Introduction to Scripting in HFSS

ANSOFT

December 2007

Ansoft Corporation
225 West Station Square Drive
Suite 200
Pittsburgh, PA 15219
USA
Phone: 412-261-3200
Fax: 412-471-9427

New editions of this manual will incorporate all material updated since the previous edition. The manual printing date, which indicates the manual's current edition, changes when a new edition is printed. Minor corrections and updates which are incorporated at reprint do not cause the date to change.

Update packages may be issued between editions and contain additional and/or replacement pages to be merged into the manual by the user. Note that pages which are rearranged due to changes on a previous page are not considered to be revised.

Edition	Date	Software Version
1	June 2003	9.0
2	May 2005	10
3	March 2006	10
4	Jun 2006	10.1
5	September 2007	11.0
6	December 2007	11.1

Getting Help

Ansoft Technical Support

To contact Ansoft technical support staff in your geographical area, please log on to the Ansoft corporate website, http://www.ansoft.com, click the **Contact** button, and then click **Support**. Your Ansoft sales engineer may also be contacted in order to obtain this information.

E-mail can work well for technical support. All Ansoft software files are ASCII text and can be sent conveniently by e-mail. When reporting difficulties, it is extremely helpful to include very specific information about what steps were taken or what stages the simulation reached. This allows more rapid and effective debugging.

Context-Sensitive Help

To access online help from the HFSS user interface, do one of the following:

- To open a help topic about an HFSS menu command, press **Shift**+**F1**, and then click the command or toolbar icon.
- To open a help topic about an HFSS dialog box, open the dialog box, and then press **F1**.

Table of Contents

12. Reporter Editor Script Commands

13. Boundary and Excitation Module Script Com-

mands

14. Mesh Operations Module Script Commands

15. Analysis Module Script Commands

16. Optimetrics Module Script Commands

21. Example Scripts

1 Introduction to VBScript

HFSS 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 HFSS interface. After recording the script from within HFSS, you can then modify it if necessary using a text editor.

Although HFSS 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 HFSS scripting documentation refers to VBScript format only.

This chapter provides an overview of key VBScript components. 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 HFSS 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 HFSS script examples, see Chapter 20, *Example Scripts*.

VBScript Variables

A VBScript variable is a placeholder representing information that may change during the time your script is running. 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"
```

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()`.

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
```

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.

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.

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.

Copyright and Trademark Notices

Ansoft Corporation
225 West Station Square Drive
Suite 200
Pittsburgh, PA 15219
(412) 261 - 3200

2 HFSS and VBScript

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

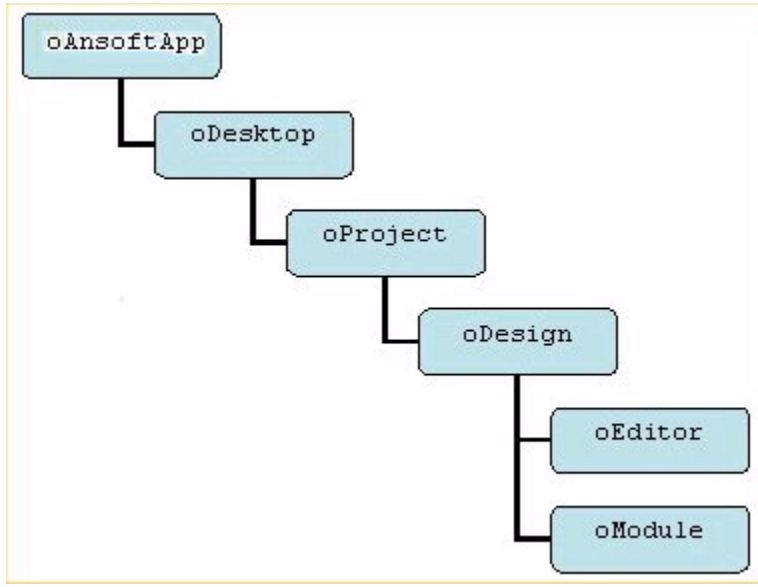
- HFSS script variables.
- Recording, running, pausing, resuming, and stopping a script.
- Modifying a script for easier playback.
- HFSS scripting conventions, including script command syntax used in this guide, named arguments, and setting numerical values.
- Executing a script from within a script.
- Modifying properties.

Overview of HFSS Script Variables

When you record an HFSS script, the beginning of the script looks like the following:

```
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")
```

The lines above define the variables used by HFSS in the script and assign values to the variables. The variables are used in the following hierarchy:



Class hierarchy of variables.

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 Chapter 3, *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:

- Chapter 4, *Project Object Script Commands*
- Chapter 5, *Material Script Commands*
- Chapter 6, *Property Script Commands*
- Chapter 7, *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:

