Introduction to Scripting in HFSS

June 2003
The information contained in this document is subject to change without notice. Ansoft makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Ansoft shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

© 2003 Ansoft Corporation. All rights reserved.

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

HFSS and Optimetrics are registered trademarks or trademarks of Ansoft Corporation. Microsoft and Visual Basic are registered trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are the property of their respective owners.

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.

<table>
<thead>
<tr>
<th>Edition</th>
<th>Date</th>
<th>Software Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>June 2003</td>
<td>9.0</td>
</tr>
</tbody>
</table>
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

1. Introduction to VBScript  
   - A Sample HFSS Script .............................. 1-2  
   - VBScript Variables ............................... 1-4  
     - Declaring Variables ........................... 1-4  
     - Array Variables ............................... 1-4  
   - VBScript Operators .............................. 1-5  
     - Operator Precedence ........................... 1-5  
     - Arithmetic Operators ......................... 1-5  
     - Comparison Operators ......................... 1-6  
     - Logical Operators ............................. 1-6  
   - Controlling Program Execution ................... 1-7  
     - Using If...Then...Else ......................... 1-7  
     - Using Select Case .............................. 1-7  
     - Using a For...Next Loop ....................... 1-8  
   - Converting Between Data Types ................... 1-9  
   - Interacting with a Script ....................... 1-9  
   - Recommended VBScript References .............. 1-10
2. HFSS and VBScript
   Overview of HFSS Script Variables .......................... 2-2
   Recording a Script ........................................... 2-6
   Stopping Script Recording ................................... 2-6
   Running a Script ............................................. 2-6
   Pausing and Resuming a Script .............................. 2-6
   Stopping a Script ............................................ 2-7
   Modifying a Script for Easier Playback .................... 2-7
   HFSS Scripting Conventions .................................. 2-7
      Syntax Conventions ......................................... 2-7
      Script Command Conventions .............................. 2-8
      Named Arguments .......................................... 2-8
      Setting Numerical Values ................................ 2-10
   Executing a Script from Within a Script .................. 2-11
   Editing Properties .......................................... 2-11

3. Desktop Object Script Commands
   CloseProject .................................................. 3-2
   GetActiveProject ............................................ 3-2
   GetProjectList ............................................... 3-2
   NewProject .................................................. 3-2
   OpenMultipleProjects .................................... 3-3
   OpenProject ................................................ 3-3
   PauseScript ................................................. 3-3
   Print ......................................................... 3-4
   QuitApplication ............................................. 3-4
   RestoreWindow ............................................... 3-4
   RunProgram .................................................. 3-5
   RunScript ................................................... 3-6
   SetActiveProject .......................................... 3-6
   Sleep ......................................................... 3-7
4. Project Object Script Commands

- Close .................................................. 4-2
- CopyDesign ........................................ 4-2
- CutDesign .......................................... 4-2
- DeleteDesign ....................................... 4-2
- GetActiveDesign ................................. 4-3
- GetDesign ......................................... 4-3
- GetName ........................................... 4-3
- GetPath ............................................. 4-3
- GetTopDesignList ............................... 4-4
- InsertDesign ...................................... 4-4
- Paste ............................................... 4-4
- Redo ............................................... 4-5
- Save ............................................... 4-5
- SaveAs ............................................. 4-5
-SetActiveDesign ................................. 4-6
- SimulateAll ....................................... 4-6
- Undo ............................................... 4-6

5. Material Script Commands

- AddMaterial ....................................... 5-2
- EditMaterial ....................................... 5-3
- ExportMaterial ................................... 5-3
- RemoveMaterial .................................. 5-4

6. Property Script Commands

- ChangeProperty .................................... 6-4

Additional Property Scripting Commands .................. 6-9

- GetProperties ..................................... 6-9
- GetPropertyValue .................................. 6-9
- GetVariables ...................................... 6-9
- GetVariableValue .................................. 6-10
- SetPropertyValue .................................. 6-10
- SetVariableValue .................................. 6-11
7. Dataset Script Commands

- AddDataset ........................................... 7-2
- DeleteDataset ........................................ 7-2
- EditDataset ............................................ 7-3

8. Design Object Script Commands

- AbortSolveAsynch .................................. 8-2
- ApplyMeshOps ....................................... 8-2
- GetModule .......................................... 8-2
- GetName ............................................. 8-3
- GetSolveAsynchStatus .............................. 8-3
- Redo ................................................... 8-3
- RenameDesignInstance .............................. 8-4
- SARSetup .......................................... 8-4
- SetActiveEditor .................................... 8-4
- SetSolutionType .................................... 8-5
- Solve ............................................... 8-5
- SolveAsynch ........................................ 8-6
- Undo .................................................... 8-6

9. Output Variable Script Commands

- AddOutputVariable .................................. 9-2
- DeleteOutputVariable ............................... 9-2
- EditOutputVariable ................................ 9-3
- GetOutputVariableValue ........................... 9-3

10. 3D Modeler Editor Script Commands

- Draw Menu Commands .................................. 10-3
  - CreateBondwire .................................. 10-3
  - CreateBox ......................................... 10-4
  - CreateCircle ...................................... 10-4
  - CreateCone ........................................ 10-5
  - CreateCutplane .................................... 10-5
Introduction to Scripting in HFSS

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateCylinder</td>
<td>10-6</td>
</tr>
<tr>
<td>CreateEllipse</td>
<td>10-6</td>
</tr>
<tr>
<td>CreateHelix</td>
<td>10-7</td>
</tr>
<tr>
<td>CreatePoint</td>
<td>10-8</td>
</tr>
<tr>
<td>CreatePolyline</td>
<td>10-8</td>
</tr>
<tr>
<td>CreateRectangle</td>
<td>10-9</td>
</tr>
<tr>
<td>CreateRegularPolyhedron</td>
<td>10-10</td>
</tr>
<tr>
<td>CreateRegularPolygon</td>
<td>10-10</td>
</tr>
<tr>
<td>CreateSphere</td>
<td>10-11</td>
</tr>
<tr>
<td>CreateSpiral</td>
<td>10-12</td>
</tr>
<tr>
<td>CreateTorus</td>
<td>10-12</td>
</tr>
<tr>
<td>EditPolyline</td>
<td>10-13</td>
</tr>
<tr>
<td>SweepAlongPath</td>
<td>10-14</td>
</tr>
<tr>
<td>SweepAlongVector</td>
<td>10-14</td>
</tr>
<tr>
<td>SweepAroundAxis</td>
<td>10-15</td>
</tr>
<tr>
<td>Edit Menu Commands</td>
<td>10-16</td>
</tr>
<tr>
<td>Copy</td>
<td>10-16</td>
</tr>
<tr>
<td>DuplicateAlongLine</td>
<td>10-16</td>
</tr>
<tr>
<td>DuplicateAroundAxis</td>
<td>10-16</td>
</tr>
<tr>
<td>DuplicateMirror</td>
<td>10-17</td>
</tr>
<tr>
<td>Mirror</td>
<td>10-17</td>
</tr>
<tr>
<td>Move</td>
<td>10-18</td>
</tr>
<tr>
<td>OffsetFaces</td>
<td>10-18</td>
</tr>
<tr>
<td>Paste</td>
<td>10-18</td>
</tr>
<tr>
<td>Rotate</td>
<td>10-19</td>
</tr>
<tr>
<td>Scale</td>
<td>10-19</td>
</tr>
<tr>
<td>3D Modeler Menu Commands</td>
<td>10-20</td>
</tr>
<tr>
<td>AssignMaterial</td>
<td>10-20</td>
</tr>
<tr>
<td>Connect</td>
<td>10-20</td>
</tr>
<tr>
<td>CoverLines</td>
<td>10-20</td>
</tr>
<tr>
<td>CoverSurfaces</td>
<td>10-20</td>
</tr>
<tr>
<td>CreateEntityList</td>
<td>10-20</td>
</tr>
<tr>
<td>CreateFaceCS</td>
<td>10-21</td>
</tr>
</tbody>
</table>
## Introduction to Scripting in HFSS

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateObjectFromFaces</td>
<td>10-23</td>
</tr>
<tr>
<td>CreateRelativeCS</td>
<td>10-24</td>
</tr>
<tr>
<td>DeleteLastOperation</td>
<td>10-24</td>
</tr>
<tr>
<td>DetachFaces</td>
<td>10-25</td>
</tr>
<tr>
<td>EditEntityList</td>
<td>10-25</td>
</tr>
<tr>
<td>EditFaceCS</td>
<td>10-26</td>
</tr>
<tr>
<td>EditRelativeCS</td>
<td>10-26</td>
</tr>
<tr>
<td>Export</td>
<td>10-26</td>
</tr>
<tr>
<td>GenerateHistory</td>
<td>10-27</td>
</tr>
<tr>
<td>Import</td>
<td>10-27</td>
</tr>
<tr>
<td>Intersect</td>
<td>10-28</td>
</tr>
<tr>
<td>MoveFaces</td>
<td>10-28</td>
</tr>
<tr>
<td>Section</td>
<td>10-29</td>
</tr>
<tr>
<td>SeparateBody</td>
<td>10-29</td>
</tr>
<tr>
<td>SetModelUnits</td>
<td>10-30</td>
</tr>
<tr>
<td>SetWCS</td>
<td>10-30</td>
</tr>
<tr>
<td>Split</td>
<td>10-31</td>
</tr>
<tr>
<td>Subtract</td>
<td>10-31</td>
</tr>
<tr>
<td>UncoverFaces</td>
<td>10-32</td>
</tr>
<tr>
<td>Unite</td>
<td>10-33</td>
</tr>
<tr>
<td>Other oEditor Commands</td>
<td>10-34</td>
</tr>
<tr>
<td>Delete</td>
<td>10-34</td>
</tr>
<tr>
<td>GetFaceByPosition</td>
<td>10-34</td>
</tr>
<tr>
<td>PageSetup</td>
<td>10-35</td>
</tr>
<tr>
<td>RenamePart</td>
<td>10-35</td>
</tr>
</tbody>
</table>

### 11. Reporter Editor Script Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateReport</td>
<td>11-2</td>
</tr>
<tr>
<td>RemoveReport</td>
<td>11-8</td>
</tr>
</tbody>
</table>
12. Boundary and Excitation Module Script Commands

General Commands Recognized by the Boundary/Excitations Module

- ChangeImpedanceMult .......................... 12-2
- DeleteAllBoundaries ......................... 12-2
- DeleteAllExcitations ......................... 12-2
- DeleteBoundaries ............................. 12-3
- ReassignBoundary ............................. 12-3
- RenameBoundary .............................. 12-3
- ReprioritizeBoundaries ...................... 12-4

