EMIT Scripting Guide

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

Release 2024 R2
July 2024

Table of Contents

1 - Introduction to Scripting

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 in the order in which they appear.

Electronics Desktop can record scripts in VBScript or IronPython, and can run external scripts written in VBScript, IronPython, CPython, or JavaScript. Additionally, it contains an IronPython command shell for executing scripts.

When running Ansys Electronics Desktop from the command line, scripts can be written in any language that provides Microsoft COM methods.

The following sections contain more information about scripting:

- Scripting Help Conventions – explains the layout of the scripting help.
- Introduction to VBscript – provides a broad overview of VBscript
- Introduction to IronPython – provides a broad overview of IronPython.
- Introduction to C-Python – provides guidance on using C-Python for Ansys Electronics Desktop scripts.
- Ansys Electronics Desktop Scripting – details instructions and tips for running, recording, and working with scripts in Electronics Desktop.
- PyAEDT (Beta) – a Python library that interacts directly with the AEDT API to make scripting simpler for the end user.

Scripting Help Conventions

The majority of this guide lists individual script commands using the following format.

[ScriptName]

[Description of script use.]

UI Access	[UI commands corresponding to the script command, if any.]
Parameters	[List of arguments taken by the script command, if any. Includes argument types and brief descriptions.]
Return Value	[The script's return value, if any.]

Python Syntax	[Correct syntax for the command in Python. Arguments are enclosed in angle brackets (<>).]
Python Example	`[Sample script]`

VB Syntax	[Correct syntax for the command in VBscript. Arguments are enclosed in angle brackets (<>)
VB Example	`[Sample script]`

Variable Types

The following data types are used throughout the help:

- **<string>** – use within quotation marks.
- **<bool>** – boolean value; should be set to either True or False.
- **<int>** – an integer. For example, 1.
- **<double>** – a double precision value. For example, 1.2.
- **<array>** – in VBscript, an array; in IronPython, a list contained in square brackets.
- **<value>** – can be an integer, string, or VBscript variable, depending on context.

Introduction to VBScript

Ansys Electronics Desktop can use Microsoft® Visual Basic® Scripting (VBScript) to record macros. VBScript is based on the Microsoft Visual Basic programming language.

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.

Simple and Composite Names

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

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

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

VBScript Variables

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

Declaring Variables

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

```
Dim box_xsize
```

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

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

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

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

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

Declaring Variables in Python

Python does not require you to declare variables before you assign a value to them.

You can directly assign information to it. For example:

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

Variable Naming Conventions

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

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

Scope and Lifetime of Variables

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

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

Array Variables

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

```
Dim Primitives(2)
```

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

```
Primitives(0) = "Box1"

Primitives(1) = "Cone1"
```

```
Primitives(2) = "Cylinder1"
```

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

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

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

```
Dim Primitives

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

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

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

```
Dim FirstArray()

ReDim SecondArray()
```

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

```
ReDim FirstArray(25)

ReDim Preserve FirstArray(30)
```

VBScript Operators

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

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

Operator Precedence

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

When an expression contains operators from more than one category, they are evaluated in the following order:

1. Arithmetic Operators
2. String Concatenation Operator (&)
3. Comparison Operators
4. Logical Operators

Arithmetic operators within a single expression are evaluated in the following order of precedence:

1. Exponentiation (^)
2. Multiplication and Division (*,/): These two operators are of equal precedence and are evaluated in the left-to-right order in which they appear within the expression.
3. Integer Division (\)
4. Modulus Arithmetic (Mod)
5. Addition and Subtraction (+,-): These two operators are of equal precedence and are evaluated in the order in which they appear within the expression.

If the same arithmetic operator appears multiple times within a single expression, they are evaluated in the left-to-right order in which they appear.

Comparison operators all have equal precedence and are evaluated in the left-to-right order in which they appear within the expression.

Logical operators all have equal precedence and are evaluated in the left-to-right order in which they appear within the expression.

Arithmetic Operators

Following is a list of VBScript's arithmetic operators:

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

> **Note:**
>
> For the order of precedence for these operators, see Operator Precedence.

String Concatenation Operator

The ampersand symbol (**&**) is used to perform string concatenation. This operator appends the second string to the first string.

Example:

```
"Hello," & " isn't it a lovely day?"
```

produces the following resultant string:

```
"Hello, isn't it a lovely day?"
```

Comparison Operators

Following is a list of VBScript's comparison operators:

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

> **Note:**
>
> All comparison operators have the same precedence. When multiple comparisons exist in a single expression, evaluate them in the left-to-right order in which they appear.

Logical Operators

Following is a list of VBScript's logical operators:

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

> **Note:**
>
> All logical operators have the same precedence. When multiple logical operators exist in a single expression, evaluate them in the left-to-right order in which they appear.

Controlling Program Execution

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

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

Using If...Then...Else

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

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

Using Select Case

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

```
Select Case primitive_name
  Case "Box1"
```

<statements to execute>

```
  Case "Cylinder1"
```

<statements to execute>

```
  Case Else
```

<statements to execute>

```
End Select
```

Looping Through Code

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

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

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

Using a For...Next Loop

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

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

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

Using a Do Loop

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

Repeating Statements While a Condition is True

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

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

Repeating a Statement Until a Condition Becomes True

Following is the syntax:

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

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

VBScript Procedures

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

Function Procedures

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

```
Function FunctionName([arguments])

  <Function statements>

End Function
```

Sub Procedures

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

```
Sub ProcedureName([arguments])

  <Procedure statements>

  End Sub
```

Converting Between Data Types

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

CStr	Syntax: CStr(variablename). Converts variablename to a string. For example, it can be used to convert the number 2.5 to the string "2.5".
CBool	Syntax: CBool(variablename). Converts variablename to a boolean. If variablename is 0 or "0", CBool returns False. Otherwise it returns True.
CDbl	Syntax: CDbl(variablename). Converts variablename to a double precision number. For example, it can be used to convert the string "2.5" to the number 2.5.
CInt	Syntax: CInt(variablename). Converts variablename to an integer.

Including Scripts

You can include one script within another using the following command:

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

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

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

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

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

Aborting Scripts

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

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

Interacting with a Script

VBScript provides two functions that enable you to interact with a script while it is running:

the InputBox function and the MsgBox function.

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

```
Dim users_string

users_string = InputBox ("text prompt", "title of the pop-up dialog _

  box", "default text for the input box")
```

The last two arguments to the function are optional.

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

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

Recommended VBScript References

Microsoft Corporation. *VBScript User's Guide.*

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

Childs, M., Lomax, P., and Petrusha, R. *VBScript in a Nutshell: A Desktop Quick Reference*.

May 2002. O'Reilly & Associates. ISBN: 1-56592-720-6.

Introduction to IronPython

IronPython is an implementation of the Python programming language targeting the .NET runtime. What this means in practical terms is that IronPython uses the Python programming language syntax and standard python libraries and can additionally use **.**NET classes and objects to give one the best of both worlds. This usage of .NET classes is fairly seamless in that a class defined in a .NET assembly can be used as a base class of a python class.

Scope

Functioning as a tutorial on Python or IronPython is way out of the scope of this document. There are several excellent resources online that do a very good job in that regard. This document only attempts to provide a limited introduction to IronPython as used to script Ansys EM products.

This document is also not a tutorial on the scripting of Ansys EM products. It complements the existing scripting guide (available from a product's Help menu) and provides a pythonic interpretation of that information. The reader might have to refer to either the scripting guide or recorded samples of VBScript to follow some of the sections.

Python compatibility

The version of IronPython in use is **2.7** and built on the .NET framework version 4.0: this version targets **Python 2.7** language compatibility. While most python files will execute under IronPython with no changes, python libraries that make use of extensions written in the C programming language (NumPy or SciPy for instance), are not expected to work under IronPython. In such cases, it might be possible to locate .NET implementation of such libraries or explore the use of IronClad.

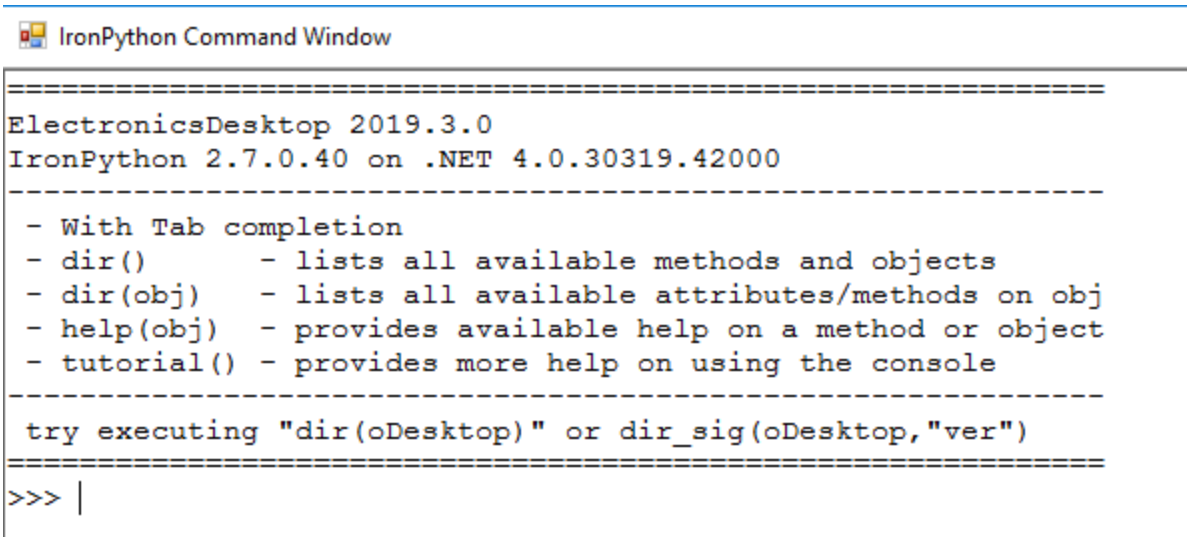
(http://code.google.com/p/ironclad/).

Advantages of IronPython

The advantages that IronPython use provides are significant:

- Python has a large eco-system with plenty of supporting libraries, Visual IDEs and debuggers. It is actively developed and enhanced.
- IronPython, in addition, has access to the entire .NET eco system. This allows us, for instance, to create a modern GUI using the **System.Windows.Forms** assembly from IronPython code and call any other .NET assembly for that matter.
- The use of IronPython's technologies enables the ability to interactively script Desktop (feature in development). This allows better discovery of the scripting APIs as well as directly programming to the scripting API in python, a language more tractable and platform independent compared with VBScript.
- The Python syntax of dictionaries is somewhat easier to read and write when supplying arguments to the scripting methods.

This document describes IronPython briefly and then goes on to describe the desktop provided IronPython scripting console and scripting with IronPython. You can open an IronPython Command Window by clicking **Tools** > **Open Command Window**.



The document assumes that you know how desktop scripting works using VBScript or Javascript.

Introduction to IronPython

IronPython Mini Cookbook

Translating Script Commands from VBScript to IronPython

Scripting Using Iron Python

Standalone IronPython and Desktop IronPython

IronPython Examples

Creating User Defined Primitives and User Defined Models in Python Scripts

IronPython Mini Cookbook

This topic presents simple counterparts between IronPython and VBScript. It does not provide a full tutorial on IronPython syntax. Because IronPython is a Python implementation, you can consult Python documentation for additional information.

Comments

VBScript	IronPython
Comments start with a single quote: ' comment	Comments start with a hash: # comment

Assigning/Creating Variables

VBScript	IronPython
Declare with a Dim: Dim doc Assignment then needs a Set instruction: Set doc = app.GetActiveProject()	No Set syntax. Simply create and assign: doc = app.GetActiveProject()

Create Lists/Arrays

VBScript	IronPython
Declare as array of String with 11 indices, from 0 through 10: Dim myArray(0 to 10) as String myArray(0) = "Hello" myArray(1) = "bye" Declare an array with no size: Dim array2() as String Re-dimension an array once size is known: ReDim array2(0 to 2) as String array2(0) = "this" array2(1) = "also"	Declare an empty array: myEmptyArray = [] Declare an array and initialize it with 5 ints: myInitedArray = [1, 2, 3, 4, 5] Python lists can have items of any type and there is no pre-declaration. Declare an array and init with mixed types: mixed = ["hello", 1 ,2 ["nested"]] Append to an array: mixed.append(3.5)

Create Dictionaries/Maps

VBScript	IronPython

Declare with a Dim:
Dim dict
Use the CreateObject function with ProgID Scripting.Dictionary:
Set dict = CreateObject _
("Scripting.Dictionary")
Add items using the object, key, item syntax:
dObject.Add key, item
	An IronPython dictionary is a collection of name value pairs. Just like arrays, there is no restriction on the keys or the values. For purposes of Ansys EM scripting, however, all keys must be strings
	Delimiters are curly braces. Use a colon between the key and the value. Separate key value pairs with a comma:
	myDict = {
	"a" : 1,
	"b" : "hello there",
	"c" : [1, 2, "abc"]
	}

Boolean Values

VBScript	IronPython
Boolean literals are in lower case: true false	The first letter is capitalized: True False

Converting Numbers to Strings and Vice Versa

VBScript	IronPython
Use **CInt, CDbl, CBool, CLng** to convert the string representation to the number representation. Use IsNumber to check before conversion: Dim nStr = "100" Dim n = CInt(nStr) Use CStr to convert a number to its string representation: Dim v, vStr v = 100 vStr = CStr(v)	Use integer() or float() or double() functions to cast a string CONTAINING the string representation of whatever you are casting to: strInt = "3" intVal = int(strVal) floatVal = float(strVal) Invoke the str() function with the int/float values as needed. You can alternately use the string formatting method listed below: strVal = str(42) strVal = str(42.345)

String Formatting/Concatenation

VBScript	IronPython
String concatenation uses the & operator: Dim allStr, str1 str1 = " how are you" allStr = "Hello " & " There" & str1 There seems to be no direct string formatting function in VBScript. Ssing string concatenation or using Replace are the two built-in options: Dim fmt = "{1} climbs stalk {2}" Dim str = Replace(fmt, "{1}", "jack") str = Replace(str, "{2"}, 10)	If you have two strings, you can always concatenate them using the '+' operator: str1 = "hello" str2 = "world" str12 = str1 + " " + str2 If you have different types (for instance a string and an int), you must use the string formatting commands. When formatting multiple arguments, they must be entered as a tuple (item1, item2,): num = 10 str3 = "%s climbs stalk %d" % ("jack", num) str4 = "%d stalks" % num

Looping over Lists

VBScript	IronPython

Dim myArray(0 to 2) as String
myArray(0) = "alpha"
myArray(1) = "bravo"
myArray(2) = "charlie"
 	
For Each i in myArray
Print i
Next
	vals = [1, 3, 3.456]
	
	def process(val):
	return 2*val
	
	for i in vals:
	  print i
	  print " -> " process(i)

Looping over a Range

VBScript	IronPython
To loop over a range, specify start, end, and step: For i = 0 To 10 Step 1 Print i Next	for i in range(0, 10): print i

Related Topics:

Indentation in IronPython

Methods in IronPython

Introduction to IronPython

Translating Script commands from VBScript to IronPython

Scripting Using Iron Python

IronPython Samples

Indentation in IronPython

Python is a language where white space (spaces and tabs) is syntactically significant. You must understand the basics of indentation before scripting in python.

Any statement that introduces a block of code should be written so that every line of the block has the same indent (leading spaces or tabs) and the indent should be at least one more than the indent of the introducing statement.

> **Note:**
>
> Python recommends the use of spaces over tabs.

Indenting Functions

Define a function that starts at 0 indentation:

```
def multInt(a,b):
```

Every line following `def multInt` that is expected to be a part of the function, must be indented to line up with the function.

```
def multInt(a,b):
  return a
```

Indenting If Conditions

Each line that belongs to the body of this function should have an indent that is more than the indent used by the if statement.

```
def multInt(a,b):
If a%2 == 0:
    return (a * b) + 100
  else:
    return (a * b) + 1000
```

Methods in IronPython

Finding Methods

To list all methods available in the string module,import the module:

```
import string
```

Then get the directory listing:

```
dir(string)
```

This returns a list of all the methods available (as well as some __somename__ internal names that can be ignored).

Help

Once you know a function name, you can get more help on it using the built-in `help` method.

Related Topics

Introduction to IronPython

Translating Script commands from VBScript to IronPython

Scripting Using Iron Python: Putting it all Together

IronPython Samples

Translating Script Commands from VBScript to IronPython

This topic briefly describes scripting methods and arguments via VBScript samples. The distinctions made here are significant and useful when translating scripts written in VBScript to IronPython.

Related Topics

Script Method Argument

VBscript Method Call Types

Converting VBScript Function calls to IronPython Syntax

Introduction to IronPython

IronPython Mini Cookbook

Scripting Using Iron Python

IronPython Samples

Script Method Argument

Script method calls in VBscript generally take the form:

```
objectName .methodName ( arg1, arg2, ...)
```

The function call syntax is a standard followed by several programming languages. However, the argument types in VBScript objects used for product scripting are restricted to the following:

- Primitive Types
- Named Arrays
- Named Functions

Primitive Types

Primitive types are the standard `bool`, `int`, `float`, `double`, and `string`.

Named Arrays

Named arrays are a special construct used very commonly and can be found in many recorded script samples.

A named array begins with **Array("NAME:someName"** followed by a collection of comma separated values which can be:

- Primitive values
- Arrays of primitive values
- Additional named arrays
- Keys, in the form **"keyName:="** followed by a primitive value or function

Named Functions

Named functions are arrays which start with **Array(** and *do not* have a leading "NAME:name" item. **They are always introduced by a key** and can contain comma separated values of the following type:

- A primitive value
- A key (of the form **"keyName:="**) followed by
  - A primitive value
  - Another function (nested function)

Related Topics

Translating Script commands from VBScript to IronPython

VBScript Method Call Types

VBScript method calls fall into two categories and the distinction between the two results in syntax differences. These syntax differences are significant when converting VBScript to IronPython.

VBScript Functions

In VBScript terminology, functions return values. The syntax for this is one shared with practically all programming languages:

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

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

> **Note:**
>
> If there are arguments, the method name is *always* followed by an argument list enclosed in parentheses. If the argument list is empty, as shown above for the *NewProject* call, the parentheses can be omitted.

VBScript Sub-Routines

VBScript subroutines are those that do not have any return value. VBScript allows these to be written without any parentheses even if they have a non-empty argument list.

```
oModule.CreateReport "XY Plot1", "Standard", "XY Plot", "optimtee :
optimtee", _
Array("Domain:=", "Sweep"), Array("Freq:=", Array("All"), "off-
set:=",_
Array("0uin")), Array("X Component:=", "Freq", "Y Component:=", _
Array("dB20(S(1,1))", "dB20(S(1,2))", "dB20(S(1,3))", _
 "dB20(S(2,1))", "dB20(S(2,2))", "dB20(S(2,3))", "dB20(S(3,1))",
 "dB20(S(3,2))", "dB20(S(3,3))")), Array()
```

Related Topics

Translating Script commands from VBScript to IronPython

Converting VBScript Function calls to IronPython Syntax

When used for scripting, IronPython function names are always followed by parentheses.

So:

- If you see a VBScript snippet that looks like a VBScript subroutine, remember to add parentheses.
- If you see a VBScript function that has no arguments and no parentheses, remember to add them around an empty argument list.

The parentheses change is the only one to keep in mind when converting VBScript function calls syntax to IronPython.

Return Values

VBscript return values are sometimes assigned via the Set declaration. IronPython return values are simple assignment (See: Iron Python Mini Cookbook).

Primitive Method Arguments

Replace each VBScript primitive with an equivalent IronPython primitive.

Boolean values in IronPython have their first letter capitalized (`True` instead of `true` and `False` instead of `false`).

For arrays, the recommended approach is to simply replace a VBScript array with a Python array. The mapping is simple:

- Change `Array(` to a square bracket `[` and close with another square bracket `]` instead of a parenthesis `)`
- Remove the line continuation underscore symbol: _
- Map Boolean values correctly

Named Array Arguments

Formatting helps readability immensely but is not required.

All that *must* be done is:

- Add the parentheses, since the VBScript subroutine omits them
- Replace the Array( ) delimiters with [ ]
- Remove the Char(34) function (which introduced a double quote) and replace it with the escaped double quote literal: \"
- Replace true with True
- Remove the line continuation symbol: _

Named Array Values with All Key Value Pairs

While it is generally not allowed to replace arrays and nested arrays with Python dictionaries, in the case where the named array consists entirely of key value pairs, you can use a dictionary and avoid typing the trailing := symbols after the keys. This further aids readability of the script.

- If all key value pairs
- Remove the trailing := after each key
- Replace the , after the key with a :
- If the named array is the top level argument, ensure that the NAME:name is present and is split into NAME : name as a key value pair
- Enclose the converted array in a { } pair to declare the dictionary.

Named Arrays with Nested Named Arrays

- Split the NAME:name field into a key value pair
- Translate array key value pair to a dictionary key value pair.
- Create a new key with the name of the nested array and keep the nested array (as an array or as a dictionary) as its value. If the nested array is being retained as an array, the NAME:name field should be retained in the array. If the nested array is being converted to a dictionary, the name is optional: if also retained in the nested array, it must match the outer key.

["NAME:name",

"key1:=" , 1,

"key2:=" , 2,

["NAME:name2", "R:=", 255]

]

Sample Script: Named array with nested named array in array syntax

The above named array with a nested named array (after conversion to IronPython as named array) can be converted to a dictionary as well. The dictionary can take any of the following forms

{ "NAME" : "name",

"key1" : 1,

"key2" : 2,

"name2" : ["NAME:name2", "R:=", 255]

}

Sample Script: Named array with nested named array as mixed dictionary + array

{ "NAME" : "name",

"key1" : 1,

"key2" : 2,

"name2" : {"R" : 255}

}

Sample Script: Named array with nested named array in all dictionary syntax

{ "NAME" : "name",

"key1" : 1,

"key2" : 2,

"name2" : {

"NAME" : "name2",

"R" : 255

}

}

Function Blocks

Function blocks in VBScript argument syntax are represented as arrays without the "NAME:.." field. However, functions are always introduced by a key in a parent structure. Function blocks

can therefore never exist as a top-level argument. They are only found as the value pairs inside a named array or inside another function block.

> **Important:**
>
> Function blocks and their items cannot be converted to dictionaries even though they might be composed entirely of key value pairs.

The reason for this is the need to main the user-entered order. Every item in a function block is expect to be transmitted to the script method in exactly the same order as typed out and this is impossible to achieve when a dictionary is used (as the keys get reordered according to the dictionary's internal tree/key sorting scheme).

When you see a function block, simply replace the Array( ) delimiters with python array delimiters [ ]

Scripting Using Iron Python

If you have existing VBScript/Javascript scripts use existing scripts them as much as possible by either embedding the test into the IronPython script or invoking them.

Translating a script in VBScript to IronPython

Read the chapter on translation and study the samples in that chapter as well as those in the appendix. For python syntax and the differences, the mini-cookbook chapter will also be useful.

Writing an IronPython script from scratch

Read through the scripting guide available from the product's help menu and translate the VBScript methods described to IronPython using the information provided in the chapter on translation. Studying the samples in the document will also prove helpful.

For python syntax and the differences, the mini-cookbook chapter will also be useful.

IronPython Script Execution Environment

Scripting using Embedded VBScript or JavaScript

Scripting with IronPython

Standalone IronPython and Desktop IronPython

Related Topics

Introduction to IronPython

IronPython Mini-cookbook

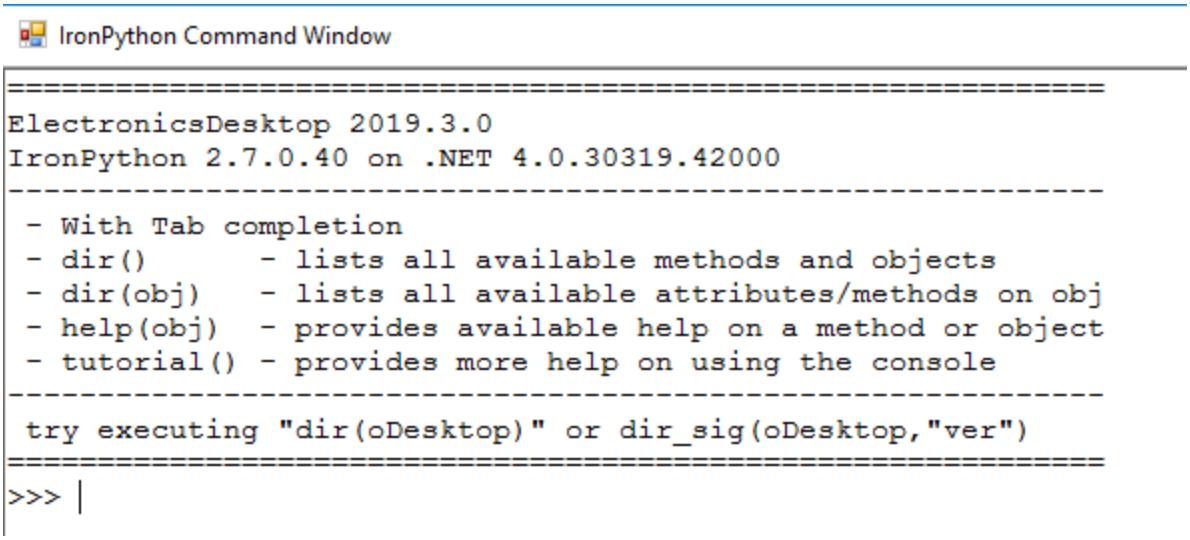
Translating Script commands from VBScript to IronPython

Appendix: IronPython Samples

IronPython Script Execution Environment

Scripts written in IronPython are executed by desktop in four different ways:

- **Tools** > **Open Command Window**, to open the **IronPython Command Window:**



- **Tools** > **Run Script** menu item, select "IronPython" from the file type drop-down list.
- Launch the product with a script argument.
- Register an IronPython script as an external tool using the **Tools** > **External Tools** menu item.

When desktop executes a script, it does so in an execution environment setup with predefined variables and functions. These predefined variables and functions are how the script communicates with the desktop, and they come in four flavors addressed in the following subtopics:

Script Argument for IronPython

Script Argument for IronPython

When scripts are launched using the **Tools > Run Script** menu item, the dialog that pops up allows the user to specify arguments.

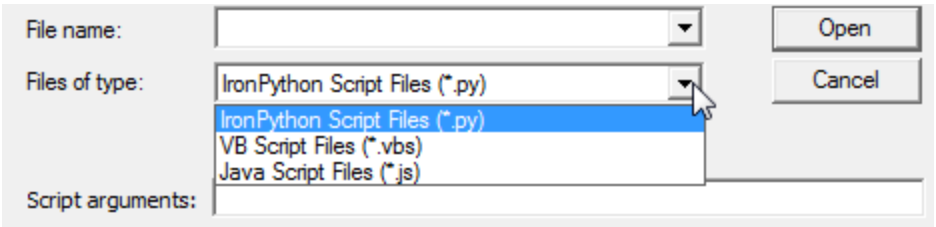


Figure 1: Run Script dialog and script arguments

Any argument specified here is communicated to the script being executed as the predefined variable **ScriptArgument**.

Related Topics

IronPython Script Execution Environment

Scripting using Embedded VBScript or JavaScript

Since script recording is still done in VBScript and users are expected to have a significant collection of VBScript or JavaScript assets, it is useful to continue to use existing script files and snippets even when scripting in IronPython. The various **Run<*>Command** methods have been designed for this purpose.

For instance: one can create a parameterized cone in HFSS by executing the following IronPython script from the **Tools** > **Run Script** menu.