- Chapter 8, *Design Object Script Commands*
- Chapter 9, *Output Variable Script Commands*
- *Chapter 11, Reporter Editor Script Commands*

oEditor

The oEditor object corresponds to an editor, such as the 3D Modeler. 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")
```

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 HFSS design object oDesign.

See Chapter 10, *3D Modeler Editor Script Commands*, for details about the script commands recognized by the oEditor object and Chapter 11, *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 AnsoftHfss product scripting supports the following modules:

Module	Name in Script	Chapter
Boundary/Excitations Module Corresponds to the **Boundaries** and **Excitations** branches in the project tree.	"BoundarySetup"	Chapter 12, *Boundary and Excitation Module Script Commands*
Mesh Operations Module Corresponds to the **Mesh Operations** branch in the project tree.	"MeshSetup"	Chapter 13, *Mesh Operations Module Script Commands*
Analysis Module Corresponds to the **Analysis** branch in the project tree.	"AnalysisSetup"	Chapter 14, *Analysis Module Script Commands*
Optimetrics Module Corresponds to the **Optimetrics** branch in the project tree.	"Optimetrics"	Chapter 15, *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"	Chapter 16, *Solutions Module Script Commands*
Field Overlays Module Corresponds to the **Field Overlays** branch in the project tree.	"FieldsReporter"	Chapter 17, *Field Overlays Module Script Commands*
Radiation Module Corresponds to the **Radiation** branch in the project tree.	"RadField"	Chapter 18, *Radiation Module Script Commands*

Recording a Script

Once you start to record a script, your subsequent actions are added to the script. Each interface command has one or more associated script commands that are recorded to the script. The script is recorded to a text file in .vbs (VBScript) file format.

1. Click **Tools**>**Record Script**.
   The **Save As** dialog box appears.
2. Use the file browser to locate the folder in which you want to save the script, such as C:\Ansoft\HFSS9\Scripts, and then double-click the folder's name.
3. Type the name of the script in the **File name** text box, and then click **Save**.
   The script is saved in the folder you selected by the file name *filename.vbs*.
4. Perform the steps that you want to record.
5. When you have finished recording the script, click **Stop Script Recording** on the **Tools** menu.

Stopping Script Recording

- Click**Tools**>**Stop Script Recording**.
  HFSS 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**.
   HFSS 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 HFSS Online Help in the Running HFSS from a Command Line section), use:

-runscriptandexit or **-runscript** arguments to the HFSS command line syntax.

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

If you run the script from DOS prompt as a .vbs file (that is, you don't launch HFSS, 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, you can write this:

```
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**.
  HFSS stops executing the script that has been paused.

Modifying a Script for Easier Playback

In the sample script on page 2-2, 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.

HFSS Scripting Conventions

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: `"SolveInside:=", 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 HFSS 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

Thare are two ways to pass arguments to scripts:

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

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

There is an issue if the user runs the script from DOS prompt as a .vbs file (that is, you don't open hfss, but 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, you can write this:

```
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 HFSS 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 HFSS 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 HFSS 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. HFSS will use the names to determine which data you provided.

For example, when editing an impedance boundary, HFSS 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")
```

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 HFSS 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)
```

Executing a Script from Within a Script

HFSS 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, 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.

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 Chapter 6, *Property Script Commands*, for an explanation of how to manipulate properties in a script.

3 Ansoft Application Object Script Commands

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

`oAnsoftApp.<CommandName> <args>`

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()
```

SetDesiredRamMBLimit

Use: Sets the Desired RAM Limit (MB) value.

Syntax: `SetDesiredRamMBLimit(<DesiredRAMLimit>)`

Return Value: None

Parameters: `<DesiredRAMLimit>`

Type: <int>

Example:

```
oAnsoftApp.SetDesiredRamMBLimit(2000)
```

SetMaximumRamMBLimit

Use: Sets the Maximum RAM Limit (MB) value.

Syntax: `SetMaximumRamMBLimit(<MaximumRAMLimit>)`

Return Value: None

Parameters: `<MaximumRAMLimit>`

Type: <int>

Example:

```
oAnsoftApp.SetMaximumRamMBLimit(2000)
```

SetNumberOfProcessors

Use: Sets the Number of Processors value.

Syntax: `SetNumberOfProcessors(<NumProcessors>)`

Return Value: None

Parameters: `<NumProcessors>`

Type: <int>

Example:

```
oAnsoftApp.SetNumberOfProcessors(2)
```

GetDesiredRamMBLimit

Use: Gets the Desired RAM Limit (MB) value.

Syntax: `GetDesiredRamMBLimit()`

Return Value: Returns the Desired RAM Limit in megabytes (MB).
Type: <int>

Parameters: None

Example:
```
desired_ram = oAnsoftApp.GetDesiredRamMBLimit()
```

GetMaximumRamMBLimit

Use: Gets the Maximum RAM Limit (MB) value.

Syntax:
```
GetMaximumRamMBLimit()
```

Return Value: Returns the Maximum RAM Limit in megabytes (MB).
Type: <int>

Parameters: None

Example:
```
max_ram = oAnsoftApp.GetMaximumRamMBLimit()
```

GetNumberOfProcessors

Use: Gets the Number of Processors value.

Syntax:
```
GetNumberOfProcessors()
```

Return Value: Returns the Number of Processors.
Type: <int>

Parameters: None

Example:
```
numprocessors = oAnsoftApp.GetNumberOfProcessors()
```

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>
```

CloseAllWindows

Use: Closes all MDI child windows on the desktop.

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

Syntax:
```
CloseAllWindows()
```

Return Value: None

Parameters: None

Example:
```
Desktop.CloseAllWindows()
```

CloseProject

Use: Closes a specified project. 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:
```
CloseProject <ProjectName>
```

Return Value: None

Parameters:
```
<ProjectName>
```
Type: <string>

Example:
```
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"
```

Count

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

Syntax:
```
Count
```

Return Value: Returns an integer value.

Parameters:
```
None
```

Example:

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

Example:

```
' iterate through projects and designs using for each
for each prj in oDesktop.GetProjects()
  msgbox prj.GetName()
  for each design in prj.GetDesigns()
    msgbox design.GetName()
  next
next

' iterate through using integer index
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
    msgbox designs(j).GetName()
  next
next
' lookup by name
```

EnableAutoSave

Use: Enable or disable autosave feature.

Syntax: EnableAutoSave(bool)

Return Value: None

Parameters: None

Example: oDesktop.EnableAutoSave(true)

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 ()
```

GetAutoSaveEnabled

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

Command: None

Syntax: `GetAutoSaveEnabled`

Return Value: Boolean

Parameters: `None`

Example:
```
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.

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()
```

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
```

GetName

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()
```

GetLibraryDirectory

Use: Gets the library directory path.