Script Commands for Creating and Modifying Boundaries 12-5

- AssignCurrent ................................ 12-5
- AssignFiniteCond ............................. 12-6
- AssignImpedance .............................. 12-7
- AssignIncidentWave ......................... 12-7
- AssignLayeredImp ............................ 12-8
- AssignLumpedPort ............................ 12-10
- AssignLumpedRLC ............................. 12-11
- AssignMagneticBias ........................... 12-12
- AssignMaster ................................ 12-13
- AssignPerfectE ............................... 12-13
- AssignPerfectH ............................... 12-14
- AssignRadiation .............................. 12-14
- AssignSlave ................................ 12-14
- AssignSymmetry .............................. 12-15
- AssignVoltage ............................... 12-16
- AssignWavePort .............................. 12-17
- EditCurrent ................................ 12-21
- EditFiniteCond ............................... 12-21
- EditImpedance ............................... 12-21
- EditIncidentWave ............................ 12-21
- EditLayeredImpedance ....................... 12-21
Introduction to Scripting in HFSS

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EditMaster</td>
<td>12-22</td>
</tr>
<tr>
<td>EditPerfectE</td>
<td>12-22</td>
</tr>
<tr>
<td>EditPerfectH</td>
<td>12-22</td>
</tr>
<tr>
<td>EditLumpedPort</td>
<td>12-22</td>
</tr>
<tr>
<td>EditLumpedRLC</td>
<td>12-22</td>
</tr>
<tr>
<td>EditMagneticBias</td>
<td>12-23</td>
</tr>
<tr>
<td>EditRadiation</td>
<td>12-23</td>
</tr>
<tr>
<td>EditSlave</td>
<td>12-23</td>
</tr>
<tr>
<td>EditSymmetry</td>
<td>12-23</td>
</tr>
<tr>
<td>EditVoltage</td>
<td>12-23</td>
</tr>
<tr>
<td>EditWavePort</td>
<td>12-23</td>
</tr>
<tr>
<td><strong>Script Commands for Creating and Modifying PMLs</strong></td>
<td>12-24</td>
</tr>
<tr>
<td>CreatePML</td>
<td>12-24</td>
</tr>
<tr>
<td>ModifyPMLGroup</td>
<td>12-26</td>
</tr>
<tr>
<td>PMLGroupCreated</td>
<td>12-26</td>
</tr>
<tr>
<td>PMLGroupModified</td>
<td>12-27</td>
</tr>
<tr>
<td>RecalculatePMLMaterials</td>
<td>12-27</td>
</tr>
</tbody>
</table>

### 13. Mesh Operations Module Script Commands

#### General Commands Recognized by the Mesh Operations Module

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeleteOp</td>
<td>13-2</td>
</tr>
<tr>
<td>RenameOp</td>
<td>13-2</td>
</tr>
</tbody>
</table>

#### Script Commands for Creating and Modifying Mesh Operations

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignLengthOp</td>
<td>13-3</td>
</tr>
<tr>
<td>AssignSkinDepthOp</td>
<td>13-4</td>
</tr>
<tr>
<td>AssignTrueSurfOp</td>
<td>13-4</td>
</tr>
<tr>
<td>EditLengthOp</td>
<td>13-5</td>
</tr>
<tr>
<td>EditSkinDepthOp</td>
<td>13-6</td>
</tr>
<tr>
<td>EditTrueSurfOp</td>
<td>13-6</td>
</tr>
</tbody>
</table>
14. Analysis Module Script Commands

- DeleteDrivenSweep .............................................. 14-2
- DeleteSetups ......................................................... 14-2
- EditDrivenSweep .................................................. 14-2
- EditSetup ............................................................. 14-3
- InsertDrivenSweep ................................................ 14-3
- InsertSetup .......................................................... 14-5
- RenameDrivenSweep .............................................. 14-7
- RenameSetup ......................................................... 14-8
- RevertAllToInitial ................................................ 14-8
- RevertSetupToInitial ............................................ 14-8
- SolveSetup ............................................................ 14-8

15. Optimetrics Module Script Commands

- General Commands Recognized by the Optimetrics Module ........................................ 15-5
  - DeleteSetups ........................................................ 15-5
  - RenameSetup ....................................................... 15-5
  - SolveSetup .......................................................... 15-5

- Parametric Script Commands ........................................ 15-6
  - EditSetup ............................................................ 15-6
  - InsertSetup .......................................................... 15-6

- Optimization Script Commands ........................................ 15-9
  - EditSetup ............................................................ 15-9
  - InsertSetup .......................................................... 15-9

- Sensitivity Script Commands ........................................ 15-13
  - EditSetup ............................................................ 15-13
  - InsertSetup .......................................................... 15-13

- Statistical Script Commands ......................................... 15-15
  - EditSetup ............................................................ 15-15
  - InsertSetup .......................................................... 15-15
16. Solutions Module Script Commands

- DeleteImportData ........................................ 16-2
- EditSources ............................................. 16-2
- DeleteSolutionVariation .............................. 16-4
- DeleteVariation ......................................... 16-5
- ExportForSpice .......................................... 16-6
- ExportForHSpice ........................................ 16-7
- ExportNetworkData ...................................... 16-9
- ExportNMFData .......................................... 16-10
- ImportSolution ......................................... 16-11
- ImportTable ............................................. 16-12

17. Field Overlays Module Script Commands

- CreateFieldPlot ........................................ 17-2
- DeleteFieldPlot ......................................... 17-6
- ModifyFieldPlot ........................................ 17-7
- RenameFieldPlot ....................................... 17-8
- RenamePlotFolder ...................................... 17-8
- SetFieldPlotSettings ................................. 17-9
- SetPlotFolderSettings ............................... 17-10

18. Fields Calculator Script Commands

- AddNamedExpr .......................................... 18-2
- CalcOp .................................................. 18-2
- CalcStack .............................................. 18-2
- ChangeGeomSettings ................................. 18-3
- ClcEval ................................................ 18-3
- ClcMaterial ........................................... 18-3
- ClearAllNamedExpr .................................... 18-4
- CopyNamedExprToStack .............................. 18-4
- DeleteNamedExpr ...................................... 18-4
- EnterComplex .......................................... 18-5
- EnterComplexVector .................................. 18-5
- EnterLine .............................................. 18-6
19. Radiation Module Script Commands

General Commands Recognized by the Radiation Module 19-2
- DeleteFarFieldSetup .......................... 19-2
- DeleteNearFieldSetup .......................... 19-2
- RenameSetup ................................. 19-3

Script Commands for Creating and Modifying Radiation Setups ................................. 19-4
- EditFarFieldSphereSetup ...................... 19-4
- EditNearFieldLineSetup ...................... 19-4
- EditNearFieldSphereSetup ................... 19-5
- InsertFarFieldSphereSetup ................... 19-6
- InsertNearFieldLineSetup .................... 19-7
- InsertNearFieldSphereSetup ................ 19-8

Script Commands for Modifying Antenna Array Setups .............................................. 19-9
- EditAntennaArraySetup ....................... 19-9

Script Commands for Exporting Antenna Parameters and Max Field Parameters ............... 19-13
- ExportRadiationParametersToFile ........... 19-13

20. Example Scripts

Variable Helix Script ............................ 20-2
HFSS Data Export Script .......................... 20-6
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 9.0
'11:03 AM May 30, 2003
------------------------------

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

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

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

Set oProject = oDesktop.GetActiveProject
oProject.InsertDesign "Hfss", "HFSSModel1", "DrivenModal", ""
```

1-2 Introduction to VBScript
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`.

```vbs
Set oDesign = oProject.SetActiveDesign("HFSSModel1")
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:="", _
   0.4, "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:

```vbs
Msgbox("Please include units when creating variables " & _
   "that require dimensions."
"
Following is an example of improper use of the underscore character:
```

```vbs
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:

```vbnet
Dim box_xsize
```

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

```vbnet
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:

```vbnet
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:

```vbnet
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:

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

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

```vbnet
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 evaluated 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.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>−</td>
<td>Unary negation</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>\</td>
<td>Integer division</td>
</tr>
<tr>
<td>Mod</td>
<td>Modulus arithmetic</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>−</td>
<td>Subtraction</td>
</tr>
<tr>
<td>&amp;</td>
<td>String concatenation</td>
</tr>
</tbody>
</table>
Comparison Operators

Following is a list of VBScript’s comparison operators:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>=</code></td>
<td>Equality</td>
</tr>
<tr>
<td><code>&lt;&gt;</code></td>
<td>Inequality</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>Less than</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less than or equal to</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td><code>Is</code></td>
<td>Object equivalence</td>
</tr>
</tbody>
</table>

Logical Operators

Following is a list of VBScript’s logical operators:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not</td>
<td>Logical negation</td>
</tr>
<tr>
<td>And</td>
<td>Logical conjunction</td>
</tr>
<tr>
<td>Or</td>
<td>Logical disjunction</td>
</tr>
<tr>
<td>Xor</td>
<td>Logical exclusion</td>
</tr>
<tr>
<td>Eqv</td>
<td>Logical equivalence</td>
</tr>
<tr>
<td>Imp</td>
<td>Logical implication</td>
</tr>
</tbody>
</table>
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:

```vbnet
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:

```vbnet
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.

- **CInt**
  - Syntax: `CInt(variablename)`.
  - Converts `variablename` to an integer.

- **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.

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:

```vb
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:

```vb
MsgBox ("message text")
```
Recommended VBScript References


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 oHfssApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oHfssApp = CreateObject(“AnsoftHfss.HfssScriptInterface”)
Set oDesktop = oHfssApp.GetAppDesktop()
Set oProject = oDesktop.SetActiveProject(“Project1”)
Set oDesign = oProject.SetActiveDesign(“HFSSModel1”)
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.
```

2-2 HFSS and VBScript
**oHfssApp**

The *oHfssApp* object provides a handle for VBScript to access the *AnsoftHfss* product. One example of accessing this object is:

```vbscript
Set oHfssApp = CreateObject("AnsoftHfss.HfssScriptInterface")
```

**oDesktop**

The *oDesktop* object is used to perform desktop-level operations, including project management. One example of accessing this object is:

```vbscript
Set oDesktop = oHfssApp.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:

```vbscript
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:

```vbscript
Set oDesign = oProject.GetActiveDesign()
```
Introduction to Scripting in HFSS

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 Ansoft Hfss product scripting supports the following editors:

<table>
<thead>
<tr>
<th>Editor</th>
<th>Name in Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Modeler Editor</td>
<td>“3D Modeler”</td>
</tr>
<tr>
<td>Reporter Editor</td>
<td>There is no Reporter editor object in the script. Instead, Reporter editor commands are executed by the HFSS design object oDesign.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Module</th>
<th>Name in Script</th>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boundary/Excitations Module</td>
<td>“BoundarySetup”</td>
<td>Chapter 12, <em>Boundary and Excitation Module Script Commands</em></td>
</tr>
<tr>
<td>Corresponds to the <strong>Boundaries</strong> and <strong>Excitations</strong> branches in the project tree.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mesh Operations Module</td>
<td>“MeshSetup”</td>
<td>Chapter 13, <em>Mesh Operations Module Script Commands</em></td>
</tr>
<tr>
<td>Corresponds to the <strong>Mesh Operations</strong> branch in the project tree.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis Module</td>
<td>“AnalysisSetup”</td>
<td>Chapter 14, <em>Analysis Module Script Commands</em></td>
</tr>
<tr>
<td>Corresponds to the <strong>Analysis</strong> branch in the project tree.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optimetrics Module</td>
<td>“Optimetrics”</td>
<td>Chapter 15, <em>Optimetrics Script Commands</em></td>
</tr>
<tr>
<td>Corresponds to the <strong>Optimetrics</strong> branch in the project tree.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solutions Module</td>
<td>“Solutions”</td>
<td>Chapter 16, <em>Solutions Module Script Commands</em></td>
</tr>
<tr>
<td>Corresponds to the operations in the <strong>Solution Data</strong> dialog box, which is accessed by clicking <strong>HFSS&gt;Results&gt;Solution Data</strong>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field Overlays Module</td>
<td>“FieldsReporter”</td>
<td>Chapter 17, <em>Field Overlays Module Script Commands</em></td>
</tr>
<tr>
<td>Corresponds to the <strong>Field Overlays</strong> branch in the project tree.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radiation Module</td>
<td>“RadField”</td>
<td>Chapter 18, <em>Radiation Module Script Commands</em></td>
</tr>
<tr>
<td>Corresponds to the <strong>Radiation</strong> branch in the project tree.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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. On the Tools menu, click 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

• On the Tools menu, click Stop Script Recording. HFSS stops recording to the script.

Running a Script

1. On the Tools menu, click 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.

Pausing and Resuming a Script

To pause a script during its execution:
• On the Tools menu, click Pause Script.

To resume a script after pausing it:
• On the Tools menu, click 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.
Example: "SolveInside:="

<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.

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:

```vbscript
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:

```vbscript
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:

```vb
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.
Desktop commands should be executed by the oDesktop object.
Set oDesktop =
    CreateObject("AnsoftHfss.HfssScriptInterface")
oDesktop.CommandName <args>
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"`

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 ()`

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()`

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:**

```python
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")
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
Introduction to Scripting in HFSS
Project commands should be executed by the `oProject` object. One example of accessing this object is:

```vbscript
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 "HFSSModel1"
```

### 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 "HFSSModel1"
```

### DeleteDesign

**Use:** Deletes a specified design in the project.

**Command:** `Edit>Delete`

**Syntax:** `DeleteDesign <DesignName>`

**Return Value:** None

**Example:**
```
oProject.DeleteDesign "HfssModel2"
```
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 ("HfssModel1")

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", "HfssModel3", "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 ("HfssModel2")`

**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`
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:

```python
oProject.RemoveMaterial "Material1", false, "mo0907", "UserLib"
```

Example:

```python
oProject.RemoveMaterial "Material1", true, "Local", "Project"
```
Property commands should be executed by the `oProject` object.

```vba
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>

6-2 Property Script Commands
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>

"Geometry3DAttributeTab"

"Geometry3DCmdTab"
**ChangeProperty**

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

**Command:** None

**Syntax:**

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

**Return Value:** None

**Parameters:**

- `<PropTabArray>`

  ```plaintext
  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>"),

  ``
Introduction to Scripting in HFSS

"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",

6-6 Property Script Commands
"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

Property Script Commands 6-7
Introduction to Scripting in HFSS

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”:

```plaintext
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”:

```plaintext
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.)

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

6-8 Property Script Commands
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>)
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.
Command: None
Syntax: GetPropertyValue(<PropTab>, <PropServer>, <PropName>)
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 \texttt{oProject}. To get the value of local variables, use \texttt{oDesign}.

Command: None

Syntax: \texttt{GetVariableValue( <VarName> )}

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

Parameters: \texttt{<VarName>}

Type: string

Name of the variable to access.

Example:

\begin{verbatim}
project_var_value_string = oProject.GetVariableValue("var_name")
\end{verbatim}

Example:

\begin{verbatim}
local_var_value_string = oDesign.GetVariableValue("var_name")
\end{verbatim}

SetPropertyChanged

Use: Sets the value of one property. This is not supported for properties of the following types: ButtonProp, PointProp, V3DPointProp, and VPointProp. Only the \texttt{ChangeProperty} command can be used to modify these properties. This can be executed by the \texttt{oProject}, \texttt{oDesign}, or \texttt{oEditor} variables.

Command: None

Syntax: \texttt{SetPropertyValue <PropTab>, <PropServer>, <PropName>, <PropValue>}

Return Value: None

Parameters: \texttt{<PropValue>}

Type: String

Contains the value to set the property. The formatting is different depending on what type of property is being edited. Use \texttt{GetPropertyValue} for the desired property to see the expected format.

Example:

\begin{verbatim}
oEditor.SetPropertyValue "Geometry3DcmdTab","Box1:CreateBox:1","XSize","3mm"
\end{verbatim}
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"
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 properties and for each one, shows the user the current value and asks if the value should be changed.

Example:

```vbscript
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 + ":" + 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 + ":" + val)
    End If
Next
```
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)))
```
7-4 Dataset 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”
```
AbortSolveAsynch

*Use:* aborts a non-blocking simulation that was started by a previous call to the `SolveAsynch` command.

*Command:* none

*Syntax:* `AbortSolveAsynch <SetupName>`

*Return Value:* None

*Example:* `oDesign.AbortSolveAsynch “Setup1”`

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”)`

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

**GetSolveAsynchStatus**

*Use:* Returns the status of an asynchronous simulation that was started by a previous call to the SolveAsynch command.

*Command:* none

*Syntax:* GetSolveAsynchStatus <SetupName>

*Return Value:* Type: <int>

  -1: completed with error
  1: still running
  0: completed with no error

*Example:* return_status = oDesign.GetSolveAsynchStatus "Setup1"

**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 "HFSSModel1", "HFSSModel2"
```

### 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`: command execution error
- `1`: simulation error
- `0`: normal completion

*Parameters:* `<SetupNameArray>`: Array(<SetupName>, <SetupName>, ...)

- `<SetupName>`:
  - Type: `<string>`
  - Name of the solution setup to solve.

*Example:* return_status = oDesign.Solve Array("Setup1", "Setup2")
### SolveAsynch

**Use:** Performs a non-blocking simulation. The next script command will be executed while the simulation is being performed. If you want multiple solution setups to simulate in parallel, execute a separate `SolveAsynch` command for each setup.

**Command:** none

**Syntax:** `SolveAsynch <SetupName>`

**Return Value:**
- Type: `<int>`
  - -1: command execution error
  - 1: simulation error
  - 0: normal completion

**Example:**
```
return_status = oDesign.SolveAsynch "Setup1"
```

### Undo

**Use:** Cancels the last design-level command.

**Command:** Edit>Undo

**Syntax:** `Undo`

**Return Value:** None

**Example:**
```
oDesign.Undo
```
Output variable commands should be executed by the oDesign object.

Set oDesign = Project.SetActiveDesign("HFSSModel1")
oDesign.CommandName <args>
AddOutputVariable

**Use:** Adds 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, Sim values, or other output variable names. It cannot have spaces or any arithmetic or other operators. The definitions can not be cyclic. For example, \( A = 2*B, B=3*A \) is not allowed.

**Syntax:**

```
AddOutputVariable <VarName>, <Expression>
```

**Return Value:** None

**Parameters:**

- `<VarName>`
  - **Type:** `string`
  - Name of the output variable.

- `<Expression>`
  - **Type:** `value`
  - Value to assign to the variable.

DeleteOutputVariable

**Use:** Deletes an existing output variable. The variable can only be deleted if it is not in use by any traces.

**Syntax:**

```
DeleteOutputVariable <VarName>
```

**Return Value:** None
**EditOutputVariable**

*Use:* Changes the name or expression of an existing output variable.

*Syntax:* 
```
EditOutputVariable <OrigVarName> <NewVarName> <NewExpression>
```

Provide empty quotes "" as the `NewVarName` or `NewExpression` if it should not be changed.

*Return Value:* None

*Parameters:* 
- `<OrigVarName>`
  - Type: `<string>`
  - Original name of the variable.

- `<NewVarName>`
  - Type: `<string>`
  - New name of the variable if any, or else pass an empty string.

- `<NewExpression>`
  - Type: `<value>`
  - New value to assign to the variable.

**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 <VarName>, <SolutionName>, <IntrinsicVariation>, <SimValueContext>
```

*Return Value:* Double value of the output variable.

*Parameters:* 
- `<SolutionName>`
  - Type: `<string>`
  - Name of the solution as listed in the Traces dialog box.
  - For example: “Setup1 : Last Adaptive”

- `<IntrinsicVariation>`
  - Type: `<string>`
  - A set of intrinsic variables, value pairs to use when evaluating the output expression.
  - Example: "Freq='20GHz' Theta='20deg' Phi='30deg'"
<SimValueContext>
  Type: <string>
  Context for which the output variable expression is being evaluated.
  This can be an empty string if there is no context (for example, for S-
  parameters).
  Example: "Infinite Sphere1" or "Line1" or ""

Example:
Dim Val
Val=oDesign.GetOutputVariableValue "OutVarTest",_
  "Setup1 : Sweep1",_
  "Freq='20GHz' Theta='20deg' Phi='30deg'",_
  "Infinite Sphere1"
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:

CreateCircle

Use: Creates a circle primitive.
Command: Draw>Circle
Syntax: CreateCircle <CircleParametersArray>, <AttributesArray>
Return Value: None
Parameters:

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:**

```
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:**

```
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”
“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.
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>,
    "WhichAxis:=" , <string>)
"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>,
  "XStartDir:="", <value>,
  "YStartDir:="", <value>,
  "ZStartDir:="", <value>,
  "NumThread:="", <value>,
  "RightHand:="", <bool>,
  "RadiusIncrement:="", <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”
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")
```
"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

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>)
Introduction to Scripting in HFSS

**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 data.

*Command:* Edit > Paste

*Syntax:* Paste

*Return Value:* None
**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>)
```
Introduction to Scripting in HFSS

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:="),
  Array("NAME:Attributes", _
    "MaterialName:="),
    "tungsten", _
    "SolveInside:="), false)`

**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>,

  3D Modeler Editor Script Commands 10-21
Introduction to Scripting in HFSS

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.