```
# assign the VBScript snippet obtained from a script recording from
HFSS to

# coneScript and replace the BottomRadius recorded value with botRa-
dius

coneScript = """Dim oAnsoftApp

Dim oDesktop

Dim oProject

Dim oDesign

Dim oEditor

Dim oModule

Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")

Set oDesktop = oAnsoftApp.GetAppDesktop()

oDesktop.RestoreWindow

Set oProject = oDesktop.GetActiveProject()

oProject.InsertDesign "HFSS", "HFSSPyTestDesign", "DrivenModal", ""

Set oDesign = oProject.SetActiveDesign("HFSSPyTestDesign")

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

oEditor.CreateCone Array("NAME:ConeParameters", _

 "XCenter:=", "0mm", "YCenter:=", "0mm", "ZCenter:=", "0mm",_

 "WhichAxis:=", "Z", "Height:=", "2mm", _
```

```
 "BottomRadius:=", "3mm", _
 "TopRadius:=", "0mm"), Array("NAME:Attributes", "Name:=", _
 "Cone1", "Flags:=", "", "Color:=", "(132 132 193)", "Trans-
parency:=", 0, _
 "PartCoordinateSystem:=", "Global", "UDMId:=", "", "Mater-
ialValue:=", _
 "" & Chr(34) & "vacuum" & Chr(34) & "", "SolveInside:=", _
 true)
"""

SetScriptingLanguageToVBScript()
RunScriptCommand(coneScript)
```

Sample Script 11: Hybrid VBScript + IronPython scripting: parameterized Cone Creation

Even though recorded VBScript is used for scripting, the incremental functionality that is provided using IronPython is the ability to write a GUI using IronPython/.NET, collect information from the user and then modify or generate the VBScript commands to actually script the Ansys EM desktop. This GUI functionality is cross platform and a significant positive. The following example demonstrates a contrived use of a .NET window form to display the argument supplied to the IronPython script (via the ***ScriptArgument*** variable).

```
#import the CLR references
import clr
clr.AddReference("System.Windows.Forms")

from System.Windows.Forms import Application, Form, Label, Button,
DockStyle

# the GUI form to show some text
# the class below derives from From (System.Windows.Forms.Form)
# imported above from the .NET assembly.
class ShowPropertiesForm(Form):
  def __init__(self, name, text):
    self.Name = name
    self._label = Label()
```

```
    self._label.Text = text
    self._label.Dock = DockStyle.Fill

    _button = Button()
    _button.Text = "Close"
    _button.Dock = DockStyle.Bottom
    _button.Click += self._buttonPressed

    self.Controls.Add(self._label)
    self.Controls.Add(_button)

  def _buttonPressed(self,sender,args):
    self.Close()

#------------------------------------------------------
# Main script code
#------------------------------------------------------
#display the ScriptArgument variable as the text label
# in the form.
gui = ShowPropertiesForm("Sample Form", ScriptArgument)

# This makes it a modal dialog.
gui.ShowDialog()

# the following will make it a non-modal dialog
#Application.Run(gui)
```

Sample Script 12: Demonstrates the use of a .NET form from IronPython

While creating cross platform user interfaces from scripts is one of the main motivations driving the adoption of IronPython, any .NET assembly can be used with the caveat that Linux use requires Mono compatibility of any used assemblies.

While this hybrid approach is useful when you have existing VBScript commands that you want to reuse or when you want to quickly parameterize a recorded sample, the one significant limitation of this approach is the inability to capture return values from VBScript or JavaScript calls that do return something. Full two way communication with the product requires the use of pure IronPython to directly invoke the script objects as described below.

Related Topics

IronPython Script Execution Environment

Scripting with IronPython

While this section talks about directly interacting with the script objects, note that you can execute VBScript or Javascript at any point using any of the available Run*Command functions. using your existing script assets in this fashion and mixing with IronPython code for new functionality as needed is a viable and option.

Access to the application scripting objects is provided via the predefined **oDesktop** object (as listed in Script Objects). Interacting with the script objects is very natural, method calls are made just like in VBScript except that the argument syntax is somewhat simplified to follow natural Python syntax. All primitive types (string, integer, double) map to the natural primitive types in python. The only differences from the VBScript syntax are seen when specifying array type arguments. The differences are described in earlier chapters.

> **Note:**
>
> The typical VBScript calls to obtain the registered COM scripting interface via CreateObject calls and then obtain the oDesktop object from it using the GetAppDesktop() is not needed (or even supported on all platforms). Since all scripting occurs in the context of a running workbench, the available Desktop object is always provided and expected to be used directly.

Scripting using the IronPython scripting API is very much like scripting with VBScript except that

- Any argument is supplied via the built in **ScriptArgument** variable
- The **oDesktop** object is always available
- The scripting method names are identical to the ones used with VBScript
- Method calls, while the name is the same have to adhere to the rule of ensuring trailing parentheses irrespective of whether the function returns anything or has any arguments.
- Any compound/block arguments should be translated to the appropriate IronPython array or dictionary syntax.

The samples section lists a collection of pure IronPython snippets: these, along with the various script snippets listed in this document should serve as a guide and reference.

Related Topics

IronPython Script Execution Environment

Standalone IronPython and Desktop IronPython

Standalone IronPython

In general, it is easier to run a script directly from Electronics Desktop. Standalone IronPython does not implement all the functionality available when a script is run from Electronics Desktop. It only implements full support for COM functions.

Running Standalone IronPython

Standalone IronPython uses COM to get the handle to the AnsysEDT app. To run standalone IronPython, you'll need to call the IronPython interpreter ipy64.exe.

It is located in:

```
\\<AnsysEDTInstallationPath>\common\IronPython\ipy64.exe
```

For example, to run myScript.py, type the following in the command line:

```
"C:\Program Files\AnsysEM\v242\Win64\common\IronPython\ipy64.exe"
"<filePath>\myScript.py"
```

You can set the interpreter to be the default program when double-clicking the .py script. You can use any recorded script as the basis for a standalone script and simply add an installation-internal path to the python module search path (as shown below) and end the script with a new shutdown call.

Using a Recorded Script

A python script recorded in AnsysEDT already has the required lines to be run as a standalone, except for the first two lines (path settings) and the final `Shutdown()` call. See the example script below.

Creating an External Script

When creating a script outside of Electronics Desktop, the following lines should be included at the beginning of your script:

- `import sys`

  # Imports the sys module containing system-specific functions native to IronPython.

- `sys.path.append("<InstallationPath>")`

  # Adds the Electronics Desktop installation path to the list of directories Python searches for modules and files.

- `sys.path.append("<InstallationPath>/PythonFiles/DesktopPlugin")`

  # Adds the PythonFiles/DesktopPlugin subfolder to the list of directories Python searches for modules and files.

- `import ScriptEnv`

  # This imports ScriptEnv.py from the installation path specified above. ScriptEnv.py performs an operating system check and defines functions used in Electronics Desktop scripts. See the annotations in the ScriptEnv.py file for more information.

- `ScriptEnv.Initialize("Ansoft.ElectronicsDesktop")`

  or `ScriptEnv.InitializeNew(NonGraphical=True)`

  # `Initialize` and `InitializeNew` are functions within ScriptEnv.py. The first option launches Electronics Desktop. The second allows you to run a script without launching Electronics Desktop. See the annotations in the ScriptEnv.py file for more information.

You must end the script with:

- `ScriptEnv.Shutdown()`

  # This stops ScriptEnv.py. If you are running multiple scripts, include this only at the end of the last script.

Example Script

```
import sys

sys.path.append(r"C:\Program Files\AnsysEM\v242\Win64")

sys.path.append(r"C:\Program Files\An-
sysEM\v242\Win64\PythonFiles\DesktopPlugin")


import ScriptEnv

ScriptEnv.Initialize("Ansoft.ElectronicsDesktop")

oDesktop.RestoreWindow()

oProject = oDesktop.NewProject()

oProject.InsertDesign("HFSS", "HFSSDesign1", "DrivenModal", "")

oDesign = oProject.SetActiveDesign("HFSSDesign1")

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

oEditor.CreateRectangle(

[

  "NAME:RectangleParameters",

  "IsCovered:= ", True,

  "XStart:= ", "-0.2mm",
```

```
  "YStart:= ", "-3mm",
  "ZStart:= ", "0mm",
  "Width:= ", "0.8mm",
  "Height:= ", "1.2mm",
  "WhichAxis:= ", "Z"
],
[
  "NAME:Attributes",
  "Name:= ", "Rectangle1",
  "Flags:= ", "",
  "Color:= ", "(132 132 193)",
  "Transparency:= ", 0,
  "PartCoordinateSystem:=", "Global",
  "UDMId:= ", "",
  "MaterialValue:= ", "\"vacuum\"",
  "SolveInside:= ", True
])
oDesign.SetDesignSettings(['NAME:Design Settings Data', 'Allow Mater-
ial Override:=', True, 'Calculate Lossy Dielectrics:=', True])
oEditor.SetModelUnits(['NAME:Units Parameter', 'Units:=', 'mil', 'Res-
cale:=' , False ])
ScriptEnv.Shutdown()
```

IronPython Samples

Change property

The following snippets show how a change property command (in this case, to change the color of a cone) looks in VBScript and its two possible IronPython variants.

```
oEditor.ChangeProperty Array("NAME:AllTabs", Array("NAME:Geo-
metry3DAttributeTab",_
  Array("NAME:PropServers", "Cone1"), _
  Array("NAME:ChangedProps", _
  Array("NAME:Color", "R:=", 255, "G:=", 255, "B:=", 0))))
```

Sample Script 13: ChangeProperty command to change color of a cone in VBScript

```
oEditor.ChangeProperty(
["NAME:AllTabs",
  ["NAME:Geometry3DAttributeTab",
  ["NAME:PropServers", "Cone1"],
  ["NAME:ChangedProps",
  ["NAME:Color", "R:=", 0, "G:=", 0, "B:=", 64]
  ]
]
    ])
```

Sample Script 14: ChangeProperty command to change color of cone using Python arrays

Any time there are named arrays composed purely of key-value pairs, they can always be represented using a Python dictionary, irrespective of the nesting of said named array.

```
oEditor.ChangeProperty(
  ["NAME:AllTabs",
  ["NAME:Geometry3DAttributeTab",
    ["NAME:PropServers", "Cone1"],
    ["NAME:ChangedProps",
    {
    "NAME":"Color",
    "R" : 0,
    "G" : 64,
    "B" : 0
  }]]
])
```

Sample Script 15: ChangeProperty command to change the color of a cone using Python arrays and dictionaries

Create a Cone using IronPython

Most scripting tasks using IronPython are expected to be formatted as the following example. One starts with the predefined **oDesktop** object and drills down to the design, editors, modules

etc and issues any required commands on the object while formatting the script command arguments in natural python syntax.

```
oProject = oDesktop.GetActiveProject()
oDesign = oProject.InsertDesign("HFSS","Random","DrivenModal","")
oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.CreateCone(
{
  "NAME" : "ConeParameters",
  "XCenter" : "0mm",
  "YCenter" : "0mm",
  "ZCenter" : "0mm",
  "WhichAxis" : "Z",
  "Height" : "2mm",
  "BottomRadius" : "1.56204993518133mm",
  "TopRadius" : "0mm"
  },
  {
  "NAME" : "Attributes",
  "Name" : "Cone1",
  "Flags" : "",
  "Color" : "(132 132 193)",
  "Transparency" : 0,
  "PartCoordinateSystem": "Global",
  "UDMId" : "",
  "MaterialValue" : "\"vacuum\"",
  "SolveInside" : True
  }
  )
```

Sample Script 16: IronPython script to create a cone

Create geometry and then create a grid from it using copy/paste/move

The following script demonstrates slightly more advanced use of scripting and the use of return values from script methods. It creates a 5x5 grid of cones and also demonstrates the adding of information messages to the application's message window.

```
oProject = oDesktop.GetActiveProject()

oDesign = oProject.InsertDesign("HFSS","Hersheys Kis-
ses","DrivenModal","")

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


# create the first cone

AddInfoMessage("Creating first cone")

firstConeName = "firstCone"

coneBotRad = "1.5mm"

oEditor.CreateCone(

  {

  "NAME" : "ConeParameters",

  "XCenter" : "0mm",

  "YCenter" : "0mm",

  "ZCenter" : "0mm",

  "WhichAxis" : "Z",

  "Height" : "2mm",

  "BottomRadius": coneBotRad,

  "TopRadius" : "0mm"

  },

  {

  "NAME" : "Attributes",

  "Name" : firstConeName,

  "Flags" : "",

  "Color" : "(132 132 193)",

  "Transparency" : 0,

  "PartCoordinateSystem": "Global",

  "UDMId" : "",
```

```
  "MaterialValue" : "\"vacuum\"",
  "SolveInside" : True
  }
  )

# Now replicate this a few times and create an array out of it
AddInfoMessage("Replicating it 24 times")
for x in range(5):
  for y in range(5):
       # leave the first one alone in it's created
       # position
       if x == 0 and y == 0:
         continue

# all other grid positions, replicate from the
# first one

# copy first
oEditor.Copy(
  {
  "NAME" : "Selections",
  "Selections" : firstConeName
  }
)

# paste it and capture the pasted name
# the pasted names come in an array as we could
# be pasting a selection cmposed of multiple objects
pasteName = oEditor.Paste()[0]
```

```
# now move the pasted item to it's final position
oEditor.Move(
   {
   "NAME" : "Selections",
   "Selections" : pasteName
   },
   {
   "NAME" : "TransalateParameters",
   "CoordinateSystemID" : -1,
   "TranslateVectorX" : "%d * 3 * %s" % (x, coneBotRad),
   "TranslateVectorY" : "%d * 3 * %s" % (y, coneBotRad),
   "TranslateVectorZ" : "0mm"
   }
)

# Now fit the display to the created grid
oEditor.FitAll()
```

Sample Script 17: Sample script to create a cone and then use copy/paste/move to replicate it.

Related Topics

Introduction to IronPython

IronPython Mini-cookbook

Translating Script commands from VBScript to IronPython

Scripting Using Iron Python: Putting it all Together

Creating User Defined Primitives and User Defined Models in Python Scripts

You can create User Defined Primitives and User Defined Models in Python scripts (based on the IronPython implementation).

Advantages Compared to C++

- No need to create and build project; all you need to do is create a Python script
- Python script is platform independent

- Scripts can inherit functionality from existing scripts
- Garbage collector - no need to free memory
- Easy debugging

Changes compared to C

Though methods, constants and structures are kept as close to the C implementation as possible, some changes had to be made to make code Python-compatible.

Structures

- Structures have the same names as in C implementation.
- Structures fields names are capitalized.
- Arrays in structures become lists in Python (Technically a.NET IList container)
- Structure instances are created using the supplied constructors and members are accessed using the provided access methods.

For a complete list of structures and examples please see UDP/UDM Structures.

Return Values for UDM and UDP Functions

For information on return values for each UDM and UDP function, see the Return Values section.

Constants

Enumeration/Enum constants have almost the same names as in C but the enum must be qualified by the type. Additionally, redundant "UDP", "UDM" or type prefixes have been removed. This allows for better human-readability.

```
# Example of specifying the LengthUnit enum by qualifying it

# with the type of the enum: UnitType

unitType = UnitType.LengthUnit
```

For a complete list of enum constants please see UDP/UDM Constants.

Methods

Methods are described in IUDPExtension methods, IUDMExtension methods, and UDMFunctionLibrary listed further in this document. A separate chapter includes a UDP IronPython example of fillet and chamfer.

The main differences in functions parameters (from C implementation):

- functions names in UDPFunctionLibrary and UDMFunctionLibrary are capitalized
- arrays become a python list of objects
- `void * callback parameter` is dropped from the parameter list
- output parameters (pointer types that are filled during the function call) usually become

return values

- 'list size' parameter usually will be omitted as redundant

Output Parameters

The rule for the output parameters is as follows:

- If the function has one output parameter variable and no return value, the variable will become function's return value.The same will happen if the return value is a 'success/failure' boolean ('None' will be returned on failure and parameter variable - on success).
- If the function has one output parameter and a return value, the function will return a Python tuple where function return value will be the first one in the tuple.
- If there is more than one out variable, the function will return a Python tuple with all output parameters in the specified order. If function has a return value, it must always be the first in the tuple.

```
# one output parameter; return value is ignored

udmDefinition = udmFunctionLibrary.GetDefinition()


# one output parameter; return value must be preserved. return
# and output values are packed into the return tupe, in order

(lRet, partIdsList) = udpFunctionLibrary.DetachFaces(nPartIds, faceId-
sList)

# Two output parameter; return value must be preserved

# the return tuple is (returnVal, output1, output2)

(bRet, udpPositionLow, udpPositionHigh) = udmFunc-
tionLibrary.GetBoundingBox(partId,exact);
```

Comparison with C function:

C	Python
bool getDefinition(UDMDefinition* udmDefinition, void* callbackData); where udmDefinition is an output parameter	udmDefinition = udmFunctionLibrary.GetDefinition() (Note: callbackData is omitted in py interface)
long detachlFaces(int nFacesAndPartIds,	(bRet, partIds) = udmFunctionLibrary.DetachlFaces (nFacesAndPartIds, faceIds)

C	Python
long* faceIds, long* partIds, void* callbackData); where partIds is an output parameter	(Note: callbackData is omitted in py interface)

'List Size' Parameters

The rule for the 'list size' is as follows:

- If function has input 'List' parameter and input 'list size' parameter, 'list size' parameter will be omitted.
- If function has output 'List' parameter and output 'list size' parameter, 'list size' parameter will be omitted.
- If function has output 'List' parameter and input 'list size' parameter, 'list size' parameter won't be omitted as it's needed for memory allocation in the corresponding C++ function from the UDP/UDM function library.

Example:

```
# input list, input list size

lret = udpFunctionLibrary.Unite(objectIds)


# output list, output list size

faceIdList = udmFunctionLibrary.GetAllFaces(PartId)


# output list, input list size

(lret, partIdList) = udpFunctionLibrary.DetachFaces(listSize,
faceIdList)
```

Comparison with C function:

C	Python
bool getAllFaces(long partId, long* numFaces, long** faceIds, void* callbackData);	faceIds = udmFunctionLibrary.GetAllFaces(partId) (ignore numFaces as redundant: fol-

C	Python
where numFaces and faceIds are output parameters and numFaces is the size of faceId.	ded into faceIds, return value is omitted: folded into the faceIds is None check callbackData is omitted)
long unite(long numObjects, long* objectIds, void* callbackData); where numObjects and objectIds are input parameters and numObjects is the size of objectIds.	lret = udpFunctionLibrary.Unite (objectIds) (ignore numObjects as redundant: folded into objectIds callbackData is omitted)
long detachFaces(long nSize, long* faceIds, long* partIds, void* callbackData); where partIds is and output list and nSize is an input parameters and nSize is the size of partIds.	(lret, partIdList) = udpFunctionLibrary.DetachFaces(nSize, faceIds) (nSize is not ignored, callbackData is omitted)

Added Parameters

There is a special case in UDPFunctionLibrary: two functions - DuplicateAlongLine and DuplicateAroundAxis - have new integer listSize parameter added to their signatures.

This parameter defines the size of the output List. This is done for compliance with C++ geometry library as the size of the List must be predefined and this size is different from the existing parameter's values.

Example:

```
(ret, cloneIDs) = funcLib.DuplicateAlongLine(partID, transVec,
numCubes, cloneIdsSize)

(ret, cloneIDs) = funcLib.DuplicateAroundAxis(partID, axis, angle,
nClones, cloneIdsSize)
```

Here cloneIdsSize is a new integer parameter.

Comparison with C function:

C	Python
long duplicateAlongLine(	(lret, cloneIds) = udmFunctionLibrary.DuplicateAlongLine(partId,

C	Python
long partId, UDPVector transVector, int nClones, long* nClones, void* callbackData);	transVec, nClones, cloneIdsSize) (callbackData is omitted cloneIdsSize is a new parameter)
long duplicateAroundAxis(long partId, UDPCoordinateSystemAxis axis, double angle, int nClones, long* nClones, void* callbackData);	(lret, cloneIds) = udmFunctionLibrary.DuplicateAroundAxis (partId, axis, angle, nClones, cloneIdsSize) (callbackData is omitted cloneIdsSize is a new parameter)

Developing a UDM/UDP

Creation

To create a User Defined Primitive in Python you write a Python script that implements UDPExtension class. To create a User Defined Model in Python you write a Python script that implements UDMExtension class (see links for full description).

Location

The scripts are located the same way the C based UDM/UDP are. They are expected to be under the UserDefinedParts or UserDefinedModels sub-directories of one of the library folders (SysLib, UserLib or PersonalLib). They will then appear under the appropriate menu items: **Draw** > **User Defined Primitives for UDP** or **Draw** > **User Defined Model for UDM**.

The sub-directories structure created in one of the specified directory will be displayed in the UDP/UDM menu.

Keep in mind that there is no difference between the menu display for C and Python implementations of UDM or UDP - only the file names without extensions are displayed

Organize

"Lib" sub-directory is a special directory. The contents of this directory is not shown in the menu. In the "Lib" directory you can create Python scripts with base classes and utilities to be used in UDP/UDM Python scripts. All the Lib directories upstream of a script (till the UserDefinedModels

or UserDefinedPrimitives) are included in the Python search path and this allows for easy import of helper modules in such directories.

To use UDM data structures, constants, and/or classes in your Lib sub-directory scripts you have to add import statement to the scripts:

For UDM:extension:

```
from UDM import *
```

For UDP:extension:

```
from UDP import *
```

Edit/Reload

Python is a scripting language, so if you have errors in your script, you will see them at the time you try to run the script. The errors will be displayed in the Message Manager Window. If you need more information, you might be able to get it from log files. See: Debug Logging.

You can always change your script, call **Update Menu** command from **Draw** > **User Defined Model** > **menu** or **Draw** > **User Defined Primitives** > **menu** and run the script again. If you delete script you might want to restart the application instead of calling **Update Menu**.

UDPExtension

Import

You do not have to add import statements for the predefined classes, structures, and constants - it is done for you and all data types described in this document can be used in your Python script.

However you have to add import statements to your helper scripts in your Lib sub-directory.

```
from UDP import *
```

Main class: UDPExtension

You must write a class derived from IUDPExtension with a mandatory name UDPExtension:

```
class UDPExtension(IUDPExtension):
```

The class should implement IUDPextension methods described below.

IUDPExtension methods

All methods are same as the methods in the C UDP implementation. The changes to the methods signatures are just to conform to the Python style.

Mandatory methods.

These methods must be implemented in the UDP Python script as methods of UDPExtension class.

GetLengthParameterUnits()

- returns string.

GetPrimitiveTypeInfo()

- returns UDPPrimitiveTypeInfo.

GetPrimitiveParametersDefinition2()

- returns a list of UDPPrimitiveParameterDefinition2 or None on failure

AreParameterValuesValid2(errorMsg, udpParams)

- errorMsg is a c# list of strings
- udpParams is a c# list of UDPParam
- returns True if udpParams are valid, False otherwise.

CreatePrimitive2(funcLib, udpParams)

- funcLib is UDMFunction library
- udpParams is a c# list of UDPParam
- returns True on success, False on failure.

Optional methods

These methods, which have default implementations, can be implemented as methods of UDPExtension class as needed. Default methods will return NULL or FALSE depending on the return type.

GetPrimitiveParameters()

- returns Python list of strings or NULL

GetRegisteredFaceNames()

- returns Python list of strings or NULL

GetRegisteredEdgeNames()

- returns Python list of strings or NULL

GetRegisteredVertexNames()

- returns Python list of strings or NULL

ProjectParametersOnToValidPlane2(currentUDPParams, projectedUDPParams)

- currentUDPParams is a list of UDPParam
- projectedUDPParams is a list of UDPParam
- returns True on success, False on failure.

MapParametersDefinitionVersions2(oldVersion, oldUDPParams)

- oldVersion is a string
- oldUDPParamsis a list of UDPParam
- returns Python list of UDPParam or NULL

GetOldPrimitiveParametersDefinition2(version)

- version is a string
- returns a list of UDPPrimitiveParameterDefinition2 or None on failure.

Example UDP

```
import sys
class UDPExtension(IUDPExtension):

  def GetLengthParameterUnits(self):
    return "mm"

  def GetPrimitiveTypeInfo(self)
    typeInfo = UDPPrimitiveTypeInfo(
      name = "SampleUDP",
      purpose = "example",
      company="Ansys",
      date="12.21.12",
      version = "1.0")

    return typeInfo
  ...
  ...
```

UDMExtension

Import

You do not have to add import statements for the predefined classes and structures - it is done for you, and all data types described in this document can be used in your Python script.

However you have to add import statements to your helper scripts in your Lib sun-directory.

```
from UDM import *
```

Main class: UDMExtension

You must write a class derived from IUDMExtension with a mandatory name UDMExtension:

```
class UDMExtension(IUDMExtension):
```

The class should implement IUDMExtension methods described below.

IUDMExtension methods

All methods are the same as the methods in the C UDM implementation. The changes to the methods signatures are just to conform to the Python style.

Mandatory methods.

These methods must be implemented in the UDM Python script as methods of UDMExtension class.

GetInfo()

- returns UDMInfo object populated with appropriate UDM information.

IsAttachedToExternalEditor()

- returns True if UDM dll is attached to external editor.
- In case of python UDMs, this should typically return False

CreateInstance(funcLib)

- funcLib is UDMFunctionLibrary
- returns UDMParameters.

GetUnits(instanceId)

- instanceId is a long
- returns string containing units for the instance.

Refresh(funcLib, udmInParams, updatedParams, refreshModifiedPartsOnly, nonEditedPartRefIds)

This Method is called every time a UDM is refreshed. Geometry creation/refresh should happen in this method.

- funcLib is UDMFunctionLibrary
- udmInParams is a list of UDMParameters that comes from desktop
- updatedParams: UDM script can change the UDM parameters it receives. Updated parameters need to be sent back to desktop. If the UDM script is not going to change any of the parameters that it received, it needs to copy udmInParams to updatedParams.
- refreshModifiedPartsOnly is a boolean

  Supporting this flag is optional. For UDMs where the refresh performance is not an issue, it is recommended to ignore this flag and update all parts every time.

  This flag can be used to optimize performance of Refresh method when the model created by UDM is large. If the UDM consists of multiple parts, and new parameters change only a few parts amongst them, UDM script can only modify parts that are changed by the new parameters.

- nonEditedPartRefIds: If RefreshModifiedPartsOnly is true and the UDM script supports partial update, Refresh method needs to return ids of parts that are unchanged.

  returns True on success, False on failure.

ReleaseInstance(instanceId)

- instanceId is a long
- This should release any resources assigned to this particular instance of UDM.
- returns True on success, False on failure.

GetAttribNameForEntityId()

- Returns string that acts as a the name of the attribute containing entity IDs.
- For example, it can return a unique string such as "ATTRIB_XACIS_ID"
- Python UDMs should implement this method.

GetAttribNameForPartId()

- Returns string that acts as a the name of the attribute containing entity IDs.
- For example, it can return a unique string such as "ATTRIB_XACIS_ID" (Can be same as GetAttribNameForEntityId())
- Python UDMs should implement this method.

Optional methods

These methods have default implementations (default is to return NULL or FALSE depending on the return type) but can be overridden by the user as needed as methods of UDMExtension class.

DialogForDefinitionOptionsAndParams(self, defData, optData, params):

Replaces the old UDMDialogForDefinitionAndOptions method, which is still supported, but users are urged to use UDMDialogForDefinitionOptionsAndParams. If both methods are present, application will use UDMDialogForDefinitionOptionsAndParams.

- UDM can pop up dialog for UDM definition, options, parameters in this method. Definition, options, and parameters are set/modified by user and returned to application. Dll can also just give default definition, options and parameters.
- Returns two booleans and a string
- First boolean returns whether the method was successful or not.
- Second boolean returns whether the application should popup a dialog box. If it is True, application will populate a dialog with definition, options, parameters that are returned.
- String returned contains length units for parameters.

DialogForDefinitionAndOptions(self, defData, optData) [Deprecated]

UDM can pop up dialog for UDM definition and options in this method. Definition, and options are set/modified by user and returned to application. Dll can also just give default definition and options.

- Returns two booleans.
- First boolean provides whether the call to this method was successful or not.
- Second boolean determines whether the application should pop up a dialog box. If this is true, application will populate the dialog with the definitions and options that are returned. As no parameters are returned, no parameters are shown in this dialog.

GetInstanceSourceInfo(instanceId)

- instanceId is a long
- returns string containing source information of UDM instance. It is used to create initial name for UDM instance.

ShouldAttachDefinitionFilesToProject()

- returns true if any of definition files needs to be attached to project
- returns python list of string containing definition names of files or NULL

Example UDM

```
class UDMExtension(IUDMExtension):


  def IsAttachedToExternalEditor(self):
```

```
    return False


  def GetInfo(self)
    udmInfo = UDMInfo(
      name = "SampleUDM",
      purpose = "udm example",
      company="Ansys",
      date="12.21.12",
      version = "1.0")


    return udmInfo
  ...
  ...
```

UDMFunctionLibrary

UDMFunctionLibrary implements IUDMFunctionLib interface. The IUDMFunctionLib object is passed as a parameter to Python script in the following functions

- CreateInstance
- Refresh

You can call any of the functions from the functions list (shown below).