Syntax: GetLibraryDirectory

Return Value: Returns a directory path.

Type: <string>

Parameters: None

Example:

```
libdir = oDesktop.GetLibraryDirectory
```

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()
```

GetProjectDirectory

Use: Gets the project directory path.

Syntax: GetProjectDirectory

Return Value: Returns a directory path.

Type: <string>

Parameters: None

Example:

```
projdir = 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()`

GetTempDirectory

Use: Gets the temp directory path.
Syntax: `GetTempDirectory`
Return Value: Returns a directory path.
Type: <string>
Parameters: None
Example: `tempdir = oDesktop.GetTempDirectory`

GetVersion

Use: `Returns a string representing the version.`
Syntax: `GetVersion()`
Return Value: `string`
Parameters: None
Example: `msgbox(oDesktop.GetVersion()), displays "10.0"`

NewProject

Use: Creates a new project. The new project becomes the active project.
Command: **File>New**
Syntax: `NewProject`
Return Value: The project that is added.
Parameters: None
Example: `Set oProject = oDesktop.NewProject`

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: The opened project.

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

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

Print

Use: Prints the contents of the active view window.

Command: **File>Print**

Syntax: `Print`

Return Value: None

Parameters: None

Example: `oDesktop.Print`

QuitApplication

Use: Exits the desktop.

Command: **File>Exit**
Syntax: `QuitApplication`
Return Value: None
Parameters: None
Example: `oDesktop.QuitApplication`

RestoreWindow

Use: Restores a minimized HFSS window.
Command: None
Syntax: `RestoreWindow`
Return Value: None
Parameters: None
Example: `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")`

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")
```

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"`

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`

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

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"
```

CutDesign

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

Command: **Edit>Cut**

Syntax: `CutDesign <DesignName>`

Return Value: None

Example:

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

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

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.

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: None

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:
```
oProject.InsertDesign "Hfss", "HFSSDesign3",_
  "DrivenModal", ""
```

Paste

Use: Pastes a design in the active project.

Command: **Edit>Paste**

Syntax:
```
Paste
```

Return Value: None

Parameters: None

Example:
```
oProject.Paste
```

Redo

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.hfss", true
```

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

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 should be executed by the `oProject` 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:

```
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")
```

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 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>
```

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
Local Variables:
<PropServer>
"LocalVariables"

<PropTab>
"LocalVariableTab"

AnsoftHfss Modules
<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"

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"))))
```

ChangeProperty

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"))))
```

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")
```

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")
```

GetVariables

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

Syntax: `GetVariables()`

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

Example:

```
Dim var_array
```

Example:

```
project_var_array = oProject.GetVariables()
```

Example:

```
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"
```

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
```

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, 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:Geometry3DAttributeTab", 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

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)))
```

9 Design Object Script Commands

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

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

Conventions Used in this Chapter

`<ModuleName>`

Name used to access one of the following HFSS modules:

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

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")
```

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>)`

Return Value: None

Parameters: `<ObjectName>`

Type: <string>\

Array of object names.

<ThicknessValue>

Type: <real>

Array of DC thickness values (including units) corresponding to each object name.

Example:

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

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.ConstructVariationString(Array("x_size",
"y_size"), Array("2mm", "1mm"))
```

DeleteLinkedDataVariation

Use: Deletes linked solution data, except field data, for the specified variations.

Syntax:

```
DeleteLinkedDataVariation
  Array(<DesignVariationKey>, <DesignVariationKey>, …)
```

Return Value: None

Parameters:

`<DesignVariationKey>`

Type: <string>

Design variation string.

Example:

```
oDesign.DeleteLinkedDataVariation Array("gap_front=" &
Chr(39) & "0.2mm" & Chr(39) & " gap_up_down=" & Chr(39) &
"0.2mm" & Chr(39) & "")
```

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"
```

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"
```

GetModule

Use: Returns the IDispatch for the specified module.

Command: none

Syntax: `GetModule <ModuleName>`
Return Value: Module object.
Example: `Set oModule = oDesign.GetModule "BoundarySetup"`

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()`

GetSolutionType

Use: Returns the solution type for the design.
Command: none
Syntax: `GetSolutionType`
Return Value: `<SolutionType>`
Type: <string>
Possible values are: `"DrivenModal"`, `"DrivenTerminal"`,or `"Eigenmode"`
Example: `oDesign.GetSolutionType`

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

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

Example:

```
oDesign.RenameDesignInstance "HFSSDesign1", "HFSSDesign2"
```

SARSetup

Use: Sets up for the specific absorption rate (SAR) computation.

Command: **HFSS>Fields>SAR Setting**

Syntax:

```
SARSetup <TissueMass>, <MaterialDensity>
```

Return Value: None

Parameters:

```
<TissueMass>
```

Type: <double>

Double between 1 and 10 in grams.

```
<MaterialDensity>
```

Type: <double>

Positive double in gram/cm^3.

