

Q3D Extractor> ReduceMatrix>Ground Net

Q3D Extractor> ReduceMatrix>Float Terminal

Q3D Extractor> ReduceMatrix>Float At Infinity

Q3D Extractor> ReduceMatrix>Return Path

Q3D Extractor> ReduceMatrix>Change Frequency

Syntax:

```
InsertRM <MatName>, <OpData>
```

Return Value: None

Parameters:

```
<MatName>
   Type: <string>
   Name of the reduce matrix
```

<OpData>

```
   Type: <string>
   One of
      "MoveSink(<Operand >)"
      "AddSink(<Operand >)"
      "JoinSeries(<Operand >, < Operand >, …)"
      "JoinParallel(<NewNet>, <NewSrc>, <NewSnk>, < Oper-
      and >, < Operand >, …)"
      "FloatNet(<Operand >, < Operand >, …)"
      "GroundNet(<Operand >, < Operand >, …)"
      "FloatTerminal(<Operand >, < Operand >, …)"
      "FloatInfinity()"
      "ChangeFreq(<Freq>)"
```

Parameters:

```
   <Operand>
      Type: <string>
      Name of the source/sink/net selected for this opera-
      tion.
```

```
<NewNet>
  Type: <string>
  Name of the new net resulting from this operation.
<NewSrc>
  Type: <string>
  Name of the new source resulting from this operation.
<NewSnk>
  Type: <string>
  Name of the new sink resulting from this operation.
```

Example:

```
oModule.InsertRM "RM1", "MoveSink('src3')"
oModule.InsertRM "RM2", "AddSink('src4')"
oModule.InsertRM "RM3", "JoinSeries('box2', 'src5',
'snk3')"
oModule.InsertRM "RM4", "JoinParallel('box1', 'src1',
'snk1', 'box1', 'src1', 'src2', 'src3', 'src4')"
oModule.InsertRM "RM5", "FloatNet('box3', 'box4')"
oModule.InsertRM "RM6", "GroundNet('box2', 'box3')"
oModule.InsertRM "RM7", "FloatTerminal('src4', 'src7')"
oModule.InsertRM "RM8", "FloatInfinity()"
oModule.InsertRM "RM9", "ChangeFreq('100MHz')"
```

EditRM

Use: Edits an existing operation in a reduce matrix.

Command: Double-click an operation in the project tree to modify its reduce matrix.

Syntax: `EditRM <MatName>, <OpName>, <OpData>`

Return Value: None

```
<MatName>
  Type: <string>
  Name of the reduce matrix being edited.
<OpName>
  Type: <string>
  Name of the reduce operation being edited
<OpData>
  See InsertRM for details.
```

Example:

```
oModule.EditRM "RM1", "MoveSink1", "MoveSink('src2')"
```

RMAddOp

Use: Adds an operation to an existing reduce matrix.

Command: Right-click a reduce matrix in the project tree, and then choose the type of operation to add.

Syntax:
```
RMAddOp <MatName>, <OpData>
```

Return Value: None

Parameters:
```
<MatName>
   Type: <string>
   Name of the reduce matrix being edited.
<OpData>
   See InsertRM for details.
```

Example:
```
oModule.RMAddOp "RM7", "MoveSink('src12')"
```

DupRMAddOp

Use: Duplicates an existing reduce matrix and then add an operation to the duplicated matrix.

Command: Right-click a reduce matrix in the project tree, and then choose the type of operation to add.

Syntax:
```
RMAddOp <MatName>, <OpData>
```

Return Value: None

Parameters:
```
<MatName>
   Type: <string>
   Name of the reduce matrix being edited.
<OpData>
   See InsertRM for details.
```

Example:
```
oModule.RMAddOp "RM7", "MoveSink('src12')"
```

RenameRM

Use: Renames an existing reduce matrix.

Command: Right-click a reduce matrix in the project tree, and then choose **Rename**.

Syntax:
```
RenameRM <OldName>, <NewName>
```

Return Value: None

Parameters:
```
<OldName>
   Type: <string>
```

```
    Name of the reduce matrix being renamed.
<NewName>
    Type: <string>
    New name for the reduce matrix.
```

Example: `oModule.RenameRM "RM1", "RM2"`

RenameRMO

Use: Renames an existing reduce operation.

Command: Right-click a reduce operation in the project tree, and then choose **Rename**.

Syntax: `RenameRMO <MatName> <OldName>, <NewName>`

Return Value: None

Parameters:

```
<Matname>
    Type: <string>
    Name of the reduce matrix being edited.
<OldName>
    Type: <string>
    Name of the reduce operation being renamed.
<NewName>
    Type: <string>
    New name for the reduce operation.
```

Example: `oModule.RenameRMO "RM1", "MoveSink1" "MoveSink"`

DeleteRM

Use: Deletes the specified reduce matrix.

Command: Right-click a reduce matrix in the project tree, and then choose **Delete.**

Syntax: `DeleteRM <MatName>`

Return Value: None

Parameters:

```
<MatName>
    Type: <string>
    Name of the reduce matrix to be deleted.
```

Example: `oModule.DeleteRM "RM1"`

DeleteRMO

Use: Deletes a reduce operation in an existing reduce matrix.

Command: Right-click a reduce matrix operation in the project tree, and then choose **Delete.**

Syntax:
```
DeleteRMO <MatName> <OpName>
```

Return Value: None

Parameters:
```
<MatName>
   Type: <string>
   Name of the reduce matrix being edited.
<OpName>
   Type: <string>
   Name of the reduce operation to be deleted.
```

Example:
```
oModule.DeleteRMO "RM1" "MoveSink2"
```

DupRM

Use: Duplicates a reduce matrix.

Command: Right-click a reduce matrix in the project tree, and then choose **Duplicate.**

Syntax:
```
DupRM <MatName>
```

Return Value: None

Parameters:
```
<MatName>
   Type: <string>
   Name of the reduce matrix to be duplicated.
```

Example:
```
oModule.DupRM "RM1"
```

2D Extractor Reduce Matrix Commands

The following 2D Extractor reduce matrix commands are available:

- InsertRM (2D Extractor)
- EditRM (2D Extractor)
- RMAddOp (2D Extractor)
- RDupMAddOp (2D Extractor)
- RenameRM (2D Extractor)
- RenameRMO (2D Extractor)
- DeleteRM (2D Extractor)
- DeleteRMO (2D Extractor)
- DupRM (2D Extractor)
- ReorderMatrix
- AlphaNumericMatrix

InsertRM (2D Extractor)

Use: Adds a new reduce matrix.

Command: **2D Extractor>Reduce Matrix**

Syntax:
```
InsertRM <ReduceMatrixName>, <ReduceOpParameters>
<ReduceOpType>(<Parameters>)
```

Return Value: None

Parameters:

```
<ReduceMatrixName>
   Type: <string>
   Name of the reduce matrix
<ReduceOpParameters>
   <ReduceOpType>
      Type:<String>
      Various types of reduce operations supported are
      listed below.
   <Parameters>
      Type:<String>
      The parameters of reduce operation varies with
      reduce operation types.

      One of the 2D Extractor Reduce Operations.
```

- Add Ground

- Set Reference Ground
- Float
- Parallel
- Diff Pair.

Example:

```
oModule.InsertRM "RM1",   "AddGround(SelectionArray[1: "
& Chr(39) & "Rectangle2" & Chr(39) & "], OverrideInfo())"
```

EditRM (2D Extractor)

Use: Edits an existing operation in a reduce matrix.

Command: Double-click an operation in the project tree to modify its reduce matrix.

Syntax:

```
EditRM <ReduceMatrixName>, <ReduceOperationName>,
<ReduceOpParameters>
```

Return Value: None

```
<ReduceMatrixName>
   Type: <string>
   Name of the reduce matrix being edited.
<ReduceOperationName>
   Type: <string>
   Name of the reduce operation being edited
<ReduceOpParameters>
   See InsertRM for details.
```

Example:

```
oModule.EditRM "RM1", "AddGround1",  _
"AddGround(SelectionArray[2: " & Chr(39) & "Rectangle2" &
Chr(39) & ", " & Chr(39) & "" & _ "Rectangle3" & Chr(39) &
"], OverrideInfo())"
```

RMAddOp (2D Extractor)

Use: Adds an operation to an existing reduce matrix.

Command: Right-click a reduce matrix in the project tree, and then choose the type of operation to add.

Syntax:

```
RMAddOp <ReduceMatrixName>, <ReduceOpParameters>
```

Return Value: None

Parameters:

```
<ReduceMatrixName>
   Type: <string>
   Name of the reduce matrix being edited.
<ReduceOpParameters>
```

```
   See InsertRM for details.
```

Example:

```
oModule.RMAddOp "RM1",  _ "AddGround(SelectionArray[1: "
& Chr(39) & "Rectangle4" & Chr(39) & "], Overrid" & _
"eInfo())"
```

DupRMAddOp (2D Extractor)

Use: Duplicates an existing reduce matrix and then add an operation to the duplicated matrix.

Command: Right-click a reduce matrix in the project tree, and then choose the type of operation to add.

Syntax:

```
DupRMAddOp <ReduceMatrixName>, <ReduceOpParameters>
```

Return Value: None

Parameters:

```
<ReduceMatrixName>
   Type: <string>
   Name of the reduce matrix being edited.
<ReduceOpParameters>
   See InsertRM for details.
```

Example:

```
oModule.DupRMAddOp "RM1",  _ "AddGround(SelectionArray[1:
" & Chr(39) & "Rectangle4" & Chr(39) & "], Overrid" & _
"eInfo())"
```

RenameRM (2D Extractor)

Renames an existing reduce matrix.

Command: Right-click a reduce matrix in the project tree, and then choose **Rename**.

Syntax:

```
RenameRM <ReduceMatrixName>, <ReduceMatrixNewName>
```

Return Value: None

Parameters:

```
<ReduceMatrixName>
   Type: <string>
   Name of the reduce matrix being renamed.
<ReduceMatrixNewName>
   Type: <string>
   New name for the reduce matrix.
```

Example:

```
oModule.RenameRM "RM1", "GroundMatrix"
```

RenameRMO (2D Extractor)

Use: Renames an existing reduce operation.

Command: Right-click a reduce operation in the project tree, and then choose **Rename**.

Syntax:
```
RenameRMO <ReduceMatrixName> <ReduceOperationName>,
<ReduceOpNewName>
```

Return Value: None

Parameters:
```
<ReduceMatrixname>
   Type: <string>
   Name of the reduce matrix.
<ReduceOperationName>
   Type: <string>
   Name of the reduce operation.
<ReduceOpNewName>
   Type: <string>
   New name for the reduce operation.
```

Example:
```
oModule.RenameRMO "RM1", "AddGround2" "Ground"
```

DeleteRM (2D Extractor)

Use: Deletes the specified reduce matrix.

Command: Right-click a reduce matrix in the project tree, and then choose **Delete.**

Syntax:
```
DeleteRM <ReduceMatrixName>
```

Return Value: None

Parameters:
```
<ReduceMatrixName>
   Type: <string>
   Name of the reduce matrix to be deleted.
```

Example:
```
oModule.DeleteRM "RM1_1"
```

DeleteRMO (2D Extractor)

Use: Deletes a reduce operation in an existing reduce matrix.

Command: Right-click a reduce matrix operation in the project tree, and then choose **Delete.**

Syntax:
```
DeleteRMO <ReduceMatrixName> <ReduceOperationName>
```

Return Value: None

Parameters:
```
<ReduceMatrixName>
```

```
    Type: <string>
    Name of the reduce matrix being edited.
<ReduceOperationName>
    Type: <string>
    Name of the reduce operation to be deleted.
```

Example: `oModule.DeleteRMO "RM1" "AddGround2"`

DupRM (2D Extractor)

Use: Duplicates a reduce matrix.

Command: Right-click a reduce matrix in the project tree, and then choose **Duplicate.**

Syntax: `DupRM <ReduceMatrixName>`

Return Value: None

Parameters:

```
<ReduceMatrixName>
    Type: <string>
    Name of the reduce matrix to be duplicated.
```

Example: `oModule.DupRM "RM1"`

ReorderMatrix

Use: Reorders the sources in the matrix.

Command: Right-click a reduce matrix in the project tree, and then click **Reorder Matrix**.

Syntax: `ReorderMatrix <ReduceMatrixName>,<SourceNamesOrder>`

Return Value: None.

Parameters:

```
<ReduceMatrixName>
    Type:<String>
    Name of the reduce matrix.
<SourceNamesOrder>
    Array(<SourceName>,<SourceName>…)
    <SourceName>
        Type: <String>
        Name of the source.
```

Example:

```
oModule.ReorderMatrix "RM1", Array("Rectangle1",
"Rectangle3", "Rectangle4",  _ "Rectangle5")
```

AlphaNumericMatrix

Use: Reorders the sources in the matrix in alphanumeric order.

Command: Right-click a reduce matrix in the project tree, and then click **Reorder Matrix**. Then select the option for alpha numeric ordering.

Syntax:
```
AlphaNumericMatrix <ReduceMatrixName>
```

Return Value: None.

Parameters:
```
<ReduceMatrixName>
   Type:<String>
   Name of the reduce matrix.
```

Example:
```
oModule.AlphaNumericMatrix "RM1"
```

2D Extractor Reduce Operations

The following reduce operations can be performed in 2DExtractor:

- Add Ground
- Set Reference Ground
- Float
- Parallel
- Diff Pair.

AddGround

Syntax:
```
AddGround
<ReduceOpType>(<SelectionParameters>,<OverrideParameters>
)
```

Parameters:
```
<ReduceOpType>
   Type: <String>
   Value: "AddGround"
<SelectionParameters>
   SelectionArray[<NumberOfSelectedConductors>: <Conduc-
   torName>,<ConductorName>,..]
   <NumberOfSelectedConductors>
      Type: <Integer>
      Number of selected conductors, followed by conductor
      names.
   <ConductorName>
      Type:<String>
```

```
      Name of selected conductor
<OverrideParameters>
   OverrideInfo(<Parameters>)
   <Parameters>
      Type:<String>
      Value: empty string for this operation.
```

Example:

```
oModule.InsertRM "RM2",  _ "AddGround(SelectionArray[2: "
& Chr(39) & "Rectangle2" & Chr(39) & ", " & Chr(39) & "" &
_ "Rectangle3" & Chr(39) & "], OverrideInfo())"
```

SetReferenceGround

Syntax:

```
SetReferenceGround <ReduceOpType>(<SelectionParameters>,
<OverrideParameters>)
```

Parameters:

```
<ReduceOpType>
   Type:<String>
   Value:"SetReferenceGround"
<SelectionParameters>
   SelectionArray[<NumberOfSelectedConductors>: <Conduc-
   torName>,<ConductorName>,..]
   < NumberOfSelectedConductors>
      Type: <Integer>
      Number of selected conductors, followed by conductor
      names.
   <ConductorName>
      Type:<String>
      Name of selected conductor
   <OverrideParameters>
      OverrideInfo(<Parameters>)
      <Parameters>
         Type:<String>
         Value: empty string for this operation.
```

Example:

```
oModule.InsertRM "RM3",  _
"SetReferenceGround(SelectionArray[1: " & Chr(39) &
"Rectangle4" & Chr(39) & "]" & _ ", OverrideInfo())"
```

Float

Syntax:

```
Float <ReduceOpType>(<SelectionParameters>,
<OverrideParameters>)
```

Parameters:

```
<ReduceOpType>
   Type: <String>
   Value: "Float"
<SelectionParameters>
   SelectionArray[<NumberOfSelectedConductors>: <Conduc-
   torName>,<ConductorName>,..]
   < NumberOfSelectedConductors>
      Type: <Integer>
      Number of selected conductors, followed by conductor
      names.
   <ConductorName>
      Type:<String>
      Name of selected conductor
   <OverrideParameters>
      OverrideInfo(<Parameters>)
      <Parameters>
         Type:<String>
         Value: empty string for this operation.
```

Example:

```
oModule.InsertRM "RM4",  _ "Float(SelectionArray[2: " &
Chr(39) & "Rectangle1" & Chr(39) & ", " & Chr(39) & "" & _
"Rectangle3" & Chr(39) & "], OverrideInfo())"
```

Parallel

Syntax:

```
Parallel <ReduceOpType>(<SelectionParameters>,
<OverrideParameters>)
```

Parameters:

```
<ReduceOpType>
   Type: <String>
   Value: "Parallel"
<SelectionParameters>
   SelectionArray[<NumberOfSelectedConductors>: <Conduc-
   torName>,<ConductorName>,..]
   < NumberOfSelectedConductors>
      Type: <Integer>
```

```
    Number of selected conductors, followed by conductor
    names.
  <ConductorName>
    Type:<String>
    Name of selected conductor
  <OverrideParameters>
    OverrideInfo(<OverrideID>,<OverrideName>)
    <OverrideID>
      Type: <integer>
      ID of Conductor with overridden name.
    <OverrideName>
      Type: <String>
      New name of the conductor.
```

Example:

```
oModule.InsertRM "RM5",  _ "Parallel(SelectionArray[2: "
& Chr(39) & "Rectangle1" & Chr(39) & ", " & Chr(39) & "" &
_ "Rectangle2" & Chr(39) & "], OverrideInfo(0, " &
Chr(39) & "Parallel1" & Chr(39) & "" & _ "))"
```

Diff Pair

Syntax:

```
DiffPair <ReduceOpType>(<SelectionParameters>,
<OverrideParameters>)
```

Parameters:

```
<ReduceOpType>
  Type: <String>
  Value: "DiffPair"
<SelectionParameters>
  SelectionArray[<NumberOfSelectedConductors>: <Conduc-
  torName>,<ConductorName>,..]
  < NumberOfSelectedConductors>
    Type: <Integer>
    Number of selected conductors, followed by conductor
    names.
  <ConductorName>
    Type:<String>
    Name of selected conductor
  <OverrideParameters>
    OverrideInfo(<OverrideID>,<OverrideName>)
```

```
<OverrideID>
  Type: <integer>
  ID of Conductor with overridden name.
<OverrideName>
  Type: <String>
  New name of the conductor.
```

Example:

```
oModule.InsertRM "RM6",  _ "DiffPair(SelectionArray[2: "
& Chr(39) & "Rectangle2" & Chr(39) & ", " & Chr(39) & "" &
_ "Rectangle4" & Chr(39) & "], OverrideInfo(1, " &
Chr(39) & "Pair1" & Chr(39) & "" & _ "))"
```

20 Field Overlays Module Script Commands

Field overlay commands should be executed by the Field Overlays module, which is called `"FieldsReporter"` in HFSS scripts.

```
Set oModule = oDesign.GetModule("FieldsReporter")
oModule.CommandName <args>
```

CreateFieldPlot

Use: Creates a field/mesh plot.

Command: **HFSS>Fields>Plot Fields>Mag_E**

Syntax:
```
CreateFieldPlot  <PlotParameterArray>
```

Return Value: None

Parameters:
```
<PlotParameterArray>
   Array("NAME:<PlotName>",
      "SolutionName:=", <string>,
      "QuantityName:=", <string>,
      "PlotFolder:=", <string>,
      "UserSpecifyName:=",  <int>,
      "UserSpecifyFolder:=", <int>,
      "IntrinsicVar:=", <string>,
      "PlotGeomInfo:=", <PlotGeomArray>,
      "FilterBoxes:=", <FilterBoxArray>,
      <PlotOnPointsSettings>,
      <PlotOnLineSettings>,
      <PlotOnSurfaceSettings>,
      <PlotOnVolumeSettings>)
```

`SolutionName`
Name of the solution setup and solution formatted as:
`"<SolveSetupName> : <WhichSolution>"`,
where `<WhichSolution>` can be `"Adaptive_<n>"`, `"LastAdaptive"`, or `"PortOnly"`.
For example: `"Setup1 : Adaptive_2"`
HFSS requires a space on either side of the ':' character. If it is missing, the plot will not be created.

`QuantityName`
Type of plot to create. Possible values are:
Mesh plots: `"Mesh"`
Field plots: `"Mag_E"`, `"Mag_H"`, `"Mag_Jvol"`, `"Mag_Jsurf"`, `"ComplexMag_E"`, `"ComplexMag_H"`, `"ComplexMag_Jvol"`,

"ComplexMag_Jsurf", "Vector_E", "Vector_H",
"Vector_Jvol", "Vector_Jsurf", "Vector_RealPoynting",
"Local_SAR", "Average_SAR"

PlotFolder
Name of the folder to which the plot should be added. Possible values are: "E Field", "H Field", "Jvol", "Jsurf", "SAR Field", and "MeshPlots".

UserSpecifyName
0 if default name for plot is used, 1 otherwise.
Not needed. <PlotName> will be respected regardless of whether this flag is set.

UserSpecifyFolder
0 if default folder for plot is used, 1 otherwise.
Not needed. The specified PlotFolder will be respected regardless of whether this flag is set.

IntrinsicVar
Formatted string that specifies the frequency and phase at which to make the plot.
For example: "Freq='1GHz' Phase='30deg'"

<PlotGeomArray>
Array(<NumGeomTypes>, <GeomTypeData>,
<GeomTypeData>, ...)
For example: Array(4, "Volume", "ObjList", 1, "Box1",
"Surface", "FacesList", 1, "12", "Line", 1,
"Polyline1", "Point", 2, "Point1", "Point2")

<NumGeomTypes>
Type: <int>
Number of different geometry types (volume, surface, line, point) plotted on at the same time.

<GeomTypeData>
<GeomType>, <ListType>, <NumIDs>, <ID>, <ID>, ...)

<GeomType>
Type: <string>
Possible values are "Volume", "Surface", "Line", "Point".

<ListType>
Type: <string>
Possible values are "ObjList", or "FacesList".
These are used for the GeomType of "Line" or "Point".

<NumIDs>
Type: <int>
Number of IDs or object names that will follow.

<ID>
Type: <int> or <string>
ID of a face or name of an object, line, or point on which to plot.

<FilterBoxArray>
Array of names of objects to use to restrict the plot range.
Array(<NumFilters>, <ObjName>, <ObjName>, ...)
Example: Array(1, "Box1")
Example: Array(0) *no filtering*

<PlotOnPointSettings>
Array("NAME:PlotOnPointSettings",
"PlotMarker:=", <bool>,
"PlotArrow:=", <bool>)

<PlotOnLineSettings>
Array("NAME:PlotOnLineSettings",
Array("NAME:LineSettingsID",
"Width:=", <int>,
"Style:=", <string>),

```
"IsoValType:=", <string>,
"ArrowUniform:=", <bool>,
"NumofArrow:=",  <int>)
```

`Style`

Possible values are `"Cylinder", "Solid", "Dashdash", "Dotdot", "Dotdash"`

`IsoValType`

Possible values are `"Tone", "Fringe", "Gourard"`

```
<PlotOnSurfaceSettings>
Array("NAME:PlotOnSurfaceSettings",
"Filled:=", <bool>,
"IsoValType:=", <string>,
"SmoothShade:=", <bool>,
"AddGrid:=", <bool>,
"MapTransparency:=", <bool>,
"Transparency:=", <doubl.e>,
"ArrowUniform:=", <bool>
"ArrowSpacing:=",  <double>
"GridColor:=", Array(<int>, <int>, <int>)
```

`IsoValType`

Possible values are: `"Tone", "Line", "Fringe", "Gourard"`

`GridColor`

Array containing the R, G, B components of the color. Components should be in the range 0 to 255.

```
<PlotOnVolumeSettings>
Array("NAME:PlotOnVolumeSettings",
"PlotIsoSurface:=", <bool>,
"CloudDensity:=", <double>,
"PointSize:=", <int>,
"ArrowUniform:=", <bool>,
```

```
"ArrowSpacing:=", <double>)
```

Example:

```
oModule.CreateFieldPlot Array("NAME:Mag_E1", _
  "SolutionName:=",  "Setup1 : LastAdaptive", _
  "QuantityName:=", "Mag_E", _
  "PlotFolder:=", "E Field1", _
  "UserSpecifyName:=",  0, _
  "UserSpecifyFolder:=", 0, _
  "IntrinsicVar:=", "Freq='1GHz' Phase='0deg'",_
  "PlotGeomInfo:=", Array( 1, "Surface",_
    "FacesList", 1, "7"),_
  "FilterBoxes:=", Array(0),
  Array("NAME:PlotOnSurfaceSettings", _
    "Filled:=", false, _
    "IsoValType:=", "Fringe", _
    "SmoothShade:=", true, _
    "AddGrid:=", false, _
    "MapTransparency:=", true, _
    "Transparency:=", 0, _
    "ArrowUniform:=", true, _
    "ArrowSpacing:=",  0.100000001490116, _
    "GridColor:=", Array(255, 255, 255)))
```

For Q3D Extractor and 2D Extractor, the command details are as follows:

Use: Creates a field/mesh plot.

Command: **Q3D Extractor** or **2D Extractor>Fields**

Syntax: `CreateFieldPlot  <PlotParameterArray>`

Return Value: None

Parameters:

```
<PlotParameterArray>
  Array("NAME:<PlotName>",
    "SolutionName:=", <string>,
    "QuantityName:=", <string>,
    "PlotFolder:=", <string>,
    "UserSpecifyName:=",  <int>,
```

```
    "UserSpecifyFolder:=", <int>,
    "IntrinsicVar:=", <string>,
    "PlotGeomInfo:=", <PlotGeomArray>,
    "FilterBoxes:=", <FilterBoxArray>,
    <PlotOnPointSettings>, <PlotOnLineSettings>,
    <PlotOnSurfaceSettings>, <PlotOnVolumeSettings>)
SolutionName
  Name of the solution setup and solution formatted as:
  "<SolveSetupName> : <WhichSolution>",
  where <WhichSolution> can be "Adaptive_<n>",
  "LastAdaptive", or "PortOnly".
  For example: "Setup1 : Adaptive_2"
  HFSS requires a space on both sides of the ':' charac-
  ter. Otherwise, the plot is not be created.
QuantityName
  Type of plot to create. Possible values are:
  Mesh plots: "Mesh"
  Q3D Field Plots:
```

Field type	Plot quantity names
AC R/L Fields	"SurfaceJac", "Mag_SurfaceJac"
DC R/L PEC Fields	"SurfaceJdc", "Mag_SurfaceJdc"
DC R/L Fields	"VolumeJdc", "Mag_VolumeJdc", "Phidc"
C Fields	"SmoothQ", "ABS_Q"

2D Extractor Field Plots:

Field type	Plot quantity names
CG Fields	"Mag_Phi", "PhiAtPhase", "Mag_E", "VectorE", "Mag_Jcg", "VectorJcg" and "energyCG"
RL Fields	"Flux Lines", "VectorA", "Mag_B", "VectorB", "Mag_H", "VectorH", "Jrl", "VectorJrl", "energyRL", "coenergy", "appenergy" and "emloss"

```
PlotFolder
```

Name of the folder to which the plot should be added. Possible values are: "Q", "ABS_Q", "JDC Vol", "Phi", "JDC Surf", and "JAC".

UserSpecifyName

0 if default name for plot is used, 1 otherwise.

This parameter is not essential. <PlotName> is respected regardless of whether this flag is set.

UserSpecifyFolder

0 if the default folder for plot is used, 1 otherwise.

This parameter is not essential. The specified Plot-Folder is respected regardless of whether this flag is set.

IntrinsicVar

Formatted string that specifies the frequency and phase at which to create the plot.

For example: "Freq='1GHz' Phase='30deg'"

<PlotGeomArray>

Array(<NumGeomTypes>, <GeomTypeData>,

<GeomTypeData>, ...)

For example: Array(4, "Volume", "ObjList", 1, "Box1",

"Surface", "FacesList", 1, "12", "Line", 1,

"Polyline1", "Point", 2, "Point1", "Point2")

<NumGeomTypes>

Type: <int>

Number of different geometry types (volume, surface, line, point) plotted at the same time.

<GeomTypeData>

<GeomType>, <ListType>, <NumIDs>, <ID>, <ID>, ...)

<GeomType>

Type: <string>

Possible values are "Volume", "Surface", "Line", "Point".

<ListType>

Type: <string>

Possible values are "ObjList" or "FacesList".

These are used for GeomType values "Line" or "Point".

<NumIDs>

```
   Type: <int>
   Number of IDs or object names that will follow.
<ID>
   Type: <int> or <string>
```

ID of a face or name of an object, line, or point on which to plot.

```
<FilterBoxArray>
   Array of object names used to restrict the plot range.
   Array(<NumFilters>, <ObjName>, <ObjName>, ...)
   Example: Array(1, "Box1")
   Example: Array(0)   no filtering
<PlotOnPointSettings>
   Array("NAME:PlotOnPointSettings",
   "PlotMarker:=", <bool>,
   "PlotArrow:=", <bool>)
<PlotOnLineSettings>
   Array("NAME:PlotOnLineSettings",
   Array("NAME:LineSettingsID",
   "Width:=", <int>,
   "Style:=", <string>),
   "IsoValType:=", <string>,
   "ArrowUniform:=", <bool>,
   "NumofArrow:=",  <int>)
Style
   Possible values are "Cylinder", "Solid", "Dashdash",
   "Dotdot", "Dotdash".
IsoValType
   Possible values are "Tone", "Fringe", "Gourard".
<PlotOnSurfaceSettings>
   Array("NAME:PlotOnSurfaceSettings",
   "Filled:=", <bool>,
   "IsoValType:=", <string>,
   "SmoothShade:=", <bool>,
   "AddGrid:=", <bool>,
   "MapTransparency:=", <bool>,
   "Transparency:=", <double>,
```

```
   "ArrowUniform:=", <bool>
   "ArrowSpacing:=",  <double>
   "GridColor:=", Array(<int>, <int>, <int>)
IsoValType
   Possible values are: "Tone", "Line", "Fringe", "Gou-
   rard".
GridColor
   Array containing the R, G, B components of the color.
   Components should be in the range 0 to 255.
<PlotOnVolumeSettings>
   Array("NAME:PlotOnVolumeSettings",
   "PlotIsoSurface:=", <bool>,
   "CloudDensity:=", <double>,
   "PointSize:=", <int>,
   "ArrowUniform:=", <bool>,
   "ArrowSpacing:=", <double>)
```

Example:

```
oModule.CreateFieldPlot Array("NAME:Mag_E1", _
   "SolutionName:=",  "Setup1 : LastAdaptive", _
   "QuantityName:=", "Mag_E", _
   "PlotFolder:=", "E Field1", _
   "UserSpecifyName:=",  0, _
   "UserSpecifyFolder:=", 0, _
   "IntrinsicVar:=", "Freq='1GHz' Phase='0deg'",_
   "PlotGeomInfo:=", Array( 1, "Surface",_
      "FacesList", 1, "7"),_
   "FilterBoxes:=", Array(0),
   Array("NAME:PlotOnSurfaceSettings", _
      "Filled:=", false, _ "IsoValType:=", "Fringe", _
      "SmoothShade:=", true, _
      "AddGrid:=", false, _
      "MapTransparency:=", true, _
      "Transparency:=", 0, _
      "ArrowUniform:=", true, _
      "ArrowSpacing:=",  0.100000001490116, _
      "GridColor:=", Array(255, 255, 255)))
```

DeleteFieldPlot

Use: Deletes one or more plots.

Command: **HFSS>Fields>Delete Plot**

Command: **Q3D Extractor** or **2D Extractor>Fields>Delete Plot**

Syntax: `DeleteFieldPlot <NameArray>`

Return Value: None

Parameters: `<NameArray>`

Array of strings – the names of the plots to delete.

Example:

```
oModule.DeleteFieldPlot Array("Mag_E1", "Vector_E1")
```

GetFieldPlotNames

Use: Gets the names of field overlay plots defined in a design.

Syntax: `GetFieldPlotNames()`

Return Value: Array of field plot names.

Parameters: `None`

Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsoftHFSS.HFSSScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.GetActiveProject
Set oDesign = oProject.GetActiveDesign
Set oReportModule = oDesign.GetModule("ReportSetup")
Dim names
names = oReportModule.GetAllReportNames
For index = 0 to UBound(names)
  MsgBox(names(index))
Next
Set oFieldReportModule = oDesign.GetModule("FieldsReporter")
```

```
Set collection = oFieldReportModule.GetFieldPlotNames
For index = 0 to collection.Count-1
  MsgBox(collection.Item(index))
Next
```

For Q3D Extractor, the command details are as follows:

Use: Gets the names of field overlay plots defined in a design.

Syntax: `GetFieldPlotNames()`

Return Value: Array of field plot names.

Parameters: `None`

Example:

```
Set plotnames = oModule.GetFieldPlotNames()
For Each name in plotnames
     Msgbox name
Next
```

ModifyFieldPlot

Use: Modifies a plot definition.

Command: **HFSS>Fields>Modify Plot**

Command: **Q3D Extractor** or **2D Extractor>Fields>Modify Plot**

Syntax: `ModifyFieldPlot <OriginalName> <PlotParameterArray>`

Return Value: None

Example:

```
oModule.ModifyFieldPlot "Vector_E1",_
   Array("NAME:Vector_E2", _
      "SolutionName:=",  "Setup1 : LastAdaptive", _
      "QuantityName:=", "Vector_E", _
      "PlotFolder:=", "E Field1",  _
      "UserSpecifyName:=", 0, _
      "UserSpecifyFolder:=", 0, _
      "IntrinsicVar:=","Freq='1GHz' Phase='30deg'", _
      "PlotGeomInfo:=", Array(1,_
        "Surface","FacesList", 1, "7"), _
      "FilterBoxes:=", Array(0), _
      Array("NAME:PlotOnSurfaceSettings", _
        "Filled:=",   false, _
```

```
"IsoValType:=", "Fringe", _
"SmoothShade:=", true, _
"AddGrid:=", false, _
"MapTransparency:=",  true, _
"Transparency:=", 0, _
"ArrowUniform:=", true, _
"ArrowSpacing:=",  0.100000001490116, _
"GridColor:=", Array(255, 255, 255)))
```

RenameFieldPlot

Use: Renames a plot.

Command: Right-click the plot you want to rename in the project tree, and then click **Rename** on the shortcut menu.

Syntax: `RenameFieldPlot <OldName> <NewName>`

Return Value: None

Parameters:

`<OldName>`

Type: <string>

Original name of the plot.

`<NewName>`

Type: <string>

New name of the plot.

Example:

```
oModule.RenameFieldPlot "Vector_E1", "Vector_E2"
```

RenamePlotFolder

Use: Renames a plot folder.

Command: Right-click a plot folder in the project tree, and then click **Rename** on the shortcut menu.

Syntax: `RenamePlotFolder <OldName> <NewName>`

Return Value: None

Parameters:

`<OldName>`

Type: <string>

Original name of the folder.

<NewName>

Type: <string>

New name of the folder.

Example:

```
oModule.RenamePlotFolder "E Field", "Surface Plots"
```

SetFieldPlotSettings

Use: Sets plot attributes.

Command: **HFSS>Fields>Modify Plot Attributes**, under the **Plots** tab.

Command: **Q3D Extractor** or **2D Extractor>Fields>Modify Plot Attributes**, under the **Plots** tab.

Syntax: `SetFieldPlotSettings <PlotName> <PlotItemAttributes>`

Return Value: None

Parameters:

<PlotName>

Type: <string>

Name of the plot to modify.

<PlotItemAttributes>

```
Array("NAME:FieldsPlotItemSettings",
     <PlotOnPointsSettings>,
     <PlotOnLineSettings>,
     <PlotOnSurfaceSettings>,
     <PlotOnVolumeSettings>)
```

See description of `CreateFieldPlot` command for details.

Example:

```
oModule.SetFieldPlotSettings "Mag_E2", _
  Array("NAME:FieldsPlotItemSettings", _
    Array("NAME:PlotOnLineSettings", _
      Array("NAME:LineSettingsID", _
        "Width:=", 4, _
        "Style:=", "Cylinder"), _
      "IsoValType:=", "Tone", _
      "ArrowUniform:=", true, _
      "NumofArrow:=",  100), _
    Array("NAME:PlotOnSurfaceSettings", _
```

```
        "Filled:=", false, _
        "IsoValType:=", "Tone", _
        "SmoothShade:=",  true, _
        "AddGrid:=", false, _
        "MapTransparency:=", true, _
        "Transparency:=", 0, _
        "ArrowUniform:=",  true, _
        "ArrowSpacing:=", 0.100000001490116, _
        "GridColor:=", Array(255, 255, 255)))
```

SetPlotFolderSettings

Use: Sets the attributes of all plots in the specified folder.

Command: **HFSS>Fields>Modify Plot Attributes**

Command: **Q3D Extractor** or **2D Extractor>Fields>Modify Plot Attributes**

Syntax:
```
SetPlotFolderSettings <PlotFolderName>
<PlotFolderAttributes>
```

Return Value: None

Parameters:

`<PlotFolderName>`

Type: <string>

Name of the folder with the attributes to modify.

```
<PlotFolderAttributes>
   Array("NAME:FieldsPlotSettings",
      "Real time mode:=", <bool>,
      <ColorMapSettings>,
      <Scale3DSettings>,
      <Marker3DSettings>,
      <Arrow3DSettings>)

<ColorMapSettings>
   Array("NAME:ColorMapSettings",
      "ColorMapType:=", <string>,
      "SpectrumType:=", <string>,
      "UniformColor:=", Array(<int>, <int>, <int>),
      "RampColor:=", Array(<int>, <int>, <int>)
```

ColorMapType

Possible values are "Uniform", "Ramp", "Spectrum"

SpectrumType

Possible values are "Rainbow", "Temperature", "Magenta", "Gray"

UniformColor, RampColor

Array containing the R, G, B components of the color. Components should be in the range 0 to 255.

```
<Scale3DSettings>
   Array("NAME:Scale3DSettings",
      "m_nLevels:=",  <int>,
      "m_autoScale:=", <bool>,
      "minvalue:=", <double>,
      "maxvalue:=",  <double>,
      "log:=", <bool>,
      "IntrinsicMin:=", <double>,
      "IntrinsicMax:=",  <double>)

<Marker3DSettings>
   Array("NAME:Marker3DSettings",
      "MarkerType:=", <int>,
      "MarkerMapSize:=",  <bool>,
      "MarkerMapColor:=", <bool>,
      "MarkerSize:=", <double>)
```

MarkerType

9: Sphere

10: Box

11: Tetrahedron

12: Octahedron

default: Sphere

```
<Arrow3DSettings>
  Array("NAME:Arrow3DSettings",
    "ArrowType:=",  <int>,
    "ArrowMapSize:=", <bool>,
    "ArrowMapColor:=", <bool>,
    "ShowArrowTail:=", <bool>,
    "ArrowSize:=",   <double>)
```

ArrowType

0 : Line

1 : Cylinder

2 : Umbrella

default: Line

Example:

```
oModule. SetPlotFolderSettings "E Field1", _
  Array("NAME:FieldsPlotSettings", _
    "Real time mode:=",  true,  _
    Array("NAME:ColorMapSettings", _
      "ColorMapType:=", "Spectrum", _
      "SpectrumType:=", "Rainbow", _
      "UniformColor:=", Array(127, 255, 255), _
      "RampColor:=", Array(255,  127, 127)), _
    Array("NAME:Scale3DSettings", _
      "m_nLevels:=",  27, _
      "m_autoScale:=", true, _
      "minvalue:=", 9.34379863739014, _
      "maxvalue:=",  13683.755859375, _
      "log:=", false, _
      "IntrinsicMin:=", 9.34379863739014, _
      "IntrinsicMax:=",  13683.755859375), _
    Array("NAME:Marker3DSettings", _
      "MarkerType:=", 0, _
      "MarkerMapSize:=",  true, _
      "MarkerMapColor:=", false, _
      "MarkerSize:=", 0.25), _
```

```
Array("NAME:Arrow3DSettings", _
   "ArrowType:=",  1, _
   "ArrowMapSize:=", true, _
   "ArrowMapColor:=", true, _
   "ShowArrowTail:=", true, _
   "ArrowSize:=",   0.25))
```

21 Fields Calculator Script Commands

Fields Calculator commands should be executed by the Field Overlays module, which is called `"FieldsReporter"` in HFSS scripts.

```
Set oModule = oDesign.GetModule("FieldsReporter")
oModule.CommandName <args>
```

The command associated with each of the following scripting commands will be a button pressed in the Fields Calculator.

AddNamedExpression
AddNamedExpr
CalcOp
CalculatorRead
CalcStack
CalculatorWrite
ChangeGeomSettings
ClcEval
ClcMaterial
ClearAllNamedExpr
CopyNamedExprToStack
DeleteNamedExpr
EnterComplex
EnterComplexVector
EnterLine
EnterPoint

EnterQty
EnterScalar
EnterScalarFunc
EnterSurf
EnterVector
EnterVectorFunc
EnterVol
ExportOnGrid
ExportToFile
ExportOnGrid (2DExtractor)
GetTopEntryValue
LoadNamedExpressions
SaveNamedExpressions

AddNamedExpression

Use: Creates a named expression using the expression at the top of the stack.

Command: Click **Add**.

Syntax: `AddNamedExpression <Name>`

Return Value: None

Parameters: `<ExpressionName> and <FieldType>.`

Type: <string>

Name for the new named expression.

<FieldType>

Type: <string>

Example:

```
oModule.AddNamedExpression "Mag_JxE", "Fields"
```

AddNamedExpr

Use: Creates a named expression using the expression at the top of the stack.

Command: Click **Add**.

Syntax: `AddNamedExpr <Name>`

Return Value: None

Parameters: `<ExpressionName>`

Type: <string>

Name for the new named expression.

<FieldType>

Type: <string>

Example:

```
oModule.AddNamedExpr "Mag_JxE", "Fields"
```

CalcOp

Use: Performs a calculator operation.

Command: Operation commands like **Mag**, **+**, etc.

Syntax: `CalcOp <OperationString>`

Return Value: None

Parameters: `<OperationString>`

Type: String

The text on the corresponding calculator button.

Examples: **Mag**, **+**

CalcRead(deprecated)

Use: Reads a file that is written out by the CalcWrite command, and puts the result into a calculator numeric.

Syntax: `CalcRead <FileName> <SolutionName> <VariablesArray>`

Return Value: None

Parameters:

<FileName>

Type: <string>

<*SolutionName*>

Type: <string>

<*VariablesArray*>

Array of variable name and value pairs.

```
oModule.CalcRead _
"c:\example.reg" "Setup1: LastAdaptive",_
Array ("Freq:=", "10GHz", "Phase:=", "0deg")
```

CalculatorRead

Use: Gets a register file and applies it to the calculator stack.

Command: Click **Read**

Syntax: `CalculatorRead <InputFilePath>, <SolutionName>, <FieldType>, <VariablesArray>`

Return Value: None

Parameters:

`<InputFilePath>`

Path to and including name of input register file.

`<SolutionName>`

Type: <string>

Example: `"Setup1 : LastAdaptive"`

`<FieldType>`

Type: <string>

`<VariablesArray>`

Array of variable names, value pairs.

Example:

```
oModule.CalculatorRead "c:\test.reg", _
   "Setup1 : LastAdaptive", "Fields", _
```

```
    Array("Freq:=", "1GHz", "Phase:=", "0deg")
```

For Q3D Extractor, the command example is as follows:

Example:

```
oModule.CalculatorRead "C:\Ansoft\smoothedtemper.fld",
"Setup1 : LastAdaptive", "Fields",
Array("$conductivity:=", "50000000")
```

CalcStack

Use: Performs an operation on the stack.

Command: Stack operation buttons such as **Push** and **Pop**.

Syntax: `CalcStack <OperationString>`

Return Value: None

Parameters:

`<Operation String>`

Type: <string>

The text on the corresponding calculator button.

Example:

```
oModule.CalcStack "push"
```

CalculatorWrite

Use: Writes contents of top register to file.

Command: Click **Write**

Syntax: `CalculatorWrite <OutputFilePath>, <SolutionNameArray>, <VariablesArray>`

Return Value: None

Parameters:

`<OutputFilePath>`

Path to and including name of output register file.

`<SolutionNameArray>`

`Array("Solution:=", <string>)`

`<VariablesArray>`

Array of variable names, value pairs.

Example:

```
oModule.CalculatorWrite "c:\test.reg", _
  Array("Solution:=", "Setup1 : LastAdaptive"), _
  Array("Freq:=", "1GHz", "Phase:=", "0deg")
```

For Q3D Extractor, the command example is as follows:

Example:

```
oModule.CalculatorWrite "C:\Ansoft\smoothedTemp.fld",
Array("Solution:=", "Setup1 : LastAdaptive"),
Array("$conductivity:=", "50000000")
```

ChangeGeomSettings

Use: Changes the line discretization setting.

Command: **Geom Settings**

Syntax: `ChangeGeomSettings <int>`

Return Value: None

Parameters: The line discretization setting.

ClcEval

Use: Evaluates the expression at the top of the stack using the provided solution name and variable values.

Command: Click **Eval**.

Syntax: `ClcEval <SolutionName> <VariablesArray>`

Return Value: None

Parameters:

`<SolutionName>`

Type: <string>

`<VariablesArray>`

Array of variable name, value pairs.

Example:

```
oModule.ClcEval "Setup1: LastAdaptive", _
  Array ("Freq:=", "10GHz",_
    "Phase:=", "0deg")
```

ClcMaterial

Use: Performs a material operation on the top stack element.

Command: Click **Matl**.

Syntax: `ClcMaterial <MaterialString>, <OperationString>`

Return Value: None

Parameters:

`<Material String>`

Type: <string>

The material property to apply.

`<OperationString>`

Type: <string>

Possible values are `"mult"`, or `"div"`.

Example:

```
oModule.ClcMaterial "Permeability (mu)" "mult"
```

ClearAllNamedExpr

Use: Clears all user-defined named expressions from the list.

Command: Click **ClearAll**.

Syntax: `ClearAllNamedExpr`

Return Value: None

Parameters: None

CopyNamedExprToStack

Use: Copies the named expression selected to the calculator stack.

Command: Select a named expression and then click **Copy to stack**.

Syntax: `CopyNamedExprToStack <Name>`

Return Value: None

Parameters: `<Name>`

Type: <string>

The name of the expression to be copied to the top of the stack.

Example:

```
oModule.CopyNamedExprToStack "Mag_JxE"
```

DeleteNamedExpr

Use: Deletes the selected named expression from the list.

Command: Select a named expression and then click **Delete**.

Syntax: `DeleteNamedExpr <Name>`

Return Value: None

Parameters: `<Name>`

Type: <string>

The name of the named expression to be deleted.

Example:

```
oModule.DeleteNamedExpr "Mag_JxE"
```

EnterComplex

Use: Enters a complex number onto the stack.

Command: Click **Number**, and then click **Scalar**. **Complex** option is selected.

Syntax:
```
EnterComplex "<Real> + <Imaginary> j"
```

Return Value: None

Parameters:

`<Real>`

Type: <double>

Real component of the scalar.

`<Imaginary>`

Type: <double>

Imaginary component of the scalar.

Example:
```
oModule.EnterComplex "1 + 2 j"
```

EnterComplexVector

Use: Enters a complex vector onto the stack.

Command: Click **Number**, and then click **Vector**. **Complex** option is selected.

Syntax:
```
EnterComplexVector Array ("<X Re> + <X Im> j",
"<Y Re> + <Y Im> j", "<Z Re> + <Z Im> j")
```

Return Value: None

Parameters:

`<X Re>, <YRe>, <ZRe>`

Type: <double>

Real components of the X, Y, and Z values respectively.

`<X Im>, <YIm>, <ZIm>`

Type: <double>

Imaginary components of the X, Y, and Z values respectively.

Example:
```
oModule.EnterComplexVector Array("1 + 2 j",_
   "1 + 2 j",_
   "1 + 2 j")
```

EnterLine

Use: Enters a line defined in the 3D Modeler editor.

Command: Click **Geometry** and then select **Line**.

Syntax:
```
EnterLine <LineName>
```

Return Value: None

Parameters:
```
<LineName>
```
Type: <string>

Name of a line defined in the 3D Modeler editor.

Example:
```
oModule.EnterLine "Line1"
```

EnterPoint

Use: Enters a point defined in the 3D Modeler editor.

Command: Click **Geometry** and then select **Point**.

Syntax:
```
EnterPoint <PointName>
```

Return Value: None

Parameters:
```
<PointName>
```
Type: <string>

Name of a point defined in the 3D Modeler editor.

Example:
```
oModule.EnterPoint "Point1"
```

EnterQty

Use: Enters a field quantity.

Command: Click **Quantity**, and then select from the list.

Syntax:
```
EnterQty <FieldQuantityString>
```

Return Value: None

Parameters:
```
<Field Quantity String>
```
Type: <string>

The field quantity to be entered onto the stack.

Example:
```
oModule.EnterQty "E"
```

EnterScalar

Use: Enters a scalar onto the stack.

Command: Click **Number** and then click **Scalar**. **Complex** option not selected.

Syntax: `EnterScalar <Scalar>`

Return Value: None

Parameters: `<Scalar>`

Type: <double>

The real number to enter onto the stack.

EnterScalarFunc

Use: Enters a scalar function.

Command: Click **Function** and then select **Scalar**.

Syntax: `EnterScalarFunc <VarName>`

Return Value: None

Parameters: `<VarName>`

Type: <string>

Name of a variable to enter as a scalar function onto the stack.

Example:

```
oModule.EnterScalarFunc "Phase"
```

EnterSurf

Use: Enters a surface defined in the 3D Modeler editor.

Command: Click **Geometry** and then select **Surface**.

Syntax: `EnterSurf <SurfaceName>`

Return Value: None

Parameters: `<SurfaceName>`

Type: <string>

Name of a surface defined in the 3D Modeler editor.

Example:

```
oModule.EnterSurf "Rectangle1"
```

EnterVector

Use: Enters a vector onto the stack.

Command: Click **Number**, and then click **Vector**. **Complex** option not selected.

Syntax: `EnterVector Array (<X>, <Y>, <Z>)`

Return Value: None

Parameters: `<X>`

Type: <double>

X component of the vector.

`<Y>`

Type: <double>

Y component of the vector.

`<Z>`

Type: <double>

Z component of the vector.

Example:

```
oModule.EnterVector Array (1.0, 1.0, 1.0)
```

EnterVectorFunc

Use: Enters a vector function.

Command: Click **Function** and then select **Vector**.

Syntax: `EnterVectorFunc Array(<XVarName>, <YVarName>, <ZVarName>)`

Return Value: None

Parameters: `<XVarName>, <YVarName>, <ZVarName>`

Type: <string>

Name of a variable for the X, Y, and Z coordinates, respectively, to enter as a vector function on the stack.

Example:

```
oModuleEnterVectorFunc Array("X", "Y", "Z")
```

EnterVol

Use: Enters a volume defined in the 3D Modeler editor.

Command: Click **Geometry** and then select **Volume**.

Syntax: `EnterVol <VolumeName>`

Return Value: None

Parameters: `<VolumeName>`

Type: <string>

Name of a volume defined in the 3D Modeler editor.

Example:

```
oModule.EnterVol "Box1"
```

ExportOnGrid

Use: Evaluates the top stack element at a set of points specified by a grid and exports the data to a file.

Command: Click **Export**, and then click **On Grid**.

Syntax:

```
ExportOnGrid <OutputFile> <MinArray> <MaxArray>
<SpacingsArray>, <setup>, <Boolean>, <coordinate system>
[<CS Offset Array>]
```

Return Value: None

Parameters:

`<OutputFile>`

Type: <string>

Name of the output file.

`<MinArray>, <MaxArray>, <SpacingsArray>`

Type: Array<double, double, double>

Min, Max, and Spacing for the coordinate components of the grid system, “Cartesian” (default), “Cylindrical” or “Spherical”.

`<setup>`

Type: <string>

Name of the simulation setup

`<Setup Array>`

Type: Array(“Freq:=”,<ValueUnits>”, “Phase:=”,”<ValueDeg>”),

Freq

`<Boolean>`

Type: True, False

Whether a non-default coordinate system is specified

`<Coordinate sytem>`

Type: <string>,

`[“Cartesian” (default)| “Cylindrical” | “Spherical”]`

`<CS Offset Array>`

Type: Array<double, double, double>

Origin for the offset coordinate system.

Example:

```
oModule.ExportOnGrid "J:\TestSuite\EdgeLossDensity\aaa.fld",
Array( "0mm", "0deg", "-25mm"),
Array("20mm", "90deg", "125mm"),
Array("10mm", "45deg", "50mm"),
"Setup1 : LastAdaptive",
Array("Freq:=", "10000Hz", "Phase:=", "0deg"),
true, "Cylindrical",
Array("0mm", "0mm", "0mm")
```

For Q3D Extractor the command details are as follows:

Use: Evaluates the top stack element at a set of points specified by a grid, and exports the data to a file.

Command: Click **Export**, and then click **On Grid**.

Syntax:
```
ExportOnGrid <OutputFile> <MinArray> <MaxArray>
<SpacingsArray>
```

Return Value: None

Parameters:
```
<OutputFile>
   Type: <string>
   Name of the output file.
<MinArray>, <MaxArray>, <SpacingsArray>
   Type: Array<double, double, double>
   Min, Max, and Spacing for the X, Y, and Z components of
   the grid.
```

Example:
```
oModule.ExportOnGrid
   "C:\Q3D Extractor6OutputFiles\GridExport.reg",_
   Array("1", "1", "1"),_
   Array("4", "4", "4"),_
Array("2", "2", "2")
```

ExportOnGrid (2D Extractor)

Use: Evaluates the top stack element at a set of points specified by a grid, and exports the data to a file.

Command: Click **Export**, and then click **On Grid**.

Syntax:
```
ExportOnGrid <OutputFile> <MinArray> <MaxArray>
<SpacingsArray>
```

Return Value: None

Parameters:

```
<OutputFile>
   Type: <string>
   Name of the output file.
<MinArray>, <MaxArray>, <SpacingsArray>
   Type: Array<double, double, double>
   Min, Max, and Spacing for the X and Y components of the
   grid.
```

Example:

```
oModule.ExportOnGrid
   "C:\2D ExtractorOutputFiles\GridExport.reg",_
   Array("1", "1", "1"),_
Array("4", "4", "4"),_
```

ExportToFile [Fields Calculator]

Use: Evaluates the top stack element at a set of points specified in an external file and exports the data to a file.

Command: Click **Export**, and then click **To File**.

Syntax: `ExportToFile <OutputFile> <PtsFile>`

Return Value: None

Parameters:

`<OutputFile>`

Type: <string>

Name of the output file.

`<PtsFile>`

Type: <string>

Name of the file containing the points at which to evaluate the top stack element. The file should contain tab- or space-separated x,y,z values of data points.

GetTopEntryValue

Use: Gets the value of the top entry of the calculator stack.

Syntax: `GetTopEntryValue(<SolutionName>, <VariablesArray>)`

Return Value: Returns an array of variants, which is either a scalar (one double) or a vector (3 doubles) based on the quantity on top of the stack.

Parameters:

`<SolutionName>`

Type: <string>

Example: "Setup1: LastAdaptive"

<VariablesArray>

Array of variable name, value pairs.

Example:

```
dim topvalue
topvalue = _
oModule.GetTopEntryValue("Setup1:LastAdaptive", _
Array("Freq:=", "1GHz", "Phase:=", "0deg", _
"x_size:=", "2mm"))
If cdbl(topvalue(0)) <- 180.0 then ...
```

LoadNamedExpressions

Use: Loads a named expression definition from a saved file.

Command: In the Fields Calculator, click **Load From...** in the Library area.

Syntax: `LoadNamedExpressions <FileName>, <FieldType>, <NamedExpressions>`

Return Value: None

Parameters:

<FileName>

Type:<String>

Filename and full path to the file to hold the named expression definition.

<FieldType>

Type:<String>

For products with just one filed type, it is set to "Fields".

<NamedExpressions>

Type: Array<string, string,...>

Array of strings containing the names of expression definitions to load from the file.

Example:

```
oModule.LoadNamedExpressions "C:\Ansoft\Personal-
Lib\smth.clc", "Fields", Array("smoothedtemp")
```

SaveNamedExpressions

Use: Saves a named expression definition to a file.

Command: In the Fields Calculator, click **Save To...** in the Library area.

Syntax: `SaveNamedExpressions <FileName>, <NamedExpressions>, <BooleanFlag>`

Return Value: None

Parameters:

<FileName>

Type:<String>

Filename and full path to the file to hold the named expression definition.

<NamedExpressions>

Type: Array<string, string,...>

Array of strings containing the names of expression definitions to load from the file.

<BooleanFlag>

Type:<Boolean>

True: Overwrite the file.

False: Append to the file.

Example:

```
oModule.SaveNamedExpressions "C:\Ansoft\Personal-
Lib\smth.clc", Array("smoothedtemp"), true
```

22 Radiation Module Script Commands

Radiation field commands should be executed by the `"RadField"` module.

```
Set oModule = oDesign.GetModule("RadField")
oModule.CommandName <args>
```

Note: HFSS-IE does not support Radiation module commands.

Conventions Used in this Chapter

`<SetupName>`

Type: <string>

Name of a radiation setup.

`<FaceListName>`

Type: <string>

Name of a qualifying face list. Used for specifying custom radiation surfaces. In order to be valid for use in a radiation surface, the face list should not contain any faces on PML objects and should contain only model faces.

`<CSName>`

Type: string

Name of a coordinate system.

General Commands Recognized by the Radiation Module

DeleteFarFieldSetup

Use: Deletes an existing far-field setup.

Command: **Delete** command in the **List** dialog box. Click **HFSS>List** to access the **List** dialog box.

Syntax:
```
DeleteFarFieldSetup <NameArray>
```

Return Value: None

Parameters:

`<NameArray>`

Type: Array of strings.

An array of radiation setup names.

Example:

```
oModule.DeleteFarFieldSetup Array("Infinite Sphere1")
```

DeleteNearFieldSetup

Use: Deletes an existing near-field setup (line and sphere).

Command: **Delete** command in the **List** dialog box. Click **HFSS>List** to access the **List** dialog box.

Syntax:
```
DeleteNearFieldSetup <NameArray>
```

Return Value: None

Parameters:

`<NameArray>`

Type: Array of strings.

An array of radiation setup names.

Example:

```
oModule.DeleteNearFieldSetup Array("Line1", "Sphere1")
```

GetSetupNames

Use: Gets the names of far field and near field radiation setups in a design.

Syntax:
```
GetSetupNames(<RadiationType>)
```

Return Value: Array of setup names.

Parameters:

`<RadiationType>`

`Type: <string>`

For example: "Sphere"

Example:

```
Set setupnames = oModule.GetSetupNames("Infinite Sphere")
For Each setup in setupnames
     Msgbox setup
Next
```

RenameSetup [Radiation]

Use: Renames an existing radiation setup.

Command: Right-click a radiation setup in the project tree, and then click **Rename** on the shortcut menu.

Syntax: `RenameSetup <OldName>, <NewName>`

Return Value: None

Parameters:

`<OldName>`

Type: <string>

`<NewName>`

Type: <string>

Example:

```
oModule.RenameSetup "Sphere1", "MyNearSphere"
```

Script Commands for Creating and Modifying Radiation Setups

EditFarFieldSphereSetup

Use: Modifies an existing far-field infinite sphere setup.

Command: Double-click a radiation setup in the project tree to modify its settings.

Syntax: `EditFarFieldSphereSetup <InfSphereParams>`

Return Value: None

Example:

```
oModule.EditFarFieldSphereSetup Array("NAME:InfSphere",_
   "UseCustomRadiationSurface:=", true, _
   "CustomRadiationSurface:=", "FaceList1", _
   "ThetaStart:=", "0deg", _
   "ThetaStop:=", "180deg", _
   "ThetaStep:=", "10deg", _
   "PhiStart:=", "15deg", _
   "PhiStop:=", "36deg", _
   "PhiStep:=", "10deg", _
   "UseLocalCS:=", false)
```

EditNearFieldLineSetup

Use: Modifies an existing near-field line setup.

Command: Double-click the radiation setup in the project tree to modify its settings.

Syntax: `EditNearFieldLineSetup <LineParams>`

Return Value: None

Example:

```
oModule.EditNearFieldLineSetup Array("NAME:MyLine", _
   "UseCustomRadiationSurface:=", false, _
   "Line:=", "Polyline2", _
```

```
    "NumPts:=", "100")
```

EditNearFieldSphereSetup

Use: Modifies an existing near-field sphere setup.

Command: Double-click a radiation setup in the project tree to modify its settings.

Syntax: `EditNearFieldSphereSetup <SphereParams>`

Return Value: None

Example:

```
oModule.EditNearFieldSphereSetup Array("NAME:MySphere", _
  "UseCustomRadiationSurface:=", true, _
  "CustomRadiationSurface:=", "FaceList1", _
  "Radius:=", "35mm", _
  "ThetaStart:=", "0deg", "ThetaStop:=", "180deg", _
  "ThetaStep:=", "10deg", "PhiStart:=", "15deg", _
  "PhiStop:=", "36deg", "PhiStep:=", "10deg", _
  "UseLocalCS:=", false)
```

Example: Partial values can be specified, in which case default values will be used to populate the rest of the fields:

```
oModule.EditNearFieldSphereSetup "NAME:MyInfSphere", _
  Array("NAME:MySphere", _
    "UseCustomRadiationSurface:=", true, _
    "CustomRadiationSurface:=", "FaceList1", _
    "Radius:=", "45mm")
```

This will cause default values to be used for the rest of the fields such as `ThetaStop`, `ThetaStart`, `ThetaStep`, `PhiStep`, `PhiStart`, and `PhiStop`;however, the value for the key `CustomRadiationSurface` has to be specified if custom radiation surfaces are used.

InsertFarFieldSphereSetup

Use: Creates/inserts a far-field infinite sphere radiation setup.

Command: **HFSS>Radiation>Insert Far Field Setup>Infinite Sphere**

Syntax: `InsertFarFieldSphereSetup <InfSphereParams>`

Return Value: None

Parameters: `<InfSphereParams>`

```
Array("NAME:<SetupName>",
   "UseCustomRadiationSurface:=", <bool>,
   "CustomRadiationSurface:=", <FaceListName>,
   "ThetaStart:=", <value>,
   "ThetaStop:=", <value>,
   "ThetaStep:=", <value>,
   "PhiStart:=", <value>,
   "PhiStop:=", <value>,
   "PhiStep:=", <value>,
   "UseLocalCS:=", <bool>,
   "CoordSystem:=", <CSName>)
```

`UseCustomRadiationSurface`
If true, provide `CustomRadiationSurface` parameter.
If false, radiation boundary/PML boundaries will be used as radiation surfaces.

`UseLocalCS`
If true, provide `CoordSystem` parameter.
If false, global coordinate system will be used.

Example:

```
oModule.InsertFarFieldSphereSetup
Array("NAME:InfiniteSphere1",_
   "UseCustomRadiationSurface:=", false, _
   "ThetaStart:=", "0deg",_
   "ThetaStop:=", "180deg",_
   "ThetaStep:=", "10deg",_
   "PhiStart:=", "0deg",_
   "PhiStop:=", "36deg",_
   "PhiStep:=", "10deg",_
   "UseLocalCS:=", true,_
   "CoordSystem:=", "RelativeCS1")
```

InsertNearFieldLineSetup

Use: Inserts a near-field line setup. Requires the presence of lines in the model.

Command: **HFSS>Radiation>Insert Near Field Setup>Sphere**

Syntax: `InsertNearFieldLineSetup <LineParams>`

Return Value: None

Parameters:

```
<LineParams>
   Array("NAME:<SetupName>",
      "UseCustomRadiationSurface:=", <bool>,
      "CustomRadiationSurface:=", <FaceListName>,
      "Line:=", <PolyLineName>,
      "NumPts:=", <int>)
```

`<PolyLineName>`

Type: String.

Name of the polyline as determined by name in the history tree.

`UseCustomRadiationSurface`

If true, provide `CustomRadiationSurface` parameter.

If false, radiation boundary/PML boundaries will be used as radiation surfaces.

Example:

```
oModule.InsertNearFieldLineSetup Array("NAME:MyLine", _
   "UseCustomRadiationSurface:=", false, _
   "Line:=", "Polyline1", _
   "NumPts:=", "100")
```

InsertNearFieldSphereSetup

Use: Creates/inserts a near-field sphere radiation setup.

Command: **HFSS>Radiation>Insert Near Field Setup>Sphere**

Syntax: `InsertNearFieldSphereSetup <SphereParams>`

Return Value: None

Parameters:

```
<SphereParams>
   Array("NAME:<SetupName>",
      "UseCustomRadiationSurface:=", <bool>,
      "CustomRadiationSurface:=", <FaceListName>,
      "Radius:=", <value>,
      "ThetaStart:=", <value>,
      "ThetaStop:=", <value>,
      "ThetaStep:=", <value>,
```

```
        "PhiStart:=", <value>,
        "PhiStop:=", <value>,
        "PhiStep:=", <value>,
        "UseLocalCS:=", <bool>,
        "CoordSystem:=", <CSName>)
```

`UseCustomRadiationSurface`
If true, provide `CustomRadiationSurface` parameter.
If false, radiation boundary/PML boundaries will be used as radiation surfaces.

`UseLocalCS`
If true, provide `CoordSystem` parameter.
If false, global coordinate system will be used.

Example:

```
oModule.InsertNearFieldSphereSetup _
  Array("NAME:MySphere", _
    "UseCustomRadiationSurface:=", true, _
    "CustomRadiationSurface:=", "FaceList1", _
    "ThetaStart:=", "0deg", "ThetaStop:=", "180deg", _
    "ThetaStep:=", "10deg", "PhiStart:=", "0deg", _
    "PhiStop:=", "360deg", "PhiStep:=", "10deg", _
    "UseLocalCS:=", true, _
    "CoordSystem:=", "FaceCS1")
```

Script Commands for Modifying Antenna Array Setups

EditAntennaArratSetup

EditAntennaArraySetup

Use: Modifies the antenna array setup. There are 3 choices in the setup. The default is set to **No Array Setup**. There are two (other) kinds of arrays that the user can set: **Regular Array Setup** and **Custom Array Setup**.

Command: **HFSS>Radiation>Antenna Array Setup**

Syntax: `EditAntennaArraySetup <AntennaArrayParams>`

Return Value: None

Parameters: `<AntennaArrayParams>`

```
  Array("NAME:ArraySetupInfo",
```

```
    "UseOption:=", <ArrayOption>,
    <RegularArrayParams>,
    <CustomArrayParams>)
```

<ArrayOption>
Type: <string>
Can be one of three strings: "NoArray", or "RegularArray", "CustomArray".
If "RegularArray" is specified, then <RegularArrayParams> must be specified. If "CustomArray" is specified, <CustomArrayParams> must be specified. You can also supply both the custom and regular array specifications and switch between them by setting this flag to the option you want to use.

```
<RegularArrayParams>
  Array("NAME:RegularArray",
    "NumUCells:=", <value>,
    "NumVCells:=", <value>,
    "CellUDist:=", <value>,
    "CellVDist:=", <value>,
    "UDirnX:=", <value>,
    "UDirnY:=", <value>,
    "UDirnZ:=", <value>,
    "VDirnX:=", <value>,
    "VDirnY:=", <value>,
    "VDirnZ:=", <value>,
    "FirstCellPosX:=", <value>,
    "FirstCellPosY:=", <value>,
    "FirstCellPosZ:=", <value>,
    "UseScanAngle:=", <bool>,
    "ScanAnglePhi:=", <value>,
    "ScanAngleTheta:=", <value>,
    "UDirnPhaseShift:=", <value>,
    "VDirnPhaseShift:=", <value>)
```

UseScanAngle

If true, the values of the ScanAnglePhi and ScanAngleTheta parameters will be used and need to be specified.

If false, the values of the UDirnPhaseShift and VDirnPhaseShift parameters will be used and must be specified.

```
<CustomArrayParams>
   Array("NAME:CustomArray",
      "NumCells:=", <int>,
      <CellsParamsArray

<CellsParamsArray>
   Array("NAME:Cell",
      <CellParams>, <CellParams>, ...)