```
partRefId = udmFunctionLib.GetPartRefId(partId)
```

For example sample code that calls GetBoundingBox in Python script can look like this:

```
partId = 10
exact = True
udpPosition = UDPPosition(0,0,0)


(bret, udpPositionLow, udpPositionHigh) = udmFunc-
tionLibrary.GetBoundingBox(partId, exact);


if bret:
  udpPosition.X = udpPositionLow.X
```

As you can see udpPositionLow and udpPositionHigh output parameters are defined in the call to GetBoundingBox function. There is no need to define them before the function call.

Functions list:

1. ***List_of_UDMDefinition:*** udmDefinitionList = **GetDefinition**()
2. ***List_of_UDMOption:*** udmOptionList = **GetOptions**()
3. ***bool:*** bret = **SetMaterialName**(***string:*** matName, ***int:*** partId)
4. ***bool:*** bret = **SetMaterialName2**(***string:*** matName, ***string:*** partName)
5. ***bool:*** bret = **SetPartName**(***string:*** partName, ***int:*** partId)
6. ***int:*** iret = **GetInstanceId**()
7. ***string:*** str = **GetPartRefId**(***int:*** partId)
8. ***bool:*** bret = **SetPartRefId**(***int:*** partId, ***string:*** refId)
9. ***List_of_int:*** faceIds = **GetAllFaces**(***int:*** partId)
10. ***List_of_int:*** edgeIds = **GetAllEdges**(***int:*** partId)
11. ***List_of_int:*** vertexIds = **GetAllVertices**(***int:*** partId)
12. ***bool:*** bret = **SetFaceAttribs**(***List_of_int:*** faceIds, ***List_of_string:*** attribs)
13. ***bool:*** bret = **SetEdgeAttribs**(***List_of_int:*** edgeIds, ***List_of_string:*** attribs)
14. ***bool:*** bret = **SetVertexAttribs**(***List_of_int:*** vertexIds, ***List_of_string:*** attribs)
15. ***string:*** str = **GetModelerUnit**()
16. ***string:*** str = **GetCacheFileForUDMResume**()
17. ***bool:*** bret = **SetPartColor**(***int:*** partId, ***int:*** nColor)
18. ***bool:*** bret = **SetPartFlags**(***int:*** partId, ***int:*** nFlags)
19. (***bool:*** bret, ***UDPPosition:*** low, ***UDPPosition:*** high) = **GetBoundingBox**(***int:*** partId, ***bool:*** exact)
20. ***bool:*** bret = **IsParametricUpdate**()
21. ***bool:*** bret = **SetMaterialNameByRefId**(***string:*** partRefID, ***string:*** matName)
22. ***bool:*** bret = **SetPartNameByRefId**(***string:*** partRefId, ***string:*** partName)
23. ***bool:*** bret = **SetPartColorByRefId**(***string:*** partRefId, ***int:*** nColor)
24. ***bool:*** bret = **SetPartFlagsByRefId**(***string:*** partRefId, ***int:*** nFlags)

In addition to the above functions all functions defined in the UDPFunctionLib are available in the IUDMFunctionLib and can be called directly exactly the same way as the IUDMFunctionLib functions.

Example:

```
udmFunctionLib.CreateCircle(center,radius,ratio,isCovered)
```

UDM/UDP Functions

Return Values for Each UDM and UDP Function

ID – ID of created Object

SI – Success Indicator. Identifies whether or not operation was successful.

Functions list:

1. ***bool***: SI = **AddMessage(*MessageSeverity*:** messageSeverity, ***string***: message**)**
2. ***bool***: SI = **NameAFace(*UDPPosition***: pointOnFace, ***string***: faceName**)**
3. ***bool***: SI = **NameAEdge(*UDPPosition***: pointOnEdge, ***string***: edgeName**)**
4. ***bool***: SI = **NameAVertex(UDPPosition**: pointOnVertex, ***string***: vertexName**)**
5. ***int***: ID = **GetFaceIDFromPosition(*UDPPosition***: pointOnFace**)**
6. ***int***: ID = **GetEdgeIDFromPosition(*UDPPosition***: pointOnEdge**)**
7. ***int***: ID = **CreatePolyline(*UDPPolylineDefinition***: polylineDefinition**)**
8. ***int***: ID = **CreateRectangle(*CoordinateSystemPlane***: whichPlane, ***UDPPosition***: centerPoint, ***List_of_double***: widthAndHeight, ***int***: isCovered**)**
9. ***int***: ID = **CreateArc(*CoordinateSystemPlane***: whichPlane, ***UDPPosition***: centerPoint, ***UDPPosition***: startPoint, ***double***: fAngle**)**
10. ***int***: ID = **CreateCircle(*CoordinateSystemPlane***: whichPlane, ***UDPPosition***: centerPoint, ***double***: fRadius, ***int***: isCovered**)**
11. ***int***: ID = **CreateEllipse(*CoordinateSystemPlane***: whichPlane, ***UDPPosition***: centerPoint, ***double***: fMajorRadius, ***double***: fRadiusRatio, ***int***: isCovered**)**
12. ***int***: ID = **CreateRegularPolygon(*CoordinateSystemPlane***: whichPlane, ***UDPPosition***: centerPoint, ***UDPPosition***: startPoint, ***int***: numOfSides, ***int***: isCovered**)**
13. ***int***: ID = **CreateEquationBasedCurve(*UDPEquationBasedCurveDefinition***: curveDefinition**)**
14. ***int***: ID = **CreateEquationBasedSurface(*UDPEquationBasedSurfaceDefinition***: surfaceDefinition**)**
15. ***int***: ID = **CreateSpiral(*UDPSpiralDefinition***: spiralDefinition**)**
16. ***int***: ID = **CreateBox(*UDPPosition***: startPoint, ***List_of_double***: boxXYZsize**)**
17. ***int***: ID = **CreateSphere(*UDPPosition***: centerPoint, ***double***: fRadius**)**
18. ***int***: ID = **CreateCylinder(*CoordinateSystemAxis***: whichAxis, ***UDPPosition***: centerPoint, ***double***: fRadius, ***double***: fHeight**)**
19. ***int***: ID = **CreateCone(*CoordinateSystemAxis***: whichAxis, ***UDPPosition***: centerPoint, ***double***: fBottomRadius, ***double***: fTopRadius, ***double***: fHeight**)**
20. ***int***: ID = **CreateTorus(*CoordinateSystemAxis***: whichAxis, ***UDPPosition***: centerPoint, ***double***: fMajorRadius, ***double***: fMinorRadius**)**
21. ***int***: ID = **CreatePolyhedron(*CoordinateSystemAxis***: whichAxis, ***UDPPosition***: centerPoint, ***UDPPosition***: startPosition, ***int***: numOfSides, ***double***: fHeight**)**

22. ***int***: ID = **CreateHelix(*UDPHelixDefinition***: helixDefinition**)**
23. ***bool***: SI = **Unite(*List_of_int***: pObjectIDArray**)**
24. ***bool***: SI = **Subtract(*List_of_int***: pBlankObjectIDArray, ***List_of_int***: pToolObjectIDArray**)**
25. ***bool***: SI = **Intersect(*List_of_int***: pObjectIDArray**)**
26. ***bool***: SI = **Imprint(*List_of_int***: pBlankObjectIDArray, ***List_of_int***: pToolObjectIDArray**)**
27. ***bool***: SI = **SweepAlongVector(*int***: profileID, ***UDPVector***: sweepVector, UDPSweep-Options: sweepOptions**)**
28. ***bool***: SI = **SweepAroundAxis(*int***: profileID, ***CoordinateSystemAxis***: whichAxis, ***double***: sweepAngle, ***UDPSweepOptions***: sweepOptions**)**
29. ***bool***: SI = **SweepAlongPath(*int***: profileID, ***int***: pathID, ***UDPSweepOptions***: sweep-Options**)**
30. ***bool***: SI = **Translate(*int***: partID, ***UDPVector***: translateVector**)**
31. ***bool***: SI = **Rotate(*int***: partID, ***CoordinateSystemAxis***: whichAxis, ***double***: rotateAngle**)**
32. ***bool***: SI = **Mirror(*int***: partID, ***UDPPosition***: mirrorPlaneBasePosition, ***UDPVector***: mir-rorPlaneNormalVector**)**
33. ***bool***: SI = **Transform(*int***: partID, ***List_of_double***: rotationMatrix, ***UDPVector***: trans-lateVector**)**
34. ***bool***: SI = **Scale(*int***: partID, ***double***: xScale, ***double***v: yScale, ***double***: zScale**)**
35. (***bool***: SI, ***List_of_int***: cloneIDs**)** = **DuplicateAlongLine(*int***: partID, ***UDPVector***: trans-lateVector, ***int***: numTotalObjs, ***int***: cloneIDsListSize**)**
36. (***bool***: SI, ***List_of_int***: cloneIDs) = **DuplicateAroundAxis(*int***: partID, ***Coordin-ateSystemAxis***: whichAxis, ***double***: rotateAngle, ***int***: numTotalObjs, ***int***: cloneIDsListS-ize**)**
37. ***int***: ID = **DuplicateAndMirror(*int***: partID, ***UDPPosition***: mirrorPlaneBasePosition, ***UDPVector***: mirrorPlaneNormalVector**)**
38. ***bool***: SI = **Connect(*List_of_int***: objectIDArray**)**
39. ***bool***: SI = **Offset(*int***: partID, ***double***: offsetDistance**)**
40. ***int***: ID? = **Section(*int***: partID, ***CoordinateSystemPlane***: sectionPlane**)**
41. (***bool***: SI , ***int***: ID) = **Split(*int***: partID, ***CoordinateSystemPlane***: splitPlane, **SplitWhichSideToKeep**: whichSideToKeep, ***bool***: bSplitCrossingObjectsOnly**)**
42. (***bool***: SI , ***List_of_int***: importedObjectIDs) = **ImportNativeBody2(*string***: fileNameWithFullPath**)**
43. (***bool***: SI, ***List_of_int***: importedObjectIDs) = **ImportAnsoftGeometry(*string***: fileNameWithFullPath, ***List_of_string***: overridingParamsNameArray, ***List_of_ UDPParam***: overridingParamsArray**)**
44. ***int***:ID = **Clone(*int***: partID**)**
45. ***bool***: SI = **DeletePart(*int***: partID**)**
46. ***int***: ID = **CreateObjectFromFace(*int***: faceID**)**

47. ***bool***: SI = **Fillet(*UDPBLNDElements***: entitiesToFillet, ***UDPBLNDFilletOptions***: filletOptions**)**
48. ***bool***: SI = **Chamfer(*UDPBLNDElements***: entitiesToChamfer, ***UDPBLNDChamferOptions***: chamferOptions**)**
49. (***bool***: SI, ***List_of_int***: newPartIDs) = **DetachFaces(*int***: newPartIDArraySize, ***List_of_int***: faceIDs**)**
50. (***bool***: SI, ***List_of_int***: newPartIDs) = **DetachEdges(*int***: newPartIDArraySize, ***List_of_int***: edgeIDs**)**
51. ***int***: ID = **CreateObjectFromEdge(*int***: edgeID**)**
52. ***bool***: SI = **SheetThicken(*int***: partID, ***double***: fThickness, ***bool***: bThickenBothSides**)**
53. (***bool***: SI , ***List_of_int***: newPartIDArray) = **SweepFaceAlongNormal(*int***: newPartIDArraySize, ***List_of_int***: faceIDArray, ***double***: sweepLength**)**
54. ***bool***: SI = **CoverLine(*int***: partID**)**
55. ***bool***: SI = **CoverSurface(*int***: partID**)**
56. ***bool***:SI= **UncoverFaces(*List_of_int***: faceIDArray**)**
57. (***bool***: SI , ***int***: numPartsCreated, ***List_of_int***: faceIDArray) = **SeparateBodies(*int***: partID, ***int***: numPartsCreated**)**
58. ***bool***: SI = **MoveFaces(*List_of_int***: faceIDArray, ***bool***: bMoveAlongNormal, ***double***: fOffsetDistance, ***UDPVector***: moveVector**)**
59. ***bool***: SI = **WrapSheet(*int***: sheetBodyID, ***int***: targetBodyID**)**
60. ***bool***: SI = **ImprintProjection(*int***: blankBodyID, ***List_of_int***: toolBodyIDArray, ***bool***: bNormalProjection, ***UDPVector***: projectDirection, ***double***: projectDistance**)**
61. ***string***: path = **GetTempDirPath()**
62. ***string***: path = **GetSysLibDirPath()**
63. ***string***: path = **GetUserLibDirPath()**
64. ***string***: path = **GetPersonalLibDirPath()**
65. ***string***: path = **GetInstallDirPath()**
66. ***string***: path = **GetProjectPath()**
67. (***bool***: SI, ***bool***: abort) = **SetProgress(*UDPProgress***: progress**)**

UDP/UDM Structures and Constants

The following sections describe:

- UDP/UDM Structures
- UDP/UDM Constants

UDP/UDM Structures

Differences compared to C API

- **UDMDefinition**
- **UDMOptions**
- **UDMParameters**

  Instead of containing arrays of data, the structures contain single fields where each field corresponds to an item in a different array from the original C API. The structure objects thus constructed are added to the Python list. Alternately the Python list can be initialized using the structure objects.

  Example (creating UDMParameter list):

```
udmParamList = [
  UDMParameter(
    "cubeSizeName", UnitType.LengthUnit,
    UDPParam(ParamDataType.Double, cubeSize),
    ParamPropType.Value,
    ParamPropFlag.MustBeReal),
  UDMParameter(
    "cubeDistanceName", UnitType.LengthUnit,
    UDPParam(ParamDataType.Double, cubeDistance),
    ParamPropType.Value,
    ParamPropFlag.MustBeReal),
  UDMParameter("numCubesName", UnitType.LengthUnit,
    UDPParam(ParamDataType.Int, numCubes),
  ParamPropType.Number,
  ParamPropFlag.MustBeInt]
```

- **UDPParam**
- **UDPParamData**

  Data field in UDPParam is now an object - the same for all types of data used - as Python can work with any type of data.

  UDPParamData is obsolete, thus not implemented. Be sure to set proper data type to UDPParam.DataType when setting UDPParam.Data.

  Example:

```
nCubesParam = UDPParam(ParamDataType.Int, numCubes)
nCubes = nCubesParam.Data


distanceParam = UDPParam()
distanceParam.setDouble(10.5)
doubleDistance = distanceParam.Data * 2
```

- **UDP3x3Matrix**

  The structure is not implemented. Use size 9 Python List of doubles instead.

  Example:

  ```
  rotationMatrix =[0,0,1, 1,0,0, 0,0,1]
  udpFunctionLib.Transform(partId, rotationMatrix, trans-
  lationVector)
  ```

List of structures

You can use constructors to create a structure. You can also modify fields - directly or by provided methods.

Example:

```
pos1 = UDPPosition(1,2,3)
pos2 = UDPPosition(x=1,y=10,z=0)
pos2.Z = pos1.Z
udpParam = UDPParam(ParamDataType.Double,1)
value = udpParam.Data
```

Structure	Construction	Members
UDPPrimitiveTypeInfo	UDPPrimitiveTypeInfo(string name, string purpose, string company, string date, string version)	string Name string Purpose string Company string Date string Version
UDPPrim-itiveParameterDefinition	UDPPrim-itiveParameterDefinition(	string Name string Description

Structure	Construction	Members
	string name, string description, UnitType unitType, double defaultValue)	UnitType UnitType double DefaultValue
UDPParam	UDPParam() UDPParam(ParamDataType dataType, object data) object can be int, double, string, bool or UDPPosition **methods:** setInt(int val) setBool(bool val) setString(string val) setDouble(double val) setPosition(UDPPosition val)	ParamDataType DataType object Data object can be int, double , string, bool or UDPPosition
UDPPrim-itiveParameterDefinition2	UDPPrim-itiveParameterDefinition2(string name, string description, UnitType unitType, ParamPropType propType, ParamPropFlag propFlag, UDPParam defaultValue)	string Name string Description UnitType UnitType ParamPropType PropType ParamPropFlag PropFlag UDPParam DefaultValue
UDPPosition	UDPPosition(double x, double y,	double X double Y double Z

Structure	Construction	Members
	double z)	
UDPVector	UDPVector(double x, double y, double z)	double X double Y double Z
UDPSweepOptions	UDPSweepOptions(SweepDraftType draftType, double draftAngle, double twistAngle)	SweepDraftType DraftType double DraftAngle double TwistAngle
UDPPolylineSegmentDefinition	UDPPolylineSegmentDefinition(PolylineSegmentType segmentType, int segmentStartIndex, int numberOfPoints, double angle, UDPPosition centerPoint, CoordinateSystemPlane arcPlane)	PolylineSegmentType SegmentType int segmentStartIndex, int numberOfPoints, double angle, UDPPosition centerPoint, CoordinateSystemPlane arcPlane)
UDPPolylineDefinition	UDPPolylineDefinition() UDPPolylineDefinition(List_of_UDPPosition positions, List_of_UDPPolylineSegmentDefinition segDefs, int closed, int covered)	int IsClosed int IsCovered List_of_UDPPosition ArrayOfPosition List_of_UDPPolylineSegmentDefinition ArrayOfSegmentDefinition
UDPEquationBasedCurveDefinition	UDPEquationBasedCurveDefinition(string functionXt, string functionYt, string functionZt,	string FunctionXt string FunctionYt string FunctionZt double TStart double TEnd

Structure	Construction	Members
	double tStart, double tEnd, int numOfPointsOnCurve)	int NumOfPointsOnCurve
UDPEquationBasedSurfaceDefinition	UDPEquationBasedSurfaceDefinition(string functionXuv, string functionYuv, string functionZuv, double uStart, double uEnd, double vStart, double vEnd int reserved1 int reserved2)	string FunctionXuv string FunctionYuv string FunctionZuv double UStart double UEnd double VStart double VEnd two integer arguments that are reserved for future use. They need to be provided, for example as 0. For example: `theSurfaceDefinition = UDPEquationBasedSurfaceDefinition ("u","v","1",0,1,0,1,0,0)`
UDPHelixDefinition	UDPHelixDefinition(int profileID, UDPPosition ptOnAxis, UDPPosition axisDir, double noOfTurns, bool isRightHanded, double radiusChangePerTurn, double pitch)	int ProfileID UDPPosition PtOnAxis UDPPosition AxisDir double NoOfTurns bool IsRightHanded double RadiusChangePerTurn double Pitch
UDPSpiralDefinition	UDPSpiralDefinition(int profileID, UDPPosition ptOnAxis, UDPPosition axisDir, double noOfTurns,	int ProfileID UDPPosition PtOnAxis UDPPosition AxisDir double NoOfTurns bool IsRightHanded

Structure	Construction	Members
	bool isRightHanded, double radi-usChangePerTurn)	double RadiusChangePerTurn
UDPBLNDElements	UDPBLNDElements(int partID, int noOfEdges; int* listOfEdges;) UDPBLNDElements(int partID, int noOfVertices; int* listOfVertices;)	UDPBLNDElements can hold either edges or vertices, but not both at the same time. Edges should be applied to solids, and vertices should be applied to sheets. int PartID /* part to be blended i.e. filleted/chamfered */ int noOfEdges; int* listOfEdges; /* edges to be blended */ int noOfVertices; int* listOfVertices; /* vertices to be blended */
UDPBLNDFilletOptions	UDPBLNDFilletOptions(bool supressFillet, BLNDFilletRadiusLaw fil-letRadiusLaw, double filletStartRadius, double filletEndRadius, bool fol-lowSmoothEdgeSequence, BLNDFilletType filletType, double setbackDistance, double bulgeFactor)	bool SupressFillet /* Reserved for future */ BLNDFilletRadiusLaw Fil-letRadiusLaw double FilletStartRadius double FilletEndRadius bool Fol-lowSmoothEdgeSequence /* Reserved for future */ BLNDFilletType FilletType double SetbackDistance double BulgeFactor /* Reserved for future */
UDPBLNDChamferOptions	UDPBLNDChamferOptions(bool supressChamfer, BLNDChamferRangeLaw chamferRangeLaw,	bool SupressChamfer BLNDChamferRangeLaw Cham-ferRangeLaw double ChamferLeftRange

Structure	Construction	Members
	double chamferLeftRange, double chamferRightRange)	double ChamferRightRange
UDPProgress	UDPProgress(int prog, int subProg, string mesg, string subMesg)	int Prog int SubProg string Mesg string SubMesg
UDMInfo	UDMInfo(string name, string purpose, string company, string date, string version)	string Name string Purpose string Company string Date string Version
UDMDefinition	UDMDefinition() UDMDefinition(string name, UDParam value, ParamPropType propType, ParamPropFlag propFlag)	string DefName UDPParam DefValue ParamPropType PropType ParamPropFlag PropFlag
UDMOption	UDMOption() UDMOption(string name, UDParam value, ParamPropType propType, ParamPropFlag propFlag)	string OptName UDPParam OptValue ParamPropType PropType ParamPropFlag PropFlag
UDMParameter	UDMParameter() UDMParameter(	string ParamName UDPParam ParamValue UnitType UnitType

Structure	Construction	Members
	string name, UDParam value, UnitType unitType, ParamPropType propType, ParamPropFlag propFlag)	ParamPropType PropType ParamPropFlag PropFlag

UDP/UDM Constants

Full names of enum constants must be used in scripts.

Example:

```
unitType = UnitType.LengthUnit
dataType = ParamDataType.Int
```

Enum constants:

enum Constant	Parameters
UnitType	NoUnit LengthUnit AngleUnit
ParamDataType	Int Double String Bool Position Unknown
ParamPropType	Text Menu Number Value FileName Checkbox Position Unknown

enum Constant	Parameters
ParamPropFlag	NoFlag ReadOnly MustBeInt MustBeReal Hidden Unknown
CoordinateSystemAxis	XAxis YAxis ZAxis
CoordinateSystemPlane	XYPlane YZPlane ZXPlane
SweepDraftType	ExtendedDraft RoundDraft NaturalDraft MixedDraft
SplitWhichSideToKeep	SplitKeepBoth SplitKeepPositiveOnly SplitKeepNegativeOnly
PolylineSegmentType	LineSegment ArcSegment SplineSegment AngularArcSegment
MessageSeverity	WarningMessage ErrorMessage InfoMessage IncompleteMessage FatalMessage
BLNDFilletRadiusLaw	BLNDConstantRadius BLNDVariableRadius

enum Constant	Parameters
BLNDFilletType	BLNDRound /* The outward surface of the fillet is curved.*/ BLNDMitered /* The outward surface of the fillet is flat and cut at an angle.*/
BLNDChamferRangeLaw	BLNDConstantRange BLNDVariableRange
PartPropertyFlags	PropNonModel PropDisplayWireFrame PropReadOnly PostprocessingGeometry PropInvisible PropShowDirection PropDummy

Introduction to CPython

An Ansys Electronics Desktop 2024R1 feature allows CPython to be used for standalone scripting, as an alternative to IronPython, to:

- Launch Ansys Electronics Desktop (InitializeNew)
- Connect with a running instance of Ansys Electronics Desktop (Initialize)
- Execute Ansys Electronics Desktop script functions

One advantage of CPython is the large set of libraries and tools that are available. See below for instructions on modifying a script so that it can be launched with CPython interpreters.

CPython Script Engine and ansysedt process are comunicated using GRPC, the ansysedt process started as GRPC server, the Script Script Engine acted as the GRPC clinet.

> **Important:**
>
> (Supported on both Windows and Linux in 24R1)
>
> Launching Ansys Electronics Desktop on a remote machine is not supported in this release. Initialize() will connect to a remote instance that is already running on the remote machine, but cannot launch a new instance on the remote machine.
>
> Connect() to an opened project in remote machine needs to use a share network path.

Creating an External Script

While the same as IronPython when run externally, a CPython recorded script must be modified by adding the following lines to the beginning of your script *before* `import ScriptEnv`

```
import sys

  # Imports the sys module containing system-specific functions nat-
  ive to Python.

sys.path.append(r"<InstallationPath>/PythonFiles/DesktopPlugin")

  # Adds the PythonFiles/DesktopPlugin subfolder to the list of dir-
  ectories Python searches for modules and files.
```

Those lines are followed by:

```
import ScriptEnv

  # This imports ScriptEnv.py from the installation path specified
  above.
```

Follow that with:

- Either InitializeNew() *or* Initialize(), as described below.
- Any desired Electronics Desktop scripting commands.
- Closing command, as described below.

Start ansysedt as GRPC server

But default ansysedt process will be started as GRPC server. When there is no argument provided ansysedt.exe it start listening on the first available port from 50051.

-grpcsrv Flag

ansysedt -grpcsrv <optional port number or port range>.

ansysedt –grpcsrv

If no Port Number specified, same as the default like no grpcsrv flag

ansysedt –grpcsrv portNumber

ansysedt process will be listening on the port number, but report error as the port is used by other application.

ansysedt –grpcsrv FirstPortNumber:LastPortNumber

ansysedt process will be listening on the first available port within the port range, but report error if all ports in the range used.

ansysedt –grpcsrv FirstPortNumber:NumberOfPorts

ansysedt process will be listening on the first available port within the port range, but report error if all ports in the range used.

Note: If the last number if smaller than the first number the last number will be treated as the number of ports. (50051: 50250 has the same range as 50051:200)

Connect Functions:

1. Connect(projectPath)
    a. Connect to the opened project.
    b. Launch ansysedt.exe, open the project, then connect to it.
    c. Error if the project does not exist.
2. Connect(portNumber) Connect to running ansysedt process in the local machine. Error if no ansysedt process listening on this port.
3. Connect(machine, projectPath) connect to an open project in remote Machine. projectPath must be a network path. Error if no ansysedt process opened the project.
4. Connect(machine, portNumber) connect to a running ansysedt process on the remote machine. Error if no ansysedt process listening on this port.

Example:

```
import sys

sys.path.append(r"<InstallationPath>/PythonFiles/DesktopPlugin")

import ScriptEnv

ScriptEnv.Connect(r”c:\myProjects\Project1.aedt”)

oProject = oDesktop.GetActiveProject()

print(oProject.GetName())
```

Launching Electronics Desktop

To launch a new instance of Electronics Desktop and connect oApplication and oDesktop to it:

```
InitializeNew(NonGraphical = <True|False>, Module = None, Machine =
"", Port = <Port#>)
```

Where:

- **NonGraphical** – Specifies whether to launch Electronics Desktop in non-graphical mode.
- **Module** – Behavior remains unchanged from Iron Python and should be left defaulted to "None." See the code in ScriptEnv.py for more details.
- **Machine** – Currently an empty string, as InitializeNew() will only launch Electronics Desktop on the current machine.
- **Port** – Electronics Desktop will launch using the first unused port it finds starting at <Port#>. If Port = 0, the starting port will be 50051.

> **Note:**
>
> InitializeNew() will *always* launch a new instance of Electronics Desktop. Please use Initialize() to connect to an existing instance. See below.

Connecting with a Running Instance of Electronics Desktop

To connect oApplication and oDesktop to an existing Electronics Desktop instance, or launch a new instance and connect to it if necessary:

```
Initialize(name, NG = <True|False>, machine = "", port = <Port#>)
```

Where:

- **Name** – Ignored.
- **NG** – If launch is necessary, specifies whether to launch Electronics Desktop in non-graphical mode.
- **Machine** – The machine on which to launch/connect. For current machine, pass empty string or use localhost.
- **Port** – If port is nonzero, the script tries to connect to an existing instance on <port#> running on <machine>. If there is no instance running on that <port>, a new instance of Electronics Desktop launches on that port and then connects to it. If port = 0, the new instance is launched on the first free port, starting at 50051.

Closing Electronics Desktop/Ending the Script

To close Electronics Desktop, add the following line to the end of the script:

```
ScriptEnv.Shutdown()

  # This stops ScriptEnv.py. If you are running multiple scripts,
  include this only at the end of the last script.
```

-grpcsrv Flag

This flag will launch the application in a mode where the executable serves as a scripting server that can be used for CPython scripting in conjunction with the CPython stand alone scripting instructions that were mentioned earlier. The
-ng flag can be combined with -grpcsrv.

On Windows:

```
ansysedt.exe -grpcsrv <optional port number>.
```

On Linux:

```
ansysedt -grpcsrv <optional port number>.
```

If the port number is omitted, the default of 50051 will be used.

With -grpcsrv, a message will be displayed in the **Messages** window indicating that the server was started. If the requested port is in use by another application, starting the sever may fail.

Related Topics:

Standalone Scripting in Iron Python

Ansys Electronics Desktop Scripting

This chapter provides an overview of scripting in Ansys Electronics Desktop.

Overview of Ansys Electronics Desktop Scripting Objects

Running a Script

Recording a Script

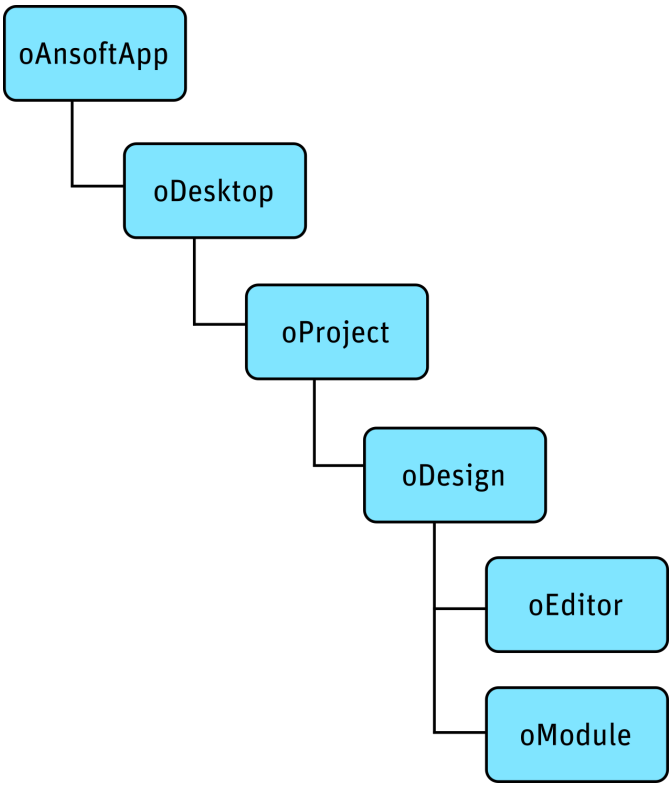
Working with Project Scripts

Executing a Script from Within a Script

Ansys Electronics Desktop Scripting Conventions

Overview of Electronics Desktop Scripting Objects

When you record a script using Ansys Electronics Desktop, the beginning of the script must contain some standard commands, as illustrated in the following chart. The commands in the chart define the objects used by Electronics Desktop in the script and assign values to these objects. The objects are used in the hierarchical order shown.



The commands are described below, followed by examples.

oAnsoftApp

The **`oAnsoftApp`** object provides a handle for Iron Python or VBscript to access the `Ansoft.ElectronicsDesktop` product.

In Iron Python, for example:

```
oAnsoftApp = CreateObject('Ansoft.ElectronicsDesktop')
```

In VBscript, for example:

```
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
```

oDesktop

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

In Iron Python, for example:

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

In VBscript, for example:

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

Consult the following for details about script commands recognized by `oDesktop`:

- Desktop Object Script Commands.

oProject

The **`oProject`** object corresponds to one project open in Electronics Desktop. It is used to manipulate the project and its data. Its data includes variables, material definitions, and one or more designs.

In Iron Python, for example:

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

In VBscript, for example:

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

Consult the following for details about script commands recognized by `oProject`:

- Project Object Script Commands

oDesign

The **`oDesign`** object corresponds to a design in the project. This object is used to manipulate the design and its data, including variables, modules, and editors.

In Iron Python, for example:

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

In VBscript, for example:

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

Consult the following for details about script commands recognized by `oDesign`:

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

oEditor

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

In Iron Python, for example:

```
oEditor = oDesign.SetActiveEditor('3D Modeler')
oEditor = oDesign.SetActiveEditor('SchematicEditor')
```

In VBscript, for example:

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

Consult the following for details about script commands recognized by `oEditor`:

- 3D Modeler Editor Script Commands
- Schematic Editor Script Commands

> **Important:**
>
> There is no Reporter Editor object for `oEditor`. Reporter Editor commands are executed by `oDesign`.
>
> See: Reporter Editor Script Commands.

oModule

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

In IronPython, for example:

```
oModule = oDesign.GetModule('BoundarySetup')
```

In VBscript, for example:

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

Consult the following for details about script commands recognized by `oModule`:

- Analysis Module Script Commands
- Boundary and Excitation Module Script Commands
- Field Overlays Module Script Commands
- Mesh Operations Module Script Commands
- Optimetrics Module Script Commands
- Radiation Module Script Commands
- Reduce Matrix Module Script Commands
- Solutions Module Script Commands

Example Script Opening

Combining the above objects, a script in Iron Python could begin like the following:

```
oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
oDesktop = oAnsoftApp.GetAppDesktop()
oProject = oDesktop.SetActiveProject("Project1")
oDesign = oProject.SetActiveDesign("Design1")
oEditor = oDesign.SetActiveEditor("3D Modeler")
oModule = oDesign.GetModule("BoundarySetup")
```

In VBscript, this would be:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
Set oProject = oDesktop.SetActiveProject("Project1")
Set oDesign = oProject.SetActiveDesign("Design1")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")
Set oModule = oDesign.GetModule("BoundarySetup")
```

GetActiveProject and GetActiveDesign for Wider Use

The sample script above only works for "Design1" within "Project1". To create a script that is usable for any open project, you can use one or both of `GetActiveProject` and `GetActiveDesign`.

In IronPython:

```
oProject = oDesktop.GetActiveProject()
oDesign = oProject.GetActiveDesign()
```

In VBscript:

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

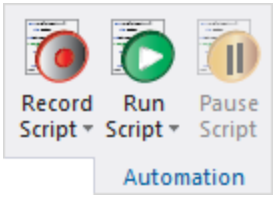
Running a Script

Electronics Desktop scripts can be run from within the software or from the command line.

Within Electronics Desktop

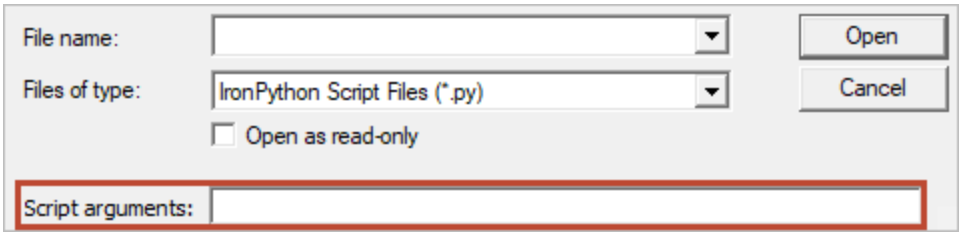
To run scripts in Electronics Desktop:

1. Click **Tools** > **Run Script**, or select the **Automation** tab and click the **Run Script** icon:



   The **Run Script** file browser appears.

2. Use the file browser to locate the script file (*.vbs, *.py, or *.js).
3. If desired, type script arguments in the Script Arguments field:



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

4. Click **Open**.

   Electronics Desktop executes the script.

   While script execution is in progress, the **Run Script** button transforms into a **Stop Script** button. Click **Stop Script** to stop the script execution.

   To temporarily pause a running script, click **Pause Script**. This button transforms into a **Resume Script** button, which you can click to resume script execution.

From the Command Line

To run a script from a command line, add the `-runscriptandexit` or `-runscript` argument to the Electronics Desktop command line syntax.

To use script arguments, add the `-scriptargs` parameter and specify the arguments. For example:

```
ansysedt.exe -scriptargs "hello there"
```

In Iron Python, the command line parameter following `-scriptargs` is passed without modification as a single string in the ScriptArgument python variable.

In VBscript, the command line parameter following `-scriptargs` is split into multiple strings and converted to a VBscript collection which is accessible via the **`AnsoftScript.Arguments`** collection. To access these arguments, for example:

```
msgbox AnsoftScript.Arguments(0) // Returns 'hello'

msgbox AnsoftScript.Arguments(1) // Returns 'there'
```

For more information about running a script from the command line, consult the EMIT help topic "Running Ansys Electronics Desktop from the Command Line".

Direct Launch

If you run a script directly from the command line without launching Electronics Desktop, script arguments will be in the **`WSH.arguments`** collection instead of the **`AnsoftScriptHost.arguments`** collection. The following script ensures the arguments are accessed regardless of the collection in which they reside:

```
on error resume next

dim args

Set args = AnsoftScript.arguments

if(IsEmpty(args)) then

Set args = WSH.arguments

End if

on error goto 0

  // At this point, args has the arguments regardless of collection

msgbox "Count is " & args.Count

for i = 0 to args.Count - 1

msgbox args(i)

next
```

Recording a Script

Electronics Desktop can record a script based on UI actions and save this script in either IronPython (*.py) or VBscript (*.vbs) format.

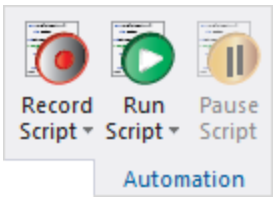
Scripts can be saved to an external file, or to the project.

> **Important:**
>
> When you record a script, every subsequent action you take is recorded. You must manually stop recording.

Recording a Script to File

To record a script to file:

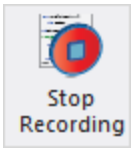
1. Click **Tools** > **Record Script to File**, or select the **Automation** tab and click the **Record Script** icon:



   A **Save As** file browser appears.

2. Navigate to the location where you want to save the script.
3. In the **File Name** field, type a name for the script file.
4. Use the **Save as Type** drop-down menu to select either IronPython or VBscript.
5. Click **Save**.

   The **Record Script** button transforms into a **Stop Recording** button, and Electronics Desktop begins recording your actions.



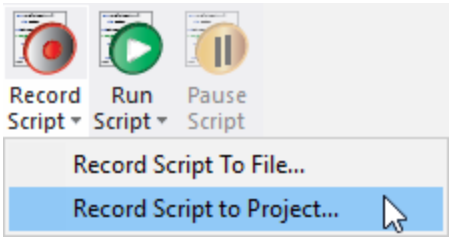
6. Perform the steps you want to record.
7. When you have finished recording the script, click **Stop Recording**, or select **Tools** > **Stop Script Recording**.

   The recorded script is saved to the folder you specified.

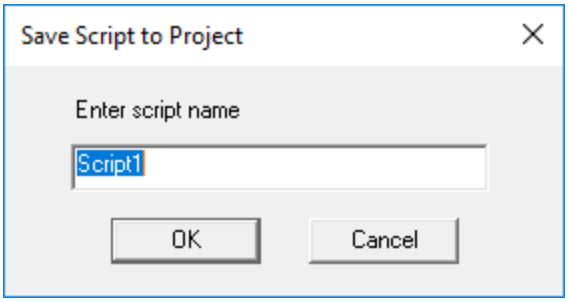
Recording a Script to a Project

To record a script to a project:

1. Click **Tools** > **Record Script to Project**, or select the **Automation** tab and use the **Record Script** drop-down menu to select **Record Script to Project**.



The **Save Script to Project** dialog box appears:



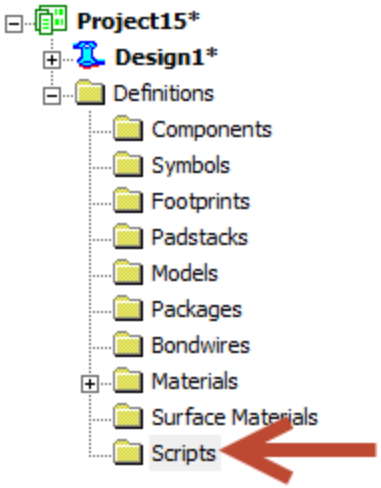
2. Enter a name for the script in the text box, then click **OK**.
3. Perform the steps you want to record.
4. When you have finished, click **Stop Recording**, or select **Tools** > **Stop Script Recording**.

   The recorded script is saved to *scriptname.vbs* in the Scripts library and can be accessed from the Project Manager. See: Working with Project Scripts.

Working with Project Scripts

Scripts can be recorded to a project.

Once a script has been recorded to the project, you can manage it in the **Project Manager** from the **Definitions** folder:



Individual scripts appear in this folder. Right-click a script to edit or run it:

You can also run project scripts from the **Automation** tab by selecting **Run Script** > **Project Scripts** > [**Script Name**].

> **Note:**
>
> Project scripts are stored in the project scripts library. Refer to the topic "Managing Library Contents" for information on working with libraries.

Executing a Script from Within a Script

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

In IronPython:

```
oDesktop.RunScript(<ScriptName>)
```

In VBscript:

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

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

1. Personal Library Directory (PersonalLib)
2. User Library Directory (UserLib)
3. System Library Directory (SysLib)
4. Installation Directory

Each of the library directories can be specified in Electronics Desktop under **Tools** > **Options** > **General Options**, on the **Project Options** tab.

Electronics Desktop Scripting Conventions

A number of scripting conventions exist for Electronics Desktop regarding syntax, arguments, and numerical values.

Consult the following topics:

- Named Arguments
- Setting Numerical Values
- Event Callback Scripting

Named Arguments

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

1. Named data, where name precedes data.

   For example:

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

2. Named Array, where name precedes array.

   For example:

   ```
   ..., "Attributes:=", Array(...),...
   ```

3. Named Array, where name is inside an array.

   For example:

   ```
   ..., Array("NAME:Attributes",...),...
   ```

In the first and second examples, the name is formatted as `"<Name>:="`. This signals to Electronics Desktop that this is a name for the next argument in the script command. In the third example, the name is formatted as `"NAME:<name>"` and is the first element of the array.

The names are used both to identify what the data means to you and to inform Electronics Desktop which data is being given. The names must be included or the script will not play back correctly. However, if you are writing a script, you do not need to pass in every piece of data that

the command can take. For example, if you are modifying a boundary, the script will be recorded to include every piece of data needed for the boundary, whether or not it was modified. If you are writing a script by hand, you can add only the data that changed and omit anything that you do not want to change. Electronics Desktop will use the names to determine which data you provided.

VBscript Example

For example, when editing an impedance boundary, Electronics Desktop records the `EditImpedance` command as follows in VBscript:

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

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

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

IronPython Example

When editing a port excitation, Electronics Desktop records the `Edit` command as follows in IronPython:

```
oModule.Edit("Port1",
  ["NAME:Port1",
    ["NAME:Properties",
    "PortSolver:=", "true",
    "Phase:=", "0deg",
    "Magnitude:=", "2mA",
    "Impedance:=", "50Ohm",
    "Theta:=", "0deg",
    "Phi:=", "0deg",
    "PostProcess:=", "false",
    "Renormalize:=", "50Ohm + 0i Ohm",
    "Deembed:=", "0mm",
    "RefToGround:=", "false"
    ],
  "Type:=", "EdgePort",
```

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

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

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

Setting Numerical Values

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

- Pass in the number directly.

  In IronPython:

  ```
  oModule.EditVoltage('Voltage1', ['NAME:Voltage1', 'Voltage:=',
  3.5])
  ```

  In VBscript:

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

- Pass in a string containing the number with units.

  In IronPython:

  ```
  oModule.EditVoltage('Voltage1', ['NAME:Voltage1', 'Voltage:=',
  '3.5V'])
  ```

  In VBscript:

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

- Pass in a variable name.

  In IronPython:

  ```
  var = 3.5

  oModule.EditVoltage('Voltage1', ['NAME:Voltage1', 'Voltage:=',
  var]
  ```

  In VBscript:

  ```
  dim var

  var = "3.5V"

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

Event Callback Scripting

Event Callback scripting allows you to define custom JavaScript and VBScript routines that will run automatically after detecting a triggering event, such as placing a component or running a simulation. When defining an Event Callback script, specify one or more scripts that will be run after a particular event is detected.

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

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

Callback Event	**Scripts Visible from the Event Callback Script**	**Functions Callable from the Visible Script**
Place Component	CompInstance	**CompInstance.GetParentDesign()** — Returns a oDesign item that can be used to call Design functions. **CompInstance.GetPropserverName()** – Returns a CompInstance identification name that can be used in oEditor property-method scripts, such as GetPropertyValue(), SetPropertyValue(), etc. **CompInstance.GetComponentName()** — Returns the component name, e.g. "MS_TRL".

Simulate Component	CompInstance	**CompInstance.GetDesign()** — Returns the interface to the specified design simulation. **CompInstance.GetProgress()** – Returns the completion percentage (from 0 to 100) of the specified design simulation. **CompInstance.GetRunStatus()** — Returns the status number of the specified design simulation. **CompInstance.Abort()** — Aborts the specified design simulation.

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

2 - Application Object Script Commands

The Application object commands permit you to get the AppDesktop. Application object commands should be executed by the oAnsoftApp object.

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

General Application Script Commands

The following are general script commands recognized by the **oAnsoftApp** object:

- GetAppDesktop

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

- GetDesiredRamMBLimit (deprecated)
- GetHPCLicenseType (deprecated)
- GetMaximumRamMBLimit (deprecated)
- GetMPISpawnCmd(deprecated)
- GetMPIVendor (deprecated)
- GetNumberOfProcessors (deprecated)
- GetUseHPCForMP (deprecated)
- SetDesiredRamMBLimit (deprecated)
- SetHPCLicenseType (deprecated)
- SetMaximumRamMBLimit (deprecated)
- SetMPISpawnCmd (deprecated)
- SetMPIVendor (deprecated)
- SetNumberOfProcessors (deprecated)
- SetUseHPCForMP (deprecated)

GetAppDesktop

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

UI Access	NA
Parameters	Name | Type | Description None | |
Return Value	Object

Python Syntax	GetAppDesktop()
Python Example	`oDesktop = oAnsoftApp.GetAppDesktop()`

VB Syntax	GetAppDesktop()
VB Example	`Set oDesktop = oAnsoftApp.GetAppDesktop()`

3 - Desktop Object Script Commands

Desktop commands should be executed by the oDesktop object. Some new commands permit you to query objects when you do not know the names. See: Object Oriented Property Scripting.

```
Set oDesktop =

  CreateObject("Ansoft.ElectronicsDesktop")

  oDesktop.CommandName <args>
```

SetTempDirectory

ShowDockingWindow

Sleep

StopSimulations

SubmitJob

TileWindows

Related Topics:

Desktop Commands For Registry Values

ImportExport Tool Commands

AddMessage

Add a message with severity and context to message window.

UI Access	N/A		
Parameters	Name	Type	Description
	<projectName>	String	Project name. Passing an empty string adds the message as the desktop global message.
	<designName>	String	Design name. Ignored if project name is empty. Passing an empty string adds the message to project node in the message tree.
	<severity>	Integer	One of "Error", "Warning" or "Info". Anything other than the first two is treated as "Info" 0 = Informational, 1 = Warning, 2 = Error, 3 = Fatal
	<msg>	String	The message for the message window.
	<category>	String	Optional. The category is created with the message under the design tree node if the category does not exist. If the category already exists, the new mes-

			sage is added to the end of the existing category. It is ignored if the project or design is empty. If missing or empty, the message is added to the Design node in the message tree.
Return Value	None.		

Python Syntax	AddMessage(<*projectName*>, <*designName*>, <*severity*>, <*msg*>, <*category*>)
Python Example	`oDesktop.AddMessage("Project1", "", 0, "This is a test message", "")`

VB Syntax	AddMessage <*projectName*>, <*designName*>, <*severity*>, <*msg*>, <*category*>
VB Example	`oDesktop.AddMessage "Project1", "1", 0, "This is a test message", ""`

ClearMessages

For a specified project and design, this command clears all messages in a specified severity range. The user-specified integral severity level is interpreted as:

```
0 => info, 1 => warning, 2 => error, and 3 => fatal error.
```

The ClearMessages function accepts four input arguments. The first two specifies project and design, respectively. This function clears messages in the range specified by the third (interpreted as start severity) and fourth (interpreted as stop severity) arguments. The fourth argument has a default value of 0. The last two arguments need not be specified in any given order, i.e., they can indicate either ascending or descendingtrend. The function clears all messages having severity level within this specified range. The start and stop severity need not be specified in any given order (i.e., they can indicate either ascending or descending severity).

<table>
<tr><td>UI Access</td><td>In Message Manager, right-click and select Clear messages for [ProjectName]...</td></tr>
<tr><td>Parameters</td><td>
<table>
<tr><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td><projectName></td><td>String</td><td>Name of the project from which to clear messages.</td></tr>
<tr><td><designName></td><td>String</td><td>Name of the design under the <projectName> from which to clear messages.</td></tr>
<tr><td><startSeverity></td><td>Integer</td><td>User-specified severity level at which messages start getting cleared:
0 = informational
1 = warning
2 = error
3 = fatal error</td></tr>
<tr><td><stopSeverity></td><td>Integer</td><td>Optional. User-specified stop severity level until which messages will be cleared.
0 = informational
1 = warning
2 = error
3 = fatal error
If not specified, <stopSeverity> has a default value of 0.</td></tr>
</table>
</td></tr>
<tr><td>Return Value</td><td>None.</td></tr>
</table>

<table>
<tr><td>Python Syntax</td><td>ClearMessages(<<i>projectName</i>>, <<i>designName</i>>, <<i>startSeverity</i>>, <<i>stopSeverity</i>>)</td></tr>
<tr><td>Python Example</td><td>Preserve Current Functionality; stopSeverity = 0 (DEFAULT VALUE)
<pre>
# startSeverity = 0; clears just the Info messages
oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor", "Maxwell3DDesign1", 0)
# startSeverity = 1; clears all the Info and Warning messages
</pre></td></tr>
</table>

```
oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor", "Maxwell3DDesign1", 1)

# startSeverity = 2; clears all the Info, Warning, Error

oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor", "Maxwell3DDesign1", 2)

# startSeverity = 3; clears all messages, i.e., Info, Warning, Error, Fatal

oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor", "Maxwell3DDesign1", 3)
```

Extend the current functionality by enabling range-based deletion

```
# startSeverity = 0; clears the Info messages;

oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor", "Maxwell3DDesign1", 0)

# startSeverity = 0 and stopSeverity = 0; clears all the Info messages

oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor", "Maxwell3DDesign1", 0,
0)

# startSeverity = 0 and stopSeverity = 1; clears all the Info and Warning mes-
sages

oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor", "Maxwell3DDesign1", 0,
1)

# startSeverity = 0 and stopSeverity = 2; clears all the Info, Warning, and Error
messages

oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor", "Maxwell3DDesign1", 0,
2)

# startSeverity = 0 and stopSeverity = 3; clears all the Info, Warning, Error,
and Fatal messages

oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor", "Maxwell3DDesign1", 0,
3)
```

```
# startSeverity = 1; clears all the Info and Warning messages;
oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor","Maxwell3DDesign1",1)
# startSeverity = 1 and stopSeverity = 1; clears all the Warning messages
oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor","Maxwell3DDesign1",1, 1)
# startSeverity = 1 and stopSeverity = 2; clears all the Warning and Error mes-
sages
oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor","Maxwell3DDesign1",1, 2)
# startSeverity = 1 and stopSeverity = 3; clears all the Warning, Error, and
Fatal messages
oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor","Maxwell3DDesign1",1, 3)