Example: `oDesign.SARSetup 1, 1`

SetActiveEditor

Use: Sets the active editor.

Command: None

Syntax: `SetActiveEditor(<EditorName>)`

Return Value: Editor object

Example:

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

SetSolutionType

Use: Sets the solution type for the design.

Command: **HFSS>Solution Type**

Syntax: `SetSolutionType <SolutionType>`

Return Value: None

Parameters: `<SolutionType>`

Type: <string>

Possible values are: `"DrivenModal"`, `"DrivenTerminal"`,or `"Eigenmode"`

Example: `oDesign.SetSolutionType "DrivenTerminal"`

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")
```

Undo

Use: Cancels the last design-level command.

Command: **Edit>Undo**

Syntax: `Undo`

Return Value: None

Example:

```
oDesign.Undo
```

10 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

CreateBondwire

Use: Creates a bondwire primitive.

Command: **Draw>Bondwire**

Syntax: `CreateBondwire <ParametersArray>, <AttributesArray>`

Return Value: None

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: None

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: None

Parameters:

```
<CircleParametersArray>
   Array("NAME:CircleParameters",
      "XCenter:=", <value>,
      "YCenter:=", <value>,
      "ZCenter:=", <value>,
      "Radius:=", <value>,
      "WhichAxis:=", <string>)
```

`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: None

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: None

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: None

Parameters:
```
<CylinderParametersArray>
  Array("NAME:CylinderParameters",
    "XCenter:=", <value>,
    "YCenter:=", <value>,
    "ZCenter:=", <value>,
    "Radius:=", <value>,
    "Height:=", <value>,
    "WhichAxis:=", <string>)

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: None

Parameters:
```
<EllipseParametersArray>
  Array("NAME:EllipseParameters",
    "XCenter:=", <value>,
    "YCenter:=", <value>,
    "ZCenter:=", <value>,
    "MajRadius:=", <value>,
    "Ratio:=", <value>,
    "WhichAxis:=", <string>)

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.

CreateHelix

Use: Creates a helix by sweeping the specified 2D objects.

Command: **Draw>Helix**

Syntax:
```
CreateHelix <SelectionsArray>, <HelixParametersArray>
```

Return Value: None

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: None

Parameters:

```
<PointParametersArray>
  Array("NAME:PointParameters",
    "PointX:=", <value>,
    "PointY:=", <value>,
    "PointZ:=", <value>)
```

CreatePolyline

Use: Creates a polyline primitive.

Command: **Draw>Polyline**

Syntax:

```
CreatePolyline <PolylineParametersArray>,
  <AttributesArray>
```

Return Value: None

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"
```

CreateRectangle

Use: Creates a rectangle primitive.

Command: **Draw>Rectangle**

Syntax:
```
CreateRectangle <RectangleParametersArray>,
  <AttributesArray>
```

Return Value: None

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.

CreateRegion

Use: Defines a region containing the design.

Command: **Draw>Create Region**

Syntax:
```
CreateRegion <RegionParameters> <RegionAttributes>
```

Return Value: None

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>)
```

Example:

```
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.CreateRegion Array("NAME:RegionParameters", _
"CoordinateSystemID:=", -1, _
"+XPadding:=", "0", "-XPadding:=", "0", _
"+YPadding:=", "0", "-YPadding:=", "0", _
"+ZPadding:=", "0", "-ZPadding:=", "0"), _
Array("NAME:Attributes", "Name:=", "Region", _
"Flags:=", "Wireframe#", _
"Color:=", "(255 0 0)", _
"Transparency:=", 0.400000005960464, _
"PartCoordinateSystem:=", "Global", _
"MaterialName:=", "vacuum", _
"SolveInside:=", true)
```

CreateRegularPolyhedron

Use: Creates a regular polyhedron primitive.

Command: **Draw>Regular Polyhedron**

Syntax:
```
CreateRegularPolyhedron <PolyhedronParametersArray>,
  <AttributesArray>
```

Return Value: None

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: None

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.

CreateSphere

Use: Creates a sphere primitive.

Command: **Draw>Sphere**

Syntax: `CreateSphere <SphereParametersArray>, <AttributesArray>`

Return Value: None

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: None

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: None

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"

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: None

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"`

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"
```

Edit Menu Commands

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>
```

Return Value: None

Parameters:

```
<DupLineParametersArray>
  Array("NAME:DuplicateToAlongLineParameters",
    "XComponent:=", <value>,
    "YComponent:=", <value>,
    "ZComponent:=", <value>,
    "NumClones:=", <value>)

NumClones
```

Specify a number greater than 1.

DuplicateAroundAxis

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", "Z"
Example: "WhichAxis:=", "Z"

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>)
```

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>)
```

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

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"

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>)
```

3D Modeler Menu Commands

AssignMaterial

Use: Assigns a material to the specified objects. Only the `MaterialName` and `SolveInside` parameters of `<AttributesArray>` are supported.

Command: **3D 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: **3D Modeler>Surface>Connect**

Syntax:
```
Connect <SelectionsArray>
```

Return Value: None

CoverLines

Use: Covers the specified 1D objects to form a sheet.

Command: **3D Modeler>Surface>Cover Lines**

Syntax:
```
CoverLines <SelectionsArray>
```

Return Value: None

CoverSurfaces

Use: Covers the specified objects to form a solid object.

Command: **3D 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: **3D Modeler>List>Create>Object List**

3D 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.

CreateFaceCS

Use: Creates a face coordinate system. Only the `Name` attribute of the `<AttributesArray>` parameter is supported.