<CellParams>
   Array("Name:<CellName>",
      "XCoord:=", <double>,
      "YCoord:=", <double>,
      "ZCoord:=", <double>,
      "Amplitude:=", <double>,
      "Phase:=", <double>)
```

The <double> values above should be in SI units.

<CellName>

Type: <string>

Format is: "Cell_n"

Replace n with the index number of the cell, for example: "Cell_1"

Example: Using the "NoArray" option:

```
oModule.EditAntennaArraySetup _
   Array("NAME:ArraySetupInfo", "UseOption:=", "NoArray")
```

Example: Using the "RegularArray" option:

```
oModule.EditAntennaArraySetup _
   Array("NAME:ArraySetupInfo",_
      "UseOption:=", "RegularArray", _
```

```
    Array("NAME:RegularArray", _
    "NumUCells:=", "10", "NumVCells:=", "10",_
    "CellUDist:=", "10mm", "CellVDist:=", "10mm", _
    "UDirnX:=", "1", "UDirnY:=", "0", "UDirnZ:=", _
      "0", _
    "VDirnX:=", "0", "VDirnY:=", "1", "VDirnZ:=",_
      "0", _
    "FirstCellPosX:=", "0mm", _
    "FirstCellPosY:=", "0mm", _
    "FirstCellPosZ:=", "0mm",
    "UseScanAngle:=", true, _
    "ScanAnglePhi:=","45deg", _
    "ScanAngleTheta:=", "45deg"))
```

Example: Using the `"CustomArray"` option:

```
oModule.EditAntennaArraySetup _
  Array("NAME:ArraySetupInfo",
    "UseOption:=", "CustomArray",_
    Array("NAME:CustomArray", _
      "NumCells:=", 3,_
      Array("NAME:Cell", _
        Array("NAME:Cell_1", _
        "XCoord:=", 0, "YCoord:=", 0, "ZCoord:=",0,_
        "Amplitude:=", 1, "Phase:=", 0), _
        Array("NAME:Cell_2", _
        "XCoord:=",0.06729,"YCoord:=","ZCoord:=",0,_
        "Amplitude:=", 1, "Phase:=", 0), _
        Array("NAME:Cell_3", _
        "XCoord:=",0.13458,"YCoord:=",0,"ZCoord:=",0,_
        "Amplitude:=", 1, "Phase:=", 0))))
```

Script Commands for Exporting Antenna Parameters and Max Field Parameters

ExportRadiationParametersToFile

ExportRadiationParametersToFile

Use: Exports radiation parameters to a file. This command can be used to export the max quantities of a near-field setup and, in the case of far fields, the antenna parameters to the specified file.

Command: **HFSS>Radiation>Compute Max/Antenna Params**

Syntax:
```
ExportRadiationParametersToFile <ExportToFileParams>
```

Return Value: None

Parameters:
```
<ExportToFileParams>
   Array("ExportFileName:=", <FilePath>
      "SetupName:=", <SetupName>
      "IntrinsicVariationKey:=", <string>,
      "DesignVariationKey:=", <string>,
      "SolutionName:=", <string>)
```

`<FilePath>`

Type: String.

Specifies the file to export to, for example: "C:\projects\exportantparams.txt".

`IntrinsicVariationKey`

Specifies the frequency at which to extract the parameters. Example: `"Freq='10GHz'"`

`DesignVariationKey`

Specifies the design variations at which to extract the parameters. Example: `"width=5mm"`

Example:
```
oModule.ExportRadiationParametersToFile _
   Array("ExportFileName:=", _
      "C:\projects\exportantparams.txt",_
      "SetupName:=", "Infinite Sphere1", _
      "IntrinsicVariationKey:=", "Freq='10GHz'", _
      "DesignVariationKey:=", "",
```

```
"SolutionName:=", "LastAdaptive")
```

23 User Defined Solutions Commands

User Defined Solution commands should be executed by the "UserDefinedSolutionModule" module.

```
Set oDesign = oProject.SetActiveDesign("TestDesign1")
Set oModule =
oDesign.GetModule("UserDefinedSolutionModule")
```


CreateUserDefinedSolution

Use: Creates a new user defined solution.

Command: **Create User Defined Solution** popup menu is available in the Result folder context menu when applicable.

Syntax:
```
CreateUserDefinedSolution <SolutionName>,
<PluginFileLocation>, <PluginFileRelativePathName>,
<PropertyValuesArray>, <ProbeSelectionArray>,
<DynamicProbesArray>
```

Return Value: The name of the user defined solution that was created. **Note**: if the requested user defined solution name is not available because it is already in use, the user defined solution will be created with a different name which will be returned.

Parameters:

`<SolutionName>`

Type: String

Requested name of new user defined solution.

<PluginFileLocation>

Type: String

Indicates the library where the UDS plugin file is located. This parameter must be one of the following values: "SysLib", "UserLib", "PersonalLib".

<PluginFileRelativePathName>

Type: String

The path of the UDS plugin file relative to the "UserDefinedOutputs" subdirectory of the library specified by <PluginFileLocation>.

<PropertyValuesArray>

Type: Array of strings

Strings specify name-value pairs corresponding to the UDS properties specified in the plugin file.

For example:

```
Array("multiply_factor:=", "2.0", "component_name:=", "resistor1")
```

<ProbeSelectionArray>

Type: Array of <ProbeSelection>'s (see below)

The probe specification array specifies how UDS probes are defined and mapped to traces in the design.

<ProbeSelection>

Type: Array of strings representing how a single probe is defined by a trace. The array contains the below items:

<ReportType>

Type: String

See the CreateReport command for more information.

<ProbeName>

Type: String

Name of the probe being specified. Note: this must match a probe name specified in the UDS plugin file.

<SolutionName>

Type: String

See the CreateReport command for more information.

<SimulatedValueContexArray>

Type: Array of strings

See the CreateReport command for more information.

<PointSetDefinitionArray>

Type: Array of values with optional overriding values and optional variable values.

See the CreateReport command for more information.

<TraceExpressionArray>

Type: Array of strings and values.

This is similar to the TracesExpressionsArray used in the CreateReport command, but there will only be a single component expression named "Probe Component." See the CreateReport command for more information.

<ExtendedTraceInformationArray>

Type: Array of strings and values

See the CreateReport command for more information.

<DynamicProbesArray>

Type: Array of <ProbeSelection>'s, representing the probes that are used by dynamic-probes.

Example:

```
oModule.CreateUserDefinedSolution "User Defined Solution 1", _
"SysLib", "Example",
Array("multiplication_factor:=", "1.2"),
Array(Array("Modal Solution Data",
"Probe 1", "Setup1 : LastAdaptive",
Array(),
Array("Freq:=", Array( "All")),
Array("Probe Component:=", Array("dB(S(1,1))")), Array()),
Array( "Modal Solution Data", "Probe 2",
"Setup1 : LastAdaptive", Array(),
Array("Freq:=", Array( "All")),
Array("Probe Component:=", Array("mag(S(1,1))")), Array())),
Array(Array( "Modal Solution Data",
"Dynamic Probe 1", "Setup1 : LastAdaptive", Array(),
Array("Freq:=", Array( "All")),
Array("Probe Component:=", Array("Freq")), Array()))
```

DeleteUserDefinedSolutions

Use: Deletes one or more user defined solutions.

Command: 'Delete' button from the "User Defined Solutions" dialog.

Syntax: DeleteUserDefinedSolutions <UserDefinedSolutionNames>

Return Value: None

Parameters:

<UserDefinedSolutionNames>

Type: Array of strings

Name of User Defined Solutions to be deleted.

Example: Example:

```
oModule.DeleteUserDefinedSolutions Array("Solution1", "Solution2")
```

EditUserDefinedSolution

Use: Updates an existing user defined solution

Command: Command: 'Edit' button in the User Defined Solutions dialog

Syntax: Syntax: EditUserDefinedSolution <ExistingSolutionName>, <NewSolutionName>, <PluginFileLocation>, <PluginFileRelativePathName>, <PropertyValuesArray>, <ProbeSelectionArray>, <DynamicProbesArray>

Return Value: Return Value: the name of the user defined solution after being updated. **Note**: if the requested user defined solution name is not available because it is already in use, the user defined solution will be created with a different name which will be returned.

Parameters:

<ExistingSolutionName>

Type: String

Name of the existing user defined solution

<NewSolutionName>

Type: String

Requested name for the updated user defined solution.

<PluginFileLocation>

See CreateUserDefinedSolution for more information.

<PluginFileRelativePathName>

See CreateUserDefinedSolution for more information.

<PropertyValuesArray>

See CreateUserDefinedSolution for more information.

<ProbeSelectionArray>

See CreateUserDefinedSolution for more information.

<DynamicProbesArray>

See CreateUserDefinedSolution for more information.

Example:

```
oModule.EditUserDefinedSolution "User Defined Solution 1",  _
```

```
"User Defined Solution 2",
"SysLib", "Example", Array("multiplication_factor:=", "1.2"),
Array(Array("Modal Solution Data", "Probe 1",
"Setup1 : LastAdaptive", Array(),
Array("Freq:=", Array("All")),
Array("Probe Component:=", Array("dB(S(1,1))")), Array()),
Array( "Modal Solution Data", "Probe 2",
"Setup1 : LastAdaptive", Array(),
Array("Freq:=", Array( "All")),
Array("Probe Component:=", Array("mag(S(1,1))")), Array())),
Array(Array("Modal Solution Data",
"Dynamic Probe 1",
"Setup1 : LastAdaptive", Array(),
Array("Freq:=", Array("All")),
Array("Probe Component:=", Array("Freq")), Array()))
```

24 NdExplorer Script Commands

The definition manager controls the use of NdExplorer for HFSS scripts.

```
Set oProject = oDesktop.SetActiveProject("Project1")
Set oDefinitionManager = oProject.GetDefinitionManager()
```

The topics for this section include:

NdExplorer Manager Script Commands

The NdExplorer manager provides access to scripting for NdExplorer in HFSS. The manager object is accessed via the definition manager.

```
Set oDefinitionManager = oProject.GetDefinitionManager()
Set oNdExplorerManager = oDefinitionManager.GetManager("NdExplorer")
```

The NdExplorere manager script commands are listed below.

ExportFullWaveSpice [NdExplorer]

ExportNetworkData [NdExplorer]

ExportNMFData [NdExplorer]

ExportFullWaveSpice [NdExplorer]

Use: Export FullWaveSpice data in a format of your choice.

Command: File > Export MacroModel > Broadband (SYZ, FWS….)

Syntax: ExportFullWaveSpice

"DesignName", // Design name. Can be left blank, if loading solution from a file.

true/false, // true - solution loaded from file, false- loaded from design

"Name", // If loading from design this is the solution name, else this is the

// full path of the file from which the solution is loaded

"variation", // Pick a particular variation. Leave blank if no variation.

Array("NAME:Frequencies"), // Optional; if none defined all frequencies are used

Array("NAME:SpiceData", // Spice export options object

"SpiceType:=", "SSS", // SpiceType can be "PSpice", "HSpice", "Spectre", "SSS",

// "Simplorer", "TouchStone1.0", "TouchStone2.0"

"EnforcePassivity:=", false, // Enforce Passivity true/false

"EnforceCausality:=", false, // Enforce Causality true/false

"UseCommonGround:=", false, // Use common ground true/false

"FittingError:=", 0.5, // Fitting error

"MaxPoles:=", 400, // Maximum Order

"PassivityType:=", "ConvexOptimization", // Passivity Type can be "ConvexOptimization",

// "PassivityByPerturbation", or "IteratedFittingOfPV"

"ColumnFittingType:=", "Column", // Column FittingType can be "Column", "Entry", "Matrix"

"SSFittingType:=", "TWA", // SS Fitting Type can be "TWA", "IterativeRational"

"RelativeErrorToleranc:=", false, // Relative error tolerance true/false

```
"TouchstoneFormat:=",  "MA",              // Touchstone Format "MA", "RI", "DB"
"TouchstoneUnits:=", "Hz",                // Touchstone Units "Hz", "KHz", "MHz",
"MHz"
"TouchStonePrecision:=", 8,               // Touchstone precision
"ExportDirectory:=", "C:/Examples/LNA/",  // Directory to export to
"ExportSpiceFileName:=", "Linckt_HBTest_2.sss",  // Spice export file
"FullwaveSpiceFileName:=", "Linckt_HBTest.sss", // FWS file
"CreateNPortModel:=",   true   // Create a model based on the exported file  true/false
)
```

ExportNetworkData [NdExplorer]

Use: Export the solution in a format of your choice (Citifile, Spreadsheet, Matlab)

Command: File > Export SYZ Data

Syntax:

```
ExportNetworkData
"DesignName", // Design name. Can be left blank, if loading solution from a file.
true/false,        // true - solution loaded from file, false- loaded from design
"Name",          // If loading from design this is the solution name, else this is the
                 // full path of the file from which the solution is loaded
"ExportFile",      // full path of file to export to
"variation",     // Pick a particular variation. Leave blank if no variation
Array("NAME:Frequencies"), // optional, if none defined all frequencies are used
Array("NAME:Options",            //  Export options object
"DataTypes:=", Array("S"),       // DataTypes can be "S", "Y","Z", "G", and "Z0",
                                 // for S , Y, Z matrix, Gamma and Z0 (zero)
"DisplayFormat:=",  "MA",        // DisplayFormat "MA", "RI", "DB"
"FileType:=", "",                 // Export File Type
                      // 2 - Spreadsheet(*.tab)
                      // 3 - Touchstone(*.sNp)
                      // 4 - Citifile(*.cit)
                      //  6 - Neutral format(*.nmf)
                      //  7 - Matlab format(*.m)
"Renormalize:=", false,           //  Renormalize   true/false
"RefImpedance:=",50,              // Reference Impedance
"Precision:=", 8,                 //  Number of digits Precision
"CreateNPortModel:=",   true   // Create a model based on the exported file  true/false
```

)

ExportNMFData [NdExplorer]

Use: Export the solution in NMF format.

Command: File > Export SYZ Data

Syntax:

```
ExportNMFData
"DesignName", // Design name. Can be left blank, if loading solution from a file.
true/false,        // true - solution loaded from file, false- loaded from design
"Name",           // If loading from design this is the solution name, else this is the
                  // full path of the file from which the solution is loaded
"ExportFile",      // full path of file to export to
Array("NAME:Frequencies"), // optional, if none defined all frequencies are used
Array("NAME:NMFOptions",            //  Export NMF options object
"DataTypes:=", Array("S"),         // DataTypes can be "S", "Y","Z", "G", and "Z0",
                                   // for S , Y, Z matrix, Gamma and Z0 (zero)
"DisplayFormat:=",  "MA",         // DisplayFormat "MA", "RI", "DB"
"FileType:=", "",                  // Export File Type
                        // 2 - Spreadsheet(*.tab)
                        // 3 - Touchstone(*.sNp)
                        // 4 - Citifile(*.cit)
                        //  6 - Neutral format(*.nmf)
                        //  7 - Matlab format(*.m)
"Renormalize:=", false,            //  Renormalize   true/false
"RefImpedance:=",50,               // Reference Impedance
"Precision:=", 8,                   //  Number of digits Precision
"Variables:=", ARRAY("FF", "cap", "Rs")   // Array of variables
"Variations:=", ARRAY("", "", "")   // Array of variations to export solutions for
Array("NAME:ConstantVars")        // Array of variables that are constant, can be
empty
Array("NAME: DependentVars")      // Array of variables that are dependent, can be
empty
"MatrixSize:=", 2,                    // Matrix size, optional (used in nmf file header)
"CreateNPortModel:=",   true   // Create a model based on the exported file  true/false
)
```

25 CompInstance Script Commands

CompInstance commands should be executed by the **oDesign2** or **oPropHost2** object.

```
Set oDesign2 =CompInstance.GetParentDesign
Set oPropHost2 = CompInstance.GetPropHost
```

The topics for this section include:

Callback Scripting Using CompInstance Object

Callback scripts are scripts that can be set in the Property Dialog for individual properties by clicking the button in the Callback column and choosing a script that is saved with the project. Callback scripts can contain any legal script commands including general ANSYS script function calls (e.g., GetApplicationName()). In addition, Callback scripts can also call functions on a special object named CompInstance.

You can obtain an interface to a CompInstance in a schematic or layout by calling oEditor.GetCompInstanceFromRefDes(refDes). For more information see Layout Scripting and Schematic Scripting. This interface is also available as a CompInstance object in CompInstance event callbacks, such as placing a component in a layout or schematic.

Definitions

<**propName**> = text string

<**value**> = double

<**valueText**> = text string

<**fileName**> = full path file name

<**choices**> = string containing menu choices separated by commas

<**initialChoice**> = string containing initial choice for menu; must be one of the <choices>

<**scriptName**> = string containing name of script stored in project

<**bool**> is 1 for true or 0 for false

<**editorName**> is either "Layout" or "SchematicEditor"

The topics for this section include:

CompInstance Functions

CompInstance Functions

Following are commands that can be used to manipulate properties from a CompInstance script.

The topics for this section include:

GetComponentName

GetEditor

GetInstanceID

GetInstanceName

GetParentDesign

GetPropHost

GetPropServerName

GetComponentName

Use: Returns the name of the component corresponding to this CompInstance.

Command: None

Syntax: `GetComponentName()`

Return Value: String

Example:

```
name = CompInstance.GetComponentName();
Returns name of component (e.g. MS_TRL) and stores it in "name".
```

GetEditor [Component Instance]

Use: Returns an interface to the editor requested — if the compInstance is contained within that type of editor. Common editor names are "SchematicEditor" and "Layout".

Command: None

Syntax: `GetEditor(<editorName>)`

Return Value: Returns interface to editor requested.

Example:

```
Set oLayout2 = PropHost.GetEditor("Layout");
Returns the interface to the layout containing a selected component.
This interface can be used to call Layout Scripting functions.
```

GetInstanceID [Component Instance]

Use: Returns the instanceID of the CompInstance.

Command: None

Syntax: `GetInstanceID()`

Return Value: String

Example:

```
id = CompInstance.GetInstanceID();
Returns id of compInstance (e.g. 7) and stores it in "id".
Note that this is not the same number as the one used in the RefDes.
```

GetInstanceName [Component Instance]

Use: Returns the instance name of the component corresponding to this CompInstance.

Command: None

Syntax: `GetInstanceName()`

Return Value: String

Example:

```
name = CompInstance.GetInstanceName();
```

Returns instanceName (e.g. A7) of compInstance and stores it in "name".

Note that the Instance Name is not the same as the RefDes.

GetParentDesign

Use: Returns an interface to the compInstance's parent design.

Command: None

Syntax: `GetParentDesign()`

Return Value: Returns interface to design.

Example:

```
Set oDesign2 = CompInstance.GetParentDesign();
```

Returns the interface to the design containing the compInstance.

This interface can be used to call Design functions; for more information see Nexxim Scripting or PlanarEM Scripting.

GetPropHost

Use: Returns an interface to the PropHost of the CompInstance, which gives access to its properties.

Command: None

Syntax: `GetPropHost()`

Return Value: Returns interface to PropHost.

Example:

```
Set oPropHost2 = CompInstance.GetPropHost();
```

Returns the interface to the properties of the compInstance.

This interface can be used to call PropHost functions; for more information see Call-back Scripting Using PropHost Object.

GetPropServerName

Use: Returns the PropServerName of the Component corresponding to this CompInstance.

Command: None

Syntax: `GetPropServerName()`

Return Value: String

Example:

```
name = CompInstance.GetPropServerName();
Returns propserver name of compInstance (e.g., Comp-
Inst@MS_TRL;7) and stores it in "name".
```

26 Layout Scripting

The Layout scripting interface is a set of commands that can be executed by the *Layout Editor* interface of *Design* objects. Accessing the *Layout Editor* interface can be done by the *SetActiveEditor* method of a *Design* object, for example:

```
Dim oEditor
oEditor = DesignObject.SetActiveEditor ("Layout")
```

The call to each scripting method contains three elements:

- Layout editor
- Method name
- Method arguments (if any)

For instance, in Visual Basic, a call might look like this:

```
oEditor.Delete ("rect_1")
```

The belhavior of the scripting methods is similar to the corresponding commands in the Layout editor. This correspondence is particularly evident when recording a script in Layout Editor (**Tools>Record Script**).

The topics for this section include:

Object Identifiers and Script Recording

Many scripting methods apply to one or more *existing* objects. Objects in scripts are uniquely identified by their *names*, and the object names are typically referenced as:

Array("name1", "name2") // `name1 and name2` are the names of the objects.

While working interactively in the Layout Editor, the user may observe (but not modify, except in some special cases) the *names* of the currently selected objects through the "Selected Object Properties" dialogs. These same names are the ones that the scripting methods use in order to identify the object the method applies to.

When particular objects are involved in interactive script recording, they are typically selected before the actual execution the particular interactive command. In this case, the *selected objects*' names would appear within the generated script method argument(s).

Create Primitives

The following create primitives are available.

CreateCircle (Layout Editor)

Use: Creates a circle primitive object and adds it to the current layout. Returns the name of the newly created object.

Syntax: **`CreateCircle`** `<circle_description>`

Parameters: **`<circle_description>:`**

```
Array("NAME:Contents",
"circleGeometry:=", <circle_geometry>) // object
```

```
  description

<circle_geometry> :
  Array("LayerName:=", <layer_name>,  // name of the
  layer
  "lw:=",        <value>, // border line width
  "x:=",         <value>, // center x coordinate
  "y:=",         <value>, // center y coordinate
  "r:=",         <value>) // radius

<layer_id> :
  integer; never used in scripting

<object_name> :
  quoted string, uniquely identifying an object

<value> :
  quoted_string_parseable_as_value (e.g. "0.111mm")
```

Example:

```
oEditor.CreateCircle
Array("NAME:Contents",
"circleGeometry:=",
Array("Layer:=", 6,
"Name:=",  "circle_150",
"LayerName:=", "Top",
"lw:=", "0mm",
"x:=", "34mm",
"y:=", "-15mm",
"r:=",  "8.60232526704263mm"))
```

CreateLine (Layout Editor)

Use: Creates a line primitive object.

Syntax: **CreateLine** <line_description>

Return Value: Returns the name of the newly created object.

Parameters: **<line_description>:**

Array("NAME:Contents",

```
    "lineGeometry:=", <line_geometry>)
<line_geometry> :
   Array("LayerName:=",<layer_name>,
   "lw:=",       <value>, // line width
   "endstyle:=", <end_style>,
   "joinstyle:=",<join_style>,
   <vertex_sequence> )

<end_style> : integer
   FlatEnd = 0
   ExtendedEnd = 1
   RoundEnd = 2

<join_style> : integer
   UnmiteredJoin = 0
   MiteredJoin = 1
   OptimallyMiteredJoin = 2
   RadiallyMiteredJoin = 3
   ChordMiteredJoin = 4

<vertex_sequence>:
   "n:=", integer, // the total count of the vertices
   "x0:=",<value>, "y0:=", <value>,
   "x1:=",<value>, "y1:=", <value>,
   etc. up to vertex count)
```

Example:

```
oEditor.CreateLine
Array("NAME:Contents",
"lineGeometry:=",
Array("Layer:=", 6,
"Name:=",  "line_1",
"LayerName:=", "Top",
"lw:=", "2mil",
"endstyle:=", 0,
"joinstyle:=",  0,
```

```
"n:=", 4,
"x0:=", "-32mm", "y0:=", "2mm",
"x1:=", "-5mm", "y1:=", "12mm",
"x2:=",  "1mm", "y2:=", "-4mm",
"x3:=", "19mm", "y3:=", "3mm"))
```

CreateRectangle (Layout Editor)

Use: Creates a rectangle primitive object and adds it to the current layout.

Syntax: **CreateRectangle** <rectangle_description>

Return Value: Returns the name of the newly created object.

Syntax: **<rectangle_description>**:

```
 Array("NAME:Contents",
"rectGeometry:=", <rectangle_geometry>)
<rectangle_geometry> :
Array("LayerName:=", <layer_name>,  // placement layer
"lw:=", <value>,   // line width
"x:=",  <value>,   // center X coordinate
"y:=",  <value>,   // center Y coordinate
"w:=",  <value>,   // width (X-direction size)
"h:=",  <value>,   // height (Y-direction size)
"ang:=", <value>)) // rotation
```

Example:

```
oEditor.CreateRectangle
Array("NAME:Contents",
"rectGeometry:=",
Array("Layer:=", 6,
"Name:=",  "rect_4",
"LayerName:=", "Top",
"lw:=", "0mm",
"x:=", "29mm",
"y:=", "9.5mm",
"w:=",  "24mm",
"h:=", "15mm",
"ang:=", "0deg"))
```

CreatePolygon (Layout Editor)

Use: Creates a polygon primitive object.

Syntax: **CreatePolygon** <polygon_description>

Return Value: Returns the name of the newly created object.

Parameters:

```
<polygon_description>:
Array("NAME:Contents",
   "lineGeometry:=", <polygon_geometry>)

<polygon_geometry> :
   Array("LayerName:=", <layer_name>,  // layer
   "lw:=",        <value>,       // border line width
   <vertex_sequence> )          // polygon boundary
```

Example:

```
oEditor.CreatePolygon
Array("NAME:Contents",
"polyGeometry:=",
Array("Layer:=", 6,
"Name:=",  "poly_5",
"LayerName:=", "Top",
"lw:=", "0mm",
"n:=", 5,
"x0:=", "-35mm", "y0:=", "17mm",
"x1:=", "-37mm", "y1:=", "5mm",
"x2:=", "-30mm", "y2:=", "8mm",
"x3:=", "-30mm", "y3:=", "8mm",
"x4:=", "-30mm", "y4:=", "8mm"))
```

CreateText (Layout Editor)

Use: Creates a text primitive object and adds it to the current layout.

Syntax: **CreateText** <text_description>

Return Value: Returns the name of the newly created object.

Parameters:

```
<text_description>:
      Array("NAME:Contents",
      "textGeometry:=", <text_geometry>)
<text_geometry> :
```

```
  Array("LayerName:=", <layer_name>,
  "x:=",          <value>,  // origin X
  "y:=",          <value>,  // origin Y
  "ang:=",        <value>,  // rotation
  "isPlot:=", <bool>,   // is it plotted or not
  "font:=",       <font_name>,
  "size:=",       <value>,
  "weight:=",     <text_weight>,
  "just:=",       <text_justification>,
  "mirror:=",     <bool>,
  "text:=",       <quoted_string>)  // text itself
```

<bool> : *true* | *false*

<font_name> : quoted string, a font name

<text_weight> : integer

```
  Thin = 0,
  ExtraLight = 1
  Light = 2
  Normal = 3
  Medium  = 4
  SemiBold = 5
  Bold = 6
  ExtraBold = 7
  Heavy =8
```

<text_justification> : integer

```
  LeftTop = 0
  LeftBase = 1
  LeftBottom = 2
  CenterTop = 3
  CenterBase= 4
  CenterBottom = 5
  RightTop = 6
```

```
    RightBase = 7
    RightBottom = 8
```

Example:

```
oEditor.CreateText
Array("NAME:Contents",
"textGeometry:=",
Array("Layer:=", 6,
"Name:=",  "text_6",
"LayerName:=", "Top",
"x:=", "-26mm",
"y:=", "-16mm",
"ang:=", "0deg",
"isPlot:=",  true,
"font:=", "Roman",
"size:=", "0.508mm",
"weight:=", 3,
"just:=", 4,
"mirror:=",  false,
"text:=", "Sample"))
```

Create Voids in Primitives

The following primitives are available.

CreateLineVoid

CreatePolygonVoid

CreateCircleVoid

CreateRectangleVoid

CreateCircleVoid (Layout Editor)

Use: Creates a circle void and adds it to a particular parent primitive.

Syntax: **CreateCircleVoid** <circle_void_description>

Return Value: Returns the name of the newly created object

Parameters: **<circle_void_description>:**

```
Array("NAME:Contents",
"owner:=", <object_name>, // parent primitive name
"circle voidGeometry:=", <circle_geometry>) // definition
```

Example:

```
oEditor.CreateCircleVoid
Array("NAME:Contents",
"owner:=", "rect_4",
"circle voidGeometry:=",
Array ("Layer:=",  6,
"Name:=", "circle void_10",
"LayerName:=", "Top",
"lw:=", "0mm",
"x:=",  "26mm",
"y:=", "11mm",
"r:=", "1.41421356237309mm"))
```

CreateLineVoid (Layout Editor)

Use: Creates a line void and adds it to a specified as parameter parent primitive.

Syntax: **CreateLineVoid** <line_void_description>

Return Value: Returns the name of the newly created object.

Parameters: **<line_void_description>**:

```
Array("NAME:Contents",
"owner:=", <object_name>, // parent primitive name
"line voidGeometry:=", <line_geometry>) // definition
```

Example:

```
oEditor.CreateLineVoid
Array("NAME:Contents",
"owner:=", "rect_4",
"line voidGeometry:=",
Array("Layer:=",  6,
"Name:=", "line void_14",
"LayerName:=", "Top",
"lw:=", "2mil",
"endstyle:=",  0,
"joinstyle:=", 0,
"n:=", 3,
"x0:=", "27mm", "y0:=", "5mm",
"x1:=", "35mm", "y1:=",  "5mm",
"x2:=", "36mm", "y2:=", "9mm"))
```

CreatePolygonVoid (Layout Editor)

Use: Creates a polygon void and adds it to a specified as parameter parent primitive.

Syntax: **CreatePolygonVoid** <polygon_void_description>

Return Value: Returns the name of the newly created object.

Parameters:

```
<polygon_void_description>:
Array("NAME:Contents",
   "owner:=", <object_name>, // owner name
   "poly voidGeometry:=", <polygon_geometry>) // defini-
   tion
```

Example:

```
oEditor.CreatePolygonVoid
Array("NAME:Contents",
"owner:=", "rect_4",
"poly voidGeometry:=",
Array("Layer:=", 6,
"Name:=", "poly void_18",
"LayerName:=", "Top",
"lw:=", "0mm",
"n:=", 5,
"x0:=", "21mm", "y0:=", "9mm",
"x1:=", "21mm", "y1:=", "5mm",
"x2:=", "24mm", "y2:=", "7mm",
"x3:=", "24mm", "y3:=", "7mm",
"x4:=", "24mm", "y4:=", "7mm"))
```

CreateRectangleVoid (Layout Editor)

Use: Creates a rectangle void and adds it to a specified as parameter parent primitive.

Syntax: **CreateRectangleVoid** <rectangle_void_description>

Return Value: Returns the name of the newly created object.

Syntax:

```
<rectangle_void_description>:
Array("NAME:Contents",
      "owner:=", <object_name>, // owner_name
      "rect voidGeometry:=", <rectangle_geometry>)
```

Example:

```
oEditor.CreateRectangleVoid
Array("NAME:Contents",
"owner:=", "rect_4",
"rect voidGeometry:=",
Array("Layer:=", 6,
"Name:=", "rect void_16",
"LayerName:=", "Top",
"lw:=", "0mm",
"x:=",  "34.5mm",
"y:=", "13mm",
"w:=", "3mm",
"h:=", "2mm",
"ang:=", "0deg"))
```

Other Creation Methods

The following creation methods are available.

AddCircuitRefPort <...>
AddRefPort <...>
AssignCircuitRefPort <...>
AssignRefPort <...>
CreateCircuitPort <...>
CreateComponent <...>
CreateEdgePort <...>
CreateHole <...>
CreateMeasure <...>
CreateNPort <...>
CreatePin <...>
CreateTrace <...>
CreateVia <...>

AddCircuitRefPort (Layout Editor)

Use: Assign an edge as a reference to an existing port; model as a circuit port.

Command: AddCircuitRefPort

Syntax:

```
AddCircuitRefPort
Array(<"port name">, <"port name">, ...),
Array("NAME:Contents",
"edge:=", Array(<edge description>), "edge:=, Array(<edge description>), ...)
```

Return Value: None.

Parameters:

`<edge description> for primitive edges`

```
"et:=", "pe", "prim:=", <"prim">, "edge:=", <edge#>
```

`<"prim">:`
text that is the primitive name

`<edge#>:`
integer that is the edge number on the primitive

`<edge description>:`
for via edges

```
"et:=", "pse", "sel:=", <"via">, "layer:=",  <layer id>,
 "sx:=", <start X location>, "sy:=", <start Y location>,
"ex:=", <end X location>,
"ey:=", <end Y location>, "h:=", <arc height>, "rad:=",
<radians>
```

`<"via">:`
text that is the name of the via to use
`<layer id>:`
an integer that is the id of the layer of the pad of the via to use

`<start X location>, <start Y Location>:`
doubles that are the X, Y location of the start point of the edge arc
`<end X location>, <end Y Location>:`
doubles that are the X, Y location of the end point of the edge arc
`<arc height>:`
double giving the height of the edge arc (0 for a straight edge)
`<radians>:`

double giving the arc size in radians (0 for a straight edge)

Example:

```
oEditor.AddCircuitRefPort Array("Port3"),
Array("NAME:Contents", "edge:=", _
Array("et:=", "pe", "prim:=", "rect_2", "edge:=", 1))
```

AddRefPort (Layout Editor)

Use: Create a reference port from edges and associate with each of the named ports.

Command: (When more than 2 edges are selected.)

Draw > Port > Create

Right-click > Port > Create

Also available through toolbar icon.

Syntax:

```
AddRefPort
Array(<"port name">, <"port name">, ...),
Array("NAME:Contents",
"edge:=", Array(<edge description>), "edge:=, Array(<edge
description>), ...)
```

Return Value: None.

Parameters:

`<edge description>` for primitive edges

```
"et:=", "pe", "prim:=", <"prim">, "edge:=", <edge#>
```

`<"prim">:`

text that is the primitive name

`<edge#>:`

integer that is the edge number on the primitive

`<edge description>`

for via edges

```
"et:=", "pse", "sel:=", <"via">, "layer:=",  <layer id>,
 "sx:=", <start X location>, "sy:=", <start Y location>,
"ex:=", <end X location>,
```

"ey:=", <end Y location>, "h:=", <arc height>, "rad:=", <radians>

<"via">:

text that is the name of the via to use

<layer id>:

an integer that is the id of the layer of the pad of the via to use

<start X location>, <start Y Location>:

doubles that are the X, Y location of the start point of the edge arc

<end X location>, <end Y Location>:

doubles that are the X, Y location of the end point of the edge arc

<arc height>:

double giving the height of the edge arc (0 for a straight edge)

<radians>:

double giving the arc size in radians (0 for a straight edge)

Example:

```
oEditor.AddRefPort Array("Port3"), Array("NAME:Contents", "edge:=",
Array("et:=", "pe", "prim:=", "line_998", "edge:=", 0))
oEditor.AddRefPort Array("Port1"), Array("NAME:Contents", "edge:=",
Array("et:=", "pse", "sel:=", "via_5", "layer:=", 10, "sx:=", 0.0015, "sy:=", 0.0015,
"ex:=", -0.0015, "ey:=", 0.0015, "h:=", 0, "rad:=", 0))
```

AssignCircuitRefPort (Layout Editor)

Use: Assign the internal port as a reference to an existing port; model as a circuit port.

Command: AssignCircuitRefPort

Syntax:

AssignCircuitRefPort

Array(<"port name">, <"port name">, ...),

Array("NAME:Contents",

"edge:=", Array(<edge description>), "edge:=, Array(<edge description>), ...)

Return Value: None

Parameters:

<"layer">

Type: text
Description: layer name.

<"port">
Type: text
Description: a port or pin name.

<edge description> for primitive edges
"et:=", "pe", "prim:=", <"prim">, "edge:=", <edge#>

<"prim">
Type: text
Description: primitive name

<edge#>
Type: integer
Description: edge number on the primitive

<edge description> for via edges
"et:=", "pse", "sel:=", <"via">, "layer:=", <layer id>,
"sx:=", <start X location>, "sy:=", <start Y location>,
"ex:=", <end X location>,
"ey:=", <end Y location>, "h:=", <arc height>, "rad:=",
<radians>

<"via">:
text that is the name of the via to use
<layer id>:
an integer that is the id of the layer of the pad of the via to use
<start X location>, <start Y Location>:
doubles that are the X, Y location of the start point of the edge arc
<end X location>, <end Y Location>:
doubles that are the X, Y location of the end point of the edge arc
<arc height>:
double giving the height of the edge arc (0 for a straight edge)
<radians>:

double giving the arc size in radians (0 for a straight edge)

Example: `oEditor.AssignCircuitRefPort Array("pin_1"), "pin_2"`

AssignRefPort (Layout Editor)

Use: Assign the named internal port as a reference for each of the named ports.

Command: AssignRefPort

Syntax:
`AssignRefPort`
Array(<"port name">, <"port name">, ...),
Array("NAME:Contents",
"edge:=", Array(<edge description>), "edge:=, Array(<edge description>), ...)

Return Value: None

Parameters:
`<"layer">`
Type: text
Description: layer name.

`<"port">`
Type: text
Description: a port or pin name.

`<edge description>`
for primitive edges
`"et:=", "pe", "prim:=", <"prim">, "edge:=", <edge#>`

`<"prim">`
Type: text
Description: primitive name

`<edge#>`
Type: integer
Description: edge number on the primitive

`<edge description>`
for via edges
`"et:=", "pse", "sel:=", <"via">, "layer:=",  <layer id>,`

"sx:=", <start X location>, "sy:=", <start Y location>, "ex:=", <end X location>,
"ey:=", <end Y location>, "h:=", <arc height>, "rad:=", <radians>

<"via">:
text that is the name of the via to use

<layer id>:
an integer that is the id of the layer of the pad of the via to use

<start X location>, <start Y Location>:
doubles that are the X, Y location of the start point of the edge arc

<end X location>, <end Y Location>:
doubles that are the X, Y location of the end point of the edge arc

<arc height>:
double giving the height of the edge arc (0 for a straight edge)

<radians>:
double giving the arc size in radians (0 for a straight edge)

Example:

```
oEditor.AssignRefPort Array("pin_1"), "pin_2"
```

CreateCircuitPort (Layout Editor)

Use: Create a circuit port between two points.

Command: Draw > Create Circuit Ports

Syntax:

```
CreateCircuitPort Array("NAME:Location", "PosLayer:=",
"layer name", "X0:=",  _
```

x-value, "Y0:=", y-value, "NegLayer:=", "layer name", "X1:=", x-value, "Y1:=", y-value)

Return Value: None

Parameters:

<"layer">

Type: text

Description: layer name.

<"port">
Type: text
Description: a port or pin name.

<edge description>
for primitive edges
"et:=", "pe", "prim:=", <"prim">, "edge:=", <edge#>

<"prim">
Type: text
Description: primitive name

<edge#>
Type: integer
Description: edge number on the primitive

<edge description>
for via edges

"et:=", "pse", "sel:=", <"via">, "layer:=", <layer id>,
"sx:=", <start X location>, "sy:=", <start Y location>,
"ex:=", <end X location>,
"ey:=", <end Y location>, "h:=", <arc height>, "rad:=",
<radians>

<"via">:
text that is the name of the via to use
<layer id>:
an integer that is the id of the layer of the pad of the via to use

<start X location>, <start Y Location>:
doubles that are the X, Y location of the start point of the edge arc

<end X location>, <end Y Location>:
doubles that are the X, Y location of the end point of the edge arc

<arc height>:

double giving the height of the edge arc (0 for a straight edge)

<radians>:

double giving the arc size in radians (0 for a straight edge)

Example:

```
oEditor.CreateCircuitPort Array("NAME:Location",
"PosLayer:=", "Top", "X0:=",  _
-0.003, "Y0:=", 0.003, "NegLayer:=", "Top", "X1:=", 0.002, "Y1:=", 0.003)
```

CreateComponent (Layout Editor)

Use: Places a component.

Syntax: **CreateComponent <Array>**

Return Value: Returns the name of the newly created component.

Parameters:

```
Array("NAME:Contents",
"definition_name:=", <component name>,
"placement:=", Array("x:=", <x position>, "y:=", <y posi-
tion>),
"layer:=", <placement layer name>,
"StackupLayers:=", Array("<footprint stackup
layer>:<design stackup layer>",  …),
"DrawLayers:=", Array("<footprint layer>:<design layer>",
…))
```

Example:

```
oEditor.CreateComponent Array("NAME:Contents",
"definition_name:=", "MSTRL",
"placement:=", Array("x:=", "-13mm", "y:=", "5mm"),
"layer:=", "Top",
"StackupLayers:=", Array("Top:Top",  "0:Dielectric",
"0:Ground"),
"DrawLayers:=", Array("Measures:Measures", "Assembly
Top:Assembly Top", "Silkscreen Top:Silkscreen Top",
"Wirebonds:0"))
```

Notes:

Each Layer mapping is specified as a single text string in quotes, "<footprint layer>:<design layer>". If the layer does not map to anything, use a "0" for the layer. For example, you can use

"Measures:0", if the footprint "Measures" layer does not map to a design layer. Use "0:Dielectric", if the design "Dielectric" layer does not map to a footprint layer.

Note, also, that the "StackupLayers" and the "DrawLayers" arguments may be omitted, in which case, the mappings are determined automatically.

CreateEdgePort (Layout Editor)

Use: Creates an edge port using the specified edges.

Command: **Draw > Port > Create**

Right-click > Port > Create

Also available through Tool Bar icon

Syntax:

```
CreateEdgePort
Array("NAME:Contents",
"edge:=", Array(<edge description>), "edge:=, Array(<edge description>), ...
"external:=", <flag>)
```

Return Value: Text containing the name of the created edge port. (Returns an empty name if the edge port is not created.)

Parameters:

```
<edge description> for primitive edges
"et:=", "pe", "prim:=", <"prim">, "edge:=", <edge#>

<"prim">: text that is the primitive name

<edge#>: integer that is the edge number on the primitive

<edge description>
```

for via edges

```
"et:=", "pse", "sel:=", <"via">, "layer:=",  <layer id>,
 "sx:=", <start X location>, "sy:=", <start Y location>,
"ex:=", <end X location>,
"ey:=", <end Y location>, "h:=", <arc height>, "rad:=",
<radians>

<"via">: text that is the name of the via to use
<layer id>:
```

an integer that is the id of the layer of the pad of the via to use

<start X location>, <start Y Location>:
doubles that are the X, Y location of the start point of the edge arc
<end X location>, <end Y Location>:
doubles that are the X, Y location of the end point of the edge arc

<arc height>:
double giving the height of the edge arc (0 for a straight edge)
<radians>:
double giving the arc size in radians (0 for a straight edge)
<flag>
true if the port is an external port
false if the port is an internal port

Example:

```
oEditor.CreateEdgePort Array("NAME:Contents", "edge:=",
Array("et:=", "pe",
"prim:=", "rect_167", "edge:=", 0), "edge:=",
Array("et:=", "pe", "prim:=",
"rect_167", "edge:=", 3), "external:=", true)

oEditor.CreateEdgePort Array("NAME:Contents", "edge:=",
Array("et:=", "pse",
"sel:=",  "via_0", "layer:=", 10, "sx:=", -0.0015,
"sy:=", -0.0015, "ex:=", 0.0015,
"ey:=", -0.0015, "h:=", 0, "rad:=", 0), "external:=",
true)
```

CreateHole (Layout Editor)

Use: Places a Hole object.

Syntax: **CreateHole** <Array>

Return Value: Returns the name of the newly created Hole

Parameters:

```
Array("NAME:Contents",
Array("NAME:full_definition",
"type:=", "hole",
Array("NAME:Properties",
"VariableProp:=",
  Array("radius", "D", "", <value>), // radius
```

```
"VariableProp:=",
Array("sides", "D", "", integer)),//side count
"from_layer:=", <layer_name>,
"to_layer:=",   <layer_name>),
"placement:=", <vpoint>,  // placement position
"layer:=",  <layer_name>) // placement layer

<vpoint> :
Array("x:=", <value>,  // X coordinate
"y:=", <value>)  // Y coordinate
```

Example:

```
oEditor.CreateHole
Array("NAME:Contents",
Array("NAME:full_definition",
"type:=", "hole",
Array("NAME:Properties",
"VariableProp:=",
Array("radius", "D", "", "0.635mm"),
"VariableProp:=",
Array("sides", "D", "", "6")),
"from_layer:=",  6,
"to_layer:=", 6),
"placement:=", Array("x:=", "-36mm", "y:=", "-9mm"),
"layer:=",  "Top")
```

CreateMeasure (Layout Editor)

Use: Creates a measurement.

Syntax: `CreateMeasure`

Use: Returns the name of the created object.

Parameters:

```
Array("NAME:Contents",
"MeasurementGeometry:=",
Array("LayerName:=", <layer_name>,  // layer
"lw:=", <value>, // line width
"sx:=", <value>, // start X coordinate
"sy:=", <value>, // start Y coordinate
```

```
"ex:=", <value>, // end X coordinate
"ey:=", <value>, // end Y coordinate
   <text_style>)

<text_style> :
"name:=",   <quoted string>, // its name
"isPlot:=", <bool>,
"font:=",   <font_name>,
"size:=", double,     // size in current units
"angle:=",  <value>,
"weight:=", <text_weight>,
"just:=",   <text_justification>,
"mirror:=", <bool>,
"scales:=", <bool>))
```

Example:

```
oEditor.CreateMeasure
Array("NAME:Contents",
"MeasurementGeometry:=",
Array("Layer:=",  0,
"Name:=", "Measurement_2",
"LayerName:=", "Measures",
"lw:=", "0mm",
"sx:=", "-32mm",
"sy:=", "-13mm",
"ex:=", "32mm",
"ey:=", "-11mm",
"name:=", "<DefaultAnnotation>",
"isPlot:=", false,
"font:=", "Arial",
"size:=", 10,
"angle:=",  "0deg",
"weight:=", 3,
"just:=", 4,
"mirror:=", false,
"scales:=", false))
```

CreateNPort (Layout Editor)

Use: Creates an N-Port definition and component and adds them to the current project, layout, and schematic.

Syntax: **CreateNPort**<Array>

Return Value: Returns the name of the newly created component.

Parameters:

```
Array("NAME:Contents",
"definition_name:=",<nport_definition_name>,
"placement:=", <component_placement>,
"layer:=", <placement_layer_name>,
<nport_data_definition>)
```

<nport_definition_name>:

quoted string (name of the component definition)

<component_placement>:

Array("x:=", <value>, // x coordinate
"y:=", <value>, // y coordinate
"scaling:=", <value>) // double

<nport_data_definition> - see the I/O format in TODO

Example:

```
oEditor.CreateNPort
    Array("NAME:Contents",
          "definition_name:=", "NetworkData3",
          "placement:=",
              Array("x:=",  "-9mm",
                    "y:=", "-5mm",
                    "scaling:=", "2"),
          "layer:=", "Symbols",
           Array("NAME:NPortData",
               Array("NAME:NetworkData2",
                     "filelocation:=", "UsePath",
                     "filename:=", "",
                     "domain:=", "frequency",
                     "numberofports:=", 2,
```

```
                                "datamode:=", "Import",
                                "devicename:=", "", "
                                ImpedanceTab:=", 1,
                                "NoiseDataTab:=", 1,
                                "DCBehaviorTab:=",  1,
                                "SolutionName:=", "",
                                "dcbehavior:=", "DCOpen",
                               "displayformat:=", "MagnitudePhase",
                                "datatype:=", "SMatrix",
                                "interptype:=", "Linear",
                                "extraptype:=",  "Same as interpola-
tion",
                                "ShowRefPin:=", 0,
                                "RefNodeCheckbox:=", 1)))
```

CreatePin (Layout Editor)

Use: Creates a pin.

Syntax: **CreatePin** <pin_description>

Return Value: Returns the name of the newly created pin.

Parameters:

<pin_description>:

Array("NAME:Contents",
Array("NAME:Port", "Name:=", <object_name>), // not used
"Rotation:=", Array(<value>), // rotation
"Offset:=", <vpoint>, // position
"Padstack:=", <quoted_string>) // padstack name

Example:

```
oEditor.CreatePin
Array("NAME:Contents",
Array("NAME:Port", "Name:=", "Pin_1"),
                  "Rotation:=",  Array("0deg"),
                 "Offset:=",    Array("x:=", "-2mm", "y:=",
"-1mm"),
                  "Padstack:=", "NoPad SMT East")
```

CreateTrace (Layout Editor)

Use: Create a trace with a manually specified path between pins, ports, or selected edges.

Command: **Draw > Route > Manual**

Syntax:

```
CreateTrace
Array("NAME:options", "Layer1:=", <"layer">, "Layer2:=",  <"layer">),
Array("NAME:Lines", "Num:=", <n_lines>, "l0:=",
Array("Name:=", <"line">, "LayerName:=", <"layer">, "lw:=", <"width">,
"endstyle:=", <endstyle>, "joinstyle:=", <joinstyle>, "n:=", <n_vertex>,
"U:=", <"units">, "x:=", <value>, "y:=", <value>, ), …),
Array("NAME:Vias", "Num:=", <n_vias>,
Array("NAME:v1", "name:=", <"via">, "ReferencedPadstack:=", <"padstack">,
"vposition:=", Array("x:=", <"vertex">, "y:=", <"vertex">), "vrotation:=",
Array(<"angle">), "overrides hole:=", <override>, "hole diameter:=",
Array(<"diameter">), "highest_layer:=", <"layer">, "lowest_layer:=", <"layer">), …),
Array("NAME:elements", <"port">, …),
Array("NAME:EdgeRefs", "edge:=", Array(<edge description>), …)
```

Return Value: None

Parameters:

`<"layer">`

Type: text

Description: layer name

`<n_lines>`

Type: integer

Description: number of line definitions following

`<"line">`

Type: text

Description: line name

<"width">
Type: text
Description: line width; value with units, e.g. "1mm"

<endstyle>
Type: integer
Description: end (cap) style value for the line.

<joinstyle>
Type: integer
Description: join style value.

<n_vertex>
Type: integer
Description: number of vertices in the line

<value>
Type: double
Description: simple value, e.g. 10

<n_vias>
Type: integer
Description: number of via defintions following

<"via">
Type: text
Description: via name

<"padstack">
Type: text
Description: padstack defintion name.

<"angle">
Type: text

Description: via orientation; value with units, e.g. "180deg"

<override>
Type: boolean (true or false)
Description: if true, the diameter is used to override the via hole definition.

<"diameter">
Type: text
Description: via hole diameter override; a value with units, e.g. "1mm"

<"port">
Type: text
Description: a port or pin name.

<edge description>
for primitive edges
"et:=", "pe", "prim:=", <"prim">, "edge:=", <edge#>

<"prim">
Type: text
Description: primitive name

<edge#>
Type: integer
Description: edge number on the primitive

<edge description>
for via edges
"et:=", "pse", "sel:=", <"via">, "layer:=", <layer id>,
"sx:=", <start X location>, "sy:=", <start Y location>,
"ex:=", <end X location>,
"ey:=", <end Y location>, "h:=", <arc height>, "rad:=",
<radians>

<"via">:
text that is the name of the via to use

<layer id>:
an integer that is the id of the layer of the pad of the via to use

<start X location>, <start Y Location>:
doubles that are the X, Y location of the start point of the edge arc
<end X location>, <end Y Location>:
doubles that are the X, Y location of the end point of the edge arc

<arc height>:
double giving the height of the edge arc (0 for a straight edge)
<radians>:
double giving the arc size in radians (0 for a straight edge)

Example:

```
oEditor.CreateTrace Array("NAME:options", "Layer1:=",
"Top",
"Layer2:=", "Ground"), Array("NAME:elements"),
Array("NAME:EdgeRefs",
"edge:=", Array("et:=", "pe", "prim:=", "rect_4",
"edge:=", 3), "edge:=",
Array("et:=", "pse", "sel:=", "via_3", "layer:=", 8,
"sx:=", -0.0015, "sy:=", 0.0015,
"ex:=", -0.0015, "ey:=", -0.0015, "h:=", 0, "rad:=",  0))
```

CreateVia (Layout Editor)

Use: Creates a new via.

Syntax: **CreateVia** <via_description>

Return Value: Returns the name of the created via

Parameters:

```
<via_description>:
Array("NAME:Contents",
"name:=",      <object_name>,  // not used
"vposition:=", <vpoint>,    // position
"rotation:=",  double,      // rotation in radians
"overrides hole:=", <bool>,// overrides or not padstack
hole
"hole diameter:=", Array(<value>), // via diameter
"ReferencedPadstack:=", <quoted_string>, // padstack name
```

```
                "highest_layer:=", <layer_name>,
                "lowest_layer:=",  <layer_name>)
```

Example:

```
oEditor.CreateVia
    Array("NAME:Contents",
          "name:=", "",
          "vposition:=", Array("x:=", "-22mm", "y:=",
"5mm"),
          "rotation:=", 0,
          "overrides hole:=", false,
          "hole diameter:=", Array( "1mm"),
          "ReferencedPadstack:=", "Round 1mm/0.5mm",
          "highest_layer:=", "Top",
          "lowest_layer:=",  "Bottom")
```

Object Movement and Modification Methods

The documented object movement methods are available.

Connect (Layout Editor)

Copy (Layout Editor)

Delete (Layout Editor)

Disconnect (Layout Editor)

Edit (Layout Editor)

FlipHorizontal (Layout Editor)

FlipVertical (Layout Editor)

Move (Layout Editor)

Paste (Layout Editor)

Rotate (Layout Editor)

Connect (Layout Editor)

Use: Causes connecting of the referenced object. After the command the objects will be in the same electrical net.

Syntax:

```
Connect Array ("NAME: elements",
   <object_name>,   // 1st object
   <object_name>,   // 2nd object, if any
   …)               // etc
```

Example:

```
oEditor.Connect Array("NAME:elements", "2:n2", "cir-
cle_4")
```

Copy (Layout Editor)

Use: Copies the referenced objects to the clipboard.

Syntax:
```
Copy Array (<object_name>,   // 1st object
<object_name>,   // 2nd object
```

Return Value: Returns the name of the new net.

Example:
```
oEditor.Copy Array("circle_0", "rect_2")
```

Delete (Layout Editor)

Use: Deletes one or more objects.

Syntax:
```
Delete <object_name_list>  // names of the objects to be deleted
<object_name_list>:
    Array ("NAME:elements", <object_name>, <object_name>, …)
```

Example:
```
oEditor.Delete Array("NAME:elements", "circle_0", "rect_2")
```

Disconnect (Layout Editor)

Use: Removes the specified object(s) from their current electrical net(s) (if any).

Syntax:
```
Disconnect Array ("NAME: elements",
<object_name>,   // 1st object
<object_name>,   // 2nd object, if any
…)               // etc
```

Example:
```
oEditor.Disonnect Array("NAME:elements", "2:n2")
```

Edit (Layout Editor)

Use: Causes modification of one or more existing object(s).

Syntax:
```
Edit <Array("NAME:items".....)>,
```

Parameters:
```
<Array("NAME:items"
<edit_object_info>, // one object info
<edit_object_info>, // another one
…)               // etc
```

```
<edit_object_info>:
    Array("NAME:item",
          "name:=", <object_name>, // name of the object
          <object_description>)   // 'new' object state,
type should be consistent with <object_name>:

<object_description>:
    <circle_description>         |
    <rectangle_description>      |
    <line_description>           |
    <polyon_description>         |
    <text_description>           |
    <circle_void_description>    |
    <rectangle_void_description> |
    <line_void_description>      |
    <polyon_void_description>    |
    <component_description>      |
    <pin_description>            |
    <via_description>
```

Example:

```
oEditor.Edit Array("NAME:items", Array("NAME:item",
"name:=", "circle_0", Array("NAME:contents",
"circleGeometry:=", Array("Layer:=", 6,
"Name:=", "circle_0", "LayerName:=", "Top",
"lw:=", "0mm", "x:=", "-0.008meter",
"y:=", "13.2924281984334mm", "r:=",
10.2924281984334mm")))
```

FlipHorizontal (Layout Editor)

Use: Causes horizontal flipping of one or more specified objects (mirroring about a vertical axis given its X coordinate).

Syntax:

```
FlipHorizontal <object_name_list>,//objects to be flipped
<position_or_center>  // (X of flip_axis / not used)
```

Example:

```
oEditor.FlipHorizontal Array("NAME:elements", "circle_0",
"rect_2"),
```

```
Array(0.01, -0.001)
```

FlipVertical (Layout Editor)

Use: Causes vertical flipping of one or more specified objects (mirroring about a horizontal axis given its Y coordinate).

Syntax:
```
FlipVertical <object_name_list>, //objects to be flipped
<position_or_center>  // not used, flip axis
```

Example:
```
oEditor.FlipVertical Array("NAME:elements", "circle_0",
"rect_2"), Array(0.01, -0.001)
```

Move (Layout Editor)

Use: Causes movement of one or more specified objects by a specified offset.

Syntax:
```
Move <object_name_list>,// names of the objects to be
moved
<position_or_center>  // move offset
<position_or_center>:
Array(double, double)) // X and Y in SI (meters)
```

Example:
```
oEditor.Move Array("NAME:elements", "circle_0",
"rect_2"),
Array(0.0165613577023499, -0.001)
```

Paste (Layout Editor)

Use: Pastes the objects currently in clipboard; the argument specifies the offset of the new (copy) object(s) from their initial position(s).

Syntax:
```
Paste Array("NAME:offset",
"xy:=",
Array(double, double))  // X/Y offset
```

Example:
```
oEditor.Paste Array("NAME:offset", "xy:=", Array(0.034,
0.013))
```

Rotate (Layout Editor)

Use: Causes rotation of one or more specified objects 90 degrees counterclockwise about a specified center.

Syntax:
```
Rotate <object_name_list>,  // names of the objects to be
rotated
```

```
<position_or_center>  // rotation center
```

Example:

```
oEditor.Rotate Array("NAME:elements", "circle_0",
"rect_2"),
Array(0.0165613577023499, -0.001))
```

Activation and Deactivation Methods

The following activation/deactivation methods are available.

Activate <object_names>

DeactivateOpen <object_names>

DeactivateShort <object_names>

Delete (Layout Editior)

Activate (Layout Editor)

Use: Causes activation of one or more components.

Syntax:

```
Activate Array("NAME:components",
<object_name>,   // 1st component name
<object_name>,   // 2nd component, if any
   …)               // etc
```

Example:

```
oEditor.Activate Array("NAME:components", "2", "3")
```

DeactivateOpen (Layout Editor)

Use: Causes deactivation of one or more components to *open circuit* state.

Syntax:

```
ActivateOpen Array ("NAME:components",
    <object_name>,  // 1st component name
    <object_name>,  // 2nd component, if any
     …)          // etc
```

Example:

```
oEditor.DeactivateShort Array("NAME:components", "2",
"3")
```

DeactivateShort (Layout Editor)

Use: Causes deactivation of one or more components to *short circuit* state.

Syntax:

```
ActivateShort Array("NAME:components",
<object_name>,    // 1st component name
<object_name>,    // 2nd component, if any
…)        // etc
```

Example:

```
oEditor.DeactivateShort Array("NAME:components", "2",
"3")
```

Delete (Layout Editor)

Use: Deletes one or more objects.

Syntax:

```
Delete <object_name_list>  // names of the objects to be
deleted
<object_name_list>:
Array ("NAME:elements", <object_name>,  <object_name>, …)
```

Example:

```
oEditor.Delete Array("NAME:elements", "circle_0",
"rect_2")
```

Layout and Geometry Interrogation Methods

From the Layout scripting object, the following methods are available for layout and geometry interrogation.

The topics for this section include:

Layout Interrogation

Point Object

Polygon Object

Layout Interrogation (Layout Editor)

Methods employing Point and Polygon objects are specified in SI units (e.g. meters).

The following methods are available.

- Point (Layout Editor)
- Polygon (Layout Editor)
- FindObjects (Layout Editor
- FilterObjectList (Layout Editor)
- GetCSObjects (Layout Editor)
- GetPolygon (Layout Editor)

Point (Layout Editor)

Use: Create and return a Point Object.

Syntax: `oLayout.Point()`

Return Value: Point object.

Example:
```
oPoint = oLayout.Point()
```

Polygon (Layout Editor)

Use: Create and return a Polygon Object.

Syntax: `oLayout.Polygon()`

Return Value: Polygon object.

Example:
```
oPolygon = oLayout.Polygon()
```

FindObjects (Layout Editor)

Use: Get a list of object names using a key/value pair.

Syntax: `oLayout(<Key>, <Value)`

Parameters: `Key, Value.`

Queries use a basic string match, so '*', for instance, matches anything. All matching is case insensitive.

Valid keys are:

— 'Name': By object name

— 'Type': By object type {'pin', 'via', 'rect', 'arc', 'line', 'poly', 'plg', 'circle void', 'line void', 'rect void', 'poly void', 'plg void', 'text', 'cell', 'Measurement', 'Port', 'Port Instance', 'Port Instance Port', 'Edge Port', 'component', 'CS', 'S3D'}.

— 'Layer': By layer. '*' matches all layers. Vias and pins match using their defined layer range. 'Multi' matches (non via/pin) multi-layer objects.

— 'Net': By net assignment.

Return Value: List of object names.

Example:
```
vias = oLayout.FindObjects('Type', 'Via')
```

FilterObjectList (Layout Editor)

Use: Filter a list of objects using a key/value pair. Valid key/value combinations can be found in Find Objects.

Syntax: `oLayout.FindObjects(<Key>, <Value)`

Return Value: Filtered list of object names.

Example:
```
RFIN_pins = oLayout.FilterObjectList('Type', 'Pin',
oLayout.FindObjects('Net', 'RFIN'))
```

GetCSObjects (Layout Editor)

Use: Given a coordinate system name, returns a list of the objects inside. Coordinate system names can be found using FindObjects('Type', 'CS'). See Find Objects.

Syntax: `oLayout.GetCSObjects(<CS_name>)`

Return Value: List of objects in the coordinate system.

Example:
```
CS_objects = oLayout.GetCSObjects("CS_1")
```

GetPolygon (Layout Editor)

Use: Get a Polygon Object for the specified object name, or None if the object does not exist. Returned polygon is as rendered, in SI units

Syntax: `oLayout.GetPolygon(<Name>)`

Return Value: Polygon object.

Example:
```
oPolygon = oLayout.GetPolygon('line_37')
```

GetPolygonDef (Layout Editor)

Use: Get a Polygon Object for the specified object name, or None if the object does not exist. Returned polygon is as rendered, in SI units. For trace types, this corresponds to the center line.

Syntax: `oLayout.GetPolygonDef(<Name>)`

Return Value: Polygon object.

Example:
```
oPolygon = oLayout.GetPolygonDef('line_37')
```

GetBBox (Layout Editor)

Use: Get a Polygon Object defining the bounding box for the specified object name, or None if the object does not exist.

Syntax: `oLayout.GetBBox(<Name>)`

Return Value: Polygon object.

Example:

```
oPolygon = oLayout.GetBBox('rect_10')
```

FindObjectsByPolygon (Layout Editor)

Use: Get a list of all objects intersecting the specified Polygon object on the given layer.

Syntax:

```
oLayout.FindObjectsByPolygon(<oPolygopn>, <Layer>)
```

Return Value: Object list.

Example:

```
# Find all objects intersecting box = {(0,0), (1e-3,1e-3)}
p0 = oLayout.Point().Set(0, 0)
p1 = oLayout.Point().Set(1e-3, 0)
p2 = oLayout.Point().Set(1e-3, 1e-3)
p3 = oLayout.Point().Set(0, 1e-3)
box = oLay-
out.Polyon().AddPoint(p0).AddPoint(p1).AddPoint(p2).AddPo
int(p3).SetClosed(True)
objs = oLayout.FindObjectsByPolygon(box, '*')
```

FindObjectsByPoint (Layout Editor)

Use: Get a list of all objects intersected by the specified Point object on a given layer.

Syntax:

```
oLayout.FindObjectsByPoint(<oPoint>, <Layer>)
```

Return Value: Object list.

Example:

```
objs = oLayout.FindObjectsByPoint(oLayout.Point().Set(-1.149e-3, 3.465e-3), 'L3')
```

CreateObjectFromPolygon (Layout Editor)

Use: Create a polygon on the specified layer and net from the provided Polygon object. Return the name of the new object. Net is optional.

Syntax:

```
oLayout.CreateObjectFromPolygon(<oPolygon>, <Layer>, <Net>)
```

Return Value: Object name.

Example:

```
newobj = oLayout.CreateObjectFromPolygon(oPolygon, 'L1',
'GND')
```

CreateLineFromPolygon (Layout Editor)

Use: Create a line/trace object on the specified layer and net from the provided Polygon object. Net is optional.

Syntax:
```
oLayout.CreateLineFromPolygon(<oPolygon>, <width>,
<bendType>, <startCapType>, <endCapType>, <Layer>, <Net>)
```

Parameters:

`width - line width, in meters`

bendType - {'corner', 'round'}

startCapType, endCapType - {'flat', 'extended', 'round'}

Example:

```
newobj = oLayout.CreateLineFromPolygon(oPolygon, 0.1e-3,
'round', 'flat', 'round', 'L1', 'DQ0')
```

Point Object (Layout Editor)

The Point object provides a point container in a 2D Cartesian coordinate system. Point objects have no link to specific Layout primitives. All values and interfaces are specified in meters.

A Layout object is used to create a Point as:

```
oPoint = oLayout.Point()
```

The following methods are available.

Set
SetX
GetX
SetY
GetY
SetArc
IsArc
IsEqual
Mag
Distance
Cross
Move
Rotate
Normalize

DistanceFromLine

ClosestPointOnLine

Set (Layout Editor)

Use: Set both the x and y -coordinates of the Point, in meters. Return *this* Point.

Syntax: `oPoint.Set(<x>, <y>)`

Return Value: This Point object.

Example:
```
oPoint = oPoint.Set(1e-3, 1e-3)
```

SetX (Layout Editor)

Use: Set the x-coordinate of the Point, in meters. Return *this* Point.

Syntax: `oPoint.SetX(<x>)`

Return Value: This Point object.

Example:
```
oPoint = oPoint.SetX(1e-3)
```

GetX (Layout Editor)

Use: Get the x-coordinate of the Point, in meters.

Syntax: `oPoint.GetX()`

Return Value: X-coordinate.

Example:
```
x = oPoint.GetX()
```

SetY (Layout Editor)

Use: Set the y-coordinate of the Point, in meters. Return *this* Point.

Syntax: `oPoint.SetY(<y>)`

Return Value: This Point object.

Example:
```
oPoint = oPoint.SetY(1e-3)
```

GetY (Layout Editor)

Use: Get the y-coordinate of the Point, in meters.

Syntax: `oPoint.GetY()`

Return Value: Y-coordinate.

Example:
```
y = oPoint.GetY()
```

SetArc (Layout Editor)

Use: Set whether the given point is an arc. For arc-points, the x-coordinate is used to specify signed height. Return *this* Point.

Syntax: `oPoint.SetArc(<isArc>)`

Return Value: This Point object.

Example: `oPoint = oPoint.SetArc(True)`

IsArc (Layout Editor)

Use: Get whether the given point is an arc.

Syntax: `oPoint.IsArc()`

Example: `isArc = oPoint.IsArc()`

IsEqual (Layout Editor)

Use: Test whether this point is equal to another. If tolerance is not specified, a default 1e-9 is used.

Syntax: `oPoint.IsEqual(<oPoint>, [<Tolerance>])`

Example:

```
isEqual1 = oPointThis.IsEqual(oPointOther)
isEqual2 = oPointThis.IsEqual(oPointOther, 1e-6)
```

Mag (Layout Editor)

Use: Get the magnitude of the given Point

Syntax: `oPoint.Mag()`

Example: `mag = oPoint.Mag()`

Distance (Layout Editor)

Use: Get the distance from this point to another.

Syntax: `oPointThis.Distance(oPointOther)`

Example: `dist = oPointThis.Distance(oPointOther)`

Cross (Layout Editor)

Use: Get the cross product of this point with another. In 2D, this is simply the determinant of [x1 x2; y1 y2].

Syntax: `oPointThis.Cross(oPointOther)`

Example: `cross = oPointThis.Cross(oPointOther)`

Move (Layout Editor)

Use: Translate this point by the vector specified in another. Return this point.

Syntax: `oPoint.Move(<oPoint>)`

Return Value: This Point object.

Example: `oPoint = oPoint.Move(oLayout.Point().Set(1,1))`

Rotate (Layout Editor)

Use: Rotate this point counter clockwise (CCW) about a center specified by another. Angle is in radians. Return this point.

Syntax: `oPoint.Rotate(<oAngRad>, <oPointCenter>)`

Return Value: This Point object.

Example: `oPoint = oPoint.Rotate(pi/2, oLayout.Point())`

Normalize (Layout Editor)

Use: Normalize this point. Return this point.

Syntax: `oPoint.Normalize()`

Return Value: This Point object.

Example: `oPoint = oPoint.Normalize()`

DistanceFromLine (Layout Editor)

Use: Get the distance from this point to a line defined by two additional points.

Syntax: `oPoint.DistanceFromLine(<oPoint>,<oPoint>)`

Example:
```
dist = oPoint.DistanceFromLine(oLayout.Point().Set(0,0),
oLayout.Point().Set(1,0))
```

ClosestPointOnLine (Layout Editor)

Use: Get the closest point to this on a line defined by the two input points.

Syntax: `oPoint.ClosestPointOnLine(<oPoint>, <oPoint>)`

Return Value: The closest point, as a Point Object.

Example:
```
oPointClosest =
oPoint.ClosestPointOnLine(oLayout.Point().Set(0,0),
oLayout.Point().Set(1,0))
```

Polygon Object (Layout Editor)

The Polygon object provides a polygon container in a 2D Cartesian coordinate system. Objects have no link to specific Layout primitives. All values and interfaces are specified in meters.

A Layout object can be used to create an empty, and open, Polygon as:

```
oPoint = oLayout.Polygon()
```

The following methods are available.

AddPoint
SetClosed
IsClosed
Move
Rotate
Scale
MirrorX
GetPoints
AddHole
HasHoles
GetHoles
HasArcs
HasSelfIntersections
BBoxLL
BBoxUR
IsParametric
IsConvex
IsPoint
IsSegment
IsArc
IsBox
IsCircle
GetBoundingCircleCenter
GetBoundingCircleRadius
Area
PointInPolygon
CircleIntersectsPolygon
GetIntersectionType

GetClosestPoint
GetClosestPoints
Unite
Intersect
Subtract
Xor

AddPoint (Layout Editor)

Command: Add a point to the polygon. Return this Polygon.

Syntax: `oPolygon.AddPoint(<oPoint>)`

Return Value: This Polygon object.

Example:
```
oPolygon=oPolygon.AddPoint(oLayout.Point().Set(0,1))
```

SetClosed (Layout Editor)

Use: Set whether a polygon is open or closed. Return this Polygon.

Syntax: `oPolygon.SetClosed(isClosed)`

Return Value: This Polygon object.

Example:
```
oPolygon = oPolygon.SetClosed(True)
```

IsClosed (Layout Editor)

Use: Get whether a polygon is open or closed.

Syntax: `oPolygon.IsClosed()`

Example:
```
isClosed = oPolygon.IsClosed()
```

Move(Layout Editor)

Use: Translate this Polygon by the vector specified in the provided Point. Return this Polygon.

Syntax: `oPolygon.Move(<oPoint>)`

Return Value: This Polygon object.

Example:
```
oPolygon = oPolygon.Move(oLayout.Point().Set(1,1))
```

Rotate (Layout Editor)

Use: Rotate this Polygon counter clockwise (CCW) about a center specified by the provided Point. Angle is in radians. Return this Polygon.

Syntax: `oPolygon.Rotate(<AngRad>, <oPoint>)`

Return Value: This Polygon object.

Example: `oPolygon = oPolygon.Rotate(pi/2, oLayout.Point())`

Scale (Layout Editor)

Use: Scale the polygon by the specified factor. Return this Polygon.

Syntax: `oPolygon.Scale(<factor>)`

Return Value: This Polygon object.

Example: `oPolygon = oPolygon.Scale(2)`

MirrorX (Layout Editor)

Use: Mirror this Polygon about the line defined by the provided X-coordinate. Return this Polygon.

Syntax: `oPolygon.MirrorX(x-coord)`

Return Value: This Polygon object.

Example: `oPolygon = oPolygon.MirrorX(0.0)`

GetPoints (Layout Editor)

Use: Get a list of Point objects defining this Polygon.

Syntax: `oPolygon.GetPoints()`

Return Value: List of Point objects.

Example: `ptList = oPolygon.GetPoints()`

AddHole (Layout Editor)

Use: Add a hole to a Polygon. The hole is defined by the input Polygon object. Return this Polygon.

Syntax: `oPolygon.AddHole(oPolygon)`

Return Value: This Polygon object.

Example: `oPolygon = oPolygon.AddHole(oPolygonHole)`

HasHoles (Layout Editor)

Use: Get whether a Polygon has holes.

Syntax: `oPolygon.HasHoles()`

Example: `HasHoles = oPolygon.HasHoles()`

GetHoles (Layout Editor)

Use: Get a list of holes owned by this Polygon.

Syntax: `oPolygon.GetHoles()`

Return Value: List of holes as Polygon objects.

Example:
```
HoleList = oPolygon.GetHoles()
```

HasArcs (Layout Editor)

Use: Get whether this Polygon has any arc-segments.

Syntax: `oPolygon.HasArcs()`

Example:
```
HasArcs = oPolygon.HasArcs()
```

HasSelfIntersections (Layout Editor)

Use: Check whether this Polygon has any self-intersections. Default tolerance is 1e-9, or it may be explicitly provided.

Syntax: `oPolygon.HasSelfIntersections([<tolerance>])`

Example:
```
hasSelfInt1 = oPolygon.HasSelfIntersections()  # default tol=1e-9
hasSelfInt2 = oPolygon.HasSelfIntersections(1e-6)
```

BBoxLL (Layout Editor)

Use: Get the lower-left point for the bounding-box region of this Polygon.

Syntax: `oPolygon.BBoxLL()`

Return Value: A Point object defining the lower-left coordinate.

Example:
```
oPoint = oPolygon.BBoxLL()
```

BBoxUR (Layout Editor)

Use: Get the upper-right point for the bounding-box region of this Polygon.

Syntax: `oPolygon.BBoxUR()`

Return Value: A Point object defining the upper-right coordinate.

Example:
```
oPoint = oPolygon.BBoxUR()
```

IsParametric (Layout Editor)

Use: Check whether this Polygon is defined parametrically.

Syntax: `oPolygon.IsParametric()`

Example: `isParametric = oPolygon.IsParametric()`

IsConvex (Layout Editor)

Use: Check whether this Polygon is convex.

Syntax: `oPolygon.IsConvex()`

Example: `isConvex = oPolygon.IsConvex()`

IsPoint (Layout Editor)

Use: Check whether this Polygon is defined by a point.

Syntax: `oPolygon.IsPoint()`

Example: `isPt = oPolygon.IsPoint()`

IsSegment (Layout Editor)

Use: Check whether this Polygon is defined by a segment.

Syntax: `oPolygon.IsSegment`

Example: `isSeg = oPolygon.IsSegment`

IsArc (Layout Editor)

Use: Check whether this Polygon is defined by an arc-segment.

Syntax: `oPolygon.IsArc()`

Example: `isArc = oPolygon.IsArc()`

IsBox (Layout Editor)

Use: Check whether this Polygon defines a box.

Syntax: `oPolygon.IsBox()`

Example: `isBox = oPolygon.IsBox()`

IsCircle (Layout Editor)

Use: Check whether this Polygon defines a circle.

Syntax: `oPolygon.IsCircle()`

Example: `isCircle = oPolygon.IsCircle()`

GetBoundingCircleCenter (Layout Editor)

Use: Get a Point defining the center of a bounding circle for this Polygon.

Syntax: `oPolygon.GetBoundingCircleCenter()`

Return Value: A Point object defining the bounding circle's center coordinate.

Example: `oPoint = oPolygon.GetBoundingCircleCenter()`

GetBoundingCircleRadius (Layout Editor)

Use: Get the radius of a bounding circle for this Polygon.

Syntax: `oPolygon.GetBoundingCircleRadius()`

Return Value: The bouding circle's radius.

Example: `rad = oPolygon.GetBoundingCircleRadius()`

Area (Layout Editor)

Use: Get the area of the Polygon, in square meters.

Syntax: `oPolygon.Area()`

Example: `area = oPolygon.Area()`

PointInPolygon (Layout Editor)

Use: Test whether the provided Point is in this Polygon. Points on the Polygon's boundary are considered in the Polygon.

Syntax: `oPolygon.PointInPolygon(<oPoint>)`

Example: `hit = oPolygon.PointInPolygon(oPoint)`

CircleIntersectsPolygon (Layout Editor)

Use: Test whether the circle defined by the input Point and radius hit this Polygon. Circles tangent to the Polygon's boundary are considered intersecting.

Syntax: `oPolygon.CircleIntersectsPolygon(<oPointCenter>, <radius>)`

Example: `hit = oPolygon.CircleIntersectsPolygon(oPointCenter, 1e-3)`

GetIntersectionType (Layout Editor)

Use: Get the intersection type of this Polygon with another. A tolerance can be optionally specified, or the default 1e-9 will be used.

Syntax: `oPolygon.GetIntersectionType(<oPolygon>)`

Return Value: Integer:

0 - No intersection

1 - This fully inside other

2 - Other fully inside this

3 - Common contour points

4 - Undefined intersection

Example:

```
typ1 = oPolygon.GetIntersectionType(oPolygonOther) # default tol=1e-9
typ2 = oPolygon.GetIntersectionType(oPolygonOther, 1e-6)
```

GetClosestPoint (Layout Editor)

Use: Find the closest Point on this Polygon's boundary to the Point provided.

Syntax: `oPolygon.GetClosestPoint(<oPoint>)`

Return Value: A Point object defining the closest location on this Polygon's boundary.

Example:

```
oPointClosest = oPolygon.GetClosestPoint(oLayout.Point().Set(1.0,0))
```

GetClosestPoints (Layout Editor)

Use: Given another Polygon, find the closest Points on each. The closest Point on this Polygon is returned as the first Point in the list. The closest Point on the other Polygon is returned as the last (or second) Point in the list.

Syntax: `oPolygon.GetClosestPoints(oPolygon)`

Return Value: A list of Point objects.

Example:

```
closestPointList = oPolygon.GetClosestPoints(oPolygonOther)
```

Unite (Layout Editor)

Use: Unite this Polygon with another and return the result as a list of Polygons.

Syntax: `oPolygon.Unite(oPolygon)`

Return Value: Union as a list of Polygon objects.

Example:

```
resList = oPolygon.Unite(oPolygonOther)
```

Intersect (Layout Editor)

Use: Intersect this Polygon with another and return the result as a list of Polygons.

Syntax: `oPolygon.Intersect(oPolygon)`

Return Value: Intersection as a list of Polygon objects.

Example:

```
resList = oPolygon.Intersect(oPolygonOther)
```

Subtract (Layout Editor)

Use: Subtract another Polygon from this and return the result as a list of Polygons.

Syntax: `oPolygon.Subtract(oPolygon)`

Return Value: Subtraction as a list of Polygon objects.

Example:

```
resList = oPolygon.Subtract(oPolygonOther)
```

Xor (Layout Editor)

Use: Exclusive or (XOR) this Polygon with another and return as a list of Polygons.

Syntax: `oPolygon.Xor(oPolygon)`

Return Value: Xor as a list of Polygon objects.

Example:

```
resList = oPolygon.Xor(oPolygonOther)
```

Boolean Operations on Primitives

The following boolean operations on primitives are available.

Unite

Intersect

Subtract

Unite (Layout Editor)

Use: Causes Boolean uniting of 2 or more *primitive* (polygons, rectangles, lines, or circles) objects.

Syntax:

```
Unite Array ("NAME: primitives",
 <object_name>,   // 1st primitive name
 <object_name>,   // 2nd primitive, if any
 …)               // etc
```

Example:

```
oEditor.Unite Array("NAME:primitives", "circle_0",
"rect_2")
```

Intersect (Layout Editor)

Use: Causes Boolean intersecting of 2 or more *primitive* (polygons, rectangles, lines, or circles) objects.

Syntax:

```
Intersect Array ("NAME: primitives",
<object_name>,   // 1st primitive name
<object_name>,   // 2nd primitive, if any
…)       // etc
```

Example:

```
oEditor.Intersect Array("NAME:primitives", "circle_0",
"rect_2")
```

Subtract (Layout Editor)

Use: Causes boolean subtracting of one or more *primitive* (polygons, rectangles, lines, or circles) object(s) from another one.

Syntax:

```
Subtract Array ("NAME: primitives",
<object_name>,   // Primitive to subtract from
<object_name>,   // 1nd primitive to subtract, if any
…) // etc
```

Example:

```
oEditor.Intersect Subtract ("NAME:primitives", "cir-
cle_0", "rect_2")
```

Coordinate System Methods

The following scripts are availabe for use with the Designer Coordinate System.

CreateCS (Layout Editor)

ClearRelative (Layout Editor)

Create3DStructure (Layout Editor)

Group (Layout Editor)

CreateGroupSelected (Layout Editor)

PositionRelative (Layout Editor)

GetCSObjects (Layout Editor)

SetCS (Layout Editor)

CreateCS (Layout Editor)

Use: Inserts a new coordinate system (CS) into the currently active Layout design.

Command: Draw > Coordinate System > Create.

Syntax:

```
There are two forms available:

CreateCS
Array("NAME:Contents",
Array("NAME:full_definition", "type:=", "CS", "N:=", <"CS name">),
"placement:=", Array("x:=", <"x_coord">, "y:=", <"y_coord">),
"Clf:=", <color flag>, "Col:=",  <color> )

CreateCS
Array("NAME:Contents",
"Name:=", <"CS name">),
"RelPos:=", Array(edge description))
```

Return Value: Text containing the name of the created relative coordinate system.

Parameters:

`<"CS name">`

Text with the requested name for the relative coordinate system

`<x_coord>`

Text that is the X location for the relative coordinate system origin

`<y_coord>`

Text that is the Y location for the relative coordinate system origin

// the following arguments are optional

`<color flag>`

True if a color is being specified by the next parameter

<color>
Integer specifying the color for the relative coordinate system.
Only used if color flag is true;

<edge description> for primitive edges

"t:=", "pe", "from:=", <"prim">, "pos:=", <edge position>, "et:=", "pe", "prim:=", <"prim">, "edge:=", <edge#>

<"prim">: text that is the primitive name

<edge position>
double between 0 and 1, inclusive
0 indicates the start of the edge
1 indicates the end of the edge
values in between are positions along the edge

<edge#>: integer that is the edge number on the primitive

<edge description> for via edges

"t:=", "pe", "from:=", <"via">, "pos:=", <edge position>,
"et:=", "pse", "sel:=", <"via">, "layer:=", <layer id>,
"sx:=", <start X location>, "sy:=", <start Y location>,
"ex:=", <end X location>, "ey:=", <end Y location>,
"h:=", <arc height>, "rad:=", <radians>

<"via">:
text that is the name of the via to use

<layer id>:
an integer that is the id of the layer of the pad of the via to use <start X location>,

<start Y Location>:
doubles that are the X, Y location of the start point of the edge arc <end X loca-

tion>, <end Y Location>:

doubles that are the X, Y location of the end point of the edge arc

<arc height>:

double giving the height of the edge arc (0 for a straight edge)

<radians>:

double giving the arc size in radians (0 for a straight edge)

Example:

```
oEditor.CreateCS Array("NAME:Contents",
Array("NAME:full_definition",
"type:=", "CS", "N:=", "CS_232"), "placement:=",
Array("x:=", "-15mm", "y:=",
"15mm"), "Clf:=", false, "Col:=",  8421504)

oEditor.CreateCS Array("NAME:Contents", "Name:=",
"CS_15", "RelPos:=",
Array("t:=",  "pe", "from:=", "poly_1", "pos:=", 0,
"et:=", "pe", "prim:=", "poly_1",
"edge:=",  1))

oEditor.CreateCS Array("NAME:Contents", "Name:=",
"CS_196", "RelPos:=",
Array("t:=",  "pe", "from:=", "via_0", "pos:=", 0,
"et:=", "pse", "sel:=", "via_0",
"layer:=",  10, "sx:=", -0.0015, "sy:=", -0.0015, "ex:=",
0.0015, "ey:=", -0.0015,
"h:=", 0, "rad:=", 0))
```

ClearRelative (Layout Editor)

Use: Clear the relative positioning between a coordinate system (CS) and another object.

Command: None.

Syntax: `oEditor.ClearRelative Array("NAME:elements", <CS1>, <CS2>, …)`

Return Value: None.

Example:

```
oEditor.ClearRelative Array("NAME:elements", "CS_13")
```

Create3DStructure (Layout Editor)

Use: Place a set of primitives into a 3D structure coordinate system (i.e. a 3D via or cross-layer plate).

Command: None.

Syntax:
```
oEditor.Create3DStructure <3D structure name>,
Array("NAME:elements", <element1>, <element2>, …)
```

Return Value: None.

Example:
```
oEditor.Create3DStructure "S3D_11",
Array("NAME:elements", "circle_10", "circle_9")
```

Group (Layout Editor)

Use: Groups a set of primitives into a coordinate system.

Command: None.

Syntax:
```
oEditor.Group <CS name>, Array("NAME:elements",
<element1>, <element2>, …)
```

Return Value: None.

Example:
```
oEditor.Group "CS_7", Array("NAME:elements", "rect_4",
"rect_6")
```

CreateGroupSelected (Layout Editor)

Use: Groups a set of primitives into a subdesign.

Command: Layout > Group into Subdesign

Syntax:
```
oEditor.CreateGroupSelected Array("NAME:elements",
<selectable name>, …)
```

Return Value: None.

Example:
```
oEditor.CreateGroupSelected Array("NAME:elements", "1",
"2", "3",  "4", "5", "6")
```

PositionRelative (Layout Editor)

Use: Position a coordinate system (CS), including 3D structures and components, relative to another object.

Command: Draw > Position Relative

Syntax: There are two forms available:

PositionRelative
Array("NAME:Contents",
"RelPos:=", Array(<edge description>)),

Array("NAME:elements", <coordinate system name>, < coordinate system name >, …)

PositionRelative Array("NAME:Contents",
"RelPos:=", Array("t:=", "pr", "from:=", <edge port or pin name>)),
Array("NAME:elements", <coordinate system name>, <coordinate system name>, …)

Return Value: None.

Parameters:

Array("NAME:elements", "CS_23")

<edge description> for primitive edges

"t:=", "pe", "from:=", <"prim">, "pos:=", <edge position>, "et:=", "pe", "prim:=", <"prim">, "edge:=", <edge#>

<"prim">: text that is the primitive name

<edge position>
double between 0 and 1, inclusive
0 indicates the start of the edge
1 indicates the end of the edge
values in between are positions along the edge

<edge#>: integer that is the edge number on the primitive

<edge description> for via edges

"t:=", "pe", "from:=", <"via">, "pos:=", <edge position>, "et:=", "pse", "sel:=", <"via">, "layer:=", <layer id>, "sx:=", <start X location>, "sy:=", <start Y location>, "ex:=", <end X location>, "ey:=", <end Y location>, "h:=", <arc height>, "rad:=", <radians>

<"via">: text that is the name of the via to use
<layer id>: an integer that is the id of the layer of the pad of the via to use

<start X location>, <start Y Location>:
doubles that are the X, Y location of the start point of the edge arc
<end X location>, <end Y Location>:
doubles that are the X, Y location of the end point of the edge arc

<arc height>: double giving the height of the edge arc (0 for a straight edge)
<radians>: double giving the arc size in radians (0 for a straight edge)

< coordinate system name >
Text that contains the name of the coordinate system

<edge port or pin name>
Text that contains the name of the edge port or via

Example:

```
oEditor.PositionRelative Array("NAME:Contents", "Rel-
Pos:=", Array("t:=", "pe",
"from:=", "poly_0", "pos:=", 0, "et:=", "pe", "prim:=",
"poly_0", "edge:=", 3)),
Array("NAME:elements", "CS_23")

oEditor.PositionRelative Array("NAME:Contents", "Rel-
Pos:=", Array("t:=", "pe",
"from:=", "via_0", "pos:=", 0, "et:=", "pse", "sel:=",
"via_0", "layer:=", 10, "sx:=",
0.0015, "sy:=", 0.0015, "ex:=", -0.0015, "ey:=", 0.0015,
"h:=", 0, "rad:=", 0)),
Array("NAME:elements", "CS_201")
```

GetCSObjects (Layout Editor)

Use: Given a coordinate system name, returns a list of the objects inside. Coordinate system names can be found using FindObjects('Type', 'CS'). See Find Objects.

Syntax: `oLayout.GetCSObjects(<CS_name>)`

Return Value: List of objects in the coordinate system.

Example:

```
CS_objects = oLayout.GetCSObjects("CS_1")
```

SetCS (Layout Editor)

Use: Activate a coordinate system (CS).

Command: None.

Syntax:
```
oEditor.SetCS <CS name or blank>
```

Return Value: None.

Example:
```
oEditor.SetCS "CS_7"  // activate CS_7
oEditor.SetCS "" // activate the 'global CS', i.e. no CS is active
```

Ungroup (Layout Editor)

Use: Reverses a group operation; deletes a coordinate system but leaves the primitives.

Command: None.

Syntax:
```
oEditor.Ungroup Array("NAME:elements", <CS1>, <CS2>, …)
```

Return Value: None.

Example:
```
oEditor.Ungroup Array("NAME:elements", "CS_7")
```

NetClass Methods

The following net class primitives are available.

CreateNetClass

ModifyNetClass

DelNetClass

GetNetClassNets

GetNetClasses

CreateNetClass

Use: Create a new net class into a layout.

Command: CreateNetClass.

Syntax:
```
CreateNetClass (<name> <description> <nets name list>)
```

Return Value: None

Example:
```
oEditor.CreateNetClass "power", "power nets",
Array("A_D2D_VDD_0", "A_D2D_VDD_1",   "A_D2D_VSS_0")
```

ModifyNetClass

Use: Modify an existing net class in a layout.

Command: ModifyNetClass

Syntax:
```
ModifyNetClass (<name> <new name> <new description> <new nets name list>)
```

Return Value: None

Example:
```
oEditor.ModifyNetClass "power", "power", "new power nets", Array("A_D2D_VDD_0", "A_D2D_VDD_1", "A_D2D_VSS_0", "VDD_DIG_0", "VDD_DIG_1")
```

DelNetClass

Use: Delete a net class in a layout.

Command: DelNetClass

Syntax:
```
DelNetClass (<names>)
```

Return Value: None

Example:
```
oEditor.DelNetClass Array ("power")
```

GetNetClassNets

Use: Gets the list of nets in a net class.

Command: GetNetClassNets

Syntax:
```
GetNetClassNets (<name>)
```

Return Value: None

Example:
```
oEditor. GetNetClassNets Array ("power")
```

GetNetClasses

Use: Gets all the net classes in a layout.

Command: GetNetClasses

Syntax:
```
GetNetClasses ()
```

Return Value: None

Example:
```
oEditor. GetNetClasses
```

Miscellaneous Methods

The following miscellaneous methods are available.

AddLayer (Layout Editor)

Use: Adds a layer.

Command: Add Layer in a layout or footprint definition.

Syntax:
```
AddLayer Array ("NAME:layer",
"Name:=", <LayerName>,
"Type:=", <Type>,
"Top Bottom:=" , <TB>,
"Color:=", <ColorNumber>,
"Pattern:=", <FillPattern>,
"Visible:=", <Visibility>,
"Selectable:=", <Selectability>,
"Locked:=", <Locked>)
```

Return Value: None.

Parameters:

<LayerName>
Type: <String>
<Type >
Type: <String> (Same choices as in the layer dialog.)
<TB >
Type: <String> Choices are "Top"|"Neither"|"Bottom"|"Template"|"Invalid"
<ColorNumber>
Type: integer representing rbg in hex
<FillPattern>
Type: integer
<Visibility>
true | false

<Selectability>

true | false

<Locked>

true | false

Example:

```
oDefinitionEditor.AddLayer Array("NAME:layer", "Name:=",
"junk footprint", _
 "Type:=", "soldermask", "Top Bottom:=", "neither",
"Color:=", 4144959, _
"Pattern:=",  1, "Visible:=", true, "Selectable:=", true,
"Locked:=", false)
```

Note As with other Layout scripting interface commands that modifiy the layout, this command is not intended for use within scripts that define footprints. The command behavior from within such a script is undefined and may be unexpected. Use the LayoutHost scripting interface commands within scripts that define footprints.

AddStackupLayer (Layout Editor)

Use: Adds a stackup layer.

Command: Add Stackup Layer in a layout or footprint definition.

Syntax: **AddStackupLayer** Array ("NAME:layer",

"Name:=", <LayerName>,

"Type:=", <Type>,

"Top Bottom:=" , <TB>,

"Color:=", <ColorNumber>,

"Pattern:=", <FillPattern>,

"Visible:=", <Visibility>,

"Selectable:=", <Selectability>,

"Locked:=", <Locked>

"ElevationEditMode:=", <Elevation>, <SublayerArray>)

Return Value: None.

Parameters: <LayerName>

Type: <String>

<Type >

Type: <String> (Same choices as in the layer dialog.)

<TB >

Type: <String> Choices are "Top"|"Neither"|"Bottom"|"Template"|"Invalid"

<ColorNumber>

Type: integer representing rbg in hex

<FillPattern>

Type: integer

<Visibility>

true | false

<Selectability>

true | false

<Locked>

true | false

<Elevation>

Type: <String> Choices are "snap to middle" | "snap to top" | "snap to bottom" | "none"

<SublayerArray>

Type: Array(("NAME:Sublayer", "Thickness:=", <Thickness>, "LowerElevation:=", <Elevation>, "Roughness:=" , <Roughness>, "Material:=", <MaterialName)

<Thickness>

Type: <String> containing number and units (e.g. "0mil")

<Elevation>

Type: <String> containing number and units (e.g. "0mil")

<Roughness>

Type: <String> containing number and units (e.g. "0mil")

<Material>

Type: <String>

Example:

```
oEditor.AddStackupLayer Array("NAME:stackup layer",
"Name:=", "MyLayer2", _
  "Type:=", "Signal", "Top Bottom:=", "neither",
"Color:=", 127, "Pattern:=", 7, _
"Visible:=", true, "Selectable:=", true, "Locked:=",
false, "ElevationEditMode:=", "none",_
Array("NAME:Sublayer", "Thickness:=", "25mil", "LowerEle-
vation:=", "0mil", _
"Roughness:=", "0mil", "Material:=", "Al2_O3_ceramic"),
"UseR:=", true, _
 "RMdl:=", "Huray", "NR:=", "2mil", "HRatio:=", 2.8)
```

AlignObjects (Layout Editor)

Use: Align objects, e.g. primitives, relative to each other.

Command: AlignObjects

Syntax:
```
AlignObjects
Array("NAME:align", <"alignment">, …),
Array("NAME:elements", <"elem">, …)
```

Return Value: Return Value:None

Parameters:

`<"alignment">`

Type: text

Description: any one of:

"AlignLeft"

"AlignRight"

"CenterHorz" - align centers horizontally

"DistributeCentersHorz" - distribute object centers evenly horizontally

"SpaceEdgesHorz" - space objects equal distances apart horizontally

"AlignTop"

"AlignBottom"

"CenterVert" - align centers vertically

"DistributeCentersVert" - distribute object centers evenly vertically

"SpaceEdgesVert" - space objects equal distances apart vertically

"elem">

Type: text

Description: name of element (primitive, component, etc.) to align

Example:

```
Set oDesign = oProject.SetActiveDesign("PlanarEM1")
Set oEditor = oDesign.SetActiveEditor("Layout")
oEditor.AlignObjects Array("NAME:align", "AlignLeft"),
Array("NAME:elements", "circle_2",  "rect_1", "poly_3")
```

AlignPorts (Layout Editor)

Use: Causes automatic alignment of the microwave ports of the components, EdgePorts, and Pins referenced in the argument. In case the list is empty, aligns ALL the microwave ports in the layout.

Command: None.

Syntax:
```
AlignPorts Array ("NAME: elements",
        <object_name>,  // 1st object
        <object_name>,  // 2nd object, if any
        …)          // etc
```

Return Value: None.

Parameters:
```
The LayoutComp's ID
```

Example:
```
oEditor.Paste Array ("NAME:elements", "1", "2")
                        oEditor.Paste Array
("NAME:elements")
```

ChangeLayers (Layout Editor)

Use: Causes changing of the layers.

Command: None.

Syntax:
```
ChangeLayers
  Array("NAME:layers",
  <full_layer_description>,  // 1st layer
     <full_layer_description>,  // 2nd layer
     …) // etc
```

Return Value: None.

Parameters:
```
The LayoutComp's ID
oEditor.ChangeLayers Array("NAME:layers", "Mode:=", "Lam-
inate", Array("NAME:stackup layer", "Name:=",  _
  "Top", "ID:=", 7, "Type:=", "signal", "Top Bottom:=",
"neither", "Color:=",  _
  32512, "Transparency:=", 95, "Pattern:=", 1, "Vis-
Flag:=", 31, "Locked:=", false, "DrawOverride:=", 0,
"ElevationEditMode:=",  _
  "none", Array("NAME:Sublayer", "Thickness:=", "0mil",
"LowerElevation:=",  _
  "124.992125984252mil", "Roughness:=", "0mil", "Mate-
rial:=", "copper", "FillMaterial:=",  _
  "FR4_epoxy"), "Usp:=", true, Array("NAME:Sp", "Sn:=",
"HFSS", "Sv:=", "so(si=1)"), Array("NAME:Sp", "Sn:=",  _
  "PlanarEM", "Sv:=", "so(ifg=1, vly=1)"), "UseEtch:=",
true, "UseR:=", true), Array("NAME:stackup layer",
"Name:=",  _
```

```
  "Dielectric", "ID:=", 0, "Type:=", "dielectric", "Top
Bottom:=", "neither", "Color:=",  _
  127, "Pattern:=", 1, "VisFlag:=", 31, "Locked:=",
false, "DrawOverride:=", 0, "ElevationEditMode:=",  _
  "none", Array("NAME:Sublayer", "Thickness:=", "62mil",
"LowerElevation:=",  _
  "62.992125984252mil", "Roughness:=", "0mil", "Mate-
rial:=", "FR4", "FillMaterial:=",  _
  "FR4_epoxy")), Array("NAME:stackup layer", "Name:=",
"Ground", "ID:=", 6, "Type:=",  _
  "ground", "Top Bottom:=", "bottom", "Color:=", 4144959,
"Pattern:=", 1, "VisFlag:=",  _
  31, "Locked:=", false, "DrawOverride:=", 0, "Eleva-
tionEditMode:=", "none", Array("NAME:Sublayer", "Thick-
ness:=",  _
  "0mil", "LowerElevation:=", "62.992125984252mil",
"Roughness:=", "0mil", "Material:=",  _
  "copper", "FillMaterial:=", "FR4_epoxy"), "Neg:=",
true, "UseR:=", true), Array("NAME:stackup layer",
"Name:=",  _
  "Dielectric0", "ID:=", 9, "Type:=", "dielectric", "Top
Bottom:=", "neither", "Color:=",  _
  8421376, "Pattern:=", 1, "VisFlag:=", 31, "Locked:=",
false, "DrawOverride:=",  _
  0, "ElevationEditMode:=", "none", Array("NAME:Sub-
layer", "Thickness:=", "1.6mm", "LowerElevation:=",  _
  "0", "Roughness:=", "0", "Material:=", "FR4")),
Array("NAME:stackup layer", "Name:=",  _
  "Signal", "ID:=", 10, "Type:=", "signal", "Top Bot-
tom:=", "neither", "Color:=",  _
  16512, "Pattern:=", 1, "VisFlag:=", 31, "Locked:=",
false, "DrawOverride:=", 0, "ElevationEditMode:=",  _
  "none", Array("NAME:Sublayer", "Thickness:=", "0mm",
"LowerElevation:=", "0", "Roughness:=",  _
  "0", "Material:=", "copper", "FillMaterial:=", "FR4_ep-
oxy")), Array("NAME:layer", "Name:=",  _
  "Measures", "ID:=", 8, "Type:=", "measures", "Top Bot-
tom:=", "neither", "Color:=",  _
```

```
  4144959, "Transparency:=", 0, "Pattern:=", 1, "Vis-
Flag:=", 31, "Locked:=", false, "DrawOverride:=",  _
  0), Array("NAME:layer", "Name:=", "Rats", "ID:=", 3,
"Type:=", "rat", "Top Bottom:=",  _
  "neither", "Color:=", 16711680, "Pattern:=", 1, "Vis-
Flag:=", 0, "Locked:=",  _
  false, "DrawOverride:=", 0), Array("NAME:layer",
"Name:=", "Errors", "ID:=", 4, "Type:=",  _
  "error", "Top Bottom:=", "neither", "Color:=", 255,
"Transparency:=", 0, "Pattern:=", 1, "VisFlag:=",  _
  31, "Locked:=", true, "DrawOverride:=", 0),
Array("NAME:layer", "Name:=", "Symbols", "ID:=",  _
  5, "Type:=", "symbol", "Top Bottom:=", "neither",
"Color:=", 8323199, "Pattern:=",  _
  1, "VisFlag:=", 31, "Locked:=", false, "DrawOver-
ride:=", 0), Array("NAME:layer", "Name:=",  _
  "Assembly Top", "ID:=", 2, "Type:=", "assembly", "Top
Bottom:=", "top", "Color:=",  _
  16711680, "Pattern:=", 1, "VisFlag:=", 31, "Locked:=",
false, "DrawOverride:=",  _
  0), Array("NAME:layer", "Name:=", "Silkscreen Top",
"ID:=", 1, "Type:=", "silkscreen", "Top Bottom:=",  _
  "top", "Color:=", 65280, "Pattern:=", 1, "VisFlag:=",
31, "Locked:=", false, "DrawOverride:=",  _
  0))
```

ChangeOptions (Layout Editor)

Use: Changes options for an existing layout. (Does not change global options specified in the registry.) Only those options being changed need to be specified. Options not specified are not affected.

Command: None.

Syntax: `ChangeOptions Array("NAME:options",<OptionData>,…)`

Return Value: None

Parameters: `<OptionData>` can be of varying forms:

Type: <String>

Type: integer

Type: Array(float, float, float, float)

Example:

```
oEditor.ChangeOptions Array("NAME:options", _
```

```
"MajorSize:=", "20", _
   "MinorSize:=", "1", "MajorColor:=",  8421376, _
"MinorColor:=", 16776960,  _
   "ShowGrid:=", false, "PageExtent:=", _
Array(  -0.3, -0.1, 0.1, 0.1),   _
   "background color:=", 4194368, _
"DefaultToSketchMode:=",  true, _
   "fillMode:=", false, "PixelSnapTolerance:=", 22, _
"SnapTargetVertex_on:=",  _
   false, "SnapTargetEdgeCenter_on:=", false, _
"SnapTargetObjCenter_on:=", _
   true, "SnapTargetEdge_on:=", true, _
"SnapTargetElecConnection_on:=", true, _
   "SnapTargetIntersection_on:=",  true, _
"SnapTargetGrid_on:=", false, _
   "SnapSourceVertex_on:=", false, _
"SnapSourceEdgeCenter_on:=",  false,  _
   "SnapSourceObjCenter_on:=", true, _
"SnapSourceEdge_on:=", true, _
   "SnapSourceElecConnection_on:=",  true, _
"ConstrainToGrid:=", false _
"defaultholesize:=", "5mil",  _
   "show connection points:=", true, _
"display vertex labels:=", true, _
"NetColor:=", 8421440,  _
   "rectangle description:=", 1, "snaptoport:=", false, _
"sym footprint scaling:=", _
   0.385, "primary selection color:=",  32768, _
"secondary selection color:=", _
   22784, "preview selection:=", true, "anglesnap:=", _
"59deg", "AllowDragOnFirstClick:=", true, _
"useFixedDrawingResolution:=", true, _
   "DrawingResolution:=",  "0.002mm", "Tol:=", _
Array(3E-009, 1.5E-008, 1E-012))
```

Note: An error will be returned if this script command is not used at the top-level hierarchy.

- Global layout defaults are stored in the registry (Layout\Preferences\)
- Names of registry items were chosen for backwards compatibility and are consistent with adsn, technology file, and scripting naming
- Local options are both script-able and undo-able
- Global options are neither script-able nor undo-able

Option	Description	Installed default	Registry, Scripting Names, Data Type
Arc Drawing Resolution	Controls drawing of segments for arcs - either dynamic and based on current zoom or fixed to specified value	Dynamic	"useFixedDrawingResolution"[1] "DrawingResolution" [2]
Major and Minor Grid lines	Spacing, color, and visibility	The grid is displayed. Spacing based on current default length units. Major grid lines are 10 minor grid lines apart. The line colors are light blue grays, with major grid lines being darker.	"MajorColor" [3] "MinorColor" [3] "MajorSize" [2] "MinorSize" [2] "ShowGrid" [1]
Drawing Extent	Specifies size and coordinates of the layout	Lower left of the layout is (-0.1, -0.1) and the upper right is (0.1, 0.1)	"PageExtent" [4]
Background color	Color of the layout background	white	"background color" [3]

Sketch Mode	Fill patterns and center lines are not drawn.	off	"DefaultToSketchMode" [1]
Solid Mode	Objects are filled in with solid color.	off	"fillMode" [1]
Draw Connection Points	Display pin symbols in the layout.	off	"show connection points" [1]
Draw Rats	Display lines indicating missing physical connections in nets	on	"draw rats" [1]
Label Vertices	Display property text labeling the vertices of selected polygons and lines.	off	"display vertex labels" [1]
Net Color	Color used for highlighting nets	light yellow	"NetColor" [3]
Rectangle Description	Specifies if rectangles are described with two points or center point, width, and height	2 points	"rectangle description" [5]
Default Hole Size	Default size for actual holes in a PC board	25 mil	"defaultholesize" [2]
Align Microwave Components	Snaps components on entry.	on	"snaptoport" [1]
Symbol Footprint Scaling	Used to size symbol footprints placed in layout.	Based on data extent and current length units.	"sym footprint scaling" [6]

Primary and Secondary Selection Colors	Colors used to indicate the first item selected and other items currently selected.	Primary color is red, secondary color is a darker red	"primary selectioncolor" [3] "secondary selection color" [3]
Preview Selection	Highlight the object that would be selected with a left mouse button click in the current location.	off	"preview selection" [1]
Snapping options	Choose snapping sources and targets and the maximum distance (in pixels) for snapping to occur. Constrain edits to grid if desired. **Note**: Off-grid objects are ignored if ConstrainToGrid is asserted.	Snap distance is 20 pixels. Snapping sources are vertex, edge center, object center, & elec. conn. Snapping targets are vertex, edge center, object center, elec. conn, & grid. Edits are constrained to grid.	"PixelSnapTolerance"[7] "SnapTargetVertex_on"[1] "SnapTargetEdgeCenter_on"[1] "SnapTargetObjCenter_on"[1] "SnapTargetEdge_on"[1] "SnapTargetElecConnection_on"[1] "SnapTargetIntersection_on"[1] "SnapTargetGrid_on"[1] "SnapSourceVertex_on"[1] "SnapSourceEdgeCenter_on"[1] "SnapSourceObjCenter_on"[1] "SnapSourceEdge_on"[1] "SnapSourceElecConnection_on"[1] "ConstrainToGrid"[1]

Always Show Merge Layers Dialog	Controls display of the layer merging dialog.	off – Only show dialog when the user must be involved.	"AlwaysShowLayerMergeDlg" [1]
Rotation Increment	During rotation, objects are rotated by multiples of this amount.	Based on current angle units.	"anglesnap" [2]
Allow Drag on first click	Selection and move with one left mouse button click.	false	"AllowDragOnFirstClick" [1]

The footnotes in the table above correspond to the following:

[1] = Integer with a value of 1 for on/true and 0 for off/false

[2] = String containing a amount and units

[3] = Integer representing a Color

[4] = An Array containing (lower left x, lower left y, upper right x, upper right y)

[5] = Integer with a value of 0 for two points, and 1 for center/width/height

[6] = String holding a double.

[7] = Integer

ClearLayerMappings (Layout Editor)

Use: Clear layer mappings.

Command: None.

Syntax: `ClearLayerMappings <component name>`

Return Value: None.

Parameters: `<component name> is a string that specifies the name of the component.`

Example:
```
oEditor.ClearLayerMappings "2"
```

ClearRefPort (Layout Editor)

Use: Clear references (or referencing) ports from each of the named ports.

Command: Draw > Clear reference Port

Syntax: **`ClearRefPort`** `Array("Port1", …)`

Return Value: None.

Example: `oEditor.ClearRefPort Array("Port1", …)`

CopyToPlanarEM (Layout Editor)

Use: Generate a single, flat, PlanarEM design from a selection of components and/or sub-circuits.

Command: CopyToPlanarEM

Syntax: `CopyToPlanarEM Array("NAME:elements", <"obj1">, <"obj2">, …)`

Return Value: None

Parameters: `<"obj1">`

Type: text

// component or sub-circuit instance name.

Example:

```
Example:
Set oDesign = oProject.SetActiveDesign("Nexxim1")
Set oEditor = oDesign.SetActiveEditor("Layout")
oEditor.CopyToPlanarEM Array("NAME:elements", "1", "2",
"3")
```

CreatePortInstancePorts (Layout Editor)

Use: Create port on port instances.

Command: Right-Click-Menu > Port > Create

Syntax: `CreatePortInstancePortsArray("NAME:elements", "element-name",…)`

Return Value: None

Parameters: `<element-name>`

Type: string

// Name of a port instance on which ports will be created.

Example:

```
Example:
oEditor.CreatePortInstancePortsArray("NAME:elements",
"0:106")
```

CreatePortsOnComponents (Layout Editor)

Use: Create port on port instances of selected components.

Command: Right-Click-Menu > Port > Create Ports on Component

Syntax: `CreatePortsOnComponentsArray("NAME:elements", "element-name",…)`

Return Value: None

Parameters:
```
<element-name>
```
Type: string

// Name of a component.

// Ports are created on the unconnected port instances of the selected components.

Example:
```
Example:
oEditor.CreatePortsOnComponents Array("NAME:elements",
"0")
```

CutOutSubDesign (Layout Editor)

Use: Cut out a subdesign.

Command: CutOutSubDesign

Syntax:
```
oEditor.CutOutSubDesign Array(
"NAME:Params",
"Name:=", <"name">,
"EMDesign:=", <boolean>,
"SubDesign:=", <boolean>,
"Within:=", <boolean>,
"Without:=", <boolean>,
“AutoGenExtent”:=<boolean>,
“Expansion”:=<double>,
“RoundCorner”:=<boolean>,
“Increments”:=<integer>,
"ExtentSel:=", Array(<"extent-poly">, …),
Array("NAME:Nets", "net:=", Array(<"net-name">, <clip>), …))

In place of "ExtentSel:=", Array(<"extent-poly">, …) an explicit polygon can be used:
  "Extent:=", Array("cl:=", true,
    "pt:=", Array(U:=", <"units">, "x:=", <double>, "y:="  <double>, …))
```

Parameters:
```
Name — name of the cutout design.
```
EMDesign — if true, create a EM design, otherwise create a Nexxim design.

Within — boolean; cut out the interior region.

Without — boolean; cut out the exterior region.

AutoGenExtent — boolean; when true, Designer disregards both ExtentSel and

Extent. Designer will instead generate a polygonal shape around all nets which are not clipped.

Expansion — double; similar to the Auto Generate Extent Dialogue. However, this is always a unitless fraction. By default it is set to .1.

RoundCorner — boolean; identical to the Auto Generate Dialogue. By default this is set to true. In general, rounded corners are preferred when the cutout shape has acute or nea-acute angles

Increments — integer; this is from the Auto Generate Dialogue, and by default is set to true. This can greatly increase the running time and a small increase can make a big difference. It is probably best to experiment with this parameter first, rather than set it arbitrarily.

ExtentSel— an array of extent polygon names.

Extent — alternative to ExtentSel; an explicit polygon defined by coordinates.

cl — must always be true; indicates that the polygon is closed.

U — coordinate units, e.g. "mm"

pt — array of coordinate values.

x — x coordinate value

y — y coordinate value

Nets — the net information.

net — array of net information.

<”net-name”>

"<no net trace>" — special value referring to traces not in a net.

"<no net fill>" — special value refereeing to fill polygons not in a net.

<design>:<net> — net within a particular design.

<net> — net within the active design.

<clip> — boolean true/false; if true, the net is clipped against the extent else the net is included but not clipped.

Return Value: None

Example: Example using "extent_poly" as the selection extent:

```
oEditor.CutOutSubDesign Array("NAME:Params",
"Name:=", "EMDesign1_cutout",
"EMDesign:=",  true,
"SubDesign:=", false,
"Within:=", true,
"Without:=", false,
"ExtentSel:=", Array("extent_poly", …),
Array("NAME:Nets",
```

```
"net:=", Array("<no net trace>", true),
"net:=", Array("<no net fill>", true),
"net:=", Array("EMDesign1:GND", true) … ))
```

Example: Example using an explicit polygon as the selection extent:

```
oEditor.CutOutSubDesign Array("NAME:Params",
"Name:=", "EMDesign1_cutout",
"EMDesign:=", true,
"SubDesign:=", false,
"Within:=", true,
"Without:=", false,
"Extent:=", Array(
"cl:=", true,
"pt:=", Array(
"U:=", "mm",
"x:=", 0,
"y:="  0, ) ),
Array("NAME:Nets",
"net:=", Array("<no net trace>", true),
"net:=", Array("<no net fill>", true),
"net:=", Array("EMDesign1:GND", true) … ))
```

Example: Example that will create a cutout around first the pos and then the neg trace from the Sample Project "Diff_Via" under Open Samples/EM/SI:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsysDesigner.Design-
erScript")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Diff_Via")
Set oDesign = oProject.SetActiveDesign("diffViaNominal")
```

```
Set oEditor = oDesign.SetActiveEditor("Layout")
oEditor.CutOutSubDesign Array("NAME:Params", "Name:=",
"diffViaNominal_pos", "EMDesign:=",  _
  true, "SubDesign:=", false, "Within:=", true, "With-
out:=", false, "AutoGenExtent:=",  _
  true, "Expansion:=", 0.1, "RoundCorners:=", false,
"Increments:=", 1, "ExtentSel:=", Array(),
Array("NAME:Nets", "net:=", Array( _
  "diffViaNominal:neg", true), "net:=", Array("diffVi-
aNominal:pos", false)))
oEditor.CutOutSubDesign Array("NAME:Params", "Name:=",
"diffViaNominal_neg", "EMDesign:=",  _
  true, "SubDesign:=", false, "Within:=", true, "With-
out:=", false, "AutoGenExtent:=",  _
  true, "Expansion:=", 0.1, "RoundCorners:=", false,
"Increments:=", 1, "ExtentSel:=", Array(),
Array("NAME:Nets", "net:=", Array( _
  "diffViaNominal:neg", false), "net:=", Array("diffVi-
aNominal:pos", true)))
```

Note that in this example, each sub design should have its own name, otherwise the results are not well defined. Also note that the "false" after the net indicates that it will be used to build the extent outline. "True" means that the net will be included but will be trimmed.

DefeatureObjects (Layout Editor)

Use: Remove irrelevant features from a primitive. Vertices that deviate less than the tolerance from a chord are removed; narrow concave regions less than the tolerance wide are also eliminated. The command can also be used to fix self-intersections (only the 'positive' areas are kept).

Command: DefeatureObjects

Syntax: `DefeatureObjects Array("NAME:options", "tolerance:=", <value>, "fix:=", <fix>), Array("NAME:elements", <"primitive">, …)`

Return Value: None

Parameters: `<"value">`

Type: double, e.g. 0.001

Description: tolerance value

<fix>

Type: boolean (true or false)

Description: if true, then self-intersections are fixed.

< primitive >

Type: text

Description: primitive name.

Example:

```
Set oDesign = oProject.SetActiveDesign("PlanarEM1")
Set oEditor = oDesign.SetActiveEditor("Layout")
oEditor.DefeatureObjects Array("NAME:options", "toler-
ance:=", 1E-006, "fix:=", true), Array("NAME:elements",
"poly_3", "poly_4", "poly_5")
```

Duplicate (Layout Editor)

Use: Duplicates layout objects.

Command: The specified objects are duplicated by the given count with the specified offset.

Syntax: `Duplicate Array("NAME:options", "count:=", <count data>), Array("NAME:elements", <element names>), Array( <x>, <y>)`

Return Value: None

Parameters: `<count data> is the number of sets of new objects to be generated (and does not include the original objects)`

<element names> is one or more strings with the names of layout objects

<x> is the x value for the offset

<y> is the y value for the offset

Example:

```
oEditor.Duplicate Array("NAME:options", "count:=", 2),
Array("NAME:elements", "rect_56"), Array( -0.018, 0.017)
```

DuplicateAcrossLyrs (Layout Editor)

Use: Duplicate selected objects (layout and footprint) to other layers.

Command: Draw > Duplicate > Across Layers

Syntax: `DuplicateAcrossLyrs Array("NAME:elements", "element-name",…), Array("NAME:layers", "layer-name",…)`

Return Value: None

Parameters: `<element-name> // The name of the element to be duplicated.`

<*layer-name*> // The name of the layer to duplicate elements to.

Example:

```
oEditor.DuplicateAcrossLyrs Array("NAME:elements",
"poly_550"), Array("NAME:layers", "Top")
```

EraseMeasurements (Layout Editor)

Use: Causes erasing of ALL measurements currently present.

Command: None.

Syntax:

```
EraseMeasurements
```

Return Value: None.

Parameters:

```
None.
```

Example:

```
oEditor.EraseMeasurements
```

ExportDXF (Layout Editor)

Use: Export DXF

Command: File > Export > AutoCAD

Syntax:

```
ExportDXF
Array("NAME:options", "FileName:=", <"filename">, "ScaleFactor:=", <scale>),
Array("NAME:layers", <"layer">, …)
```

Return Value: None

Parameters:

```
<"filename">
```

Type: text

Description: file path for the export file

<scale>

Type: double

Description: the scaling of the values written out, e.g. if the data is written out in "mm", the scaling factor will be 1000.

<"layer">

Type: text

Description: names of the layers to be exported.

Example:

```
Set oDesign = oProject.SetActiveDesign("PlanarEM1")
Set oEditor = oDesign.SetActiveEditor("Layout")
oEditor.ExportDXF Array("NAME:options", "FileName:=",
"C:/Projects/Temp/output.dxf", "ScaleFactor:=", 1000),
Array("NAME:layers", "Assembly Bottom", "Assembly Top",
"Bottom", "Bottom Dielectric", "Errors", "Ground", "Mea-
```

```
sures", "Rats", "Silkscreen Bottom", "Silkscreen Top",
"Symbols", "Top", "Top Dielectric")
```

ExportGDSII (Layout Editor)

Use: Export GDSII

Command: File > Export > GDSII

Syntax:

```
ExportGDSII
Array("NAME:options",
"FileName:=", <"file name">,
"NumVertices:=", <num vertices>,
"ArcTol:=", <tolerance>,
"LayerMap:=", Array(
"entry:=", Array(
"layer:=", <"layer">,
"id:=", <id>,
"include:=", <include>), …))
```

Return Value: None

Parameters:

<"file name">

Type: text

Description: GDSII export file name.

<num vertices>

Type: integer

Description: Maximum number of vertices allowed in a polygon (specify zero if unlimited).

<tolerance>

Type: double

Description: Arc tolerance (in meters); the value used to discretize arcs. Smaller the value the greater the number of edges exported. For 0.1mm use 0.0001.

<"layer">

Type: text

Description: The name of a layer to export.

<id>

Type: integer

Description: The GDSII layer ID to assign the exported layer.

<include>

Type: boolean

Description: if true, the layer is exported (otherwise it is not).

Example:

```
Set oDesign = oProject.SetActiveDesign("PlanarEM2")
Set oEditor = oDesign.SetActiveEditor("Layout")
oEditor.ExportGDSII
Array("NAME:options",
"FileName:=", "C:/Projects/ design.gds",
"NumVertices:=", 0,
"ArcTol:=", 2E-006,
"LayerMap:=", Array(
"entry:=", Array(
"layer:=", "Bottom",
"id:=", 2,
"include:=", true),
"entry:=", Array(
"layer:=", "Top",
"id:=", 1,
"include:=", true)))
```

ExportGerber (Layout Editor)

Use: Export the current design to Gerber files.

Command: File > Export > Gerber

Syntax:

```
ExportGerber
Array("NAME:options",
"FileName:=", <"file name">,
"Units:=", <"units">,
"IntPlace:=", <integer places>,
"DecimalPlace:=", <decimal places>,
"Suppress:=", <"suppress">,
Array("NAME:GerberPages",
"<page>:=", Array(<"layer">, …), …))
```

Return Value: None

Parameters:

```
<"file name">
```

Type: text

Description: prefix for the exported Gerber files; the actual files will be named <file

name>_<page>.ger
<"units">
Type: text
Description: unit code, e.g. "in", "mm", etc.
<integer places>
Type: integer
Description: number of integer places to use when formatting the numerical output.
<decimal places>
Type: integer
Description: number of decimal places to use when formatting the numerical output.
<"suppress">
Type: text
Description: either "Leading Zeros", or "Trailing Zeros".
<page>
Type: integer
Description: Gerber page number
<"layer">
Type: text
Description: Designer layers to be exported to the specified Gerber page.

Example:

```
Set oDesign = oProject.SetActiveDesign("PlanarEM1")
Set oEditor = oDesign.SetActiveEditor("Layout")
oEditor.ExportGerber
Array("NAME:options",
"FileName:=", "C:/ Projects/design.ger",
"Units:=", "in",
"IntPlace:=", 5,
"DecimalPlace:=", 5,
"Suppress:=", "Leading Zeros",
Array("NAME:GerberPages", "1:=", Array("Ground"), "2:=",
Array("Top") ))
```

ExportNCDrill (Layout Editor)

Use: Export the vias to an NC Drill file

Command: File > Export > NCDrill

Syntax:

```
ExportNCDrill
Array("NAME:options",
"FileName:=",  <"file name">,
"Units:=", <"units">,
"IntPlace:=", <integer places>,
"DecimalPlace:=",  <decimal places>,
"SuppressLeadingZero:=", <suppress leading zeros>,
Array("NAME:NCDFileOptsVec",
Array("NAME:NCDrillOptions",
"Top:=",  <"layer">,
"Bottom:=", <"layer">,
"Create:=", <create>), …))
```

Return Value: None

Parameters:

`<"file name">`

`Type: text`

Description: prefix for the exported Gerber files; the actual files will be named <file name>_<page>.ger

`<"units">`

Type: text

Description: unit code, e.g. "in", "mm", etc.

`<integer places>`

Type: integer

Description: number of integer places to use when formatting the numerical output.

`<decimal places>`

Type: integer

Description: number of decimal places to use when formatting the numerical output.

`<suppress leading zeros>`

Type: boolean

Description: if true, then suppress leading zeros in the output file.

`<layer>`

Type: text

Description: Start/end layer names; each layer range is written to a separate NC drill file.

`<create>`

Type: boolean

Description: if true, create the corresponding NC drill file.

Example:

```
Set oDesign = oProject.SetActiveDesign("PlanarEM1")
Set oEditor = oDesign.SetActiveEditor("Layout")
oEditor.ExportNCDrill
Array("NAME:options",
"FileName:=",  "C:/ Projects/drill.ncd",
"Units:=", "mm",
"IntPlace:=", 5,
"DecimalPlace:=",  5,
"SuppressLeadingZero:=", true,
Array("NAME:NCDFileOptsVec",
Array("NAME:NCDrillOptions",
"Top:=",  "Top",
"Bottom:=", "Ground",
"Create:=", true),
Array("NAME:NCDrillOptions",
"Top:=",  "Top",
"Bottom:=", "Bottom",
"Create:=", true)))
```

GetAllLayerNames (Layout Editor)

Use: Informational.

Command: None.

Syntax: `GetAllLayerNames`

Return Value: Array of strings which are the names of all layers in the layout, blackbox, or footprint.

Example: `None.`

GetComponentInfo (Layout Editor)

Use: Informational.

Command: None.

Syntax: `GetComponentInfo<compID>`

Return Value: array of strings, as follows:

"ComponentName=*string*"

"PlacementLayer=*string*"
"LocationX=*number*"
"LocationY=*number*"
"BBoxLLx=*number*"
"BBoxLLy=*number*"
"BBoxURx=*number*"
"BBoxURy=*number*"
"Angle=*number" (in degrees)*
"Flip=true *or* false"
"Scale=*number*"

Parameters: The LayoutComp's ID

Example:
```
dim info
                                    sys= oEditor.GetComponentInfo
("1")
```

GetCompInstanceFromRefDes (Layout Editor)

Use: Informational.

Command: None.

Syntax: `GetCompInstanceFromRefDes (<object_name>)`

Return Value: Returns IDispatch for CompInstance.

Parameters:
```
<object_name>          // string is refDes
```

Example:
```
oEditor.GetCompInstanceFromRefDes("A1")
```

GetComponentPins (Layout Editor)

Use: Informational.

Command: None.

Syntax: `GetComponentPins<compID>`

Return Value: array of strings, which are the names of all the component's pins

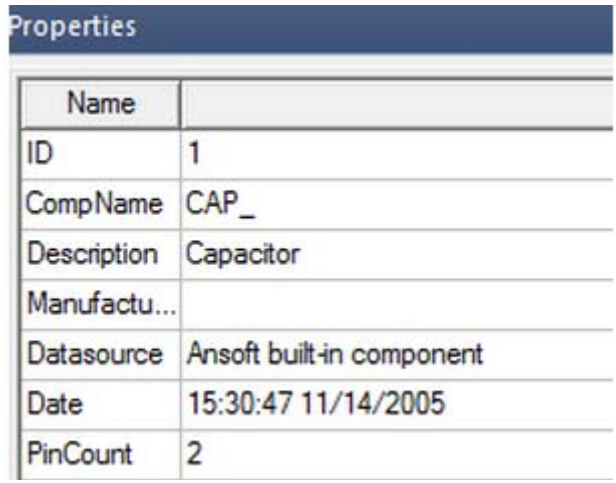
Parameters:
```
The LayoutComp's ID
CompInst@<ComponentName>;<CompInstID>
```

Example:
```
'----------------------------------------------------------------------------------------
'Script Recorded by ANSYS Electronics Desktop
'Component Name is CAP_ and ID is 1
'----------------------------------------------------------------------------------------
```

```
dim pins

pins = oEditor.GetComponentPins ("CompInst@CAP_;1")
```

Note: For the documented example, the capacitor component name is CAP_ and its ID is 1. If you can select the item of interest in the schematic, the **Properties** window gets updated with the corresponding component name, its ID, and other details as shown in the following figure.



GetComponentPinInfo (Layout Editor)

Use: Informational.

Command: None.

Syntax: `GetComponentPinInfo<compID, pinName>`

Return Value: retval = array of strings, as follows:

"X=val"

"Y=val"

"Angle=val"

"Flip=true/false"

"WireID=string"

Parameters: `compID = The LayoutComp's ID`

pinName = The name of the pin

Example:

```
dim info
                                info =
oEditor.GetComponentPinInfo ("1", "n1")
```

GetEditorName (Layout Editor)

Use: Informational.

Command: None.

Syntax: `GetEditorName()`

Return Value: Returns the name of the editor.

Example:
```
dim info
info = oEditor.GetEditorName
```

GetLayerInfo (Layout Editor)

Use: Informational.

Command: None.

Syntax: `GetLayerInfo<layer_name>`

Return Value: array of strings, as follows:

Type: typename
TopBottomAssociation: "Top"|"Neither"|"Bottom"|"Template"|"Invalid"
Color: integer [representing rgb in hex]
IsVisible: "true"|"false" [true if any type of object below is visible]
  IsVisibleShape: true [for stackup layer only]
  IsVisiblePath: true [for stackup layer only]
  IsVisiblePad: true [for stackup layer only]
  IsVisibleHole: true [for stackup layer only]
  IsVisibleComponent: true [for stackup layer only]
IsLocked: "true"|"false"
LayerId: integer [the ID for the layer]

The following are also in the array if the layer is a stackup layer:
Index: integer [the stackup order index]
LayerThickness: double [total layer thickness, in meters]
EtchFactor: double [won't show if the layer has no etch factor defined]
IsIgnored: "true"|"false"
NumberOfSublayers: 1 [always 1]
Material0: materialName
FillMaterial0: materialName
Thickness0: expression_with_units
LowerElevation0: expression_with_units

Roughness0Type: Groisse | Huray [won't show if the layer has no roughness defined]

Roughness0: expression_with_units | expression_with_units, double [Groisse roughness or Huray roughness]

Parameters: `The name of the layer`

GetMaterialList (Layout Editor)

Use: Get the names of all the materials for a layout.

Command: None.

Syntax: `GetMaterialList`

Return Value: Array of strings.

Parameters: `None.`

Example:

```
Dim materialNames
materialNames = oEditor.GetMaterialList
```

GetNetConnections (Layout Editor)

Use: Informational.

Command: None.

Syntax: `GetNetConnections(<netName>)`

Return Value: Array of strings containing the connections for the net identified by the netName argument. The strings in the array are in one of four formats:

"ComponentPin compID pinname x y EdgePort"

"ComponentPin compID pinname x y PinPadstack: padstackName"

"InterfacePort portname portID x y Padstack: padstackName"

"EdgePort portname portID EdgeInfo: Primitive id, edge index[Primitive id, edge index]"

where *compID* is the component instance identifier, *pinname* is the name of the connected pin, *x* and *y* are the connection point, *padstackName* is the name of the padstack, *portname* is the name of the port, *portID* is the identifier for the interface port, and *id* and *index* identify edges involved in an edgeport.

Parameters: `<netName>`

Type: String

The name of the net.

Example:

```
netArray = oEditor.GetNetConnections("net_1")
```

GetPortInfo (Layout Editor)

Use: Request information on a port or pin.

Command: None.

Syntax: `GetPortInfo<"name">`

Return Value: an array of text as follows:

For pins and ports that are not edge ports:

`Name=<name>`

Type=Pin, Padstack: <padstack definition name>

`X=<X coordinate>`

`Y=<Y coordinate>`

ConnectionPoints=<connection description>; <connection description>,...

`NetName=<net name>`

For edge ports:

`Name=<name>`

`Type=EdgePort`

`X=<X coordinate>`

`Y=<Y coordinate>`

ConnectionPoints=<connection description>; <connection description>,...

`NetName=<net name>`

`<name>`

Text containing the name of the pin or port

`<padstack definition name>`

Text containing the name of the associated padstack definition

`<X coordinate>`

Double indicating the X location of the pin or port location

`<Y coordinate>`

Double indicating the Y location of the pin or port location

`<connection description>`

```
< X coordinate> <Y coordinate> Dir:<direction> Layer:
<layer name>
```

```
<direction>
```

Either NONE or a double giving the angle in degrees for the connection direction

```
<layer name>
```

Name of the layer for the connection point being described.

```
Example returned values:

Name=Pin1
Type=Pin, Padstack: Padstack
X=0.000744
Y=0.015537
ConnectionPoints= 0.000744 0.015537 Dir:NONE Layer: Top;
0.000744 0.015537
Dir:NONE Layer: Ground; 0.000744 0.015537 Dir:NONE Layer:
Bottom
NetName=Pin1

Name=Port1
Type=EdgePort
X=-0.008120
Y=0.004264
ConnectionPoints= -0.008120 0.004264 Dir:90.000000 Layer:
Bottom
NetName=Port1
```

Parameters:

```
<"name">
```

Text that contains the name of the port or pin for which information is being requested.

Example:

```
Dim conns
conns = oEditor.GetPortInfo("Port1")
```

GetProperties (Layout Editor)

Use: Gets a list of all the properties belonging to a specific **PropServer** and **PropTab**. This can be executed by the **oProject**, **oDesign**, or **oEditor** objects.

Command: None

Syntax: `GetProperties( <PropTab>, <PropServer> )`

Return Value: Variant array of strings – the names of the properties belonging to the prop server.

Example:

```
Dim all_props
all_props = oDesign.GetProperties("BaseElementTab",_
"rect_1")
```

GetPropertyValue (Layout Editor)

Use: Gets the value of a single property. This can be executed by the **oProject**, **oDesign**, or **oEditor** objects.

Command: None

Syntax: `GetPropertyValue(<PropTab>, <PropServer>, <PropName>)`

Return Value: String representing the property value.

Example:

```
value_string = _
oEditor.GetPropertyValue("BaseElementTab",_
"rect_1", "Name")
```

GetSelections (Layout Editor)

Use: Informational.

Command: None.

Syntax: `GetSelections`

Return Value: array of IDs

Parameters: `None.`

Example:

```
dim sels
                                    sels = oEditor.GetSelections
```

GetStackupLayerNames (Layout Editor)

Use: Informational.

Command: None.

Syntax: `GetStackupLayerNames`

Return Value: array of strings which are the names of all layers in the layout, blackbox, or footprint.

Parameters: None.

HighlightNet (Layout Editor)

Use: Highlight (or un-highlight) all the elements in a net.

Command: HighlightNet

Syntax: `HighlightNet Array("NAME:Args", "Name:=", "<net-name>", "Hi:=", <flag>)`

Return Value: None

Parameters: `<" net-name ">`

Type: text

Description: name of the net

<flag>

Type: boolean (true or false)

Description: if true, the net is highlighted, else the highlighting is removed.

Example:

```
Set oDesign = oProject.SetActiveDesign("PlanarEM1")
Set oEditor = oDesign.SetActiveEditor("Layout")
oEditor.HighlightNet Array("NAME:Args", "Name:=",
"net_0", "Hi:=", false)
```

PageSetup (Layout Editor)

Use: Specifies page setup for printing.

Command: **File>Page Setup**

Syntax: `PageSetup <ArgArray>`

Return Value: None.

Parameters: `<Margins>`: Page margins in implicit units of inches.

<Border>: Integer value indicating to draw border (1) or not to draw (0).

<DesignVars>: Integer value indicating to draw design vars (1) or not to draw (0).

Example:

```
Set oProject = oDesktop.GetActiveProject()
Set oDesign = oProject.GetActiveDesign()
Set oEditor = oDesign.GetActiveEditor()
oEditor.PageSetup Array("NAME:PageSetupData", "mar-
gins:=", Array("left:=", 550, "right:=",  _
  550, "top:=", 500, "bottom:=", 500), "border:=", 1,
"DesignVars:=", 0)
```

RemoveLayer (Layout Editor)

Use: Removes a layer or stackup layer.

Command: Remove Layer from a layout or footprint definition

Syntax: `RemoveLayer (<LayerName>)`

Return Value: None.

Parameters: `<LayerName>`

Type: <String>

Example:
```
oEditor.RemoveLayer ("T3 C1 sub")
oDefinitionEditor.RemoveLayer("Top3 Footprint")
```

Note As with other Layout scripting interface commands that modifiy the layout, this command is not intended for use within scripts that define footprints. The command behavior from within such a script is undefined and may be unexpected. Use the LayoutHost scripting interface commands within scripts that define footprints.

RemovePortsOnComponents (Layout Editor)

Use: Remove ports on port instances of selected components.

Command: Right-Click-Menu > Port > Remove Ports From Component

Syntax: `RemovePortsOnComponentsArray("NAME:elements", "element-name",…)`

Return Value: None

Parameters: `<element-name>`

Type: string

// Name of a component.

// Ports are removed from the port instances of the selected components.

Example:
```
Example:
oEditor.RemovePortsOnComponentsArray("NAME:elements",
"0")
```

SelectAll (Layout Editor)

Use: `Select all elements in the layout editor.`

Command: None.

Syntax: `SelectAll()`

Parameters: `None`

Example:

```
Dim removedDefs
removedDefs = oDefinitionEditor.SelectAll()
```

SetLayerMapping (Layout Editor)

Use: Set layer mapping.

Command: None.

Syntax: `SetLayerMapping <component name>, <design layer>, <footprint layer>`

Return Value: None.

Parameters:

`<component name>` is a string that specifies the name of the component

<design layer> is a string that specifies the name of the design layer

<footprint layer> is a string that specifies the name of the footprint layer

Example:

```
oEditor.SetLayerMapping "2", "Bottom Signal", "Top"
```

SetPropertyValue (Layout Editor)

Use: Sets the value of one property. This is not supported for properties of the following types: **ButtonProp, PointProp,** and **VPointProp**. Only the **ChangeProperty** command can be used to modify these properties. This can be executed by the **oProject**, **oDesign**, or **oEditor** objects.

Command: None

Syntax: `SetPropertyValue <PropTab>, <PropServer>, <PropName>, <PropValue>`

Return Value: None

Parameters:

`<PropValue>`

Type: String

Contains the value to set the property. The formatting is different depending on what type of property is being edited. Use **GetPropertyValue** for the desired property to see the expected format.

Example:

```
oEditor.SetPropertyValue _
"BaseElementTab","rect_1",_
"LineWidth", "3mm"
```

StitchLines (Layout Editor)

Use: Stitch together connected or crossing lines into polygons and lines.

Command: StitchLines

Syntax: `StitchLines Array("NAME:elements", <"line">, …)`

Return Value: None

Parameters: `<"line">`

Type: text

Description: line name.

Example:

```
Set oDesign = oProject.SetActiveDesign("PlanarEM1")
Set oEditor = oDesign.SetActiveEditor("Layout")
oEditor.StitchLines Array("NAME:elements", "line_1",
"line_2", "line_3")
```

UnselectAll (Layout Editor)

Use: `Unselect all active selections in the layout editor.`

Command: Edit > Unselect All

Syntax: `UnselectAll()`

Parameters: `None`

Example: `oLayout.UnselectAll()`

ZoomToFit (Layout Editor)

Use: Set the current schematic zoom to fit the contents of the currently visible page

Command: None

Syntax: `ZoomToFit()`

Return Value: None

27 Schematic Scripting

The Schematic scripting interface is a set of commands that match the data changing methods available in the UI of the Schematic, plus some selection and query methods. Examples of the commands available through the scripting interface are adding items, removing items, and modifying items on the Schematic. Identifying objects on the Schematic is done by a unique ID that is generated and returned by all methods that add objects, and by the FindElements and GetSelections methods. This ID can then be passed into commands to modify or remove Schematic objects. Since most Schematic objects can have sub-components (such as property displays or segments on a wire, etc), IDs can also be in the format of "TopLevelID:SubComponentIndex". The SubComponentIndex is a one-based index into the sub components of the item thus making the second segment of component CAP1 identified by "CAP1:2". A SubComponentIndex of zero will refer to the TopLevel item only. When an ID is mentioned later in this document, it will refer to either a simple string ID representing a top-level Schematic item or a "TopLevelID:SubComponentIndex" pair unless otherwise stated. If a specified ID or SubComponentIndex does not exist, then the function will ignore that entry and proceed with other specified IDs. Even if no valid IDs are specified, the functions will still return success.

The topics for this section include:

Method Format

In the following formats, [Opt=x] appearing after a parameter indicates that the parameter is optional and the default value is x.

When calling object-creation methods as functions, parenthesis are required in order to retrieve the name of the created object. Parenthesis are also required in JavaScript. When using VB/VBScript to call a method as a subroutine, however, parenthesis are not required.