# startSeverity = 2; clears all the Info, Warning, and Error messages;
oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor","Maxwell3DDesign1",2)
# startSeverity = 2 and stopSeverity = 2; clears all the Error messages
oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor","Maxwell3DDesign1",2, 2)
# startSeverity = 2 and stopSeverity = 3; clears all the Error and Fatal messages
oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor","Maxwell3DDesign1",2, 3)
# startSeverity = 2 and stopSeverity = 0; clears all the Info, Warning, and Error
messages
oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor","Maxwell3DDesign1",2, 0)
```

	# startSeverity = 3; clears all the Info, Warning, Error, and Fatal messages;
	oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor","Maxwell3DDesign1",3)
	# startSeverity = 3 and stopSeverity = 3; clears all the Fatal error messages
	oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor","Maxwell3DDesign1",3, 3)
	# startSeverity = 3 and stopSeverity = 0; clears all the Info, Warning, Error, and Fatal messages
	oDesktop.ClearMessages("Ex_EC_1_Asymmetrical_Conductor","Maxwell3DDesign1",3, 0)

VB Syntax	ClearMessages <*projectName*>, <*designName*>, <*startSeverity*>, <*stopSeverity*>
VB Example	Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop") Set oDesktop = oAnsoftApp.GetAppDesktop() oDesktop.ClearMessages "MyProject", "MyDesign", 0, 3

CloseAllWindows

Closes all MDI child windows on the desktop.

UI Access	From main menu, **Window** > **CloseAll**.
Parameters	None.
Return Value	None.

Python Syntax	CloseAllWindows()
Python Example	`oDesktop.CloseAllWindows()`

VB Syntax	CloseAllWindows
VB Example	`oDesktop.CloseAllWindows`

CloseProject

Closes a specified project. Changes to the project are not saved. Save the project using the Project command **Save** or **Save As** before closing to save changes.

UI Access	**File** > **Close**
Parameters	Name | Type | Description <ProjectName> | String | The name of the project already in the Desktop that is to be closed, without path or extension
Return Value	None

Python Syntax	CloseProject (<*ProjectName*>)
Python Example	`oDesktop.CloseProject("MyProject")`

VB Syntax	CloseProject <*ProjectName*>
VB Example	`oDesktop.CloseProject "MyProject"`

CloseProjectNoForce

Use: Close a named project currently open in the Desktop, unless a simulation is running. Changes to the project will not be saved. Save the project using the Project command **Save** or **Save As** before closing to save changes. To determine if the project has been closed, use `GetProjectList` and see if the named project is present.

UI Access	**File** > **Close**		
Parameters	Name	Type	Description
	<ProjectName>	String	The name of the project already on the Desktop that is to be closed, without path or extension
Return Value	None		

Python Syntax	CloseProjectNoForce (<*ProjectName*>)
Python Example	`oDesktop.CloseProjectNoForce("MyProject")`

VB Syntax	CloseProjectNoForce <*ProjectName*>
VB Example	`oDesktop.CloseProjectNoForce "MyProject"`

Count

Gets the total number of queried projects or designs obtained by GetProjects() and GetDesigns() commands.

UI Access	N/A
Parameters	None.
Return Value	Integer
Python Syntax	Count
Python Example	`projects = oDesktop.GetProjects()` `numprojects = projects.Count`

VB Syntax	Count

VB Example	`Dim oAnsoftApp`
	`Dim oDesktop`
	`Dim oProject`
	`Dim oDesign`
	`Dim oEditor`
	`Dim oModule`
	`Dim oProjects`
	`Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")`
	`Set oDesktop = oAnsoftApp.GetAppDesktop()`
	`oDesktop.RestoreWindow`
	`Dim projects`
	`set projects = oDesktop.GetProjects()`
	`for i = 0 to projects.Count - 1`

	msgbox projects(i).GetName() dim designs set designs = projects(i).GetDesigns() for j = 0 to designs.Count - 1 msgbox designs(j).GetName() next next

DeleteProject

Deletes a project from disk.

UI Access	**Edit > Delete**
Parameters	Name | Type | Description <ProjectName> | String | Name of the project
Return Value	None

Python Syntax	DeleteProject (<*ProjectName*>)
Python Example	oDesktop.DeleteProject("MyProject")

VB Syntax	DeleteProject <*ProjectName*>
VB Example	`oDesktop.DeleteProject "MyProject"`

DownloadJobResults

This command is for downloading results from Ansys Cloud. Before using this script command, the command SelectScheduler() must be used first to select “ansys cloud” scheduler. This makes sure that current scheduler is Ansys Cloud, and user is logged in. Then, either a valid .q file or .q.completed file must be in the project folder, or, a valid job ID and a “batchinfo” folder containing the corresponding .jobid file are required in the project folder. When the download requirements are met, the command downloads results from Ansys Cloud using the specified filters to the given folder.

UI Access	Select Scheduler
Parameters	

Name	Type	Description
<jobID>	String	Provide the job ID of the target job. The job ID must be able to be found in current .q (or .q.completed) file, or inside the “batchinfo” folder in projectPath. If the job ID is empty, the job ID in current .q (or .q.completed) file will be used.
<projectPath>	String	A string of path to locate the project folder. The project file may be not necessary, but .q (or .q.completed) file or “batchinfo” folder with valid .jobid files are required.
<resultPath>	String	A string giving the folder path for the download to save to.
<Filters> (optional)	String	A string containing filters to download. The delimiter of file types is “;”. If no filter specified, the default filter “*” will be applied, which requests all files for download.

Return Value	A Boolean result about download complete or not

Python Syntax	DownloadJobResults(*jobID*, *projectPath*, *resultsPath*, *filters*)
Python Example	`boolDownloadCompleted = oDesktop.DownloadJobResults("01234567890123456789", "C:\\projects\\basic.aedt", "C:\\projects\\DownloadResults\\", "*")`

VB Syntax	DownloadJobResults(*jobID*, *projectPath*, *resultsPath*, *filters*)
VB Example	`boolDownloadCompleted = oDesktop. DownloadJobResults ("01234567890123456789", "C:\\projects\\basic.aedt", "C:\\projects\\DownloadResults\\", "*")`

EnableAutoSave

Enable or disable autosave feature.

UI Access	N/A
Parameters	Name | Type | Description <*enable*> | Boolean | True to enable autosave; False disables it.
Return Value	None.

Python Syntax	EnableAutoSave(<*enable*>)
Python Example	`oDesktop.EnableAutoSave(True)`

VB Syntax	EnableAutoSave <*enable*>
VB Example	`oDesktop.EnableAutoSave True`

ExportOptionsFiles

Copies the options config files to the *DestinationDirectory*.

UI Access	**Tools** > **Options** > **Export Options Files ...**
Parameters	Name | Type | Description <DestinationDirectory> | String | The path to the destination directory.
Return Value	None

Python Syntax	ExportOptionsFiles(*<DestinationDirectory>)*
Python Example	`oDesktop.ExportOptionsFiles("D:/test/export/")`

VB Syntax	ExportOptionsFiles *<DestinationDirectory>*
VB Example	`oDesktop.ExportOptionsFiles"D:/test/export/"`

GetActiveProject

Obtains the project currently active in the Desktop, as an object.

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

UI Access	N/A

Parameters	None.
Return Value	Object: the project that is currently active in the desktop.

Python Syntax	GetActiveProject()
Python Example	`oProject = oDesktop.GetActiveProject()`

VB Syntax	GetActiveProject
VB Example	`Set oProject = oDesktop.GetActiveProject`

GetAutoSaveEnabled

Checks whether the autosave feature is enabled.

UI Access	N/A
Parameters	None.
Return Value	Integer: • 1 – Autosave is enabled. • 0 – Autosave is not enabled.

Python Syntax	GetAutoSaveEnabled()
Python Example	`Enabled = oDesktop.GetAutoSaveEnabled()`

VB Syntax	GetAutoSaveEnabled
VB Example	`Enabled = oDesktop.GetAutoSaveEnabled()`

GetBuildDateTimeString

Returns a string representing the build date and time of the product;

UI Access	N/A
Parameters	None.
Return Value	String build date and time, in the format: year-month-day hour:minute:second. Example: 2019-01-18 21:59:33

Python Syntax	GetBuildDateTimeString()
Python Example	`oDesktop.GetBuildDateTimeString()`

VB Syntax	GetBuildDateTimeString
VB Example	`dnt = oDesktop.GetBuildDateTimeString`

GetDefaultUnit

Returns the default unit for a physical quantity.

<table>
<tr><td>UI Access</td><td>Tools > Options > General Options > Default Units. Note that this menu only displays units that can be changed, while the script can be used to view additional default units.</td></tr>
<tr><td>Parameters</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td><type></td><td>String</td><td>String containing a type of measurement.
Valid strings are (case insensitive):
"Acceleration"
"Angle"
"AngularAcceleration"
"AngularDamping"
"AngularSpeed"
"Capacitance"
"Conductance"
"Current"
"CurrentChangeRate"
"DataRate"
"DeltaH" (Magnetic Field Strength)
"Density"
"Flux"
"Force"
"Frequency"</td></tr>
</table>
</td></tr>
</table>

			• "Inductance" • "Length" • "MagneticReluctance" • "Mass" • "MassFlowRate" • "MomentInertia" • "Power" • "Pressure" • "PressureCoefficient" • "Resistance" • "SaturateMagnetization" (Magnetic Inductance) • "Speed" • "Temperature" • "Time" • "Torque" • "Voltage" • "VoltageChangeRate" • "Volume" • "VolumeFlowPerPressureRoot" • "VolumeFlowRate"
Return Value	String containing the default unit (for example, "mm").		

Python Syntax	GetDefaultUnit(<type>)
Python Example	`oDesktop.GetDefaultUnit("Length")`

VB Syntax	GetDefaultUnit <type>
VB Example	`oDesktop.GetDefaultUnit("Length")`

GetDesigns

Obtains all designs in the current project.

UI Access	N/A
Parameters	None.
Return Value	List of objects for all designs in the project.

Python Syntax	GetDesigns()
Python Example	`oProject.GetDesigns()`

VB Syntax	GetDesigns
VB Example	`oProject.GetDesigns`

GetDistributedAnalysisMachines

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

UI Access	N/A
Parameters	None.
Return Value	Returns a collection of names of machines used for distributed analysis.

Python Syntax	GetDistributedAnalysisMachines()
Python Example	`oDesktop.GetDistributedAnalysisMachines()`

VB Syntax	GetDistributedAnalysisMachines
VB Example	`oDesktop.GetDistributedAnalysisMachines()`

GetDistributedAnalysisMachinesForDesignType

To obtain a list of the machines set up for analysis of the specified design type.

UI Access	NA
Parameters	Name | Type | Description <designTypeName> | String | The name of the type of design, such as "Twin Builder", "HFSS", HFSS-IE", "Maxwell 3D" , "Maxwell 2D", "RMxprt", "EM Design", "Circuit", "System", "Q3D Extractor", "2D Extractor"

Return Value	Object; returns a collection of machine names.

Python Syntax	GetDistributedAnalysisMachinesForDesignType (<designTypeName>)

Python Example	`machineNames =oDesktop.`
	`GetDistributedAnalysisMachinesForDesignType`
	`("")`

VB Syntax	GetDistributedAnalysisMachinesForDesignType <designTypeName>

VB Example	`Set machineNames =oDesktop.`
	`GetDistributedAnalysisMachinesForDesignType`
	`("")`

GetExeDir

Returns the path where the executable is located.

UI Access	N/A
Parameters	None.
Return Value	String path where executable is located. Example: 'C:/Program Files/AnsysEM/v242/Win64/'

Python Syntax	GetExeDir()
Python Example	`oDesktop.GetExeDir()`

VB Syntax	GetExeDir
VB Example	`oDesktop.GetExeDir`

GetGDIObjectCount

> **Note:**
> This command is for internal Ansys use only.

Python Syntax	GetGDIObjectCount()
Python Example	`oDesktop.GetGDIObjectCount()`

VB Syntax	GetGDIObjectCount()
VB Example	`oDesktop.GetGDIObjectCount()`

GetLibraryDirectory

Get the path to the `SysLib` directory.

UI Access	NA

Parameters	Name | Type | Description None | |
Return Value	String The path to the SysLib directory.

Python Syntax	GetLibraryDirectory()
Python Example	`AddInfoMessage(str(oDesktop.GetLibraryDirectory()))`

VB Syntax	GetLibraryDirectory
VB Example	`MsgBox oDesktop.GetLibraryDirectory`

VB Example:

```
------------------------------------------------------
message box returns the path in this example
------------------------------------------------------
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
```

```
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
libdir = oDesktop.GetLibraryDirectory
msgbox(oDesktop.GetLibraryDirectory())
```

GetLocalizationHelper

> **Note:**
> This command is for internal Ansys use only.

Returns the object for the localization helper.

UI Access	NA
Parameters	Name | Type | Description None | |
Return Value	Object Localization helper object, such as "IDispatch(ILocalizationHelper)"

Python Syntax	GetLocalizationHelper()
Python Example	`oDesktop.GetLocalizationHelper()`

VB Syntax	GetLocalizationHelper
VB Example	`oDesktop.GetLocalizationHelper()`

GetMessages

Get the messages from a specified project and design.

UI Access	NA
Parameters	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td><*ProjectName*></td><td>String</td><td>Name of the project for which to collect messages. An incorrect project name results in no messages (design is ignored). An empty project name results in all messages (design is ignored)</td></tr><tr><td><*DesignName*></td><td>String</td><td>Name of the design in the named project for which to collect messages. An incorrect design name results in no messages for the named project. An empty design name results in all messages for the named project</td></tr><tr><td><*Severity*></td><td>Integer</td><td>Severity is 0-3, and is tied in to info/warning/error/fatal types as follows: 0 – info and above 1 – warning and above 2 – error and fatal 3 – fatal only (rarely used) </td></tr></table>
Return Value	Array of string messages.

Python Syntax	GetMessages (<*ProjectName*>, <*DesignName*>, <*Severity*>)
Python Example	`Messages = oDesktop.GetMessages("MyProject", "1", 1)`

VB Syntax	GetMessages <*ProjectName*>, <*DesignName*>, <*Severity*>
VB Examples	`Messages = oDesktop.GetMessages "MyProject", "1", 1`

GetName [Desktop]

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

UI Access	N/A
Parameters	None.
Return Value	String containing name.

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

VB Syntax	GetName()
VB Example	`----------`

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

GetPersonalLibDirectory

Get the path to the `PersonalLib` directory.

UI Access	N/A

Parameters	None.
Return Value	String path to the `PersonalLib` directory.

Python Syntax	GetPersonalLibDirectory()
Python Example	`oDesktop.GetPersonalLibDirectory()`

VB Syntax	GetPersonalLibDirectory
VB Example	`oDesktop.GetPersonalLibDirectory`

GetProcessID

Returns the process ID of ansysedt.exe.

UI Access	N/A
Parameters	None.
Return Value	Integer process ID of ansysedt.exe. For example, 12716.

Python Syntax	GetProcessID ()
Python Example	`oDesktop.GetProcessID()`

VB Syntax	GetProcessID
VB Example	`oDesktop.GetProcessID`

GetProjects

Returns a list of all the projects that are currently open in Electronics Desktop. Once you have the projects, you can iterate through them using standard VBScript methods.

UI Access	N/A
Parameters	None.
Return Value	Returns a collection containing objects for all open projects in Electronics Desktop.

Python Syntax	GetProjects()
Python Example	`oDesktop.GetProjects()`

VB Syntax	GetProjects
VB Example	`oDesktop.GetProjects`

GetProjectDirectory

Gets the path to the Project directory.

UI Access	N/A
Parameters	None.
Return Value	String path to the Project directory.

Python Syntax	GetProjectDirectory()
Python Example	`oDesktop.GetProjectDirectory()`

VB Syntax	GetProjectDirectory
VB Example	`oDesktop.GetProjectDirectory`

GetProjectList

Returns a list of all projects that are open in Electronics Desktop.

UI Access	N/A
Parameters	None.
Return Value	Array of strings containing the names of all open projects in Electronics Desktop.

Python Syntax	GetProjectList()
Python Example	`list_of_projects = oDesktop.GetProjectList()`

VB Syntax	GetProjectList
VB Example	`list_of_projects = oDesktop.GetProjectList`

GetScriptingToolsHelper

> **Note:**
> This command is for internal Ansys use only.

Returns the object for the scripting tools helper.

UI Access	NA
Parameters	Name | Type | Description None | |
Return Value	Object ScriptingTools helper object

Python Syntax	GetScriptingToolsHelper()
Python Example	`oDesktop.GetScriptingToolsHelper()`

VB Syntax	GetScriptingToolsHelper

VB Example	`oDesktop.GetScriptingToolsHelper()`

GetSysLibDirectory

Get the path to the `SysLib` directory.

UI Access	N/A
Parameters	None.
Return Value	String path to the `SysLib` directory.

Python Syntax	GetSysLibDirectory()
Python Example	`oDesktop.GetSysLibDirectory()`

VB Syntax	GetSysLibDirectory
VB Example	`oDesktop.GetSysLibDirectory`

GetTempDirectory

Gets the path to the `Temp` directory.

UI Access	N/A
Parameters	None.
Return Value	String path to the `Temp` directory.

Python Syntax	GetTempDirectory()
Python Example	`oDesktop.GetTempDirectory()`

VB Syntax	GetTempDirectory
VB Example	`oDesktop.GetTempDirectory`

GetUserLibDirectory

Gets the path to the UserLib directory.

UI Access	N/A
Parameters	None.
Return Value	Stringpath to the `UserLib` directory.

Python Syntax	GetUserLibDirectory()
Python Example	`oDesktop.GetUserLibDirectory()`

VB Syntax	GetUserLibDirectory

VB Example	`oDesktop.GetUserLibDirectory`

GetVersion

Returns a string representing the version.

UI Access	N/A
Parameters	None.
Return Value	String containing version of the product.

Python Syntax	GetVersion()
Python Example	`oDesktop.GetVersion()`

VB Syntax	GetVersion()
VB Example	`oDesktop.GetVersion`

ImportANF

Imports an ANF file into a new project. For older ANFv2 files, use ImportANFv2.

UI Access	**File** > **Import** > **ANF**.
Parameters	Name | Type | Description *<ANFfilename>* | String | Full path of ANF file.
Return Value	None.

Python Syntax	ImportANF(<*ANFfilename*>)

Python Example	`oDesktop.RestoreWindow()`
	`Set oTool = oDesktop.GetTool('ImportExport')`
	`oTool.ImportANF('C:\\AnsTranslator\\results\\package4.anf')`

VB Syntax	ImportANF <*ANFfilename*>

VB Example	`Dim oAnsoftApp`
	`Dim oDesktop`
	`Dim oProject`
	`Dim oDesign`
	`Dim oEditor`
	`Dim oModule`
	`Dim oProjects`
	`Dim omachine`
	`Dim oTool`
	`Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")`
	`Set oDesktop = oAnsoftApp.GetAppDesktop()`
	`oDesktop.RestoreWindow`

	Set oTool = oDesktop.GetTool("ImportExport") oTool.ImportANF("%UserProfile%\Documents\HFSS Examples\package.anf")

ImportAutoCAD

Imports an AutoCAD file into a new project.

UI Access	**File** > **Import** > **AutoCAD**.
Parameters	<table><tr><td>Name</td><td>Type</td><td>Description</td></tr><tr><td><*dxfFileName*></td><td>String</td><td>Full path of DXF file.</td></tr><tr><td><*outputPathFileName*></td><td>String</td><td>Full path of EDB file to create during import.</td></tr><tr><td><*controlFileName*></td><td>String</td><td>Full path of XML control file. Pass empty string if none.</td></tr></table>
Return Value	None.

Python Syntax	ImportAutoCAD (<*dxfFileName*>, <*outputPathFileName*>, <*controlFileName*>)
Python Example	Set oTool = oDesktop.GetTool('ImportExport') oTool.ImportAutoCAD('C:/MyPath/a4lines.dxf', 'C:/MyPath/a4lines.aedb.edb', 'C:/MyPath/a4lines.xml')

VB Syntax	ImportAutoCAD <*dxfFileName*>, <*outputPathFileName*>, <*controlFileName*>
VB Example	Set oTool = oDesktop.GetTool("ImportExport") oTool.ImportAutoCAD "C:/MyPath/a4lines.dxf", "C:/MyPath/a4lines.aedb.edb", "C:/MyPath/a4lines.xml"

ImportGDSII

Imports a GDSII file into a new project.

UI Access	**File** > **Import** > **GDSII**.
Parameters	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td><*gdsiiFileName*></td><td>String</td><td>Full path of GDSII file.</td></tr><tr><td><*outputPathName*></td><td>String</td><td>Full path of EDB file to create during import.</td></tr><tr><td><*controlFileName*></td><td>String</td><td>Optional. Full path of XML control file. Full path of “technology” (corresponding to `-t=technologyfile` argument in anstranslator). Pass empty string if none.</td></tr><tr><td><*mapFileName*></td><td>String</td><td>If technology file was used in place of the control file. Full path to either gdsii mapping file (corresponding to `-g=gdsMappingFile` argument in anstranslator) or XML control file. Otherwise, pass empty string.</td></tr></table>
Return Value	None.

Python Syntax	ImportGDSII(<*gdsiiFileName*>, <*outputPathName*>, <*controlFileName*>, <*mapFileName*>)

Python Example	`oDesktop.RestoreWindow()`
	`Set oTool = oDesktop.GetTool('ImportExport')`
	`oTool.ImportGDSII('C:/Files/test.gds', 'C:/Files/test.aedb',`
	`'C:/Files/test.ircx', 'C:/Files/test.xml')`

VB Syntax	ImportGDSII <*gdsiiFileName*>, <*outputPathName*>, <*controlFileName*>, <*mapFileName*>

VB Example	`Set oTool = oDesktop.GetTool("ImportExport")`
	`oTool.ImportGDSII "C:/Files/test.gds", "C:/Files/test.aedb.edb", _`
	`"C:/Files/test.xml"`

ImportODB

Imports an ODB++ file into a new project.

UI Access	**File** > **Import** > **ODB++**.		
Parameters	Name	Type	Description
	<*odbFileName*>	String	Full path of ODB++ file.
	<*outputPathName*>	String	Full path of EDB file to create during import.
	<*controlFileName*>	String	Optional. Full path of XML control file. Pass empty string if none.
Return Value	None.		

Python Syntax	ImportODB(<*odbFileName*>, <*outputPathName*>, <*controlFileName*>)

Python Example	`oDesktop.RestoreWindow()`
	`Set oTool = oDesktop.GetTool('ImportExport')`
	`oTool.ImportODB('C:/Files/test.odb', 'C:/Files/test.aedb',`
	`'C:/Files/test.xml')`

VB Syntax	ImportODB <*odbFileName*>, <*outputPathName*>, <*controlFileName*>

VB Example	`Set oTool = oDesktop.GetTool("ImportExport")`
	`oTool.ImportODB "C:/Files/test.odb", "C:/Files/test.aedb", _`
	`"C:/Files/test.xml`

LaunchJobMonitor

Use: For use in starting job monitoring. This brings up the **Monitor Job** dialog box.

UI Access	Launch Job Monitor		
Parameters	Name	Type	Description
	<projectPath>	String	Path to the project file to be monitored
Return Value	None		

Python Syntax	LaunchJobMonitor()
Python Example	`oDesktop.LaunchJobMonitor("C:\\projects\\basic.aedt")`

VB Syntax	LaunchJobMonitor()
VB Example	`oDesktop.LaunchJobMonitor("C:\\projects\\basic.aedt")`

NewProject

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

UI Access	**File** > **New**.
Parameters	None.
Return Value	Object, the project that is added.

Python Syntax	NewProject()
Python Example	`oProject = oDesktop.NewProject()`

VB Syntax	NewProject
VB Example	`Set oProject = oDesktop.NewProject`

OpenAndConvertProject

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

UI Access	Click **File** > **Open**, and choose a legacy project
Parameters	

Name	Type	Description
<itemPath>	String	full project path of the legacy project
<legacyChoice>	Integer	0: show conversion dialog box, (same as **File** > **Open** of a legacy file) 1: rename (changes extension to .aedt, the original file and results are renamed) 2: copy (creates new file with .aedt extension, and the original file and results remain available)

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

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

Python Syntax	OpenAndConvertProject(*filePath*, *legacyChoice*)

Python Example	`oProject = oDesktop.OpenAndConvertProject("c:\files\optimtee.hfss", 1)`
	Note: optimtee.hfss is gone after this code executes
	`oProject = oDesktop.OpenAndConvertProject("c:\files\optimtee.hfss", 2)`
	Note: optimtee.hfss remains after this code executes

VB Syntax	OpenAndConvertProject(*filePath*, *legacyChoice*)
VB Example	`Set oProject = oDesktop.OpenAndConvertProject("c:\files\optimtee.hfss", 1)` **Note:** optimtee.hfss is gone after this code executes

OpenMultipleProjects

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

UI Access	N/A
Parameters	

Name	Type	Description
<directory>	String	Path to the projects.
<fileType>	String	Type of projects to open.

Return Value	None.

Python Syntax	OpenAndConvertProject(<*filePath*>, <*legacyChoice*>)
Python Example	`oProject = oDesktop.OpenAndConvertProject("c:\files\optimtee.", "*.aedt")`

VB Syntax	OpenAndConvertProject <*filePath*>, < *legacyChoice*>
VB Example	`Set oProject = oDesktop.OpenAndConvertProject "c:\files\optimtee.", "*.aedt"`

OpenProject

Opens a specified project.

UI Access	Click **File** > **Open**.
Parameters	Name | Type | Description <*FileName*> | String | Full path of the project to open.
Return Value	An object reference to the newly opened project.

Python Syntax	OpenProject(<*FileName*>)
Python Example	`oDesktop.OpenProject("D:/Projects/Project1.aedt")`

VB Syntax	OpenProject <*FileName*>

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

OpenProjectWithConversion

> **Note:**
> This command is for internal Ansys use only.

Python Syntax	OpenProjectWithConversion()
Python Example	`oDesktop.OpenProjectWithConversion()`

Paste (Project Object)

Pastes a design in the active project.

UI Access	**Edit** > **Paste**
Parameters	None
Return Value	None

Python Syntax	Paste()
Python Example	`oProject.Paste()`

VB Syntax	Paste
VB Example	`oProject.Paste`

PauseRecording

Temporarily stop script recording.

UI Access	NA		
Parameters	Name	Type	Description
	None		
Return Value	None		

Python Syntax	PauseRecording()
Python Example	`oDesktop.PauseRecording()`

VB Syntax	PauseRecording
VB Example	`oDesktop.PauseRecording`

PauseScript

Pause the execution of the script and pop up a message to the user. The script execution will not resume until the user chooses.

UI Access	**Tools** > **Pause Script**
Parameters	<table><tr><td>Name</td><td>Type</td><td>Description</td></tr><tr><td><*Message*></td><td>String</td><td>Any Text.</td></tr></table>
Return Value	None

Python Syntax	PauseScript (<*Message*>)
Python Example	`oDesktop.PauseScript("Text to display in pop-up dialog box.")`

VB Syntax	PauseScript <*Message*>
VB Example	`oDesktop.PauseScript "Text to display in pop-up dialog box."`

Print

Prints the contents of the active view window.

UI Access	**File** > **Print**.
Parameters	None.
Return Value	None.

Python Syntax	Print()
Python Example	`oDesktop.Print()`

VB Syntax	Print
VB Example	`oDesktop.Print`

QuitApplication

Exits the desktop.

UI Access	**File** > **Exit**.
Parameters	None.
Return Value	None.

Python Syntax	QuitApplication()
Python Example	`oDesktop.QuitApplication()`

VB Syntax	QuitApplication
VB Example	`oDesktop.QuitApplication`

RefreshJobMonitor

For use in monitoring a job.

UI Access	**Tools** > **Job Management** > **Monitor Jobs**.
Parameters	None.
Return Value	A string specifing the job state. The result can be any of the following strings: • "Monitor Not Visible" • "Queued" • "Running" • "Shutting Down" • "Unknown" • "Completed" • "Not Monitoring" • "Starting Monitoring"

Python Syntax	RefreshJobMonitor()
Python Example	`oDesktop.RefreshJobMonitor()`

VB Syntax	RefreshJobMonitor

VB Example	`oDesktop.RefreshJobMonitor`

ResetLogging

Redirects simulation log file to a specified directory and log level.

UI Access	N/A
Parameters	Name | Type | Description *<logFile>* | String | Path to log file. *<logLevel>* | Integer | Specified log level.
Return Value	None.

Python Syntax	ResetLogging(*<logFile>*, *<logLevel>*)
Python Example	`oDesktop.ResetLogging("C:/Project1.aedtresults/", 1)`

VB Syntax	ResetLogging *<logFile>*, *<logLevel>*
VB Examples	`oDesktop.ResetLogging "C:/Project1.aedtresults/", 1`

RestoreProjectArchive

Restores a previously archived project to a specified path.

UI Access	**File** > **Restore Archive**.

Parameters	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td><*ArchiveFilePath*></td><td>String</td><td>Path to archived file</td></tr><tr><td><*ProjectFilePath*></td><td>String</td><td>Path to restore location</td></tr><tr><td><*OverwriteExistingFiles*></td><td>Boolean</td><td>True to overwrite an existing file of the same name; else False.</td></tr><tr><td><*OpenProjectAfterRestore*></td><td>Boolean</td><td>True to open the project after it is restored; else False.</td></tr></table>
Return Value	None.

Python Syntax	RestoreProjectArchive (<*Archivefilepath*>, <*ProjectFilePath*>, <*OverwriteExistingFiles*>, <*OpenProjectAfterRestore*>)
Python Example	`oDesktop.RestoreProjectArchive("C:\Users\jdoe\Documents\OptimTee.aedtz", "C:\Documents\OptimTee.aedt", False, True)`

VB Syntax	RestoreProjectArchive <*Archivefilepath*>, <*ProjectFilePath*>, <*OverwriteExistingFiles*>, <*OpenProjectAfterRestore*>
VB Example	`oDesktop.RestoreProjectArchive "C:\Users\jdoe\Documents\OptimTee.aedtz", _` `"C:\Documents\OptimTee.aedt", false, true`

RestoreWindow

Restores a minimized Desktop window.

UI Access	N/A

Parameters	None.
Return Value	None.

Python Syntax	RestoreWindow()
Python Example	`oDesktop.RestoreWindow()`

VB Syntax	RestoreWindow
VB Example	`oDesktop.RestoreWindow`

ResumeRecording

Resume recording a script.

UI Access	N/A
Parameters	None.
Return Value	None

Python Syntax	ResumeRecording()
Python Example	`oDesktop.ResumeRecording()`

VB Syntax	ResumeRecording
VB Example	`oDesktop.ResumeRecording`

RunACTWizardScript

> **Note:**
> This command is for internal Ansys use only.

Python Syntax	RunACTWizardScript()
Python Example	`oDesktop.RunACTWizardScript()`

RunProgram

Runs an external program.

UI Access	NA		
Parameters	Name	Type	Description
	<ProgName>	String	Name of the program to run.
	<ProgPath>	String	Location of the program. Pass in an empty string to use the system path.
	<WorkPath>	String	Working directory in which program will start.
	<ArgArray>	Array of Strings	Arguments to pass to the program. If no arguments, pass in `None`
Return Value	None		

Python Syntax	RunProgram (<*ProgName*>, <*ProgPath*>, <*WorkPath*>, <*ArgArray*>)

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

VB Syntax	RunProgram <*ProgName*>, <*ProgPath*>, <*WorkPath*>, <*ArgArray*>

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

RunScript

Launches another script from within the script currently being executed.

UI Access	Tools>Run Script		
Parameters	Name	Type	Description
	<SCriptPath>	String	Name or full path of the script to execute. If the full path to the script is not specified, Twin Builder searches for the specified script in the following locations, in this order: 1. 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. 2. User library directory. This is the **userlib** subdirectory in the library directory. The library directory can be specified **General Options** dialog in the box (click **Tools** > **Options** > **General Options** to to open this dialog box) under the **Project Options** tab. 3. 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. 4. HFSS installation directory.
Return Value	Long the return code for the script method.		

Python Syntax	RunScript (<*ScriptPath*>)
Python Example	`oDesktop.RunScript("C:/Project/test1.vbs")`

VB Syntax	RunScript <*ScriptPath*>

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

RunScriptWithArguments

Similar to RunScript, launch another script from within the currently executing script, but with arguments.

UI Access	NA

Parameters

Name	Type	Description
<*ScriptPath*>	String	The name or full path of the script to execute. If the full path to the script is not specified, the software looks for the script in the following locations: • Personal library directory: "PersonalLib" The PersonalLib directory can be specified in **Tools** > **Options** > **General Options** on the 'Project Options' tab. • User library directory: "userlib" The UserLib directory can be specified in **Tools** > **Options** > **General Options** on the 'Project Options' tab. • System library directory: "syslib" The SysLib directory can be specified in **Tools** > **Options** > **General Options** on the 'Project Options' tab. • Software installation directory
<*Arguments*>	String	The arguments to supply to the specified script.

Return Value	Long the return code for the script method.

Python Syntax	RunScriptWithArguments (<*ScriptPath*>, <*Arguments*>)
Python Example	`oDesktop.RunScriptWithArguments` `("C:/Project/test2.py", "foo")`

VB Syntax	RunScriptWithArguments <*ScriptPath*>, <*Arguments*>
VB Example	`oDesktop.RunScriptWithArguments` `"C:/Project/test2.vbs", "foo"`

SelectScheduler

Selects the scheduler used for batch job submission. It tries non-graphical selection of the scheduler, attempting to get version information from the scheduler in order to check for successful selection. If unable to get the information, it displays the **Select Scheduler** window and waits for the user to complete the settings.

UI Access	**Tools** > **Job Management** > **Select Scheduler**.

Parameters

Name	Type	Description
<*option*>	String	One of the following options (not case sensitive): • Empty string for remote RSM service • "RSM" for local RSM • "Windows HPC" for Windows HPC • "LSF" for Load-Sharing Facility • "SGE" for Grid Engine (GE, OGE, SGE, UGE, etc.) • "PBS" for Portable Batch Scheduler/System (PBSPro, Torque, Maui, etc.)

			• "Ansys Cloud" for Ansys Cloud
	<address (optional)>	String	String specifying the IP address or hostname of the head node or for the remote host running the RSM service.
	<username (optional)>	String	Username string to use for remote RSM service (or blank to use username stored in current submission host user settings). If the (non-blank) username doesn't match the username stored in current submission host user settings, then the Select Scheduler dialog is displayed to allow for password entry prior to job submission.
	<forcePasswordEntry (optional)>	String	Boolean used to force display of the Select Scheduler GUI to allow for password entry prior to job submission.
Return Value	The selected scheduler (if selection was successful, this string should match the input option string, although it could differ in upper/lowercase).		

Python Syntax	Select Scheduler(*<option>, <address>, <username>, <forcePasswordEntry>*)
Python Example	`result = oDesktop.SelectScheduler("Windows HPC",` `"headnode.win.example.com")`

VB Syntax	Select Scheduler *<option>, <address>, <username>, <forcePasswordEntry>*
VB Example	`result = oDesktop.SelectScheduler "Windows HPC", _` `"headnode.win.example.com"`

SetActiveProject

Specifies the name of the project that should become active in the desktop. Returns that project.

UI Access	N/A
Parameters	Name | Type | Description <*ProjectName*> | String | The name of the project already in the Desktop that is to be activated.
Return Value	Object, the project that is activated.

Python Syntax	SetActiveProject (<*ProjectName*>)
Python Example	`oProject = oDesktop.SetActiveProject("Project1")`

VB Syntax	SetActiveProject <*ProjectName*>
VB Example	`Set oProject = oDesktop.SetActiveProject "Project1"`

SetActiveProjectByPath

Specifies the name of the project that should become active in the desktop. Returns that project. If a user has two projects open with the same name, the result of `SetActiveProject` is ambiguous (the first one listed in selected). This command permits unambiguous specification of the active project.

UI Access	N/A
Parameters	Name | Type | Description <*ProjectName*> | String | The full path name of the project already in the Desktop that is to be activated.
Return Value	Object, the project that is activated.

Python Syntax	SetActiveProjectByPath(<*ProjectName*>)
Python Example	`oProject = oDesktop.SetActiveProjectByPath("c:\Projects\MyProject.aedt")`

VB Syntax	SetActiveProjectByPath <*ProjectName*>
VB Example	`Set oProject = oDesktop.SetActiveProjectByPath "c:\Projects\MyProject.aedt"`

SetLibraryDirectory

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

UI Access	NA
Parameters	Name / Type / Description <DirectoryPath> / String / The path to the SysLib Directory
Return Value	None

Python Syntax	SetLibraryDirectory (<*DirectoryPath*>)
Python Example	`oDesktop.SetLibraryDirectory("c:\libraries")`

VB Syntax	SetLibraryDirectory <*DirectoryPath*>

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

SetProjectDirectory

Sets the project directory path.

UI Access	N/A
Parameters	Name | Type | Description *<DirectoryPath>* | String | The path to the project directory. This should be writeable by the user.
Return Value	None.

Python Syntax	SetProjectDirectory (*<DirectoryPath>*)
Python Example	`oDesktop.SetProjectDirectory("c:\projects")`

VB Syntax	SetProjectDirectory *<DirectoryPath>*
VB Example	`oDesktop.SetProjectDirectory "c:\projects"`

SetTempDirectory

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

UI Access	N/A

Parameters	<table><tr><td>Name</td><td>Type</td><td>Description</td></tr><tr><td><*DirectoryPath*></td><td>String</td><td>The path to the Temp directory. This should be writeable by the user.</td></tr></table>
Return Value	None.

Python Syntax	SetTempDirectory (<*DirectoryPath*>)
Python Example	`oDesktop.SetTempDirectory("c:\tmp")`

VB Syntax	SetTempDirectory <*DirectoryPath*>
VB Example	`oDesktop.SetTempDirectory "c:\tmp"`

ShowDockingWindow

Shows or hides a docking window.

UI Access	Right click docking window > Show/Hide.
Parameters	<table><tr><td>Name</td><td>Type</td><td>Description</td></tr><tr><td><*windowName*></td><td>String</td><td>The window name (for example, "Message Manager", "Component Libraries", "Properties")</td></tr><tr><td><*show*></td><td>Boolean</td><td>True to show; False to hide.</td></tr></table>
Return Value	None.

Python Syntax	ShowDockingWindow (<*windowName*>, <show>)
Python Example	`oDesktop.ShowDockingWindow('Message Manager',False)`

VB Syntax	ShowDockingWindow <*windowName*> <*show*>
VB Example	`oDesktop.ShowDockingWindow "Message Manager" False`

Sleep

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

UI Access	NA
Parameters	Name: <*TimeInMilliseconds*> — Type: Integer — Description: The time that the execution should be suspended in milliseconds
Return Value	None

Python Syntax	Sleep (<*TimeInMilliseconds*>)
Python Example	`oDesktop.Sleep(1000)`

VB Syntax	Sleep <*TimeInMilliseconds*>

VB Example	`oDesktop.Sleep 1000`

SubmitJob

Submits a batch job to a scheduler. When submitting the same project file multiple times, you should have the script wait for each job (or jobs for multi-step) to finish, which can be done via the monitoring functions LaunchJobMonitor() and RefreshJobMonitor(), checking the result of RefreshJobMonitor() in a loop until it returns completed ("Monitor Not Visible") status.

UI Access	**Tools** > **Job Management** > **Submit Job**.
Parameters	

Name	Type	Description
<*settingsPath*>	String	Path to the settings file (exported from the Submit Job GUI) to use for submission.
<*projectPath*>	String	Path to the project file to use in the batch job. This could be an archive (.aedtz file) or an un-archived project.
<*design (optional)*>	String	Name of the design to use for batch solve.
<*setup (optional)*>	String	Name of the specific setup to solve.

Return Value	Array of job ID strings (empty if no jobs submitted).

Python Syntax	SubmitJob(<*settingsPath*>, <*projectPath*>, <*design*>, <*setup*>)
Python Example	`jobIDs = oDesktop.SubmitJob("C:\\hpc-settings\\Submit_Job_Settings.areg", "C:\\projects\\basic.aedt"))` `moreIDs = oDesktop.SubmitJob("C:\\hpc-settings\\Submit_Job_Settings.areg", "C:\\projects\\basic.aedt", "Design1", "Setup1")`

VB Syntax	SubmitJob <*settingsPath*>, <*projectPath*>, <*design*>, <*setup*>
VB Example	`jobIDs = oDesktop.SubmitJob "C:\\hpc-settings\\Submit_ Job_Settings.areg", "C:\\projects\\basic.aedt" moreIDs = oDesktop.SubmitJob "C:\\hpc-settings\\Submit_ Job_Settings.areg", "C:\\projects\\basic.aedt", "Design1", "Setup1"`

TileWindows

Arrange all open windows in a tiled format.

UI Access	From main menu, **Window** >**Tile Horizontally** or **Window** >**Tile Vertically**.
Parameters	Name: <*TilingFlag*> — Type: Integer — Description: • 0 – Tile vertically. • 1 – Tile horizontally.
Return Value	None.

Python Syntax	TileWindows(<*TilingFlag*>)
Python Example	`oDesktop.TileWindows(0)`

VB Syntax	TileWindows <*TilingFlag*>

VB Example	`oDesktop.TileWindows 0`

Desktop Commands For Registry Values

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

For example, to set the DSO & HPC analysis setup for HFSS using a Python script:

1. Start .
2. Go to the DSO and HPC options and create a setup named "test".
3. Export the setup to a file (for example, c:\temp\test.acf).
4. Copy the exported file to a target PC (for example, f:\temp\test.acf).
5. Run the following script:

```
#import the setup

oDesktop.SetRegistryFromFile("f:\\temp\\test.acf")

# Set Active Setup to "test"

oDesktop.SetRegistryString("Desktop/ActiveDSOConfigurations/", "test")
```

See the following subtopics:

DeleteRegistryEntry

DoesRegistryValueExist

GetRegistryInt

GetRegistryString

SetRegistryFromFile

SetRegistryInt

SetRegistryString

DoesRegistryValueExist

Determines whether a registry value exists.

UI Access	N/A
Parameters	<table><tr><td>Name</td><td>Type</td><td>Description</td></tr><tr><td><*KeyName*></td><td>String</td><td>Full name of registry key, including path.</td></tr></table>
Return Value	Boolean: • **True** – Key exists. • **False** – Key does not exist.

Python Syntax	DoesRegistryValueExist(<*KeyName*>)
Python Example	`Exist = oDesktop.DoesRegistryValueExist('Desktop/ActiveDSOConfigurations/')`

VB Syntax	DoesRegistryValueExist(<*KeyName*>)
VB Example	`bExist = oDesktop.DoesRegistryValueExist("Desktop/ActiveDSOConfigurations/")`

GetRegistryInt

Obtains registry key integer value.

UI Access	N/A
Parameters	Name | Type | Description <*KeyName*> | String | Full name of registry key, including path.
Return Value	Integer if the integer value is found. Return as Bad-Argument-Value if registry key does not exist or it is not an integer value.

Python Syntax	GetRegistryInt(<*KeyName*>)
Python Example	`num = oDesktop.GetRegistryInt('Desktop/Set-tings/ProjectOptions//UpdateReportsDynamicallyOnEdits')`

VB Syntax	GetRegistryInt(<*KeyName*>)
VB Example	`num = oDesktop.GetRegistryInt("Desktop/Set-tings/ProjectOptions//UpdateReportsDynamicallyOnEdits")`

GetRegistryString

Obtains registry key string value.

UI Access	N/A
Parameters	Name | Type | Description <*KeyName*> | String | Full name of registry key, including path.
Return Value	String if the string value is found. Return as Bad-Argument-Value if registry key does not exist or it is not a string

	value.

Python Syntax	GetRegistryString(<*KeyName*>)
Python Example	`activeDSO = oDesktop.GetRegistryString('Desktop/ActiveDSOConfigurations/')`

VB Syntax	GetRegistryString(<*KeyName*>)
VB Example	`activeDSO = oDesktop.GetRegistryString("Desktop/ActiveDSOConfigurations/")`

SetRegistryFromFile

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

UI Access	N/A
Parameters	<table><tr><td>Name</td><td>Type</td><td>Description</td></tr><tr><td><*filePath*></td><td>String</td><td>Full file path of registry file.</td></tr></table>
Return Value	Success if configuration is imported. Bad argument value if the file is not found or does not contain valid analysis configuration data.

Python Syntax	SetRegistryFromFile(<*filePath*>)
Python Example	`oDesktop.SetRegistryFromFile('c:/temp/test.acf')`

VB Syntax	SetRegistryFromFile <*filePath*>
VB Example	`oDesktop.SetRegistryFromFile "c:/temp/test.acf"`

SetRegistryInt

Sets registry key to an integer value.

UI Access	N/A
Parameters	<table><tr><td>Name</td><td>Type</td><td>Description</td></tr><tr><td><*KeyName*></td><td>String</td><td>Full name of registry key, including path.</td></tr><tr><td><*int*></td><td>Integer</td><td>Integer value to be assigned to registry key.</td></tr></table>
Return Value	Success if the key is defined as an integer. Bad argument value if a key is not defined, or if the value is a text string.

Python Syntax	SetRegistryInt(<*KeyName*>, <*int*>)
Python Example	`oDesktop.SetRegistryInt('Desktop/Set-tings/ProjectOptions//UpdateReportsDynamicallyOnEdits', 0)`

VB Syntax	SetRegistryInt <*KeyName*> <*int*>
VB Example	`oDesktop.SetRegistryInt "Desktop/Set-tings/ProjectOptions//UpdateReportsDynamicallyOnEdits", 0`

SetRegistryString

Sets registry key to a string value.

UI Access	N/A
Parameters	Name | Type | Description <*KeyName*> | String | Full name of registry key, including path. <*value*> | String | String value to be assigned to registry key.
Return Value	Success if the key is defined as a text string. Bad argument value if the key is not defined or requires an integer value.

Python Syntax	SetRegistryString(<*KeyName*>, <*value*>)
Python Example	`oDesktop.SetRegistryString('Desktop/ActiveDSOConfigurations/', 'Local')`

VB Syntax	SetRegistryString <*KeyName*>, <*value*>
VB Example	`oDesktop.SetRegistryString "Desktop/ActiveDSOConfigurations/", "Local"`

4 - Running Instances Manager Script Commands

The Running Instances Manager is a scripting object that lets you identify and connect to all running instances of Electronics Desktop. oDesktop objects that are returned provide full scripting functionality. Running Instances Manager commands should be executed by the oDesktop object. For example:

```
Set oRunningInstances = oDesktop.GetRunningInstancesMgr()
```


GetAllRunningInstances

Returns a list of running instances of Ansys Electronics Desktop.

UI Access	N/A
Parameters	None.
Return Value	Array containing list of Ansys Electronics Desktop instances.

Python Syntax	GetAllRunningInstances()
Python Example	`obj = oRunningInstances.GetAllRunningInstances()`

VB Syntax	GetAllRunningInstances
VB Example	`set obj = oRunningInstances.GetAllRunningInstances()`

GetRunningInstanceByProcessID

Returns the instance of Ansys Electronics Desktop that is running a specified process.

UI Access	N/A
Parameters	Name | Type | Description <processID> | Integer | Process ID
Return Value	String of the returned instance.

Python Syntax	GetRunningInstanceByProcessID(<processID>)
Python Example	`obj = oRunningInstances.GetRunningInstanceByProcessID(12345)`

VB Syntax	GetRunningInstanceByProcessID <processID>
VB Example	`set obj = oRunningInstances.GetRunningInstanceByProcessID(12345)`

GetRunningInstancesMgr

Returns the object of the Running Instances Manager.

UI Access	N/A
Parameters	Name | Type | Description None	

Return Value	Object Running instances manager object

Python Syntax	GetRunningInstancesMgr()
Python Example	`oRunningInstances = oDesktop.GetRunningInstancecMgr()`

VB Syntax	GetRunningInstancesMgr()
VB Example	`Set oRunningInstances = oDesktop.GetRunningInstancesMgr()`

This page intentionally
left blank.

5 - Project Object Script Commands

Project commands should be executed by the oProject object.

One example of accessing this object is:

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

Dataset Script Commands:

AddDataset

DeleteDataset

EditDataset

ExportDataset

HasDataset

ImportDataset

Other Project Object Script Commands:

AnalyzeAll

ChangeProperty

ClearMessages

CloneMaterial

Close

CopyDesign

CutDesign

DeleteDesign

AddDataset

Adds a dataset. This can be executed by the oProject, or oDesign variables.

UI Access	**Project** > **Datasets** > **Add**.
Parameters	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td><*DatasetDataArray*></td><td>Array</td><td>Array("NAME:<DatasetName>", Array("NAME:Coordinates", <CoordinateArray>, <CoordinateArray>, ...)</td></tr><tr><td><*DatasetName*></td><td>String</td><td>Name of the dataset.</td></tr><tr><td><*CoordinateArray*></td><td>Array</td><td>Array("NAME:Coordinate", "X:=", <double>, "Y:=",<double>)</td></tr></table>
Return Value	None.

Python Syntax	AddDataset <*DatasetDataArray*>

Python Example