10-22 3D Modeler Editor Script Commands
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"

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)))
Introduction to Scripting in HFSS

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:**
```plaintext
DetachFaces <SelectionsArray>,
   <DetachFacesParametersArray>
```

**Return Value:** None

**Parameters:**
- `<DetachFacesParametersArray>`
  ```plaintext
  Array("NAME:Parameters",
   <FacesOfOneObjToDetach>,
   <FacesOfOneObjToDetach>, ...)
  ```

- `<FacesOfOneObjToDetach>`
  ```plaintext
  Array("Name:DetachFacesToParameters",
   "FacesToDetach:="; <array>)
  ```

**Example:**
```plaintext
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:**
```plaintext
EditEntityList <SelectionsArray>,
   <EntityListParametersArray>
```

**Return Value:** None
Introduction to Scripting in HFSS

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>,
        
10-26 3D Modeler Editor Script Commands
Introduction to Scripting in HFSS

"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.

**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:**

Array("NAME:NativeBodyParameters",
    "AutoHeal:=", <bool>,
    "Options:=", <string>,
    "SourceFile:=", <string>)
Introduction to Scripting in HFSS

**Intersect**

*Use:* Intersects specified objects.

*Command:* 3D Modeler>Boolean>Intersect

*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:

```plaintext
goEditor.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)))
```

---

**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>Separeate Bodies

**Syntax:**
SeparateBody <SelectionsArray>

**Return Value:**
None
Introduction to Scripting in HFSS

**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.
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.
Introduction to Scripting in HFSS

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

GetFaceByPosition

*Use:* Gets the ID of a face by position.

*Command:* None

*Syntax:* `GetFaceByPosition <FaceByPositionParametersArray>`

*Return Value:* An integer containing the face ID.

*Parameters:* `<FaceByPositionParametersArray>`

```
Array("NAME:Parameters",
    "BodyName:=" , <string>,
    "XPosition:=" , <value>,
    "YPosition:=" , <value>,
    "ZPosition:=" , <value>)
```

*Example:

```vbscript
Dim oFaceID
Set oFaceID = oEditor.GetFaceByPosition_ 
Array("NAME:Parameters",_ 
    "BodyName:=" , "Box1",_ 
    "XPosition:=" , "0mm",_ 
    "YPosition:=" , "0mm",_ 
    "ZPosition:=" , "0mm")
```
**PageSetup**

*Use:* Specifies the page settings for printing.

*Command:* File>Page Setup

*Syntax:* PageSetup <PageSetupParametersArray>

*Return Value:* None

*Parameters:* 
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:* 
Array("NAME:Rename Data",
    "Old Name:=" , <string>,
    "New Name:=" , <string>)
Introduction to Scripting in HFSS

10-36 3D Modeler Editor Script Commands
Reporter commands should be executed by the oDesign object. One example of accessing this object is:

```vbscript
Set oDesign = Project.SetActiveDesign("HFSSModel1")
oDesign.CommandName <args>
```

Note: HFSS version 9 does not fully support Reporter scripting. Operations performed in the Reporter will not be automatically recorded; however the script commands in this chapter can be used to create and modify reports from a script.
### CreateReport

**Use:**
Creates a new report and adds it to the Results branch in the project tree.

**Command:**
None

**Syntax:**
CreateReport <ReportDataArray>

**Return Value:**
None

**Parameters:**
<ReportDataArray>

<table>
<thead>
<tr>
<th>Array</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;NAME:&lt;ReportName&gt;&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;ReportType:=&quot;</td>
<td>&lt;string&gt;</td>
</tr>
<tr>
<td>&quot;DisplayType:=&quot;</td>
<td>&lt;string&gt;</td>
</tr>
<tr>
<td>&lt;TraceArray&gt;</td>
<td></td>
</tr>
</tbody>
</table>

**ReportType**

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.

**DisplayType**

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:
Introduction to Scripting in HFSS

"Rectangular Plot", "Polar Plot", "Data Table", "3D Rectangular Plot", or "3D Polar Plot"

<TraceArray>
Array("NAME:Traces",
   <OneTraceArray>, <OneTraceArray>,...)

<OneTraceArray>
Array("NAME:<TraceName>,
   "SolutionName:=","string",
   "Context:=","string",
   <DisplayTypeDependentData>)

<SolutionName>
Name of the solution as listed in the Traces dialog box.
For example: “Setup1 : Last Adaptive”

<Context>
Context for which the output variable expression is being evaluated.
This can be an empty string if there is no context (for example, for S parameters).
Example: "Infinite Sphere1" or "Line1" or ""

<DisplayTypeDependentData>
This data will vary according to the display type. See the examples below.

Example:
```plaintext
doDesign.CreateReport Array("NAME:Rept2DRectTime",
   "ReportType:=","Modal S Parameters",
   "DisplayType:=","Rectangular Plot",
   Array("NAME:Traces",
      Array("NAME:Trace1",
         "SolutionName:=","Setup2 : Sweep1",
         "Context:=","string",
         <DisplayTypeDependentData>)
   )
   )
```

Reporter Editor Script Commands 11-3
"YComponent:=",
    "db(S(WavePort1:1,WavePort2:1))",_
    "YAxis:=" 1)))

Example:

oDesign.CreateReport Array("NAME:Rept2DRectFreq",_
    "ReportType:=" "Modal S Parameters",_
    "DisplayType:=" "Rectangular Plot", _
    Array("NAME:Traces", _
        Array("NAME:Trace2", _
            "SolutionName:=" "Setup1 : Sweep1",_
            "Context:=" "", _
            "Freq:=" Array("All"), _
            "XComponent:=" "freq",_
            "YComponent:=",
            "db(S(WavePort1:1,WavePort2:1))",_
            "YAxis:=" 1)))

Example:

oDesign.CreateReport Array("NAME:Rept2DRectOuVar2Trac",_
    "ReportType:=" "Modal S Parameters",_
    "DisplayType:=" "Rectangular Plot", _
    Array("NAME:Traces", _
        Array("NAME:Trace1", _
            "SolutionName:=" "Setup1 : Sweep1",_
            "Context:=" "", _
            "Freq:=" _
            Array("12GHz","13GHz","14GHz","15GHz"), _
            "XComponent:=" "freq",_
            "YComponent:=" "outputvarS11",_
            "YAxis:=" 1), _
        Array("NAME:Trace2", _
            "SolutionName:=" "Setup1 : Sweep1",_
            "Context:=" "", _
            "Freq:=" Array("All"), _
            "XComponent:=" "freq",_
            "YComponent:=",
            "mag(S(WavePort1:1,WavePort2:1))",_
            "YAxis:=" 0))

11-4 Reporter Editor Script Commands
Example:

```plaintext
oDesign.CreateReport Array("NAME:Rept3DPolarNearFields", _
  "ReportType:=","Near Fields", _
  "DisplayType:=","3D Polar Plot", _
  Array("NAME:Traces", _
    Array("NAME:Tracea", _
      "SolutionName:=","Setup1 : LastAdaptive", _
      "Context:=","Sphere1", _
      "phi:="", Array("0deg","360deg","10deg"), _
      "theta:="", Array("0deg","180deg","10deg"), _
      "freq:="", Array("12GHz"), _
      "XComponent:="", "phi", _
      "YComponent:="", "theta", _
      "ZComponent:="", "NearETotal")))
```

Example:

```plaintext
oDesign.CreateReport Array("NAME:Rept3DPolarFarFields", _
  "ReportType:=","Far Fields", _
  "DisplayType:=","3D Polar Plot", _
  Array("NAME:Traces", _
    Array("NAME:Trace1", _
      "SolutionName:=","Setup1 : LastAdaptive", _
      "Context:=","Infinite Sphere1", _
      "phi:="", Array("0deg","360deg","10deg"), _
      "theta:="", Array("0deg","180deg","10deg"), _
      "freq:="", Array("All"), _
      "XComponent:="", "phi", _
      "YComponent:="", "theta", _
      "ZComponent:="", "rETotal")))
```

Example:

```plaintext
oDesign.CreateReport Array("NAME:ReptFieldsRadiation", _
  "ReportType:=","Fields", _
  "DisplayType:=","Radiation Pattern", _
  Array("NAME:Traces", _
    Array("NAME:Trace1", _
      "SolutionName:=","Setup1 : LastAdaptive", _
      "Context:=","Polyline1", _
      "NormalizedDistance:=",
      Array("0","1","0.01"), _
```

Report Editor Script Commands 11-5
"Freq:=" , Array("12GHz"), _
"Phase:=" , Array("0deg", "0deg", "0deg"), _
"XComponent:=" , "NormalizedDistance", _
"YComponent:=" , "Mag_E")}

Example: oDesign.CreateReport Array("NAME:Rept3DRectPlot2Traces" , _
"ReportType:" , "Modal S Parameters" , _
"DisplayType:" , "3D Rectangular Plot" , _
Array("NAME:Traces" , _
Array("NAME:Trace1" , _
"SolutionName:" , "Setup1 : Sweep1" , _
"Context:" , "" , _
"freq:" , Array("All") , _
"$height:" , Array("All") , _
"XComponent:" , "freq" , _
"YComponent:" , "$height" , _
"ZComponent:" , _
"db(S(WavePort1:1,WavePort1:1))" , _
"YAxis:" , 0) , _
Array("NAME:Trace2" , _
"SolutionName:" , "Setup1 : Sweep1" , _
"Context:" , "" , _
"freq:" , Array("All") , _
"$height:" , Array("All") , _
"XComponent:" , "freq" , _
"YComponent:" , "$height" , _
"ZComponent:" , _
"mag(S(WavePort1:1,WavePort2:1))" , _
"YAxis:" , 1)))

Example: oDesign.CreateReport Array("NAME:ReptDataTable" , _
"ReportType:" , "Modal S Parameters" , _
"DisplayType:" , "Data Table" , _
Array("NAME:Traces" , _
Array("NAME:Trace1" , _
"SolutionName:" , "Setup1 : Sweep1" , _
"Context:" , "" , _
"freq:" , Array("All") , _
"$height:" , Array("All") , _
"XComponent:" , "freq" , _
"YComponent:" , "$height" , _
"ZComponent:" , _
"db(S(WavePort1:1,WavePort1:1))" , _
"YAxis:" , 0) , _
Array("NAME:Trace2" , _
"SolutionName:" , "Setup1 : Sweep1" , _
"Context:" , "" , _
"freq:" , Array("All") , _
"$height:" , Array("All") , _
"XComponent:" , "freq" , _
"YComponent:" , "$height" , _
"ZComponent:" , _
"mag(S(WavePort1:1,WavePort2:1))" , _
"YAxis:" , 1)))

11-6 Reporter Editor Script Commands
Introduction to Scripting in HFSS

"Freq:=", Array("All"), _
"XComponent:=", "freq", _
"YComponent:=", _
   "db(S(WavePort1:1,WavePort2:1))", _
   "YAxis:="", 0))

Example:
doDesign.CreateReport Array("NAME:ReptSmithTime", _
   "ReportType:=""Modal S Parameters", _
   "DisplayType:=""Smith Chart", _
   Array("NAME:Traces", _
      Array("NAME:Trace1", _
         "SolutionName:=","Setup2 : Sweep1", _
         "Context:=", _
         "Domain:=Time,IsImpulse:=true,RiseTime:=0", _
         "Time:="", Array("All"), _
         "XComponent:="", "Time", _
         "YComponent:=", "S(WavePort1:1,WavePort2:1)")))

Example:
doDesign.CreateReport Array("NAME:ReptSmithFreq", _
   "ReportType:=""Modal S Parameters", _
   "DisplayType:=""Smith Chart", _
   Array("NAME:Traces", _
      Array("NAME:Trace1", _
         "SolutionName:=","Setup1 : Sweep1", _
         "Context:=", _
         "Freq:="", Array("All"), _
         "XComponent:="", "freq", _
         "YComponent:=", _
         "S(WavePort1:1,WavePort2:1)")))

Example:
doDesign.CreateReport Array("NAME:ReptPolarSParam", _
   "ReportType:=""Modal S Parameters", _
   "DisplayType:=""Polar Plot", _
   Array("NAME:Traces", _
      Array("NAME:Trace2", _
         "SolutionName:=","Setup1 : Sweep2", _
         "Context:=", _
         "Freq:="", Array("All"), _
"YComponent:="_,
``S(WavePort1:1,WavePort1:1)``)

**RemoveReport**

**Use:** Deletes an existing report.

**Command:** Right-click the report to delete in the project tree, and then click *Delete* on the shortcut menu.

**Syntax:** `RemoveReport <ReportName>`

**Return Value:** None
Boundary and excitation commands should be executed by the “Boundary-Setup” 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>)
General Commands Recognized by the Boundary/Excitations Module

**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”)
```