All methods to create Schematic objects have the following form:

```
Create[type](VARIANT parameters,
                  // Array of type specific parameters
              VARIANT attributes,
// attributes is in the format of:
       Array("NAME:Attributes", _
              "Page:=", int,      _ // [Opt=1] Page number
(one-based)
                                    //   Note: Page 1 always
exists
              "X:=", double,      _ // X position of the
object
              "Y:=", double,      _ // Y position of the
object
             "Angle:=", double, _ // Rotation angle (radi-
ans)
              "Flip:=", bool)       // True if mirrored
              [out,retval] string id)
```

The algorithm used in the create methods is:

1) Create SchAdd*[type]*Command object with parameters passed in

2) Execute the command

All methods to modify Schematic objects have the following form:

```
[modification](Array("NAME:Selections",
                     "Page:=", Page number [optional:
Default=1]
                    "Selections:=", IDs to modify see for-
mat below),
               VARIANT params
               // Array of modification specific parame-
ters)
```

The algorithms in these types of methods will be:

1) Go through ids and get the SelectableObjs they correspond to
2) Select the SelectableObjs found in (1)
3) Select any sub components specified
4) Create the Sch*[modification]*Command
5) Execute the command

ID Format

When IDs are passed to a function it can be in one of the following formats:

- Array of integers of top-level items only
- String of IDs separated by spaces or commas
- Array of strings with each string element being an ID

Point Format

When a method takes an array of points, each element in the array is a string in the format of: “x y”, “x,y”, or “(x,y)”.

Editor Scripting IDs

Objects in the schematic are identified by text IDs. These IDs are used in scripts to perform actions on objects. These IDs are returned by all methods that add objects, and by the FindElements and GetSelections methods. The IDs are then passed into commands to modify or remove Schematic objects.

Format of IDs for different schematic objects:

Components: CompInst@<CompName>;<compInstID>;<optional schematicID>

Geometric primitive: SchObj@<schematicID>

Global Port/ground:: GPort@<portName>;<schematicID>

Interface Port:: IPort@<portName>;<schematicID>

Page Port: PagePort@<portName>;<schematicID>

Wire: Wire@<netName>;<schematicID>;<segment index list>

Find this information in the Properties Window for the selected object as follows:

<CompName> General tab/CompName

<compInstID> General tab/ID

<schematicID> Symbol tab/SchematicID

<portName> Param Values tab/PortName

<netName> General tab/NetName

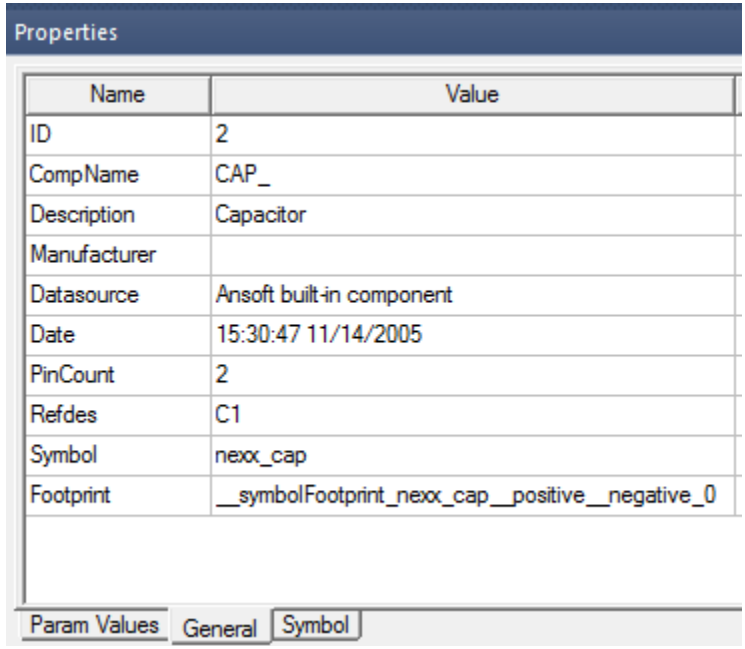
<segment index list> numbers separated by commas; Symbol tab/Segmentn - e.g. Segment0 refers to index "0"

In the Layout, the ID for a selected object is always the Name or Net property in the Footprint tab of the Properties Window

Format for Components

```
Array("CompInst@CAP_;2;4"))
```

The format for Components is CompInst@<CompName>;<compInstID>;<optional schematicID>. In the documented example, <CompName> stands for the component name CAP_, <compInstID> stands for its ID 2, and <optional schematicID> is the schematic ID 4. If you select the component, the **Properties** window gets updated and displays its details. For example, the selected object's component name and ID appear in the **General** tab of the **Properties** window as shown below.



The SchematicID is shown in the **Symbol** tab.

Create Method List

This section lists the scripts that are available in the Schematic scripting interface to create and change data.

Script Position Parameters

In Create Method scripts, X and Y position parameters are expressed in meters.

- If your layout grid units are metric, you may enter the X and Y positions directly into a script as parameters.
- If your layout grid is expressed in mils, you must first convert the component's positional coordinates to meters.

To convert an X or Y position to meters based on the minor grid size:

X or Y position in meters = desired_position x (0.00254 / minor_grid_size)

where desired_position and minor_grid_size have the same units. For example, to position an object at 500mm with a minor grid size of 20mm:

X or Y position in meters = 500 x (0.00254 / 20)

This section lists the following commands:

CreateArc (Schematic Editor)

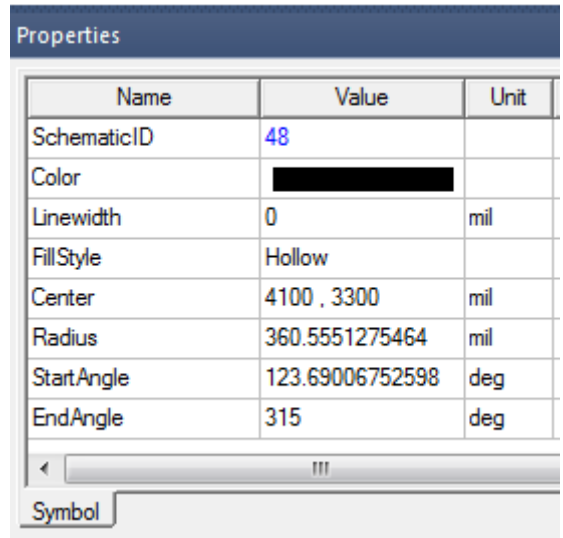
Use: Create an arc

Syntax:

```
CreateArc(
        Array("NAME:ArcData"          _
              "x:=", double,          _ // X position of the object
              "y:=", double,          _ // Y position of the object
              "Radius:=", double,     _ // Radius of the circle
             "StartAng:=", double,_ // Start angle of the arc (radians)
              "EndAng:=", double,     _ // End angle of the arc (radians)
              "Id:=", int),             // [Opt=New id] Id for this item
        Array("NAME:Attributes", _
              "Page:=", int)        _ // [Opt=1] Page number (one-based)
                  )
```

Return Value: Arc is created with the format `SchObj@<schematicID>`

The Schematic ID can be found on the **Symbols** tab of the **Properties** window when you select the arc.



Example:

```
' ----------------------------------------------
' Script Recorded by ANSYS Electronics Desktop Version 2015.0.0
' 10:37:08  Oct 30, 2014
' ----------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project4")
Set oDesign = oProject.SetActiveDesign("Circuit1")
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
Dim ArcID
```

```
ArcID = oEditor.CreateArc(Array("NAME:ArcData", "X:=", 0.10414,
"Y:=", 0.08382, "Radius:=",  _
  0.00915810023967853, "StartAng:=", 2.15879893034246, "EndAng:=",  _
  5.49778714378214, "LineWidth:=", 0, "Color:=", 0, "Id:=", 48),
Array("NAME:Attributes", "Page:=",  _
  1))
MsgBox "Arc ID = " & ArcID
```

CreateCircle (Schematic Editor)

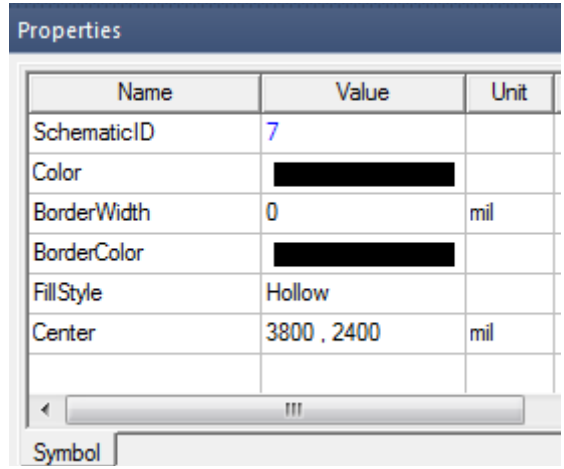
Use: Create a circle

Syntax:

```
CreateCircle(
        Array("NAME:CircleData"    _
              "x:=", double,       _    // X position of the object
              "y:=", double,       _    // Y position of the object
              "Radius:=", double,  _    // Radius of the circle
              "Id:=", int),             // [Opt=New id] Id for this item
        Array("NAME:Attributes", _
              "Page:=", int)       _ // [Opt=1] Page number (one-based)
              )
```

Return Value: Circle is created and the returned value has the format `SchObj@<schematicID>`

The schematic ID of the circle can be located in the **Properties** window on the **Symbols** tab.



CreateComponent (Schematic Editor)

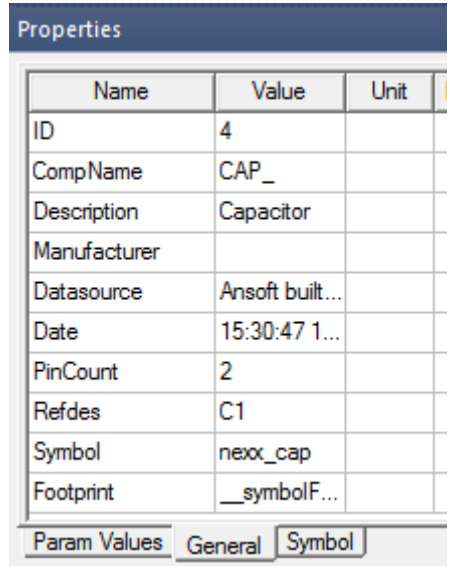
Use: Create a component of a given type

Syntax:

```
CreateComponent(
        Array("NAME:ComponentProps", _
              "Name:=", string,       _  // Component name (CAP, etc)
              "Id:=", string),           // Component id
        Array("NAME:Attributes", _
              "Page:=", int,      _ // [Opt=1] Page number (one-based)
                                    //   Note: Page 1 always exists
              "X:=", double,      _ // X position of the object
              "Y:=", double,      _ // Y position of the object
              "Angle:=", double, _ // Rotation angle (radians)
              "Flip:=", bool) // True if mirrored
             )
```

Return Value: Component is created and the returned value has the format `CompInst@<CompName>;<compInstID>;<optional schematicID>`

For instance a return value `CompInst@CAP_;4;35` indicates that the component name is CAP_, its ID is 5, and its Schematic ID is 35. These details can be found in the **Properties** window when you select the created component. The component name and ID can be found in the **General** tab and the Schematic ID can be found in the **Symbols** tab.



Example:

```
' ----------------------------------------------------------------
' Script Recorded by ANSYS Electronics Desktop Version 2015.0.0
' 10:07:39  Oct 30, 2014
' ----------------------------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project4")
Set oDesign = oProject.SetActiveDesign("Circuit1")
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
Dim CompID
CompID = oEditor.CreateComponent(Array("NAME:ComponentProps",
"Name:=",  _
  "Nexxim Circuit Elements\Capacitors:CAP_", "Id:=", "4"),
Array("NAME:Attributes", "Page:=",  _
  1, "X:=", 0.11176, "Y:=", 0.0635, "Angle:=", 0, "Flip:=", false))

MsgBox "Comp ID = " & CompID
```

CreateGlobalPort (Schematic Editor)

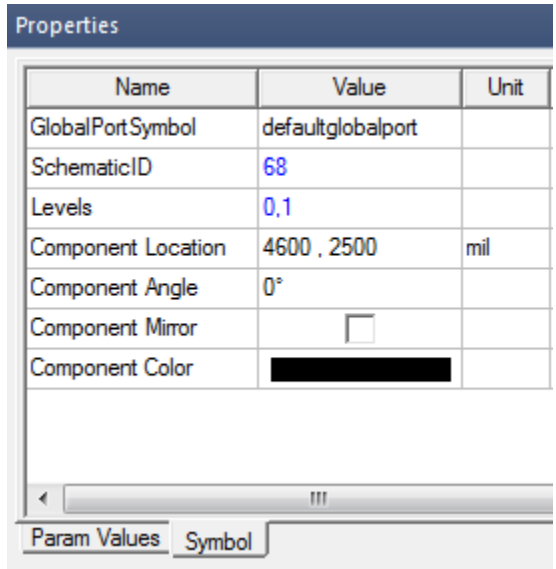
Use: Create a global port

Syntax:

```
CreateGlobalPort(
       Array("NAME:GlobalPortProps",_
"Name:=", string), // [Opt=default name] Name for this port
"Id:=", int),        // [Opt=New id] Id number for this item
Array("NAME:Attributes", _
"Page:=", int,     _ // [Opt=1] Page number (one-based)
```

```
//   Note: Page 1 always exists
"X:=", double,      _ // X position of the object
"Y:=", double,      _ // Y position of the object
"Angle:=", double, _ // Rotation angle (radians)
"Flip:=", bool) // True if mirrored
)
```

Return Value: Global port is created and the return value has the format `GPort@<portName>;<schematicID>`
The Schematic ID can be found in the **Properties** window on the **Symbols** tab when you select the global port.



Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesk-
top")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
```

```
Set oProject = oDesktop.SetActiveProject("Project4")
Set oDesign = oProject.SetActiveDesign("Circuit1")
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
Dim GPort
GPort = oEditor.CreateGlobalPort(Array("NAME:GlobalPort-
Props", "Name:=", "G_1", "Id:=", 68), Array("NAME:Attri-
butes", "Page:=",  _
  1, "X:=", 0.11684, "Y:=", 0.0635, "Angle:=", 0,
"Flip:=", false))
MsgBox "Port = " & GPort
```

CreateGround (Schematic Editor)

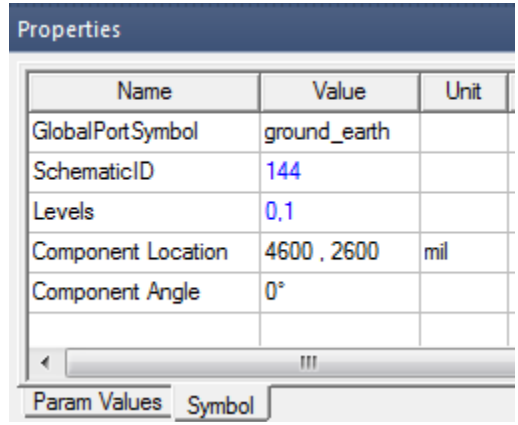
Use: Create a ground

Syntax:

```
CreateGround(
        Array("NAME:GroundProps",_
              "Id:=", int),        // [Opt=New id] Id for this item
        Array("NAME:Attributes", _
              "Page:=", int,     _ // [Opt=1] Page number (one-based)
                                   //   Note: Page 1 always exists
              "X:=", double,     _ // X position of the object
              "Y:=", double,     _ // Y position of the object
              "Angle:=", double, _ // Rotation angle (radians)
              "Flip:=", bool) // True if mirrored
                  )
```

Return Value: Ground is created and the returned value has the format `GPort@<portname>;<schematicID>`

The Schematic ID for ground can be found on the **Symbols** tab of the **Properties** window when you select the object.



Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesk-
top")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project4")
Set oDesign = oProject.SetActiveDesign("Circuit1")
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
Gnd = oEditor.CreateGround(Array("NAME:GroundProps",
"Id:=", 144), Array("NAME:Attributes", "Page:=",  _
  1, "X:=", 0.11684, "Y:=", 0.06604, "Angle:=", 0,
"Flip:=", false))

MsgBox  "GndIndex =" & Gnd
```

CreateLine (Schematic Editor)

Use: Create a line

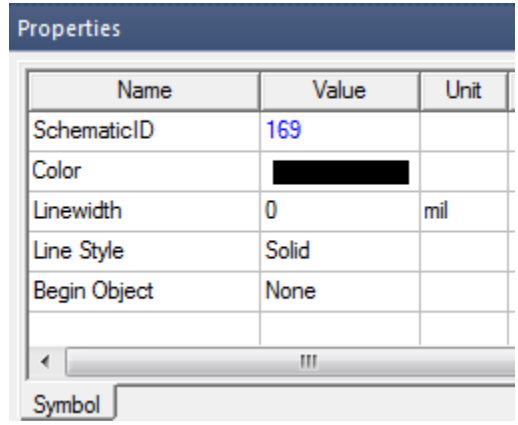
Syntax:

```
CreateLine(
```

```
Array("NAME:LineData"    _
      "Points:=", Array of points, _
      "Id:=", int),         // [Opt=New id] Id for this item
Array("NAME:Attributes", _
      "Page:=", int)      _ // [Opt=1] Page number (one-based)
     )
```

Return Value: Line is created and the returned value has the format `SchObj@<schematicID>`

The Schematic ID can be found on the **Symbol** tab of the **Properties** window when you select the line.



Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesk-
top")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project4")
```

```
Set oDesign = oProject.SetActiveDesign("Circuit1")
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
LineID = oEditor.CreateLine(Array("NAME:LineData",
"Points:=", Array("(0.050800, 0.106680)",  _
  "(0.124460, 0.106680)", "(0.124460, 0.106680)"), "Line-
Width:=", 0, "Color:=",  _
  0, "Id:=", 169), Array("NAME:Attributes", "Page:=", 1))
MsgBox "LineID = " & LineID
```

CreateNPort (Schematic Editor)

Use: Creates an N-Port definition and component and adds them to the current project, layout, and schematic.

Command: CreateNPort

Syntax:

```
CreateNPort
Array("NAME:Contents",
    "definition_name:=",<nport_definition_name>,
    "placement:=",      <component_placement>,
    "layer:=",          <placement_layer_name>,
    <nport_data_definition>)

<nport_definition_name>:
    quoted string (name of the component definition)

<component_placement>:
    Array("x:=",       <value>, // x coordinate
    "y:=",       <value>, // y coordinate
    "scaling:=", <value>) // double
```

Return Value: Returns the name of the newly created component.

<nport_data_definition> - see the I/O format in TODO

Example:

```
Example:
oEditor.CreateNPort
    Array("NAME:Contents",
          "definition_name:=", "NetworkData3",
          "placement:=",
              Array("x:=",  "-9mm",
```

```
                    "y:=", "-5mm",
                    "scaling:=", "2"),
               "layer:=", "Symbols",
                Array("NAME:NPortData",
                    Array("NAME:NetworkData2",
                          "filelocation:=", "UsePath",
                          "filename:=", "",
                          "domain:=", "frequency",
                          "numberofports:=", 2,
                          "datamode:=", "Import",
                          "devicename:=", "", "
                          ImpedanceTab:=", 1,
                          "NoiseDataTab:=", 1,
                          "DCBehaviorTab:=",  1,
                          "SolutionName:=", "",
                          "dcbehavior:=", "DCOpen",
                          "displayformat:=", "MagnitudePhase",
                          "datatype:=", "SMatrix",
                          "interptype:=", "Linear",
                          "extraptype:=",  "Same as interpolation",
                          "ShowRefPin:=", 0,
                          "RefNodeCheckbox:=", 1)))
```

CreatePagePort (Schematic Editor)

Use: Create a pageport

Syntax:

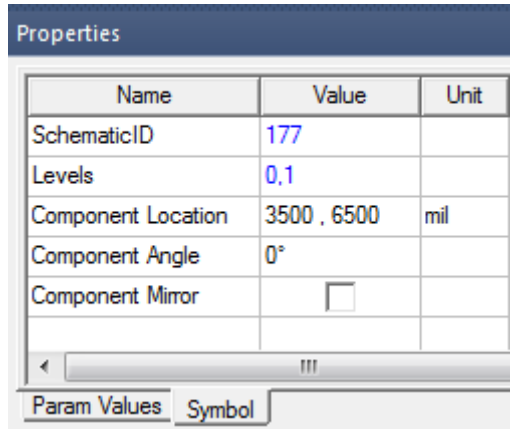
```
CreatePagePort(
       Array("NAME:PagePortProps"),
       Array("NAME:Attributes", _
             "Page:=", int,     _ // [Opt=1] Page number (one-based)
                                  //   Note: Page 1 always exists
             "X:=", double,     _ // X position of the object
             "Y:=", double,     _ // Y position of the object
             "Angle:=", double, _ // Rotation angle (radians)
             "Flip:=", bool) // True if mirrored
```

```
)
```

Return Value: The pageport is created and it has the following format:

```
PagePort@<portName>;<schematicID>
```

The schematic ID can be found in the **Properties** window on the **Symbols** tab when you select the pageport.



Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesk-
top")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project4")
Set oDesign = oProject.SetActiveDesign("Circuit1")
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
PagePortID = oEditor.CreatePagePort(Array("NAME:PagePort-
Props", "Name:=", "pageport_0", "Id:=",  _
  177), Array("NAME:Attributes", "Page:=", 1, "X:=",
0.0889, "Y:=", 0.1651, "Angle:=",  _
```

```
                0, "Flip:=", false))
            MsgBox "PagePort = " & PagePortID
```

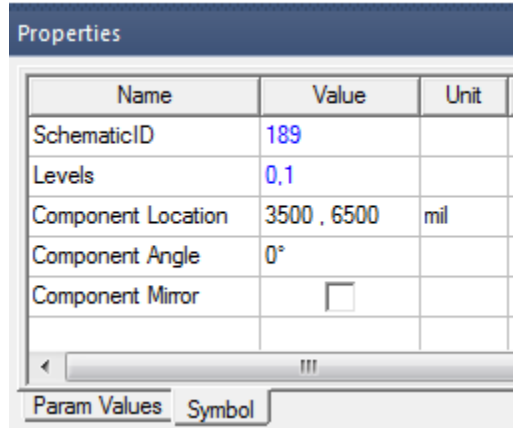
CreateIPort (Schematic Editor)

Use: Create an interface port

Syntax:

```
CreateIPort(
        Array("NAME:IPortProps"),
              "Name:=", string),   // [Opt=default name] Name for this
port
              "Id:=", int),        // Port id number
        Array("NAME:Attributes", _
              "Page:=", int,     _ // [Opt=1] Page number (one-based)
                                   //   Note: Page 1 always exists
              "X:=", double,     _ // X position of the object
              "Y:=", double,     _ // Y position of the object
              "Angle:=", double, _ // Rotation angle (radians)
              "Flip:=", bool) // True if mirrored
```

Return Value: Interface port is created and the returned value has the following format: IPort@<portName>;<schematicID>
The interface port schematic ID can be found in the **Properties** window on the **Symbols** tab.



Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesk-
top")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project4")
Set oDesign = oProject.SetActiveDesign("Circuit1")
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
IPortID = oEditor.CreateIPort(Array("NAME:IPortProps",
"Name:=", "Port1", "Id:=", 189), Array("NAME:Attributes",
"Page:=",  _
  1, "X:=", 0.0889, "Y:=", 0.1651, "Angle:=", 0,
"Flip:=", false))
MsgBox "IPort ID = " & IPortID
```

CreatePolygon (Schematic Editor)

Use: Create a polygon

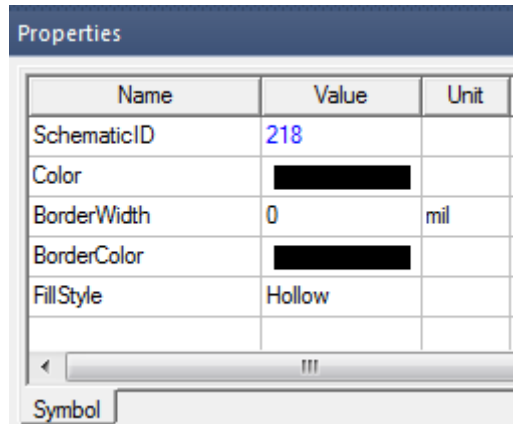
Syntax:

```
CreatePolygon(
       Array("NAME:PolygonData"   _
             "Points:=", Array of points, _
             "Id:=", int),          // [Opt=New id] Id for this item
       Array("NAME:Attributes", _
             "Page:=", int)     _ // [Opt=1] Page number (one-based)
              )
```

Return Value: Polygon is created and the return value has the format

`SchObj@<schematicID>`

The Schematic ID can be found in the **Properties** window on the **Symbol** tab when you select the polygon.



Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesk-
top")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project4")
Set oDesign = oProject.SetActiveDesign("Circuit1")
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
polygonID = oEditor.CreatePolygon(Array("NAME:Polygon-
Data", "Points:=", Array( _
  "(0.101600, 0.058420)", "(0.101600, 0.060960)",
"(0.101600, 0.058420)",  _
  "(0.099060, 0.060960)", "(0.101600, 0.060960)",
"(0.101600, 0.058420)",  _
  "(0.101600, 0.060960)", "(0.101600, 0.060960)"), "Line-
Width:=", 0, "BorderColor:=",  _
 0, "Fill:=", 0, "Color:=", 0, "Id:=", 218),
Array("NAME:Attributes", "Page:=", 1))
MsgBox "Polygon ID = " & polygonID
```

CreateRectangle (Schematic Editor)

Use: Create a rectangle

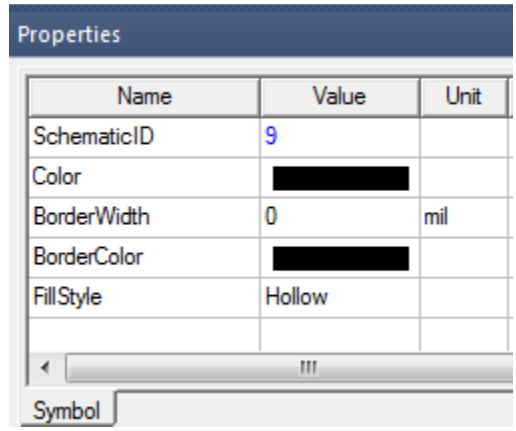
Syntax:

```
CreateRectangle(
        Array("NAME:RectData" _
              "x1:=", double, _   // X position of the upper left
              "y1:=", double, _   // Y position of the upper left
              "x2:=", double, _   // X position of the lower right
              "y2:=", double",_   // Y position of the lower right
              "Id:=", int),       // [Opt=New id] Id for this item
        Array("NAME:Attributes", _
              "Page:=", int)      _ // [Opt=1] Page number (one-based)
             )
```

Return Value: Rectangle is created and the return value has the format

`SchObj@<schematicID>`

The Schematic ID can be found in the **Properties** window on the **Symbol** tab when you select the rectangle.



Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
```

```
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesk-
top")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project4")
Set oDesign = oProject.SetActiveDesign("Circuit1")
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
rectangleID = oEditor.CreateRectangle(Array("NAME:Rect-
Data", "X1:=", 0.1016, "Y1:=", 0.0635, "X2:=",  _
  0.10414, "Y2:=", 0.06096, "LineWidth:=", 0, "Border-
Color:=", 0, "Fill:=", 0, "Color:=",  _
  0, "Id:=", 9), Array("NAME:Attributes", "Page:=", 1))
MsgBox "Rectangle ID = " & rectangleID
```

CreateText (Schematic Editor)

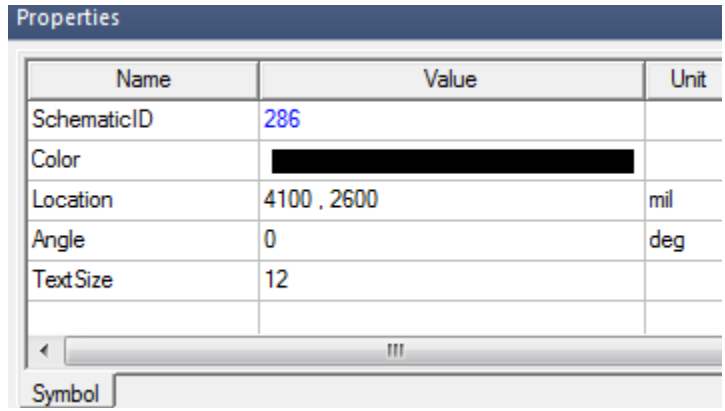
Use: Create text

Syntax:

```
CreateText(
      Array("NAME:TextData"      _
            "x:=", double,       _ // X position of the object
            "y:=", double,       _ // Y position of the object
            "Text:=", string,    _ // Text to display
            "Id:=", int),          // Id for this item
      Array("NAME:Attributes", _
            "Page:=", int)     _ // [Opt=1] Page number (one-based)
           )
```

Return Value: Text is created and the return value has the format: SchObj@<SchematicID>

The Schematic ID can be found in the **Properties** window on the **Symbols** tab when you select the text.



Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project4")
Set oDesign = oProject.SetActiveDesign("Circuit1")
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
textobj = oEditor.CreateText(Array("NAME:TextData", "X:=", 0.10414,
"Y:=", 0.06604, "Size:=",  _
  12, "Angle:=", 0, "Text:=", "CreateDefaultTectxt" & Chr(13) &
Chr(10) & "", "Color:=",  _
  0, "Id:=", 286, "ShowRect:=", false, "X1:=", 0.100141851851836,
"Y1:=",  _
  0.0670983333333376, "X2:=", 0.140123333333477, "Y2:=",
0.0565149999999619, "RectLineWidth:=",  _
  0, "RectBorderColor:=", 0, "RectFill:=", 0, "RectColor:=", 0),
Array("NAME:Attributes", "Page:=",  _
  1))
MsgBox "text = " & textobj
```

CreateWire (Schematic Editor)

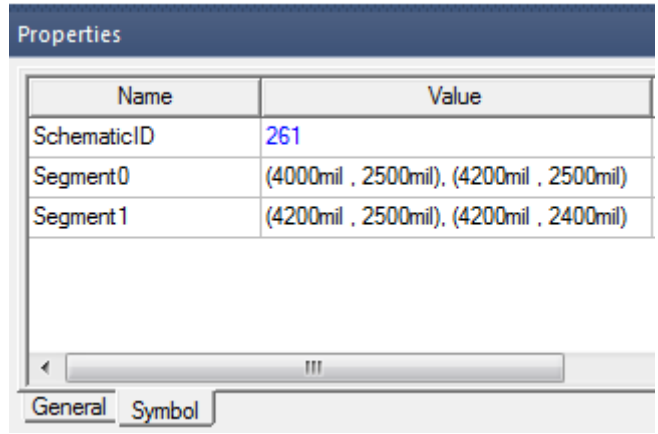
Use: Create a wire

Syntax:

```
CreateWire(
       Array("NAME:WireData" _
            "Name:=", string), // [Opt=default name] Name
for this wire
             "Id:=", int),         // Wire idnum
             "Points:=", Points on wire),
       Array("NAME:Attributes", _
             "Page:=", int,      _ // [Opt=1] Page number
(one-based)
                                    //   Note: Page 1 always
exists
)
```

Return Value: Wire is created and its return value has the format `Wire@<netName>;<schematicID>;<segment index list>`

The Schematic ID can be found in the **Symbol** tab of the **Properties** window when you select the wire. The wire net name appears in the **General** tab. The parameter segment index list indicates the number of segments i.e. the segment count.



Example:

```
Dim oAnsoftApp
Dim oDesktop
```

```
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesk-
top")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project4")
Set oDesign = oProject.SetActiveDesign("Circuit1")
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
WireID = oEditor.CreateWire(Array("NAME:WireData",
"Name:=", "", "Id:=", 261, "Points:=", Array( _
  "(0.101600, 0.063500)", "(0.106680, 0.063500)",
"(0.106680, 0.060960)")), Array("NAME:Attributes",
"Page:=",  _
  1))
MsgBox "Wire ID = " & WireID
```

General Method List

This section presents the general script methods that are available.

The general method script commands are listed here.

Activate (Schematic Editor)

AddPinGrounds (Schematic Editor)

AddPinIPorts (Schematic Editor)

AddPinPageConnectors (Schematic Editor)

AlignHorizontal (Schematic Editor)

AlignVertical (Schematic Editor)

BringToFront (Schematic Editor)

CloseEditor (Schematic Editor)

Copy (Schematic Editor)

CreatePage (Schematic Editor)

Cut (Schematic Editor)

DeactivateOpen (Schematic Editor)

DeactivateShort (Schematic Editor)

Delete (Schematic Editor)

Activate (Schematic Editor)

```
// Enable items previously disabled
Activate (
       Array("NAME:Selections", _
             "Page:=", page number, _  // [Opt=1] Page number
             "Selections:=", IDs to modify))
```

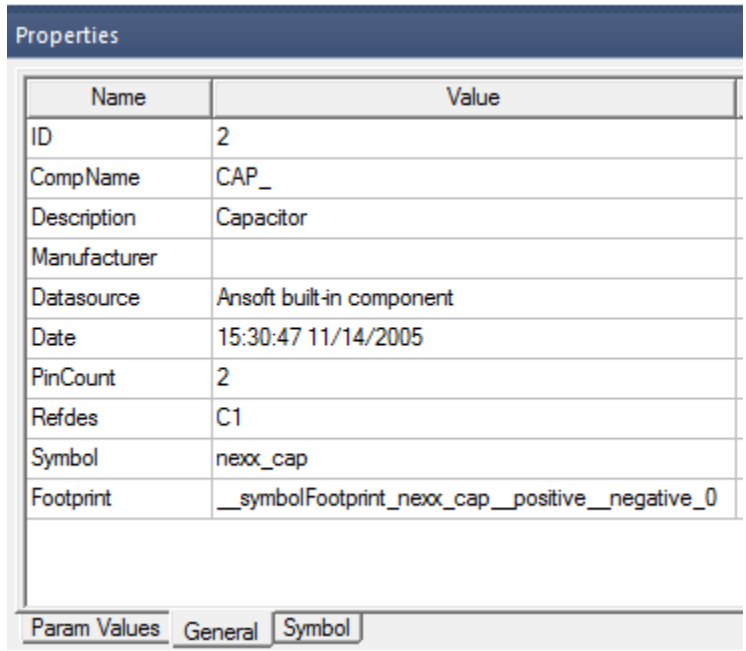
Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

AddPinGrounds (Schematic Editor)

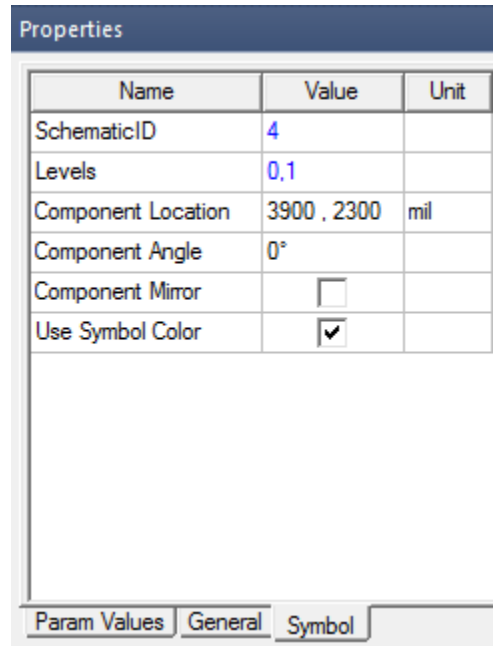
```
Adds grounds at all unconnected pins
AddPinGrounds (
```

```
        Array("NAME:Selections", _ // PagePort Name
              "Selections:=", _ // Net Name
Array("CompInst@CAP_;2;4"))
```

Note: The format for Components is `CompInst@<CompName>;<compInstID>;<optional schematicID>`. In the documented example, `<CompName>` stands for the component name CAP_, `<compInstID>` stands for its ID 2, and `<optional schematicID> is the` schematic ID 4. If you select the component, the **Properties** window gets updated and displays its details. For example, the selected object's component name and ID appear in the **General** tab of the **Properties** window as shown below.



To view the SchematicID click the **Symbol** tab.



AddPinIPorts (Schematic Editor)

```
AddPinPageIPorts (
       Array("NAME:Selections", _ // PagePort Name
             "Selections:=", _ // Net Name
Array("CompInst@C;2;3"))
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

AddPinPageConnectors (Schematic Editor)

```
GeneralAddPinPageConnectors
// Adds PagePorts at unconnected pins of all selected components
AddPinPageConnectors (
       Array("NAME:Selections", _ // PagePort Name
             "Selections:=", _ // Net Name
Array("CompInst@C;2;3"))
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

AlignHorizontal (Schematic Editor)

```
GeneralAlignHorizontal
// Align items horizontally
AlignHorizontal(
            Array("NAME:Selections", _
                  "Page:=", page number, _  // [Opt=1] Page number
                  "Selections:=", IDs to modify)),
            Array("NAME:AlignParameters", _
            "Disconnect:=", bool  _  // [Opt=0] Should wires disconnect
             "Rubberband:=", bool) _  // [Opt=1] Should wires staircase
                                  // Note: Alignment occurs relative
                                          to the first item in ids
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

AlignVertical (Schematic Editor)

```
// Align items vertically
AlignVertical(
            Array("NAME:Selections", _
                  "Page:=", page number, _  // [Opt=1] Page number
                  "Selections:=", IDs to modify)),
            Array("NAME:AlignParameters", _
            "Disconnect:=", bool  _  // [Opt=0] Should wires disconnect
             "Rubberband:=", bool) _  // [Opt=1] Should wires staircase
                                 // Note: Alignment occurs relative
                                         to the first item in ids
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

BringToFront (Schematic Editor)

Use: Changes the drawing for the schematic so that the specified objects are drawn on top of other overlapping objects.

Command: Draw > Bring To Front

Syntax:
```
BringToFront Array("NAME:Selections", "Selections:=",
Array (<Object>, <Object>, ...))
```

Return Value: None

Parameters:
```
<Object>
<string>  // object to bring to the front
```

Example:
```
oDefinitionEditor.BringToFront Array("NAME:Selections",
"Selections:=", Array(  "SchObj@10"))
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

CloseEditor (Schematic Editor)

```
// Close an Editor
CloseEditor(
VARIANT ptDelta) // The specified Editor to close
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

Copy (Schematic Editor)

```
GeneralCopy
// Copy items for pasting
Copy(
       Array("NAME:Selections", _
            "Page:=", page number, _  // [Opt=1] Page number
            "Selections:=", IDs to modify))
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

CreatePage (Schematic Editor)

```
// Create a page to the end of the pages
CreatePage(
string name,           // Page name
           [out,retval] int num)  // Page number
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

Cut (Schematic Editor)

```
// Cut Page Selection
Cut(
       Array("NAME:Selections", _
            "Page:=", page number, _  // [Opt=1] Page number
            "Selections:=", IDs to modify))
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

DeactivateOpen (Schematic Editor)

```
// Disable items and replace with an open circuit
DeactivateOpen(
           Array("NAME:Selections", _
                "Page:=", page number, _  // [Opt=1] Page number
                "Selections:=", IDs to modify))
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

DeactivateShort (Schematic Editor)

```
// Disable items and replace with a closed circuit
DeactivateShort (
           Array("NAME:Selections", _
                "Page:=", page number, _  // [Opt=1] Page number
                "Selections:=", IDs to modify))
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

Delete (Schematic Editor)

```
// Delete items
Delete(
      Array("NAME:Selections", _
            "Page:=", page number, _  // [Opt=1] Page number
            "Selections:=", IDs to modify))
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

DeletePage (Schematic Editor)

```
// Delete a page
DeletePage(
int page)   // Page number
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

ElectricRuleCheck (Schematic Editor)

```
// Check Electric Rule
ElectricRuleCheck(
            Array(string, ...), bool)

Array
// Array of strings containing names of electric rules to perform.
// Valid names are:
// "PinRule": Checks for components with unconnected pins
// "OutputPinRule": Checks for nets with more than one output pin

bool
// True if this should check all subcircuits
// False if this should just check this circuit
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

ExportImage (Schematic Editor)

Use:	To export a picture for a specified page of the current design to a file. The image size can also be specified. The filename extension determines the type of image exported.
Command:	Product > Add Subcircuit
Syntax:	`ExportImage (<filename>, <pagenum>, <dx>, <dy>)`
Return Value:	None
Parameters:	`<filename>` Type: String The name of the file, with format-specific extension. Extensions supported are: bmp, gif, jpg, jpeg, png, tif, tiff. <pagenum>

Type: Integer

The page number of the current schematic. Page numbers start at 1. If the number is out of range, the current schematic page will be printed.

<dx>

Type: Integer

The width of the image. If dx is less than 160, 160 will be used for the width.

<dy>

Type: Integer

The height of the image. If dy is less than 160, 160 will be used for the height.

Example:

```
oEditor.ExportImage "c:\mysch.png", 1, 800, 400
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

FindElements (Schematic Editor)

```
// Select elements based on properties
FindElements(
ARRAY props, // specified properties
ARRAY params) // specified parameters
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

GridSetup (Schematic Editor)

```
// Changes the settings on the schematic grid
GridSetup(
          Array("NAME:Options", _
                "MajorGrid:=", string, _    // Major grid size with
units
                "Divisions:=", int, _       // Number of minor grid
divisions
               "MajorColor:=", int, _      // RGB Color of major grid
lines
               "MinorColor:=", int, _      // RGB Color of minor grid
lines
              "ShowGrid:=", bool, _       // Should the grid be shown?
               "SnapToGrid:=", bool, _     // Should objects snap to
grid?
                "BackgroundColor:=", int, _ // RGB Color of the back-
ground
```

```
                    "SaveAsDefault:=", bool))   // Should these settings
be the default for new schematics

For example:

oEditor.GridSetup Array(
"NAME:Options", _
"MajorGrid:=", "10mm", _
"Divisions:=", 10, _
"MajorColor:=", 0x00ff0000, _ // Red
"MinorColor:=", 0x0000ff00, _ // Green
"ShowGrid:=", true, _
"SnapToGrid:=", true, _
"BackgroundColor:=", 0x00ffffff, _ // White
"SaveAsDefault:=", false)
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

FlipHorizontal (Schematic Editor)

```
// Flip items about the horizontal (x) axis
FlipHorizontal(
           Array("NAME:Selections", _
                 "Page:=", page number, _  // [Opt=1] Page number
                 "Selections:=", IDs to modify)),
           Array("NAME: FlipParameters", _
           "Disconnect:=", bool  _  // [Opt=0] Should wires disconnect
            "Rubberband:=", bool) _  // [Opt=1] Should wires staircase
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

FlipVertical (Schematic Editor)

```
// Flip items about the vertical (y) axis
FlipVertical(
           Array("NAME:Selections", _
                 "Page:=", page number, _  // [Opt=1] Page number
                 "Selections:=", IDs to modify)),
           Array("NAME:FlipParameters", _
```

```
            "Disconnect:=", bool  _  // [Opt=0] Should wires disconnect
             "Rubberband:=", bool) _  // [Opt=1] Should wires staircase
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

Move (Schematic Editor)

```
// Move items
Move(
     Array("NAME:Selections", _
           "Page:=", page number, _  // [Opt=1] Page number
           "Selections:=", IDs to modify)),
     Array("NAME:MoveParameters", _
           "xdelta:=", double     _  // X distance to move
           "ydelta:=", double     _  // Y distance to move
          "Disconnect:=", bool   _  // [Opt=0] Should wires disconnect
          "Rubberband:=", bool)  _  // [Opt=1] Should wires staircase
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

NameNets (Schematic Editor)

```
// Change names of nets on the schematic
NameNets(
       bool display, // Show Net names or not
       Array("NAME:NetNames", _
                "string:=", _   // Old name of the net (e.g. net_1)
                string, _       // New name of the net (e.g. my_net)
               ...)           // Repeat for additional nets to change
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

PageBorders (Schematic Editor)

```
// Sets up visible page borders on the schematic
PageBorders(
Array("NAME:Options", _
                 "PageSize:=", _
                Array("x:=", string, _  // Width of page border with
units
```

```
                        "y:=", string), _ // Height of page border with
units
                    "PageMargins:=", _
                    Array("x:=", string, _  // Margin Width with units
                          "y:=", string), _ // Margin Height with units
                  "ZonesHoriz:=", int, _  // Number of Horizontal zones
                    "ZonesVert:=", int))    // Number of Vertical zones
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

Pan (Schematic Editor)

```
// Pan Display
Pan(Array(
double, _  // Move the visable area by X meters
          double))   // Move the visable area by Y meters
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

Paste (Schematic Editor)

```
// Paste copied items
Paste(
VARIANT attrs); // specified attributes
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

PushExcitations (Schematic Editor)

```
// Allows access to computed excitations
PushExcitations "<refdes>",
Array("NAME:options", "transient:=",
Array("start:=", <start time>,
"stop:=", <stop time>,
"maxHarmonics:=", <max harmonics>,
"winType:=", <window>,
["widthPct:=",  <width percentage>,]
["kaiser:=", <Kaiser value>,]
["correctCoherentGain:=", true]),
"Sol:=", "<solution name>")
```

```
winType can have the following values:
    Rectangular
    Bartlett
    Blackman
    Hamming
    Hanning
    Kaiser
    Welch
    Weber
    Lanzcos
```

Example:

```
Examples:

Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
oEditor.PushExcitations "U3", Array("NAME:options",
"transient:=", Array("start:=",  _
0, "stop:=", 5E-005, "maxHarmonics:=", 100, "winType:=",
"Rectangular", "widthPct:=",  _
100, "kaiser:=", 0, "correctCoherentGain:=", true),
"Sol:=", "Transient")

Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
oEditor.PushExcitations "<refdes>", Array("NAME:options",
"transient:=", Array("start:=",  _
<start time>, "stop:=", <stop time>, "maxHarmonics:=",
<max harmonics>, "winType:=", <window>, ["widthPct:=",  _
<width percentage>,] ["kaiser:=", <Kaiser value>,] ["cor-
rectCoherentGain:=", true]), "Sol:=", "<solution name>")

Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
oEditor.PushExcitations "U2", Array("NAME:options",
"Freqs:=", Array(100000000,  _
200000000, 300000000, 400000000, 500000000, 600000000,
700000000, 800000000,  _
900000000, 1000000000), "Sol:=", "LNA")
```

Note If no frequencies are specified, all frequencies from the solution are used.

```
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
oEditor.PushExcitations "U2", Array("NAME:options", "Sol:=", "LNA")
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

Rotate (Schematic Editor)

```
// Rotate items
Rotate(
       Array("NAME:Selections", _
             "Page:=", page number, _  // [Opt=1] Page number
             "Selections:=", IDs to modify)),
       Array("NAME:RotateParameters", _
            "Disconnect:=", bool _  // [Opt=0] Should wires disconnect
             "Rubberband:=", bool _  // [Opt=1] Should wires staircase
             "Angle:=", double)      // [Opt=/2] Angle to rotate
                       // Note: Rotation occurs around center of ids
```

Note When you record a script, the schematic editor's Rotate command shows the rotation angle for some older scripts in radians, while newer scripts show rotation angle in degrees.

```
oEditor.Rotate Array("NAME:Selections", "Selections:=", _
Array("CompInst@RES_;1;1:1")), Array("NAME:RotateParameters", _
"Angle:=", 1.5707963267949, "Disconnect:=", false, "Rubberband:=", false)
```

If “Degrees=xxx” is not shown, the rotation must be scaled.

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

SelectAll (Schematic Editor)

```
// Select all elements on the given page.
```

```
SelectAll(
int pageNum)                // Page number

// The first page number is 1.  If an invalid page number is passed in
(i.e. 0, -1, etc), SelectAll will use the active page on the currently
active view.  If there is no active view, the first page, page 1, is
used.
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

SelectPage (Schematic Editor)

```
// Select a page in the UI
SelectPage(
int page)                // Page number
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

SendToBack(Schematic Editor)

Use: Changes the drawing for the schematic so that the specified objects are drawn behind other overlapping objects.

Command: Draw > Send To Back

Syntax:
```
SendToBack Array("NAME:Selections", "Selections:=",
Array (<Object>, <Object>, ...))
```

Return Value: None

Parameters:
```
<Object>
<string>  // object to send to the back
```

Example:
```
oDefinitionEditor.SendToBack Array("NAME:Selections",
"Selections:=", Array(  "SchObj@10"))
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

SortComponents (Schematic Editor)

```
// Sorts all components, or a block of components, using the specified
method
SortComponents (
       Array(type, _ // string specifying sort type
method)) // string specifying sort method


type
```

```
// string argument specifying sort type: "All Components" or "Blocks"
// An empty string corresponds to "All Components"

method
// string argument specifying sort method: "By Name", "Left to Right",
or "Signal Flow"
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

ZoomArea (Schematic Editor)

```
// Zoon Area
ZoomArea(
Array(double, _   // Base X value of the area to zoom into
              double), _  // Base Y value of the area to zoom into
        Array(double, _   // X size of the area to zoom
              double))    // Y Size of the area to zoom

                          // NOTE: All values in meters
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

ZoomIn (Schematic Editor)

```
// Zoom into the schematic
ZoomIn()
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

ZoomOut (Schematic Editor)

```
// Zoom out from the schematic
ZoomOut()
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

ZoomPrevious (Schematic Editor)

```
// Restore the zoom to the previous settings
ZoomPrevious()
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

ZoomToFit (Schematic Editor)

```
// Set the Zoom to Fit
ZoomToFit() // Set the current schematic zoom to fit the contents of the currently visible page
```

Note: For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

Property Method List

This section presents the property script methods that are available.

ChangeProperty (Schematic Editor)

```
// Change a property
ChangeProperty(
Array("NAME:AllTabs",
Array("NAME:string",  // tab name -
                              ComponentTab,       - component instance
information properties
                             PassedParameterTab, - component instance
parameters
                                BaseElementTab      - symbol data
                               PropdisplayTab      - property displays
Array("NAME:PropServers",
instId,   // inst IDs of objects whose properties are changing)
Array("NAME:keyword ", // ChangedProps, NewProps, or DeletedProps
Array("NAME:propname",
"PropType:=", type,  // TextProp
                               MenuProp
                               CheckboxProp
                               VariableProp
                               VPointProp
                               V3DPointProp
                               NumberProp
                               PointProp
                               ValueProp
                              ButtonProp (only netlist and file props)
                               SeparatorProp
"UserDef:=" bool  // NewProps: true or false
"Value:=", propvalue, // NewProps or ChangedProps
```

```
["SplitWires:=", true/false] // additional option for net and port
names with NewProps or ChangedProps)))))
```

GetEvaluatedPropertyValue (Schematic Editor)

Use: Get value.

Command: None.

Syntax: `GetEvaluatedPropertyValue<tabDescription, componentID, propName>`

Return Value: Evaluated value of variable property in double format.

Parameters: `tabDescription = name of property tab where property is found`

componentID = id of component instance where property is found,

in the format: "CompInst@<name of component type>;<id of compInstance>"

propName = name of the variable property

Example:

```
dim info
evalValue = oEditor.GetEvaluatedPropertyValue("PassedPar-
ameterTab", "CompInst@CAP_;3", "C")
```

Notes:

1. This function is only available with the schematic editor.
2. Calling this function on non-numeric properties (e.g. Text properties) returns 0.

GetProperties (Schematic Editor)

```
GetProperties(string tab,                    // Tab with the property
              string item,                   // Name of the object
              [out, retval] VARIANT props) // Array of prop names
```

GetPropertyValue (Schematic Editor)

```
// Get a property
GetPropertyValue(string tab,                  // Tab with the property
                 string item,                  // Name of the object
                 string propname,              // Name of the property
                [out, retval] string value)  // The value of the prop
```

SetPropertyValue (Schematic Editor)

```
// Set a property
SetPropertyValue(string tab,           // Tab with the property
                 string item,          // Name of the object
                 string propname,      // Name of the property
                 string value)         // The new value of the prop
```

Information Method List

This section presents the information methods that are available.

GetCompInstanceFromRefDes (Schematic Editor)
GetComponentInfo (Schematic Editor)
GetComponentPins (Schematic Editor)
GetComponentPinInfo (Schematic Editor)
GetEditorName (Schematic Editor)
GetNetConnections (Schematic Editor)
GetPortInfo (Schematic Editor)
GetSelections (Schematic Editor)
GetSignals (Schematic Editor)

GetCompInstanceFromRefDes (Schematic Editor)

Use: Informational.

Command: None.

Syntax: `GetCompInstanceFromRefDes(string)`
// string is refDes for the component

Return Value: Returns IDispatch for CompInstance.

Parameters: `None.`

GetComponentInfo (Schematic Editor)

Use: Informational.

Command: None.

Syntax: `GetComponentInfo<compID>`

Return Value: array of strings, as follows:
"ComponentName=*string*"
"PlacementLayer=*string*"
"LocationX=*number*"
"LocationY=*number*"

"BBoxLLx=*number*"
"BBoxLLy=*number*"
"BBoxURx=*number*"
"BBoxURy=*number*"
"Angle=*number" (in degrees)*
"Flip=true *or* false"
"Scale=*number*"

Parameters: `The LayoutComp's ID`

Example:
```
dim info
sys= oEditor.GetComponentInfo ("1")
```

GetComponentPins (Schematic Editor)

Use: Informational.

Command: None.

Syntax: `GetComponentPins<compID>`

Return Value: array of strings, which are the names of all the component's pins

Parameters: `The LayoutComp's ID`

Example:
```
dim pins
pins = oEditor.GetComponentPins ("1")
```

GetComponentPinInfo (Schematic Editor)

Use: Informational.

Command: None.

Syntax: `GetComponentPinInfo<compID, pinName>`

Return Value: retval = array of strings, as follows:

"X=val"
"Y=val"
“Angle=val”
"Flip=true/false”
“WireID=string”

Parameters: `compID = The LayoutComp's ID`
pinName = The name of the pin

Example:

```
dim info
                                    info =
oEditor.GetComponentPinInfo ("1", "n1")
```

GetEditorName (Schematic Editor)

Use: Informational.

Command: None.

Syntax:

```
GetEditorName( [out, retval] string)          // name of
the editor
```

Return Value: String containing the name of the editor.

Example:

```
dim info
                                    info = oEditor.GetEditorName
```

GetNetConnections (Schematic Editor)

Use: Informational.

Command: None.

Syntax: `GetNetConnections(<netName>)`

Return Value: Array of strings containing the connections for the net identified by the netName argument. Strings in the array are in one of four formats:

"IPort portID x y pinName"

"GPort portID x y pinName"

"PagePort portID x y pinName"

"Component componentName compID x y pinName"

where *IPort, GPort, PagePort, and Component* are key words that indicate what the rest of the string specifies:

IPort indicates that *portID* specifies the name of the interface port,

GPort indicates that *portID* specifies the name of the global port,

PagePort indicates that *portID* specifies the name of the page port,

Component indicates that *compID* specifies the component instance identifier,

x and *y* are the connection point,

pinName is the name of the connected pin,

componentName is the name of the component.

Parameters: `<netName>`

Type: String

The name of the net. Specifying "0" as the netName will return all objects connected to ground including all ground ports.

Example:

```
netArray = oEditor.GetNetConnections("net_1")
```

GetPortInfo (Schematic Editor)

Use: Informational.

Command: None.

Syntax: `GetPortInfo<portID>`

Return Value: array of strings, as follows:

"Name=string" // Name of the port.

"X=val" // Connection point X location.

"Y=val" // Connection point Y location.

"Angle=val" // Component angle.

"Flip=true or false" // Whether the component is flipped or not.

"WireId=string" // Unique net name.

Parameters: `The ID of the port`

Example:

```
dim info
info = oEditor.GetPortInfo("2")
```

GetSelections (Schematic Editor)

Use: Informational.

Command: None.

Syntax: `GetSelections`

Return Value: array of IDs

Parameters: `None.`

Example:

```
dim sels
sels = oEditor.GetSelections
```

GetSignals (Schematic Editor)

Use: Informational.

Command: None.

Syntax: `GetSignals`

Return Value: Array of strings which contain the signal names in the wirename argument.

Parameters: `GetSignals(string wirename, [out, retval] array of strings)`

GetWireConnections (Schematic Editor)

Use: Informational.

Command: None.

Syntax: `GetWireConnections`

Return Value: Array of strings containing the wire connections in the wirename argument.

Parameters:

```
GetWireConnections(string wire id, [out, retval] array of strings)
```

// the strings in the array are in one of four formats:

"InterfacePort=portname portID x,y"

"GlobalPort=portname portID x,y"

"PagePort=portname portID x,y"

"ComponentPin=compId pinname x,y"

where x,y is the connection point.

GetWireInfo (Schematic Editor)

Use: Informational.

Command: None.

Syntax:

```
GetWireInfo
```

Return Value: Array of strings containing the wire info in the wirename argument.

Parameters:

```
GetWireInfo(string wire id, [out, retval] array of strings)
```

// the strings in the array are:

"Page=number"

"WireName=string"

"SegmentCount=number"

"IPortCount=number"

"GPortCount=number"

"PagePortCount=number"

"PinCount=number"

"BBoxLLx=val"

"BBoxLLy=val"

"BBoxURx=val"

"BBoxURy=val"

GetWireSegments (Schematic Editor)

Use: Informational.

Command: None.

Syntax:

```
GetWireSegments
```

Return Value: Array of strings containing the wire segments in the wirename argument.

Parameters:

```
GetWireSegments (string wire id, [out, retval] array of strings)
```

```
// the strings in the array are:
"Segment=val,val val,val number" and the "val" quantities
are
segment endpoints x1, y1 x2, y2
```

28 Definition Manager Script Commands

The definition manager controls the use of materials and scripts in a Designer project. It also provides access to the managers for symbols, footprints, padstacks, and components in a Designer project.

```
Set oProject = oDesktop.SetActiveProject("Project1")
Set oDefinitionManager = oProject.GetDefinitionManager()
```

The topics for this section include:

Component Manager Script Commands

The component manager provides access to components in a Designer project. The manager object is accessed via the `definition manager`.

```
Set oDefinitionManager = oProject.GetDefinitionManager()
Set oComponentManager = oDefinitionManager.GetManager("Compo-
nent")
```

The component manager script commands are listed here:

Add
AddDynamicNPortData
AddNPortData
Edit
EditWithComps
Export
GetData
GetNames
GetNPortData
IsUsed
Remove
RemoveUnused

Add [component manager]

Use: Add a component

Command: Tools > Edit Configured Libraries > Components > Add Component

Syntax:

```
Add Array("NAME:<ComponentName>",
"Info:=", <ComponentInfo>,
"RefBase:=", <string>,              // reference designator
"NumParts:=", <int>,                // parts per component
"OriginalComponent:=", <string>
"Terminal:=", <TerminalInfo>,
"Terminal:=", <TerminalInfo>, ...
// The remaining parameters are optional
Array("NAME:Parameters",    // any combo of the following
"VariableProp:=", <VariableInfo>,
"CheckboxProp:=", <CheckBoxInfo>,
"ButtonProp:=", <ButtonInfo>,
"TextProp:=", <TextInfo>,
```

"NumberProp:=", <NumberInfo>,
"SeparatorProp:=", <SeparatorInfo>,
"ValueProp:=", <ValueInfo>,
"MenuProp:=", <MenuInfo>),
Array("NAME:Properties", // any combo of the following
"CheckboxProp:=", <CheckBoxInfo>,
"TextProp:=", <TextInfo>,
"NumberProp:=", <NumberInfo>,
"SeparatorProp:=", <SeparatorInfo>,
"ValueProp:=", <ValueInfo>,
"MenuProp:=", <MenuInfo>),
"VPointProp:=", <VPointInfo>,
"PointProp:=", <PointInfo>),
Array("Quantities",
"QuantityProp:=", <QuantityPropInfo>...),
Array("NAME:CosimDefinitions",
<CosimDefInfo>,
<CosimDefInfo>...)

Return Value: <string>
// composite name of the component.
// If the name requested conflicts with the name of an existing
// component, the requested name is altered to be unique.
// The name returned reflects any change made to be unique.

Parameters: `<ComponentName>:`
<string> // simple name of the component

<ComponentInfo>:
Array("Type:=", <TypeInfo>,
"NumTerminals:=", <int>,
"DataSource:=", <string>,
"ModifiedOn:=", <ModifiedOnInfo>,
"Manufacturer:=", "<string>,
"Symbol:=", <string>,
"Footprint:=", <string>,

"Description:=", <string>,
"InfoTopic:=", <string>,
"InfoHelpFile:=", <string>,
"IconFile:=", <string>,
"LibraryName:=", ““,
"OriginalLocation:=", “Project”, // Project Location
"Author:=", <string>,
"OriginalAuthor:=", <string>,
"CreationDate:= ", <int>)

<TypeInfo>:

An integer that is the or-ing of bits for each product listed below. The default setting is 0xffffffff (4294967295) which indicates valid for all products. In the component editing dialog, checking different boxes in the "Specify products for which this component is valid" grid control sets specific flags that correspond to the following hex/decimal settings:

Nexxim -- 100 binary, 4 decimal, 0x4

SIwaveDeNovo -- 1000 binary, 8 decimal, 0x8

Simplorer -- 10000 binary, 16 decimal, 0x10

MaxwellCircuit -- 100000 binary, 32 decimal, 0x20

<ModifiedOnInfo>:

An integer that corresponds to the number of seconds that have elapsed since 00:00 hours, Jan 1, 1970 UTC from the system clock.

<TerminalInfo>:

Array(<string>, // symbol pin
<string> // footprint pin
<string >, // gate name
<bool>, // shared
<int>, // equivalence number
<int>, // what to do if unconnected: flag as error:0, ignore:1
<string> // description
<Nature>)

<Nature>:

<string> // content varies as follows

Nexxim/Designer:
"Electrical" // the only choice

Simplorer:
// several choices
"Electrical", "Magnetic", "Fluidic", "Translational",
"Translational_V", "Rotational", "Rotational_V",
""Radiant", "Thermal", or <VHDLPackageName>

<VHDLPackageName>:
<string> // in the form <Library>.<Package>

<Library>:
<string> // name of the VHDL library

<Package>:
<string> // name of the VHDL package

<VariableInfo>:
Array(<string>, // name
<FlagLetters>,
<string>, // description
"CB:=", <string>, // optional - script for call back
<string>) // value: number, variable, or expression

<FlagLetters>:
<string> // "D" - has description parameter,
// "RD" - readonly & has description parameter,
// or "RHD" - readonly, hidden, & has description parameter

<CheckBoxInfo>:
Array(<string>, // name

```
<FlagLetters>,
<string>,                 // description
"CB:=", <string>,     // optional - script for call back
<bool>)                   // value: true or false

<ButtonInfo>:
Array(<string>,                  //  name
<FlagLetters>,
<string>,                 // description
<string>,                // button title
<string>,                // extra text
<ClientID>,
"ButtonPropClientData:= ", <ClientDataArray>)

<ClientID>:
<int>    // specifies Button Prop Client
// 0 - unknown, ButtonPropClientData
//       array will be empty
// 1 - Netlist Prop Client
// 2 - not used
// 3 - File Name Prop Client

<ClientDataArray>:
varies with <ClientID>

<ClientId> is 0 or 1: empty array
Array()

<ClientID> is  3:
Array("InternalFormatText:=",  "<prefix><RelativePath>")

<prefix>:
<string>  // "<Project>", "<PersonalLib>", "<UserLib>", or "<SysLib>"

<RelativePath>:
```

```
<string>  // relative path to file from <prefix>

<TextInfo>:
Array(<string>,                 //  name
<FlagLetters>,
<string>,                 // description
"CB:=", <string>,     // optional - script for call back
<string>)                 // value: a text string

<NumberInfo>:
Array(<string>,                 //  name
<FlagLetters>,
<string>,                 // description
"CB:=", <string>,     // optional - script for call back
<real>,                 // value: a number
<string>)                // units

<SeparatorInfo>:
Array(<string>,                 //  name
<FlagLetters>,
<string>,                 // description
"CB:=", <string>,     // optional - script for call back
<string>)                 // value: a text string

<ValueInfo>:
Array(<string>,                 //  name
<FlagLetters>,
<string>,                 // description
"CB:=", <string>,     // optional - script for call back
<string>)                 // value: a number, variable or expression

<MenuPropInfo>:
Array(<string>,                 // name
<FlagLetters>,
<string>,                 // description
```

```
<string>,                 // menu choices - separated by commas
<int>)                    // 0 based index of current menu choice

<VPointInfo>:
Array(<string>,                 //  name
<FlagLetters>,
<string>,                 // description
"CB:=", <string>,     // optional - script for call back
<string>,                 // x value: number with length units
<string>)                 // y value: number with length units

<PointInfo>:
Array(<string>,                 //  name
<FlagLetters>,
<string>,                 // description
"CB:=", <string>,     // optional - script for call back
<real>,                   // x value
<real>)                   // y value

<QuantityPropInfo>:
Array(<string>,                  //  name
<FlagLetters>,
<string>,                 // description
<string>,                 // value
<TypeString>,
<TypeStringDependentInfo>)

<TypeString>:
<string>  //  "Across", "Through", or "Free"

<TypeStringDependentInfo>:

<TypeString> is "Free" :
<string>,   //  direction: "In", "Out", "InOut", or "DontCare"
// Following <string> is not present if direction is "DontCare"
```

<string> // when to calculate: "BeforeAnalogSolver",
// "BeforeStateGraph", "AfterStateGraph", or "DontCareWhen"

<TypeString> is "Across" or "Through":
<int>, // terminal 1
<int> // terminal 2

<CosimDefInfo>:
Array("NAME:CosimDefinition",
"CosimulatorType:=", <int>,
"CosimDefName:=", <string> // "Planar EM", "Designer",
// "Custom", or "Netlist"
"IsDefinition:=", <bool>,
final array member(s) vary with CosimDefName)

final array members for Planar EM:
"CosimStackup:=", <string>,
"CosimDmbedRatio:=", <int>

final array members forDesigner:
"ExportAsNport:=", <int>,
"UsePjt:=", <int>

final array member for Custom:
"DefinitionCompName:=", <string>

final array member for Netlist:
"NetlistString:=", <string>

Example:

```
Dim name
oComponentManager.Add (Array("NAME:MyComponent", _
"Info:=", Array("Type:=", 4294901767, _
"NumTerminals:=", 2, _
"DataSource:=", "", _
```

```
"ModifiedOn:=", 1071096503, _
"Manufacturer:=", "Ansys", _
"Symbol:=", "bendo", _
"Footprint:=", "BENDO", _
"Description:=", "",_
"InfoTopic:=", "", _
"InfoHelpFile:=", "", _
"IconFile:=", "", _
"LibraryName:=", "", _
"OriginalLocation:=", "Project", _
"Author:=", "", _
"OriginalAuthor:=", "", _
"CreationDate:= ", 1147460679), _
"Refbase:=", "U", _
"NumParts:=", 1, _
"OriginalComponent:=", "", _
"Terminal:=", Array("n1", _
"n1", _
"A", _
false, _
0, _
1, _
"", _
"Electrical"), _
"Terminal:=", Array("n2", _
"n2", _
"A", _
false, _
1, _
0, _
"", _
"Electrical"), _
Array("NAME:Parameters", _
"MenuProp:=", Array("CoSimulator", _
"D", _
```

```
"", _
"Default,Planar EM,Designer,Custom,Netlist", _
0), _
"ButtonProp:=", Array("CosimDefinition", _
"D", _
"", _
"", _
"Edit", _
0, _
"ButtonPropClientData:=", Array())),  _
Array("NAME:CosimDefinitions", _
Array("NAME:CosimDefinition", _
"CosimulatorType:=",  0,  _
"CosimDefName:=", "Planar EM", _
"IsDefinition:=", true,  _
"CosimStackup:=",  "Layout stackup",  _
"CosimDmbedRatio:=", 3),  _
Array("NAME:CosimDefinition",  _
"CosimulatorType:=",  1, _
"CosimDefName:=", "Designer", _
"IsDefinition:=", true, _
"ExportAsNport:=", 0, _
"UsePjt:=",  0),  _
Array("NAME:CosimDefinition", _
"CosimulatorType:=", 2, _
"CosimDefName:=", "Custom", _
"IsDefinition:=", true,  _
"DefinitionCompName:=", ""),  _
Array("NAME:CosimDefinition",  _
"CosimulatorType:=",  3, _
"CosimDefName:=", "Netlist", _
"IsDefinition:=", true, _
"NetlistString:=", ""))))
```

AddDynamicNPortData [component manager]

Use: Adds a component using the specified data

Command: Project Menu > Add Model > Add 2DExtractor Model
Project Menu > Add Model > Add HFSS Model
Project Menu > Add Model > Add Nexsys Matlab Model
Project Menu > Add Model > Add Nport Model
Project Menu > Add Model > Add Parametric Model
Project Menu > Add Model > Add Q3D Model
Project Menu > Add Model > Add SIwave Model
Project Menu > Add Model > Add State-space Model

Syntax:
```
AddDynamicNPortData Array("NAME:<ComponentDataName>",
"ComponentDataType:=", <DataType>,
"name:=", <string>,              // Name of the item
"filename:=", <string>,          // Path to the file to find the data>
"numberofports:=", <int>,
"DesignName:=", <string>,    // Name of the internal design
"SolutionName:=", <string>,  // Name of the solution to reference
"Simulate:=", <bool>,
"CloseProject:=", <bool>,
"SaveProject:=", <bool>,
"RefNode:=", <bool>,
"InterpY:=",  <bool>,          // true to choose interpolating
"InterpAlg:=", <InterpolationAlgorithm>,
"NewToOldMap:=", <NewToOldMapPairs>,
"OldToNewMap:=", <OldToNewMapPairs>,
"PinNames:=", Array(<string>, <string>...))
```

Return Value: <string>
// composite name of the component.
// If the name requested conflicts with the name of an existing
// component, the requested name is altered to be unique.
// The name returned reflects any change made to be unique.

Parameters:
<ComponentDataName>:
<string> // simple name of the component

<DataType>:

<string> // "DesignerData" or "Q3DData"

<InterpolationAlgorithm>:

<string> // "auto", "lin", "shadH", or "shadNH"

<NewToOldMapPairs>:

Array of name/value pairs such as "V1", "V2" that will have the new variable name first and the old variable name second.

<OldToNewMapPairs>:

Array of name/value pairs such as "V1", "V2" that will have the old variable name first and the new variable name second.

Example:

```
oComponentManager.AddDynamicNPortData
Array("NAME:ComponentData", _
"ComponentDataType:=",  "DesignerData", _
"name:=", "DesignerData", _
"filename:=",  _
"C:/Program Files/AnsysEM/Designer/Examples/Projects/
optiguides/optiguides.adsn",  _
"numberofports:=",  2, _
"DesignName:=", "DesignerModel1", _
"SolutionName:=", "Setup1 : Adaptive_1",  _
"Simulate:=",  true, _
"CloseProject:=", false, _
"SaveProject:=", true,  _
"RefNode:=", false, _
"InterpY:=",  true, _
"InterpAlg:=", "auto", _
"NewToOldMap:=", Array("nport_height:=", "$height", _
"nport_length:=",  "$length", _
"nport_width:=", "$width"),  _
"OldToNewMap:=", Array("$height:=", "nport_height", _
"$length:=", "nport_length", _
"$width:=", "nport_width"), _
"PinNames:=", Array( "WavePort1:1", "WavePort2:1"))
```

AddNPortData [component manager]

Use: Adds a component using the specified data

Command: Project Menu > Add Model > Add Nport Model

Syntax:

```
AddNPortData Array("NAME:<ComponentDataName>",
"ComponentDataType:=", "NportData",
"name:=", <string>,              //  Name of the item
"filename:=", <string>,          //  Path to the file to find the data
"numberofports:=", <int>,
"filelocation:=", <LocationType>,
"domain:=", <string>,            // "time" or "frequency"
"datamode:=", <string>           //  "EnterData", "Import", or "Link"
"devicename:=", <string>,
"ImpedanceTab:=", <bool>,
"NoiseDataTab:=", <bool>,
"DCBehaviorTab:=", <bool>,
"SolutionName:=", <string>,
"displayformat:=", <DisplayInfo>,
"datatype:=", <string>,          //  "SMatrix", "YMatrix", or "ZMatrix"
"ShowRefPin:=", <bool>,
"RefNodeCheckbox:=", <bool>, ...
<ProductOptionsInfo>)
```

Return Value:

```
<string>
// composite name of the component.
// If the name requested conflicts with the name of  an existing
// component, the requested name is altered to be unique.
//  The name returned reflects any change made to be unique.
```

Parameters:

```
<ComponentDataName>:
<string>  // simple name of the component

<LocationType>:
<string> //  one of "UsePath", "PersonalLib", "UserLib", "SysLib",
// or "Project".
```

```
<dcInfo>:
<string> // one of "DCOpen", "DCShort", "DCShShort",
// "DCNone", or "DCEmpty".