```
oProject.AddDataset(
[
"NAME:$ds1",
  [
  "NAME:Coordinates",
    [
      "NAME:Coordinate",
      "X:=", 2,
      "Y:=", 4
    ],
    [
      "NAME:Coordinate",
      "X:=", 6,
      "Y:=", 8
    ]
  ]
]
)
oDesign.AddDataset(
[
"NAME:$ds1",
```

	["NAME:Coordinates", ["NAME:Coordinate", "X:=", 2, "Y:=", 4], ["NAME:Coordinate", "X:=", 6, "Y:=", 8]]])

VB Syntax	AddDataset <*DatasetDataArray*>
VB Example	oProject.AddDatasetArray("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))) oDesign.AddDatasetArray("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)))

AnalyzeAll [project]

Runs the project-level script command from the script, which simulates all solution setups and Optimetrics setups for all design instances in the project. The UI waits until simulation is finished before continuing with the script.

UI Access	**Project** > **Analyze All**.
Parameters	None.
Return Value	None.

Python Syntax	AnalyzeAll()
Python Example	`oProject.AnalyzeAll()`

VB Syntax	AnalyzeAll
VB Example	`oProject.AnalyzeAll`

ClearMessages

Clears information in the **Messages** window.

> **Note:**
>
> Additional options are available when using the Desktop-level ClearMessages command.

UI Access	N/A
Parameters	None.
Return Value	None.

Python Syntax	ClearMessages()
Python Example	`oProject.ClearMessages()`

VB Syntax	ClearMessages
VB Example	`oProject.ClearMessages`

Close

Closes the active project.

> **Warning:**
>
> Unsaved changes will be lost.

UI Access	N/A
Parameters	None.
Return Value	None.

Python Syntax	Close()
Python Example	`oProject.Close()`

VB Syntax	Close
VB Example	`oProject.Close`

CopyDesign

Copies a specified design.

UI Access	**Edit** > **Copy**.
Parameters	Name | Type | Description

	<DesignName>	String	Name of the design to copy from.
Return Value	None.		

Python Syntax	CopyDesign (*<DesignName>*)
Python Example	`oProject.CopyDesign ("HFSSDesign1")`

VB Syntax	CopyDesign *<DesignName>*
VB Example	`oProject.CopyDesign "HFSSDesign1"`

CutDesign

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

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

UI Access	**Edit** > **Cut**.		
Parameters	Name	Type	Description
	<DesignName>	String	Name of the design.

Return Value	None.

Python Syntax	CutDesign (<*DesignName*>)
Python Example	`oProject.CutDesign("SimplorerDesign1")`

VB Syntax	CutDesign <*DesignName*>
VB Example	`oProject.CutDesign "SimplorerDesign1"`

DeleteDataset

Deletes a specified dataset. This can be executed by the oProject, or oDesign variables.

UI Access	**Project** > **Datasets** > **Remove**.
Parameters	Name / Type / Description <*DatasetName*> / String / Name of the dataset found in the project.
Return Value	None.

Python Syntax	DeleteDataset (<*DatasetName*>)
Python Example	`oProject.DeleteDataset('$ds1')` `oDesign.DeleteDataset('$ds1')`

VB Syntax	DeleteDataset <*DatasetName*>
VB Example	`oProject.DeleteDataset "$ds1"` `oDesign.DeleteDataset "$ds1"`

DeleteDesign

Deletes a specified design in the project.

UI Access	**Edit** > **Delete**, or **Delete** in the ribbon.
Parameters	Name | Type | Description <*DesignName*> | String | Name of the design.
Return Value	None.

Python Syntax	DeleteDesign (<*DesignName*>)
Python Example	`oProject.DeleteDesign("Design2")`

VB Syntax	DeleteDesign <*DesignName*>
VB Example	`oProject.DeleteDesign "Design2"`

DeleteToolObject

Note:
This command is for internal Ansys use only.

UI Access	N/A
Parameters	Name | Type | Description <*ObjectName*> | String | Name of the tool object.
Return Value	None.

Python Syntax	DeleteToolObject(<*ObjectName*>)
Python Example	`oProject.DeleteToolObject("object1")`

VB Syntax	DeleteToolObject <*ObjectName*>
VB Example	`oProject.DeleteToolObject "object1"`

EditDataset

Modifies a dataset. This can be executed by the oProject, or oDesign variables.

UI Access	**Project** > **Datasets** > **Edit**.
Parameters	Name | Type | Description <*OriginalName*> | String | Name of the original dataset.

	<DatasetDataArray>	Array	Data for the modified dataset.
Return Value	None.		

Python Syntax	EditDataset (*<OriginalName>* *<DatasetDataArray>*)

Python Example	`oProject.EditDataset ("ds1"`
	`["NAME:ds2",`
	`  ["NAME:Coordinates",`
	`  [`
	`  "NAME:Coordinate",`
	`  "X:=", 1, "Y:=", 2`
	`  ],`
	`  [`
	`  "NAME:Coordinate",`
	`  "X:=", 3, "Y:=", 4`
	`  ]`
	`  ]`
	`]`
	`)`
	`oDesign.EditDataset ("ds1"`

	["NAME:ds2",
	  ["NAME:Coordinates",
	  [
	  "NAME:Coordinate",
	  "X:=", 1, "Y:=", 2
	  ],
	  [
	  "NAME:Coordinate",
	  "X:=", 3, "Y:=", 4
	  ]
	  ]
	]
	)

VB Syntax	EditDataset <*OriginalName*> <*DatasetDataArray*>
VB Example	oProject.EditDataset "ds1" Array("NAME:ds2",_ Array("NAME:Coordinates",Array("NAME:Coordinate", "X:=", 1, "Y:=", 2), Array("NAME:Coordinate", "X:=", 3, "Y:=", 4))) oDesign.EditDataset "ds1" Array("NAME:ds2",_ Array("NAME:Coordinates",Array("NAME:Coordinate",

	"X:=", 1, "Y:=", 2), Array("NAME:Coordinate", "X:=", 3, "Y:=", 4)))

GetActiveDesign

Returns the design in the active project

> **Note:** `GetActiveDesign` will return normally if there are no active objects.

UI Access	N/A
Parameters	None.
Return Value	Object of the active design.
Python Syntax	GetActiveDesign()
Python Example	`oDesign = oProject.GetActiveDesign()`

VB Syntax	GetActiveDesign
VB Example	`Set oDesign = oProject.GetActiveDesign`

GetChildNames [Project]

Returns the names of the project's child objects.

UI Access	N/A

Parameters	<table><tr><td>Name</td><td>Type</td><td>Description</td></tr><tr><td><*type*></td><td>String</td><td>(Optional) Default is "design"; Any value returned by GetChildTypes() can be used.</td></tr></table>
Return Value	Array of names of children for the queried object.

Python Syntax	GetChildNames (<*type*>)
Python Example	`arrDesignNames = oProject.GetChildNames()` `arrVarbleNames = oProject.GetChildNames("Variable")`

VB Syntax	GetChildNames (<*type*>)
VB Example	`arrDesignNames = oProject.GetChildNames()` `arrVarbleNames = oProject.GetChildNames ("Variable")`

GetChildObject [Project]

Returns the project's child objects.

> **Note:**
>
> `GetChildObject` will return normally if there are no active objects.

UI Access	N/A
Parameters	<table><tr><td>Name</td><td>Type</td><td>Description</td></tr><tr><td><*path*></td><td>String</td><td>The path may include multiple generations (for example, "designOb-</td></tr></table>

	ject/moduleObj/SetupObject"). See Object Path.
Return Value	Object of a found child.

Python Syntax	GetChildObject(<*path*>)

Python Example	`oProject = oDeskop.GetActiveProject()`
	`oDesign = oProject.GetChildObject("TeeModel")`
	`oVariable = oProject.GetChildObject("VariableName")`
	`oReport = oProject.GetChildObject("TeeModel/Results/S Parameter Plot 1")`

VB Syntax	GetChildObject(<*path*>)

VB Example	`Set oProject = oDesktop.GetActiveproject()`
	`Set oDesign = oProject.GetChildObject("TeeModel")`
	`Set oVariable = oProject.GetChildObject("VariableName")`
	`Set oReport = oProject.GetChildObject("TeeModel/Results/S Parameter Plot 1")`

GetChildTypes [Project]

Returns the types of the project's child objects.

UI Access	N/A

Parameters	None.
Return Value	Array of string represents types of the child objects.

Python Syntax	GetChildTypes()
Python Example	`oProject.GetChildTypes()`

VB Syntax	GetChildTypes
VB Example	`oProject.GetChildTypes`

GetConfigurableData (Project)

> **Note:**
> This command is for internal Ansys use only.

Python Syntax	GetConfigurableData()
Python Example	`oProject.GetConfigurableData()`

GetDefinitionManager

Gets the `DefinitionManager` object.

UI Access	N/A
Parameters	None.
Return Value	`DefinitionManager` object.

Python Syntax	GetDefinitionManager()
Python Example	`oDefinitionManager = oProject.GetDefinitionManager()`

VB Syntax	GetDefinitionManager
VB Example	`Set oDefinitionManager = oProject.GetDefinitionManager`

Note: For more information on commands for the DefinitionManager, see Definition Manager Script Commands.

GetDependentFiles

Provides a list of the external files referenced in the project, including characteristic (for example, MDX) and coupled project files.

UI Access	N/A
Parameters	None.
Return Value	List of referenced files.

Python Syntax	GetDependentFiles()
Python Example	`files = oProject.GetDependentFiles()`

VB Syntax	GetDependentFiles
VB Example	`files = oProject.GetDependentFiles`

GetEDBHandle

Returns the EDB handle for the project.

> **Important:**
>
> This script is for internal Ansys use only.

UI Access	N/A
Parameters	None.
Return Value	String indicating the EDB handle for the project.

Python Syntax	GetEDBHandle()
Python Example	`oProject.GetEDBHandle()`

VB Syntax	GetEDBHandle

VB Example	`oProject.GetEDBHandle`

GetLegacyName

Obtains the legacy name of a project.

> **Note:**
> This command is for internal Ansys use only.

UI Access	N/A
Parameters	None.
Return Value	String containing the legacy project name.

Python Syntax	GetLegacyName()
Python Example	`oProject.GetLegacyName()`

VB Syntax	GetLegacyName
VB Example	`oProject.GetLegacyName`

GetName [Project]

Obtains the project name

UI Access	N/A
Parameters	None.
Return Value	String containing the project name, not including the path or extension.

Python Syntax	GetName()
Python Example	`oProject.GetName()`

VB Syntax	GetName
VB Example	`oProject.GetName`

GetPath

Returns the location of the project on disk.

UI Access	N/A
Parameters	None.
Return Value	String containing the path to the project, not including the project name.

Python Syntax	GetPath()
Python Example	`oProject.GetPath()`

VB Syntax	GetPath
VB Example	`oProject.GetPath`

GetPropNames [Project]

Obtains the property name of the object. At the project level, GetPropNames always returns empty because the project is not associated with any property.

UI Access	N/A
Parameters	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td><*includeReadOnly*></td><td>Boolean</td><td>(Optional) • True – Include read only props. • False – Do not include read only props.</td></tr></table>
Return Value	Empty array.

Python Syntax	GetPropNames ()
Python Example	`oProject.GetPropNames()`

VB Syntax	GetPropNames

VB Example	`oProject.GetPropNames`

GetPropValue [Project]

Returns the property value for the active project object, or specified property values.

UI Access	N/A
Parameters	Name / Type / Description <*propPath*> / String / A child object's property path. See: Property Function Summary.
Return Value	String of property value.

Python Syntax	GetPropValue(<*propPath*>)
Python Example	`oProject.GetPropValue("TeeModel/offset")`

VB Syntax	GetPropValue <*propPath*>
VB Example	`oProject.GetPropValue "TeeModel/offset"`

GetTopDesignList

Returns a list of top-level design names.

UI Access	N/A
Parameters	None.
Return Value	List of strings containing name of top-level designs.

Python Syntax	GetTopDesignList()
Python Example	`oProject.GetTopDesignList()`

VB Syntax	GetTopDesignList
VB Example	`oProject.GetTopDesignList`

ImportDataset

Imports a dataset from a named file. This can be executed by the oProject, or oDesign variables. The name of the dataset is file-name+index number (e.g., dsdata1) unless the filename ends with a trailing number. When there is a trailing number at the end, we will remove the number and use first unused index. Alternatively, the name of the dataset can be explicitly defined by providing a string as an optional second argument.

UI Access	**Project** > **Datasets** > **Import**.
Parameters	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td><*datasetFileFullPath*></td><td>String</td><td>The full path to the file containing the dataset values. *.tab files recommended (see note below).</td></tr><tr><td><*optionalDatasetName*></td><td>String</td><td>*Optional*. User-defined dataset name.</td></tr></table>
Return Value	None.

Python Syntax	ImportDataset (<*datasetFileFullPath*>,<*optionalDatasetName*>)

Python Example	`oProject.ImportDataset('e:\tmp\dsdata.tab')`
	`oDesign.ImportDataset('e:\tmp\dsdata.tab')`
	`oProject.ImportDataset('e:\tmp\dsdata.tab', 'MyDatasetName')`
	`oDesign.ImportDataset('e:\tmp\dsdata.tab', 'MyDatasetName')`

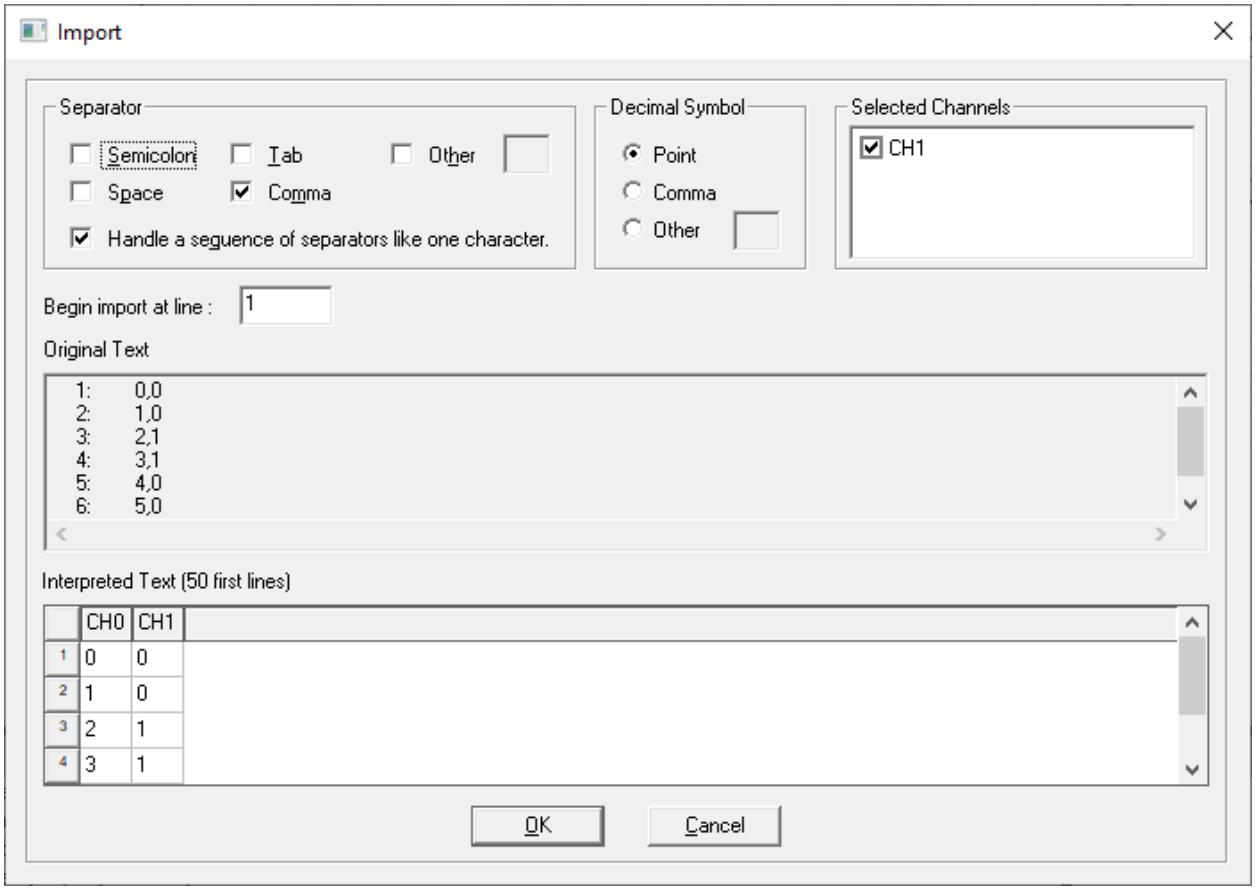
VB Syntax	ImportDataset <*datasetFileFullPath*>, <*optionalDatasetName*>

VB Example	`oProject.ImportDataset "e:\tmp\dsdata.tab"`
	`oDesign.ImportDataset "e:\tmp\dsdata.tab"`
	`oProject.ImportDataset "e:\tmp\dsdata.tab", "MyDatasetName"`
	`oDesign.ImportDataset "e:\tmp\dsdata.tab", "MyDatasetName"`

Note About File Types:

Tab-delimited or space-delimited files with the extension *.tab are the recommended file type. When using ImportDataset at the Design level, *.tab is the only file type supported.

At the Project level, other file types are supported (for example, *.csv). However, after calling the command, you must configure the file import format manually through the Electronics Desktop GUI by selecting **Project** > **Datasets** and clicking **Import**.



InsertDesignWithWorkflow

Inserts a design with a named workflow and returns an IDispatch string.

UI Access	N/A
Parameters	

Name	Type	Description
<type>	String	Type of design.
<workflowName>	String	Name of the workflow.
<specName>	String	Name of the spec.
<fileName>	String	Name of the file.
<libLoc>	String	Type of library, such as SysLib.
<stationaryPath>	String	Path.

Return Value	IDispatch string, such as 'IDispatch(IAltraSimScript)'

Python Syntax	InsertDesignWithWorkflow(*<type>*, *<workflowName>*, *<specName>*, *<fileName>*, *<libLoc>*, *<stationaryPath>*)
Python Example	`oProject.InsertDesignWithWorkflow ("Circuit Design", "Serial Design", "PCIe3 Stressed", "LongChannel", "SysLib", "C:\\Program Files\\AnsysEM\\v242\\Win64\\syslib\\MS - RT_duroid 6010 (Er=10.2) 0.010 inch, 0.5 oz copper.asty")`

VB Syntax	InsertDesignWithWorkflow *<type>*, *<workflowName>*, *<specName>*, *<fileName>*, *<libLoc>*, *<stationaryPath>*
VB Example	`oProject.InsertDesignWithWorkflow "Circuit Design", "Serial Design", _ "PCIe3 Stressed", "LongChannel", "SysLib", _ "C:\Program Files\AnsysEM\v242\Win64\syslib\MS - RT_duroid 6010" & _ " (Er=10.2) 0.010 inch, 0.5 oz copper.asty"`

InsertToolObject

> **Note:**
> This command is for internal Ansys use only.

Python Syntax	InsertToolObject()
Python Example	`oProject.InsertToolObject()`

Paste (Project Object)

UI Access	**Edit** > **Paste**.
Parameters	None.
Return Value	

Python Syntax	Paste()
Python Example	`oProject.Paste()`

VB Syntax	Paste
VB Example	`oProject.Paste`

Redo [Project Level]

Reapplies the last project-level command.

UI Access	**Edit** > **Redo**.
Parameters	None.
Return Value	None.

Python Syntax	Redo()
Python Example	`oProject.Redo()`

VB Syntax	Redo
VB Example	`oProject.Redo`

Rename

Renames the project and saves it. Similar to SaveAs().

UI Access	**Edit** > **Rename**.
Parameters	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td><*NewName*></td><td>String</td><td>Desired name of the project. The path is optional.</td></tr><tr><td><*OverWriteOk*></td><td>Boolean</td><td>• True - overwrite the file on disk if it exists. • False - prevent overwrite.</td></tr></table>
Return Value	None.

Python Syntax	Rename(<*NewName*>,<*OverWriteOK*>)
Python Example	`oProject.Rename("c:\projects\MyProject.aedt", True)`

VB Syntax	Rename <*NewName*>,<*OverWriteOK*>
VB Example	`oProject.Rename "c:\projects\MyProject.aedt", true`

RestoreProjectArchive

Restores a previously archived project to a specified path.

UI Access	**File** > **Restore Archive**.
Parameters	

Name	Type	Description
<*ArchiveFilePath*>	String	Path to archived file
<*ProjectFilePath*>	String	Path to restore location
<*OverwriteExistingFiles*>	Boolean	True to overwrite an existing file of the same name; else False.
<*OpenProjectAfterRestore*>	Boolean	True to open the project after it is restored; else False.

Return Value	None.

Python Syntax	RestoreProjectArchive (<*Archivefilepath*>, <*ProjectFilePath*>, <*OverwriteExistingFiles*>, <*OpenProjectAfterRestore*>)

Python Example	`oDesktop.RestoreProjectArchive("C:\Users\jdoe\Documents\OptimTee.aedtz", "C:\Documents\OptimTee.aedt", False, True)`

VB Syntax	RestoreProjectArchive <*Archivefilepath*>, <*ProjectFilePath*>, <*OverwriteExistingFiles*>, <*OpenProjectAfterRestore*>
VB Example	`oDesktop.RestoreProjectArchive "C:\Users\jdoe\Documents\OptimTee.aedtz", _` `"C:\Documents\OptimTee.aedt", false, true`

Save

Saves the active project.

UI Access	**File** > **Save**.
Parameters	None.
Return Value	None.

Python Syntax	Save()
Python Example	`oProject.Save()`

VB Syntax	Save
VB Example	`oProject.Save`

SaveAs

Saves the project under a new name. Requires a full path.

> **Note:**
>
> This script takes two parameters for non-schematic/layout designs and four parameters for schematic/layout designs.

UI Access	**File** > **Save As**.
Parameters	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td><*NewName*></td><td>String</td><td>The desired name of the project, with directory and extension.</td></tr><tr><td><*OverWriteOK*></td><td>Boolean</td><td>True to overwrite the file of the same name, if it exists. False to prevent overwrite.</td></tr><tr><td><*DefaultAction*></td><td>String</td><td>For Schematic/Layout projects only. Otherwise omit. See note below. Valid actions: ef_overwrite , ef_copy_no_overwrite, ef_make_path_absolute, or empty string.</td></tr><tr><td><*OverwriteActions*></td><td>Array</td><td>For Schematic/Layout projects only. Otherwise omit. See note below. Structured array: Array("Name: <Action>", <FileName>, <FileName>, ...) Valid actions: ef_overwrite , ef_copy_no_overwrite, ef_make_path_absolute, or empty string.</td></tr></table>
Return Value	None.

Python Syntax	For non-Schematic/Layout project: SaveAs (<*NewName*> <*OverWriteOK*>) For Schematic/Layout project: SaveAs (<*NewName*> <*OverWriteOK*> <*DefaultAction*> <*OverrideActions*>)

Python Example	oProject.SaveAs('D:/projects/project1.aedt', True) ---- oProject.SaveAs('D:/Projects/Project1.aedt', True, 'ef_overwrite', ['NAME:OverrideActions', ['NAME:ef_copy_no_overwrite', ['NAME:Files', '$PROJECTDIR/circuit_models.inc']], ['NAME:ef_make_path_absolute', ['NAME:Files', '$PROJECTDIR/SL_6s.sp']]])

VB Syntax	For non-Schematic/Layout project: SaveAs <*NewName*> <*OverWriteOK*> For Schematic/Layout project: SaveAs <*NewName*> <*OverWriteOK*> <*DefaultAction*> <*OverrideActions*>
VB Examples	oProject.SaveAs "D:/projects/project1.aedt", true ---- oProject.SaveAs "F:\Designer Projects\TA33097\HighSpeedChannel.aedt", true, "ef_overwrite", Array ("NAME:OverrideActions", Array("NAME:ef_copy_no_overwrite", Array("NAME:Files","$PROJECTDIR/circuit_models.inc")), Array("NAME:ef_make_path_absolute", Array("NAME:Files","$PROJECTDIR\SL_6s.sp")))

Important:

The DefaultAction and OverrideActions strings correspond to the following actions:

- **ef_overwrite** – Copy file to new project directory and overwrite.
- **ef_copy_no_overwrite** – Copy file to new project directory and don't overwrite.
- **ef_make_path_absolute** – Change reference to point to file in old project directory.
- **Empty String** – Do nothing.

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

- If OverrideActions are not specified, DefaultAction is applied to ALL files in project directory.

SaveAsStandAloneProject

Saves the project as a standalone copy.

Note:

This script is not supported when the application is being controlled by Ansys Workbench.

<table>
<tr><td>UI Access</td><td>N/A</td></tr>
<tr><td>Parameters</td><td>
<table>
<tr><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td><projectName></td><td>String</td><td>The desired name of the project, with directory and extension.</td></tr>
</table>
</td></tr>
<tr><td>Return Value</td><td>None.</td></tr>
</table>

Python Syntax	SaveAsStandAloneProject(<*projectName*>)
Python Example	`oProject.SaveAsStandAloneProject('D:/projects/project1.aedt')`

VB Syntax	SaveAsStandAloneProject <*projectName*>
VB Examples	`oProject.SaveAsStandAloneProject "D:/projects/project1.aedt"`

SaveProjectArchive

Saves the active project as an archive to the specified file path.

UI Access	**File** > **Archive**.
Parameters	<table><tr><td>Name</td><td>Type</td><td>Description</td></tr><tr><td><*archiveFilePath*></td><td>String</td><td>Path to archived file.</td></tr><tr><td><*IncludeExternalFiles*></td><td>Boolean</td><td>True to include external files; False to exclude.</td></tr><tr><td><*IncludeResultsFiles*></td><td>Boolean</td><td>True to include simulation files associated with the project; False to exclude.</td></tr><tr><td><*AdditionalFiles*></td><td>Array</td><td>Additional specified files to include.</td></tr><tr><td><*ArchiveNotes*></td><td>String</td><td>String describe the archive.</td></tr></table>
Return Value	None.

Python Syntax	SaveProjectArchive(<*archivefilepath*>, <*IncludeExternalFiles*>, <*IncludeResultsFiles*>, <*AdditionalFiles*>, <*ArchiveNotes*>)
Python	`oProject.SaveProjectArchive("C:\\Users\\Documents\\Ansoft\\Project27.aedtz", True,`

Example	`False, [], ""`)

VB Syntax	SaveProjectArchive <*archivefilepath*>, <*IncludeExternalFiles*>, <*IncludeResultsFiles*>, <*AdditionalFiles*>, <*ArchiveNotes*>
VB Example	`oProject.SaveProjectArchive "C:\Documents\OptimTee.aedtz", true,  false, Array(), "My notes"`

SetActiveDefinitionEditor

Obtains a specified definition editor.

UI Access	N/A
Parameters	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td><*EditorName*></td><td>String</td><td>Name of the definition editor to set active, one of "SymbolEditor", "FootprintEditor".</td></tr><tr><td><*DefinitionName*></td><td>String</td><td>The combination name for the symbol or footprint, <*libname*>:<*def-name*></td></tr></table>
Return Value	Object for the definition to be edited.

Python Syntax	SetActiveDefinitionEditor(<*EditorName*>, <*DefinitionName*>)
Python Example	`oProject.SetActiveDefinitionEditor("SymbolEditor",` `"Simplorer Elements\Basic Elements\Circuit\Passive Elements:R")`

VB Syntax	SetActiveDefinitionEditor <*EditorName*>, <*DefinitionName*>
VB Example	`oProject.SetActiveDefinitionEditor "SymbolEditor",_` `"Simplorer Elements\Basic Elements\Circuit\Passive Elements:R"`

SetActiveDesign

Sets a design to be the active design.

UI Access	N/A
Parameters	Name / Type / Description <*DesignName*> / String / Name of the design to set as the active design.
Return Value	None.

Python Syntax	SetActiveDesign (<*DesignName*>)
Python Example	`oDesign = oProject.SetActiveDesign("SimplorerDesign2")`

VB Syntax	SetActiveDesign <*DesignName*>
VB Example	`Set oDesign = oProject.SetActiveDesign "SimplorerDesign2"`

SetPropValue [Project]

Sets a property value for an active project's child object.

UI Access	Edit Properties on ProjectTree objects.
Parameters	<table><tr><td>Name</td><td>Type</td><td>Description</td></tr><tr><td><*propPath*></td><td>String</td><td>A child object's property path. See: Property Function.</td></tr><tr><td><*newValue*></td><td>String</td><td>New property value.</td></tr></table>
Return Value	Boolean: • **True** – property found. • **False** – property not found.

Python Syntax	SetPropValue(<*propPath*>, <*newValue*>)
Python Example	`oProject.SetPropValue("TeeModel/offset", "2mm")` `oProject.SetPropValue("TeeModel/Results/S Parameter Plot 1/Display Type", "Data Table")`

VB Syntax	SetPropValue <*propPath*>, <*newValue*>
VB Example	`oProject.SetPropValue "TeeModel/offset", "2mm"` `oProject.SetPropValue "TeeModel/Results/S Parameter Plot 1/Display Type", "Data Table"`

SimulateAll

Simulates all solution setups and Optimetrics setups for all design instances in the project. Script processing only continues when all analyses are finished.

UI Access	N/A
Parameters	None.
Return Value	None.

Python Syntax	SimulateAll()
Python Example	`oProject.SimulateAll()`

VB Syntax	SimulateAll
VB Example	`oProject.SimulateAll`

Undo [Project]

Cancels the last project-level command.

UI Access	**Edit** > **Undo**.
Parameters	None.
Return Value	None.

Python Syntax	Undo()
Python Example	`oProject.Undo()`

VB Syntax	Undo
VB Example	`oProject.Undo`

UpdateDefinitions

Updates all definitions. The **Messages** window reports when definitions are updated, or warns when definitions cannot be found.

UI Access	**Tools** > **Project Tools** > **Update Definitions**. Click **Select All**, then **Update**.
Parameters	None.
Return Value	None.

Python Syntax	UpdateDefinitions()
Python Example	`oProject.UpdateDefinitions()`

VB Syntax	UpdateDefinitions
VB Example	`oProject.UpdateDefinitions`

ValidateDesign

Returns whether a design is valid.

UI Access	> **Validation Check**.
Parameters	None.
Return Value	Integer: • **1** – Validation passed. • **0** – Validation failed.

Python Syntax	ValidateDesign()
Python Example	`oDesign.ValidateDesign()`

VB Syntax	ValidateDesign
VB Example	`oDesign.ValidateDesign`

This page intentionally
left blank.

6 - Property Script Commands

Property script commands allow you to navigate through all objects and properties in a project. You can get and set all properties for all objects in the Project tree with simple data types.

Property Object is the base class defined for all script objects that support the properties Get and Set.

`GetName()`

- Returns the name of the object.

`GetChildTypes()`

- An object may have different types of children. For example, a design may have variables, modules, and editors.
- Returns an array of text strings; may be empty if the children are not divided into different types.

`GetChildNames(<type>)`

- <type> – Child type name. By default, returns all children names for all types.
- Returns an array of immediate children names, belonging to a type if specified.

`GetChildObject(<objPath>)`

- <objPath> – A child object path; can contain multiple generations (for example, designObject/moduleObject/SetupObject).
- Returns a child property object if the object is found.

`GetPropNames(<bIncludeReadOnly>)`

- <bIncludeReadOnly) – Optional; defaults to true. True includes read-only properties; False excludes read-only properties.
- Returns an array of the object's property names.

`GetPropValue(<propertyPath>)`

- <propertyPath> – The property's path; may be a child object's path appended with a property name (for example, TeeModel/Offset/SIValue).

- Returns the property value if found. Otherwise causes script error.

`SetPropValue(<propertyPath>, <data>)`

- <propertyPath> – The property's path; may be a child object's path appended with a property name (for example, TeeModel/Offset/SIValue).
- <data> – New data; type depends on property type.
- Returns True if updated successfully; False if new data is invalid.

For a detailed summary of how Property script commands are used in a range of contexts, including Variable objects, see: Object Script Property Function Summary. Additional examples for these commands are listed under Project Objects, Design Objects, 3D Modeler, Optimetrics, Radiation Module and Reporter.

> **Note:**
>
> Older property commands should be executed by the oProject object.
>
> ```
> Set oProject = oDesktop.SetActiveProject("Project1")
>
> oProject.CommandName <args>
> ```

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

Object Script Property Function Summary

Object Path

The Object path can be used to navigate through objects and properties in an Ansys EM project.

- An Object path consisted of one or multiple Object-ID-Nodes separated by "/" .
- Object-ID-Node; may exist in the following forms:
  - A simple object name or property name.
  - Type[Name] for object; Tab[name] for property.
  - Name[attr1="v1", attr2 = "v2", ...]. When more than one child object have the same name, use attributes to specify the difference.
  - ArrayName[index]. For example, in an Optimetric setup with multiple calculations, "Calculation[0]" could be used to identify the first calculation.
  - Name beginning with '@' character denoted as a property name, when an object has a child and property with the same name.

Property Object

The Property Object is the base class defined for all script object that support property Get & Set.

- GetName()
  - Returns the name of the object.

- GetChildTypes()
    - An object may have different type of children. For example, a design may have variables, modules, and editors.
    - Returns array of text strings; may be empty if the children are *not* divided to different types.
- GetChildNames(<*type*>)
    - <*type*> – children type name; default returns all children names for all types.
    - Returns an array of immediate children names, belonging to the type if specified.
- GetChildObject(<*objPath*>)
    - <*objPath*> – A child object path. The path may include multiple generations, such as (designObject/moduleObj/SetupObject).
    - Returns a child property object if the object found.
- GetPropEvaluatedValue(<*propName*>)
    - Return the Evaluated-Value for Value-Property and Variable.
    - Return the Property-value as text string for other property types.
- GetPropSIValue(<*propName*>)
    - Return the SI-Value for Value-Property and Variable.
    - Return NAN for other property type if its value is cannot be converted to a double-floating point value.
- GetPropNames(<*bIncludeReadOnly*>);
    - <*bIncludeReadOnly*> – optional, default to true; True will include read-only properties, False will exclude read-only properties.
    - Returns an array of the object's property names.
- GetPropValue(<*propertyPath*>)
    - <*propertyPath*> – the property's full path. A property name or child object's path appended with a property name, like "TeeModel/Offset/SIValue"
    - Returns the property value if the property is foun; otherwise causes script error.

- SetPropValue(<*propertyPath*>, <*data*>)
  - <*propertyPath*> – the property's full path. A property name or child object's path appended with a property name, like "TeeModel/Offset/Value"
  - <*data*> – new data, type is dependent on property type.
  - Returns True if property data is updated successfully; False if the new data is invalid.

Project Object

Project Object inherited all functions defined in the Property Object. But it doesn't have property, GetPropValue() & SetPropValue() function can be used to set its child object's property.

- GetChildTypes() always return ["Design", "Variable"].
- GetChildNames(type)
  GetChildNames() & GetChildName("Design") will return all Design names of the project.
  GetChildNames("Variable") return all project variable names.
- GetChildObject(objPath)

  oDesign = oProject("TeeModel")

  oVariable = oProject.GetChildObject("VariableName")

  oReport = oProject.GetChildObject("TeeModel/Results/S Parameter Plot 1")

- GetPropNames(bIncludeReadOnly) always return empty array since the project has no property.
- GetPropValue(propertyPath)

  oProject.GetPropValue("TeeModel/offset") //get the offset variable value in the TeeModel Design
  oProject.GetPropValue("TeeModel/Results/S Parameter Plot 1/Display Type") // Get the report display type.

- SetPropValue(propertyPath, newValue)

  oProject.SetPropValue("TeeModel/offset", "2mm") //Set the offset variable value to "2mm" in the TeeModel Design
  oProject.SetPropValue("TeeModel/Results/S Parameter Plot 1/Display Type", "Data Table) // Set the report display type to data table.

Design Object

Design Object inherited all functions defined in the Property Object. But it doesn't have property, GetPropValue() & SetPropValue() function can be used to set its child object's property..

- GetChildTypes() always return ['Module', 'Editor', 'Variable'].
- GetChildNames(type)
  GetChildNames() will return modules & editor child names.
  GetChildNames(“Variable”) will return all variable names
  GetChildNames(“Module”) will return all module names that support property-object-script like ['Optimetrics', 'RadField', 'Results']
  GetChildNames(“Editor”) will return a 3D editor name for all 3D Designs
- GetChildObject()
  oVariable = oDesign.GetChildObject("VariableName")
  oReport = oDesign.GetChildObject("Results/S Parameter Plot 1")
  oRptModule = oDesign.GetChildObject("ReportSetup")
- GetPropNames(bIncludeReadOnly) always return empty array since the design has no property.
- GetPropValue()
  oDesign GetPropValue("offset/SIValue") //get the offset variable SI value in the DesignoDesign GetPropValue("offset/SIValue") //get the offset variable SI value in the Design
  oDesign.GetPropValue("Results/S Parameter Plot 1/Display Type") // Get the report display type
- SetPropValue()
  oDesign.SetPropValue("offset", "2mm") //Set the offset variable value to "2mm" in the Design
  oDesign.SetPropValue("Results/S Parameter Plot 1/Display Type", "Data Table) // Set the report display type to data table.

3D Modeler Object

GetChild commands returns the appropriate properties for modeler objects. For 3D Components and UDMs, these commands do not return parts, coordinate systems, plans, as top-level modeler children.