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>

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.

AssignCurrent

*Use:* Creates a current source.

*Command:* HFSS>Excitations>Assign>Current

*Syntax:* AssignCurrent <CurrentArray>

*Return Value:* None

*Parameters:*  
\[
\text{<CurrentArray>} = \left\{
\begin{array}{l}
\text{Array("NAME:<BoundName>",}\n\text{ "Objects:="}, <AssignmentObjects>,\n\text{ "Current:="}, <value>,\n\text{ <DirectionArray>,}\n\text{ "Faces:="}, <AssignmentFaces>)
\end{array}
\right.
\]

\[
\text{<DirectionArray>} = \left\{
\begin{array}{l}
\text{Array("NAME:Direction",}\n\text{ "Start:="}, <LineEndPoint>,\n\text{ "End:="}, <LineEndPoint>)
\end{array}
\right.
\]

*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:*

```plaintext```
oModule.AssignFiniteCond Array("NAME:FiniteCond1"," 
  "UseMaterial:=" , false, 
  "Conductivity:=" , "58000000", 
  "Permeability:=" , "1", 
  "InfGroundPlane:=" , false, 
  "Faces:=" , Array(12))```

*Example:*

```plaintext```
oModule.AssignFiniteCond Array("NAME:FiniteCond1"," 
  "UseMaterial:=" , true, _ 
  "Material:=" , "copper",_ 
  "InfGroundPlane:=" , false, _ 
  "Faces:=" , Array(12))```

12-6 Boundary and Excitation Module Script
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>,
      "Objects:="; <AssignmentObjects>,
      "Faces:="; <AssignmentFaces>)
Example: oModule.AssignIncidentWave Array("NAME:Incident1",_
   "IsCartesian:"; true,
   "EoX:"; 0,
   "EoY:"; 0,
   "EoZ:"; 0,
   "kX:"; 0,
   "kY:"; 0,
   "kZ:"; 0,
   "PhiStart:"; 0,
   "PhiStop:"; 360,
   "PhiPoints:"; 1000,
   "Objects:"; Array(12),
   "Faces:"; Array(12))
IsCartesian
   If true, provide the EoX, EoY, EoZ, kX, kY, kZ parameters.
   If false, provide the PhiStart, PhiStop, PhiPoints, ThetaStart, ThetaStop, 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",,

Boundary and Excitation Module Script Commands 12-9
AssignLumpedPort

Use: Creates a lumped port.

Command: HFSS>Excitations>Assign>Lumped Port

Syntax: AssignLumpedPort <LumpedPortArray>

Return Value: None

Parameters: <LumpedPortArray>

AssignLumpedPort

Example: oModule.AssignLumpedPort Array("NAME:LumpPort1", _

Example: oModule.AssignLumpedPort Array("NAME:LumpPort1", _

Example: oModule.AssignLumpedPort Array("NAME:LumpPort1", _

Example: oModule.AssignLumpedPort Array("NAME:LumpPort1", _

Example: oModule.AssignLumpedPort Array("NAME:LumpPort1", _

12-10 Boundary and Excitation Module Script
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:=""100Ohm",_
  "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)),

Boundary and Excitation Module Script Commands 12-11
**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>

Example:
```plaintext
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>

Example:
```plaintext
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>,
     <bool>)

12-14 Boundary and Excitation Module Script
Introduction to Scripting in HFSS

“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>”,
Boundary and Excitation Module Script Commands 12-15


Introduction to Scripting in HFSS

AssignSymmetry

Example:

```python
oModule.AssignSymmetry Array("NAME:Sym1",
    "IsPerfectE:=", true,
    "Faces:=", Array(12))
```

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>)
```

Example:

```python
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>",
    "NumModes:="<int>,
    "PolarizeEField:="<bool>,
    "DoDeembed:="<bool>,
    "DeembedDist:="<value>,
    "DoRenorm:="<bool>,
    "RenormValue:="<value>,
    <ModesArray>,
    <TerminalsArray>,
    <DifferentialPairsArray>,
    "Faces:="<AssignmentFaces>)`

- `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”

<TerminalsArray>
Specify for terminal problems.
Array("NAME:Terminals",
    <OneTerminalArray>, <OneTerminalArray>, ...)

<OneTerminalArray>
Array("NAME:<TerminalName>", "TermNum:="<int>,
    <TermLineArray>)

<TerminalName>
Type: <string>
Name of the terminal.

<TermLineArray>
Array("NAME:TermLine",
    "Start:="<LineEndPoint>,
    "End:="<LineEndPoint>)

<DifferentialPairsArray>
Array("NAME:DiffPairs",
    <OneDiffPairArray>, <OneDiffPairArray>, ...)

12-18 Boundary and Excitation Module Script
<OneDiffPairArray>
  Array("NAME:Pair1",_
    "PosTerm:=", <string>,
    "NegTerm:=", <string>,
    "CommonName:=", <string>,
    "CommonRefZ:=", <value>,
    "DiffName:=", <string>,
    "DiffRefZ:=", <value>)

PosTerm
  Name of the terminal to use as the positive terminal.

NegTerm
  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: Modal problem:
  oModule.AssignWavePort Array("NAME:WavePort1",_
    "NumModes:="", 2,_
    "PolarizeEField:="",false,_
    "DoDeembed:="", true,_
    "DeembedDist:="", "10mil",_
    "DoRenorm:="", true,_
    "RenormValue:="", "50Ohm",
  Array("NAME:Modes",_
Introduction to Scripting in HFSS

Array("NAME:Model1", _
   "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:

```plaintext
doModule.AssignWavePort Array("NAME:WavePort1", _
   "NumModes:=" , 2, _
   "PolarizeEField:=" , false, _
   "DoDeembed:=" , false, _
   "RenormImp:=" , "50Ohm", _
   Array("NAME:Terminals", _
      Array("NAME:T1", _
         "TermNum:=" , 1 , Array("NAME:TermLine", _
            "Start:=" , Array("-0.2mm","-1.6mm","-1.8mm"), _
            "End:=" , Array("-0.2mm","-1.6mm","0mm"))), _
      Array("NAME:T2", _
         "TermNum:=" , 2 , Array("NAME:TermLine", _
            "Start:=" , Array("-0.2mm","-1.4mm","-1.8mm"), _
            "End:=" , Array("-0.2mm","-1.4mm","0mm"))), _
      Array("NAME:DiffPairs", _
         Array("NAME:Pair1", _
            "PosTerm:=" , "T1", _
            "NegTerm:=" , "T2", _
            "CommonName:=" , "Comml", _
            "CommonRefZ:=" , "100Ohm", _
            "DiffName:=" , "Diff1", _
            "DiffRefZ:=" , "300Ohm")), _
```
"Faces:=" Array(11))

**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

**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> <LumpedRLCArrary>

*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

**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
Script Commands for Creating and Modifying PMLs

Following are script commands for creating and modifying 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>)
```

For automatically created PMLs:

```
CreatePML Array("UserDrawnGroup:="", false,
             "PMLFaces:="", <AssignmentFaces>,
             "CreateCornerObjs:="", <bool>,
             "Thickness:="", <value>,
             "RadDist:="", <value>,
             "UseFreq:="", <bool>,
             "MinFreq:="", <value>,
             "MinBeta:="", <double>)
```

*Return Value:* None
Introduction to Scripting in HFSS

Boundary and Excitation Module Script Commands

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:

```plaintext
oModule.CreatePML Array("UserDrawnGroup:="", false,_
  "PMLFaces:="", Array(120), "CreateCornerObjs:="", true,_
  "Thickness:="", "0.33mm", "RadDist:="", "1.6mm",_
  "UseFreq:="", true, "MinFreq:="", "1GHz")
```

Example:

```plaintext
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:**
```plaintext
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:**
```plaintext
oModule.ModifyPMLGroup Array("NAME:PMLGroup1",
    "RadDist:="., "1.166666667mm",
    "UseFreq:="., false, "MinFreq:="., 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:**
```plaintext
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
Introduction to Scripting in HFSS

12-28 Boundary and Excitation Module Script
Mesh setup and operations commands should be executed by the “Mesh-Setup” module.

```vbscript
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")

**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:

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, _
"MaxLength:=" ,1000, _
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:**

```
Array("NAME:<OpName>",
    "Faces:=" , <AssignmentFaces>,
    "RestrictElem:=" , <bool>,
    "NumMaxElem:=" , <int>,
    "SkinDepth:=" , <value>,
    "SurfTriMaxLength:=" , <value>,
    "NumLayers:=" , <int>)
```

**RestrictElem**

If true, NumMaxElem should be specified.