Command: **3D 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"

CreateObjectFromEdges

Use: Creates a polyline from the specified object edge.

Command: 3D 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: **3D 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: **3D Modeler>Coordinate System>Create>Relative CS->Offset**
3D Modeler>Coordinate System>Create>Relative CS->Rotated
3D 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>)
```

DeleteLastOperation

Use: Deletes the last operation for specified objects.

Command: **3D Modeler>Delete Last Operation**

Syntax:

```
DeleteLastOperation <SelectionsArray>
```

Return Value: None

DetachFaces

Use: Detaches the specified faces.

Command: **3D 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: **3D 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: **3D Modeler->Coordinate System->Edit**

Syntax:

```
EditFaceCS <FaceCSParametersArray>, <AttributesArray>
```

Return Value: None

EditRelativeCS

Use: Modifies a relative coordinate system. Use `<AttributesArray>` to indicate the name of the coordinate system to modify.

Command: **3D 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>)
```

Export

Use: Exports the model to a file.

Command: **3D 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.

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: **3D Modeler>Generate History**

Syntax: `GenerateHistory <SelectionsArray>`

Return Value: None

Import

Use: Imports a 3D model file.

Command: **3D Modeler>Import**

Syntax: `Import <ImportParametersArray>`

Return Value: None

Parameters:

```
<ImportParametersArray>
  Array("NAME:NativeBodyParameters",
    "AutoHeal:=", <bool>,
    "Options:=",  <string>,
    "SourceFile:=", <string>)
```

Intersect

Use: Intersects specified objects.

Command: **3D Modeler>Boolean>Instersect**

Syntax: `Intersect <SelectionsArray>, <IntersectParametersArray>`

Return Value: None

Parameters:

```
<IntersectParametersArray>
  Array("NAME:IntersectParameters",
```

```
    "KeepOriginals:=",  <bool>)
```

MoveFaces

Use: Moves the specified faces along normal or along a vector.

Command: **3D Modeler>Surface>Move Faces>Along Normal**

3D 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)))
```

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:=", <string>)
```

`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: **3D 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"

SeparateBody

Use: Separates bodies of specified multi-lump objects.

Command: **3D Modeler>Boolean>Separate Bodies**

Syntax: `SeparateBody <SelectionsArray>`

Return Value: None

SetModelUnits

Use: Sets the model units.

Command: **3D 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: **3D 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: **3D 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"`

Subtract

Use: Subtracts specified objects.

Command: **3D 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)
```

UncoverFaces

Use: Uncovers specified faces.

Command: **3D 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: **3D Modeler>Boolean>Unite**

Syntax:

```
Unite <SelectionsArray>, <UniteParametersArray>
```

Return Value: None

Parameters:

```
<UniteParametersArray>
  Array("NAME:UniteParameters",
       "KeepOriginals:=", <bool>)
```

Other oEditor Commands

Delete

Use: Deletes specified objects, coordinate systems, points, planes, etc.

Command: None

Syntax:

```
Delete <SelectionsArray>
```

Return Value: None

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()
```

GetEdgeByPosition

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>,
    "ZPosition:=", <value>)
```

Example:

```
edgeid =
oEditor.GetEdgeByPosition(Array("NAME:EdgeParameters", _
  "BodyName:=", "Box1", "XPosition:=", "3.4mm", _
  "YPosition:=", "2.8mm", "ZPosition:=", "0.4mm"))
```

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(<PositionParameters>)`

Return Value: Returns an integer face id

Parameters:

```
<PositionParameters>
    Array("NAME:FaceParameters",
        "BodyName:=", <string>,
        "XPosition:=", <value>,
```

"YPosition:=", <value>,
"ZPosition:=", <value>)

Example:

```
Dimfaceid
faceid = oEditor.GetFaceByPosition(Array("NAME:FaceParameters", _
"BodyName:=" "Box1", "XPosition:=", "3.4mm", "YPosition:=, _
"2.8mm", "ZPosition:=", "0.4mm"))
```

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
```

GetObjectName

Use: Gets an object name corresponding to the input face id.

Syntax: `GetObjectName(<FaceID>)`

Return Value: Returns string name of corresponding object.

Parameters: `<FaceID>`

Type: <string>

Example:

```
objectname = oEditor.GetObjectName(Face10)
```

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*")
```

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
```

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>)
```

11 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 outputvariable 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")
```

CreateOutputVariable

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 and other operators in it. The definitions can not 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")
```

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”
```

DoesOutputVariableExist

Use: Determines whether a specified output variable exists.

Syntax: `DoesOutputVariableExist <OutputVarName>`

Return Value: Boolean

Parameters: `<OutputVarName>`

Type: <string>

Name of the output variable.