<DisplayInfo>:
<string> // one of "MagnitudePhase", "RealImaginary",
// or "DbPhase".

<ProductOptionsInfo>:
// The remaining parameters differ by product

//  Planar EM - doesn't support interpolation/DC behavior
"DCOption:=", -1,
"InterpOption:=",  -1,
"ExtrapOption:=",-1,
"DataType:=", 0

// Nexxim
"DCOption:=", <NexximDCOption>,
"InterpOption:=",  <NexximInterpOption>,
"ExtrapOption:=", <NexximExtrapOption>,
"DataType:=", 2

<NexximDCOption>:
<int>   // 0 : Zero Padding
// 1 : Same as last point
// 2 :  Linear extrapolation from last 2 points
// 3 : Constant magnitude, linear phase extrapolation
// 4 : Leave all signal lines open circuited
// 5 : Short all signal lines together
// 6 : Short all signal lines to ground

<NexximInterpOption>:
<int>   // 0 : Step
// 1 : Linear
```

<NexximExtrapOption>:
<int> // 0 : Zero padding
// 1 : Same as last point
// 2 : Linear extrapolation from last 2 points
// 3 : Constant magnitude, linear phase extrapolation

<CircuitDCOption>:
<int> // 0 : Leave all signal lines open circuited
// 1 : Short all signal lines together
// 2 : Short all signal lines to ground
// 3 : Extrapolate from data provided (not recommended)

<CircuitInterpOption>:
<int> // 0 : Linear
// 1 : Cubic spline
// 2 : Rational polynomial

<CircuitExtrapOption>:
<int> // 0 : Same as interpolation
// 1 : Zero padding
// 2 : Same as last point

Example:

```
oComponentManager.AddNPortData Array("NAME:Component-
Data", _
"ComponentDataType:=",  "NPortData", _
"name:=", "NportData", _
"filename:=", "", _
"numberofports:=", 2, _
"filelocation:=",  "UsePath", _
"domain:=", "frequency", _
"datamode:=", "Import",  _
"devicename:=", "", _
"ImpedenceTab:=",  true,  _
"NoiseDataTab:=", true, _
```

```
"DCBehaviorTab:=", true,  _
"SolutionName:=", "",  _
"displayformat:=", "MagnitudePhase", _
"datatype:=", "SMatrix", _
"ShowRefPin:=", true, _
"RefNodeCheckbox:=", false, _
"DCOption:=", 3, _
"InterpOption:=", 1, _
"ExtrapOption:=", 3, _
"DataType:=",  2,  _
"DCOption:=", 3, _
"InterpOption:=", 1, _
"ExtrapOption:=", 3, _
"DataType:=", 2)
```

Edit [[component manager]

Use: Modifies an existing component

Command: Tools > Edit Configured Libraries > Components > Edit Component

Syntax:

```
Edit <ComponentName>,
Array("NAME:<NewComponentName>",
"Info:=",  <ComponentInfo>,
"RefBase:=", <string>,              // reference designator
"NumParts:=", <int>,                // parts per component
"OriginalComponent:=", <string>
"Terminal:=", <TerminalInfo>,
"Terminal:=", <TerminalInfo>, ...
// The remaining parameters are optional
Array("NAME:Parameters",    // any combo of the following
"VariableProp:=", <VariableInfo>,
"CheckboxProp:=", <CheckBoxInfo>,
"ButtonProp:=", <ButtonInfo>,
"TextProp:=", <TextInfo>,
"NumberProp:=", <NumberInfo>,
"SeparatorProp:=", <SeparatorInfo>,
"ValueProp:=", <ValueInfo>,
```

"MenuProp:=", <MenuInfo>),
Array("NAME:Properties", // any combo of the following
"CheckboxProp:=", <CheckBoxInfo>,
"TextProp:=", <TextInfo>,
"NumberProp:=", <NumberInfo>,
"SeparatorProp:=", <SeparatorInfo>,
"ValueProp:=", <ValueInfo>,
"MenuProp:=", <MenuInfo>),
"VPointProp:=", <VPointInfo>,
"PointProp:=", <PointInfo>),
Array("Quantities",
"QuantityProp:=", <QuantityPropInfo>...),
Array("NAME:CosimDefinitions",
<CosimDefInfo>,
<CosimDefInfo>...)

Return Value: <string>
// composite name of the component.
// If the name requested conflicts with the name of an existing
// component, the requested name is altered to be unique.
// The name returned reflects any change made to be unique.

Parameters: <ComponentName>:
<string> // composite name of the component to edit

<NewComponentName>:
<string> // new simple name for the component

<ComponentInfo>:
Array("Type:=", <TypeInfo>,
"NumTerminals:=", <int>,
"DataSource:=", <string>,
"ModifiedOn:=", <ModifiedOnInfo>,
"Manufacturer:=", "<string>,
"Symbol:=", <string>,
"Footprint:=", <string>,

"Description:=", <string>,
"InfoTopic:=", <string>,
"InfoHelpFile:=", <string>,
"IconFile:=", <string>,
"LibraryName:=", <string>,
"OriginalLocation:=", <string>, // Project Location
"Author:=", <string>,
"OriginalAuthor:=", <string>,
"CreationDate:= ", <int>)

<TypeInfo>:

An integer that is the or-ing of bits for each product listed below. The default setting is 0xffffffff (4294967295) which indicates valid for all products. In the component editing dialog, checking different boxes in the "Specify products for which this component is valid" grid control sets specific flags that correspond to the following hex/decimal settings:

Nexxim -- 100 binary, 4 decimal, 0x4

SIwaveDeNovo -- 1000 binary, 8 decimal, 0x8

Simplorer -- 10000 binary, 16 decimal, 0x10

MaxwellCircuit -- 100000 binary, 32 decimal, 0x20

<ModifiedOnInfo>:

An integer that corresponds to the number of seconds that have elapsed since 00:00 hours, Jan 1, 1970 UTC from the system clock.

<TerminalInfo>:

Array(<string>, // symbol pin
<string> // footprint pin
<string >, // gate name
<bool>, // shared
<int>, // equivalence number
<int>, // what to do if unconnected: flag as error:0, ignore:1
<string>, // description
<Nature>)

<Nature>:

<string> // content varies as follows

Nexxim/Designer:
"Electrical" // the only choice

Simplorer:
// several choices
"Electrical", "Magnetic", "Fluidic", "Translational",
"Translational_V", "Rotational", "Rotational_V",
""Radiant", "Thermal", or <VHDLPackageName>

<VHDLPackageName>:
<string> // in the form <Library>.<Package>

<Library>:
<string> // name of the VHDL library

<Package>:
<string> // name of the VHDL package

<VariableInfo>:
Array(<string>, // name
<FlagLetters>,
<string>, // description
"CB:=", <string>, // optional - script for call back
<string>) // value: number, variable, or expression

<FlagLetters>:
<string> // "D" - has description parameter,
// "RD" - readonly & has description parameter,
// or "RHD" - readonly, hidden, & has description parameter

<CheckBoxInfo>:
Array(<string>, // name

```
<FlagLetters>,
<string>,                 // description
"CB:=", <string>,     // optional - script for call back
<bool>)                   // value: true or false

<ButtonInfo>:
Array(<string>,                  //  name
<FlagLetters>,
<string>,                 // description
<string>,                // button title
<string>,                // extra text
<ClientID>,
"ButtonPropClientData:= ", <ClientDataArray>)

<ClientID>:
<int>    // specifies Button Prop Client
// 0 - unknown, "ButtonPropClientData
//       array will be empty
// 1 - Netlist Prop Client
// 2 - not used
// 3 - File Name Prop Client

<ClientDataArray>:
varies with <ClientID>

<ClientID> is 0 or 1: empty array
Array()

<ClienID> is 3:
Array("InternalFormatText:=",  "<prefix><RelativePath>")

<prefix>:
<string>  // "<Project>", "<PersonalLib>", "<UserLib>", or "<SysLib>"

<RelativePath>:
```

```
<string>  // relative path to file from <prefix>

<TextInfo>:
Array(<string>,                 //  name
<FlagLetters>,
<string>,                 // description
"CB:=", <string>,     // optional - script for call back
<string>)                 // value: a text string

<NumberInfo>:
Array(<string>,                 //  name
<FlagLetters>,
<string>,                 // description
"CB:=", <string>,     // optional - script for call back
<real>,                 // value: a number
<string>)               // units

<SeparatorInfo>:
Array(<string>,                 //  name
<FlagLetters>,
<string>,                 // description
"CB:=", <string>,     // optional - script for call back
<string>)                 // value: a text string

<ValueInfo>:
Array(<string>,                 //  name
<FlagLetters>,
<string>,                 // description
"CB:=", <string>,     // optional - script for call back
<string>)                 // value: a number, variable or expression

<MenuPropInfo>:
Array(<string>,                 // name
<FlagLetters>,
<string>,                 // description
```

```
<string>,                 // menu choices - separated by commas
<int>)                    // 0 based index of current menu choice

<VPointInfo>:
Array(<string>,                  //  name
<FlagLetters>,
<string>,                 // description
"CB:=", <string>,         // optional - script for call back
<string>,                 // x value: number with length units
<string>)                 // y value: number with length units

<PointInfo>:
Array(<string>,                  //  name
<FlagLetters>,
<string>,                 // description
"CB:=", <string>,         // optional - script for call back
<real>,                   // x value
<real>)                   // y value

<QuantityPropInfo>:
Array(<string>,                   //  name
<FlagLetters>,
<string>,                 // description
<string>,                 // value
<TypeString>,
<TypeStringDependentInfo>)

<TypeString>:
<string>  //  "Across", "Through", or "Free"

<TypeStringDependentInfo>:

"Free" :
<string>,   //  direction: "In", "Out", "InOut", or "DontCare"
// Following <string> is not present if direction is "DontCare"
```

<string> // when to calculate: "BeforeAnalogSolver",
// "BeforeStateGraph", "AfterStateGraph", or "DontCareWhen"

"Across" or "Through"
<int>, // terminal 1
<int> // terminal 2

<CosimDefInfo>:
Array("NAME:CosimDefinition",
"CosimulatorType:=", <int>,
"CosimDefName:=", <string> // "Planar EM", "Designer",
// "Custom", or "Netlist"
"IsDefinition:=", <bool>,
final array member(s) vary with CosimDefName)

final array members for Planar EM:
"CosimStackup:=", <string>,
"CosimDmbedRatio:=", <int>

final array members forDesigner:
"ExportAsNport:=", <int>,
"UsePjt:=", <int>

final array member for Custom:
"DefinitionCompName:=", <string>

final array member for Netlist:
"NetlistString:=", <string>

Example:

```
Dim name
name = oComponentManager.Edit ("Nexxim Circuit Ele-
ments\BJTs:Level01_NPN", _
Array("NAME:Level01_NPN", _
"Info:=", Array("Type:=", 4294901764,_
"NumTerminals:=", 3, _
```

```
"DataSource:=", "Ansys built-in component", _
"ModifiedOn:=", 1152722112, _
"Manufacturer:=", "", _
"Symbol:=", "nexx_bjt_npn", _
"Footprint:=", "", _
"Description:=", "BJT, GP, NPN", _
"InfoTopic:=", "NXBJT1.htm", _
"InfoHelpFile:=", "nexximcomponents.chm", _
"IconFile:=", "bjtsn.bmp", _
"Library:=", "Nexxim Circuit Elements\BJTs", _
"OriginalLocation:=", "SysLibrary ", _
"Author:=", "", _
"OriginalAuthor:=", "", _
"CreationDate:=", 1152722102), _
"Refbase:=", "Q", _
"NumParts:=", 1, _
"Terminal:=", Array("collector", _
"collector", _
"A", _
false,  _
6, _
0, _
"", _
"Electrical"), _
"Terminal:=", Array("base", _
"base", _
"A", _
false, _
7, _
0, _
"", _
"Electrical"), _
"Terminal:=", Array("emitter", _
"emitter", _
"A", _
```

```
false, _
8, _
0, _
"", _
"Electrical"), _
Array("NAME:Parameters", _
"TextProp:=", Array("LabelID", _
"HD",  _
"Property string for netlist ID", _
"Q@ID"), _
"TextProp:=", Array("MOD", _
"D",  _
"Name of model data reference", _
"required"), _
"VariableProp:=", Array("AREA", _
"D",  _
 "Emitter area multiplying factor, which affects curents,
resistances, and capacitances", _ "1"), _
"VariableProp=", Array("AREAB", _
"D", _
"Base AREA", _
"1"), _
"VariableProp:=", Array( "AREAC", _
"D", _
Collector AREA", _
"1"), _
"VariableProp:=", Array("DTEMP", _
"D",  _
"The difference between element and circuit temperature
deg Cel)", _
"0"), _
"VariableProp:=", Array("M", _
"D", _
"Multiplier factor to simulate multiple BJTs in paralel",
_ "1"), _
"ButtonProp:=", Array("NexximNetlist", _
```

```
"HD", _
"",  _
"Q@ID %0 %1 %2 *MOD(@MOD) *AREA(AREA=@AREA)" & _
" *AREAB(AREAB=@AREAB) *AREAC(AREAC=@" & _
"AREAC) *DTEMP(DTEMP=@DTEMP) *M(M=@M)",  _
"Q@ID %0 %1 %2 *MOD(@MOD) " & _  "*AREA(AREA=@AREA)
AREAB(AREAB=@AREAB) *AREAC(AREAC=@" & _
"AREAC) *DTEMP(DTEMP=@DTEMP) *M(M=@M)", _
1, _
"ButtonPropClientData:=", Array()), _
"TextProp:=", Array( "ModelName", _
"HD", _
"", _
"Q"))))
```

Example:

```
Dim name2
name2 = oComponentManager.Edit "MyComponent", _
(Array("NAME:MyOtherComponent", _
"Info:=", Array("Type:=", 4294901767, _
"NumTerminals:=", 2, _
"DataSource:=", "", _
"ModifiedOn:=", 1071096503, _
"Manufacturer:=", "Ansys", _
"Symbol:=", "bendo", _
"Footprint:=", "BENDO", _
"Description:=", "",_
"InfoTopic:=", "", _
"InfoHelpFile:=", "", _
"IconFile:=", "", _
"LibraryName:=", "", _
"OriginalLocation:=", "Project", _
"Author:=", "", _
"OriginalAuthor:=", "", _
"CreationDate:= ", 1147460679), _
"Refbase:=", "U", _
```

```
"NumParts:=", 1, _
"OriginalComponent:=", "", _
"Terminal:=", Array("n1", _
"n1", _
"A", _
false, _
0, _
0, _
"", _
"Electrical"), _
"Terminal:=", Array("n2", _
"n2", _
"A", _
false, _
1, _
0, _
"", _
Electrical"), _
Array("NAME:Parameters", _
"MenuProp:=", Array("CoSimulator", _
"D", _
"", _
"Default,Planar EM,Designer,Custom,Netlist", _
0), _
"ButtonProp:=", Array("CosimDefinition", _
"D", _
"", _
"", _
"Edit", _
0, _
"ButtonPropClientData:=", Array())),  _
Array("NAME:CosimDefinitions", _
Array("NAME:CosimDefinition", _
"CosimulatorType:=",  0,  _
"CosimDefName:=", "Planar EM", _
```

```
"IsDefinition:=", true,  _
"CosimStackup:=",  "Layout stackup",  _
"CosimDmbedRatio:=", 3),  _
Array("NAME:CosimDefinition",  _
"CosimulatorType:=",  1, _
"CosimDefName:=", "Designer", _
"IsDefinition:=", true, _
"ExportAsNport:=", 0, _
"UsePjt:=",  0),  _
Array("NAME:CosimDefinition", _
"CosimulatorType:=", 2, _
"CosimDefName:=", "Custom", _
"IsDefinition:=", true,  _
"DefinitionCompName:=", ""),  _
Array("NAME:CosimDefinition",  _
"CosimulatorType:=",  3, _
"CosimDefName:=", "Netlist", _
"IsDefinition:=", true, _
"NetlistString:=", "")))
```

EditWithComps [component manager]

Use: Edit an existing component.

Command: None

Syntax:

```
EditWithComps <ComponentName>,
Array("NAME:<NewComponentName>",
"ModTime:=", <ModifiedTimeInfo>,
"Library:=", <string>,          // Library name
"LibLocation:=", <string>,    // Project Location
<PinDefInfo>,
<PinDefInfo>,...                // optional, to define pins
<GraphicsDataInfo>, // optional, to define graphics
<PropDisplayMapInfo>), // optional, to define property displays
Array(<ListOfComponentNames>) // Component names
```

Return Value: <string>

// composite name of the component.
// If the name requested conflicts with the name of an existing
// component, the requested name is altered to be unique.
// The name returned reflects any change made to be unique.

Parameters:

<ComponentName>:
<string> // composite name of the component being edited

<NewComponentName>:
<string> // new simple name for the component

<ModifiedOnInfo>:
An integer that corresponds to the number of seconds that have elapsed since 00:00 hours, Jan 1, 1970 UTC from the system clock.

<PinDefInfo>:
Array("NAME:PinDef",
"Pin:=", Array (<string>, // pin name
<real>, // x location
<real>, // y location
<real>, // angle in radians
<PinType>,
<real>, // line width
<real>, // line length
<bool>, // mirrored
<int>, // color
<bool>, // true if visible, false if not
<string>, // hidden net name
<OptionalPinInfo>, // optional info
<PropDisplayMapInfo>)) // optional

<PinType>:
<string> // "N" : normal pin
// "I" : input pin
// "O" : output pin

```
<OptionalPinInfo>:
// Specify both or neither
<bool>,    // true if name is to be shown
<bool>,    // true if number is to be shown

<PropDisplayMapInfo>:
                Array("NAME:PropDisplayMap",
                      <PropDisplayInfo>,
                      <PropDisplayInfo>,...)

                 <PropDisplayInfo>:
                  <NameString>,  Array(<DisplayTypeInfo>,
                                       <DisplayLocationInfo>,
                                       <int>,     // optional, level number
                                       <TextInfo>)

<NameString>:
                   <string>   // PropertyName:=,  where PropertyName is the name of
                              // the property to be displayed

                  <DisplayTypeInfo>:
                  <int>    // 0 :  No display
                           // 1 :  Display name only
                           // 2 :  Display value only
                           // 3 :  Display both name and value
                           //  4:  Display evaluated value only
                           //  5:  Display both name and evaluated value

                   <DisplayLocationInfo>:
                   <int>    // 0 :  Left
                            // 1 :  Top
                            // 2 :  Right
                            // 3 :  Bottom
                            // 4 :  Center
```

```
                                    // 5 :  Custom placement

                                        <GraphicsDataInfo>:
                                 Array("NAME:Graphics",
                                        // one or more of the following
                                       <RectInfo>,
                                        <CircleInfo>,
                                         <ArcInfo>,
                                          <LineInfo>,
                                          <PolygonInfo>,
                                          <TextInfo>,
                                         <ImageInfo>)

                                <RectInfo>:
                                 "Rect:=", Array(<real>,    // line width
                                                 <int>,     //  fill pattern
                                                 <int>,     // color
                                                 <real>,    // angle, in radians
                                                 <real>,    // x position of center
                                                 <real>,    // y position of center
                                                          <real>,   // width
                                                          < real>)  // height

                                 <CircleInfo>:
                                  "Circle:=", Array(<real>,    // line width
                                                    <int>,     //  fill pattern
                                                    <int>,     // color
                                                    <real>,    // x position of center
                                                    <real>,    // y position of center
                                                             < real>)   // radius

                               <ArcInfo>:
                                "Arc:=", Array(<real>,    // line width
                                               <int>,     //  line pattern
                                               <int>,     // color
```

```
                    <real>,    // x position of center
                    <real>,    // y position of center
                                < real>,    // radius
                                 <real>,    // start angle, in radians
                                 <end>)    // end angle, in radians

<LineInfo>:
  "Line:=", Array(<real>,    // line width
                    <int>,     //  line pattern
                    <int>,     // color
                                    <PointInfo>,    // must specify at least 2 points
                                    <PointInfo>...)
                <PointInfo>:
                 <real>,    // x position
                 <real>     // y position

<PolygonInfo>:
  "Polygon:=", Array(<real>,    // line width
                        <int>,     //  fill pattern
                        <int>,     // color
                                    <PointInfo>,     // must specify at least 3 points
                                    <PointInfo>...)

<TextInfo>:
  "Text:=", Array(<real>,    // x position
                    <real>,   // y position
                    <real>,   // angle, in radians
                    <Justification>,
                     <bool>,    // is plotter font
                     <string>,    // font name
                      <int>,       // color
                      <string>)    // text string

                <Justification>:
                <int>  // 0 : left top
```

// 1 : left base
// 2 : left bottom
// 3 : center top
// 4 : center base
// 5 : center bottom
// 6 : right top
// 7 : right base
// 8 : right bottom

<ImageInfo>:
"Image:=", Array(<RectInfo>,
<ImageData>,
<bool>) // is mirrored

<ImageData>:
<string>, // file path
<int>, // 0 : use the file path and link to it
// 1 : ignore file path and use next parameter
<string> // text data, only present if preceding int is 1

<ListOfComponentNames>:
<string>,<string> ...
// The list may be empty. When not empty, each string that is listed is a component
// that references the component to be edited. Prior to editing, a clone of the component is
// made, and the components that are listed are modified so that they now refer to
// the clone.

Example:

```
Dim nam
oModelManager.EditWithComps_
("Nexxim Circuit Elements\Distributed\Distributed:bendo",
_
Array("NAME:bendo new name", _
"ModTime:=", 1152722165, _
```

```
"Library:=", "Nexxim Circuit Elements\Distributed\Dis-
tributed", _
"LibLocation:=", "SysLibrary", _
Array("NAME:PinDef", _
"Pin:=", Array( "n1", _
-0.00254, _
0.00254,  _
0, _
"N", _
0, _
0, _
false, _
0, _
true, _
"",  _
false, _
false)),  _
Array("NAME:PinDef", _
"Pin:=", Array( "n2", _
0.00254, _
-0.00254, _
1.5708, _
"N", _
0, _
0, _
false, _
0, _
true, _
"", _
false, _
false)),  _
Array("NAME:Graphics", _
"Polygon:=", Array( 0, 0, 12566272, 0, 0.00381, 0,
.00127, 0.00127, _
0.00127, 0.00127, 0, 0.00381, 0, 0.00381,  _
0.002032, 0.00127, 0.00381), _
```

```
"Line:=", Array(0, 1, 12566272,  -0.00254, 0.00254, 0,
.00254),  _
"Line:=", Array(0, 1, 12566272, 0.00254, -0.00254,
.00254, 0)),  _
Array("NAME:PropDisplayMap", _
"W:=", Array(3, _
5, _
0, _
"Text:=", Array( 2.1684E-019, _
0.00504119, _
0, _
1, _
5, _
false, _
"Arial", _
0, _
"W=***"))))),  _
Array("MY_COMP")
```

Export [component manager]

Use: Export component(s) to a library

Command: Tools > Edit Configured Libraries > Components > Export to Library

Syntax:
`Export Array("NAME:<LibraryName>",`
<ComponentName>,
<ComponentName>...),
<LibraryLocation>

Return Value: None

Parameters:
`<LibraryName>:`
<string> // name of the library

<ComponentName>:
<string> // composite name of the component to export

<LibraryLocation>:

<string> // location of the library in <LibraryName>
// One of "Project", "PersonalLib", or "UserLib"

Example:

```
oComponentManager.Export  Array("NAME:mylib", "Nexxim
Circuit Elements\BJTs:Level01_NPN"),"PersonalLib"
```

GetData [component manager]

Use: Gets data that describes the definition.

Command: None

Syntax: `GetData(<DefinitionName>)`

Return Value: <DefinitionData> This is an array of data for the definition.

Parameters: `<DefinitionName>:`

<string> // composite name of the definition to edit

Example:

```
Dim compData
compData = oComponentManager.GetData("Level01_NPN")
```

Note GetData allows the user to access definition information, make modifications, and then use the Edit or EditWithComps script commands to save the modified definition. Accordingly, for each type of definition, the array data returned to GetData should be the same array information that is supplied to the Edit or EditWithComps commands.

GetNames [component manager]

Use: Returns the names of the components (used and unused) in a design. The following script command, **IsUsed,** can then be used to separate used and unused components.

Command: None

Syntax: `GetNames()`

Return Value: An array of strings

Parameters: `None`

Example:

```
Dim componentNames
```

```
componentNames = oComponentManager.GetNames()
```

GetNPortData [component manager]

Use: Returns NPort data for the component with the specified name.

Command: None

Return Value: Variant array, whose contents depend on the type of component. The array will be empty if the component does not have NPort data. See the syntax for AddDynamicNPortData and AddNPortData for descriptions of the array contents for components with those types of NPort data.

Parameters:

`<ComponentName>:`

<string>

Example: This script displays each item in the returned array for each component in the design.

```
Sub DisplayVariant(x)
Dim index
 index = 0

 For Each info In x
    curr = x(index)
    If TypeName(curr) <> "Variant()" Then
       If TypeName(curr) <> "Empty" And  TypeName(curr)
<> "Null" Then
          str = CStr(curr)
          If str = "" Then
                str = ChrW(34) & ChrW(34)
          End If
          Msgbox str
       Else
          str = "Empty/Null item."
          Msgbox str
        End If
    Else
        DisplayVariant curr
    End If
```

```
    index = index + 1
Next

End Sub


Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim dnpInfo
Dim index

Set oAnsoftApp = CreateObject("AnsysDesigner.Design-
erScript")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.GetActiveProject()
Set oDefinitionManager = oProject.GetDefinitionManager()

Set oComponentManager = oDefinitionManager.GetMan-
ager("Component")

Dim componentNames
componentNames = oComponentManager.GetNames()

index = 0
For Each name In componentNames
    name = componentNames(index)
    message = "NPort data for component " + name
    Msgbox message
   dnpInfo = oComponentManager.GetNPortData(name)

    DisplayVariant dnpInfo
    index = index + 1
Next
```

IsUsed [component manager]

Use: Used to determine if a component is used in the design.

Command: None

Syntax: `IsUsed(<ComponentName>)`

Return Value: <Boolean> // true if the specified component is used in the design

Parameters:

`<ComponentName>:`
<string>

Example:

```
Dim isUsed
isUsed = oComponentManager.IsUsed("MyComponent")
```

Remove [component manager]

Use: Remove a component from a library

Command: Tools > Edit Configured Libraries > Components > Remove Component

Syntax:

```
Remove <ComponentName>,
<IsProjectComponent>,
<LibraryName>,
<LibraryLocation>
```

Return Value: None

Parameters:

`<ComponentName>:`
<string> // composite name of the component to remove

<IsProjectComponent>:
<bool>

<LibraryName>:
<string> // name of the library

<LibraryLocation>:
<string> // location of the library in <LibraryName>
// One of "Project", "PersonalLib", or "UserLib"

Example:

```
oComponentManager.Remove "Nexxim Circuit Ele-
ments\BJTs:Level01_NPN", _ true, "Project"
```

RemoveUnused [component manager]

Use: Removes components that are not used in the design.

Command: **Project->Remove Unused Definitions** is similar but operates slightly different and does not record script commands.

Syntax: `RemoveUnused()`

Return Value: <bool> True if one or more components are removed.

Parameters: `None`

Example:

```
Dim removedDefs
removedDefs = oComponentManager.RemoveUnused()
```

Note The order of calls to RemoveUnused is significant. As a result, removing definitions in an unordered fashion may cause other components in dependent definitions to be rendered unusable.

Also, the symbol and footprint of an unused component are not unusable until after the component itself is removed using the Component Manager Remove script.

Component Manager SOD Script Commands

The component manager Solver On Demand (SOD) scripts provide access to components in a Designer project. The manager object is accessed via the definition manager.

```
Set oDefinitionManager = oProject.GetDefinitionManager()
Set oComponentManager = oDefinitionManager.GetManager("Compo-
nent")
```

The component manager SOD script commands are listed below:

AddAddSolverOnDemandModel

EditSolverOnDemandModel

GetSolverOnDemandData

GetSolverOnDemandModelList

RemoveSolverOnDemandModel

AddAddSolverOnDemandModel

Use: This method looks for a local component of the name passed in, and to this component it adds an SOD model definition using the information passed in the VARIANT. It returns the name of the SOD model added.

Parameters: `BSTR component name.`

Parameters: `VARIANT which is the SOD model data.`

Return Value: Returns the name of the model added.

EditSolverOnDemandModel

Use: This method looks for a local component of the name passed in, and in this component it looks for an SOD model using the name passed in the second BSTR. It modifies the SOD model using the data in the VARIANT. It returns the name of the SOD model edited.

Parameters: `BSTR component name.`

Parameters: `BSTR SOD model name.`

Parameters: `VARIANT which is the new SOD model data (can include changed name).`

Return Value: Returns the name of the model edited.

GetSolverOnDemandData

Use: This method looks for a local component of the name passed in, and in this component it looks for an SOD model of the name passed in and returns the SOD data pertaining to that model.

Parameters: `BSTR component name.`

Parameters: `BSTR SOD model name`

Return Value: VARIANT which is the SOD data.

GetSolverOnDemandModelList

Use: This method looks for a local component of the name passed in, and returns a list of SOD model names defined in the component.

Parameters: `BSTR component name.`

Return Value: VARIANT which is a list of SOD model names.

RemoveSolverOnDemandModel

Use: This method looks for a local component of the name passed in, and in this component it looks for an SOD model of the name passed in and deletes the SOD model definition from the component.

Parameters: `BSTR component name.`

Parameters: `BSTR SOD model name`

Return Value: None.

Model Manager Script Commands

The model manager provides access to models in a Designer project. The manager object is accessed via the definition manager.

```
Set oDefinitionManager = oProject.GetDefinitionManager()
Set oModelManager = oDefinitionManager.GetManager("Model")
```

The model manager script commands are listed below:

Add
ConvertToDynamic
ConvertToParametric
Edit
EditWithComps
Export
GetData
GetNames
IsUsed
Remove
RemoveUnused

Add [model manager]

Use: Add a model

Command: Tools > Edit Configured Libraries > Models > Add Model

Syntax:

```
Add Array("NAME:<modelName>",
"ModTime:=", <ModifiedTimeInfo>,
"Library:=", "",          // Library name
"LibLocation:=", "Project",    // Project Location
<PinDefInfo>,
<PinDefInfo>,...              // optional, to define pins
```

<GraphicsDataInfo>, // optional, to define graphics
<PropDisplayMapInfo>)) // optional, to define property displays

Return Value: <string>
// composite name of the model.
// If the name requested conflicts with the name of an existing
// model, the requested name is altered to be unique.
// The name returned reflects any change made to be unique.

Parameters: <modelName>:
<string> // simple name of the model being added

<ModifiedOnInfo>:
An integer that corresponds to the number of seconds that have elapsed since 00:00 hours, Jan 1, 1970 UTC from the system clock.

<PinDefInfo>:
Array("NAME:PinDef",
"Pin:=", Array (<string>, // pin name
<real>, // x location
<real>, // y location
<real>, // angle in radians
<PinType>,
<real>, // line width
<real>, // line length
<bool>, // mirrored
<int>, // color
<bool>, // true if visible, false if not
<string>, // hidden net name
<OptionalPinInfo>, // optional info
<PropDisplayMapInfo>)) // optional

<PinType>:
<string> // "N" : normal pin
// "I" : input pin

```
                    // "O" : output pin

<OptionalPinInfo>:
// Specify both or neither
<bool>,    // true if name is to be shown
<bool>,    // true if number is to be shown


<PropDisplayMapInfo>:
                Array("NAME:PropDisplayMap",
                      <PropDisplayInfo>,
                      <PropDisplayInfo>,...)

                 <PropDisplayInfo>:
                  <NameString>,  Array(<DisplayTypeInfo>,
                                       <DisplayLocationInfo>,
                                        <int>,     // optional, level number
                                        <TextInfo>)
                  <NameString>:
                   <string>   // PropertyName:=,  where PropertyName is the name of
                              // the property to be displayed

                  <DisplayTypeInfo>:
                  <int>    // 0 :  No display
                           // 1 :  Display name only
                           // 2 :  Display value only
                           // 3 :  Display both name and value
                           //  4:  Display evaluated value only
                           //  5:  Display both name and evaluated value

                   <DisplayLocationInfo>:
                   <int>    // 0 :  Left
                            // 1 :  Top
                            // 2 :  Right
                            // 3 :  Bottom
```

```
                                    // 4 :  Center
                                    // 5 :  Custom placement

<GraphicsDataInfo>:
  Array("NAME:Graphics",
            // one or more of the following
           <RectInfo>,
            <CircleInfo>,
             <ArcInfo>,
              <LineInfo>,
              <PolygonInfo>,
              <TextInfo>,
             <ImageInfo>)

 <RectInfo>:
  "Rect:=", Array(<real>,    // line width
                        <int>,     //  fill pattern
                        <int>,     // color
                        <real>,   // angle, in radians
                        <real>,    // x position of center
                        <real>,    // y position of center
                                          <real>,   // width
                                          < real>)  // height

 <CircleInfo>:
  "Circle:=", Array(<real>,    // line width
                          <int>,     //  fill pattern
                          <int>,     // color
                          <real>,    // x position of center
                          <real>,    // y position of center
                                          < real>)   // radius

<ArcInfo>:
  "Arc:=", Array(<real>,    // line width
                       <int>,     //  line pattern
```

```
                    <int>,     // color
                    <real>,    // x position of center
                    <real>,    // y position of center
                                < real>,    // radius
                                 <real>,    // start angle, in radians
                                 <end>)    // end angle, in radians

<LineInfo>:
  "Line:=", Array(<real>,    // line width
                    <int>,     //  line pattern
                    <int>,     // color
                                <PointInfo>,    // must specify at least 2 points
                                <PointInfo>...)
                <PointInfo>:
                 <real>,    // x position
                 <real>     // y position

<PolygonInfo>:
  "Polygon:=", Array(<real>,    // line width
                      <int>,     //  fill pattern
                      <int>,     // color
                                  <PointInfo>,     // must specify at least 3 points
                                  <PointInfo>...)

<TextInfo>:
  "Text:=", Array(<real>,    // x position
                    <real>,   // y position
                    <real>,   // angle, in radians
                    <Justification>,
                     <bool>,    // is plotter font
                     <string>,    // font name
                      <int>,       // color
                      <string>)    // text string

                <Justification>:
```

<int> // 0 : left top
// 1 : left base
// 2 : left bottom
// 3 : center top
// 4 : center base
// 5 : center bottom
// 6 : right top
// 7 : right base
// 8 : right bottom

<ImageInfo>:
"Image:=", Array(<RectInfo>,
<ImageData>,
<bool>) // is mirrored

<ImageData>:
<string>, // file path
<int>, // 0 : use the file path and link to it
// 1 : ignore file path and use next parameter
<string> // text data, only present if preceding int is 1

Example:

```
oModelManager.Add Array("NAME:MyModel",_

"ModTime:=", 1070989137, _

"Library:=", "", _

"LibLocation:=", "Project", _

Array("NAME:PinDef",  _

"Pin:=", Array ("newpin0", _
0.00254, _
0, _
0, _
```

```
"N",_
0, _
0.00254, _
false, _
0, _
true, _
"",_
false, _
false)), _
Array("NAME:PinDef",  _
"Pin:=", Array ("newpin1", _
-0.00254, _
0, _
3.14159265358979, _
"N",_
0, _
0.00254, _
false, _
0, _
true, _
"",_
false, _
false)),
Array("NAME:Graphics", _
"Rect:=", Array(0,  _
0,  _
12566272, _
0,  _
4.33680868994202e-019, _
-0.000635, _
0.00508,  _
0.002794), _
"Circle:=", Array(0, _
0,  _
12566272,  _
```

```
0.000127, _
-0.000635,  _
0.000635)))
```

ConvertToDynamic

Use: Build a new dynamic model based on an existing parametric model.

Command: Right-click on a model under Definitions/Models in the Project Tree and choose ConvertToDynamic.

Syntax: `ConvertToDynamic(defName, newname)`

Return Value: <newname> // Name of the new model added

Parameters: `<defName> // Model that is the base for the new conversion`

Example:

```
Dim newname
newname = oModelManager.ConvertToDynamic([in] BSTR def-
Name,  [out, retval] BSTR* newName);
```

ConvertToParametric

Use: Build a new parametric model based on an existing dynamic model.

Command: Right-click on a model under Definitions/Models in the Project Tree and choose ConvertToParametric.

Syntax: `ConvertToParametric(defName, newname)`

Return Value: <newname> // Name of the new model added

Parameters: `<defName> // Model that is the base for the new conversion`

Example:

```
Dim newname
```

newname = oModelManager.ConvertToParametric([in] BSTR defName, [out, retval] BSTR* newName);

Edit [depricated]

Deprecated command — please use EditWithComps.

EditWithComps [model manager]

Use: Edit an existing model.

Command: None

Syntax: `EditWithComps <ModelName>,`

Array("NAME:<NewModelName>",
"ModTime:=", <ModifiedTimeInfo>,
"Library:=", <string>, // Library name
"LibLocation:=", <string>, // Project Location
<PinDefInfo>,
<PinDefInfo>,... // optional, to define pins
<GraphicsDataInfo>, // optional, to define graphics
<PropDisplayMapInfo>), // optional, to define property displays
Array(<ListOfComponentNames>) // Component names

Return Value: <string>
// composite name of the model.
// If the name requested conflicts with the name of an existing
// model, the requested name is altered to be unique.
// The name returned reflects any change made to be unique.

Parameters: <ModelName>:
<string> // composite name of the model being edited

<NewModelName>:
<string> // new simple name for the model

<ModifiedOnInfo>:
An integer that corresponds to the number of seconds that have elapsed since 00:00 hours, Jan 1, 1970 UTC from the system clock.

<PinDefInfo>:
Array("NAME:PinDef",
"Pin:=", Array (<string>, // pin name
<real>, // x location
<real>, // y location
<real>, // angle in radians
<PinType>,
<real>, // line width
<real>, // line length

```
                                <bool>,  // mirrored
                                <int>,     // color
                                <bool>,   // true if visible, false if not
                                <string>,  // hidden net name
                                <OptionalPinInfo>, // optional info
<PropDisplayMapInfo>))  // optional

<PinType>:
  <string>   // "N" : normal pin
             // "I"  : input pin
             // "O" : output pin

<OptionalPinInfo>:
// Specify both or neither
<bool>,    // true if name is to be shown
<bool>,    // true if number is to be shown

<PropDisplayMapInfo>:
              Array("NAME:PropDisplayMap",
                    <PropDisplayInfo>,
                    <PropDisplayInfo>,...)

               <PropDisplayInfo>:
                <NameString>,  Array(<DisplayTypeInfo>,
                                     <DisplayLocationInfo>,
                                      <int>,     // optional, level number
                                      <TextInfo>)

<NameString>:
                 <string>   // PropertyName:=,  where PropertyName is the name of
                            // the property to be displayed

                <DisplayTypeInfo>:
                <int>    // 0 :  No display
                         // 1 :  Display name only
```

```
                    // 2 :  Display value only
                    // 3 :  Display both name and value
                    //  4:  Display evaluated value only
                    //  5:  Display both name and evaluated value

                <DisplayLocationInfo>:
                <int>   // 0 :  Left
                        // 1 :  Top
                        // 2 :  Right
                        // 3 :  Bottom
                        // 4 :  Center
                        // 5 :  Custom placement

                        <GraphicsDataInfo>:
 Array("NAME:Graphics",
          // one or more of the following
         <RectInfo>,
          <CircleInfo>,
           <ArcInfo>,
            <LineInfo>,
            <PolygonInfo>,
            <TextInfo>,
           <ImageInfo>)

<RectInfo>:
 "Rect:=", Array(<real>,   // line width
                 <int>,    //  fill pattern
                 <int>,    // color
                 <real>,   // angle, in radians
                 <real>,    // x position of center
                 <real>,    // y position of center
                          <real>,   // width
                          < real>)  // height

<CircleInfo>:
```

```
  "Circle:=", Array(<real>,    // line width
                    <int>,     //  fill pattern
                    <int>,     // color
                    <real>,    // x position of center
                    <real>,    // y position of center
                              < real>)   // radius

<ArcInfo>:
  "Arc:=", Array(<real>,    // line width
                 <int>,     //  line pattern
                 <int>,     // color
                 <real>,    // x position of center
                 <real>,    // y position of center
                           < real>,    // radius
                            <real>,    // start angle, in radians
                            <end>)     // end angle, in radians

<LineInfo>:
  "Line:=", Array(<real>,    // line width
                  <int>,     //  line pattern
                  <int>,     // color
                             <PointInfo>,    // must specify at least 2 points
                             <PointInfo>...)
            <PointInfo>:
             <real>,    // x position
             <real>     // y position

<PolygonInfo>:
  "Polygon:=", Array(<real>,    // line width
                     <int>,     //  fill pattern
                     <int>,     // color
                                <PointInfo>,     // must specify at least 3 points
                                <PointInfo>...)

<TextInfo>:
```

```
"Text:=", Array(<real>,    // x position
                <real>,   // y position
                <real>,   // angle, in radians
                <Justification>,
                 <bool>,    // is plotter font
                 <string>,    // font name
                  <int>,        // color
                  <string>)    // text string

              <Justification>:
              <int>  // 0 : left top
                     // 1 : left base
                     // 2 : left bottom
                     // 3 : center top
                     // 4 : center base
                     // 5 : center bottom
                     // 6 : right top
                     // 7 : right base
                     // 8 : right bottom

                <ImageInfo>:
                 "Image:=", Array(<RectInfo>,
                                  <ImageData>,
                                   <bool>)             // is mirrored

                  <ImageData>:
                   <string>,     // file path
                   <int>,          // 0 : use the file path and link to it
                                   // 1 : ignore file path and use next parameter
                    <string>      // text data, only present if preceding int is 1

<ListOfComponentNames>:
<string>,<string> ...
// The list may be empty.  When not empty, each string that is listed is a component
// that references the model to be edited.  Prior to editing, a clone of the model is
```

```
// made, and the components that are listed are modified so that they now refer to
// the clone.
```

Example:

```
Dim nam
oModelManager.EditWithComps_
("Nexxim Circuit Elements\Distributed\Distributed:bendo",
_
     Array("NAME:bendo new name", _
               "ModTime:=", 1152722165, _
               "Library:=", "Nexxim Circuit Elements\Dis-
tributed\Distributed", _
               "LibLocation:=", "SysLibrary", _
              Array("NAME:PinDef", _
                        "Pin:=", Array( "n1", _
-0.00254, _
0.00254,  _
0, _
"N", _
0, _
0, _
false, _
0, _
true, _
"",  _
false, _
false)),  _
Array("NAME:PinDef", _
"Pin:=", Array( "n2", _
0.00254, _
-0.00254, _
1.5708, _
"N", _
0, _
0, _
false, _
```

```
0, _
true, _
"", _
false, _
false)),  _
Array("NAME:Graphics", _
"Polygon:=", Array( 0, 0, 12566272, 0, 0.00381, 0,
.00127, 0.00127, _
0.00127, 0.00127, 0, 0.00381, 0, 0.00381,  _
0.002032, 0.00127, 0.00381), _
"Line:=", Array(0, 1, 12566272,  -0.00254, 0.00254, 0,
.00254),  _
"Line:=", Array(0, 1, 12566272, 0.00254, -0.00254,
.00254, 0)),  _
Array("NAME:PropDisplayMap", _
"W:=", Array(3, _
5, _
0, _
"Text:=", Array( 2.1684E-019, _
0.00504119, _
0, _
1, _
5, _
false, _
"Arial", _
0, _
"W=***")))),  _
rray("MY_COMP")
```

Export [model manager]

Use: Exports model(s) to a library

Command: Tools > Edit Configured Libraries > Models > Export to Library

Syntax:

```
Export Array("NAME:<LibraryName>",
<ModelName>,
<ModelName>...),
<LibraryLocation>
```

Return Value: None

Parameters:

```
<LibraryName>:
<string> // name of the library

        <ModelName>:
        <string> // composite name of model to export

        <LibraryLocation>:
         <string> // location of the library in <LibraryName>
// One of  "Project", "PersonalLib", or "UserLib"
```

Example:

```
oModelManager.Export _
Array("NAME Nexxim Circuit Elements\Distributed\Distrib-
uted:bendo:mylib", _ "myModel"), "PersonalLib"
```

GetData [model manager]

Use: Gets data that describes the definition.

Command: None

Syntax: `GetData(<DefinitionName>)`

Return Value: <DefinitionData> This is an array of data for the definition.

Parameters:

```
<DefinitionName>:
<string> // composite name of the definition to edit
```

Example:

```
Dim ModelData
ModelData = oModelManager.GetData("Nexxim Circuit Ele-
ments\Hspice:nexx_bjt_npn")
```

Note GetData allows the user to access definition information, make modifications, and then use the Edit or EditWithComps script commands to save the modified definition. Accordingly, for each type of definition, the array data returned to GetData should be the same array information that is supplied to the Edit or EditWithComps commands.

GetNames [model manager]

Use: Returns the names of the models (used and unused) in a design. The following script command, **IsUsed,** can then be used to separate used and unused models.

Command: None

Syntax: `GetNames()`

Return Value: An array of strings

Parameters: `None`

Example:

```
Dim modelNames
modelNames = oModelManager.GetNames()
```

IsUsed [model manager]

Use: Used to determine if a model is used in the design.

Command: None

Syntax: `IsUsed(<ModelName>)`

Return Value: <Boolean> // true if the specified model is used in the design

Parameters: `<ModelName>:`

<string>

Example:

```
Dim isUsed
isUsed = oModelManager.IsUsed("MyModel")
```

Remove [model manager]

Use: Removes a model from a library

Command: Tools > Edit Configured Libraries > Models > Remove Model

Syntax: `Remove <ModelName>,`
<IsProjectModel>,
<LibraryName>,
<LibraryLocation>

Return Value: None

Parameters: `<ModelName>:`

<string> // composite name of the model to remove

<IsProjectModel>:

<bool>

<LibraryName>:

<string> // name of the library

<LibraryLocation>:

<string> // location of the library in <LibraryName>

// One of "Project", "PersonalLib", or "UserLib"

Example:

```
oModelManager.Remove "Nexxim Circuit Elements\Distrib-
uted\Distributed:bendo", true, "Project"
```

RemoveUnused [model manager]

Use: Removes models that are not used in the design.

Command: **Project->Remove Unused Definitions** is similar but operates slightly different and does not record script commands.

Syntax: `RemoveUnused()`

Return Value: <bool> True if one or more models are removed.

Parameters: `None`

Example:

```
Dim removedDefs
removedDefs = oModelManager.RemoveUnused()
```

Note The order of calls to RemoveUnused is significant. As a result, removing definitions in an unordered fashion may cause other models in dependent definitions to be rendered unusable.

Also, the model and footprint of an unused component are not unusable until after the component itself is removed using the Component Manager Remove script.

Symbol Manager Script Commands

The symbol manager provides access to symbols in a Designer project. The manager object is accessed via the definition manager.

```
Set oDefinitionManager = oProject.GetDefinitionManager()
Set oSymbolManager = oDefinitionManager.GetManager("Symbol")
```

The symbol manager script commands are listed below.

Add
BringToFront
Edit
EditWithComps
Export
GetData
GetNames
IsUsed
Remove
RemoveUnused

Add [symbol manager]

Use: Add a symbol

Command: Tools > Edit Configured Libraries > Symbols > Add Symbol

Syntax:

```
Add Array("NAME:<SymbolName>",
"ModTime:=", <ModifiedTimeInfo>,
"Library:=", “”,          // Library name
"LibLocation:=", “Project”,    // Project Location
<PinDefInfo>,
<PinDefInfo>,...              // optional, to define pins
<GraphicsDataInfo>, // optional, to define graphics
<PropDisplayMapInfo>)) // optional, to define property displays
```

Return Value:

```
<string>
// composite name of the symbol.
// If the name requested conflicts with the name of  an existing
// symbol, the requested name is altered to be unique.
//  The name returned reflects any change made to be unique.
```

Parameters:

```
<SymbolName>:
<string>  // simple name of the symbol being added

  <ModifiedOnInfo>:
  An integer that corresponds to the number of seconds that have elapsed
  since 00:00 hours, Jan 1, 1970 UTC from the system clock.

 <PinDefInfo>:
  Array("NAME:PinDef",
        "Pin:=", Array (<string>,   // pin name
                        <real>,     // x location
                        <real>,     // y location
                        <real>,     // angle in radians
                        <PinType>,
                        <real>,   // line width
                        <real>,   // line length
                        <bool>,  // mirrored
                        <int>,    // color
                        <bool>,   // true if visible, false if not
                        <string>,  // hidden net name
                        <OptionalPinInfo>, // optional info
<PropDisplayMapInfo>))  // optional

<PinType>:
  <string>   // "N" : normal pin
             // "I"  : input pin
             // "O" : output pin

<OptionalPinInfo>:
// Specify both or neither
<bool>,    // true if name is to be shown
<bool>,    // true if number is to be shown
```

```
<PropDisplayMapInfo>:
        Array("NAME:PropDisplayMap",
              <PropDisplayInfo>,
              <PropDisplayInfo>,...)

        <PropDisplayInfo>:
        <NameString>,  Array(<DisplayTypeInfo>,
                             <DisplayLocationInfo>,
                             <int>,     // optional, level number
                             <TextInfo>)
        <NameString>:
        <string>   // PropertyName:=,  where PropertyName is the name of
                   // the property to be displayed

        <DisplayTypeInfo>:
        <int>    // 0 :  No display
                 // 1 :  Display name only
                 // 2 :  Display value only
                 // 3 :  Display both name and value
                 //  4:  Display evaluated value only
                 //  5:  Display both name and evaluated value

        <DisplayLocationInfo>:
        <int>    // 0 :  Left
                 // 1 :  Top
                 // 2 :  Right
                 // 3 :  Bottom
                 // 4 :  Center
                 // 5 :  Custom placement

<GraphicsDataInfo>:
  Array("NAME:Graphics",
        // one or more of the following
        <RectInfo>,
        <CircleInfo>,
```

```
                <ArcInfo>,
                 <LineInfo>,
                 <PolygonInfo>,
                 <TextInfo>,
                <ImageInfo>)

            <RectInfo>:
             "Rect:=", Array(<real>,    // line width
                             <int>,     //  fill pattern
                             <int>,     // color
                             <real>,    // angle, in radians
                             <real>,    // x position of center
                             <real>,    // y position of center
                                        <real>,   // width
                                        < real>)  // height

            <CircleInfo>:
             "Circle:=", Array(<real>,    // line width
                               <int>,     //  fill pattern
                               <int>,     // color
                               <real>,    // x position of center
                               <real>,    // y position of center
                                          < real>)   // radius

            <ArcInfo>:
             "Arc:=", Array(<real>,    // line width
                            <int>,     //  line pattern
                            <int>,     // color
                            <real>,    // x position of center
                            <real>,    // y position of center
                                       < real>,    // radius
                                        <real>,    // start angle, in radians
                                        <end>)     // end angle, in radians

           <LineInfo>:
```

```
  "Line:=", Array(<real>,    // line width
                  <int>,     //  line pattern
                  <int>,     // color
                              <PointInfo>,    // must specify at least 2 points
                              <PointInfo>...)
            <PointInfo>:
             <real>,    // x position
             <real>     // y position

<PolygonInfo>:
  "Polygon:=", Array(<real>,    // line width
                     <int>,     //  fill pattern
                     <int>,     // color
                                 <PointInfo>,     // must specify at least 3 points
                                 <PointInfo>...)

<TextInfo>:
  "Text:=", Array(<real>,    // x position
                  <real>,   // y position
                  <real>,   // angle, in radians
                  <Justification>,
                   <bool>,    // is plotter font
                   <string>,    // font name
                    <int>,       // color
                    <string>)    // text string

            <Justification>:
            <int>  // 0 : left top
                   // 1 : left base
                   // 2 : left bottom
                   // 3 : center top
                   // 4 : center base
                   // 5 : center bottom
                   // 6 : right top
                   // 7 : right base
```

```
                    // 8 : right bottom

                  <ImageInfo>:
                   "Image:=", Array(<RectInfo>,
                                    <ImageData>,
                                     <bool>)          // is mirrored

                   <ImageData>:
                    <string>,    // file path
                    <int>,        // 0 : use the file path and link to it
                                  // 1 : ignore file path and use next parameter
                     <string>     // text data, only present if preceding int is 1
```

Example:

```
oSymbolManager.Add Array("NAME:MySymbol",_

"ModTime:=", 1070989137, _

"Library:=", "", _

"LibLocation:=", "Project", _

Array("NAME:PinDef",  _

"Pin:=", Array ("newpin0", _
0.00254, _ 0, _
0, _
"N",_
0, _
0.00254, _
false, _
0, _
true, _
"",_
false, _
false)), _
```

```
Array("NAME:PinDef",  _
"Pin:=", Array ("newpin1", _
-0.00254, _
0, _
3.14159265358979, _
"N",_
0, _
0.00254, _
false, _
0, _
true, _
"",_
false, _
false)),
Array("NAME:Graphics", _
"Rect:=", Array(0,  _
0,  _
12566272, _
0,  _
4.33680868994202e-019, _
-0.000635, _
0.00508,  _
0.002794), _
"Circle:=", Array(0, _
0,  _
12566272,  _
0.000127, _
0.000635,  _
0.000635)))
```

BringToFront [symbol manager]

Use: Changes the drawing for the symbol so that the specified objects are drawn on top of other overlapping objects.

Command: Draw > Bring To Front

Syntax:
```
BringToFront Array("NAME:Selections", "Selections:=",
Array (<Object>, <Object>, ...))
```

Return Value: None

Parameters:
```
<Object>
<string>  // object to bring to the front
```

Example:
```
oDefinitionEditor.BringToFront Array("NAME:Selections",
"Selections:=", Array(  "SchObj@10"))
```

Edit [depricated]

Deprecated command — please use EditWithComps.

EditWithComps [symbol manager]

Use: Edit an existing symbol.

Command: None

Syntax:
```
EditWithComps <SymbolName>,
Array("NAME:<NewSymbolName>",
"ModTime:=", <ModifiedTimeInfo>,
"Library:=", <string>,          // Library name
"LibLocation:=", <string>,    // Project Location
<PinDefInfo>,
<PinDefInfo>,...              // optional, to define pins
<GraphicsDataInfo>, // optional, to define graphics
<PropDisplayMapInfo>), // optional, to define property displays
Array(<ListOfComponentNames>) // Component names
```

Return Value:
```
<string>
// composite name of the symbol.
// If the name requested conflicts with the name of  an existing
// symbol, the requested name is altered to be unique.
//  The name returned reflects any change made to be unique.
```

Parameters:
```
<SymbolName>:
<string>  // composite name of the symbol being edited
```

<NewSymbolName>:
<string> // new simple name for the symbol

<ModifiedOnInfo>:
An integer that corresponds to the number of seconds that have elapsed since 00:00 hours, Jan 1, 1970 UTC from the system clock.

<PinDefInfo>:
Array("NAME:PinDef",
"Pin:=", Array (<string>, // pin name
<real>, // x location
<real>, // y location
<real>, // angle in radians
<PinType>,
<real>, // line width
<real>, // line length
<bool>, // mirrored
<int>, // color
<bool>, // true if visible, false if not
<string>, // hidden net name
<OptionalPinInfo>, // optional info
<PropDisplayMapInfo>)) // optional

<PinType>:
<string> // "N" : normal pin
// "I" : input pin
// "O" : output pin

<OptionalPinInfo>:
// Specify both or neither
<bool>, // true if name is to be shown
<bool>, // true if number is to be shown

<PropDisplayMapInfo>:

```
Array("NAME:PropDisplayMap",
        <PropDisplayInfo>,
        <PropDisplayInfo>,...)

 <PropDisplayInfo>:
  <NameString>,  Array(<DisplayTypeInfo>,
                       <DisplayLocationInfo>,
                       <int>,     // optional, level number
                       <TextInfo>)

<NameString>:
        <string>   // PropertyName:=,  where PropertyName is the name of
                   // the property to be displayed

        <DisplayTypeInfo>:
        <int>    // 0 :  No display
                 // 1 :  Display name only
                 // 2 :  Display value only
                 // 3 :  Display both name and value
                 //  4:  Display evaluated value only
                 //  5:  Display both name and evaluated value

         <DisplayLocationInfo>:
         <int>    // 0 :  Left
                  // 1 :  Top
                  // 2 :  Right
                  // 3 :  Bottom
                  // 4 :  Center
                  // 5 :  Custom placement

                  <GraphicsDataInfo>:
  Array("NAME:Graphics",
          // one or more of the following
         <RectInfo>,
          <CircleInfo>,
```

```
                <ArcInfo>,
                 <LineInfo>,
                 <PolygonInfo>,
                 <TextInfo>,
                <ImageInfo>)

 <RectInfo>:
  "Rect:=", Array(<real>,    // line width
                        <int>,     //  fill pattern
                        <int>,     // color
                        <real>,   // angle, in radians
                        <real>,    // x position of center
                        <real>,    // y position of center
                                    <real>,   // width
                                    < real>)  // height

 <CircleInfo>:
  "Circle:=", Array(<real>,    // line width
                          <int>,     //  fill pattern
                          <int>,     // color
                          <real>,    // x position of center
                          <real>,    // y position of center
                                      < real>)   // radius

<ArcInfo>:
  "Arc:=", Array(<real>,    // line width
                        <int>,     //  line pattern
                        <int>,     // color
                        <real>,    // x position of center
                        <real>,    // y position of center
                                    < real>,    // radius
                                     <real>,    // start angle, in radians
                                     <end>)    // end angle, in radians

<LineInfo>:
```

```
  "Line:=", Array(<real>,    // line width
                  <int>,     //  line pattern
                  <int>,     // color
                             <PointInfo>,    // must specify at least 2 points
                             <PointInfo>...)
            <PointInfo>:
             <real>,    // x position
             <real>     // y position

<PolygonInfo>:
  "Polygon:=", Array(<real>,    // line width
                     <int>,     //  fill pattern
                     <int>,     // color
                                <PointInfo>,     // must specify at least 3 points
                                <PointInfo>...)

<TextInfo>:
  "Text:=", Array(<real>,    // x position
                  <real>,   // y position
                  <real>,   // angle, in radians
                  <Justification>,
                   <bool>,    // is plotter font
                   <string>,    // font name
                    <int>,       // color
                    <string>)    // text string

            <Justification>:
            <int>  // 0 : left top
                   // 1 : left base
                   // 2 : left bottom
                   // 3 : center top
                   // 4 : center base
                   // 5 : center bottom
                   // 6 : right top
                   // 7 : right base
```

// 8 : right bottom

<ImageInfo>:
"Image:=", Array(<RectInfo>,
<ImageData>,
<bool>) // is mirrored

<ImageData>:
<string>, // file path
<int>, // 0 : use the file path and link to it
// 1 : ignore file path and use next parameter
<string> // text data, only present if preceding int is 1

<ListOfComponentNames>:
<string>,<string> ...
// The list may be empty. When not empty, each string that is listed is a component
// that references the symbol to be edited. Prior to editing, a clone of the symbol is
// made, and the components that are listed are modified so that they now refer to
// the clone.

Example:

```
Dim nam
oSymbolManager.EditWithComps _
("Nexxim Circuit Elements\Distributed\Distributed:bendo",
_
Array("NAME:bendo new name", _
"ModTime:=", 1152722165, _
"Library:=", "Nexxim Circuit Elements\Distributed\Dis-
tributed", _
"LibLocation:=", "SysLibrary", _
Array("NAME:PinDef", _
"Pin:=", Array( "n1", _
-0.00254, _
0.00254, _
0, _
"N", _
```

```
0, _
0, _
false, _
0, _
true, _
"",  _
false, _
false)),  _
Array("NAME:PinDef", _
"Pin:=", Array( "n2", _
0.00254, _
-0.00254, _
1.5708, _
"N", _
0, _
0, _
false, _
0, _
true, _
"", _
false, _
false)),  _
Array("NAME:Graphics", _
"Polygon:=", Array( 0, 0, 12566272, 0, 0.00381, 0,
.00127, 0.00127, _
0.00127, 0.00127, 0, 0.00381, 0, 0.00381,  _
0.002032, 0.00127, 0.00381), _
"Line:=", Array(0, 1, 12566272,  -0.00254, 0.00254, 0,
.00254),  _
"Line:=", Array(0, 1, 12566272, 0.00254, -0.00254,
.00254, 0)),  _
Array("NAME:PropDisplayMap", _
"W:=", Array(3, _
5, _
0, _
"Text:=", Array( 2.1684E-019, _
```

```
0.00504119, _
0, _
1, _
5, _
false, _
"Arial", _
0, _
"W=***"))))),  _
Array("MY_COMP")
```

Export [symbol manager]

Use: Exports symbol(s) to a library

Command: Tools > Edit Configured Libraries > Symbols > Export to Library

Syntax:

```
Export Array("NAME:<LibraryName>",
<SymbolName>,
<SymbolName>...),
<LibraryLocation>
```

Return Value: None

Parameters:

```
<LibraryName>:
<string> // name of the library

        <SymbolName>:
        <string> // composite name of symbol to export

        <LibraryLocation>:
         <string> // location of the library in <LibraryName>
// One of  "Project", "PersonalLib", or "UserLib"
```

Example:

```
oSymbolManager.Export _
Array("NAME Nexxim Circuit Elements\Distributed\Distrib-
uted:bendo:mylib", _ "mySymbol"), "PersonalLib"
```

GetData [symbol manager]

Use: Gets data that describes the definition.

Command: None

Syntax: `GetData(<DefinitionName>)`

Return Value: <DefinitionData> This is an array of data for the definition.

Parameters: `<DefinitionName>:`

<string> // composite name of the definition to edit

Example:

```
Dim symbolData
symbolData = oSymbolManager.GetData("Nexxim Circuit Ele-
ments\Hspice:nexx_bjt_npn")
```

Note GetData allows the user to access definition information, make modifications, and then use the Edit or EditWithComps script commands to save the modified definition. Accordingly, for each type of definition, the array data returned to GetData should be the same array information that is supplied to the Edit or EditWithComps commands.

GetNames [symbol manager]

Use: Returns the names of the symbols (used and unused) in a design. The following script command, **IsUsed,** can then be used to separate used and unused symbols.

Command: None

Syntax: `GetNames()`

Return Value: An array of strings

Parameters: `None`

Example:

```
Dim symbolNames
symbolNames = oSymbolManager.GetNames()
```

IsUsed [symbol manager]

Use: Used to determine if a symbol is used in the design.

Command: None

Syntax: `IsUsed(<SymbolName>)`

Return Value: <Boolean> // true if the specified symbol is used in the design

Parameters:
```
<SymbolName>:
        <string>
```

Example:
```
Dim isUsed
isUsed = oSymbolManager.IsUsed("MySymbol")
```

Remove [symbol manager]

Use: Removes a symbol from a library

Command: Tools > Edit Configured Libraries > Symbols > Remove Symbol

Syntax:
```
Remove <SymbolName>,
       <IsProjectSymbol>,
       <LibraryName>,
       <LibraryLocation>
```

Return Value: None

Parameters:

`<SymbolName>:`

<string> // composite name of the symbol to remove

<IsProjectSymbol>:

<bool>

<LibraryName>:

<string> // name of the library

<LibraryLocation>:

<string> // location of the library in <LibraryName>

// One of "Project", "PersonalLib", or "UserLib"

Example:
```
oSymbolManager.Remove "Nexxim Circuit Elements\Distrib-
uted\Distributed:bendo", true, "Project"
```

RemoveUnused [symbol manager]

Use: Removes symbols that are not used in the design.

Command: **Project->Remove Unused Definitions** is similar but operates slightly different and does not record script commands.

Syntax: `RemoveUnused()`

Return Value: <bool> True if one or more symbols are removed.

Parameters: `None`

Example:

```
Dim removedDefs
removedDefs = oSymbolManager.RemoveUnused()
```

Note The order of calls to RemoveUnused is significant. As a result, removing definitions in an unordered fashion may cause other symbols in dependent definitions to be rendered unusable.

Also, the symbol and footprint of an unused component are not unusable until after the component itself is removed using the Component Manager Remove script.

Footprint Manager Script Commands

The footprint manager provides access to fooprints in a Designer project. The manager object is accessed via the definition manager.