**Example:**

```python
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:**

```
Array("NAME:<OpName>",
    "Faces:=" , <AssignmentFaces>,
    "RestrictElem:=" , <bool>,
    "NumMaxElem:=" , <int>,
    "SkinDepth:=" , <value>,
    "SurfTriMaxLength:=" , <value>,
    "NumLayers:=" , <int>)
```

13-4 Mesh Operations Module Script Commands
"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")
**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, you must 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)
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:**

```plaintext
oModule.DeleteSetups Array("Setup1", "Setup2")
```

### EditDrivenSweep

**Use:** Modifies an existing frequency sweep.

**Command:** Double-click a frequency sweep in the project tree to modify its settings.

**Syntax:** `EditDrivenSweep <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>",
          "Type:=", <SweepType>,
          "SetupType:=", <SetupType>,
          ...
Introduction to Scripting in HFSS

<FrequencyInformation>,
<DEXtrapInfo>)

See the InsertDrivenSweep command for details.

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.

InsertDrivenSweep

Use: Adds a frequency sweep to a Driven solution-type setup.

Command: HFSS>Analysis Setup>Add Sweep

Syntax: InsertDrivenSweep <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>", "Type:="<SweepType>,
"SetupType:="<SetupType>,
<FrequencyInformation>,
<DEXtrapInfo>)

<SweepType>
Can be “Discrete”, “Fast”, or “Interpolating”.

Analysis Module Script Commands 14-3
<SetupType>
Can be “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: A Discrete sweep, specified using a list of single frequency points:
```
oModule.InsertDrivenSweep "Setup1", _
   Array("NAME:Sweep1", "Type:="; "Discrete", _
   "SetupType:="; "SinglePoints", _
   "FrequencyList:="; Array("1GHz", "2GHz", "3GHz"), _
   "SaveFieldsList:="; Array(false, true, false))
```

Example: A Fast sweep, specified using the starting and stopping frequencies and the step size:
```
oModule.InsertDrivenSweep "Setup1", _
   Array("NAME:Sweep2", "Type:="; "Fast", _
   "SetupType:="; "LinearStep", _
   "StartFreq:="; "1GHz", "StopFreq:="; "10GHz", _
   "StepSize:="; "0.1GHz", "SaveFields:="; true, _
   "ExtrapToDC:="; false)
```

Example: An Interpolating sweep, specified using the starting and stopping frequencies and the number of steps, and extrapolating to DC:
```
oModule.InsertDrivenSweep "Setup1", _
   Array("NAME:Sweep3", "Type:="; "Interpolating", _
   "InterpTolerance:="; 0.5, "InterpMaxSolns:="; 20, _
   "SetupType:="; "LinearCount", _
   "StartFreq:="; "0GHz", "StopFreq:="; "10GHz", _
   "Count:="; 101, "SaveFields:="; false,
   "ExtrapToDC:="; true, "MinSolvedFreq:="; "0.1GHz", _
   "DoSnapping:="; true, "SnapTolerance:="; 0.01)
```
**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. See the examples below.

*Example:* A Driven solution type with no ports:

```plaintext
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)
```

*Example:* A Driven solution type with ports:

```plaintext
oModule.InsertSetup "HfssDriven", _
    Array("NAME:Setup1", _
        "Frequency:="", "1GHz", _
        "PortsOnly:="", false, _
        "MaxDeltaS:="", 0.02, _
        "UseMatrixConv:="", false, _
```


Introduction to Scripting in HFSS

"MaximumPasses:=", 3, _
"MinimumPasses:=", 1, _
"MinimumConvergedPasses:=", 1, _
"PercentRefinement:=", 20, _
"ReducedSolutionBasis:=", false, _
"DoLambdaRefine:=", true, _
"DoMaterialLambda:=", true, _
"Target:=", 0.3333, _
"PortAccuracy:=", 2, _
"SetPortMinMaxTri:=", false)

Example:
An Eigenmode solution type:

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

14-6 Analysis Module Script Commands
Introduction to Scripting in HFSS

Analysis Module Script Commands 14-7

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
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.
Introduction to Scripting in HFSS

<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.

15-2 Optimetrics Module Script Commands
<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:=”,
      <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**
*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")

**RenameSetup**
*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**
*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
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
Use: Inserts a new parametric setup.
Command: Right-click the Optimetrics folder in the project tree, and then click Add> Parametric on the shortcut menu.
Syntax: InsertSetup “OptiParametric”, <ParametricParams>
Return Value: None
Parameters:
<Parametric Params>
Array("NAME:<SetupName>“, “SaveFields:=”,
<SaveField>, <StartingPoint>, “Sim. Setups:=”,
<SimSetups>,
<SweepDefs>, <SweepOps>,
Array(“NAME:Goals”, Array(“NAME:Goal”,
<OptiGoalSpec>), ... Array(“NAME:Goal”,
<OptiGoalSpec>)

<SetupName>
Type: <string>
Name of the parametric setup.

<SimSetups>
Type: Array of strings.
An array of HFSS 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, _

Optimetrics Module Script Commands 15-7
"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", _
"Start:=", "8GHz", "Stop:=", "8GHz"), _
Array("NAME:Goal", _
"Solution:=", "Setup1 : LastAdaptive", _
"Calculation:=", "reflect", _
"Context:=", "", _
Array("NAME:Ranges", _
"Start:=", "8GHz", "Stop:=", "8GHz")))
)
Optimization Script Commands

**EditSetup**

*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**

*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>

```

<OptimizationVars>
```
Introduction to Scripting in HFSS

```
"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>

15-10 Optimetrics Module Script Commands
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:
```plaintext
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:=", "", _
```

Optimetrics Module Script Commands 15-11
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**

*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

*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>)

  Array("NAME:Variables",
  "VarName:=",
  Array("i:=",
  <IncludeVar>,
  "Min:=",
  <MinV>, "Max:=",
  <MaxV>,
  "IDisp:=",
  <InitialDisp>),

  "VarName:=",
  Array("i:=",
  <IncludeVar>,
  "Min:=",
  <MinV>, "Max:=",
  <MaxV>,
  "IDisp:=",
  <InitialDisp>)

  Array("NAME:Variables",
  "VarName:=",
  Array("i:=",
  <IncludeVar>,
  "Min:=",
  <MinV>, "Max:=",
  <MaxV>,
  "IDisp:=",
  <InitialDisp>),

  "VarName:=",
  Array("i:=",
  <IncludeVar>,
  "Min:=",
  <MinV>, "Max:=",
  <MaxV>,
  "IDisp:=",
  <InitialDisp>))

  Array("NAME:Variables",
  "VarName:=",
  Array("i:=",
  <IncludeVar>,
  "Min:=",
  <MinV>, "Max:=",
  <MaxV>,
  "IDisp:=",
  <InitialDisp>),

  "VarName:=",
  Array("i:=",
  <IncludeVar>,
  "Min:=",
  <MinV>, "Max:=",
  <MaxV>,
  "IDisp:=",
  <InitialDisp>))

  Array("NAME:Variables",
  "VarName:=",
  Array("i:=",
  <IncludeVar>,
  "Min:=",
  <MinV>, "Max:=",
  <MaxV>,
  "IDisp:=",
  <InitialDisp>),

  "VarName:=",
  Array("i:=",
  <IncludeVar>,
  "Min:=",
  <MinV>, "Max:=",
  <MaxV>,
  "IDisp:=",
  <InitialDisp>))

  Array("NAME:Variables",
  "VarName:=",
  Array("i:=",
  <IncludeVar>,
  "Min:=",
  <MinV>, "Max:=",
  <MaxV>,
  "IDisp:=",
  <InitialDisp>),

  "VarName:=",
  Array("i:=",
  <IncludeVar>,
  "Min:=",
  <MinV>, "Max:=",
  <MaxV>,
  "IDisp:=",
<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", _
         )
      )
   )
)
Introduction to Scripting in HFSS

Optimetrics Module Script Commands

Statistical Script Commands

EditSetup

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

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:=",
"PreSolvePSetup:=",
"Preceed>, <StatisticalVars>,
Array("NAME:Goals", Array("NAME:Goal",
<OptiGoalSpec>), …, Array("NAME:Goal",
<OptiGoalSpec>)),

<StatisticalVars>

Array("NAME:Variables",
"VarName:=",
<DistType>, "Tol:=",
"StdD:=",
"Min:=",
"Max:=",
<MaxCutoff>, …)
"VarName:=", Array("i:=", <IncludeVar>, "Dist:=",
<DistType>, "Tol:=", <Tolerance>, "StdD:=",
<StdD>, "Min:=", <MinCutoff>, "Max:=",
<MaxCutoff>))

<DistType>
Type: <string>
Distribution 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"))}
Introduction to Scripting in HFSS

Optimetrics Module Script Commands 15-17

```
"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"))})
```
Introduction to Scripting in HFSS

15-18 Optimetrics Module Script Commands
Solutions commands should be executed by the “Solutions” module.
Set oModule = oDesign.GetModule("Solutions")
oModule.CommandName <args>
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")
Introduction to Scripting in HFSS

**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:
```python
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:
```python
DeleteVariation <VariationArray>, <FullVariations>,
    <MeshAndFieldsOnly>, <FieldsOnly>
```

Return Value: None

Parameters:

- **<VariationArray>**
  ```python
  Array(<DesignVariationKey>, <DesignVariationKey>,...)
  ```

- **<FullVariations>**
  ```python
  Type: <bool>
  Specifies whether to delete meshes, fields, matrix data, profile, and convergence data.
  ```

- **<MeshAndFieldsOnly>**
  ```python
  Type: <bool>
  Specifies whether to delete only meshes and fields.
  ```

- **<FieldsOnly>**
  ```python
  Type: <bool>
  Specifies whether to delete fields only.
  ```

Example:
```python
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:

```python
oModule.ExportForSpice "width='2in'", _
   Array("Setup1:Sweep1"), 2, 0, _
   "c:\mydir\Sweep1.fws", "", "", "", ""
```

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>
Introduction to Scripting in HFSS

<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

16-8 Solutions Module Script Commands
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
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)
Field overlays 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_Hx", "Vector_Hy", "Vector_Hz"
"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>, ...
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>,

17-4 Field Overlays Module Script Commands
"Style:="<string>),
"IsoValType:="<string>,
"ArrowUniform:="<bool>,
"NumofArrow:="<int>)

Style
Possible values are “Cylinder”, “Solid”, “Dashdash”,
“Dotdot”, “Dotdash”

IsoValType
Possible values are “Tone”, “Fringe”, “Gourard”

<PlotOnSurfaceSettings>
Array("NAME:PlotOnSurfaceSettings",
"Filled:="<bool>,
"IsoValType:="<string>,
"SmoothShade:="<bool>,
"AddGrid:="<bool>,
"MapTransparency:="<bool>,
"Transparency:="<double>,
"ArrowUniform:="<bool>
"ArrowSpacing:="<double>
"GridColor:="<int>, 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>,
"Color:="<string>,
"IsoValType:="<string>,
"ArrowUniform:="<bool>,
"ArrowSpacing:="<double>
"GridColor:="<int>, Array(<int>, <int>, <int>)