Example:

```
OutputVarAntennaGainExists = oDesign.DoesOutputVariableExist
(“efield_online”)
```

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")
```

GetOutputVariables

Use: Gets a list of output variables.

Syntax: `GetOutputVariables`

Return Value: An array of output variable names.

Parameters: None

Example:

```
ov = oDesign.GetOutputVariables
```

GetOutputVariableValue

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 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”

<Context>

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).

Example:

Array("Context:=","Infinite Sphere1")

or Array("Context:=","Polyline1")

or Array()

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("AnsoftHfss.HfssScriptInterface")
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", _
 "Freq='1GHz'",_
 "Setup1 : LastAdaptive", _
 "Modal Solution Data", _
 Array())

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

```
' Now, look at the original output variable in a different design
' variation
' ------------------------------------------------------------
val = oModule.GetOutputVariableValue ( "fieldOV", _
 "Distance='0' _
 Freq='1GHz' xsize='0.4mm' ysize='4.1mm'", _
 "Setup1 : LastAdaptive", "Fields", _
 Array("Context:=", "Polyline1", "PointCount:=", 1 ) )
' ------------------------------------------------------------
' Look at the same variable at a position 1mm along the line
' ------------------------------------------------------------
val = oModule.GetOutputVariableValue ( "fieldOV", _
 "Distance='1mm'
 Freq='1GHz'", _
 "Setup1 : LastAdaptive", _
 "Fields", _
 Array("Context:=", "Polyline1", "PointCount:=", 3 ) _
     )
 MsgBox( "2 val " & FormatNumber(val) )
```

12 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")
```

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 supports Reporter scripting. When you execute **Tools>Record Script**, HFSS Operations performed in the Reporter are automatically recorded.

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"
```

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"

<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:=", <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()
```

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"
```

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))")
```

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")
```

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))")
```

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:=", <Variable-Name> | <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,LumpPort1))")), _
        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()
```

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"
```

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,LumpPort1))"), “XY Plot 2:=”, Array(“Mag_E”))
```

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:

```
oDesign.ExportToFile “Plot1”, “c:\report1.dat”
```

GetAllReportNames

Use: Gets the names of existing reports in a design.

Syntax: `GetAllReportNames()`

Return Value: Array of report names.

Parameters: None

Example:

```
Set reportnames = oDesign.GetAllReportNames()
For Each name in reportnames
  Msgbox name
Next
```

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"
```

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:

```
oDesign.ImportIntoReport “Plot1”, “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"
```

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"
```

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:=", <Variable-Name> | <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,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(WavePort1,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()
```

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"))
```

13 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

AutoIdentifyPorts

Use: Automatically assign ports and terminals in a terminal design.

Command: **HFSS>Excitations>Assign>Wave Port|Lumped Port**

Syntax: `AutoIdentifyPorts <FaceIDArray> <IsWavePort>, <ReferenceConductorsArray>`

Return Value: None.

Parameters:

`<FaceIDArray>`

Array("NAME:Faces", <FaceID>, <FaceID>, ...)

`<IsWavePort>`

Type: Boolean

true = waveport, false = lumped port

`<ReferenceConductorsArray>`

Array("NAME:ReferenceConductors", <ConductorName>, <Conductor-Name>, ...)

Example:

```
Set oModule = oDesign.GetModule("BoundarySetup"
oModule.AutoIdentifyPorts Array("NAME:Faces", 52), true, _
Array("NAME:ReferenceConductors", “Conductor1”)
```

AutoIdentifyTerminals

Use: Automatically identify the terminals within the given ports.

Command: **HFSS>Excitations>Assign>Auto Assign Terminals**

Syntax: `AutoIdentifyTerminals <ReferenceConductorsArray>, <PortNames>`

Return Value: None

Parameters:

`<ReferenceConductors>`

Array(“NAME:ReferenceConductors”, <ConductorName>, <Conductor-Name>, …)

`<portNames>`

List of names.

Example:

```
Set oModule = oDesign.GetModule("BoundarySetup"
oModule.AutoIdentifyTerminals Array("NAME:ReferenceConductors",
“Conductor1”), “WavePort1”
```

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`

DeleteAllExcitations

Use: Deletes all excitations.

Command: **HFSS>Excitations>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")
```

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")
```

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()
```

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()`

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))
```

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"
```

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.ReprioritizeBoundaries Array("NAME:NewOrder", _
"Imped1", "PerfE1", "PerfH1")
```

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/excluded as desired and may appear in any order.

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>,
"InfGroundPlane:=", <bool>,
"Objects:=", <AssignmentObjects>,
"Faces:=", <AssignmentFaces>)
```

`UseMaterial`
If `True`, provide `Material` parameter.
If `False`, provide `Conductivity` and `Permeability` parameters.

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))
```

AssignImpedance

Use: Creates an impedance boundary.

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"`.

Example:

```
oModule.AssignLayeredImp Array("NAME:Layered1",_
  "Frequency:=", "10GHz",_
  "Roughness:=", "0um",_
  "IsInternal:=", false,_
  Array("NAME:Layers",_
    Array("NAME:Layer1",_
      "Thickness:=","1um",_
      "Material:=", "tin"),_
    Array("NAME:Layer2",_
      "LayerType:=", "Infinite",_
      "Material:=", "copper")),_
  "Faces:=", Array(12))
```

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>,
      <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")
```

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>)
```

Example:

```
oModule.AssignRadiation Array("NAME:Rad1",_
  "Faces:=", Array(12))
```

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")
```

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

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

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>`
Return Value: None

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

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

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>", "TerminalResistance:=," <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**

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"
```

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

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>,
   "CreateCornerObjs:=", <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), "CreateCornerObjs:=", 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`

14 Mesh Operations Module Script Commands

Mesh setup and operations commands should be executed by the "Mesh-Setup" 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 **3D Modeler>Measure>Area** command. The face ID is given in the **Measure Information** dialog box.

General Commands Recognized by the Mesh Operations Module

DeleteOp

Use: Deletes the specified mesh operations.

Command: **Delete** command in the **List** dialog box. Click **HFSS>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

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")
```

AssignModelResolutionOp

Use: Assigns a model resolution name, value and unit for mesh operations, or specify to UseAutoFeatureIength. If UseAutoFeature length is true, the Defeature length is not used.