```
Set oDefinitionManager = oProject.GetDefinitionManager()
Set oFootprintManager = oDefinitionManager.GetManager("Footprint")
```

The footprint manager script commands are listed below:

Add
Edit
EditWithComps
Export
GetData
GetNames
IsUsed
Remove
RemoveUnused

Add [footprint manager]

Use: Add a footprint

Command: Tools > Edit Configured Libraries > Footprints > Add Footprint

Syntax:

```
Add Array("NAME:<FootprintName>,
                                "ModTime:=", <ModifiedOnInfo>,
                                 "Library:=", “”,
                                 "LibLocation:=", “Project”,
                                "OkayToMirror:=", <bool>,
                                 "DefUnits:=", <UnitType>,
                                 Array(NAME:Lyrs",
                                       "Layer:=",  <LayerArray>,
                                        "Layer:=",  <LayerArray>…,
                                        "SLayer:=",  <StackupLayerArray>,
                                        "SLayer:=",  <StackupLayerArray>...),
                                "ActLyr:=", <string>,                  // name of active layer
                                 "Tol:=", <ToleranceArray> // optional
<PrimitivesInfo>,                      // optional
                                 <PinsInfo>,                           // optional
                                 <ViasInfo>,                           // optional
                                 <EdgeportsInfo>,                      // optional
                                <ComponentPropertyInfo>,
                                 <ScriptInfo>)     // optional, specified for scripted foot-
prints
```

Return Value:

<string>
// composite name of the footprint.
// If the name requested conflicts with the name of an existing
// footprint, the requested name is altered to be unique.
// The name returned reflects any change made to be unique.

Parameters:

<FootprintName>:
<string> // simple name of footprint to create

<ModifiedOnInfo>:
An integer that corresponds to the number of seconds that have elapsed since 00:00 hours, Jan 1, 1970 UTC from the system clock.

<UnitType>:

<string> // default length units to use if units are not specified in other
// parameters

<LayerArray>:
Array("N:=", <string>, // layer name
"ID:=", <int> ,
"T:=", <LayerTypeInfo>, // layer type
"TB:=", <TopBottomInfo>,
"Col:=", <int>, // optional - color
"Pat:=", <int>, // optional - fill pattern
"Vis:=", <bool>, // optional - are objects on layer visible
"Sel:=", <bool>, // optional - are objects on layer selectable
"L:=", <bool>) // optional
// are objects on layer locked (can't be edited)

<LayerTypeInfo>:
<string> // one of: signal, dielectric, metalized signal, assembly, silkscreen, solder-mask, solderpaste, glue, or user

<TopBottomInfo>:
<string> // one of: top, neither, bottom, or template

<StackupLayerArray>:
Array(<LayerArray>,
"Elev:=", <ElevationInfo>,
"SubL:=", Array("Th:=", <Dimension>,
"LElev:=", <Dimension>,
"R:=", <Dimension>,
"M:=", <MaterialInfo>))

<ElevationInfo>:
<string> // "top" - snap to top
// "mid" - snap to middle
// "bot" - snap to bottom
// "edit" - manual edit

```
                                    //  "none"

                                <Dimension>:
                                <string>   // real number, may include units

                                <MaterialInfo>:
                                 <string>  // name of the layer material

                                <ToleranceArray>:
                                 Array(<real>,     // distance tolerance
                                        <real>,    // angle tolerance (radians)
                                        <real>)    // dimensionless tolerance

<PrimitivesInfo>:
  Array("NAME:Prims",
          // one or more of the following
          <RectInfo>,
           <CircleInfo>,
            <ArcInfo>,
             <LineInfo>,
             <PolygonInfo>,
             <TextInfo>,
            <ImageInfo>)

 <RectInfo>:
  "Rect:=", Array(<real>,    // line width
                    <int>,     //  fill pattern
                    <int>,     // color
                    <real>,   // angle, in radians
                    <real>,    // x position of center
                    <real>,    // y position of center
                                <real>,   // width
                                < real>)  // height

 <CircleInfo>:
```

```
 "Circle:=", Array(<real>,    // line width
                   <int>,     //  fill pattern
                   <int>,     // color
                   <real>,    // x position of center
                   <real>,    // y position of center
                              < real>)   // radius

<ArcInfo>:
 "Arc:=", Array(<real>,    // line width
                <int>,     //  line pattern
                <int>,     // color
                <real>,    // x position of center
                <real>,    // y position of center
                           < real>,    // radius
                            <real>,    // start angle, in radians
                            <end>)     // end angle, in radians

<LineInfo>:
 "Line:=", Array(<real>,    // line width
                 <int>,     //  line pattern
                 <int>,     // color
                            <PointInfo>,    // must specify at least 2 points
                            <PointInfo>...)
          <PointInfo>:
           <real>,    // x position
           <real>     // y position

<PolygonInfo>:
  "Polygon:=", Array(<real>,    // line width
                     <int>,     //  fill pattern
                     <int>,     // color
                                <PointInfo>,     // must specify at least 3 points
                                <PointInfo>...)

<TextInfo>:
```

```
"Text:=", Array(<real>,    // x position
                <real>,    // y position
                <real>,    // angle, in radians
                <Justification>,
                <bool>,    // is plotter font
                <string>,  // font name
                <int>,     // color
                <string>)  // text string
              <Justification>: <int>
// 0 : left top
                // 1 : left base
                // 2 : left bottom
                // 3 : center top
                // 4 : center base
                // 5 : center bottom
                // 6 : right top
                // 7 : right base
                // 8 : right bottom

             <ImageInfo>:
              "Image:=", Array(<RectInfo>,
                               <ImageData>,
                               <bool>)          // is mirrored

              <ImageData>:
               <string>,   // file path
               <int>,      // 0 : use the file path and link to it
                           // 1 : ignore file path and use next parameter
                <string>   // text data, only present if preceding int is 1

<PinsInfo>:
                Array("NAME:Pins",
                      "P:=", <PinArray>,
                       "P:=", <PinArray>,...)
```

<PinArray>:
Array("Port:=", Array("Id:=", <int>,
"Clr:=", <real>, // optional - clearance
"N:=", <string>), // pin name
"Pos:=", Array("x:=", <Location>, // padstack (x,y) position
"y:=", <Location>),
"VRt:=", <Angle>, // optional - rotation
"HD:=", <Size>, // optional - hole diameter
<PadstackImplementationInfo>)

<Location>:
<string> specifying real number and units (may use variables)

<Angle>:
<string> specifying angle with a real number and units

<Size>:
<string> specifying size with a real number and units

<PadstackImplementationInfo>:
If another with the same implementation has already been specified:
"Ref:=", <int> // id of the other
If not:
"Ref:=", <string>, // name
"Frm:=", <int>, // id of highest layer
"To:=", <int>, // id of lowest layer
<LayerPlacementInfo>,
"Man:=", <int>, // optional, 1 if manually placed, 0 if not (default)
"Use:=", Array(<PadUseInfo>, <PadUseInfo>…) // array may be empty

<LayerPlacementInfo>:
"Lyr:=", Array("Mrg:=", <int>, // optional,
// 1 if all layers have been merged (default)
// 0 if layer are not merged

```
                                    "Flp:=", <int>,  // optional,
                                                // 1 if placed bottom up
                                                // 0 if not (default)
                                    "Map:=", <ParentToLocalLayerInfo>)

                                <ParentToLocalLayerInfo>:
                                Array("U:=", <DirectionOfUniqueness>,
                                        "F:=", Array(<int>, <int>, …),  // forward mapping
                                                                        // -1 is not mapped
                                        "B:=", Array(<int>, <int>, …))  // backward mapping
                                                                        // -1 is not mapped

<PadUseInfo>:
                            "Pad:=", Array("Lid:=", <int>,       // layer id
                                        "T:=", <string>,     // type : "connected", "thermal",
                                                             // "no_pad, "not_connected",
                                                             // or "not_connected_thermal"
                                        "Man:=", <int>)      // optional, 1 if manually placed
                                                             //  0 if not (default)

                            <DirectionOfUniqueness>:
                            <string>  // one of "forward", "backward", or "two ways"

                            <ViasInfo>:
                            Array("NAME:Vias", <ViaInfo>, <ViaInfo>…)

                            <ViaInfo>:
                            "V:=", Array("Id:=", <int>,
                                        "N:=", <string>,     // name
                                        "Pos:=", Array("x:=", <Location>,  // via (x,y) posi-
tion
                                                        "y:=", <Location>),
                            "VRt:=", <Angle>,      // optional - rotation
                            <ImplementationInfo>
```

<EdgeportsInfo>:
Array("NAME:EPorts", <EdgePortArray>, <EdgePortArray>…)

<EdgePortArray>:
Array("NAME:EP",
"LP:=", Array("Id:=", <int>, // port id
"N:=", <string>), // port name
"Eo:=", Array(<edge description>, <edge description>,...))

<edge description> for primitive edges

"et:=", "pe", "pr:=", <id>, "ei:=", <edge#>

<id>: integer that is the primitive id

<edge#>: integer that is the edge number on the primitive

<edge description> for via edges

"et:=", "pse", "layer:=", <layer id>, "se:=", <via id>,
"sx:=", <start X location>, "sy:=", <start Y location>, "ex:=", <end X location>,
"ey:=", <end Y location>, "h:=", <arc height>, "rad:=", <radians>

<via id>: an integer that is the id of the via to use
<layer id>: an integer that is the id of the layer of the pad of the via to use
<start X location>

<start Y Location>:
doubles that are the X, Y location of the start point of the edge arc <end X location>
<end Y Location>:
doubles that are the X, Y location of the end point of the edge arc

<arc height>: double giving the height of the edge arc (0 for a straight edge)
<radians>: double giving the arc size in radians (0 for a straight edge)

<ComponentPropertyInfo>:
Array("NAME:CProps",
"VariableProp:=", <VariableInfo>,
"VariableProp:=", <VariableInfo>,
…)

<VariableInfo>:
Array(<string>, // name
<FlagLetters>,
<string>, // description
"CB:=", <string>, // optional - script for call back
<string>) // value: number, variable, or expression

<ScriptInfo>:
Array("NAME:script",
"language:=", <string>, // one of "javascript" or "vbscript"
"UsesScript:=", true,
"script:=", <string>) // contents of script

Example:

```
oFootprintManager.Add (Array("NAME:BCL", _
"ModTime:=", 1023388445, _
"Library:=", "", _
"LibLocation:=", "Project", _
"OkayToMirror:=", false, _
"DefUnits:=", "mm", _
Array("NAME:Lyrs", _
"Layer:=", Array("N:=",  "Measures", _
"ID:=", 8, _
"T:=", "measures", _
"Col:=", 4144959, _
"Pat:=", 1), _
"Layer:=", Array("N:=", "Rats",  _
"ID:=", 1, _
"T:=", "rat", _
```

```
"Col:=", 16711680, _
"Pat:=", 1), _
"Layer:=", Array("N:=", "Errors", _
"ID:=", 2, _
"T:=", "error", _
"Col:=", 255, _
"Pat:=", 1, _
"L:=", true), _
"Layer:=", Array("N:=", "Symbols", _
"ID:=", 3, _
"T:=", "symbol", _
"Col:=", 8323199, _
"Pat:=", 4), _
"Layer:=", Array("N:=", "Assembly", _
"ID:=", 4, _
"T:=", "assembly", _
"TB:=", "top",  _
"Col:=", 16711680, _
Pat:=", 3), _
"Layer:=", Array("N:=", "Silkscreen", _
"ID:=", 5, _
"T:=", "silkscreen", _
TB:=", "top",  _
"Col:=", 8454143, _
"Pat:=", 6), _
SLayer:=", Array("Layer:=", Array("N:=",  "Cover", _
"ID:=", 12, _
"T:=", "metalizedsignal", _
"Col:=", 32639,  _
Pat:=", 7), _
"Elev:=", "mid",  _
"SubL:=", Array("Th:=", "0mm", _
"LElev:=", "118.110236220472mil", _
"R:=",  "0mm", _
"Mat:=", "")), _
```

```
"SLayer:=", Array("Layer:=", Array("N:=", "Dielec3", _
"ID:=",  11, _
"T:=", "dielectric", _
"Col:=", 8323199, _
"Pat:=", 6),  _
"Elev:=", "none", _
"SubL:=", Array("Th:=",  "1mm",  _
"LElev:=", "78.740157480315mil", _
"R:=", "0mm", _
"Mat:=", "")),  _
"SLayer:=", Array("Layer:=", Array("N:=",  "Trace2", _
"ID:=", 10, _
"T:=", "signal", _
"Col:=", 8355584, _
"Pat:=", 5), _
"Elev:=",  "mid", _
"SubL:=", Array("Th:=", "0mm", _
"LElev:=", "78.740157480315mil",  _
"R:=", "0mm", _
"Mat:=", "")), _
"SLayer:=", Array("Layer:=", Array("N:=", "Dielec2", _
"ID:=", 6, _
"T:=", "dielectric", _
"Col:=", 8323072, _
"Pat:=", 4), _
"Elev:=", "none", _
"SubL:=", Array("Th:=",  "1mm", _
"LElev:=", "39.3700787401575mil", _
"R:=", "0mm", _
"Mat:=", "")),  _
"SLayer:=", Array("Layer:=", Array("N:=", "Trace1", _
"ID:=", 0, _
"T:=", "signal", _
"Col:=", 32512, _
"Pat:=", 3),  _
```

```
"Elev:=",  "mid", _
"SubL:=", Array("Th:=", "0mm", _
"LElev:=", "39.3700787401575mil",  _
"R:=", "0mm",  _
"Mat:=", "")), _
"SLayer:=", Array("Layer:=", Array("N:=", "Dielec1", _
"ID:=", 7, _
"T:=", "dielectric", _
"Col:=", 127, _
"Pat:=", 7), _
"Elev:=", "none", _
"SubL:=", Array("Th:=", "1mm", _
"LElev:=", "0mil", _
"R:=", "0mil", _
"Mat:=", "")), _
"SLayer:=", Array("Layer:=", Array("N:=", "Ground", _
"ID:=", 9,  _
"T:=", "metalizedsignal",  _
"Col:=", 4144959, _
"Pat:=", 6),  _
"Elev:=",  "mid", _
"SubL:=", Array("Th:=", "0mil", _
"LElev:=", "0mil", _
"R:=", "0mil", _
"Mat:=",  "")))), _
"ActLyr:=", "Trace2", _
Array("NAME:Prims",
"rect:=", Array("Id:=", 10003, _
"Lyr:=", 10,  _
"N:=", "rect101", _
"x:=", "0mm", _
"y:=", "0mm", _
"w:=", "P", _
"h:=", "W",  _
"Vds:=", Array()),  _
```

```
"rect:=", Array("Id:=", 10103,  _
"Lyr:=", 0, _
"N:=", "rect103", _
"x:=", "0mm", _
"y:=", "0mm", _
"w:=", "P", _
"h:=", "W", _
"Vds:=", Array())), _
Array("NAME:Pins",
"P:=", Array("Port:=", Array("Id:=", 3, "N:=", "n1"), _
"Pos:=", Array("x:=", "-P/2", "y:=", "0mm"), _
"VRt:=", "180deg",  _
"Ref:=", "NoPad SMT East", _
"Frm:=", 12, _
"To:=", 9, _
"Lyr:=", Array("Map:=", Array("U:=",   "forward",_
"F:=", Array(-1, -1, -1, -1,_
-1, -1, -1, -1,_
-1, -1, 0, -1, -1),  _
"B:=", Array(10))), _
"Use:=", Array()),  _
"P:=", Array("Port:=", Array("Id:=", 4, "N:=", "n4"),_
"Pos:=", Array("x:=","P/2", "y:=", "0mm"),_
"HD:=", "1mm", _
"Ref:=", 3), _
"P:=", Array("Port:=", Array("Id:=",5, "N:=", "n2"),_
"Pos:=", Array("x:=", "-P/2", "y:=", "0mm"),_
"VRt:=", "180deg",_
"Ref:=", "NoPad SMT East",_
"Frm:=", 12, _
"To:=", 9, _
"Lyr:=", Array("Map:=", Array("U:=", "forward",_
"F:=", Array(0, -1, -1, -1, -1,_
-1, -1, -1, -1, _
-1, -1, -1, -1), _
```

```
"B:=", Array(0))), _
"Use:=", Array()), _
"P:=", Array("Port:=", Array("Id:=", 6, "N:=", "n3"), _
"Pos:=", Array("x:=", "P/2", "y:=", "0mm"),_
"HD:=", "1mm", _
"Ref:=", 5)),  _
Array("NAME:Nets"), _
Array("NAME:CProps",
"VariableProp:=", Array("W", _
"UD", _
"", _
"1.5mm"),_
"VariableProp:=", Array("P", _
"UD", _
"", _
10mm"))))
```

Edit [footprint manager]

Deprecated command — please use EditWithComps.

EditWithComps [footprint manager]

Use: Edit an existing footprint.

Command: None

Syntax:

```
EditWithComps <FootprintName>,
Array("NAME:<NewFootprintName>,
"ModTime:=", <ModifiedOnInfo>,
"Library:=", "",
"LibLocation:=", "Project",
"OkayToMirror:=", <bool>,
"DefUnits:=", <UnitType>,
Array(NAME:Lyrs",
"Layer:=", <LayerArray>,
"Layer:=", <LayerArray>…,
"SLayer:=", <StackupLayerArray>,
```

"SLayer:=", <StackupLayerArray>...),
"ActLyr:=", <string>, // name of active layer
"Tol:=", <ToleranceArray> // optional
<PrimitivesInfo>, // optional
<PinsInfo>, // optional
<ViasInfo>, // optional
<EdgeportsInfo>, // optional
<ComponentPropertyInfo>,
<ScriptInfo>, // optional, specified for scripted footprints
Array(<ListofComponentNames>) // Component names

Return Value: <string>
// composite name of the footprint.
// If the name requested conflicts with the name of an existing
// footprint, the requested name is altered to be unique.
// The name returned reflects any change made to be unique.

Parameters: `<FootprintName>:`
<string> // composite name of the footprint being edited

<NewFootprintName>:
<string> // new simple name for the footprint

<ModifiedOnInfo>:
An integer that corresponds to the number of seconds that have elapsed since 00:00 hours, Jan 1, 1970 UTC from the system clock.

<**UnitType**>:
<string> // default length units to use if units are not specified in other
// parameters

<**LayerArray**>:
Array("N:=", <string>, // layer name
"ID:=", <int> ,
"T:=", <LayerTypeInfo>, // layer type
"TB:=", <TopBottomInfo>,

"Col:=", <int>, // optional - color
"Pat:=", <int>, // optional - fill pattern
"Vis:=", <bool>, // optional - are objects on layer visible
"Sel:=", <bool>, // optional - are objects on layer selectable
"L:=", <bool>) // optional
// are objects on layer locked (can't be edited)

<**LayerTypeInfo**>:
<string> // one of: signal, dielectric, metalized signal, assembly, silkscreen, solder-mask, solderpaste, glue, or user

<**TopBottomInfo**>:
<string> // one of: top, neither, bottom, or template

<**StackupLayerArray**>:
Array(<LayerArray>,
"Elev:=", <ElevationInfo>,
"SubL:=", Array("Th:=", <Dimension>,
"LElev:=", <Dimension>,
"R:=", <Dimension>,
"M:=", <MaterialInfo>))

<**ElevationInfo**>:
<string> // "top" - snap to top
// "mid" - snap to middle
// "bot" - snap to bottom
// "edit" - manual edit
// "none"

<**Dimension**>:
<string> // real number, may include units

<**MaterialInfo**>:
<string> // name of the layer material

<**ToleranceArray**>:
Array(<real>, // distance tolerance
<real>, // angle tolerance (radians)
<real>) // dimensionless tolerance

<**PrimitivesInfo**>:
Array("NAME:Prims",
// one or more of the following
<RectInfo>,
<CircleInfo>,
<ArcInfo>,
<LineInfo>,
<PolygonInfo>,
<TextInfo>,
<ImageInfo>)

<**RectInfo**>:
"Rect:=", Array(<real>, // line width
<int>, // fill pattern
<int>, // color
<real>, // angle, in radians
<real>, // x position of center
<real>, // y position of center
<real>, // width
< real>) // height

<**CircleInfo**>:
"Circle:=", Array(<real>, // line width
<int>, // fill pattern
<int>, // color
<real>, // x position of center
<real>, // y position of center
< real>) // radius

<**ArcInfo**>:

"Arc:=", Array(<real>, // line width
<int>, // line pattern
<int>, // color
<real>, // x position of center
<real>, // y position of center
< real>, // radius
<real>, // start angle, in radians
<end>) // end angle, in radians

<**LineInfo**>:
"Line:=", Array(<real>, // line width
<int>, // line pattern
<int>, // color
<PointInfo>, // must specify at least 2 points
<PointInfo>...)
<PointInfo>:
<real>, // x position
<real> // y position

<**PolygonInfo**>:
"Polygon:=", Array(<real>, // line width
<int>, // fill pattern
<int>, // color
<PointInfo>, // must specify at least 3 points
<PointInfo>...)

<**TextInfo**>:
"Text:=", Array(<real>, // x position
<real>, // y position
<real>, // angle, in radians
<Justification>,
<bool>, // is plotter font
<string>, // font name
<int>, // color
<string>) // text string

<Justification>: <int>
// 0 : left top
// 1 : left base
// 2 : left bottom
// 3 : center top
// 4 : center base
// 5 : center bottom
// 6 : right top
// 7 : right base
// 8 : right bottom

<**ImageInfo**>:
"Image:=", Array(<RectInfo>,
<ImageData>,
<bool>) // is mirrored

<**ImageData**>:
<string>, // file path
<int>, // 0 : use the file path and link to it
// 1 : ignore file path and use next parameter
<string> // text data, only present if preceding int is 1

<**PinsInfo**>:
Array("NAME:Pins",
"P:=", <PinArray>,
"P:=", <PinArray>,...)

<**PinArray**>:
Array("Port:=", Array("Id:=", <int>,
"Clr:=", <real>, // optional - clearance
"N:=", <string>), // pin name
"Pos:=", Array("x:=", <Location>, // padstack (x,y) position
"y:=", <Location>),
"VRt:=", <Angle>, // optional - rotation
"HD:=", <Size>, // optional - hole diameter

<PadstackImplementationInfo>)

<**Location**>:
<string> specifying real number and units (may use variables)

<**Angle**>:
<string> specifying angle with a real number and units

<**Size**>:
<string> specifying size with a real number and units

<**PadstackImplementationInfo**>:
If another with the same implementation has already been specified:
"Ref:=", <int> // id of the other
If not:
"Ref:=", <string>, // name
"Frm:=", <int>, // id of highest layer
"To:=", <int>, // id of lowest layer
<LayerPlacementInfo>,
"Man:=", <int>, // optional, 1 if manually placed, 0 if not (default)
"Use:=", Array(<PadUseInfo>, <PadUseInfo>…) // array may be empty

<**LayerPlacementInfo**>:
"Lyr:=", Array("Mrg:=", <int>, // optional,
// 1 if all layers have been merged (default)
// 0 if layer are not merged
"Flp:=", <int>, // optional,
// 1 if placed bottom up
// 0 if not (default)
"Map:=", <ParentToLocalLayerInfo>)

<**ParentToLocalLayerInfo**>:
Array("U:=", <DirectionOfUniqueness>,
"F:=", Array(<int>, <int>, …), // forward mapping
// -1 is not mapped

"B:=", Array(<int>, <int>, …)) // backward mapping
// -1 is not mapped

<**PadUseInfo**>:
"Pad:=", Array("Lid:=", <int>, // layer id
"T:=", <string>, // type : "connected", "thermal",
// "no_pad, "not_connected",
// or "not_connected_thermal"
"Man:=", <int>) // optional, 1 if manually placed
// 0 if not (default)

<**DirectionOfUniqueness**>:
<string> // one of "forward", "backward", or "two ways"

<**ViasInfo**>:
Array("NAME:Vias", <ViaInfo>, <ViaInfo>…)

<**ViaInfo**>:
"V:=", Array("Id:=", <int>,
"N:=", <string>, // name
"Pos:=", Array("x:=", <Location>, // via (x,y) position
"y:=", <Location>),
"VRt:=", <Angle>, // optional - rotation
<ImplementationInfo>

<**EdgeportsInfo**>:
Array("NAME:EPorts", <EdgePortArray>, <EdgePortArray>…)

<**EdgePortArray**>
Array("NAME:EP",
"LP:=", Array("Id:=", <int>, // port id
"N:=", <string>), // port name
"Eo:=", Array(<edge description>, <edge description>,...))

<edge description> for primitive edges

"et:=", "pe", "pr:=", <id>, "ei:=", <edge#>

<id>: integer that is the primitive id

<edge#>: integer that is the edge number on the primitive

<edge description> for via edges

"et:=", "pse", "sel:=", <"via">, "layer:=", <layer id>,
"sx:=", <start X location>, "sy:=", <start Y location>, "ex:=", <end X location>,
"ey:=", <end Y location>, "h:=", <arc height>, "rad:=", <radians>

<"via">: text that is the name of the via to use
<layer id>: an integer that is the id of the layer of the pad of the via to use

<start X location>, <start Y Location>:
doubles that are the X, Y location of the start point of the edge arc
<end X location>, <end Y Location>:
doubles that are the X, Y location of the end point of the edge arc

<arc height>: double giving the height of the edge arc (0 for a straight edge)
<radians>: double giving the arc size in radians (0 for a straight edge)

<**ComponentPropertyInfo**>:
Array("NAME:CProps",
"VariableProp:=", <VariableInfo>,
"VariableProp:=", <VariableInfo>,
…)

<**VariableInfo**>:
Array(<string>, // name
<FlagLetters>,
<string>, // description
"CB:=", <string>, // optional - script for call back

<string>) // value: number, variable, or expression

<**ScriptInfo**>:
Array("NAME:script",
"language:=", <string>, // one of "javascript" or "vbscript"
"UsesScript:=", true,
"script:=", <string>) // contents of script

<ListOfComponentNames>:
<string>,<string> ...
// The list may be empty. When not empty, each string that is listed is a component
// that references the footprint to be edited. Prior to editing, a clone of the footprint is
// made, and the components that are listed are modified so that they now refer to
// the clone.

Example:

```
Dim nam
oFootprintManager.EditWithComps _
("Nexxim Circuit Elements\Distributed\Distributed:bendo",
_
Array("NAME:bendo new name", _
"ModTime:=", 1152722165, _
"Library:=", "Nexxim Circuit Elements\Distributed\Dis-
tributed", _
"LibLocation:=", "SysLibrary", _
Array("NAME:PinDef", _
"Pin:=", Array( "n1", _
-0.00254, _
0.00254,  _
0, _
"N", _
0, _
0, _
false, _
0, _
true, _
```

```
"",  _
false, _
false)),  _
Array("NAME:PinDef", _
"Pin:=", Array( "n2", _
0.00254, _
-0.00254, _
1.5708, _
"N", _
0, _
0, _
false, _
0, _
true, _
"", _
false, _
false)),  _
Array("NAME:Graphics", _
"Polygon:=", Array( 0, 0, 12566272, 0, 0.00381, 0,
.00127, 0.00127, _
0.00127, 0.00127, 0, 0.00381, 0, 0.00381,  _
0.002032, 0.00127, 0.00381), _
"Line:=", Array(0, 1, 12566272,  -0.00254, 0.00254, 0,
.00254),  _
"Line:=", Array(0, 1, 12566272, 0.00254, -0.00254,
.00254, 0)),  _
Array("NAME:PropDisplayMap", _
"W:=", Array(3, _
5, _
0, _
"Text:=", Array( 2.1684E-019, _
0.00504119, _
0, _
1, _
5, _
false, _
```

```
"Arial", _
0, _
"W=***")))),  _
Array("MY_COMP")
```

Export [footprint manager]

Use: Export a footprint to a library

Command: Tools > Edit Configured Libraries > Footprints > Export to Library

Syntax:
```
Export Array("NAME:<LibraryName>",
<FootprintName>,
<FootprintName>...),
<LibraryLocation>
```

Return Value: None

Parameters:

`<LibraryName>:`
<string> // name of the library

<FootprintName>:
<string> // composite name of footprint to export

<LibraryLocation>:
<string> // location of the library in <LibraryName>
// One of "Project", "PersonalLib", or "UserLib"

Example:

```
oFootprintManager.Export Array("NAME:mylib", "Distributed
Footprints:BPAD"), "PersonalLib"
```

GetData [footprint manager]

Use: Gets data that describes the definition.

Command: None

Syntax:
```
GetData(<DefinitionName>)
```

Return Value: <DefinitionData> This is an array of data for the definition.

Parameters:

`<DefinitionName>:`
<string> // composite name of the definition to edit

Example:

```
Dim footprintData
footprintData = oFootprintManager.GetData("Nexxim Circuit
Elements\Distributed\Nexxim_Footprints:MCPL13_Nexx")
```

Note GetData allows the user to access definition information, make modifications, and then use the Edit or EditWithComps script commands to save the modified definition. Accordingly, for each type of definition, the array data returned to GetData should be the same array information that is supplied to the Edit or EditWithComps commands.

GetNames [footprint manager]

Use: Returns the names of the footprints (used and unused) in a design. The following script command, **IsUsed,** can then be used to separate used and unused footprints.

Command: None

Syntax: `GetNames()`

Return Value: An array of strings

Parameters: `None`

Example:

```
Dim footprintNames
footprintNames = oFootprintManager.GetNames()
```

IsUsed [footprint manager]

Use: Used to determine if a footprint is used in the design.

Command: None

Syntax: `IsUsed(<FootprintName>)`

Return Value: <Boolean> // true if the specified footprint is used in the design

Parameters: `<FootprintName>:`

<string>

Example:

```
Dim isUsed
isUsed = oFootprintManager.IsUsed("MyFootprint")
```

Remove [footprint manager]

Use: Removes a footprint from a library

Command: Tools > Edit Configured Libraries > Footprints > Remove Footprint

Syntax:
```
Remove <FootprintName>,
<IsProjectFootprint>,
<LibraryName>,
<LibraryLocation>
```

Return Value: None

Parameters:

`<FootprintName>:`
<string> // composite name of the footprint to remove

<IsProjectFootprint>:
<bool>

<LibraryName>:
<string> // name of the library

<LibraryLocation>:
<string> // location of the library in <LibraryName>
// One of "Project", "PersonalLib", or "UserLib"

Example:

```
oFootprintManager.Remove "BPAD", true, "Distributed Foot-
prints", "Project"
oFootprintManager.Remove "BPAD", false, "MyLib", "Person-
alLib"
```

RemoveUnused [footprint manager]

Use: Removes footprints that are not used in the design.

Command: **Project->Remove Unused Definitions** is similar but operates slightly different and does not record script commands.

Syntax:
```
RemoveUnused()
```

Return Value: <bool> True if one or more footprints are removed.

Parameters:
```
None
```

Example:

```
Dim removedDefs
removedDefs = oFootprintManager.RemoveUnused()
```

Note The order of calls to RemoveUnused is significant. As a result, removing definitions in an unordered fashion may cause other footprints in dependent definitions to be rendered unusable.

Also, the symbol and footprint of an unused component are not unusable until after the component itself is removed using the Component ManagerRemove script.

Padstack Manager Script Commands

The padstack manager provides access to padstacks in a Designer project. The manager object is accessed via the definition manager.

```
Set oDefinitionManager = oProject.GetDefinitionManager()
Set oPadstackManager = oDefinitionManager.GetManager("Padstack")
```

The padstack manager script commands are listed below.

AddPortsToAllNets
AddPortsToNet
Add
Edit
EditWithComps
Export
GetData
GetNames
IsUsed
Remove
RemovePortsFromAllNets
RemovePortsFromNet
RemoveUnused

AddPortsToAllNets [padstack manager]

Use: Adds ports to all the pins in all the nets.

Command: Layout tab under Netsr-click>Create Ports

Syntax: `AddPortsToAllNets`

Return Value: None

Parameters: `None.`

Example:

```
oEditor. AddPortsToAllNets
```

AddPortsToNet [padstack manager]

Use: Add ports to all the pins on the designated nets.

Command: In Layoutr-click>Port>Create Ports on Net

Layout tab under Netsr-click>Create Ports

Syntax: `AddPortsToNet Array("NAME:Nets", "net-name", …)`

Return Value: None

Parameters: `net-namethe name of a net; all pins on this particular net receive ports.`

Example:

```
oEditor.AddPortsToNet Array("NAME:Nets", "CB1", "CB5")
```

Add [padstack manager]

Use: Add a padstack

Command: Tools > Edit Configured Libraries > Padstacks > Add Padstack

Syntax:

```
Add Array("NAME:<PadstackName>",
"ModTime:=", <ModifiedOnInfo>,
"Library:=", "", // name of the library
"LibLocation:=", "Project", // location of the named library
Array("NAME:psd",
"nam:= ", <PadstackName>,
"lib:=", "",   // name of the library
"mat:=", "",    // hole plating material
"plt:=", "0",    // percent of hole plating
Array("NAME:pds",
```

<LayerGeometryArray>,
<LayerGeometryArray....),
"hle:=", <PadInfo>
"hRg:=", <HoleRange>,
"sbsh:=", <SolderballShape>,
"sbpl:=", <SolderballPlacement>,
"sbr:=", <string>, // solderball diameter, real with units
"sb2:=", <string>, // solderball mid diameter, real with units
"sbn:=", <string>), // name of solderball material
"ppl:=", <PadPortLayerArray>)

Return Value: simple name of the added padstack
// If the name requested conflicts with the name of an existing
// padstack, the requested name is altered to be unique.
// The name returned reflects any change made to be unique.

Parameters: `<PadstackName>:`
<string> // simple name of padstack to create

<ModifiedOnInfo>:
An integer that corresponds to the number of seconds that have elapsed since 00:00 hours, Jan 1, 1970 UTC from the system clock.

<LayerGeometryArray>:
Array("Name:lgm",
"lay:=", <string>, // definition layer name
"id:=", <int>, // definition layer id
"pad:=", <PadInfo>, // pad
"ant:=", <PadInfo>, // antipad
"thm:=", <PadInfo>, // themal pad
"X:=", <string>, // pad x connection, real with units
"Y:=", <string>, // pad y connection, real with units
"dir:=", <DirectionString>) // pad connection direction

<PadInfo>:
Array("shp:=", <PadShape>,

"Szs:=", <DimensionArray>,
"X:=", <string>, // x offset, real with units
"Y:=", <string>, // y offset, real with units
"R:=", <string>) // rotation, real with units

<PadShape>:
<string> one of these choices
"No" // no pad
"Cir" // Circle
"Sq" // Square
"Rct" // Rectangle
"Ov" // Oval
"Blt" // Bullet
"Ply" // Polygons
"R45" // Round 45 thermal
"R90" // Round 90 thermal
"S45" // Square 45 thermal
"S90" // Square 90 thermal

<DimensionArray>:
Array(<string>, ...) // each string is a real with units for one of the dimensions of the shape

<DirectionString>:
<string> one of these choices
"No" // no direction
"Any" // any direction
"0" // 0 degrees
"45" // 45 degrees
"90" // 90 degrees
"135" // 135 degrees
"180" // 180 degrees
"225" // 225 degrees
"270" // 270 degrees
"315" // 315 degrees

<HoleRange>:
<string> one of these choices
"Thr" // through all layout layers
"Beg" // from upper pad layer to lowest layout layer
"End" // from upper layout layer to lowest pad layer
"UTL" // from upper pad layer to lowest pad layer

<SolderballShape>:
<string> one of these choices
"None" // no solderball
"Cyl" // cylinder solderball
"Sph" // spheroid solderball

<SolderballPlacement>:
<string> one of these choices
"abv" // above padstack
"blw" // below padstack

<PadPortLayerArray>:
Array(<int>, <int>,....) where each int is a layer id

Example:

```
oPadstackManager.Add Array("NAME:Circle - through3",
"ModTime:=", 1235765635, "Library:=",  _
  "", "LibLocation:=", "Project", Array("NAME:psd",
"nam:=", "Circle - through3", "lib:=",  _
  "", "mat:=", "", "plt:=", "0", Array("NAME:pds",
Array("NAME:lgm", "lay:=", "Top Signal", "id:=",  _
  0, "pad:=", Array("shp:=", "Cir", "Szs:=",
Array("2.5mm"), "X:=", "0mm", "Y:=",  _
  "0mm", "R:=", "0deg"), "ant:=", Array("shp:=", "Cir",
"Szs:=", Array("3.5mm"), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0"), "thm:=",
Array("shp:=", "No", "Szs:=", Array(), "X:=",  _
```

```
  "0mm", "Y:=", "0mm", "R:=", "0"), "X:=", "0mm", "Y:=",
"0mm", "dir:=", "Any"), Array("NAME:lgm", "lay:=",  _
  "SignalA", "id:=", 1, "pad:=", Array("shp:=", "Cir",
"Szs:=", Array("2mm"), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0deg"), "ant:=",
Array("shp:=", "Cir", "Szs:=", Array( _
  "3mm"), "X:=", "0mm", "Y:=", "0mm", "R:=", "0"),
"thm:=", Array("shp:=", "No", "Szs:=", Array(), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0"), "X:=", "0mm", "Y:=",
"0mm", "dir:=", "Any"), Array("NAME:lgm", "lay:=",  _
  "SignalB", "id:=", 2, "pad:=", Array("shp:=", "Cir",
"Szs:=", Array("2mm"), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0deg"), "ant:=",
Array("shp:=", "Cir", "Szs:=", Array( _
  "3mm"), "X:=", "0mm", "Y:=", "0mm", "R:=", "0deg"),
"thm:=", Array("shp:=", "No", "Szs:=", Array(), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0"), "X:=", "0mm", "Y:=",
"0mm", "dir:=", "Any"), Array("NAME:lgm", "lay:=",  _
  "Ground", "id:=", 3, "pad:=", Array("shp:=", "No",
"Szs:=", Array(), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0deg"), "ant:=",
Array("shp:=", "No", "Szs:=", Array(), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0"), "thm:=",
Array("shp:=", "R90", "Szs:=", Array( _
  "3mm", "0.75mm", "1mm"), "X:=", "0mm", "Y:=", "0mm",
"R:=", "0"), "X:=", "0mm", "Y:=",  _
  "0mm", "dir:=", "Any"), Array("NAME:lgm", "lay:=",
"Bottom signal", "id:=", 5, "pad:=", Array("shp:=",  _
  "Cir", "Szs:=", Array("1mm"), "X:=", "0mm", "Y:=",
"0mm", "R:=", "0deg"), "ant:=", Array("shp:=",  _
  "Cir", "Szs:=", Array("2mm"), "X:=", "0mm", "Y:=",
"0mm", "R:=", "0deg"), "thm:=", Array("shp:=",  _
  "No", "Szs:=", Array(), "X:=", "0mm", "Y:=", "0mm",
"R:=", "0"), "X:=", "0mm", "Y:=",  _
  "0mm", "dir:=", "Any")), "hle:=", Array("shp:=", "Cir",
"Szs:=", Array("1.5mm"), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0deg"), "hRg:=", "End",
"sbsh:=", "Sph", "sbpl:=",  _
```

```
  "abv", "sbr:=", "750um", "sb2:=", "1200um", "1200um",
"sbn:=", "solder"), "ppl:=", Array( _
  0, 1, 2, 3, 5))
```

Edit [padstack manager]

Use: Edit an existing padstack.

Command: Tools > Edit Configured Libraries > Padstacks > Edit Padstack

Syntax:

```
Edit  <PadstackName>,
Array("NAME:<NewPadstackName>",
"ModTime:=", <ModifiedOnInfo>,
"Library:=", "", // name of the library
"LibLocation:=", "Project", // location of the named library
Array("NAME:psd",
"nam:= ", <PadstackName>,
"lib:=", "",   // name of the library
"mat:=", "",    // hole plating material
"plt:=", "0",    // percent of hole plating
Array("NAME:pds",
<LayerGeometryArray>,
<LayerGeometryArray....),
"hle:=", <PadInfo>
"hRg:=", <HoleRange>,
"sbsh:=", <SolderballShape>,
"sbpl:=",  <SolderballPlacement>,
"sbr:=", <string>,   // solderball diameter, real with units
"sb2:=", <string>,   // solderball mid diameter, real with units
"sbn:=", <string>),  // name of solderball material
"ppl:=", <PadPortLayerArray>)
```

Return Value:

```
<string> // composite name of the padstack
// If the name requested conflicts with the name of  an existing
// padstack, the requested name is altered to be unique.
//  The name returned reflects any change made to be unique.
```

Parameters:

```
<PadstackName>:
<string> // composite name of padstack to edit
```

<NewPadstackName>:
<string> // new simple name for padstack

<ModifiedOnInfo>:
An integer that corresponds to the number of seconds that have elapsed since 00:00 hours, Jan 1, 1970 UTC from the system clock.

<LayerGeometryArray>:
Array("Name:lgm",
"lay:=", <string>, // definition layer name
"id:=", <int>, // definition layer id
"pad:=", <PadInfo>, // pad
"ant:=", <PadInfo>, // antipad
"thm:=", <PadInfo>, // themal pad
"X:=", <string>, // pad x connection, real with units
"Y:=", <string>, // pad y connection, real with units
"dir:=", <DirectionString>) // pad connection direction

<PadInfo>:
Array("shp:=", <PadShape>,
"Szs:=", <DimensionArray>,
"X:=", <string>, // x offset, real with units
"Y:=", <string>, // y offset, real with units
"R:=", <string>) // rotation, real with units

<PadShape>:
<string> one of these choices
"No" // no pad
"Cir" // Circle
"Sq" // Square
"Rct" // Rectangle
"Ov" // Oval
"Blt" // Bullet
"Ply" // Polygons

"R45" // Round 45 thermal
"R90" // Round 90 thermal
"S45" // Square 45 thermal
"S90" // Square 90 thermal

<DimensionArray>:
Array(<string>, ...) // each string is a real with units for one of the
// dimensions of the shape

<DirectionString>:
<string> one of these choices
"No" // no direction
"Any" // any direction
"0" // 0 degrees
"45" // 45 degrees
"90" // 90 degrees
"135" // 135 degrees
"180" // 180 degrees
"225" // 225 degrees
"270" // 270 degrees
"315" // 315 degrees

<HoleRange>:
<string> one of these choices
"Thr" // through all layout layers
"Beg" // from upper pad layer to lowest layout layer
"End" // from upper layout layer to lowest pad layer
"UTL" // from upper pad layer to lowest pad layer

<SolderballShape>:
<string> one of these choices
"None" // no solderball
"Cyl" // cylinder solderball
"Sph" // spheroid solderball

<SolderballPlacement>:
<string> one of these choices
"abv" // above padstack
"blw" // below padstack

<PadPortLayerArray>:
Array(<int>, <int>,....) where each int is a layer id

Example:

```
oPadstackManager.Edit "Circle - through1",
Array("NAME:Circle - through1", "ModTime:=",  _
  1235765635, "Library:=", "", "LibLocation:=", "Proj-
ect", Array("NAME:psd", "nam:=",  _
  "Circle - through1", "lib:=", "", "mat:=", "", "plt:=",
"0", Array("NAME:pds", Array("NAME:lgm", "lay:=",  _
  "Top Signal", "id:=", 0, "pad:=", Array("shp:=", "Cir",
"Szs:=", Array("2.5mm"), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0deg"), "ant:=",
Array("shp:=", "Cir", "Szs:=", Array( _
  "3.5mm"), "X:=", "0mm", "Y:=", "0mm", "R:=", "0"),
"thm:=", Array("shp:=", "No", "Szs:=", Array(), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0"), "X:=", "0mm", "Y:=",
"0mm", "dir:=", "Any"), Array("NAME:lgm", "lay:=",  _
  "SignalA", "id:=", 1, "pad:=", Array("shp:=", "Cir",
"Szs:=", Array("2mm"), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0deg"), "ant:=",
Array("shp:=", "Cir", "Szs:=", Array( _
  "3mm"), "X:=", "0mm", "Y:=", "0mm", "R:=", "0"),
"thm:=", Array("shp:=", "No", "Szs:=", Array(), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0"), "X:=", "0mm", "Y:=",
"0mm", "dir:=", "Any"), Array("NAME:lgm", "lay:=",  _
  "SignalB", "id:=", 2, "pad:=", Array("shp:=", "Cir",
"Szs:=", Array("2mm"), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0deg"), "ant:=",
Array("shp:=", "Cir", "Szs:=", Array( _
  "3mm"), "X:=", "0mm", "Y:=", "0mm", "R:=", "0deg"),
"thm:=", Array("shp:=", "No", "Szs:=", Array(), "X:=",  _
```

```
  "0mm", "Y:=", "0mm", "R:=", "0"), "X:=", "0mm", "Y:=",
"0mm", "dir:=", "Any"), Array("NAME:lgm", "lay:=",  _
  "Ground", "id:=", 3, "pad:=", Array("shp:=", "No",
"Szs:=", Array(), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0deg"), "ant:=",
Array("shp:=", "No", "Szs:=", Array(), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0"), "thm:=",
Array("shp:=", "R90", "Szs:=", Array( _
  "3mm", "0.75mm", "1mm"), "X:=", "0mm", "Y:=", "0mm",
"R:=", "0"), "X:=", "0mm", "Y:=",  _
  "0mm", "dir:=", "Any"), Array("NAME:lgm", "lay:=",
"Bottom signal", "id:=", 5, "pad:=", Array("shp:=",  _
  "Cir", "Szs:=", Array("1mm"), "X:=", "0mm", "Y:=",
"0mm", "R:=", "0deg"), "ant:=", Array("shp:=",  _
  "Cir", "Szs:=", Array("2mm"), "X:=", "0mm", "Y:=",
"0mm", "R:=", "0deg"), "thm:=", Array("shp:=",  _
  "No", "Szs:=", Array(), "X:=", "0mm", "Y:=", "0mm",
"R:=", "0"), "X:=", "0mm", "Y:=",  _
  "0mm", "dir:=", "Any")), "hle:=", Array("shp:=", "Cir",
"Szs:=", Array("1.5mm"), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0deg"), "hRg:=", "End",
"sbsh:=", "Sph", "sbpl:=",  _
  "abv", "sbr:=", "750um", "sb2:=", "1200um", "1200um",
"sbn:=", "solder"), "ppl:=", Array( _
  0, 1, 2, 3, 5))
```

EditWithComps [padstack manager]

Use: Edit an existing padstack.

Command: None

Syntax:

```
EditWithComps <PadstackName>,
Array("NAME:<NewPadstackName>",
"ModTime:=", <ModifiedOnInfo>,
"Library:=", "", // name of the library
"LibLocation:=", "Project", // location of the named library
Array("NAME:psd",
"nam:= ", <PadstackName>,
"lib:=", "",   // name of the library
```

"mat:=", "", // hole plating material
"plt:=", "0", // percent of hole plating
Array("NAME:pds",
<LayerGeometryArray>,
<LayerGeometryArray....),
"hle:=", <PadInfo>
"hRg:=", <HoleRange>,
"sbsh:=", <SolderballShape>,
"sbpl:=", <SolderballPlacement>,
"sbr:=", <string>, // solderball diameter, real with units
"sb2:=", <string>, // solderball mid diameter, real with units
"sbn:=", <string>), // name of solderball material
"ppl:=", <PadPortLayerArray>,
Array(<ListOfComponentNames>) // Component names

Return Value:

<string>
// composite name of the padstack.
// If the name requested conflicts with the name of an existing
// padstack, the requested name is altered to be unique.
// The name returned reflects any change made to be unique.

Parameters:

`<PadstackName>:`
<string> // composite name of the padstack being edited

<NewPadstackName>:
<string> // new simple name for the padstack

<ModifiedOnInfo>:
An integer that corresponds to the number of seconds that have elapsed since 00:00 hours, Jan 1, 1970 UTC from the system clock.

<LayerGeometryArray>:
Array("Name:lgm",
"lay:=", <string>, // definition layer name
"id:=", <int>, // definition layer id

"pad:=", <PadInfo>, // pad
"ant:=", <PadInfo>, // antipad
"thm:=", <PadInfo>, // themal pad
"X:=", <string>, // pad x connection, real with units
"Y:=", <string>, // pad y connection, real with units
"dir:=", <DirectionString>) // pad connection direction

<PadInfo>:
Array("shp:=", <PadShape>,
"Szs:=", <DimensionArray>,
"X:=", <string>, // x offset, real with units
"Y:=", <string>, // y offset, real with units
"R:=", <string>) // rotation, real with units

<PadShape>:
<string> one of these choices
"No" // no pad
"Cir" // Circle
"Sq" // Square
"Rct" // Rectangle
"Ov" // Oval
"Blt" // Bullet
"Ply" // Polygons
"R45" // Round 45 thermal
"R90" // Round 90 thermal
"S45" // Square 45 thermal
"S90" // Square 90 thermal

<DimensionArray>:
Array(<string>, ...) // each string is a real with units for one of the
// dimensions of the shape

<DirectionString>:
<string> one of these choices
"No" // no direction

"Any" // any direction
"0" // 0 degrees
"45" // 45 degrees
"90" // 90 degrees
"135" // 135 degrees
"180" // 180 degrees
"225" // 225 degrees
"270" // 270 degrees
"315" // 315 degrees

<HoleRange>:
<string> one of these choices
"Thr" // through all layout layers
"Beg" // from upper pad layer to lowest layout layer
"End" // from upper layout layer to lowest pad layer
"UTL" // from upper pad layer to lowest pad layer

<SolderballShape>:
<string> one of these choices
"None" // no solderball
"Cyl" // cylinder solderball
"Sph" // spheroid solderball

<SolderballPlacement>:
<string> one of these choices
"abv" // above padstack
"blw" // below padstack

<PadPortLayerArray>:
Array(<int>, <int>,....) where each int is a layer id

<ListOfComponentNames>:
<string>,<string> ...
// The list may be empty. When not empty, each string that is listed is a component
// that references the padstack to be edited. Prior to editing, a clone of the padstack is

// made, and the components that are listed are modified so that they now refer to
// the clone.

Example:

```
oPadstackManager.EditWithComps "Circle - through1",
Array("NAME:Circle - through1", "ModTime:=",  _
  1235765635, "Library:=", "", "LibLocation:=", "Proj-
ect", Array("NAME:psd", "nam:=",  _
  "Circle - through1", "lib:=", "", "mat:=", "", "plt:=",
"0", Array("NAME:pds", Array("NAME:lgm", "lay:=",  _
  "Top Signal", "id:=", 0, "pad:=", Array("shp:=", "Cir",
"Szs:=", Array("2.5mm"), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0deg"), "ant:=",
Array("shp:=", "Cir", "Szs:=", Array( _
  "3.5mm"), "X:=", "0mm", "Y:=", "0mm", "R:=", "0"),
"thm:=", Array("shp:=", "No", "Szs:=", Array(), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0"), "X:=", "0mm", "Y:=",
"0mm", "dir:=", "Any"), Array("NAME:lgm", "lay:=",  _
  "SignalA", "id:=", 1, "pad:=", Array("shp:=", "Cir",
"Szs:=", Array("2mm"), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0deg"), "ant:=",
Array("shp:=", "Cir", "Szs:=", Array( _
  "3mm"), "X:=", "0mm", "Y:=", "0mm", "R:=", "0"),
"thm:=", Array("shp:=", "No", "Szs:=", Array(), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0"), "X:=", "0mm", "Y:=",
"0mm", "dir:=", "Any"), Array("NAME:lgm", "lay:=",  _
  "SignalB", "id:=", 2, "pad:=", Array("shp:=", "Cir",
"Szs:=", Array("2mm"), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0deg"), "ant:=",
Array("shp:=", "Cir", "Szs:=", Array( _
  "3mm"), "X:=", "0mm", "Y:=", "0mm", "R:=", "0deg"),
"thm:=", Array("shp:=", "No", "Szs:=", Array(), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0"), "X:=", "0mm", "Y:=",
"0mm", "dir:=", "Any"), Array("NAME:lgm", "lay:=",  _
  "Ground", "id:=", 3, "pad:=", Array("shp:=", "No",
"Szs:=", Array(), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0deg"), "ant:=",
Array("shp:=", "No", "Szs:=", Array(), "X:=",  _
```

```
  "0mm", "Y:=", "0mm", "R:=", "0"), "thm:=",
Array("shp:=", "R90", "Szs:=", Array( _
  "3mm", "0.75mm", "1mm"), "X:=", "0mm", "Y:=", "0mm",
"R:=", "0"), "X:=", "0mm", "Y:=",  _
  "0mm", "dir:=", "Any"), Array("NAME:lgm", "lay:=",
"Bottom signal", "id:=", 5, "pad:=", Array("shp:=",  _
  "Cir", "Szs:=", Array("1mm"), "X:=", "0mm", "Y:=",
"0mm", "R:=", "0deg"), "ant:=", Array("shp:=",  _
  "Cir", "Szs:=", Array("2mm"), "X:=", "0mm", "Y:=",
"0mm", "R:=", "0deg"), "thm:=", Array("shp:=",  _
  "No", "Szs:=", Array(), "X:=", "0mm", "Y:=", "0mm",
"R:=", "0"), "X:=", "0mm", "Y:=",  _
  "0mm", "dir:=", "Any")), "hle:=", Array("shp:=", "Cir",
"Szs:=", Array("1.5mm"), "X:=",  _
  "0mm", "Y:=", "0mm", "R:=", "0deg"), "hRg:=", "End",
"sbsh:=", "Sph", "sbpl:=",  _
  "abv", "sbr:=", "750um", "sb2:=", "1200um", "1200um",
"sbn:=", "solder"), "ppl:=", Array( _
  0, 1, 2, 3, 5), Array("")
```

Export [padstack manager]

Use: Export a padstack to a library

Command: Tools > Edit Configured Libraries > Padstacks > Export to Library

Syntax:
`Export Array("NAME:<LibraryName>",`
<PadstackName>,
<PadstackName>...),
<LibraryLocation>

Return Value: None

Parameters:
`<LibraryName>:`
<string> // name of the library

<PadstackName>:
<string> // simple name of padstack to export

<LibraryLocation>:
<string> // location of the library in <LibraryName>
// One of "Project", "PersonalLib", or "UserLib"

Example:

```
oPadstackManager.Export Array("NAME:mylib", "myPad-
stack"), "PersonalLib"
```

GetData [padstack manager]

Use: Gets data that describes the definition.

Command: None

Syntax: `GetData(<DefinitionName>)`

Return Value: <DefinitionData> This is an array of data for the definition.

Parameters: `<DefinitionName>:`

<string> // composite name of the definition to edit

Example:

```
Dim padstackData
padstackData = oPadstackManager.GetData("NoPad SMT East")
```

Note GetData allows the user to access definition information, make modifications, and then use the Edit or EditWithComps script commands to save the modified definition. Accordingly, for each type of definition, the array data returned to GetData should be the same array information that is supplied to the Edit or EditWithComps commands.

GetNames [padstack manager]

Use: Returns the names of the padstack (used and unused) in a design. The following script command, **IsUsed,** can then be used to separate used and unused padstacks.

Command: None

Syntax: `GetNames()`

Return Value: An array of strings

Parameters: `None`

Example:

```
Dim padstackNames
padstackNames = oPadstackManager.GetNames()
```

IsUsed [padstack manager]

Use: Used to determine if a component is used in the design.

Command: None

Syntax: `IsUsed(<PadstackName>)`

Return Value: <Boolean> // true if the specified padstack is used in the design

Parameters:

`<PadstackName>:`
<string>

Example:

```
Dim isUsed
isUsed = oPadstackManager.IsUsed("MyPadstack")
```

Remove [padstack manager]

Use: Removes a padstack from a library

Command: Tools > Edit Configured Libraries > Padstacks > Remove Padstacks

Syntax:

`Remove <PadstackName>,`
<IsProjectPadstack>,
<LibraryName>,
<LibraryLocation>

Return Value: None

Parameters:

`<PadstackName>:`
<string> // simple name of the padstack to remove

<IsProjectPadstack>:
<bool>

<LibraryName>:
<string> // name of the library

<LibraryLocation>:
<string> // location of the library in <LibraryName>
// One of "Project", "PersonalLib", or "UserLib"

Example:

```
oPadstackManager.Remove "Polygon SMT", true, "Padstacks",
"Project"
oPadstackManager.Remove "Polygon SMT", false, "MyLib",
"PersonalLib"
```

RemovePortsFromAllNets [padstack manager]

Use: Removes ports from all the pins in all the nets.

Command: Layout tab under Netsr-click>Remove Ports

Syntax: `RemovePortsFromAllNets`

Return Value: None

Parameters: `None.`

Example:

```
oEditor.RemovePortsFromAllNets
```

RemovePortsFromNet [padstack manager]

Use: Removes ports from all the pins on the designated nets.

Command: In Layoutr-click>Port>Remove Ports from Net

Layout tab under Netsr-click>Remove Ports

Syntax: `RemovePortsFromNet`

Return Value: None

Parameters: `None.`

Example:

```
oEditor.RemovePortsFromNet
```

RemoveUnused [padstack manager]

Use: Removes padstacks that are not used in the design.

Command: **Project->Remove Unused Definitions** is similar but operates slightly different and does not record script commands.

Syntax: `RemoveUnused()`

Return Value: <bool> True if one or more padstacks are removed.

Parameters: `None`

Example:

```
Dim removedDefs
removedDefs = oPadstackManager.RemoveUnused()
```

Note The order of calls to RemoveUnused is significant. As a result, removing definitions in an unordered fashion may cause other padstacks in dependent definitions to be rendered unusable.

Material Manager Script Commands

The material manager provides access to materials in a Designer project. The manager object is accessed via the definition manager.

```
Set oDefinitionManager = oProject.GetDefinitionManager()
Set oMaterialManager = oDefinitionManager.GetManager("Material")
```

The material manager script commands are listed below.

Add
Edit
Export
GetData
GetNames
GetProperties
IsUsed
Remove
RemoveUnused

Add [material manager]

Use: Add a material to the project.

Command: None.

Syntax: `Add <material_name> <property1_name> <property1_value> <property2_name> <property2_value> ...`

Return Value: Name of the created material.

Parameters: `<material_name>`

Type: string

Value: A string comprised of the keyword "NAME", followed by a colon and the name of the material, e.g. "NAME:MyMaterial"

<property_name> <property_value>

Type: strings

Value: Arbitrary number of property-name/property-value pairs

Example:

```
createdMaterial = oMaterialMgr.Add( Array("NAME:MyMateri-
al1", "CoordinateSystemType:=", "Cartesian", "permittiv-
ity:=", "0.123") )


Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsysDesigner.Design-
erScript")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.NewProject
oProject.InsertDesign "Nexxim Circuit", "Nexxim1", "", ""
Set oMaterialMgr = oProject.GetDefinitionManager().Get-
Manager("Material")
Dim createdMat
createdMat = oMaterialMgr.Add( Array("NAME:MyMaterial1",
"CoordinateSystemType:=", "Cartesian", "permittivity:=",
"0.123") )
Dim message
message = "Created material '" + createdMat + "'"
Msgbox message
```

Edit [material manager]

Use: Edit properties of a project material.

Command: None.

Syntax: `Edit <material_name> <material_data>`

Return Value: None

Parameters: `<material_name>`

Type: string

Value: Name of the project material to edit.

<material_data>

Type: Array

Value: New material data.

Example:

```
oMaterialMgr.Edit( "MyMaterial2", Array("NAME:MyMateri-
al2_mod", "CoordinateSystemType:=",  _
"Cartesian", Array("NAME:AttachedData"), Array("NAME:Mod-
ifierData"), "permittivity:=",  _
"123", "permeability:=", "0.987654", "conductivity:=",
"58000000", "thermal_conductivity:=",  _
"400", "mass_density:=", "8933", "specific_heat:=",
"385", "youngs_modulus:=",  _
"120000000000", "poissons_ratio:=", "0.38", "thermal_ex-
pansion_coeffcient:=", "1.77e-005"))

Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsysDesigner.Design-
erScript")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.NewProject
oProject.InsertDesign "Nexxim Circuit", "Nexxim1", "", ""

Set oMaterialMgr = oProject.GetDefinitionManager().Get-
Manager("Material")

oMaterialMgr.Add Array("NAME:MyMaterial1", "Coordinate-
SystemType:=","Cartesian", Array("NAME:AttachedData"), _
                 Array("NAME:ModifierData"),"permittiv-
ity:=", "0.123")
oMaterialMgr.Add Array("NAME:MyMaterial2", "Coordinate-
SystemType:=", "Cartesian", Array("NAME:AttachedData"), _
```

```
                    Array("NAME:ModifierData"), "permittiv-
ity:=", "0.456")
oMaterialMgr.Add Array("NAME:MyMaterial3", "Coordinate-
SystemType:=", "Cartesian", Array("NAME:AttachedData"), _
                    Array("NAME:ModifierData"), "permittiv-
ity:=", "0.789")

oMaterialMgr.Edit( "MyMaterial2", Array("NAME:MyMateri-
al2_mod", "CoordinateSystemType:=",  _
  "Cartesian", Array("NAME:AttachedData"),
Array("NAME:ModifierData"), "permittivity:=",  _
  "123", "permeability:=", "0.987654", "conductivity:=",
"58000000", "thermal_conductivity:=",  _
  "400", "mass_density:=", "8933", "specific_heat:=",
"385", "youngs_modulus:=",  _
  "120000000000", "poissons_ratio:=", "0.38", "thermal_-
expansion_coeffcient:=", "1.77e-005"))
```

Export [material manager]

Use: Export a material to a library.

Command: None

Syntax: `Remove <material_data> <library_location>`

Return Value: None

Parameters: `<material_data>`

Type: Array

Value: An array consisting of the string "NAME:", followed by the library name and the name of the material to export.

<library_location>

Type: string

Value: Location of the library to export the material to ("UserLib" or "PersonalLib").

Example:

```
oMaterialMgr.Export Array("NAME:MyLib", "MyMaterial"),
"UserLib"

Dim oAnsoftApp
Dim oDesktop
```

```
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsysDesigner.Design-
erScript")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.NewProject
oProject.InsertDesign "Nexxim Circuit", "Nexxim1", "", ""

Set oMaterialMgr = oProject.GetDefinitionManager().Get-
Manager("Material")
oMaterialMgr.Add( Array("NAME:MyMaterial1", "Coordinate-
SystemType:=", "Cartesian", "permittivity:=", "0.123") )
oMaterialMgr.Export Array("NAME:MyLib", "MyMaterial1"),
"UserLib"
```

GetData [material manager]

Use: Get material data

Command: None

Syntax: `GetData <material_name>`

Return Value: Array of material data

Parameters: `<material_name>`

Type: string

Value: Name of the project material

Example:

```
materialData = oMaterialMgr.GetData( "MyMaterial2" )
```

GetNames [material manager]

Use: Get the names of the materials in a project

Command: None

Syntax: `GetNames`

Return Value: Names of the materials in a project

Parameters: `None`

Example:

```
materialNames = oMaterialMgr.GetNames()


Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsysDesigner.Design-
erScript")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.NewProject
oProject.InsertDesign "Nexxim Circuit", "Nexxim1", "", ""
Set oMaterialMgr = oProject.GetDefinitionManager().Get-
Manager("Material")
oMaterialMgr.Add( Array("NAME:MyMaterial1", "Coordinate-
SystemType:=", "Cartesian", "permittivity:=", "0.123") )
oMaterialMgr.Add( Array("NAME:MyMaterial2", "Coordinate-
SystemType:=", "Cartesian", "permittivity:=", "0.456")
Dim materialNames
materialNames = oMaterialMgr.GetNames()
```

GetProperties [material manager]

Use: Get material properties. Differs from GetData in that only material properties available to the user are returned by GetProperties.

Command: None

Syntax: `GetProperties <material_name>`

Return Value: Array of material data

Parameters: `<material_name>`

Type: string

Value: Name of the project material

Example:

```
materialProps = oMaterialMgr.GetProperties( "MyMaterial2"
)
```

IsUsed [material manager]

Use: Checks if a project material is in use

Command: None

Syntax: `IsUsed <material_name>`

Return Value: Returns 'True' if the material is in use.

Parameters: `<material_name>`

Type: string

Value: Name of the project material to check.

Example:

```
used = oMaterialMgr.IsUsed( "MyMaterial2" )
```

Remove [material manager]

Use: Remove a material from the project.

Command: None

Syntax: `Remove <material_name> <local> <library_name> <library_location>`

Return Value: None

Parameters: `<material_name>`

Type: string

Value: Name of the material to remove

<local>

Type: boolean

Value: If 'true', the material will be removed from the project; if 'false' the material will be removed from from a library.

<library_name> <library_location>

Type: string

Value: Name and location of the library to delete the material from ("UserLib" or "PersonalLib").

If <local> is 'True',the parameters are ignored and the material is removed from the project.

Example:

```
oMaterialMgr.Remove "MyMaterial1", true, "", "Project"
```

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("AnsysDesigner.Design-
erScript")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.NewProject
oProject.InsertDesign "Nexxim Circuit", "Nexxim1", "", ""

Set oMaterialMgr = oProject.GetDefinitionManager().Get-
Manager("Material")

Dim message
Dim createdMat
createdMat = oMaterialMgr.Add( Array("NAME:MyMaterial",
"CoordinateSystemType:=", "Cartesian", "permittivity:=",
"0.123") )
message = "Created material '" + createdMat + "'"
Msgbox message

oMaterialMgr.Remove "MyMaterial1", true, "", "Project"
```

RemoveUnused [material manager]

Use: Remove all unused materials from the project.

Command: None

Syntax: `RemoveUnused`

Return Value: None

Parameters: `None`

Example:

```
oMaterialMgr.RemoveUnused()
```

NdExplorer Manager Script Commands

The NdExplorer manager provides access to scripting in NdExplorer. The manager object is accessed via the definition manager.

```
Set oDefinitionManager = oProject.GetDefinitionManager()
Set oNdExplorerManager = oDefinitionManager.GetManager("NdExplorer")
```

The NdExplorer manager script commands are listed below.

ExportFullWaveSpice [NdExplorer Manager]

Use: Export FullWaveSpice data in a format of your choice.

Command: File > Export MacroModel > Broadband (SYZ, FWS....)

Syntax:

```
ExportFullWaveSpice
"DesignName", // Design name. Can be left blank, if loading solution from a file.
true/false,         // true - solution loaded from file, false- loaded from design
"Name",          // If loading from design this is the solution name, else this is the
                 // full path of the file from which the solution is loaded
"variation",     // Pick a particular variation. Leave blank if no variation.
Array("NAME:Frequencies"), // Optional; if none defined all frequencies are used
Array("NAME:SpiceData",          //  Spice export options object
"SpiceType:=", "SSS",                 // SpiceType can be  "PSpice", "HSpice", "Spectre", "SSS",
                                      // "Simplorer",  "TouchStone1.0", "TouchStone2.0"
"EnforcePassivity:=",   false,        // Enforce Passivity  true/false
"EnforceCausality:=", false,          // Enforce Causality  true/false
"UseCommonGround:=", false,  // Use common ground true/false
"FittingError:=", 0.5,                // Fitting error
"MaxPoles:=", 400,                    //  Maximum Order
"PassivityType:=", "ConvexOptimization",  // Passivity Type can be "ConvexOptimization",
                                          // "PassivityByPerturbation", or "IteratedFittingOfPV"
"ColumnFittingType:=", "Column", // Column FittingType can be "Column", "Entry", "Matrix"
```

```
"SSFittingType:=", "TWA", // SS Fitting Type can be "TWA", "IterativeRational"
"RelativeErrorToleranc:=", false,          // Relative error tolerance true/false
"TouchstoneFormat:=",  "MA",               // Touchstone Format "MA", "RI", "DB"
"TouchstoneUnits:=", "Hz",                 // Touchstone Units "Hz", "KHz", "MHz",
"MHz"
"TouchStonePrecision:=", 8,                // Touchstone precision
"ExportDirectory:=", "C:/Examples/LNA/",   // Directory to export to
"ExportSpiceFileName:=", "Linckt_HBTest_2.sss",  // Spice export file
"FullwaveSpiceFileName:=", "Linckt_HBTest.sss", // FWS file
"CreateNPortModel:=",   true   // Create a model based on the exported file  true/false
)
```

ExportNetworkData [NdExplorer Manager]

Use: Export the solution in a format of your choice (Citifile, Spreadsheet, Matlab)

Command: File > Export SYZ Data

Syntax:

```
ExportNetworkData
"DesignName", // Design name. Can be left blank, if loading solution from a file.
true/false,        // true - solution loaded from file, false- loaded from design
"Name",          // If loading from design this is the solution name, else this is the
                 // full path of the file from which the solution is loaded
"ExportFile",      // full path of file to export to
"variation",     // Pick a particular variation. Leave blank if no variation
Array("NAME:Frequencies"), // optional, if none defined all frequencies are used
Array("NAME:Options",            //  Export options object
"DataTypes:=", Array("S"),       // DataTypes can be "S", "Y","Z", "G", and "Z0",
                                 // for S , Y, Z matrix, Gamma and Z0 (zero)
"DisplayFormat:=",  "MA",        // DisplayFormat "MA", "RI", "DB"
"FileType:=", "",                 // Export File Type
                        // 2 - Spreadsheet(*.tab)
                        // 3 - Touchstone(*.sNp)
                        // 4 - Citifile(*.cit)
                        //  6 - Neutral format(*.nmf)
                        //  7 - Matlab format(*.m)
"Renormalize:=", false,          //  Renormalize   true/false
"RefImpedance:=",50,             // Reference Impedance
```

```
"Precision:=", 8,                    // Number of digits Precision
"CreateNPortModel:=",   true   // Create a model based on the exported file  true/false
)
```

ExportNMFData [NdExplorer Manager]

Use: Export the solution in NMF format.

Command: File > Export SYZ Data

Syntax:

```
ExportNMFData
"DesignName", // Design name. Can be left blank, if loading solution from a file.
true/false,        // true - solution loaded from file, false- loaded from design
"Name",           // If loading from design this is the solution name, else this is the
                  // full path of the file from which the solution is loaded
"ExportFile",      // full path of file to export to
Array("NAME:Frequencies"), // optional, if none defined all frequencies are used
Array("NAME:NMFOptions",            //  Export NMF options object
"DataTypes:=", Array("S"),         // DataTypes can be "S", "Y","Z", "G", and "Z0",
                                   // for S , Y, Z matrix, Gamma and Z0 (zero)
"DisplayFormat:=",  "MA",          // DisplayFormat "MA", "RI", "DB"
"FileType:=", "",                   // Export File Type
                        // 2 - Spreadsheet(*.tab)
                        // 3 - Touchstone(*.sNp)
                        // 4 - Citifile(*.cit)
                        //  6 - Neutral format(*.nmf)
                        //  7 - Matlab format(*.m)
"Renormalize:=", false,             //  Renormalize   true/false
"RefImpedance:=",50,                // Reference Impedance
"Precision:=", 8,                    //  Number of digits Precision
"Variables:=", ARRAY("FF", "cap", "Rs")   // Array of variables
"Variations:=", ARRAY("", "", "")   // Array of variations to export solutions for
Array("NAME:ConstantVars")        // Array of variables that are constant, can be
empty
Array("NAME: DependentVars")      // Array of variables that are dependent, can be
empty
"MatrixSize:=", 2,                    // Matrix size, optional (used in nmf file header)
"CreateNPortModel:=",   true   // Create a model based on the exported file  true/false
```

)

Script and Library Scripts

The definition manager provides access to materials in a Designer project. The manager object is accessed via the definition manager.

```
Set oDefinitionManager = oProject.GetDefinitionManager()
```

The script and library script commands are listed below.