Field Overlays Module Script Commands 17-5
"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")
```
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>)
</Scale3DSettings>

<Marker3DSettings>
  Array("NAME:Marker3DSettings",
        "MarkerType:=". <int>,
        "MarkerMapSize:=". <bool>,
        "MarkerMapColor:=". <bool>,
        "MarkerSize:=". <double>)
</Marker3DSettings>

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>)
</Arrow3DSettings>

ArrowType
  0: Line
1: Cylinder
2: Umbrella
default: Line

**Example:**

```plaintext
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))
```
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.
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: <Name>
   Type: <string>
   Name for the new named expression.
Example: oModule.AddNamedExpr “Mag_JxE”

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, +

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”
**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>`  
  *The solution name to use.*
- `<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:

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:

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:

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: Enter 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.

WriteRegister

Use: Evaluates the top stack element for all tetrahedrons and writes the data to a file.

Command: Click Write.

Syntax: Write <OutputFile>

Return Value: None

Parameters:

<OutputFile>
Type: <string>
Name of the output file.
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")
**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:=""CS1")
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:

- 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>

```plaintext
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:**

```plaintext
oModule.InsertNearFieldSphereSetup _
  Array("NAME:MySphere", _
    "UseCustomRadiationSurface:="true, _
    "CustomRadiationSurface:="FaceList1", _
    "ThetaStart:"0deg", "ThetaStop:"180deg", _
    "ThetaStep:"10deg", "PhiStart:"0deg", _
    "PhiStop:"360deg", "PhiStep:"10deg", _
    "UseLocalCS:="true, _
```

---

19-8 Radiation Module Script Commands
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>,
```
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>, ...)
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,_
```

Radiation Module Script Commands 19-11
"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")
Example Scripts
Introduction to Scripting in HFSS

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.

```vbscript
Dim oHfssApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oHfssApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oHfssApp.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 = "m"
Case "cm"
```

20-2 Example Scripts
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)
Introduction to Scripting in HFSS

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")