Command: **HFFS>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>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>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 can not 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.EditLengthOp "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:ModelResolution1", "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)
```

15 Analysis 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")
```

DeleteDrivenSweep

Use: Deletes a frequency sweep.

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")
```

EditFrequencySweep

Use: Modifies an existing frequency sweep.

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.

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()
```

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>`

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

```
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)
```

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)
```

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 no ports:

```
oModule.InsertSetup "HfssDriven", _
  Array("NAME:Setup1", _
  "Frequency:=", "1GHz", _
  "MaxDeltaE:=", 0.1, _
  "MaximumPasses:=", 3, _
  "MinimumPasses:=", 1, _
  "MinimumConvergedPasses:=", 1, _
  "PercentRefinement:=", 20, _
  "ReducedSolutionBasis:=", false, _
  "DoLambdaRefine:=", true, _
  "DoMaterialLambda:=",  true, _
  "Target:=", 0.3333 _
  "UseConvOutputVariable:=", false, _
  "IsEnabled:=", true, _
```

```
    "ExternalMesh:=", false, _
    "UseMaxTetIncrease:=", false, _
    "MaxTetIncrease:=",  100000)
```

Example: A Driven solution type with ports:

```
oModule.InsertSetup "HfssDriven",
    Array("NAME:Setup1", _
    "Frequency:=", "1GHz", _
    "PortsOnly:=", false, _
    "MaxDeltaS:=", 0.02, _
    "UseMatrixConv:=", false, _
    "MaximumPasses:=", 3, _
    "MinimumPasses:=",  1, _
    "MinimumConvergedPasses:=", 1, _
    "PercentRefinement:=", 20, _
    "ReducedSolutionBasis:=", false, _
    "DoLambdaRefine:=", true, _
    "DoMaterialLambda:=", true, _
    "Target:=", 0.3333, _
    "UseConvOutputVariable:=", false, _
    "DependentOnSetup:=", 0, _
    "IsEnabled:=",  true, _
    "ExternalMesh:=", false, _
    "UseMaxTetIncrease:=", false, _
    "MaxTetIncrease:=",  100000, _
    "PortAccuracy:=", 2, _
    "UseABCOnPort:=", false, _
    "SetPortMinMaxTri:=", false)
```

Example: An Eigenmode solution type:

```
oModule.InsertSetup "HfssEigen", _
    Array("NAME:Setup1", _
    "MinimumFrequency:=", "75GHz", _
    "NumModes:=",  1, _
    "MaxDeltaFreq:=", 10, _
    "ConvergeOnRealFreq:=", false, _
    "MaximumPasses:=", 3, _
```

```
    "MinimumPasses:=", 1, _
    "MinimumConvergedPasses:=", 1, _
    "PercentRefinement:=", 20, _
    "ReducedSolutionBasis:=", false, _
    "DoLambdaRefine:=", true, _
    "DoMaterialLambda:=", true, _
    "Target:=", 0.25 _
    "UseConvOutputVariable:=", false, _
    "IsEnabled:=", true, _
    "ExternalMesh:=", false, _
    "UseMaxTetIncrease:=", true, _
    "MaxTetIncrease:=", 100000)
```

Example: A Driven solution type with ports and matrix convergence:

```
oModule.InsertSetup "HfssDriven", _
  Array("NAME:Setup1", _
  "Frequency:=", "1GHz", _
  "PortsOnly:=", false, _
  "MaxDeltaS:=", 0.02, _
  "UseMatrixConv:=", true, _
  Array("NAME:ConvergenceMatrix", _
    "Entry:=", _
    Array("Port1:=", "WavePort1", "ModeNum1:=", 1, _
    "Port2:=", "WavePort1", "ModeNum2:=", 1, _
    "MagLimit:=", "0.001", "PhaseLimit:=", "1deg"), _
    "Entry:=", _
    Array("Port1:=", "WavePort1", "ModeNum1:=", 1, _
    "Port2:=", "WavePort2", "ModeNum2:=", 1, _
    "MagLimit:=", "1", "PhaseLimit:=", "0.1deg"), _
    "Entry:=", _
    Array("Port1:=", "WavePort2", "ModeNum1:=", 1, _
    "Port2:=", "WavePort1", "ModeNum2:=", 1, _
    MagLimit:=", "1", "PhaseLimit:=", "0.1deg"), _
    "Entry:=", _
    Array("Port1:=", "WavePort2", "ModeNum1:=", 1, _
```

```
      "Port2:=", "WavePort2", "ModeNum2:=", 1, _
      "MagLimit:=", "0.001", "PhaseLimit:=", "1deg")), _
  "MaximumPasses:=", 3, _
  "MinimumPasses:=", 1, _
  "MinimumConvergedPasses:=", 1, _
  "PercentRefinement:=", 20, _
  "ReducedSolutionBasis:=", false, _
  "DoLambdaRefine:=", true, _
  "DoMaterialLambda:=", true, _
  "Target:=", 0.3333, _
  "UseConvOutputVariable:=", false,
  "IsEnabled:=", true,
  "ExternalMesh:=", false,
  "UseMaxTetIncrease:=", false,
  "MaxTetIncrease:=", 100000,
  "PortAccuracy:=", 2,
  "UseABCOnPort:=", false,
  "SetPortMinMaxTri:=", true,
  "PortMinTri:=", 100,
  "PortMaxTri:=", 500)
```

RenameDrivenSweep

Use: Renames an existing frequency sweep.

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.

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

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

SolveSetup

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: `SolveSetup <SetupName>`

Return Value: None

16 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.

<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.

General Commands Recognized by the Optimetrics Module

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"
```

GetSetupNames [Optimetrics]

Use: Gets a list of Optimetrics setup names.