AddScript

EditScript

ExportScript

RemoveScript

ModifyLibraries

AddScript

Use: Add Script in the script definition manager

Command: None

Syntax: `AddScript Array(<AddScriptArray>)`

Return Value: None

Parameters:

`<AddScriptArray>`

Array("NAME<ScriptName>", <string>,

<ScriptLanguage>,<string> ("vbscript" or "javascript")

<ScriptText>, <string> // text of script

Example:

```
oDefinitionManager.AddScript Array("NAME:MyScript", _
"ScriptLang:=", "vbscript", _
"ScriptText:=", "MsgBox(" & Chr(34) & "Hello World" & Chr(34) & ")")
```

EditScript

Use: Edit Script in the script definition manager

Command: None

Syntax: `EditScript <OriginalName>,Array("NAME:<NewName>", < ScriptText >)`

Return Value: None

Parameters:

`<OriginalName>`

`Type: <String>`

`Name of the script before editing.`

```
<NewName>
Type: <String>
New name for the script.
<ScriptText>
Type: <string> // text of script
```

Example:

```
oDefinitionManager.EditScript "myscript",
Array("NAME:myscript", "ScriptLang:=",  _
  "vbscript", "ScriptText:=", _
"MsgBox(" & Chr(34) & "Hello Again" & Chr(34) & ")")
```

ExportScript

Use: Export to Library in the script definition manager

Command: None

Syntax: `ExportScript <ExportData>,<Library location>`

Return Value: None

Parameters:

```
<ExportData>
  Array("NAME:<LibraryName>",<ScriptName>,<ScriptName>,…)
```

Example:

```
oProject.ExportComponent Array("NAME:mylib", "myscript"),
"PersonalLib"
```

RemoveScript

Use: Remove Script in the script definition manager

Command: None

Syntax:

```
RemoveScript <ScriptName>,<IsProjectScript>,
<LibraryName>,<LibraryLocation>
```

Return Value: None

Parameters:

```
<ScriptName>
Type: <string>
<IsProjectScript>
Type: <bool>
<LibraryName>
Type: <string>
<LibraryLocation>
```

```
Type: <string>
```

Example:

```
oDefinitionManager.RemoveScript "myscript", true,
"Local", "Project"
```

ModifyLibraries

Use: Configure Libraries on the Tools menu

Command: None

Syntax: `ModifyLibraries <DesignName>,Array(<ConfigLibArray>)`

Return Value: None

Parameters:

```
<DesignName>
Type: <string>
<ConfigLibArray>
Array("NAME:<LibraryType>,<ConfiguredLib>,<Config-
uredLib>,…),…
<ConfiguredLib> // blank to leave unchanged
<DefinitionType>
Array("<libraryname >","<libraryname>",…)
```

Example:

```
oDefinitionManager.ModifyLibraries "MyCircuit", _
Array("NAME:PersonalLib"), _
Array("NAME:UserLib"), _
Array("NAME:SystemLib", _
"Symbols:=", Array( "Circuit Elements", "Symbols", _
"ParamExtraElements\PE_Symbols", _
"Vendor Elements\Nonlinear"))
```

29 Definition Editor Script Commands

The Definition Editor controls the use of materials and scripts in a Designer project. Symbol editor script commands and footprint editor script commands are accessed using the Definition Editor.

```
Set oDefinitionEditor = oProject.SetActiveDefinitionEditor("SymbolEditor", "MySymbol")
Set oDefinitionEditor = oProject.SetActiveDefinitionEditor("FootprintEditor", "MyFootprint")
```

The topics for this section include:

Symbol Editor Scripts

Symbol editor script commands are accessed with a definition editor.

```
Set oDefinitionEditor = oProject.SetActiveDefinitionEditor("SymbolEd-
itor", "MySymbol")
```

The symbol editor script commands are listed below.

AlignHorizontal (Symbol Editor)

Use: Align items horizontally

Syntax:

```
AlignHorizontal(
Array("NAME:Selections", _
      "Page:=", page number, _  // [Opt=1] Page number
      "Selections:=", IDs to modify)),
Array("NAME:AlignParameters", _
"Disconnect:=", bool  _  // [Opt=0] Should wires disconnect
```

```
"Rubberband:=", bool) _  // [Opt=1] Should wires staircase
// Note: Alignment occurs relative to the first item in
ids
```

AlignVertical (Symbol Editor)

Use: Align items vertically

Syntax:

```
AlignVertical(
Array("NAME:Selections", _
      "Page:=", page number, _  // [Opt=1] Page number
      "Selections:=", IDs to modify)),
Array("NAME:AlignParameters", _
"Disconnect:=", bool  _  // [Opt=0] Should wires disconnect
"Rubberband:=", bool) _  // [Opt=1] Should wires staircase
                      // Note: Alignment occurs relative
                              to the first item in ids
```

ChangeProperty (Symbol Editor)

Use: Changes to properties are scripted using the **ChangeProperty** command. This command can be executed by the **oEditor** to change editor properties, by the **oDesign** to change design level properties, and by the **oProject** to change project level properties. The command can be used to create, edit, and/or remove properties. In Designer, only Variable and Separator properties can be deleted.

Command: None

Syntax: `ChangeProperty  Array("Name:AllTabs", <PropTabArray>, <PropTabArray>, ...)`

Return Value: None

Parameters:

`<PropTabArray>`

Array("Name:<PropTab>",

<PropServersArray>,

<NewPropsArray>,

<ChangedPropsArray>,

<DeletedPropsArray>)

<PropServersArray>

Array(“Name:PropServers”, <PropServer>,
<PropServer>, ...)

<NewPropsArray>
Array(“Name:NewProps”, <PropDataArray>,
<PropDataArray>,...)

<ChangedPropsArray>
Array(“Name:ChangedProps”,<PropDataArray>,
<PropDataArray>, ...)

<DeletedPropsArray>
Array(“Name:DeletedProps”, <PropName>,
<PropName>, ...)
OR (for PropDisplay deletions only)
Array("Name:DeletedProps",<PropDataArray>,
<PropDataArray>, ...)

<PropDataArray>
Array("NAME:<PropName>",
“PropType:=”, <PropType>,
“NewName:=”, <string>,
“Description:=”, <string>,
“Callback:=”, <string>,
“NewRowPosition:=”, <int>,
“ReadOnly:=”, <bool>,
“Hidden:=”, <bool>,
<PropTypeSpecificArgs>)OR (for PropDisplays only)
Array("Name:<PropName>",<PropDisplayData>)

<PropDisplayData>
for adding, changing, deleting PropDisplays
<PropDisplayAttributes>
for changing PropDisplays only

<PropDisplayNewAttributes>

<PropDisplayAttributes>

Layer & Location only used for PropDisplays in layout

For adding PropDisplays, this will add a single PropDisplay with attributes as shown; if an attribute is missing, a default value will be assigned. Adding PropDisplay to schematic with attributes that are identical to one already existing there will fail without an error message.

For deleting PropDisplays, these attributes are used to identify an existing PropDisplay to delete. If there doesn't exist a PropDisplay that matches the given attributes, then nothing will be deleted. If multiple PropDisplays match the given attributes, then all of them will be deleted. If an attribute is missing, then all PropDisplays match that missing attribute. For example, if Layer is missing, then PropDisplays on all layers that match the remaining given attributes will be deleted.

For changing PropDisplays, these attributes are used to identify an existing PropDisplay to change. If no PropDisplay matching the attributes is found, no changes will be made. If multiple PropDisplays match the attributes, all of them will be changed. If an attribute is missing, it matches all PropDisplays. For example, to change the format of PropDisplays that are on the bottom, but have any layer, style or format to show the name only, this command should have Location set to "Bottom" and all other attributes omitted.

"Format:=", <PropDisplayType>,

"Location:=", <PropDisplayLocation>,

"Layer:=", <string>,

"Style:=", <string>

<PropDisplayNewAttributes>

NewLayer & NewLocation only used for PropDisplays in layout

For changing PropDisplays, these attributes are used to identify which attributes to change and what the new value is. If the attribute should not be changed, the corresponding entry should be omitted.

"NewName:=", <string>,

"NewFormat:=", <PropDisplayType>,

"NewLocation:=", <PropDisplayLocation>,

"NewLayer:=", <string>,
"NewStyle:=", <string>

<PropDisplayType>
Type: string
Identifies the format of PropDisplay.
"Name"
"Value"
"NameAndValue"
"EvaluatedValue"
"NameAndEvaluatedValue"

<PropDisplayLocation>
Type: string
Identifies where PropDisplay is located with respect to object

"Left"
"Top"
"Right"
"Bottom"
"Custom"

<PropType>
Type: string
Identifies the type of property when a new property is added. In Designer, only separator properties and variable properties can be added.

"SeparatorProp"
"VariableProp"
"TextProp”
"NumberProp"
"ValueProp"
"CheckboxProp"
"MenuProp”
"PointProp"

"VPointProp"

"ButtonProp"

NewName

Specify the new name of a property if the property's name is being edited. In Designer, the name can only be changed for separators and variables.

Description

Specify a description of the property. In Designer, the description can only be changed for separators and variables.

Callback

Specify the name of the script callback to be run when the property value is changed.

NewRowPosition

Used to reorder rows in the **Property** dialog box. In Designer, this only applies to the **Project>Project Variables** panel and the **Designer>Design Properties** panel. Specify the new zero-based row index of the variable or separator.

ReadOnly

Used to mark a property as "read only" so it can not be modified. In Designer, this flag can only be set for variables and separators.

Hidden

Used to hide a property so it can not be viewed outside of the **Property** dialog box. In Designer, this flag can only be set for variables and separators.

<PropTypeSpecificArgs>

SeparatorProp: no arguments

TextProp: "Value:=", <string>

NumberProp: "Value:=", <double>

ValueProp: "Value:=", <value>

CheckboxProp: "Value:=", <bool>

MenuProp: "Value:=", <string>

PointProp"X:=", <double>, "Y:=", <double>

VPointProp: "X:=", <value>, "Y:=", <value>

Material Button: “Material:=”, <string>
Color Button: "R:=",<int>,"G:=",<int>,"B:=",<int>
Transparency Button:“Value:=”, <double>

<PropTypeSpecificArgs> for MenuProps
Syntax for NewProps array: "AllChoices:=",
<"choice1,choice2,..."> or <Array("choice1" "choice2", ...)>,
"Value:=", <string>
Syntax for ChangedProps array: "Value:=", <string>

<PropTypeSpecificArgs> for VariableProps
Syntax:
“Value:=”, <value>, <OptimizationFlagsArray>,
<TuningFlagsArray>, <SensitivityFlagsArray>,
<StatisticsFlagsArray>

Parameters:
<OptimizationFlagsArray>
Array("NAME:Optimization",
"Included:=", <bool>,
"Min:=", <value>,
"Max:=", <value>)

<TuningFlagsArray>
Array("NAME:Tuning",
"Included:=", <bool>,
"Step:=", <value>,
"Min:=", <value>,
"Max:=", <value>)

<SensitivityFlagsArray>
Array("NAME:Sensitivity",
"Included:=", <bool>,
"Min:=", <value>,
"Max:=", <value>,

"IDisp:=", <value>)

<StatisticsFlagsArray>
Array("NAME:Statistical",
"Included:=", <bool>,
"Dist:=", <Distribution>,
"StdD:=", <value>,
"Min:=", <value>,
"Max:=", <value>,
"Tol:=", <string>)

<Distribution>
Type: string
Value should be **"Gaussian"** or **"Uniform"**

StdD
Standard deviation.

Min
Low cut-off for the distribution.

Max
High cut-off for the distribution.

Tol
Tolerance for uniform distributions. Format is **"<int>%"**.
Example: **"20%"**.

Example:

Adding a new project level variable **"$width"**:

```
oProject.ChangeProperty Array("NAME:AllTabs",_
Array("NAME:ProjectVariableTab",_
Array("NAME:PropServers", "ProjectVariables"),_
Array("NAME:NewProps",_
Array("NAME:$width",_
```

```
"PropType:=", "VariableProp",_
"Value:=", "3mm",_
"Description:=", "my new variable"))))
```

Example: Deleting the design level variable "height":

```
oDesign.ChangeProperty Array("NAME:AllTabs",_
Array("NAME:LocalVariableTab",_
Array("NAME:PropServers", "DefinitionParameters"),_
Array("NAME:DeletedProps", "height"))
```

Changing a property's value. If the following command were executed, then the value of the property "XSize" of the PropServer "Box1:CreateBox:1" on the "Geometry3DCmdTab" tab would be changed. (oEditor is the Geometry3D editor in Designer.)

```
oEditor.ChangeProperty Array("NAME:AllTabs",_
Array("NAME:Geometry3DCmdTab",_
Array("NAME:PropServers","Box1:CreateBox:1"),_
Array("NAME:ChangedProps",_
Array("NAME:XSize", "Value:=", "1.4mil"))))
```

Example: Changing a property's value. If the following command were executed, then the values of Callback and L on the PassedParameterTab would be changed.

```
oEditor.ChangeProperty Array("NAME:AllTabs", _
Array("NAME:PassedParameterTab", _
Array("NAME:PropServers",  "CHOKE2"), _
Array("NAME:ChangedProps", _
Array("NAME:L", "Callback:=", "ac", "OverridingDef:=",
true), _
Array("NAME:L", "Value:=", "1nH"))))
```

Example: Changing the Company Name, Design Name, the background color, and the Axis scaling in a Report.

```
Set oProject = oDesktop.SetActiveProject("wgcombiner")
```

```
Set oDesign = oProject.SetActiveDesign("DesignerDesign2")
Set oModule = oDesign.GetModule("ReportSetup")
oModule.ChangeProperty Array("NAME:AllTabs",
Array("NAME:Header", _ Array("NAME:PropServers", "XY
Plot1:Header"), _
Array("NAME:ChangedProps", Array("NAME:Company Name", _
"Value:=", "My Company"))))
oModule.ChangeProperty Array("NAME:AllTabs",
Array("NAME:Header",_ Array("NAME:PropServers", "XY
Plot1:Header"), _
Array("NAME:ChangedProps", Array("NAME:Design Name", _
"Value:=", "WG Combiner"))))
oModule.ChangeProperty Array("NAME:AllTabs",
Array("NAME:General",_ Array("NAME:PropServers", "XY
Plot1:General"), _
Array("NAME:ChangedProps", Array("NAME:Back Color", _
"R:=", 128, "G:=", 255, "B:=", 255))))
oModule.ChangeProperty Array("NAME:AllTabs",
Array("NAME:Axis", _ Array("NAME:PropServers", "XY
Plot1:AxisX"), _
Array("NAME:ChangedProps", Array("NAME:Axis Scaling", _
"Value:=", "Log"))))
```

Example:

Changing a property's value. Note that the AllChoices parameter is only used when the MenuProp is being added. Also note that either a string of choices separated by commas or an Array("choice1", "choice2", "choice3") works for the AllChoices parameter.

```
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
oEditor.ChangeProperty Array("NAME:AllTabs",
Array("NAME:PassedParameterTab", Array("NAME:PropServ-
ers",  _
  "CompInst@CAP_;2"), Array("NAME:NewProps",
Array("NAME:xxxx", "PropType:=",  _
  "MenuProp", "AllChoices:=", Array("aa", "bb", "cc",
"dd"), "Value:=", "bb"))))
```

CloseEditor (Symbol Editor)

Use: Closes the specified editor.

Command: None

Syntax: `CloseEditor (VARIANT ptDelta)`

Return Value: None

CreateArc (Symbol Editor)

Use: Draws an arc in the symbol editor

Command: None

Syntax: `CreateArc`

Return Value: None

Example:

```
oDefinitionEditor.CreateArc Array("NAME:ArcData", _
"X:=", -0.004318, "Y:=", -0.00127, _
"Radius:=", 0.00297299377732279, _
"StartAng:=", 1.9195673303788, _
"EndAng:=", 3.32144615338227, _
"Id:=", 10), _
Array("NAME:Attributes", "Page:=", 1)
```

CreateCircle (Symbol Editor)

Use: Draws a circle in the symbol editor

Command: None

Syntax: `CreateCircle`

Return Value: None

Example:

```
oDefinitionEditor.CreateCircle Array("NAME:CircleData", _
"X:=", -0.004572, "Y:=",  -0.000508, _
"Radius:=", 0.001778, "Id:=", 12), _
Array("NAME:Attributes", "Page:=", 1)
```

CreateLine (Symbol Editor)

Use: Draws a line in the symbol editor

Command: None

Syntax: `CreateLine`

Example:

```
oDefinitionEditor.CreateLine Array("NAME:LineData", _Re-
turn Value:None
"Points:=", Array( "(0.001778, -0.001270)", _
"(0.004572, 0.002794)", "(0.003048, 0.003556)"), _
"Id:=", 14), Array("NAME:Attributes", "Page:=", 1)
```

CreatePin (Symbol Editor)

Use: Draws a pin in the symbol editor

Command: None

Syntax: `CreatePin`

Return Value: None

Example:

```
oDefinitionEditor.CreatePin Array("NAME:PinData", _
"Name:=", "newpin3"), Array("NAME:PinParams", _
"X:=",  -0.00762, "Y:=", 0.00254, "Angle:=", 0, "Flip:=",
false)
```

CreatePolygon (Symbol Editor)

Use: Draws a polygon in the symbol editor

Command: None

Syntax: `CreatePolygon`

Return Value: None

Example:

```
oDefinitionEditor.CreatePolygon Array("NAME:PolygonData",
_
"Points:=", Array( "(0.004826, -0.000508)", "(0.003048, -
0.003048)", _
"(0.006858, -0.003302)"), "Id:=",16), _
Array("NAME:Attributes", "Page:=", 1)
```

CreateRectangle (Symbol Editor)

Use: Draws a rectangle in the symbol editor

Command: None

Syntax: `CreateRectangle`

Return Value: None

Example:

```
oDefinitionEditor.CreateRectangle Array("NAME:RectData",
_
"X1:=", -0.001524, "Y1:=",  0.000508, _
"X2:=", 0.002286, "Y2:=", -0.002032, "Id:=", 18), _
Array("NAME:Attributes", "Page:=", 1)
```

CreateText (Symbol Editor)

Use: Draws text in the symbol editor

Command: None

Syntax: `CreateText`

Return Value: None

Example:

```
oDefinitionEditor.CreateText Array("NAME:TextData", _
"X:=", 0.001524, "Y:=", 0.00127, _
"Text:=","My text", "Id:=", 20), Array("NAME:Attributes",
"Page:=", 1)
```

Cut (Symbol Editor)

Use: Cut page selection.

Command: None

Syntax: `Cut(PageNumber, Selections)`

Return Value: None

Example:

```
oDefinitionEditor.CreateText Array("NAME:Selections", _
"Page:=", page number, _  // [Opt=1] Page number
"Selections:=", IDs to modify)) 1)
```

GetProperties (Symbol Editor)

Use: Gets a list of all the properties belonging to a specific **PropServer** and **PropTab**. This can be executed by the **oProject**, **oDesign**, or **oEditor** objects.

Command: None

Syntax: `GetProperties( <PropTab>, <PropServer> )`

Return Value: Variant array of strings – the names of the properties belonging to the prop server.

Example:
```
Dim all_props
all_props = oDesign.GetProperties("BaseElementTab",_
"rect_1")
```

GetPropertyValue (Symbol Editor)

Use: Gets the value of a single property. This can be executed by the **oProject**, **oDesign**, or **oEditor** objects.

Command: None

Syntax: `GetPropertyValue(<PropTab>, <PropServer>, <PropName>)`

Return Value: String representing the property value.

Example:
```
value_string = _
oEditor.GetPropertyValue("BaseElementTab",_
"rect_1", "Name")
```

Redo (Symbol Editor)

Use: Redo the last operation

Command: Edit>Redo

Syntax: `Redo`

Return Value: None

Parameters: `None`

Example: `oDesign.Redo`

RemovePort (Symbol Editor)

Use: Selected pins are changed to vias. Selected edge ports are removed.

Syntax: `RemovePort <NAME:elements", <object_name> ... // objects to be removed`

Return Value: None.

Parameters: `<object_name>`

Type: <String>

Example:
```
oEditor.RemovePort Array("NAME:elements", "via_195",
"Port1")
```

SelectAll (Symbol Editor)

Use: Select all elements in the symbol editor.

Command: **None**.

Syntax:
```
SelectAll()
```

Parameters: None

Example:
```
Dim removedDefs
removedDefs = oDefinitionEditor.SelectAll()
```

SendToBack (Symbol Editor)

Use: Changes the drawing for the symbol so that the specified objects are drawn behind other overlapping objects.

Command: Draw > Send To Back

Syntax:
```
SendToBack Array("NAME:Selections", "Selections:=",
Array (<Object>, <Object>, ...))
```

Return Value: None

Parameters:
```
<Object>
<string>  // object to send to the back
```

Example:
```
oDefinitionEditor.SendToBack Array("NAME:Selections",
"Selections:=", Array(  "SchObj@10"))
```

SetPropertyValue (Symbol Editor)

Use: Set a property.

Command: **None**.

Syntax:
```
SetPropertyValue(tab, item, propname, value)
```

Parameters: None

Example:
```
Dim removedDefs
removedDefs = oDefinitionEditor.SetPropertyValue(string
tab,           // Tab with the property
string item, // Name of the object
string propname, // Name of the property
string value) // The new value of the prop
```

ToggleViaPin (Symbol Editor)

Use: Selected pins are changed to vias. Selected vias are changed to pins.

Syntax:
```
ToggleViaPin <NAME:elements", <object_name> ... //
objects to be toggled
```

Return Value: None.

Parameters:
```
<object_name>
```
Type: <String>

Example:
```
oEditor.ToggleViaPin Array("NAME:elements", "via_195")
```

Undo (Symbol Editor)

Use: Undo the last operation

Command: Edit>Undo

Syntax:
```
Undo
```

Return Value: None

Parameters:
```
None
```

Example:
```
oDesign.Undo
```

ZoomToFit (Symbol Editor)

Use: Set the current symbol zoom to fit the contents of the currently visible page

Command: None

Syntax:
```
ZoomToFit()
```

Return Value: None

Footprint Editor Scripts

Footprint editor script commands are accessed with a definition editor.

```
Set oDefinitionEditor = oProject.SetActiveDefinitionEditor("Footprin-
tEditor", "MyFootprint")
```

The footprint editor script commands are listed below.

AddLayer(Footprint Editor)

AddStackupLayer (Footprint Editor)

ChangeLayers (Footprint Editor)

ChangeOptions (Footprint Editor)

CloseEditor (Footprint Editor)

CreateCircle (Footprint Editor)
CreateCircleVoid (Footprint Editor)
CreateEdgePort (Footprint Editor)
CreateLine (Footprint Editor)
CreateLineVoid (Footprint Editor)
CreateMeasure (Footprint Editor)
CreatePolygonVoid (Footprint Editor)
CreatePin (Footprint Editor)
CreatePolygon (Footprint Editor)
CreateRectangle (Footprint Editor)
CreateText (Footprint Editor)
CreateVia (Footprint Editor)
Duplicate (Footprint Editor)
Edit (Footprint Editor)
EraseMeasurements (Footprint Editor)
FlipHorizontal (Footprint Editor)
FlipVertical (Footprint Editor)
GetAllLayerNames (Footprint Editor)
GetLayerInfo (Footprint Editor)
GetProperties (Footprint Editor)
GetStackupLayerNames (Footprint Editor)
Intersect (Footprint Editor)
Move (Footprint Editor)
PageSetup (Footprint Editor)
RemoveLayer (Footprint Editor)
RemovePort (Footprint Editor)
Rotate (Footprint Editor)
Save (Footprint Editor)
SetActiveDefinitionEditor (Footprint Editor)
SetPropertyValue (Footprint Editor)
Subtract (Footprint Editor)
ToggleViaPin (Footprint Editor)
Unite (Footprint Editor)
ZoomToFit (Footprint Editor)

AddLayer (Footprint Editor)

Use: Adds a layer.

Command: Add Layer in a layout or footprint definition.

Syntax:

```
AddLayer Array ("NAME:layer",
"Name:=", <LayerName>,
"Type:=", <Type>,
"Top Bottom:=" , <TB>,
"Color:=", <ColorNumber>,
"Pattern:=", <FillPattern>,
"Visible:=", <Visibility>,
"Selectable:=", <Selectability>,
"Locked:=", <Locked>)
```

Return Value: None.

Parameters:

<LayerName>
Type: <String>
<Type >
Type: <String> (Same choices as in the layer dialog.)
<TB >
Type: <String> Choices are "Top"|"Neither"|"Bottom"|"Template"|"Invalid"
<ColorNumber>
Type: integer representing rbg in hex
<FillPattern>
Type: integer
<Visibility>
true | false
<Selectability>
true | false
<Locked>
true | false

Example:

```
oDefinitionEditor.AddLayer Array("NAME:layer", "Name:=",
"junk footprint", _
 "Type:=", "soldermask", "Top Bottom:=", "neither",
"Color:=", 4144959, _
"Pattern:=",  1, "Visible:=", true, "Selectable:=", true,
"Locked:=", false)
```

AddStackupLayer (Footprint Editor)

Use: Adds a stackup layer.

Command: Add Stackup Layer in a layout or footprint definition.

Syntax:

```
AddStackupLayer Array ("NAME:layer",
"Name:=", <LayerName>,
"Type:=", <Type>,
"Top Bottom:=" , <TB>,
"Color:=", <ColorNumber>,
"Pattern:=", <FillPattern>,
"Visible:=", <Visibility>,
"Selectable:=", <Selectability>,
"Locked:=", <Locked>
"ElevationEditMode:=", <Elevation>, <SublayerArray>)
```

Return Value: None.

Parameters:

<LayerName>

Type: <String>

<Type >

Type: <String> (Same choices as in the layer dialog.)

<TB >

Type: <String> Choices are "Top"|"Neither"|"Bottom"|"Template"|"Invalid"

<ColorNumber>

Type: integer representing rbg in hex

<FillPattern>

Type: integer

<Visibility>

true | false

<Selectability>

true | false

<Locked>

true | false

<Elevation>

Type: <String> Choices are "snap to middle" | "snap to top" | "snap to bottom" | "none"

<SublayerArray>

Type: Array(("NAME:Sublayer", "Thickness:=", <Thickness>, "LowerElevation:=", <Elevation>, "Roughness:=" , <Roughness>, "Material:=", <MaterialName)

<Thickness>

Type: <String> containing number and units (e.g. "0mil")

<Elevation>

Type: <String> containing number and units (e.g. "0mil")

<Roughness>

Type: <String> containing number and units (e.g. "0mil")

<Material>

Type: <String>

```
oEditor.AddStackupLayer Array("NAME:stackup layer",
"Name:=", "MyLayer2", _
  "Type:=", "Signal", "Top Bottom:=", "neither",
"Color:=", 127, "Pattern:=", 7, _
"Visible:=", true, "Selectable:=", true, "Locked:=",
false, "ElevationEditMode:=", "none",_
Array("NAME:Sublayer", "Thickness:=", "25mil", "LowerEle-
vation:=", "0mil", _
"Roughness:=", "0mil", "Material:=", "Al2_O3_ceramic"),
"UseR:=", true, _
 "RMdl:=", "Huray", "NR:=", "2mil", "HRatio:=", 2.8)
```

ChangeLayers (Footprint Editor)

Use: Causes changing of the layers.

Command: None.

Syntax:

```
ChangeLayers
  Array("NAME:layers",
  <full_layer_description>,  // 1st layer
      <full_layer_description>,  // 2nd layer
      …) // etc
```

Return Value: None.

Parameters:

```
The LayoutComp's ID
```

Example:

```
oEditor.ChangeLayers Array("NAME:layers", "Mode:=", "Lam-
inate", Array("NAME:stackup layer", "Name:=",  _
  "Top", "ID:=", 7, "Type:=", "signal", "Top Bottom:=",
"neither", "Color:=",  _
```

```
  32512, "Transparency:=", 95, "Pattern:=", 1, "Vis-
Flag:=", 31, "Locked:=", false, "DrawOverride:=", 0,
"ElevationEditMode:=",  _
  "none", Array("NAME:Sublayer", "Thickness:=", "0mil",
"LowerElevation:=",  _
  "124.992125984252mil", "Roughness:=", "0mil", "Mate-
rial:=", "copper", "FillMaterial:=",  _
  "FR4_epoxy"), "Usp:=", true, Array("NAME:Sp", "Sn:=",
"HFSS", "Sv:=", "so(si=1)"), Array("NAME:Sp", "Sn:=",  _
  "PlanarEM", "Sv:=", "so(ifg=1, vly=1)"), "UseEtch:=",
true, "UseR:=", true), Array("NAME:stackup layer",
"Name:=",  _
  "Dielectric", "ID:=", 0, "Type:=", "dielectric", "Top
Bottom:=", "neither", "Color:=",  _
  127, "Pattern:=", 1, "VisFlag:=", 31, "Locked:=",
false, "DrawOverride:=", 0, "ElevationEditMode:=",  _
  "none", Array("NAME:Sublayer", "Thickness:=", "62mil",
"LowerElevation:=",  _
  "62.992125984252mil", "Roughness:=", "0mil", "Mate-
rial:=", "FR4", "FillMaterial:=",  _
  "FR4_epoxy")), Array("NAME:stackup layer", "Name:=",
"Ground", "ID:=", 6, "Type:=",  _
  "ground", "Top Bottom:=", "bottom", "Color:=", 4144959,
"Pattern:=", 1, "VisFlag:=",  _
  31, "Locked:=", false, "DrawOverride:=", 0, "Eleva-
tionEditMode:=", "none", Array("NAME:Sublayer", "Thick-
ness:=",  _
  "0mil", "LowerElevation:=", "62.992125984252mil",
"Roughness:=", "0mil", "Material:=",  _
  "copper", "FillMaterial:=", "FR4_epoxy"), "Neg:=",
true, "UseR:=", true), Array("NAME:stackup layer",
"Name:=",  _
  "Dielectric0", "ID:=", 9, "Type:=", "dielectric", "Top
Bottom:=", "neither", "Color:=",  _
  8421376, "Pattern:=", 1, "VisFlag:=", 31, "Locked:=",
false, "DrawOverride:=",  _
  0, "ElevationEditMode:=", "none", Array("NAME:Sub-
layer", "Thickness:=", "1.6mm", "LowerElevation:=",  _
  "0", "Roughness:=", "0", "Material:=", "FR4")),
Array("NAME:stackup layer", "Name:=",  _
```

```
  "Signal", "ID:=", 10, "Type:=", "signal", "Top Bot-
tom:=", "neither", "Color:=",  _
  16512, "Pattern:=", 1, "VisFlag:=", 31, "Locked:=",
false, "DrawOverride:=", 0, "ElevationEditMode:=",  _
  "none", Array("NAME:Sublayer", "Thickness:=", "0mm",
"LowerElevation:=", "0", "Roughness:=",  _
  "0", "Material:=", "copper", "FillMaterial:=", "FR4_ep-
oxy")), Array("NAME:layer", "Name:=",  _
  "Measures", "ID:=", 8, "Type:=", "measures", "Top Bot-
tom:=", "neither", "Color:=",  _
  4144959, "Transparency:=", 0, "Pattern:=", 1, "Vis-
Flag:=", 31, "Locked:=", false, "DrawOverride:=",  _
  0), Array("NAME:layer", "Name:=", "Rats", "ID:=", 3,
"Type:=", "rat", "Top Bottom:=",  _
  "neither", "Color:=", 16711680, "Pattern:=", 1, "Vis-
Flag:=", 0, "Locked:=",  _
  false, "DrawOverride:=", 0), Array("NAME:layer",
"Name:=", "Errors", "ID:=", 4, "Type:=",  _
  "error", "Top Bottom:=", "neither", "Color:=", 255,
"Pattern:=", 1, "VisFlag:=",  _
  31, "Locked:=", true, "DrawOverride:=", 0),
Array("NAME:layer", "Name:=", "Symbols", "ID:=",  _
  5, "Type:=", "symbol", "Top Bottom:=", "neither",
"Color:=", 8323199, "Pattern:=",  _
  1, "VisFlag:=", 31, "Locked:=", false, "DrawOver-
ride:=", 0), Array("NAME:layer", "Name:=",  _
  "Assembly Top", "ID:=", 2, "Type:=", "assembly", "Top
Bottom:=", "top", "Color:=",  _
  16711680, "Pattern:=", 1, "VisFlag:=", 31, "Locked:=",
false, "DrawOverride:=",  _
  0), Array("NAME:layer", "Name:=", "Silkscreen Top",
"ID:=", 1, "Type:=", "silkscreen", "Top Bottom:=",  _
  "top", "Color:=", 65280, "Pattern:=", 1, "VisFlag:=",
31, "Locked:=", false, "DrawOverride:=",  _
  0))
```

ChangeOptions (Footprint Editor)

Use: Changes options for an existing layout. (Does not change global options specified in the registry.) Only those options being changed need to be specified. Options not specified are not affected.

Command: None.

Syntax: `ChangeOptions Array("NAME:options",<OptionData>,…)`

Return Value: None

Parameters: `<OptionData>` can be of varying forms:

Type: <String>

Type: integer

Type: Array(float, float, float, float)

Example:

```
oEditor.ChangeOptions Array("NAME:options", _
"MajorSize:=", "20", _
   "MinorSize:=", "1", "MajorColor:=",  8421376, _
"MinorColor:=", 16776960,  _
   "ShowGrid:=", false, "PageExtent:=", _
Array(  -0.3, -0.1, 0.1, 0.1),   _
   "background color:=", 4194368, _
"DefaultToSketchMode:=",  true, _
   "fillMode:=", false, "PixelSnapTolerance:=", 22, _
"SnapTargetVertex_on:=",  _
   false, "SnapTargetEdgeCenter_on:=", false, _
"SnapTargetObjCenter_on:=", _
   true, "SnapTargetEdge_on:=", true, _
"SnapTargetElecConnection_on:=", true, _
   "SnapTargetIntersection_on:=",  true, _
"SnapTargetGrid_on:=", false, _
   "SnapSourceVertex_on:=", false, _
"SnapSourceEdgeCenter_on:=",  false,  _
   "SnapSourceObjCenter_on:=", true, _
"SnapSourceEdge_on:=", true, _
   "SnapSourceElecConnection_on:=",  true, _
"ConstrainToGrid:=", false _
"defaultholesize:=", "5mil",  _
   "show connection points:=", true, _
```

```
"display vertex labels:=", true, _
"NetColor:=", 8421440,  _
   "rectangle description:=", 1, "snaptoport:=", false, _
"sym footprint scaling:=", _
   0.385, "primary selection color:=",  32768, _
"secondary selection color:=", _
   22784, "preview selection:=", true, "anglesnap:=", _
"59deg", "AllowDragOnFirstClick:=", true, _
"useFixedDrawingResolution:=", true, _
   "DrawingResolution:=",  "0.002mm", "Tol:=", _
Array(3E-009, 1.5E-008, 1E-012))
```

Note: An error will be returned if this script command is not used at the top-level hierarchy.

- Global layout defaults are stored in the registry (Layout\Preferences\)
- Names of registry items were chosen for backwards compatibility and are consistent with adsn, technology file, and scripting naming
- Local options are both script-able and undo-able
- Global options are neither script-able nor undo-able

Option	Description	Installed default	Registry, Scripting Names, Data Type
Arc Drawing Resolution	Controls drawing of segments for arcs - either dynamic and based on current zoom or fixed to specified value	Dynamic	"useFixedDrawingResolution"[1] "DrawingResolution" [2]
Major and Minor Grid lines	Spacing, color, and visibility	The grid is displayed. Spacing based on current default length units. Major grid lines are 10 minor grid lines apart. The line colors are light blue grays, with major grid lines being darker.	"MajorColor" [3] "MinorColor" [3] "MajorSize" [2] "MinorSize" [2] "ShowGrid" [1]
Drawing Extent	Specifies size and coordinates of the layout	Lower left of the layout is (-0.1, -0.1) and the upper right is (0.1, 0.1)	"PageExtent" [4]
Background color	Color of the layout background	white	"background color" [3]
Sketch Mode	Fill patterns and center lines are not drawn.	off	"DefaultToSketchMode" [1]
Solid Mode	Objects are filled in with solid color.	off	"fillMode" [1]
Draw Connection Points	Display pin symbols in the layout.	off	"show connection points" [1]

Draw Rats	Display lines indicating missing physical connections in nets	on	"draw rats" [1]
Label Vertices	Display property text labeling the vertices of selected polygons and lines.	off	"display vertex labels" [1]
Net Color	Color used for highlighting nets	light yellow	"NetColor" [3]
Rectangle Description	Specifies if rectangles are described with two points or center point, width, and height	2 points	"rectangle description" [5]
Default Hole Size	Default size for actual holes in a PC board	25 mil	"defaultholesize" [2]
Align Microwave Components	Snaps components on entry.	on	"snaptoport" [1]
Symbol Footprint Scaling	Used to size symbol footprints placed in layout.	Based on data extent and current length units.	"sym footprint scaling" [6]
Primary and Secondary Selection Colors	Colors used to indicate the first item selected and other items currently selected.	Primary color is red, secondary color is a darker red	"primary selectioncolor" [3] "secondary selection color" [3]
Preview Selection	Highlight the object that would be selected with a left mouse button click in the current location.	off	"preview selection" [1]

Snapping options	Choose snapping sources and targets and the maximum distance (in pixels) for snapping to occur. Constrain edits to grid if desired. **Note**: Off-grid objects are ignored if ConstrainToGrid is asserted.	Snap distance is 20 pixels. Snapping sources are vertex, edge center, object center, & elec. conn. Snapping targets are vertex, edge center, object center, elec. conn, & grid. Edits are constrained to grid.	"PixelSnapTolerance"[7] "SnapTargetVertex_on"[1] "SnapTargetEdgeCenter_on"[1] "SnapTargetObjCenter_on"[1] "SnapTargetEdge_on"[1] "SnapTargetElecConnection_on"[1] "SnapTargetIntersection_on"[1] "SnapTargetGrid_on"[1] "SnapSourceVertex_on"[1] "SnapSourceEdgeCenter_on"[1] "SnapSourceObjCenter_on"[1] "SnapSourceEdge_on"[1] "SnapSourceElecConnection_on"[1] "ConstrainToGrid"[1]
Always Show Merge Layers Dialog	Controls display of the layer merging dialog.	off – Only show dialog when the user must be involved.	"AlwaysShowLayerMergeDlg" [1]
Rotation Increment	During rotation, objects are rotated by multiples of this amount.	Based on current angle units.	"anglesnap" [2]
Allow Drag on first click	Selection and move with one left mouse button click.	false	"AllowDragOnFirstClick" [1]

The footnotes in the table above correspond to the following:

[1] = Integer with a value of 1 for on/true and 0 for off/false

[2] = String containing a amount and units

[3] = Integer representing a Color

[4] = An Array containing (lower left x, lower left y, upper right x, upper right y)

[5] = Integer with a value of 0 for two points, and 1 for center/width/height

[6] = String holding a double.

[7] = Integer

CloseEditor (Footprint Editor)

Use: Closes the active definition editor.

Command: None

Syntax: `CloseEditor`

Return Value: None

Parameters: `None`

Example:

```
oDefinitionEditor.CloseEditor
```

CreateCircle (Footprint Editor)

Use: Draws a circle in the footprint editor

Command: None

Syntax: `CreateCircle`

Return Value: None

Example:

```
oDefinitionEditor.CreateCircle Array("NAME:Contents",
"circleGeometry:=", Array("Name:=",  _
"circle_118", "LayerName:=", "Top", "lw:=", "0mm", _
"x:=", "-3.23020108044147mm", "y:=",
"1.49086199235171mm", _
"r:=", "0.605222899964955mm"))
```

CreateCircleVoid (Footprint Editor)

Use: Creates a circle void and adds it to a particular parent primitive. Returns the name of the newly created object.

Syntax:

```
CreateCircleVoid <circle_void_description>
<circle_void_description>:
    Array("NAME:Contents",
    "owner:=", <object_name>, // parent primitive name
    "circle voidGeometry:=", <circle_geometry>)//defini-
tion
```

Example:

```
oEditor.CreateCircleVoid
  Array("NAME:Contents",
        "owner:=", "rect_4",
        "circle voidGeometry:=",
```

```
Array ("Layer:=",  6,
       "Name:=", "circle void_10",
       "LayerName:=", "Top",
       "lw:=", "0mm",
       "x:=",  "26mm",
       "y:=", "11mm",
       "r:=", "1.41421356237309mm"))
```

CreateEdgePort (Footprint Editor)

Use: Creates an edge port using the specified edges.

Command: Draw > Port > Create

Right-click > Port > Create

Also available through Tool Bar icon

Syntax: `CreateEdgePort`

Array("NAME:Contents",

"edge:=", Array(<edge description>), "edge:=, Array(<edge description>), ...

"external:=", <flag>)

Return Value: Text containing the name of the created edge port. (Returns an empty name if the edge port is not created.)

Parameters: `<edge description> for primitive edges`

"et:=", "pe", "prim:=", <"prim">, "edge:=", <edge#>

<"prim">: text that is the primitive name

<edge#>: integer that is the edge number on the primitive

<edge description> for via edges

"et:=", "pse", "sel:=", <"via">, "layer:=", <layer id>,
"sx:=", <start X location>, "sy:=", <start Y location>, "ex:=", <end X location>,
"ey:=", <end Y location>, "h:=", <arc height>, "rad:=", <radians>

<"via">: text that is the name of the via to use

<layer id>:

an integer that is the id of the layer of the pad of the via to use

<start X location>, <start Y Location>:
doubles that are the X, Y location of the start point of the edge arc

<end X location>, <end Y Location>:
doubles that are the X, Y location of the end point of the edge arc

<arc height>: double giving the height of the edge arc (0 for a straight edge)
<radians>: double giving the arc size in radians (0 for a straight edge)

<flag>
true if the port is an external port
false if the port is an internal port

Example:

```
oDefinitionEditor.CreateEdgePort Array("NAME:Contents",
"edge:=", Array("et:=",
"pe", "prim:=",  "rect_6", "edge:=", 1), "edge:=",
Array("et:=", "pe", "prim:=",
"rect_6", "edge:=", 2), "external:=", true)

oDefinitionEditor.CreateEdgePort Array("NAME:Contents",
"edge:=", Array("et:=",
"pse", "sel:=", "via_4", "layer:=", 4, "sx:=", -0.0015,
"sy:=", 0.0015, "ex:=", -0.0015,
"ey:=", -0.0015, "h:=", 0, "rad:=", 0), "edge:=",
Array("et:=", "pse", "sel:=", "via_4",
"layer:=", 4, "sx:=", 0.0015, "sy:=", 0.0015, "ex:=", -
0.0015, "ey:=", 0.0015, "h:=", 0,
"rad:=",  0), "external:=", true)
```

CreateLine (Footprint Editor)

Use: Draws a line in the footprint editor
Command: None
Syntax: `CreateLine`
Return Value: None

Example:

```
oDefinitionEditor.CreateLine Array("NAME:Contents", _
"lineGeometry:=", Array("Name:=",  _
  "line_160", "LayerName:=", "Top", "lw:=", "0.5mm", _
"endstyle:=", 0, "joinstyle:=", 1, "n:=", 3, _
"x0:=", "2.22709029912949mm", "y0:=",
"0.819053850136697mm", _
"x1:=", "-0.0527859515mm", "y1:=", "0.8894385mm ",
"x2:=", "1.63943274461105mm", "y2:=",
"0.769682056256832mm"))
```

CreateLineVoid (Footprint Editor)

Use: Creates a line void and adds it to a specified as parameter parent primitive. Returns the name of the newly created object.

Syntax:

```
CreateLineVoid <line_void_description>
<line_void_description>:
Array("NAME:Contents",
"owner:=", <object_name>, // parent primitive name
"line voidGeometry:=", <line_geometry>) // definition
```

Example:

```
oEditor.CreateLineVoid
    Array("NAME:Contents",
          "owner:=", "rect_4",
          "line voidGeometry:=",
              Array("Layer:=",  6,
                    "Name:=", "line void_14",
                    "LayerName:=", "Top",
                    "lw:=", "2mil",
                    "endstyle:=",  0,
                    "joinstyle:=", 0,
                    "n:=", 3,
                    "x0:=", "27mm", "y0:=", "5mm",
                    "x1:=", "35mm", "y1:=",  "5mm",
                    "x2:=", "36mm", "y2:=", "9mm"))
```

CreateMeasure (Footprint Editor)

Use: Creates a measurement. Returns the name of the created object.

Syntax:

```
CreateMeasure
    Array("NAME:Contents",
          "MeasurementGeometry:=",
               Array("LayerName:=", <layer_name>,  //
layer
                     "lw:=", <value>, // line width
                    "sx:=", <value>, // start X coordinate
                    "sy:=", <value>, // start Y coordinate
                     "ex:=", <value>, // end X coordinate
                     "ey:=", <value>, // end Y coordinate
   <text_style>)

<text_style> :
   "name:=",   <quoted string>, // its name
   "isPlot:=", <bool>,
   "font:=",   <font_name>,
   "size:=", double,     // size in current units
   "angle:=",  <value>,
   "weight:=", <text_weight>,
   "just:=",   <text_justification>,
   "mirror:=", <bool>,
   "scales:=", <bool>))
```

Example:

```
oEditor.CreateMeasure
    Array("NAME:Contents",
          "MeasurementGeometry:=",
               Array("Layer:=",  0,
                     "Name:=", "Measurement_2",
                     "LayerName:=", "Measures",
                     "lw:=", "0mm",
                     "sx:=", "-32mm",
                     "sy:=", "-13mm",
                     "ex:=", "32mm",
```

```
                                    "ey:=", "-11mm",
                                    "name:=", "<DefaultAnnotation>",
                                    "isPlot:=", false,
                                    "font:=", "Arial",
                                    "size:=", 10,
                                    "angle:=",  "0deg",
                                    "weight:=", 3,
                                    "just:=", 4,
                                    "mirror:=", false,
                                    "scales:=", false))
```

CreatePolygonVoid (Footprint Editor)

Use: Creates a polygon void and adds it to a specified as parameter parent primitive.

Syntax:

```
CreatePolygonVoid <polygon_void_description>
<polygon_void_description>:
Array("NAME:Contents",
"owner:=", <object_name>, // owner name
"poly voidGeometry:=", <polygon_geometry>) // definition
```

Return Value: Returns the name of the newly created object.

Example:

```
oEditor.CreatePolygonVoid
    Array("NAME:Contents",
          "owner:=", "rect_4",
          "poly voidGeometry:=",
              Array("Layer:=", 6,
                    "Name:=", "poly void_18",
                    "LayerName:=", "Top",
                    "lw:=", "0mm",
                    "n:=", 5,
                    "x0:=", "21mm", "y0:=", "9mm",
                    "x1:=", "21mm", "y1:=", "5mm",
                    "x2:=", "24mm", "y2:=", "7mm",
                    "x3:=", "24mm", "y3:=", "7mm",
                    "x4:=", "24mm", "y4:=", "7mm"))
```

CreatePin (Footprint Editor)

Use: Creates a pin in the footprint editor

Command: None

Syntax: `CreatePin`

Return Value: None

Example:

```
oDefinitionEditor.CreatePin Array("NAME:Contents", _
Array("NAME:Port", "Name:=", "Pin_123"), "Rotation:=",
Array( _
"0deg"), "Offset:=", Array("x:=", "-0.310625357087702mm",
"y:=",  _
 "1.4226520434022mm"), "Padstack:=", "NoPad SMT East")
```

CreatePolygon (Footprint Editor)

Use: Draws a polygon in the footprint editor

Command: None

Syntax: `CreatePolygon`

Return Value: None

Example:

```
oDefinitionEditor.CreatePolygon Array("NAME:Contents", _
"polyGeometry:=", Array("Name:=",  _
"poly_164", "LayerName:=", "Top", "lw:=", "0mm", "n:=",
4,
"x0:=", "2.20868387259543mm", "y0:=", "-
1.80972274392843mm", _
"x1:=", "1.43564445897937mm", "y1:=", "-
2.56435642950237mm", _
"x2:=", "3.22099728509784mm", "y2:=", "-
2.02138815075159mm", _
"x3:=", "2.40194355137646mm", "y3:=", "-
2.07660533487797mm"))
```

CreateRectangle (Footprint Editor)

Use: Draws a rectangle in the footprint editor

Command: None

Syntax: `CreateRectangle`

Return Value: None

Example:

```
oDefinitionEditor.CreateRectangle Array("NAME:Contents",
_
"rectGeometry:=", Array("Name:=", "rect_120", _
"LayerName:=", "Top", "lw:=", "0mm", "Ax:=", _
"-3.28541826456785mm", "Ay:=",  _
"0.37731695920229mm", "Bx:=", "-2.42035021074116mm",
"By:=",  _
"-0.294491270324215mm", "ang:=", "0deg"))
```

CreateText (Footprint Editor)

Use: Draws text in the footprint editor

Command: None

Syntax: `CreateText`

Return Value: None

Example:

```
oDefinitionEditor.CreateText Array("NAME:Contents", _
"textGeometry:=", Array("Name:=",  _
  "text_165", "LayerName:=", "Top", _
"x:=", "-2.25469819270074mm", "y:=","-
2.09501106292009mm", _
"ang:=", "0deg", "isPlot:=", true, "font:=",  _
  "RomanSimplex", "size:=", "5mm", "weight:=", 3,
"just:=", 4, _
"mirror:=", false, "text:=", "My text"))
```

CreateVia (Footprint Editor)

Use: Creates a via in the footprint editor

Command: None

Syntax: `CreateVia`

Return Value: Name of the created via.

Example:

```
oDefinitionEditor.CreateVia Array("NAME:Contents",
"name:=", _
"", "vposition:=", _
Array("x:=", "2.28299549780786mm", _
```

```
"y:=", "1.18763139471412mm"), "rotation:=", 0, "overrides
hole:=",  _
false, "hole diameter:=", Array("0.50038mm"), _
"ReferencedPadstack:=", "Template 1mm/0.5mm", _
"highest_layer:=", "Top", "lowest_layer:=", "Top")
```

Duplicate (Footprint Editor)

Use: Duplicates footprint objects.

Command: The specified objects are duplicated by the given count with the specified offset.

Syntax:
```
Duplicate Array("NAME:options", "count:=", <count data>),
Array("NAME:elements", <element names>), Array( <x>, <y>)
```

Return Value: None

Parameters:

`<count data>` is an integer

<element names> is one or more strings with the names of footprint objects

<x> is the x value for the offset

<y> is the y value for the offset

Example:

```
oEditor.Duplicate Array("NAME:options", "count:=", 2),
Array("NAME:elements", "rect_56"), Array( -0.018, 0.017)
```

Edit (Footprint Editor)

Use: Edits an object in the footprint editor

Command: None

Syntax:
```
Edit
```

Return Value: None

Example:

```
oDefinitionEditor.Edit Array("NAME:items",
Array("NAME:item", _
"name:=", "poly118", Array("NAME:contents", _
"polyGeometry:=", Array("Name:=",  _
"poly118", "LayerName:=", "Top", "lw:=", "0mm", "n:=", 5,
"x0:=",  _
"-9.58323911802614mil", "y0:=", "d +
0.000193052692338824", _
```

```
"x1:=", "-9.58323911802614mil", "y1:=", _
"W+d + 0.000193052692338824", "x2:=", "W+d", "y2:=",  _
"0mil", "x3:=", "d", "y3:=", "0mil", "x4:=", "d", "y4:=",
"d"))))
```

EraseMeasurements (Footprint Editor)

Use: Causes erasing of ALL measurements currently present.

Command: None.

Syntax: `EraseMeasurements`

Return Value: None.

Parameters: `None.`

Example: `oEditor.EraseMeasurements`

FlipHorizontal (Footprint Editor)

Use: Flips the selected object horizontally

Command: None

Syntax: `FlipHorizontal`

Return Value: None

Example:

```
oDefinitionEditor.FlipHorizontal Array("NAME:elements",
"poly118"), _
Array(0.00082250994243756, 0.00101975944723128)
```

FlipVertical (Footprint Editor)

Use: Flips the selected object vertically

Command: None

Syntax: `FlipVertical`

Return Value: None

Example:

```
oDefinitionEditor.FlipVertical Array("NAME:elements",
"poly118"), _
Array(0.00082250994243756, 0.00101975944723128)
```

GetAllLayerNames (Footprint Editor)

Use: Informational.

Command: None.

Syntax: `GetAllLayerNames`

Return Value: Array of strings which are the names of all layers in the layout, blackbox, or footprint.

Parameters: `None.`

GetLayerInfo (Footprint Editor)

Use: Informational.

Command: None.

Syntax: `GetLayerInfo<layer_name>`

Return Value: array of strings, as follows:

Type: typename
TopBottomAssociation: "Top"|"Neither"|"Bottom"|"Template"|"Invalid"
Color: integer [representing rgb in hex]
IsVisible: "true"|"false" [true if any type of object below is visible]
IsVisibleShape: true [for stackup layer only]
IsVisiblePath: true [for stackup layer only]
IsVisiblePad: true [for stackup layer only]
IsVisibleHole: true [for stackup layer only]
IsVisibleComponent: true [for stackup layer only]
IsLocked: "true"|"false"
LayerId: integer [the ID for the layer]

The following are also in the array if the layer is a stackup layer:
Index: integer [the stackup order index]
LayerThickness: double [total layer thickness, in meters]
EtchFactor: double [won't show if the layer has no etch factor defined]
IsIgnored: "true"|"false"
NumberOfSublayers: 1 [always 1]
Material0: materialName
FillMaterial0: materialName
Thickness0: expression_with_units
LowerElevation0: expression_with_units
Roughness0Type: Groisse | Huray [won't show if the layer has no roughness defined]
Roughness0: expression_with_units | expression_with_units, double [Groisse rough-

ness or Huray roughness]

Parameters: `The name of the layer`

GetProperties (Footprint Editor)

Use: Gets a list of all the properties belonging to a specific **PropServer** and **PropTab**. This can be executed by the **oProject**, **oDesign**, or **oEditor** objects.

Command: None

Syntax: `GetProperties( <PropTab>, <PropServer> )`

Return Value: Variant array of strings – the names of the properties belonging to the prop server.

Example:

```
Dim all_props
all_props = oDesign.GetProperties("BaseElementTab",_
"rect_1")
```

GetStackupLayerNames (Footprint Editor)

Use: Informational.

Command: None.

Syntax: `GetStackupLayerNames`

Return Value: array of strings which are the names of all layers in the layout, blackbox, or footprint.

Parameters: `None.`

Intersect (Footprint Editor)

Use: Causes Boolean intersecting of 2 or more *primitive* (polygons, rectangles, lines, or circles) objects.

Syntax:

```
Intersect Array ("NAME: primitives",
        <object_name>,  // 1st primitive name
        <object_name>,  // 2nd primitive, if any
        …)              // etc
```

Example:

```
oEditor.Intersect Array("NAME:primitives", "circle_0",
"rect_2")
```

Move (Footprint Editor)

Use: Moves an object in the footprint editor

Command: None

Syntax: `Move`

Return Value: None

Example:

```
oDefinitionEditor.Move Array("NAME:elements", "poly118"),
_
Array(-0.000352530973032117, -0.000276988605037332)
```

PageSetup (Footprint Editor)

Use: Specifies page setup for printing.

Command: **File>Page Setup**

Syntax: `PageSetup <ArgArray>`

Return Value: None.

Parameters: `<Margins>`: Page margins in implicit units of inches.

<Border>: Integer value indicating to draw border (1) or not to draw (0).

<DesignVars>: Integer value indicating to draw design vars (1) or not to draw (0).

Example:

```
Set oProject = oDesktop.GetActiveProject()
Set oDefinitionEditor = oProject.SetActiveDefinitionEdi-
tor("FootprintEditor", "Footprint")
oDefinitionEditor.PageSetup Array("NAME:PageSetupData",
"margins:=", Array("left:=",  _
  500, "right:=", 800, "top:=", 500, "bottom:=", 500),
"border:=", 1, "DesignVars:=",  _
  1)
```

RemoveLayer (Footprint Editor)

Use: Removes a layer or stackup layer.

Command: Remove Layer from a layout or footprint definition

Syntax: `RemoveLayer (<LayerName>)`

Return Value: None.

Parameters: `<LayerName>`

Type: <String>

Example:

```
oEditor.RemoveLayer ("T3 C1 sub")
oDefinitionEditor.RemoveLayer("Top3 Footprint")
```

Note As with other Layout scripting interface commands that modifiy the layout, this command is not intended for use within scripts that define footprints. The command behavior from within such a script is undefined and may be unexpected. Use the LayoutHost scripting interface commands within scripts that define footprints.

RemovePort (Footprint Editor)

Use: Selected pins are changed to vias. Selected edge ports are removed.

Syntax: `RemovePort <NAME:elements", <object_name> ... // objects to be removed`

Return Value: None.

Parameters: `<object_name>`

Type: <String>

Example:

```
oEditor.RemovePort Array("NAME:elements", "via_195",
"Port1")
```

Rotate [Footprint Editor]

Use: Rotates the selected object

Command: None

Syntax: `Rotate`

Return Value: None

Example:

```
oDefinitionEditor.Rotate Array("NAME:elements",
"poly118"),
Array( 0.00046997896940544 3, 0.000742770842193945)
```

Save (Footprint Editor)

Use: Saves changes during editing of symbols and footprints

Command: None

Syntax: `Save`

Return Value: None

Parameters: `None`

Example:

```
oDefinitionEditor.Save
```

SetActiveDefinitionEditor (Footprint Editor)

Use: Selects the symbol or footprint editor.

Command: None

Syntax: `SetActiveDefinitionEditor < EditorName >,<Name>`

Return Value: None

Parameters:

`<EditorName>`

Type: <string>

Possible Values: SymbolEditor; FootprintEditor

<Name>

Type: <string>

Name of symbol or footprint to be modified

Example:

```
oProject.SetActiveDefinitionEditor("SymbolEditor",
"tqtrx_cap")
```

SetPropertyValue (Footprint Editor)

Use: Sets the value of one property. This is not supported for properties of the following types: **ButtonProp, PointProp,** and **VPointProp**. Only the **ChangeProperty** command can be used to modify these properties. This can be executed by the **oProject**, **oDesign**, or **oEditor** objects.

Command: None

Syntax: `SetPropertyValue <PropTab>, <PropServer>, <PropName>, <PropValue>`

Return Value: None

Parameters:

`<PropValue>`

Type: String

Contains the value to set the property. The formatting is different depending on what type of property is being edited. Use **GetPropertyValue** for the desired property to see the expected format.

Example:

```
oEditor.SetPropertyValue _
"BaseElementTab","rect_1",_
"LineWidth", "3mm"
```

Subtract (Footprint Editor)

Use: Causes boolean subtracting of one or more *primitive* (polygons, rectangles, lines, or circles) object(s) from another one.

```
Subtract Array ("NAME: primitives",
  <object_name>,  // Primitive to subtract from
  <object_name>,  // 1st primitive to subtract, if any
  …)              // etc
```

Example:

```
oEditor.Intersect Subtract ("NAME:primitives", "cir-
cle_0", "rect_2")
```

ToggleViaPin (Footprint Editor)

Use: Selected pins are changed to vias. Selected vias are changed to pins.

Syntax:
```
ToggleViaPin <NAME:elements", <object_name> ... //
objects to be toggled
```

Return Value: None.

Parameters:
```
<object_name>
```
Type: <String>

Example:
```
oEditor.ToggleViaPin Array("NAME:elements", "via_195")
```

Unite (Footprint Editor)

Use: Causes Boolean uniting of 2 or more *primitive* (polygons, rectangles, lines, or circles) objects.

Syntax:
```
Unite Array ("NAME: primitives",
      <object_name>,  // 1st primitive name
      <object_name>,  // 2nd primitive, if any
      …)              // etc
```

Example:

```
oEditor.Unite Array("NAME:primitives", "circle_0",
"rect_2")
```

ZoomToFit (Footprint Editor)

Use: Set the current zoom to fit the contents of the currently visible page

Command: None

Syntax: `ZoomToFit()`

Return Value: None

30 Design Verification Script Commands

The Design Verification (DV) module controls the use of DV rule sets and runs in a Designer project. The DV module is accessed via the design script object.

```
Set oDesign = oProject.GetActiveDesign()
Set oModule = oDesign.GetModule("DV")
```

The topics for this section include:

AddRuleSet

Use: Adds a rule set to the design

Command: Right-click **Design Verification** in the **Project Tree** and choose **Add Rule Set**

Syntax:
```
AddRuleSet Array("NAME:<RuleSetName>",
"ScriptNames:=", <ScriptInfo>,
"ScriptActiveFlags:=", <ScriptFlags>)
```

Return Value: None

Parameters:

<RuleSetName>:
<string> // name of the rule set to create

<ScriptInfo>:
Array(<string>, <string>,...) // names of scripts to be in the rule set

<ScriptFlags>:
sequence of "t" and "f" characters
t indicates a script that is active (used in when the rule set is run)
f indicates a script is not currently active (used when the rule set is run)
applied to the scripts as ordered in <ScriptInfo>
<string>

Example:
```
oModule.AddRuleSet Array("NAME:Rule Set 9", _
"ScriptNames:=", Array("And",  _
"Find Shorts"),  _
"ScriptActiveFlags:=", "tf")
```

AddRun

Use: Adds a run to a rule set already in the design

Command: Right-click on a rule set item under **Design Verification** in the **Project Tree** and choose **Add Run**

Syntax:
```
AddRun <RuleSetName>,
                    Array("NAME:<RunName>",
                        "TargetType:=", <TargetTypeInfo>,          // optional
                        "TargetObjects:=", <TargetObjectsInfo>,    // optional
                        "IgnoredObjects:=", <IgnoredObjectsInfo>,  // optional
                        "ArcTolerance:=", <ToleranceString>)       // optional
```

Return Value: None

Parameters: **<RuleSetName>:**

<string> // name of an existing rule set

<RunName>:
<string> // name of the run to create

<TargetTypeInfo>:
<int> // 0 for entire layout (default if not specified)
// 1 for specified portion of layout

<TargetObjectsInfo>:
objects on which to perform the check
may specify if TargetTypeInfo is 1 (specified portion of layout)
"<ObjectName>, <ObjectName>, ..."

<ObjectName>:
<string> // name of a layout object

<IgnoredObjectsInfo>:
objects which to ignore when performing the check
may specify if TargetTypeInfo is 0 (entire layout)
"<ObjectName>, <ObjectName>, ..."

<ToleranceString>:
<string> // real number and units
// if not specified, the current layout arc tolerance is used

Example:

```
oModule.AddRun "Rule Set 10", _
Array("NAME:All", _
"TargetType:=", 0)
oModule.AddRun "Rule Set 10", _
Array("NAME:Selected Objs2",  _
"TargetType:=", 1, _
```

```
"TargetObjects:=", "line_975,line_1015")
oModule.AddRun "Rule Set 10", _
Array("NAME:Objects to Ignore2", _
"TargetType:=", 0, _
"IgnoredObjects:=", "via_209,line_736")
```

DeleteRuleSet

Use: Deletes a rule set from the design

Command: Right-click on a rule set item under **Design Verification** in the **Project Tree** and choose **Delete**

Syntax: `DeleteRuleSet <RuleSetName>`

Return Value: None

Parameters:

`<RuleSetName>:`

<string> // name of the rule set to delete

Example:

```
oModule.DeleteRuleSet "Rule Set 9"
```

DeleteRun

Use: Deletes a run from an existing a rule set

Command: Right-click on a run item under **Design Verification** in the **Project Tree** and choose **Delete**

Syntax: `DeleteRun <RuleSetName>, <RunName>`

Return Value: None

Parameters:

`<RuleSetName>:`

<string> // name of an existing rule set

<RunName>:

<string> // name of the run to delete

Example:

```
oModule.DeleteRun "Rule Set 10", "All
```

EditRuleSet

Use: Edits a existing rule set

Command: Right-click on a rule set item under **Design Verification** in the **Project Tree** and choose **Properties**.

Syntax: `EditRuleSet <ExistingRuleSetName>,`

Array("NAME:<RuleSetName>",
"ScriptNames:=", <ScriptInfo>,
"ScriptActiveFlags:=", <ScriptFlags>)

Return Value: None

Parameters:

<ExistingRuleSetName>:
<string> // name of the rule set to change

<RuleSetName>:
<string> // name of the rule set after change

<ScriptInfo>:
Array(<string>, <string>,...) // names of scripts to be in the rule set

<ScriptFlags>:
sequence of "t" and "f" characters
t indicates a script that is active (used in when the rule set is run)
f indicates a script is not currently active (used when the rule set is run)
applied to the scripts as ordered in <ScriptInfo>
<string>

Example:

```
oModule.EditRuleSet   "Rule Set 9", _
Array("NAME:Rule Set 10", _
"ScriptNames:=", Array("And",  _
"Find Shorts"),  _
"ScriptActiveFlags:=", "tt")
```

EditRun

Use: Edits an existing run.

Command: Right-click on a run item under **Design Verification** in the **Project Tree** and choose **Properties**.

Syntax:

EditRun <RuleSetName>,
<ExistingRunName>,
Array("NAME:<RunName>",
"TargetType:=", <TargetTypeInfo>, // optional

"TargetObjects:=", <TargetObjectsInfo>, // optional
"IgnoredObjects:=", <IgnoredObjectsInfo>, // optional
"ArcTolerance:=", <ToleranceString>) // optional

Return Value: None

Parameters:

<RuleSetName>:
<string> // name of an existing rule set

<ExistingRunName>:
<string> // name of the run to change

<RunName>:
<string> // name of the run after the changes

<TargetTypeInfo>:
<int> // 0 for entire layout (default if not specified)
// 1 for specified portion of layout

<TargetObjectsInfo>:
objects on which to perform the check
may specify if TargetTypeInfo is 1 (specified portion of layout)
"<ObjectName>, <ObjectName>, ..."

<ObjectName):
<string> // name of a layout object

<IgnoredObjectsInfo>:
objects which to ignore when performing the check
may specify if TargetTypeInfo is 0 (entire layout)
"<ObjectName>, <ObjectName>, ..."

<ToleranceString>:
<string> // real number and units
// if not specified, the current layout arc tolerance is used

Example:

```
oModule.EditRun "Rule Set 10", _
```

```
"All", _
Array("NAME:New All", _
"TargetType:=", 0)
oModule.EditRun "Rule Set 10", _
"New All", _
Array("NAME:Selected Objs2",  _
"TargetType:=", 1, _
"TargetObjects:=", "line_975,line_1015")
oModule.EditRun "Rule Set 10", _
"Selected Objs2", _
Array("NAME:Objects to Ignore2", _
"TargetType:=", 0, _
"IgnoredObjects:=", "via_209,line_736")
```

RenameRuleSet

Use: Renames a existing rule set

Command: Right-click on a rule set item under **Design Verification** in the **Project Tree** and choose **Rename**

Syntax: `RenameRuleSet  <ExistingRuleSetName>, <NewRuleSetName>`

Return Value: None

Parameters:

`<ExistingRuleSetName>:`

<string> // name of the rule set to change

<NewRuleSetName>:

<string> // new name for the rule set

Example:

```
oModule.RenameRuleSet "Rule Set 12", "Rule Set 30"
```

RenameRun

Use: Renames an existing run.

Command: Right-click on a run item under **Design Verification** in the **Project Tree** and choose **Rename**

Syntax: `RenameRun <RuleSetName>, <ExistingRunName>, <NewRunName>`

Return Value: None

Parameters:

```
<RuleSetName>:
        <string>  // name of an existing rule set

        <ExistingRunName>:
        <string>  // name of the run to change

        <NewRunName>:
        <string>  // new name for the run
```

Example:

```
oModule.RenameRun "Rule Set 30", "New All", "Sel Objects"
```

RunAllDV

Use: Executes all runs in the Design Verification Project Tree item.