20-4 Example Scripts
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.

```vbscript
Dim oHfssApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oHfssApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oHfssApp.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(WavePort1,WavePort1))"
oDesign.AddOutputVariable "im_S", "im(S(WavePort1,WavePort1))"
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

3D Modeler editor commands

AssignMaterial 10-20
Connect 10-20
Copy 10-16
CoverLines 10-20
CoverSurfaces 10-20
CreateBondwire 10-3
CreateBox 10-4
CreateCircle 10-4
CreateCone 10-5
CreateCutplane 10-5
CreateCylinder 10-6
CreateEllipse 10-6
CreateEntityList 10-21
CreateFaceCS 10-21
CreateHelix 10-7
CreateObjectFromFaces 10-23
CreatePoint 10-8
CreatePolyline 10-8
CreateRectangle 10-9
CreateRegularPolygon 10-10
CreateRegularPolyhedron 10-10
CreateRelativeCS 10-24
CreateSphere 10-11
CreateSpiral 10-12
CreateTorus 10-12
Delete 10-34
DeleteLastOperation 10-24
DetachFaces 10-25
DuplicateAlongLine 10-16
DuplicateAroundAxi 10-16
DuplicateMirror 10-17
EditEntityList 10-25
EditFaceCS 10-26
EditPolyline 10-13
EditRelativeCS 10-26
Export 10-26
GenerateHistory 10-27
GetFaceByPosition 10-34
Import 10-27
Intersect 10-28
Mirror 10-17
Move 10-18
MoveFaces 10-28
OffsetFaces 10-18
PageSetup 10-35
Paste 10-18
RenamePart 10-35
Rotate 10-19
Scale 10-19
Section 10-29
Introduction to Scripting in HFSS

A

AbortSolveAsynch 8-2
AddDataset 7-2
AddMaterial 5-2
AddNamedExpr 18-2
AddOutputVariable 9-2

Analysis module commands
  DeleteDrivenSweep 14-2
  DeleteSetups 14-2
  EditDrivenSweep 14-2
  EditSetup 14-3
  InsertDrivenSweep 14-3
  InsertSetup 14-5
  RenameDrivenSweep 14-7
  RenameSetup 14-8
  RevertAllToInitial 14-8
  RevertSetupToInitial 14-8
  SolveSetup 14-8
  ApplyMeshOps 8-2
  arithmetic operators 1-5
  array variables 1-4
  AssignCurrent 12-5
  AssignFiniteCond 12-6
  AssignImpedance 12-7
  AssignIncidentWave 12-7
  AssignLayeredImp 12-8
  AssignLengthOp 13-3
  AssignLumpedPort 12-10
  AssignLumpedRLC 12-11
  AssignMagneticBias 12-12
  AssignMaster 12-13
  AssignMaterial 10-20
  AssignPerfectE 12-13
  AssignPerfectH 12-14
  AssignRadiation 12-14
  AssignSkinDepthOp 13-4
  AssignSlave 12-14
  AssignSymmetry 12-15
  AssignTrueSurfOp 12-16
  AssignWavePort 12-17

B

Boundary/Excitation module commands
  AssignCurrent 12-5
  AssignFiniteCond 12-6
  AssignImpedance 12-7
  AssignIncidentWave 12-7
  AssignLayeredImp 12-8
  AssignLumpedPort 12-10
  AssignLumpedRLC 12-11
  AssignMagneticBias 12-12
  AssignMaster 12-13
  AssignPerfectE 12-13
  AssignPerfectH 12-14
  AssignRadiation 12-14
  AssignSlave 12-14
  AssignSymmetry 12-15
  AssignVoltage 12-16
  AssignWavePort 12-17
  ChangeImpedanceMult 12-2
  CreatePML 12-24
  DeleteAllBoundaries 12-2
  DeleteAllExcitations 12-2
  DeleteBoundaries 12-3
  EditCurrent 12-21
  EditFiniteCond 12-21
  EditImpedance 12-21
  EditIncidentWave 12-21
  EditLayeredImpedance 12-21
  EditLumpedPort 12-22
Introduction to Scripting in HFSS

EditLumpedRLC 12-22
EditMagneticBias 12-23
EditMaster 12-22
EditPerfectE 12-22
EditPerfectH 12-22
EditRadiation 12-23
EditSlave 12-23
EditSymmetry 12-23
EditVoltage 12-23
EditWavePort 12-23
ModifyPMLGroup 12-26
PMLGroupCreated 12-26
PMLGroupModified 12-27
ReassignBoundaries 12-3
RecalculatePMLMaterials 12-27
RenameBoundary 12-3
ReprioritizeBoundary 12-4

C
CalcOp 18-2
CalcStack 18-2
ChangeGeomSettings 18-3
ChangeImpedanceMult 12-2
ChangeProperty 6-4
ClcEval 18-3
ClcMaterial 18-3
ClearAllNamedExpr 18-4
Close 4-2
CloseProject 3-2
comment lines 1-2
comparison operators 1-6
conditional statements
  If...Then... Else 1-7
  Select Case 1-7
types of 1-7
Connect 10-20
calendar-sensitive help iii
conventions
  command syntax 2-7
data types 2-7
  script command 2-8
converting data types 1-9

Copy 10-16
CopyDesign 4-2
CopyNamedExprToStack 18-4
CoverLines 10-20
CoverSurfaces 10-20
copyright notice ii
CreateBondwire 10-3
CreateBox 10-4
CreateCircle 10-4
CreateCone 10-5
CreateCutplane 10-5
CreateCylinder 10-6
CreateEllipse 10-6
CreateEntityList 10-21
CreateFaceCS 10-21
CreateFieldPlot 17-2
CreateHelix 10-7
CreateObjectFromFaces 10-23
CreatePML 12-24
CreatePoint 10-8
CreatePolyline 10-8
CreateRectangle 10-9
CreateRegularPolygon 10-10
CreateRegularPolyhedron 10-10
CreateRelativeCS 10-24
CreateReport 11-2
CreateSphere 10-11
CreateSpiral 10-12
CreateTorus 10-12
CutDesign 4-2

dataset commands
  AddDataset 7-2
  DeleteDataset 7-2
  EditDataset 7-3
Delete 10-34
DeleteAllBoundaries 12-2
DeleteAllExcitations 12-2
DeleteBoundaries 12-3
DeleteDataset 7-2
DeleteDesign 4-2
Introduction to Scripting in HFSS

DeleteDrivenSweep 14-2
DeleteFarFieldSetup 19-2
DeleteFieldPlot 17-6
DeleteImportData 16-2
DeleteLastOperation 10-24
DeleteNamedExpr 18-4
DeleteNearFieldSetup 19-2
DeleteOp 13-2
DeleteOutputVariable 9-2
DeleteSetups
  Analysis module command 14-2
  Optimization module command 15-5
DeleteSolutionVariation 16-4
DeleteVariation 16-5
Design object commands
  AbortSolveAsynch 8-2
  AddOutputVariable 9-2
  ApplyMeshOps 8-2
  CreateReport 11-2
  DeleteOutputVariable 9-2
  EditOutputVariable 9-3
  GetModule 8-2
  GetName 8-3
  GetOutputVariableValue 9-3
  GetSolveAsynchStatus 8-3
  Redo 8-3
  RemoveReport 11-8
  RenameDesignInstance 8-4
  SARSetup 8-4
  SetActiveEditor 8-4
  SetSolutionType 8-5
  Solve 8-5, 8-6
  SolveAsynch 8-6
Desktop object commands
  CloseProject 3-2
  GetActiveProject 3-2
  GetProjectList 3-2
  NewProject 3-2
  OpenMultipleProjects 3-3
  OpenProject 3-3
  PauseScript 3-3
  Print 3-4
  QuitApplication 3-4
  RestoreWindow 3-4
  RunProgram 3-5
  RunScript 3-6
  SetActiveProject 3-6
  Sleep 3-7
  DuplicateFaces 10-25
  DuplicateAlongLine 10-16
  DuplicateAroundAxis 10-16
  DuplicateMirror 10-17

E
  EditAntennaArraySetup 19-9
  EditCurrent 12-21
  EditDataset 7-3
  EditDrivenSweep 14-2
  EditEntityList 10-25
  EditFaceCS 10-26
  EditFarFieldSphereSetup 19-4
  EditFiniteCond 12-21
  EditImpedance 12-21
  EditIncidentWave 12-21
  EditLayeredImpedance 12-21
  EditLengthOp 13-5
  EditLumpedPort 12-22
  EditLumpedRLC 12-22
  EditMagneticBias 12-23
  EditMaster 12-22
  EditMaterial 5-3
  EditNearFieldLineSetup 19-4
  EditNearFieldSphereSetup 19-5
  EditOutputVariable 9-3
  EditPerfectE 12-22
  EditPerfectH 12-22
  EditPolyline 10-13
  EditRadiation 12-23
  EditRelativeCS 10-26
  EditSetup
    Analysis module command 14-3
    Optimization command 15-9
    Parametric command 15-6
    Sensitivity command 15-13
Introduction to Scripting in HFSS

statistical command 15-15
EditSkinOp 13-6
EditSlave 12-23
EditSources 16-2
EditSymmetry 12-23
EditTrueSurfOp 13-6
EditVoltage 12-23
EditWavePort 12-23
EnterComplex 18-5
EnterComplexVector 18-5
EnterLine 18-6
EnterPoint 18-6
EnterQty 18-6
EnterScalar 18-7
EnterScalarFunc 18-7
EnterSurf 18-7
EnterVector 18-8
EnterVectorFunc 18-8
EnterVol 18-9
Export 10-26
ExportForHSpice 16-7
ExportForSpice 16-6
ExportMaterial 5-3
ExportNetworkData 16-9
ExportNMFData 16-10
ExportOnGrid 18-9
ExportRadiationParametersToFile 19-13
ExportToFile 18-10

Fields Calculator commands
AddNamedExpr 18-2, 18-4
CalcOp 18-2
CalcStack 18-2
ChangeGeomSettings 18-3
ClcEval 18-3
ClcMaterial 18-3
ClearAllNamedExpr 18-4
CopyNamedExprToStack 18-4
CreateFieldPlot 17-2
DeleteFieldPlot 17-6
DeleteNamedExpr 18-4
Entering Complex 18-5
Entering Complex Vector 18-5
Entering Line 18-6
Entering Point 18-6
Entering Qty 18-6
Entering Scalar 18-7
Entering Scalar Func 18-7
Entering Surf 18-7
Entering Vector 18-8
Entering Vector Func 18-8
Entering Vol 18-9
Export On Grid 18-9
Export To File 18-10
Modify Field Plot 17-7
Rename Field Plot 17-8
Rename Plot Folder 17-8
Set Field Plot Settings 17-9
Set Plot Folder Settings 17-10
Write Register 18-10

Field Overlays module commands
AddNamedExpr 18-2
CalcOp 18-2
CalcStack 18-2
ChangeGeomSettings 18-3
ClcEval 18-3
ClcMaterial 18-3
CopyNamedExprToStack 18-4
DeleteNamedExpr 18-4
Entering Complex 18-5
Entering Complex Vector 18-5
Entering Line 18-6
Entering Point 18-6
Entering Qty 18-6
Entering Scalar 18-7
Entering Scalar Func 18-7
Entering Surf 18-7
Entering Vector 18-8
Entering Vector Func 18-8
Entering Vol 18-9
Export On Grid 18-9
Export To File 18-10
Write Register 18-10

For...Next loop 1-8
Introduction to Scripting in HFSS

G
GenerateHistory 10-27
GetActiveDesign 4-3
GetActiveProject 3-2
GetDesign 4-3
GetFaceByPosition 10-34
GetModule 8-2
GetName 4-3, 8-3
GetOutputVariableValue 9-3
GetPath 4-3
GetProjectList 3-2
GetProperty 6-9
GetPropertyValue 6-9
GetSolveAsynchStatus 8-3
GetTopDesignList 4-4
GetVariables 6-9
GetVariableValue 6-10

H
help
   Ansoft technical support iii
   context-sensitive iii
   on dialog boxes iii
   on menu commands iii
hierarchy of variables in HFSS 2-2

I
If...Then... Else statement 1-7
Import 10-27
ImportSolution 16-11
ImportTable 16-12
InputBox function 1-9
InsertDesign 4-4
InsertDrivenSweep 14-3
InsertFarFieldSphereSetup 19-6
InsertNearFieldLineSetup 19-7
InsertNearFieldSphereSetup 19-8
InsertSetup
   Analysis module command 14-5
   optimization command 15-9
   parametric command 15-6
   sensitivity command 15-13
   statistical command 15-15
Intersect 10-28

J
JavaScript, script format 1-1

K
keywords, VBScript 1-2

L
logical operators 1-6

M
material commands
   AddMaterial 5-2
   EditMaterial 5-3
   ExportMaterial 5-3
   RemoveMaterial 5-4
Mesh Operations module commands
   AssignLengthOp 13-3
   AssignSkinDepthOp 13-4
   AssignTrueSurfOp 13-4
   DeleteOp 13-2
   EditLengthOp 13-5
   EditSkinOp 13-6
   EditTrueSurfOp 13-6
   RenameOp 13-2
Microsoft
   VBScript user’s guide 1-10
   Visual Basic 1-1
Mirror 10-17
ModifyFieldPlot 17-7
modifying a script 2-7
ModifyPMLGroup 12-26
modules in HFSS scripting 2-4
Move 10-18
MoveFaces 10-28
MsgBox function 1-9

N
NewProject 3-2

O
oDesign object 2-3
oDesktop object 2-3
oEditor object 2-4
OffsetFaces 10-18
oHfssApp object 2-3
oModule object 2-4
OpenMultipleProjects 3-3
OpenProject 3-3
operators
  arithmetic 1-5
  categories in VBScript 1-5
  comparison 1-6
  logical 1-6
  precedence of 1-5
oProject object 2-3
Optimetrics module commands
  DeleteSetups 15-5
  RenameSetup 15-5
  SolveSetup 15-5
optimization commands
  EditSetup 15-9
  InsertSetup 15-9
output variable commands
  AddOutputVariable 9-2
  DeleteOutputVariable 9-2
  EditOutputVariable 9-3
  GetOutputVariableValue 9-3

P
PageSetup 10-35
parametric commands
  EditSetup 15-6
  InsertSetup 15-6
  Paste 4-4, 10-18
  PauseScript 3-3
  pausing a script 2-6
  PMLGroupCreated 12-26
  PMLGroupModified 12-27
  Print 3-4
Project object commands
  AddDataset 7-2
  AddMaterial 5-2
  ChangeProperty 6-4
  Close 4-2
  CopyDesign 4-2
  CutDesign 4-2
  DeleteDataset 7-2
  DeleteDesign 4-2
  EditDataset 7-3
  EditMaterial 5-3
  ExportMaterial 5-3
  GetActiveDesign 4-3
  GetDesign 4-3
  GetName 4-3
  GetPath 4-3
  GetProperties 6-9
  GetPropertyValue 6-9
  GetTopDesignList 4-4
  GetVariables 6-9
  GetVariableValue 6-10
  InsertDesign 4-4
  Paste 4-4
  Redo 4-5
  RemoveMaterial 5-4
  Save 4-5
  SaveAs 4-5
  SetActiveDesign 4-6
  SetPropertyValue 6-10
  SetVariableValue 6-11
  SimulateAll 4-6
  Undo 4-6
property commands
  ChangeProperty 6-4
  GetProperties 6-9
Introduction to Scripting in HFSS

GetPropertyValue 6-9
GetVariables 6-9
GetVariableValue 6-10
SetPropertyValue 6-10
SetVariableValue 6-11

Q
QuitApplication 3-4

R
Radiation module commands
DeleteFarFieldSetup 19-2
DeleteNearFieldSetup 19-2
EditAntennaArraySetup 19-9
EditFarFieldSphereSetup 19-4
EditNearFieldSphereSetup 19-5
EditNearLineSetup 19-4
ExportRadiationParametersToFile 19-13
InsertFarFieldSphereSetup 19-6
InsertNearFieldLineSetup 19-7
InsertNearFieldSphereSetup 19-8
RenameSetup 19-3
ReassignBoundaries 12-3
RecalculatePMLMaterials 12-27
recording a script 2-6
Redo
design-level command 8-3
project-level command 4-5
references, for VBScript 1-10
RemoveMaterial 5-4
RemoveReport 11-8
RenameBoundary 12-3
RenameDesignInstance 8-4
RenameDrivenSweep 14-7
RenameFieldPlot 17-8
RenameOp 13-2
RenamePart 10-35
RenamePlotFolder 17-8
RenameSetup

Analysis module command 14-8
Optimetrics module command 15-5
Radiation module command 19-3

Reporter editor commands
CreateReport 11-2
RemoveReport 11-8
ReprioritizeBoundary 12-4
RestoreWindow 3-4
resuming a script 2-6
RevertAllToInitial 14-8
RevertSetupToInitial 14-8
Rotate 10-19
running a script 2-6
RunProgram 3-5
RunScript 3-6

S
sample scripts
data export 20-6
simple HFSS 1-2
variable helix 20-2
SARSetup 8-4
Save 4-5
SaveAs 4-5
Scale 10-19
scripts
in JavaScript format 1-1
modifying for easier playback 2-7
pausing 2-6
recording 2-6
resuming 2-6
running 2-6
running from command prompt 1-1
stopping execution of 2-7
Section 10-29
Select Case statement 1-7
sensitivity commands
EditSetup 15-13
InsertSetup 15-13
SeparateBody 10-29
Index-9

Introduction to Scripting in HFSS

SetActiveDesign 4-6
SetActiveEditor 8-4
SetActiveProject 3-6
SetFieldPlotSettings 17-9
SetModelUnits 10-30
SetPlotFolderSettings 17-10
SetPropertyValue 6-10
SetSolutionType 8-5
SetVariableValue 6-11
SetWCS 10-30
SimulateAll 4-6
Sleep 3-7
Solutions module commands
  DeleteImportData 16-2
  DeleteSolutionVariation 16-4
  DeleteVariation 16-5
  EditSources 16-2
  ExportForHSpice 16-7
  ExportForSpice 16-6
  ExportNetworkData 16-9
  ExportNMF 16-10
  ImportSolution 16-11
  ImportTable 16-12
Solve 8-5
SolveAsynch 8-6
SolveSetup
  Analysis module command 14-8
  Optimetrics module command 15-5
Split 10-31
statistical commands
  EditSetup 15-15
  InsertSetup 15-15
stopping a script 2-7
stopping script recording 2-6
Sub procedures 1-2
Subtract 10-31
SweepAlongPath 10-14
SweepAlongVector 10-14
SweepAroundAxis 10-15

T

trademark notice ii

U

UncoverFaces 10-32
underscore (_) character 1-3
Undo
  design-level command 8-6
  project-level command 4-6
Unite 10-33

V

variables
  array 1-4
  assigning information 1-4
  declaring 1-4
  hierarchy in HFSS 2-2
  used as objects 1-2
  used in HFSS scripts 2-2
.vbs file format 2-6
VBScript
  Microsoft user’s guide 1-10
  operators 1-5
  overview 1-1
  references 1-10
  Sub procedures 1-2
  .vbs file format 2-6

W

WriteRegister 18-10
Introduction to Scripting in HFSS