Syntax:
```
GetSetupNames()
```

Return Value: Array of Optimetrics setup names

Parameters: None

Example:
```
For each name in oModule.GetSetupNames()
Msgbox name
Next
```

GetSetupNamesByType [Optimetrics]

Use: Gets a list of Optimetrics setup names by type.

Syntax: `GetSetupNamesByType(<Optimetrics type>)`

Return Value: Array 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
```

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"
```

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

InsertSetup [Parametric]

Use: Inserts a new 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 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", "GoalValeType:=",
  <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]

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

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:=", <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>))
```

`<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 distribution is `Gaussian`.

`<MinCutoff>`

Type: <double>

The minimum cut-off for the variable when distribution is `Gaussian`.

`<MaxCutoff>`

Type: <double>

The maximum cut-off for the variable when distribution 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"))))
```

17 Solutions Module Script Commands

Solutions commands should be executed by the "Solutions" module.

```
Set oModule = oDesign.GetModule("Solutions")
oModule.CommandName <args>
```

DeleteAllReports

Use: Deletes all items in the results folder of the project tree.

Command: **HFSS>Results>Delete All Reports**

Syntax:
```
DeleteAllReports
```

Return Value: None

Parameters: None

Example:
```
oModule.DeleteAllReports
```

DeleteImportData

Use: Deletes imported solution or table data.

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>
```

Return Value: None

Parameters:
```
<FieldType>
```
Type: <string>

Possible values are:

`"NoIncidentWave"`, `"ScatteredFields"`, `"TotalFields"`, or `"IncidentFields"`.

```
<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.

<TerminatedArray>

Array("NAME:Terminated", <IsSource1Terminated>, ...)

This array is empty if it is not 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>, ...)

This array is empty if it is not 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".

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")
```

DeleteSolutionVariation

Use: Deletes matrix solution data for specific solutions and design variations.

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") )
```

DeleteVariation

Use: Deletes matrix, field, and/or mesh solution data for specific variations, across all solutions.

Command: **HFSS>Results>Browse Solutions**

Syntax:

```
DeleteVariation <VariationArray>, <FullVariations>,
  <MeshAndFieldsOnly>, <FieldsOnly>
```

Return Value: None

Parameters:

<VariationArray>

```
Array(<DesignVariationKey>, <DesignVariationKey>,...)
```

<FullVariations>

Type: <bool>

Specifies whether to delete meshes, fields, matrix data, profile, and convergence data.

<MeshAndFieldsOnly>

Type: <bool>

Specifies whether to delete only meshes and fields.

<FieldsOnly>

Type: <bool>

Specifies whether to delete fields only.

Example:

```
oModule.DeleteVariation _
  Array("width=’2in’", "width=’2.5in’"), _
  TRUE, FALSE, FALSE
```

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>
```

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>

Example:
```
oModule.ExportForSpice "width='2in'", _
   Array("Setup1:Sweep1"), 2, 0, _
```

```
"c:\mydir\Sweep1.fws", "", "", "", ""
```

ExportEigenmodes

Use: Exports a tab delimited table of Eigenmodes.

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>,
   <FittingError>, <MinimumOrder>, <MaximumOrder>
```

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>

<FittingError>

Type: <double>

The accuracy to use in fitting the pole zero model, expressed as a fraction.

<MinimumOrder>

Type: <int>

Minimum number of poles in rational function expansion.

<MaximumOrder>

Type: <int>

Maximum number of poles in rational function expansion.

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>
```

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:

1 : HFSS 8.x format (.szg)

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.

Example: Export all frequencies:

```
oModule.ExportNetworkData "width='2in'", _
  Array("Setup1:Sweep1"), 1, "c:\mydir\out.szg", _
  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
```

ExportNMFData

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")
```

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)
```

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>)
```

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"

Example:
```
set oModule = oDesign.GetModule("Solutions")
matrixExist = 0Module.HasMatrixData("Setup1:Adaptive_1",_
"radius = 4in")
```

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, 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.

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), HFSS 8.x format (.szg), 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. 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 :
LastAdaptive", "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")
```

18 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)))
```

DeleteFieldPlot

Use: Deletes one or more plots.

Command: **HFSS>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:

```
Set plotnames = oModule.GetFieldPlotNames()
For Each name in plotnames
     Msgbox name
Next
```

ModifyFieldPlot

Use: Modifies a plot definition.

Command: **HFSS>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.

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**

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))
```

19 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

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**, **+**

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")
```

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")
```

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>
```

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:\Hfss9OutputFiles\GridExport.reg",_
  Array("1", "1", "1"),_
  Array("4", "4", "4"),_
  Array("2", "2", "2")
```

ExportToFile

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.

Parameters:

Example:

```
oModule.LoadNamedExpressions
"C:\Ansoft\PersonalLib\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\PersonalLib\smth.clc", Array("smoothedtemp"),
true
```

20 Radiation Module Script Commands

Radiation field commands should be executed by the "RadField" module.

```
Set oModule = oDesign.GetModule("RadField")
oModule.CommandName <args>
```

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

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

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

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\exportant-params.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")
```

21 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.AddOutputVariable "re_S", "re(S(port1,port1))"
  oDesign.AddOutputVariable "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

J

K

L

Q

R

S

T

U

V