Command: Right-click on **Design Verification** in the **Project Tree** and choose **Run All**

Syntax:

```
RunAllDV
```

Return Value: None

Parameters:

```
None
```

Example:

```
oModule.RunAllDV
```

RunAllRuleSetDV

Use: Executes all runs in an existing rule set

Command: Right-click on a rule set item under **Design Verification** in the **Project Tree** and choose **Run All**

Syntax:

```
RunAllRuleSetDV <RuleSetName>
```

Return Value: None

Parameters:

```
<RuleSetName>:
        <string>  // name of an existing rule set
```

Example:

```
oModule.RunAllRuleSetDV "Rule Set 30"
```

RunDV

Use: Executes the specified run in an existing rule set

Command: Right-click on a run item under **Design Verification** in the **Project Tree** and choose **Run**

Syntax:

```
RunDV <RuleSetName>, <RunName>
```

Return Value: None

Parameters:

```
<RuleSetName>:
        <string>  // name of an existing rule set

        <RunName>:
        <string>  // name of the run to execute
```

Example:

```
oModule.RunDV "Rule Set 30",  "Run 2"
```

31 PlanarEM Scripting

PlanarEM scripting commands should be executed by the **oDesign** object. For example:

```
oDesign.GetModule("Excitations")
oModule.CommandName <args>
```

The topics for this section include:

Design Level Commands

The following sections present the PlanarEM design level scripting commands that are available:

AddModelingProperties
Analyze
CopyItemCommand
DeleteDesignInstance
EditImportData
EditInfiniteArray
EditNotes
EditCoSimulationOptions
EditOptions
EMDesignOptions
ExportNetworkData
ExportForSpice
ExportForHSpice
ExportNMFData
GetActiveEditor
GetEditor
GetModule
GetName
GetSetups
GetSweeps
GetSetupData
GetSourceData
InsertDesign
OverlayCurrents
OverlayFarField
OverlayMesh
OverlayNearField
PasteItemCommand
Redo
RemoveModelingProperties
RemoveImportData
RenameDesignInstance

AddModelingProperties

Use: Add a modeling property to a design

Command: None

Syntax: `AddModelingProperties <design>`

Return Value: None

Parameters: `<design>`

Type: string

Example: `oDesign.AddModelingProperties <design>`

Analyze [Planar EM]

Use: Simulates the given setup

Command: None

Syntax: `Analyze <setup>`

Return Value: None

Parameters: `<setup>`

Type: string

Example: `oDesign.Analyze NWA1`

CopyItemCommand

Use: Copy tree items, such as Altrasim Solution Setups in Nexxim, or Solve Setups and Frequency Sweeps in Ensemble.

Command: None

Syntax: `CopyItemCommand <ItemPathList>`

Return Value: None

Parameters: `<ItemPathList>`

Type: Array of strings

Example:

```
oDesign.CopyItemCommand
Array("Project1|Nexxim1|Analysis|DCAnalysis2")
```

DeleteDesignInstance

Use: Delete the design instance

Command: None

Syntax:

```
DeleteDesignInstance <instance_name>
```

Return Value: None

Parameters:

```
<instance_name>
```

Type: string

Example:

```
oDesign.DeleteDesignInstance <instance_name>
```

EditImportData

Use: Edit an imported solution

Command: None

Syntax:

```
EditImportData <oldname> <newlink> <newpath> <newname>
<data> <file>
```

Return Value: None

Parameters:

```
<oldname>
```

Type: string

<newlink>

Type: int

<newpath>

Type: string

<newname>

Type: string

<data>

Type: array

<file>

Type: string

Example:

```
oDesign.EditImportData <oldname> <newlink> <newpath>
<newname> <data> <file>
```

EditInfiniteArray

Use: Edits the properties of an infinite array

Command: None

Syntax: `EditInfiniteArray <array_name>`

Return Value: None

Parameters: `<array_name>`

Type: string

Example: `oDesign.EditInfiniteArray <array_name>`

EditNotes [Planar EM]

Use: Edits the notes of a design

Command: None

Syntax: `EditNotes <design_name>`

Return Value: None

Parameters: `<design_name>`

Type: string

Example: `oDesign.EditNotes <design_name>`

EditCoSimulationOptions

Use: Sets options for cosimulation.

Command: None

Syntax: `EditCoSimulationOptions <array_name>`

Return Value: None

Parameters: `<array_name>`

Type: string

```
oDesign.EditCoSimulationOptions Array("NAME:CoSimOp-
tions",
"Override:=", true, _
"Setup:=", "Setup 1", _
"OverrideSweep:=", true, _
"Sweep:=", "Sweep 1", _
"SweepType:=", 4, _
```

```
"Interpolate:=", false, _
"YMatrix:=", true, _
"AutoAlignPorts:=", false, _
"InterpAlg:=", "auto")
```

EditOptions [Planar EM]

Use: Edits the properties of an infinite array

Command: None

Syntax: `EditOptions <array_name>`

Return Value: None

Parameters: `<array_name>`

Type: string

Example: `oDesign.EditOptions <array_name>`

EMDesignOptions

Use: Set options for an EM Design.

Command: Right click on design and select **EM Design Options**

Syntax: `DesignOptions <Options Array>`

Return Value: None

Parameters:

<Options Array>

Array("NAME:options",
"SaveSolFilesAsBinary:=", <boolean>,
"LowPriorityForSimulations:=", <boolean>,
"SaveNearFieldSolutions:=", <boolean>,
"SchematicEnabled:=", <boolean>,
"UseGlobalNumProc:=", <boolean>,
"ComputeBothEvenAndOddCPWModes:=", <boolean>,
"NumProcessors:=", <int>,
"NumProcessorsDistrib:=", <int>,
"CausalMaterials:=", <boolean>,
"UseHPCForMP:=", <boolean>,
"HPCLicenseType:=", <int>)

SaveSolFilesAsBinary - if true, solutions files are saved using a binary format.

LowPriorityForSimulations - if true, run simulations at a lower CPU priority.

SaveNearFieldSolutions - if true, save near field solutions

SchematicEnabled - if true, enable schematics

UseGlobalNumProc - if true, use global number of processors and ignore NumProcessors

ComputeBothEvenAndOddCPWModes - if true, compute both even and odd cpw modes

NumProcessors - number of processors

NumProcessorsDistrib- number of distributed processors

CausalMaterials - if true, use causal materials

UseHPCForMP - if true, use hpc for mp

HPCLicenseType - number indicating hpc license type

Example:

```
oDesign.DesignOptions Array("NAME:options",
"SaveSolFilesAsBinary:=", true,
"LowPriorityForSimulations:=", false,
"SaveNearFieldSolutions:=", false,
"SchematicEnabled:=", true,
"UseGlobalNumProc:=", true,
"ComputeBothEvenAndOddCPWModes:=", false,
"NumProcessors:=", 1,
"NumProcessorsDistrib:=", 1,
"CausalMaterials:=", true,
"UseHPCForMP:=", true,
"HPCLicenseType:=", 1)
```

ExportNetworkData [Planar EM]

Use: Exports a matrix solution to a file.

Command: None

Syntax:
```
ExportNetworkData <DesignVariationKey>
<SolnSelectionArray> <FileFormat> <OutFile> <FreqsArray>
<DoRenorm> <RenormImped>
```

Return Value: None

Parameters:
```
<DesignVariationKey>
```

Type: string

<SolnSelectionArray>

Type: string

<FileFormat>

Type: string

<OutFile>

Type: string

<FreqsArray>

Type: array

<DoRenorm>

Type: string

<RenormImped>

Type: string

Example:

```
oDesign.ExportNetworkData <DesignVariationKey>
<SolnSelectionArray> <FileFormat> <OutFile> <FreqsArray>
<FileFormat> <OutFile> <FreqsArray> <DoRenorm>
<RenormImped>
```

ExportForSpice [Planar EM]

Use: Export matrix solution data to a file in the given spice format.

Command: None

Syntax:

```
ExportForSpice <SolutionVariationKey>
<SolnSelectionArray> <Spice Type> <Bandwidth>
<FullWaveSpiceFilename> <LumpedElementFilenme> <Unused>
<Unused> <PartialFractionFilename> <FittingError>
<MaxOrder> <UseCommonGround> <DoPassivityCheck>
```

Return Value: None

Parameters:

<SolutionVariationKey>

Type: string

<SolnSelectionArray>

Type: array

<Spice Type: 0=PSpice, 2=MaxwellSpice, 3=Spectre>

Type: int

<Bandwidth: 0=Low Bandwidth, 1=Broad Bandwidth>

Type: int

<FullWaveSpiceFilename>

Type: string
<LumpedElementFilenme>
Type: string
<Unused>
Type: string
<Unused>
Type: string
<PartialFractionFilename>
Type: string
<FittingError [Default:0.5]>
Type: double
<MaxOrder [Default:200]>
Type: int
<UseCommonGround [Default:FALSE]>
Type: boolean
<DoPassivityCheck [Default:FALSE]>
Type: boolean

Example:

```
oDesign.ExportForSpice "", Array("LNA:LNA"), 0, 0,  _
"C:/Documents and Settings/Marcia.BMT/My Documents/Code
Modifications/Test" & _
" Designs/s-params_fws_fws.lib",  _
"C:/Documents and Settings/Marcia.BMT/My Documents/Code
Modifications/Test" & _
" Designs/s-params_fws_lfws.lib", "", "", "", 0.52, 210,
1, 1
```

ExportForHSpice [Planar EM]

Use: Export matrix solution data to an HSpice subcircuit.

Command: None

Syntax:

```
ExportForHSpice <SolutionVariationKey>
<SolnSelectionArray> <Spice Type> <Bandwidth>
<FullWaveSpiceFilename> <LumpedElementFilenme> <Unused>
<Unused> <PartialFractionFilename> <Fitting Error>
<Unusedint> <MaxOrder> <UseCommonGround>
<DoPassivityCheck>
```

Return Value: None

Parameters:

```
<SolutionVariationKey>
```

Type: string
<SolnSelectionArray>
Type: array
<Spice Type: 1=HSPICE>
Type: int
<Bandwidth: 0=Low Bandwidth, 1=Broad Bandwidth>
Type: int
<FullWaveSpiceFilename>
Type: string
<LumpedElementFilenme>
Type: string
<Unused>
Type: string
<Unused>
Type: string
<PartialFractionFilename>
Type: string
<Fitting Error>
Type: double
<Unusedint>
Type: int
<MaxOrder>
Type: int
<UseCommonGround [Default: FALSE]>
Type: boolean
<DoPassivityCheck [Default: FALSE]>
Type: boolean

Example:

```
oDesign.ExportForHSpice "", Array("Setup 1:Sweep 1"), 1,
0, "C:/Projects/MyTRL_fws.sp", "", "", "", "", 0.5, 0,
200, 0, 0
```

ExportNMFData [Planar EM]

Use: Exports a matrix solution to a file in neutral model format. Available only for driven solutions with ports. Variables can be held constant by setting their values in the variation field. All other independent variables will be treated as NMF parameters.

Command: None

Syntax: `ExportNMFData <SolnSelectionArray> <OutFile> <FreqsArray> <DesignVariationKey> <DoRenorm> <RenormImped>`

Return Value: None

Parameters:

`<SolnSelectionArray>`

Type: string

<OutFile>

Type: string

<FreqsArray>

Type: array

<DesignVariationKey>

Type: string

<DoRenorm>

Type: string

<RenormImped>

Type: array

Example:

```
oDesign.ExportNMFData <SolnSelectionArray> <OutFile>
<FreqsArray> <DesignVariationKey> <DoRenorm>
<RenormImped>
```

GetActiveEditor [Planar EM]

Use: Get the name of the active editor.

Command: None

Syntax: `GetActiveEditor <object_variable>`

Return Value: String

Parameters:

`<object_variable>`

Type: string

Example: `oDesign.GetActiveEditor <object_variable>`

Note that GetActiveEditor returns NULL if the editor is open in a non-active window.

GetEditor [Planar EM]

Use: Get the named editor without activating it.

Command: None

Syntax: `GetEditor <editorName object_variable>`

Return Value: String
Parameters: `<editorName>`
Type: string
<object_variable>
Type: string
Example: `oDesign.GetEditor <editorName> <object_variable>`

GetModule [Planar EM]

Use: Get the IDispatch for the specified module.
Command: None
Syntax: `GetModule <"modulename"> <object_variable>`
Return Value: Module IDispatch
Parameters: `<"modulename">`
Type: string
<object_variable>
Type: string
Example: `oDesign.GetModule "modulename" object_variable`

GetName [Planar EM]

Use: Get the name of the active design.
Command: None
Syntax: `GetName <namestring_variable>`
Return Value: String
Parameters: `<namestring_variable>`
Type: string
Example: `oDesign.GetName namestring_variable`

GetSetups [Planar EM]

Use: Get setups.
Command: None
Syntax: `GetSetups <setups>`
Return Value: String array of setup names
Parameters: `<setups>`
Type: string

Example:
```
oSetup = oDesign.GetModule("SolveSetups")
setups = oSetup.GetSetups()
```

GetSweeps [Planar EM]

Use: Get sweeps.

Command: None

Syntax: `GetSweeps <sweep>`

Return Value: String array of sweeps

Parameters: `<sweep>`

Type: string

Example:
```
oSetup = oDesign.GetModule("SolveSetups")
setups = oSetup.GetSetups()
sweep = oSetup.GetSweeps(setups[0])
```

GetSetupData [Planar EM]

Use: Get setup data.

Command: None

Syntax: `GetSetupData <setup>`

Return Value: String array of setup information.

Parameters: `<setup>`

Type: string

Example:
```
oSetup = oDesign.GetModule("SolveSetups")
setup = oSetup.GetSetups()
dat = oSetup.GetSetupData(setup[0])
```

GetSourceData [Planar EM]

Use: Takes a source name as input and returns a VARIANT array containing a data block of the source data.

Command: None

Syntax: `GetSourceData`

Return Value: Type: Array of i/o block data

Parameters: `<SourceName>`

Type: string

Example:
```
Dim sourceDataArr
sourceDataArr = oDesign.GetSourceData ("SourceName")
```

InsertDesign [Planar EM]

Use: Insert a design

Command: None

Syntax:
```
InsertDesign <type>, <name>, <stationerypath>,
<parentname>
```

Return Value: None

Parameters:

`<type>`

Type: string

<name>

Type: string

<stationerypath>

Type: string

<parentname>

Type: string

Example:
```
oDesign.InsertDesign <type>, <name>, <stationerypath>,
<parentname>
where, <type> is NULL for top-level design
```

OverlayCurrents [Planar EM]

Use: Creates a 3D view with a currents overlay

Command: None

Syntax:
```
OverlayCurrents <solution name>
```

Return Value: <success>

Type: boolean

Parameters:
```
<solution name>
```

Type: string

Example:
```
Dim success
success = oDesign.OverlayCurrents ("HFSS Setup : Sweep
1")
```

OverlayFarField [Planar EM]

Use: Creates a 3D view with a far field overlay

Command: None

Syntax: `OverlayFarField <solution name>`

Return Value: <success>

Type: boolean

Parameters: `<solution name>`

Type: string

Example:

```
Dim success
success = oDesign.OverlayFarField ("HFSS Setup : Sweep
1")
```

OverlayMesh [Planar EM]

Use: Creates a 3D view with a mesh overlay

Command: None

Syntax: `OverlayMesh <solution name>`

Return Value: <success>

Type: boolean

Parameters: `<solution name>`

Type: string

Example:

```
Dim success
success = oDesign.OverlayMesh ("HFSS Setup : Sweep 1")
```

OverlayNearField [Planar EM]

Use: Creates a 3D view with a near field overlay

Command: None

Syntax: `OverlayNearField <solution name>`

Return Value: <success>

Type: boolean

Parameters: `<solution name>`

Type: string

Example:

```
Dim success
success = oDesign.OverlayNearField ("HFSS Setup : Sweep
1")
```

PasteItemCommand [Planar EM]

Use: Paste tree items, such as Altrasim Solution and Solve Setups pasted to the Analysis Tree, or Frequency Sweeps pasted to the Solve Setup Tree.

Command: Right-click on the Analysis item in the Project tree and select Paste.

Syntax: `PasteItemCommand <ItemPath>`

Return Value: None

Parameters: `<ItemPath>`

Type: string

Example: `oDesign.PasteItemCommand "Project1|Nexxim1|Analysis"`

Redo [Planar EM]

Use: Redo the last operation

Command: Edit>Redo

Syntax: `Redo`

Return Value: None

Parameters: `None`

Example: `oDesign.Redo`

RemoveModelingProperties [Planar EM]

Use: Remove a modeling property from a design

Command: None

Syntax: `RemoveModelingProperties <design>`

Return Value: None

Parameters: `<design>`

Type: string

Example: `oDesign.RemoveModelingProperties <design>`

RemoveImportData [Planar EM]

Use: Remove an imported solution

Command: None

Syntax: `RemoveImportData <solution_name>`

Return Value: None

Parameters: `<solution_name>`

Type: string

Example: `oDesign.RemoveImportData <solution_name>`

RenameDesignInstance [Planar EM]

Use: Rename the design instance

Command: None

Syntax: `RenameDesignInstance <oldname> <newname>`

Return Value: None

Parameters: `<oldname>`

Type: string

<newname>

Type: string

Example: `oDesign.RenameDesignInstance <oldname> <newname>`

RenameImportData [Planar EM]

Use: Rename an imported solution

Command: None

Syntax: `RenameImportData <oldname> <newname>`

Return Value: None

Parameters: `<oldname>`

Type: string

<newname>

Type: string

Example: `oDesign.RenameImportData <oldname> <newname>`

ReportTemplates [Planar EM]

Use: Creates a report using one of the PlanarEM report templates

Command: None

Syntax: `ReportTemplates <template>`

Return Value: None

Parameters: `<template>`

Type: string

Example: `oDesign.ReportTemplates <template>`

SetActiveEditor [Planar EM]

Use: Set the active editor to the named one. (Activates window or opens window if necessary).

Command: None

Syntax: `SetActiveEditor <editorName> <object_variable>`

Return Value: None

Parameters: `<editorName>`

Type: string

<object_variable>

Type: string

Example: `oDesign.SetActiveEditor <editorName> <object_variable>`

StartAnalysis [Planar EM]

Use: Simulates all setups

Command: None

Syntax: `StartAnalysis`

Return Value: None

Parameters: `None`

Example: `oDesign.StartAnalysis`

Undo [Planar EM]

Use: Undo the last operation

Command: Edit>Undo

Syntax: `Undo`

Return Value: None

Parameters: `None`

Example: `oDesign.Undo`

ValidateCircuit [Planar EM]

Use: Validates a PlanarEM design for consistency

Command: None

Syntax: `ValidateCircuit`

Return Value: None

Parameters: `None`

Example: `oDesign.ValidateCircuit`

Simulation Setup Commands

This section presents the commands that are available in the “SolveSetups” module. For example:

oDesign.GetModule("SolveSetups")

oModule.CommandName <args>

Add
AddSweep
Analyze
AnalyzeSweep
Delete
DeleteSweep
DynamicMeshOverlays
Edit
EditSweep
GetAllSolutionNames
LayoutMeshOverlay
Layout3DMeshOverlay
ListVariations
RefreshMeshOverlays
RenameSweep

Add [Setup Planar EM]

Use: Adds a new setup

Syntax: `Add(STRING newsetup) // new setup name`

Example: `oModule. Add <newsetup>`

AddSweep [Planar EM]

Use: Adds a sweep to a setup

Use: AddSweep(STRING setup) / / setup name
`STRING newsweep) // new sweep name`

Example:

```
oModule.AddSweep "Setup 1", _
Array("NAME:Sweep 1", _
Array("NAME:Properties", "Enable:=", "true"), _
"GenerateSurfaceCurrent:=", false, _
"FastSweep:=", false, _
"ZoSelected:=", false, _
"SAbsError:=", 0.005, _
"ZoPercentError:=", 1, _
Array("NAME:Sweeps", _
"Variable:=", "F", _
"Data:=", "LINC 1GHz 10GHz 10 LINC 10GHz 12GHz 5"))
```

Analyze [Planar EM]

Use: Simulates the given setup

Syntax: `Analyze(STRING setup1) // setup name`

Example: `oModule. Analyze <setup1>`

AnalyzeSweep [Planar EM]

Use: Simulates the given sweep

Syntax: `AnalyzeSweep(STRING setup1) // setup name`
STRING sweep1) // sweep name

Example: `oModule. AnalyzeSweep <setup1> <sweep1>`

Delete [Planar EM]

Use: Removes a simulation setup

Syntax: `Delete(STRING name) // The specified setup to delete`

Example: `oModule. Delete <name>`

DeleteSweep [Planar EM]

Use: Removes a sweep from a setup

Syntax: `DeleteSweep(STRING setup1) // The specified setup`
`STRING sweep1) // The specified sweep`

Example: `oModule. DeleteSweep <setup1> <sweep1>`

DynamicMeshOverlays [Planar EM]

Use: Turns on dynamic mesh overlays

Syntax: `DynamicMeshOverlays(STRING setup) // setup name`

Example: `oModule. DynamicMeshOverlay <setup>`

Edit [Planar EM]

Use: Edit a setup

Syntax: `Edit(STRING setup) // setup name`
`ARRAY solvesetup_data) // setup data`

Example: `oModule. Edit <setup> <solvesetup_data>`

EditSweep [Planar EM]

Use: Edit a sweep

Syntax: `EditSweep(STRING setup) // setup name`
ARRAY solvesetup_data) // setup data

Example: `oModule. EditSweep <setup> <solvesetup_data>`

GetAllSolutionNames [Planar EM]

Use: Returns a list of Ensemble solution names

Syntax: `GetAllSolutionNames`

Return Value: Array of Strings

Parameters: None

Example:
```
Dim setupArr
setupArr = oModule.GetAllSolutionNames
```

LayoutMeshOverlay [Planar EM]

Use: Overlay a mesh for a setup in layout

Syntax: `LayoutMeshOverlay(STRING setup) // setup name`

Example: `oModule. LayoutMeshOverlay <setup>`

Layout3DMeshOverlay [Planar EM]

Use: Overlay a mesh for a setup in 3D

Syntax: `Layout3DMeshOverlay(STRING setup) // setup name`

Example: `oModule. Layout3DMeshOverlay <setup>`

ListVariations [Planar EM]

Use: Get a list of solution variations for a given solution

Syntax: `ListVariations(STRING SolutionName) // solution name`

Example:
```
oModule.ListVariations("NWA1")
```

RefreshMeshOverlays [Planar EM]

Use: Refreshes mesh display for a setup

Syntax: `RefreshMeshOverlays(STRING setup) // setup name`

Example: `oModule.RefreshMeshOverlays <setup>`

RenameSweep [Planar EM]

Use: Renames a simulation setup

Syntax: `RenameSweep(STRING oldname) // old setup name`
STRING newname) // new setup name

Example: `oModule. RenameSweep <oldname> <newname>`

Excitations Commands

The following commands are available in the “Excitations” module. For example:
oDesign.GetModule("Excitations")
oModule.CommandName <args>

Add
AddRefPort
AddRefPortUsingEdges
CoupleEdgePorts
DecoupleEdgePorts
Delete
DeleteProbePortAndVia
Edit
EditExcitations
GetAllBoundariesList
GetAllPortsList
Rename
RemoveRefPort
SelectInLayout

Add [Excitation Planar EM]

Use: Adds an excitation

Syntax: `Add(ARRAY excitation_data) // excitation data`

Example:

```
oModule. Add <excitation_data>
```

AddRefPort [Planar EM]

Use: Adds a reference to an existing port

Syntax: `AddRefPort(STRING portname) // name of the port`
STRING edgeportname) // name of the edgeport

Example: `oModule. AddRefPort portname edgeportname`

AddRefPortUsingEdges [Planar EM]

Use: Creates an edge port with reference using the specified edge

Command: None

Syntax:
```
<port name>,
Array("Name:EdgeRefs",
    "edge:=",  Array(<edge information>)
```

Return Value: None

Parameters:

<port name>: string that will name the created edge port

<edge information>: describes the edge.
There are two choices: primitive edge or via edge.

<edge information> for a primitive edge:
"et:=", "pe", "prim:=", <primitive name>, "edge:=", <edge number>

<primitive name>: text that is the name of the primitive to use
<edge number>: an integer indexing the edge of the primitive to use

<edge information> for via edge:
"et:=", "pse", "sel:=", <via name>, "layer:=", <layer id>,
"sx:=", <start X location>, "sy:=", <start Y location>,
"ex:=", <end X location>, "ey:=", <end Y location>,
"h:=", <arc height>, "rad:=", <radians>

<via name>: text that is the name of the via to use
<layer id>: an integer that is the id of the layer of the pad of the via to use
<start X location>, <start Y Location>:
    doubles that are the X, Y location of the start point of the edge arc
<end X location>, <end Y Location>:
    doubles that are the X, Y location of the end point of the edge arc
<arc height>: double giving the height of the edge arc (0 for a straight edge)
<radians>: double giving the arc size in radians (0 for a straight edge)

```
oModule.AddRefPortUsingEdges "Port1", Array("NAME:Edg-
eRefs", "edge:=",
Array("et:=", "pe",  "prim:=", "rect_3", "edge:=", 1))

oModule.AddRefPortUsingEdges "Port1", Array("NAME:Edg-
eRefs", "edge:=",
Array("et:=",  "pse", "sel:=", "via_5", "layer:=", 10,
"sx:=", 0.0015, "sy:=", 0.0015,
"ex:=",  -0.0015, "ey:=", 0.0015, "h:=", 0, "rad:=", 0))
```

CoupleEdgePorts

Use: Couples valid edge ports

Syntax:
```
CoupleEdgePorts(VARIANT coupledlist) //entry containing
ports to be coupled
```

Example:
```
oModule.CoupleEdgePorts Array("Port1", "Port2")
```

DecoupleEdgePorts

Use: Decouples edge ports

Syntax:
```
DecoupleEdgePorts(
VARIANT decoupledlist) //entry containing ports to be
decoupled
```

Example:
```
oModule.DecoupleEdgePorts Array("Port1:T1", "Port1:T2")
```

Delete [Excitation Planar EM]

Use: Deletes an excitation

Syntax:
```
Delete( STRING excitation_name) // excitation name
```

Example:
```
oModule. Delete <excitation_name>
```

DeleteProbePortAndVia [Planar EM]

Use: Deletes a coaxial probe port and the associated via

Syntax:
```
DeleteProbePortAndVia( STRING port_name) // Probe Port
name
STRING via_name) //Via name
```

Example: `oModule. DeleteProbePortAndVia <port_name> <via_name>`

Edit [Planar EM]

Use: Edits an excitation

Syntax:
```
Edit( STRING excitation_name) // excitation name
ARRAY excitation_data) // excitation data
```

Example: `oModule. Edit <excitation_name> <excitation_data>`

EditExcitations [Planar EM]

Use: Edits the properties of all the existing excitations

Syntax:
```
EditExcitations (Array("NAME:Excitations",
Array("NAME:portname", "phase", "magnitude"), ...),
"GapSource",
Array("NAME:PostProcess",
Array("NAME:portname", do-post-proc, "de-embedding distance", "re-normalization impedance")...),
Array("NAME:PushExParamsBlock", "IsTransient:=", bool, "StartTime:=",
"start-time", "StopTime:=", "stop-time", "MaxHarmonics:=", n-harmonics,
"WinType:=", win-type, "WidthPercentage:=", width-percent, "KaiserParam:=",
kaiser-param))
```

Return Value: None

Parameters:
```
Array NAME:Excitations // Excitations block
GapSource
Array NAME:PostProcess // PostProcess block
Array NAME:PushExParamsBlock
```

Example:
```
oModule. EditExcitations Array("NAME:Excitations",
Array("NAME:Port1",
"0deg", "1V")), Array("NAME:PostProcess",
Array("NAME:Port1", true, "1mm",
"50ohm +0i ohm")), "GapSource", Array("NAME:PushExParams-
Block",
"IsTransient:=", false,"StartTime:=", "0s", "StopTime:=",
"10s",
```

```
"MaxHarmonics:=", 100, "WinType:=", 0,"WidthPercent-
age:=", 100,
"KaiserParam:=", 0)
```

Note: GapSource should always be set.

GetAllBoundariesList [Planar EM]

Use: Get a list of all boundaries.

Command: None

Syntax: `GetAllBoundariesList <bounds>`

Return Value: String array of list of all boundaries.

Parameters: `<bounds>`

Type: string

Example:
```
oEx = oDesign.GetModule('Excitations')
bounds = oEx.GetAllBoundariesList()
```

GetAllPortsList [Planar EM]

Use: Returns the names of all Ensemble ports

Syntax: `GetAllPortsList`

Return Value: Array of Strings

Parameters: `None`

Example:
```
Dim portArr
portArr= oModule.GetAllPortsList
```

Rename [Planar EM]

Use: Renames an excitation

Syntax: `Rename(STRING oldname) // old excitation name`
STRING newname) // new excitation name

Example: `oModule. Rename <oldname> <newname>`

RemoveRefPort [Planar EM]

Use: Removes the reference for an existing port

Syntax: `RemoveRefPort(`

```
                STRING portname) // name of the port
```

Example: `oModule. RemoveRefPort portname`

SelectInLayout [Planar EM]

Use: Selects and highlights an excitation in the layout editor

Syntax:
```
SelectInLayout( STRING excitation_name) // excitation
name
```

Example: `oModule. SelectInLayout <excitation_name>`

Cavity Commands

The following commands are available in the “Cavities” module. For example:
oDesign.GetModule("Cavities")
oModule.CommandName <args>

Add
Delete
Edit
Rename
SelectInLayout

Add [Cavity Planar EM]

Use: Add a cavity to the design

Syntax: `Add(ARRAY cavity_data) // cavity data`

Example: `oModule. Add cavity_data`

Delete [Cavity Planar EM]

Use: Delete a cavity

Syntax: `Delete( STRING cavity) // cavity name`

Example: `oModule. Delete cavity`

Edit [Cavity Planar EM]

Use: Edits a cavity

Syntax: `Edit(STRING name) // cavity name`
ARRAY cavity_data) // data for the cavity
Example: `oModule. Edit name cavity_data`

Rename [Cavity Planar EM]

Use: Renames a cavity
Syntax: `Rename(STRING oldname) // old cavity name`
STRING newname) // new cavity name
Example: `oModule. Rename oldname newname`

SelectInLayout [Cavity Planar EM]

Use: Selects and highlights the cavity in the layout editor
Syntax: `SelectInLayout(STRING cavity) // cavity name`
Example: `oModule. SelectInLayout cavity`

2.5D Via Commands

The following commands are available in the “LayoutVias” module. For example:
oDesign.GetModule("LayoutVias")
oModule.CommandName <args>
ConvertPrimitives
Delete
Edit
MultipleEdit
Rename
SelectInLayout

ConvertPrimitives [Planar EM]

Use: Convert the selected primitives to 2.5D vias
Syntax: `ConvertPrimitives(ARRAY primvia_data) // primitive data`
Example: `oModule. ConvertPrimitives primvia_data`

Delete [Via Planar EM]

Use: Delete a 2.5 via

Syntax:
```
Delete(
STRING via) // 2.5 via name
```

Example:
```
oModule. Delete via
```

Edit [Via Planar EM]

Use: Edits a 2.5 via

Syntax:
```
Edit(STRING via) // 2.5 via name
ARRAY via_data) // data for the 2.5 via
```

Example:
```
oModule. Edit via via_data
```

MultipleEdit [multiple via Planar EM]

Use: Edit the properties of a collection of 2.5D via

Syntax:
```
MultipleEdit(ARRAY multiplevia_data) // multiple 2.5 via
data
```

Example:
```
oModule. MultipleEdit multiplevia_data
```

Rename [Via Planar EM]

Use: Renames a 2.5 via

Syntax:
```
Rename(STRING oldname) // old 2.5 via name
STRING newname) // new 2.5 via name
```

Example:
```
oModule. Rename oldname newname
```

SelectInLayout [Via Planar EM]

Use: Selects and highlights the 2.5 via in the layout editor

Syntax:
```
SelectInLayout(STRING via) // 2.5 via name
```

Example:
```
oModule. SelectInLayout via
```

32 Nexxim Scripting

Nexxim scripting commands should be executed by the **oDesign** object. For example,

```
oDesign.GetModule("DataBlock")
oModule.CommandName <args>
```

The topics for this section include:

Nexxim Netlist Scripting

The following sections present the Nexxim design level scripting commands that are available.

Analyze [Nexxim]

Use:	Simulates the given setup
Command:	None
Syntax:	`Analyze <setup>`
Return Value:	None
Parameters:	`<setup>` Type: string
Example:	`oDesign.Analyze NWA1`

CopyEyeItemAsCommand

Use: Make a QuickEye copy of a VerifEye analysis, or a VerifEye copy of a QuickEye analysis.

Command: Select an analysis in the Project tree, then right-click and select **Copy As QuickEye** or **Copy As VerifEye**.

Syntax: `CopyEyeItemAsCommand <AnalysisArray>`

Return Value: Copy of the analysis.

Parameters: `<AnalysisArray>`

Type: Array of strings // Type of analysis from which to copy.

Example:

```
oDesign.CopyEyeItemAsCommand Array("eye_diagram_sch|Nexx-
im1|Analysis|VerifEyeAnalysis")
oDesign.CopyEyeItemAsCommand Array("eye_diagram_sch|Nexx-
im1|Analysis|QuickEyeAnalysis")
```

DeleteDesignInstance

Use: Delete the design instance

Command: None

Syntax: `DeleteDesignInstance <instance_name>`

Return Value: None

Parameters: `<instance_name>`

Type: string

Example: `oDesign.DeleteDesignInstance <instance_name>`

EditImportData

Use: Edit an imported solution

Command: None

Syntax: `EditImportData <oldname> <newlink> <newpath> <newname> <data> <file>`

Return Value: None

Parameters: `<oldname>`

Type: string

<newlink>

Type: int

<newpath>

Type: string

<newname>

Type: string

<data>

Type: array

<file>

Type: string

Example: `oDesign.EditImportData <oldname> <newlink> <newpath> <newname> <data> <file>`

EditNotes

Use: Edits the notes of a design

Command: None

Syntax: `EditNotes <design_name>`

Return Value: None

Parameters: `<design_name>`

Type: string

Example: `oDesign.EditNotes <design_name>`

ExportForSpice [Nexxim]

Use: Export matrix solution data to a file in the given spice format.

Command: None

Syntax: `ExportForSpice <SolutionVariationKey> <SolnSelectionArray> <Spice Type> <Bandwidth> <FullWaveSpiceFilename> <LumpedElementFilenme> <Unused> <Unused> <PartialFractionFilename> <FittingError> <MaxOrder> <UseCommonGround> <DoPassivityCheck>`

Return Value: None

Parameters: `<SolutionVariationKey>`

Type: string

<SolnSelectionArray>

Type: array

<Spice Type: 0=PSpice, 2=MaxwellSpice, 3=Spectre>
Type: int
<Bandwidth: 0=Low Bandwidth, 1=Broad Bandwidth>
Type: int
<FullWaveSpiceFilename>
Type: string
<LumpedElementFilenme>
Type: string
<Unused>
Type: string
<Unused>
Type: string
<PartialFractionFilename>
Type: string
<FittingError [Default:0.5]>
Type: double
<MaxOrder [Default:200]>
Type: int
<UseCommonGround [Default:FALSE]>
Type: boolean
<DoPassivityCheck [Default:FALSE]>
Type: boolean

Example:

```
oDesign.ExportForSpice "", Array("LNA:LNA"), 0, 0,  _
"C:/Documents and Settings/Marcia.BMT/My Documents/Code
Modifications/Test" & _
" Designs/s-params_fws_fws.lib",  _
"C:/Documents and Settings/Marcia.BMT/My Documents/Code
Modifications/Test" & _
" Designs/s-params_fws_lfws.lib", "", "", "", 0.52, 210,
1, 1
```

ExportForHSpice [Nexxim]

Use: Export matrix solution data to an HSpice subcircuit.

Command: None

Syntax:

```
ExportForHSpice <SolutionVariationKey>
<SolnSelectionArray> <Spice Type> <Bandwidth>
```

```
<FullWaveSpiceFilename> <LumpedElementFilenme> <Unused>
<Unused> <PartialFractionFilename> <Fitting Error>
<Unusedint> <MaxOrder> <UseCommonGround>
<DoPassivityCheck>
```

Return Value: None

Parameters:

`<SolutionVariationKey>`

Type: string

<SolnSelectionArray>

Type: array

<Spice Type: 1=HSPICE>

Type: int

<Bandwidth: 0=Low Bandwidth, 1=Broad Bandwidth>

Type: int

<FullWaveSpiceFilename>

Type: string

<LumpedElementFilenme>

Type: string

<Unused>

Type: string

<Unused>

Type: string

<PartialFractionFilename>

Type: string

<Fitting Error>

Type: double

<Unusedint>

Type: int

<MaxOrder>

Type: int

<UseCommonGround [Default: FALSE]>

Type: boolean

<DoPassivityCheck [Default: FALSE]>

Type: boolean

Example:

```
oDesign.ExportForHSpice "", Array("Setup 1:Sweep 1"), 1,
0, "C:/Projects/MyTRL_fws.sp", "", "", "", "", 0.5, 0,
200, 0, 0
```

ExportNetlist [Nexxim]

Use: Exports a netlist solution

Command: None

Syntax: `ExportNetlist <solution> <filename>`

Return Value: None

Parameters: `<solution>`

Type: string

<filename>

Type: string

Example: `oDesign.ExportNetlist <solution> <filename>`

GetModule [Nexxim]

Use: Get the IDispatch for the specified module.

Command: None

Syntax: `GetModule <"modulename"> <object_variable>`

Return Value: Module IDispatch

Parameters: `<"modulename">`

Type: string

<object_variable>

Type: string

Example: `oDesign.GetModule "modulename" object_variable`

GetName [Nexxim]

Use: Get the name of the active design.

Command: None

Syntax: `GetName <namestring_variable>`

Return Value: String

Parameters: `<namestring_variable>`

Type: string

Example: `oDesign.GetName namestring_variable`

GetActiveEditor

Use: Get the name of the active editor.

Command: None

Syntax: `GetActiveEditor <object_variable>`

Return Value: String

Parameters: `<object_variable>`

Type: string

Example: `oDesign.GetActiveEditor <object_variable>`

Note that GetActiveEditor returns NULL if the editor is open in a non-active window.

GetEditor [Nexxim]

Use: Get the named editor without activating it.

Command: None

Syntax: `GetEditor <editorName object_variable>`

Return Value: String

Parameters: `<editorName>`

Type: string

<object_variable>

Type: string

Example: `oDesign.GetEditor <editorName> <object_variable>`

GetResultsDirectory [Nexxim]

Use: Gives access to the directory containing the specified Nexxim solution files.

Command: None

Syntax: `GetResultsDirectory ([in] BSTR solname, [in] BSTR variation, [in] BOOL finalDir, [out, retval] BSTR* dirname)`

Return Value: <dirname>

Type: String // Directory name. (The returned directory path does not contain a trailing slash.).

Parameters: `<solname>`

Type: string

<variation>

Type: string

<finalDir>

Type: boolean // True if the final directory is desired; False if the working/temporary directory is desired.

Example:

```
Dim dirName
Set oDesign = oProject.GetActiveDesign() dirName = oDesign.GetResultsDirectory ("Trans", "res='1000'", false)
```

ImportData [Nexxim]

Use: Imports a network solution

Command: None

Syntax:

```
ImportData <data> <filename> <linktofile>
```

Return Value: None

Parameters:

<data>

Type: string // data to be imported

<filename>

Type: string // name of the file

<linktofile>

Type: int // link to the file

Example:

```
oDesign.ImportData <data> <filename> <linktofile>
```

ImportDataFilePath [Nexxim]

Use: Returns the file path of the imported solution given an imported solution name. If the solution is not found it returns an empty string.

Command: None

Syntax:

```
ImportDataFilePath <solution_name>
```

Return Value: File path of imported solution, empty string if solution not found in the design.

Type: string

Parameters:

<solution_name>

Type: string

Example:

```
oDesign.ImportDataFilePath <solution_name>
```

InsertDesign [Nexxim]

Use: Insert a design

Command: None

Syntax: `InsertDesign <type>, <name>, <stationerypath>, <parentname>`

Return Value: None

Parameters: `<type>`

Type: string

<name>

Type: string

<stationerypath>

Type: string

<parentname>

Type: string

Example: `oDesign.InsertDesign <type>, <name>, <stationerypath>, <parentname>`

where, <type> is NULL for top-level design

PasteItemCommand

Use: Paste tree items, such as Altrasim Solution and Solve Setups pasted to the Analysis Tree, or Frequency Sweeps pasted to the Solve Setup Tree.

Command: Right-click on the Analysis item in the Project tree and select Paste.

Syntax: `PasteItemCommand <ItemPath>`

Return Value: None

Parameters: `<ItemPath>`

Type: string

Example: `oDesign.PasteItemCommand "Project1|Nexxim1|Analysis"`

Redo [Nexxim]

Use: Redo the last operation

Command: Edit>Redo

Syntax: `Redo`

Return Value: None

Parameters: None

Example: oDesign.Redo

RenameDesignInstance [Nexxim]

Use: Rename the design instance

Command: None

Syntax: RenameDesignInstance <oldname> <newname>

Return Value: None

Parameters: <oldname>

Type: string

<newname>

Type: string

Example: oDesign.RenameDesignInstance <oldname> <newname>

RenameImportData [Nexxim]

Use: Rename an imported solution

Command: None

Syntax: RenameImportData <oldname> <newname>

Return Value: None

Parameters: <oldname>

Type: string

<newname>

Type: string

Example: oDesign.RenameImportData <oldname> <newname>

SetActiveEditor [Nexxim]

Use: Set the active editor to the named one. (Activates window or opens window if necessary).

Command: None

Syntax: SetActiveEditor <editorName> <object_variable>

Return Value: None

Parameters: <editorName>

Type: string

<object_variable>

Type: string

Example: `oDesign.SetActiveEditor <editorName> <object_variable>`

StartAnalysis [Nexxim]

Use: Simulates all setups

Command: None

Syntax: `StartAnalysis`

Return Value: None

Parameters: `None`

Example: `oDesign.StartAnalysis`

Undo [Nexxim]

Use: Undo the last operation

Command: **Edit>Undo**

Syntax: `Undo`

Return Value: None

Parameters: `None`

Example: `oDesign.Undo`

UseCircuitSParameterDefinition

Use: Selects whether the circuit or high-frequency definition of S-parameters is used.

Command: None

Syntax: `UseCircuitSparameterDefinition <boolean_operator>`

Return Value: None

Parameters: `<boolean_operator>`

Type: string

Example: `oDesign.UseCircuitSparameterDefinition true`

`oDesign.UseCircuitSparameterDefinition true`

Nexxim Data Block Commands

The following sections present the commands that are available in the “DataBlock” Module and can be used with the Module command interface:

For example,

```
oDesign.GetModule("DataBlock")
oModule.CommandName <args>
```

Nexxim data clock commands are listed below.

AddTemperatureDataBlock

Use:	Add a Temperature Data Block
Command:	None
Syntax:	`AddTemperatureDataBlock`
Return Value:	None

Parameters: None
Example: oModule.AddTemperatureDataBlock

AddLibRefDataBlock

Use: Add a LibRef Data Block
Command: None
Syntax: AddLibRefDataBlock
Return Value: None
Parameters: None
Example: oModule.AddLibRefDataBlock

AddNetlistDataBlock

Use: Add a NetList Data Block
Command: None
Syntax: AddNetlistDataBlock
Return Value: None
Parameters: None
Example: oModule.AddNetlistDataBlock

AddPrintToAuditDataBlock

Use: Add a PrintToAudit Data Block
Command: None
Syntax: AddPrintToAuditDataBlock
Return Value: None
Parameters: None
Example: oModule.AddPrintToAuditDataBlock

AddStateVariableDataBlock

Use: Add a StateVariable Data Block
Command: None

Syntax: `AddStateVariableDataBlock`
Return Value: None
Parameters: `None`
Example: `oModule.AddStateVariableDataBlock`

AddSubstrateDataBlock

Use: Add a Substrate Data Block
Command: None
Syntax: `AddSubstrateDataBlock`
Return Value: None
Parameters: `None`
Example: `oModule.AddSubstrateDataBlock`

AddDeviceNoiseDataBlock

Use: Add a DeviceNoise Data Block
Command: None
Syntax: `AddDeviceNoiseDataBlock`
Return Value: None
Parameters: `None`
Example: `oModule.AddDeviceNoiseDataBlock`

EditTemperatureDataBlock

Use: Edit a Temperature Data Block
Command: None
Syntax: `EditTemperatureDataBlock`
Return Value: None
Parameters: `None`
Example: `oModule.EditTemperatureDataBlock`

EditLibRefDataBlock

Use: Edit a LibRef Data Block

Command: None

Syntax: `EditLibRefDataBlock`

Return Value: None

Parameters: `None`

Example: `oModule.EditLibRefDataBlock`

EditNetlistDataBlock

Use: Edit a Netlist Data Block

Command: None

Syntax: `EditNetlistDataBlock`

Return Value: None

Parameters: `None`

Example: `oModule.EditNetlistDataBlock`

EditPrintToAuditDataBlock

Use: Edit a PrintToAudit Data Block

Command: None

Syntax: `EditPrintToAuditDataBlock`

Return Value: None

Parameters: `None`

Example: `oModule.EditPrintToAuditDataBlock`

EditStateVariableDataBlock

Use: Edit a StateVariable Data Block

Command: None

Syntax: `EditStateVariableDataBlock`

Return Value: None

Parameters: `None`

Example: `oModule.EditStateVariableDataBlock`

EditSubstrateDataBlock

Use: Edit a Substrate Data Block
Command: None
Syntax: `EditSubstrateDataBlock`
Return Value: None
Parameters: `None`
Example: `oModule.EditSubstrateDataBlock`

EditDeviceNoiseDataBlock

Use: Edit a DeviceNoise Data Block
Command: None
Syntax: `EditDeviceNoiseDataBlock`
Return Value: None
Parameters: `None`
Example: `oModule.EditDeviceNoiseDataBlock`

GetTemperatureDataBlock

Use: Returns the TemperatureDataBlock name
Syntax: `GetTemperatureDataBlock`
Return Value: String
Parameters: `None`
Example:
```
Dim tempStr
tempStr = oModule.GetTemperatureDataBlock
```

GetAllLibRefDataBlocks

Use: Returns a name list of all library reference data blocks in the Nexxim Design
Syntax: `GetAllLibRefDataBlocks`
Return Value: Array of Strings
Parameters: `None`
Example:
```
Dim librefArr
librefArr = oModule.GetAllLibRefDataBlocks
```

GetAllNetlistDataBlocks

Use: Returns a name list of all netlist data blocks in the Nexxim Design

Syntax: `GetAllNetlistDataBlocks`

Return Value: Array of Strings

Parameters: `None`

Example:

```
Dim netlistArr
netlistArr = oModule.GetAllNetlistDataBlocks
```

GetAllSubstrateDataBlocks

Use: Returns a name list of all substrate data blocks in the Nexxim Design

Syntax: `GetAllSubstrateDataBlocks`

Return Value: Array of Strings

Parameters: `None`

Example:

```
Dim subsArr
subsArr = oModule.GetAllSubstrateDataBlocks
```

Remove [Nexxim]

Use: Removes a Data Block

Command: None

Syntax: `Remove <Data Block>`

Return Value: None

Parameters: `Data Block`

Example: `oModule.Remove <Data Block>`

Rename [Nexxim]

Use: Renames a Data Block

Command: None

Syntax: `Rename <Data Block>`

Return Value: None

Parameters: `Data Block`

Example: `oModule.Rename <Data Block>`

Nexxim Simulation Setup Commands

The following sections present the commands that are available in the “SimSetup” module:
For example,
oDesign.GetModule("SimSetup")
oModule.CommandName <args>

Add [Nexxim]

Use: Adds a new setup

Syntax:
```
Add(
STRING newsetup) // new setup name
```

Example:
```
oModule. Add <newsetup>
```

AddSweep [Nexxim]

Use: Adds a sweep to a setup

Syntax:
```
AddSweep(
STRING setup) // setup name
STRING newsweep) // new sweep name
```

Example:

```
oModule. AddSweep <setup> <newsweep>
```

Analyze [Nexxim]

Use: Simulates the given setup

Syntax:
```
Analyze(
STRING setup1) // setup name
```

Example:
```
oModule. Analyze <setup1>
```

AnalyzeSweep [Nexxim]

Use: Simulates the given sweep

Syntax:
```
AnalyzeSweep(
STRING setup1) // setup name
STRING sweep1) // sweep name
```

Example:
```
oModule. AnalyzeSweep <setup1> <sweep1>
```

Delete [Nexxim]

Use: Removes a simulation setup

Syntax:
```
Delete(
STRING name) // The specified setup to delete
```

Example:
```
oModule. Delete <name>
```

DisplayBiasPointInfo

Use: Display voltage/current bias

Command: None

Syntax:
```
DisplayBiasPointInfo <display_preference>
```

Return Value: None

Parameters:
```
<display_preference>
```
Type: string

Example:
```
oDesign.DisplayBiasPointInfo "1"
Where display_preference is one of the following:
0 : hide the info
```

```
1 : display voltage
2 : display current
3 : display both
```

DeleteSweep [Nexxim]

Use: Removes a sweep from a setup

Syntax:
```
DeleteSweep(
STRING setup1) // The specified setup
STRING sweep1) // The specified sweep
```

Example:
```
oModule. DeleteSweep <setup1> <sweep1>
```

DynamicMeshOverlays

Use: Turns on dynamic mesh overlays

Syntax:
```
DynamicMeshOverlays(
STRING setup) // setup name
```

Example:
```
oModule. DynamicMeshOverlay <setup>
```

Edit [Nexxim]

Use: Edit a setup

Syntax:
```
Edit(
STRING setup) // setup name
ARRAY solvesetup_data) // setup data
```

Example:
```
oModule. Edit <setup> <solvesetup_data>
```

EditSweep [Nexxim]

Use: Edit a sweep

Syntax:
```
EditSweep(
STRING setup) // setup name
ARRAY solvesetup_data) // setup data
```

Example:
```
oModule. EditSweep <setup> <solvesetup_data>
```

GetAllSolutionSetups [Nexxim]

Use: Returns an array of the names of the simulation setups in the design

Syntax:
```
GetAllSolutionSetups( )
```

Example:
```
Set oModule = oDesign.GetModule("SimSetup")
Dim names
names = oModule.GetAllSolutionSetups()
MsgBox("GetAllSolutionSetups:" & Chr(13) & Join(names,",
"))
```

ListVariations [Nexxim]

Use: Get a list of solution variations for a given solution

Syntax:
```
ListVariations(
STRING SolutionName) // solution name
```

Example:
```
oModule.ListVariations("NWA1")
```

RefreshMeshOverlays

Use: Refreshes mesh display for a setup

Syntax:
```
RefreshMeshOverlays(
STRING setup) // setup name
```

Example:
```
oModule.RefreshMeshOverlays <setup>
```

RenameSweep [Nexxim]

Use: Renames a NexximSimulation setup

Syntax:
```
RenameSweep(
STRING oldname) // old setup name
STRING newname) // new setup name
```

Example:
```
oModule. RenameSweep <oldname> <newname>
```

Nexxim Linear Network Analysis

The following linear network analysis commands are available.

AddAnalysisOptions

AddSimulationSetup

EditAnalysisOptions
EditSimulationSetup
ExportForSpice
ExportForHSpice
RemoveAnalysisOptions
RemoveSimulationSetup
RenameAnalysisOptions
RenameSimulationSetup

Some of the following commands are available only in the "SolveSetups" module:
For example,

```
oDesign.GetModule("SolveSetups")
oModule.CommandName <args>
```

AddAnalysisOptions [Nexxim]

Use: Add an analysis option
Command: None
Syntax: `AddAnalysisOptions <Option_Data>`
Return Value: None
Parameters: `<Option_Data>`
Example: `oModule.AddAnalysisOptions <Option_Data>`

AddSimulationSetup [Nexxim]

Use: Adds a new simulation setup
Command: None
Syntax: `AddSimulationSetup <Simulationsetup_data>`
Return Value: None
Parameters: `<Simulationsetup_Data>`
Example: `oModule.AddSimulationSetup <SimulationSetup_Data>`

EditAnalysisOptions Nexxim]

Use: Edit an analysis option

Command: None

Syntax: `EditAnalysisOptions <Option_Data>`

Return Value: None

Parameters: `<Option_Data>`

Example: `oModule.EditAnalysisOptions <Option_Data>`

EditSimulationSetup [Nexxim]

Use: Edit a simulation setup

Command: None

Syntax: `EditSimulationSetup <Simulationsetup_data>`

Return Value: None

Parameters: `<Simulationsetup_Data>`

Example: `oModule.EditSimulationSetup <SimulationSetup_Data>`

ExportForSpice [Nexxim]

Use: Export matrix solution data to a file in the given spice format.

Command: None

Syntax:
```
ExportForSpice <SolutionVariationKey>
<SolnSelectionArray> <Spice Type> <Bandwidth>
<FullWaveSpiceFilename> <LumpedElementFilenme> <Unused>
<Unused> <PartialFractionFilename> <FittingError>
<MaxOrder> <UseCommonGround> <DoPassivityCheck>
```

Return Value: None

Parameters: `<SolutionVariationKey>`

Type: string

<SolnSelectionArray>

Type: array

<Spice Type: 0=PSpice, 2=MaxwellSpice, 3=Spectre>

Type: int

<Bandwidth: 0=Low Bandwidth, 1=Broad Bandwidth>

Type: int

<FullWaveSpiceFilename>

Type: string

<LumpedElementFilenme>

Type: string

<Unused>

Type: string

<Unused>

Type: string

<PartialFractionFilename>

Type: string

<FittingError [Default:0.5]>

Type: double

<MaxOrder [Default:200]>

Type: int

<UseCommonGround [Default:FALSE]>

Type: boolean

<DoPassivityCheck [Default:FALSE]>

Type: boolean

Example:

```
oDesign.ExportForSpice "", Array("LNA:LNA"), 0, 0,  _
"C:/Documents and Settings/Marcia.BMT/My Documents/Code
Modifications/Test" & _
" Designs/s-params_fws_fws.lib",  _
"C:/Documents and Settings/Marcia.BMT/My Documents/Code
Modifications/Test" & _
" Designs/s-params_fws_lfws.lib", "", "", "", 0.52, 210,
1, 1
```

ExportForHSpice [Nexxim]

Use: Export matrix solution data to an HSpice subcircuit.

Command: None

Syntax:

```
ExportForHSpice <SolutionVariationKey>
<SolnSelectionArray> <Spice Type> <Bandwidth>
<FullWaveSpiceFilename> <LumpedElementFilenme> <Unused>
<Unused> <PartialFractionFilename> <Fitting Error>
<Unusedint> <MaxOrder> <UseCommonGround>
<DoPassivityCheck>
```

Return Value: None

Parameters:

<SolutionVariationKey>
Type: string
<SolnSelectionArray>
Type: array
<Spice Type: 1=HSPICE>
Type: int
<Bandwidth: 0=Low Bandwidth, 1=Broad Bandwidth>
Type: int
<FullWaveSpiceFilename>
Type: string
<LumpedElementFilename>
Type: string
<Unused>
Type: string
<Unused>
Type: string
<PartialFractionFilename>
Type: string
<Fitting Error>
Type: double
<Unusedint>
Type: int
<MaxOrder>
Type: int
<UseCommonGround [Default: FALSE]>
Type: boolean
<DoPassivityCheck [Default: FALSE]>
Type: boolean

Example:

```
oDesign.ExportForHSpice "", Array("Setup 1:Sweep 1"), 1,
0, "C:/Projects/MyTRL_fws.sp", "", "", "", "", 0.5, 0,
200, 0, 0
```

RemoveAnalysisOptions [Nexxim]

Use: Remove an analysis option

Command: None
Syntax: `RemoveAnalysisOptions <Option_Data>`
Return Value: None
Parameters: `<Option_Data>`
Example: `oModule.RemoveAnalysisOptions <Option_Data>`

RemoveSimulationSetup [Nexxim]

Use: Remove a simulation setup
Command: None
Syntax: `RemoveSimulationSetup <Simulationsetup_data>`
Return Value: None
Parameters: `<Simulationsetup_Data>`
Example: `oModule.RemoveSimulationSetup <SimulationSetup_Data>`

RenameAnalysisOptions[Nexxim]

Use: Rename an analysis option
Command: None
Syntax: `RenameAnalysisOptions <oldname> <newname>`
Return Value: None
Parameters: `<oldname>`
Type: string
<newname>
Type: string
Example: `oModule.RenameAnalysisOptions <oldname> <newname>`

RenameSimulationSetup [Nexxim]

Use: Rename a simulation setup
Command: None
Syntax: `RenameSimulationSetup <oldname> <newname>`
Return Value: None
Parameters: `<oldname>`
Type: string
<newname>
Type: string

Example: `oModule.RenameSimulationSetup "oldname", "newname"`

Nexxim Ports And Sources Commands

The following Ports and Sources commands for Nexxim are available.

ChangePortProperty

ChangeSourceProperty

DeletePort

DeleteSource

GetAllPorts

GetAllSources

RenameSource

ChangePortProperty [Nexxim]

Use: Change a port property

Command: None

Syntax: `ChangePortProperty <Port Name> <Port Info> <Port Properties>`

Return Value: None

Parameters: `<Port Name> - Type: string`

<Port Info> - Type: array

<Port Properties> - Type:array

Example:

```
oDesign.ChangePortProperty "Port2", Array("NAME:Port2",
"IIPortName:=", "Port2", "SymbolType:=",  _
  0), Array(Array("NAME:Properties", Array("NAME:New-
Props", Array("NAME:term", "PropType:=",  _
  "TextProp", "OverridingDef:=", true, "Value:=", "Mod-
el1")), Array("NAME:ChangedProps", Array("NAME:Termina-
tionData", "Value:=",  _
  "Zo"), Array("NAME:pnum", "Value:=", "2"),
Array("NAME:noisetemp", "Value:=", "16.85")))
```

ChangeSourceProperty [Nexxim]

Use: Change a source property

Command: None

Syntax: `ChangeSourceProperty <Source_Name> <Source_Data>`

Return Value: None

Parameters: `<Source_Name>`

Type: array

<Source_Data>

Type: array

Example:

```
oDesign.ChangeSourceProperty "Sinusoidal1",
Array(Array("NAME:Properties",
Array("NAME:NewProps", _
  Array("NAME:noise", "PropType:=", "TextProp", "Over-
ridingDef:=", true, "UserDef:=", true, "Value:=",
"SourceNoise1"), _
  Array("NAME:mod", "PropType:=", "TextProp", "Over-
ridingDef:=", true, "UserDef:=", true, "Value:=",
"SourceModulation1"))), _
  Array("NAME:DataBlocks", Array("NAME:NewDataBlk",
Array("NAME:DataBlock", "Name:=",   "SourceNoise1", _
 "noisedata:=", Array("FDEV=1Hz ", "RFUP=1 ", "RFLO=2 ",
"NCOR=2 2 ")), _
Array("NAME:DataBlock", "Name:=", "SourceModulation1",
"ModulationSourceDatas:=", Array( _
  "IS95", "Butterworth", "Br=1.2288MHz", "Dly=0.5",
"Iasc=1", "Qasc=1")))))
```

DeletePort [Nexxim]

Use: Delete a port

Command: None

Syntax: `DeletePort <Port_Data>`

Return Value: None

Parameters: `<Port_Data>`

Type: array

Example: `oDesign.DeletePort <Port_Data>`

DeleteSource [Nexxim]

Use: Delete the source

Command: None

Syntax: `DeleteSource <srcname>`

Return Value: None

Parameters: `<srcname>`

Type: string

Example: `oDesign.DeleteSource <srcname>"`

GetAllPorts [Nexxim]

Use: Returns a list of all port names in the Nexxim Design

Syntax: `GetAllPorts`

Return Value: Array of Strings

Parameters: `None`

Example:
```
Dim portsArr
portsArr= oDesign.GetAllPorts
```

GetAllSources [Nexxim]

Use: Returns a list of all sources names in the Nexxim Design

Syntax: `GetAllSources`

Return Value: Array of Strings

Parameters: `None`

Example:
```
Dim sourceArr
sourceArr = oDesign.GetAllSources
```

RenameSource [Nexxim]

Use: Rename the source

Command: None

Syntax: `RenameSource <oldname> <newname>`

Return Value: None

Parameters: `<oldname>`

Type: string

<newname>

Type: string

Example: `oDesign.RenameSource "oldname", "newname"`

Nexxim Component Manager Commands

The following Component Manager commands for Nexxim are available.

ImportSandWComponent

ImportXparamComponent

ImportSandWComponent

Use: Import S-element or W-element

Command: None

Syntax: `ImportSandWComponent <Files><Options>`

Return Value: Name of the component that is created

Parameters: `<Files>, <Options>`

Type: array

```
oComponentManager.ImportSandWComponent
// Array of files selected
Array("NAME:Files", "Files:=",Array(_
"C:\Projects\100ohm.s2p",_
"C:\Projects\50ohm.s2p",_
"C:\Projects\25ohm.s2p",_
// S-element options
Array("NAME:Options",
"IsWElement:=", false, // true=W-element, false=S-element
"NumPortsorLines:'", 2 // ports for S, lines for W
"CustomNetlist:=", "INTDATTYP=MA HIGHPASS=10 LOWPASS=10_
convolution=1 enforce_passivity=1",
// Netlist representing selected options for S-element
"CreateArray:="' true // true=file array false=no array
"PinConfig:="', 0, // 0=Odd/Even, 1=I/I+N
"AddCommonRef:="', 1 // 0=implied to gnd, 1=show ref pin
)
```

ImportXparamComponent

Use: Import X-Parameter component

Command: None

Syntax: `ImportXparamComponent <File>`

Return Value: Name of the component that is created

Parameters: `<File>`

Type: string

Example: `oComponentManager.ImportXparamComponent <filename>`

33 Example Scripts

Variable Helix Script

Following is a sample HFSS script that creates a tapered helix. Tapering helices is not supported from the HFSS interface. The script includes comment lines, which are preceded by an apostrophe ('), that offer explanations for each subsequent line or lines.

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp  = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
Set oProject = oDesktop.GetActiveProject()
Set oDesign = oProject.GetActiveDesign()
Set oEditor = oDesign.SetActiveEditor("3D Modeler")

' Declare the arrays and variables needed for building the polyline.
'
Dim points(), segments()
Dim NumPoints, R(2), P(2), PointsPerTurn, Turns, Units
'
' Establish the constant Pi.
Pi = 4*Atn(1)
' Retrieve the variable helix parameters from the user.
' Start with the input for unit selection.
'
Units = InputBox("Select the units:"&Chr(13)& _
    "(cm,mm,um,in,mil)", "Variable Helix","mil",50,50)
'
' Check to make sure it is a valid unit.
'
Select Case Units
    Case "m"
      Units = ""
    Case "cm"
```

```
    Case "mm"
    Case "um"
    Case "in"
    Case "mil"
    Case Else
      MsgBox("Invalid Units - defaults to m")
      Units = ""
End Select
'
' Obtain the other user-defined parameters.
'
Turns = InputBox("Select the number of turns (must be
    integer):","Variable Helix", 2,50,50)
PointsPerTurn = InputBox("Select the points per turn:", _
    "Variable Helix",16,50,50)
R(0) = InputBox("Select the initial Radius: ", _
    "Variable Helix",10,50,50)
R(1) = InputBox("Select the final Radius: ", _
    "Variable Helix",10,50,50)
P(0) = InputBox("Select the initial Pitch: ", _
    "Variable Helix", 4,50,50)
P(1) = InputBox("Select the final Pitch: ", _
    "Variable Helix", 4,50,50)
NumPoints = Turns*PointsPerTurn
'
' Initialize the points and segments arrays.
'
Redim points(NumPoints+1)
Redim segments(NumPoints)
points(0) = "NAME:PolylinePoints"
segments(0) = "NAME:PolylineSegments"
'
' Build the Point and Segment Arrays needed in the HFSS polyline call.
'
For n = 1 To (NumPoints+1)
```

```
    Angle = (n-1)*2*Pi/PointsPerTurn
    Radius = R(0) + ((n-1)/NumPoints)*(R(1)-R(0))
    Pitch = P(0) + ((n-1)/NumPoints)*(P(1)-P(0))
    Rise = (n-1)*Pitch/PointsPerTurn

    XValue = cstr(Radius*cos(Angle)) & Units
    YValue = cstr(Radius*sin(Angle)) & Units
    ZValue = cstr(Rise) & Units
    points(n) = Array("NAME:PLPoint", "X:=", XValue, "Y:=", _
      YValue, "Z:=", ZValue)
'
' Create the line segments between each of the pairs of points.
'
    If n<=NumPoints Then
      segments(n) = Array("NAME:PLSegment", "SegmentType:=", _
      "Line", "StartIndex:=", (n-1), "NoOfPoints:=", 2)
    End If
Next
'
' Create the polyline.
'
oEditor.CreatePolyline _
    Array("NAME:PolylineParameters", "IsPolylineCovered:=", true, _
    "IsPolylineClosed:=", false, points, segments), _
    Array("NAME:Attributes", "Name:=", "Line_Helix","Flags:=", _
    "", "Color:=", "(132 132 193)", "Transparency:=",0.4, _
    "PartCoordinateSystem:=", "Global", "MaterialName:=", _
    "vacuum", "SolveInside:=", true)
'
' Create the helix cross-section.
'
oEditor.CreateCircle _
    Array("NAME:CircleParameters", "IsCovered:=", true, "XCenter:=",_
    cstr(R(0))&Units, "YCenter:=", 0, "ZCenter:=", 0, "Radius:=", _
    "1"&Units, "WhichAxis:=", "Y"), _
```

```
    Array("NAME:Attributes", "Name:=", "Circle_Helix", "Flags:=", _
    "", "Color:=", "(132 132 193)", "Transparency:=", 0.4, _
    "PartCoordinateSystem:=", "Global", "MaterialName:=", "vacuum", _
    "SolveInside:=", true)
'
' Sweep the cross-section along the path.
'
oEditor.SweepAlongPath _
    Array("NAME:Selections", "Selections:=", _
    "Circle_Helix,Line_Helix"),
    Array("NAME:PathSweepParameters", "DraftAngle:=", "0deg", _
    "DraftType:=", "Round", "TwistAngle:=", "0deg")
```

HFSS Data Export Script

Following is a simple script that demonstrates how to export data from HFSS and save it to a file. The output data in the example script is in 3 columns. The first column is freq in GHz, the second is the Real part of S11, and the third is the Img part of S11. It uses a tab-delimited format. The HFSS output is done using output variables.

The frequency sweep data must be entered correctly. If it is incorrect, the script will request a freq point that does not exist and execution will stop.

The script includes comment lines, which are preceded by an apostrophe ('), that offer explanations for each subsequent line or lines.

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp  = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()
set oProject = oDesktop.GetActiveProject
set oDesign =  oProject.GetActiveDesign()
Dim oFS,ofile,x,y,z,path,range,
Dim arr2,del_f,freq,cfreq,val,temp,stn,stw,i,line
'
' Input the desired file name.
'
path = inputbox("Input the file name" &chr(13) & _
"Note: If you do not specify a path the file will " & _
"be placed in the script directory", _
"File","C:\hfss_export.txt",50,50)
'
' If the user clicks Cancel, the path will be blank, in which case the script should just exit.
If path <>"" then
'
' Create the file, open it for data entry, and output the column labels.
'
    Set oFS = CreateObject("Scripting.FileSystemObject")
```

```
    Set ofile = oFS.CreateTextFile (path)
    line = "Freq" & chr(9) & "RE(S11)" & chr(9) & "IMG(S11)"
    ofile.WriteLine line
'
' Input the needed freq, solution, and sweep data and clean it up.
'
    msgbox("For the following input make sure it matches " & _
    "the frequencies defined in your sweep")
    range = inputbox("Input the range of frequencies in GHz " & _
    "and number of points",_
    "Frequency","8,12,10",50,50)
'
' The following 2 lines define the 2 output variables.
'
    oDesign.CreateOutputVariable "re_S", "re(S(port1,port1))"
    oDesign.CreateOutputVariable "im_S", "im(S(port1,port1))"
    arr = split (range, ",")
    arr(0) = Trim(arr(0))
    arr(1) = Trim(arr(1))
    arr(2) = Trim(arr(2))
    if cint(arr(2)) <> 1 then
      del_f = (arr(1)-arr(0))/(arr(2)-1)
    else
      del_f = 0
    end if
    temp = InputBox("Input the Setup and Sweep number to use:"_
    & chr(13) & "(e.g. input 1,2 for Setup1 and Sweep2)", _
    "Solution Data","1,1",50,50)
    arr2 = split(temp,",")
    stn = arr2(0)
    swn = arr2(1)
    stn = Trim(stn)
    swn = Trim(swn)
'
' Loop through the freq points.
'
```

```
    for i=1 to arr(2) step 1
      freq = arr(0) + (cint(i)-1)*del_f
      x=freq
      cfreq="Freq='" & freq & "Ghz'"
'
' Get the values of the output variables for the desired freq.
'
      val  = oDesign.GetOutputVariableValue("re_S","Setup" & _
      stn & " :Sweep" & swn,cfreq, "")
      y = val
      val  = oDesign.GetOutputVariableValue("im_S","Setup" & _
      stn & " : Sweep" & swn,cfreq, "")
      z = val
'
' Create the line of text to send to the file and write it to the file.
'
      line = x & chr(9) & y & chr(9) & z
      ofile.WriteLine line
    Next
'
' Delete the 2 output variables before finishing.
'
    oDesign.DeleteOutputVariable "re_S"
    oDesign.DeleteOutputVariable "im_S"
'
' Close the file.
'
    ofile.close
End if
```

Index

Numbers

A

B

C

D

E

F

G

H

I

P

Q

R

S

T

U

V

W

X

Z