```
oModeler = oDesktop.GetActiveProject().GetActiveDesign().GetChildObject("3D Modeler")
oModeler.GetChildNames()
oModeler.GetChildNames("ModelParts")
oModeler.GetChildNames("AllParts")
oModeler.GetChildNames("NonModelParts")
oModeler.GetChildNames("Planes")
oModeler.GetChildNames("CoordinateSystems")
```

Variable Object

Is a Property Object that has no child. It also provides quick function call to get/set it properties by adding functions with property name appended to Get_ & Set_ prefix. To find what functions it provided enter dir(oVar) the command window. It can accessed by the project or design object's GetChildObject(VariableName) function.

```
oProjVar = oProject.GetChildObject("$VarName")
oVar = oProject.GetChildObject("DesignName/VarName")
oVar = oDesign.GetChildObject("variableName")
oProject..GetChildNames("Variable") will return all project variable names.
oDesign.GetChildNames(Variable") will return all Design Variable names.
```

- GetChildTypes() always return empty array.
- GetChildNames() always return empty array , since variable has no child.
- GetChildObject(objPath) it has no child.
- GetPropNames(bIncludeReadOnly) ['EvaluatedValue', 'SIValue'] are read-only properties
    - Independent variable :['Value', 'EvaluatedValue', 'SIValue', 'Description', 'ReadOnly', 'Hidden', 'Sweep', 'Optimization/Included', 'Optimization/Min', 'Optimization/Max', 'Sensitivity/Included', 'Sensitivity/Min', 'Sensitivity/Max', 'Sensitivity/IDisp', 'Statistical', 'Statistical/Included', 'Tuning/Included', 'Tuning/Step', 'Tuning/Min', 'Tuning/Max'].
    - Dependent variable ['Value', 'EvaluatedValue', 'SIValue', 'Description', 'ReadOnly', 'Hidden', 'Sweep']
- GetPropValue(propName)
  oVar.GetPropValue() return the variable value as text string.
  oVar.GetPropValue("Value") return the variable value as text string.
  oVar.GetPropValue("SIValue") return the SI-value of variable as number.
  oVar.Get_SIValue()also return the SI value.

- SetPropValue(propName, newValue)
  oVar.SetPropValue(“Value", 888)
  oVar.SetPropValue("Sensitivity'/Included", True)
  oVar.SetPropValue("Sensitivity/Max”, '1.8pF'])
  oVar.Set_Sensitivity_Max('1.8pF') also works as last call.
  oVar.SetPropValue("Sensitivity", ['Min:=', '0.8pF', 'Max:=', '1.8pF'])
  //set multiple attributes at one call:
  oVar.SetPropValue("@", ['Value:=',288, 'Sensitivity', ['Included', True,'Min', '0.0']])
  oVar.SetPropValue("", ['Value:=',288, 'Sensitivity', ['Included', True,'Min', '0.0']])

Optimetrics Module Object:

Optimetrics Module Object inherited all functions defined in the Property Object. But it doesn't have property, GetPropValue() & SetPropValue() function can be used to set its child object's property..

- GetChildTypes() there are six type of children, they are ['OptiParametric', 'OptiOptimization', 'OptiSensitivity', 'OptiStatistical', 'OptiDesignExplorer', 'OptiDXDOE']. But the return array only included those that have setup defined, so it may be an empty array if no optimetrics setup is defined. The GetChildNames(type) function also recognized the type name without the prefix “Opti”.
- GetChildNames(type)
  GetChildNames() will return all setup for all types.
  GetChildNames(“OptiOptimization”) & GetChildNames(“Optimization”) will return all Optimization setup.
- GetChildObject()
  oParamSetup = oOptModule.GetChildObject('ParametricSetup1') get the
  oOptSetup = oOptModule.GetChildObject('OptimizationSetup1')
- GetPropNames(bIncludeReadOnly) always return empty array since the it has no property.
- GetPropValue(propPath) may be used to get its child’s property value
  oOptModule.GetPropValue("OptimizationSetup1\Optimizer") get the optimizer name for OptimizationSetup1
- SetPropValue(propPath, newValue) may be used to set its child’s property value
  oOptModule.SetPropValue(ParametricSetup1\Enabled", False) //disable ParametricSetup1

Optimetrics Setup Object

This is a new Object inherited all functions defined in the Property Object. But it doesn't have child. It is accessible through its parents.

oOptSetup = oOptModule.GetChildObject('OptimizationSetup1')

oOptSetup = oDesign.GetChildObject('Optimetrics\OptimizationSetup1')

oOptSetup = oProject.GetChildObject('TeeModel\Optimetrics\OptimizationSetup1')

- GetChildTypes() always return empty array.
- GetChildNames(type) always return empty array
- GetChildObject()
- GetPropNames(bIncludeReadOnly) will return the property names listed in the property window when the setup is selected.
- GetPropValue(propName)
  oOptSetup.GetPropValue("Optimizer") return the selected optimizer name.
  oOptSetup.GetPropValue("Optimizer/Choices") return all optimizer names.
- SetPropValue(propName, newValue)
  oOptSetup.SetPropValue("Optimizer", “NotAnOptimzerName”; will return false.
  oOptSetup.SetPropValue("Optimizer", “Quasi Newton”; return true, since “Quasi Newton” is one of the optimizer name returned as the Optimizer Choices.
- HasResult() return true if the setup is solved. Otherwise return false.
- Validate() return true if the setup is valid for analyze. Otherwise return false. Calling the SetPropValue() function to change the property may invalid the setup.

ReportSetup(Results) Module Object:

ReportSetup module Object inherited all functions defined in the Property Object. But it doesn't have property, GetPropValue() & SetPropValue() function can be used to get/set its child object's property..

- GetChildTypes() always empty array.
- GetChildNames(type)
  GetChildNames() return all report names

- •GetChildObject(objPath)
  oRpt = oRptModule.GetChildObject("S Parameter Plot 1") return the report property object
  oTrace = oRptModule.GetChildObject("S Parameter Plot 1/dB(S(Port1,Port1))") return the trace property object
  oAxisX = oRptModule.GetChildObject("S Parameter Plot 1/AxisX") return the axis X property object
- GetPropNames(bIncludeReadOnly) always return empty array since the it \has no property.
- GetPropValue()
  oRptModule.GetPropValue(“S Parameter Plot 1/Display Type”)
- SetPropValue()
  oRptModule.SetPropValue(“S Parameter Plot 1/Display Type”, “DataTable”)

ReportSetup(Results) Module Child Objects:

These are Property Objects. Its first level of child object is report. Report has trace, axis, header, Legend, and more children. Trace has curve as child etc.

Those child objects can be accessed by calling all levels of parent object’s GetChildObject(path) function.

oRpt = oRptModule.GetChildObject(reportName)

oRpt = oDesign.GetChildObject(“Results/reportName”)

oTrace = oRpt.GetChildObject(traceName)

oTrace = oRptModule.GetChildObject(ReportName/TraceName)

- GetChildTypes() always return empty array.
- GetChildNames() get the object’s child names. What will be returned will depended on the object instance.
- GetChildObject(objPath)
- GetPropNames(bIncludeReadOnly) will return the property names listed in the property window when the object is selected.
- GetPropValue(propName)
  oRpt.GetPropValue("Display Type") return the report’s display type.
  oOptSetup.GetPropValue("Display Type/Choices") return all optimizer names.
  oTrace.GetPropValue(“X Component”)

- SetPropValue(propName, newValue)
  oTrace.SetPropValue(“Primary sweep”, “Freq”)

Radiation Module Object:

This inherited all functions defined in the Property Object. But it doesn't have property, GetPropValue() & SetPropValue() function can be used to set its child object's property.

- GetChildTypes() always return empty array, now its children
- GetChildNames(type)
  GetChildNames() return all setup names.
- GetChildObject(setupName) return the setup object as Property object.
  oOverlay= oRadModule.GetChildObject('Antenna Parameter Overlay1')
  oSphere = oRadModule.GetChildObject('Infinite Sphere1')
- GetPropNames() return empty array; it has no property.
- GetPropValue()
  oRadModule.GetPropValue('Line1/Num Points') //Get the the Line1 setups’ “Num Points” property value.
- SetPropValue()
  oRadModule.SetPropValue('Line1/Num Points', 100) ; Set the Line1 setups’ “Num Points” property to 100.

Radiation Module Child Objects:

These are Property Objects. It also provides quick function call to get/set its properties by adding functions with property name appended to Get_ & Set_ prefix. To find what functions it provides enter dir(oVar) the command window.

Those child objects can be access by call all levels of parent object’s GetChildObject(path) function.

oRadSetup = oRadModule.GetChildObject(setupName)

oRadSetup = oDesign.GetChildObject(RadField/setupName)

- GetChildTypes() always return empty array.
- GetChildNames() always return empty array , since Radiation setup has no child.
- GetChildObject(objPath) it has no child.

- GetPropNames(bIncludeReadOnly) will return the property names listed in the property window when the setup is selected.
- GetPropValue(propName)
  oRadSetup.GetPropValue(”Num Points") return the line setup’s “Num Points” property value.
  oRadSetup.Get_NumPoints() will also get the same value.
- SetPropValue(propName, newValue)
  oRadSetup.SetPropValue('Num Points', 888)
  oRadSetup.Set_NumPoints(888)

Conventions Used in this Chapter

General Definitions:

Property	A single item that can be modified in the **Properties** window or in the modal **Properties** pop-up window.
<PropServer>	The item whose properties are being modified. This is usually a compound name, giving all information needed by the editor, design, or project in order to locate the item.
<PropTab>	Corresponds to one tab in the **Properties** window, the one under which properties are being edited.
<PropName>	The name of a single property.

The following tables list specific <PropServer> and <PropTab> values for different property types.

For Project Variables:

<PropServer>	"ProjectVariables"
<PropTab>	"ProjectVariableTab"

For Local Variables:

<PropServer>	"LocalVariables"
<PropTab>	"LocalVariableTab"

For Passed Parameters:

<PropServer>	"Instance:<Name of Circuit Instance>"
<PropTab>	"PassedParameter Tab"

For Definition Parameters:

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

For Modules and Editors:

<PropServer>	<ModuleName>:<ItemName> where <ItemName is the boundary name, solution setup name, etc. For example, "BoundarySetup:PerfE1"
<PropTab>	Boundary Module: "HfssTab" Mesh Operations Module: "MeshSetupTab" Analysis Module: "HfssTab" Optimetrics Module: "OptimetricsTab" Solutions Module: *Does not support properties.* Field Overlays Module: "FieldsPostProcessorTab" Radiation Module: "RadFieldSetupTab" Circuit Module: "CCircuitTab" System Module: "SystemTab" HFSS 3D Layout Module: "HFSS 3D LayoutTab" Nexxim Module: "NexximTab" Layout elements: "BaseElementTab" Schematic elements: "ComponentTab" Optimetrics Module: "OptimetricsTab"

For 3D Model Editor objects:

<PropServer>	Name of the object. For example, "Box1".
<PropTab>	"Geometry3DAttributeTab"

For 3D Model Editor operations:

<PropServer>	<ObjName>:<OperationName>:<int> where <int> is the operation's history index. For example, "Box2:CreateBox:2" refers to the second "CreateBox" operation in Box2's history.
<PropTab>	"Geometry3DCmdTab"

For Reporter operations on Report properties:

<PropServer>	<ReportSetup>

<ChangeProperty>	Array. For example, to set the company name in a plot header to "My Company":
	`Set oModule = oDesign.GetModule("ReportSetup")`
	`oModule.ChangeProperty Array("NAME:AllTabs",_ Array("NAME:Header",_ Array ("NAME:PropServers",_`
	`"XY Plot1:Header"), Array("NAME:ChangedProps",_ Array("NAME:Company Name", "Value:=", "My Company"))))`

Note:

For scripted property changes in the various modules and editors, refer to the chapters on the System, HFSS 3D Layout, and Nexxim tools, as well as the Layout and Schematic editors.

GetArrayVariables

Returns a list of array variables. To get a list of indexed project variables, execute with oProject. To get a list of indexed local variables, use oDesign.

UI Access	N/A
Parameters	None.
Return Value	Array of strings containing names of variables.

Python Syntax	GetArrayVariables()
Python Example	`oProject.GetArrayVariables()` `oDesign.GetArrayVariables()`

VB Syntax	GetArrayVariables
VB Example	`oProject.GetArrayVariables` `oDesign.GetArrayVariables`

GetProperties

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.

UI Access	N/A
Parameters	Name | Type | Description

	<PropTab>	String	One of the following, where tab titles are shown in parentheses: • PassedParameterTab ("Parameter Values") • DefinitionParameterTab (Parameter Defaults") • LocalVariableTab ("Variables" or "Local Variables") • ProjectVariableTab ("Project variables") • ConstantsTab ("Constants") • BaseElementTab ("Symbol" or "Footprint") • ComponentTab ("General") • Component("Component") • CustomTab ("Intrinsic Variables") • Quantities ("Quantities") • Signals ("Signals")
	<PropServer>	String	An object identifier, generally returned from another script method, such as Comp⊦nst@R;2;3
Return Value	Array of strings containing the names of the appropriate properties.		

Python Syntax	GetProperties(*<PropTab>*, *<PropServer>*)
Python Example	`oEditor.GetProperties('PassedParameterTab', 'k')`

VB Syntax	GetProperties *<PropTab>*, *<PropServer>*
VB Example	`oEditor.GetProperties "PassedParameterTab", "k"`

GetPropertyValue

Returns the value of a single property belonging to a specific <*PropServer*> and <*PropTab*>. This function is available with the Project, Design or Editor objects, including definition editors.

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

<table>
<tr><td>UI Access</td><td>N/A</td></tr>
<tr><td>Parameters</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td><*PropTab*></td><td>String</td><td>One of the following, where tab titles are shown in parentheses:
PassedParameterTab ("Parameter Values")
DefinitionParameterTab (Parameter Defaults")
LocalVariableTab ("Variables" or "Local Variables")
ProjectVariableTab ("Project variables")
ConstantsTab ("Constants")
BaseElementTab ("Symbol" or "Footprint")
ComponentTab ("General")
Component("Component")
CustomTab ("Intrinsic Variables")
Quantities ("Quantities")
Signals ("Signals")</td></tr>
<tr><td><*PropServer*></td><td>String</td><td>An object identifier, generally returned from another script method, such as <code>Comp$nst@R;2;3</code></td></tr>
<tr><td><*PropName*></td><td>String</td><td>Name of the property.</td></tr>
</table>
</td></tr>
<tr><td>Return Value</td><td>String value of the property.</td></tr>
</table>

Python Syntax	GetPropertyValue (<*PropTab*>, <*PropServer*>, <*PropName*>)

Python Example	`selectionArray = oEditor.GetSelections()`
	`for k in selectionArray:`
	`val = oEditor.GetPropertyValue("PassedParameterTab", k, "R")`
	`...`

VB Syntax	GetPropertyValue (<*PropTab*>, <*PropServer*>, <*PropName*>)

VB Example	`selectionArray = oEditor.GetSelections`
	`for k in selectionArray:`
	`val = oEditor.GetPropertyValue("PassedParameterTab", k, "R")`
	`...`

GetVariables

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`.

UI Access	N/A
Parameters	None.

Return Value	Array of strings containing the variables.

Python Syntax	GetVariables ()
Python Example	`oProject.GetVariables()` `oDesign.GetVariables()`

VB Syntax	GetVariables
VB Example	`oProject.GetVariables` `oDesign.GetVariables`

GetVariableValue

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

UI Access	N/A
Parameters	Name | Type | Description <*VarName*> | String | Name of the variable to access.
Return Value	String represents the value of the variable.

Python Syntax	GetVariableValue(<*VarName*>)
Python Example	`oProject.GetVariableValue("var_name")`

VB Syntax	GetVariableValue <*VarName*>
VB Example	`oProject.GetVariableValue "var_name"`

SetPropertyValue

Sets the value of a single propertybelonging to a specific PropServer and PropTab. This function is available with the Project, Design or Editor objects, including definition editors.This is not supported for properties of the following types: ButtonProp, PointProp, V3DPointProp, and VPointProp. Only the ChangeProperty command can be used to modify these properties.

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

UI Access	N/A
Parameters	See table below

Name	Type	Description
<*propTab*>	String	One of the following, where tab titles are shown in parentheses: • PassedParameterTab ("Parameter Values") • DefinitionParameterTab (Parameter Defaults") • LocalVariableTab ("Variables" or "Local Variables") • ProjectVariableTab ("Project variables") • ConstantsTab ("Constants") • BaseElementTab ("Symbol" or "Footprint") • ComponentTab ("General") • Component("Component")

			• CustomTab ("Intrinsic Variables") • Quantities ("Quantities") • Signals ("Signals")
	<*propServer*>	String	An object identifier, generally returned from another script method, such as Compł nst@R;2;3
	<*propName*>	String	Name of the property.
	<*propValue*>	String	The value for the property
Return Value	None.		

Python Syntax	SetPropertyValue(<*propTab*>, <*propServer*>, <*propName*>, <*propValue*>)
Python Example	`oEditor.SetPropertyValue("PassedParameterTab", "k", "R", "2200")`

VB Syntax	SetPropertyValue <*propTab*>, <*propServer*>, <*propName*>, <*propValue*>
VB Example	`oEditor.SetPropertyValue "PassedParameterTab", "k", "R", "2200"`

SetVariableValue

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`.

UI Access	N/A		
Parameters	Name	Type	Description
	<*VarName*>	String	Variable name.

	<*VarValue*>	Value	New value for the variable.
Return Value	None.		

Python Syntax	SetVariableValue (<*VarName*>, <*VarValue*>)
Python Example	`oProject.SetVariableValue('$Var1', '3mm')`

VB Syntax	SetVariableValue <*VarName*>, <*VarValue*>
VB Example	`oProject.SetVariableValue "$Var1", "3mm"`

7 - Design Object Script Commands

Design object commands should be executed by the oDesign object.

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

For example:

```
Set oDesign = oProject.SetActiveDesign("EMITDesign1")
```

UpdateLink

AddLink [EMIT]

Create a new link to a HFSS or 3D Layout design.

UI Access	**EMIT>Add Link**
Parameters	*<name>* – Name of the HFSS or HFSS 3D Layout design to add as a linked design. The design must be in the current project.
Return Value	The name of the new link.

Python Syntax	AddLink(*<name>*)
Python Example	`oDesign.AddLink("HFSSDesign1")`

AddResult [EMIT]

Create a new EMIT result. The new result node is shown under Results in the project manager.

If a current result (result with revision matching the current design revision) already exists, no result node is created and an error is shown in the message manager.

UI Access	**EMIT>Analyze** when a component is selected in the schematic
Parameters	*<desiredName>* – Name of the new result node. This is an optional parameter. If omitted, a default name is assigned. If a desiredName is specified and a result with that name already exists, no new result node is created and an error is shown in the message manager.
Return Value	The name of the new result.

Python Syntax	AddResult(<*desiredName*>)
Python Example	`oDesign.AddResult("Mitigated - Updated Antenna Placement")`

CloseResultWindow

Close the result window for the specified result. This command has no effect if no window is open for the specified result.

UI Access	Click the 'X' in the upper-right corner of the result window.
Parameters	<*resultName*> – Name of the new result node. If no result with the given name exists, an error is shown in the message manager.
Return Value	None

Python Syntax	CloseResultWindow(<*resultName*>)
Python Example	`oDesign.CloseResultWindow("Mitigated - Updated Antenna Placement")`

DeleteAllResults

Deletes all results in the Results folder

UI Access	Right-click on the Results folder, and click **Delete All Results**.
Parameters	None
Return Value	None

Python Syntax	DeleteAllResults()
Python Example	`oDesign.DeleteAllResults()`

VB Syntax	DeleteAllResults
VB Example	`oDesign.DeleteAllResults`

DeleteDesign

Deletes a specified design in the project.

UI Access	**Edit** > **Delete** or **Delete** in the ribbon.
Parameters	None
Return Value	None

Python Syntax	DeleteDesign ()
Python Example	`oDesign.DeleteDesign()`

VB Syntax	DeleteDesign
VB Example	`oDesign.DeleteDesign`

DeleteLink [EMIT]

Delete an existing coupling link.

UI Access	Right-click on the link and click **Delete** or **Edit>Delete** with the link selected.
Parameters	<*name*> – Name of the HFSS or HFSS 3D Layout design link to delete (unlink) from the EMIT design. .
Return Value	The name of the link to be deleted.

Python Syntax	DeleteLink(<*name*>)
Python Example	`oDesign.DeleteLink("HFSSDesign1")`

DeleteResult [EMIT]

Delete an EMIT result.

UI Access	Right-click on result node, and click **Delete**.
Parameters	<*resultName*> –Name of the result node to delete. If no result with the given name exists, an error is shown in the message manager.
Return Value	None

Python Syntax	DeleteResult(<*resultName*>)
Python Example	`oDesign.DeleteResult("Mitigated - Updated Antenna Placement")`

EditComponentNodes [EMIT]

Edit the nodes for an EMIT component. The component properties are stored in a hierarchy (tree) of nodes. These nodes are visible on the left-hand side of the component Edit dialog box. The usage of this method can be observed by editing some properties of a component while recording a script.

UI Access	With a component selected in the schematic, select **Edit>Configure.**
Parameters	<*componentName*> – The component to edit. <*nodeChanges*> – The list of component property nodes and the changes to apply to each. <*nodesToDelete*> – A list of property nodes to remove from the component property tree.
Return Value	None

Python Syntax	EditComponentNodes(<*componentName*>, <*nodeChanges*>, <*nodesToDelete*>)

Python Example	`oDesign.EditComponentNodes("Clock_CTL",`
	`[`
	`[`
	`"NAME:node", "fullname:=", "NODE-*-RF Systems-*-RF System-*-Radios-form",`
	`[`
	`"NAME:properties",`
	`"ChannelSpacing:=" , "3400000000.0",`
	`"StartFrequency:=" , "2400000000.0",`

```
                                                    "StopFrequency:="       , "5800000000.0"
                                        ]
                                ]
                ],
                []
```

EditNotes

Update the notes for the design.

UI Access	**EMIT** > **Edit Notes**.
Parameters	Name | Type | Description <*DesignNotes*> | String | New design notes that are applied, replacing any current notes. .
Return Value	None.

Python Syntax	EditNotes(<*DesignNotes*>)
Python Example	`oDesign.EditNotes("Initial Cosite analysis, prior to expanded WiFi installation.")`

VB Syntax	EditNotes(<*DesignNotes*>)
VB Example	`oDesign.EditNotes("Initial Cosite analysis, prior to expanded WiFi installation.")`

GetAvailableLinkNames [EMIT]

Get the names of linkable designs (excluding those that have been linked to already).

UI Access	N/A
Parameters	None.
Return Value	The list of linkable design names.

Python Syntax	GetAvailableLinkNames ()
Python Example	`names = oDesign.GetAvailableLinkNames()`

GetComponentNodeNames [EMIT]

Get the properties nodes for an EMIT component. The component properties are stored in a hierarchy (tree) of nodes. These nodes are visible on the left-hand side of the component Edit Dialog.

UI Access	N/A
Parameters	*<componentName>* – The component that will be queried for property nodes.
Return Value	The list of property nodes for the specified component.

Python Syntax	GetComponentNodeNames(*<componentName>*)
Python Example	`oDesign.GetComponentNodeNames("Clock_CTL")`

GetComponentNodeProperties [EMIT]

Get the properties nodes for an EMIT component. The component properties are stored in a hierarchy (tree) of nodes. These nodes are visible on the left-hand side of the component Edit Dialog.

UI Access	N/A
Parameters	*<componentName>* – The component that will be queried for property nodes. *<nodeFullName>* – The node of the property tree.
Return Value	The list of properties and values for the specified component property node.

Python Syntax	GetComponentNodeProperties(*<componentName>*, *<nodeFullName>*)
Python Example	`oDesign.GetComponentNodeProperties("Clock_CTL", 'NODE-*-RF Systems-*-RF System-*-Radios-*-Radio-*-Band')`

GetComponentWarnings [EMIT]

Get the warning messages associated with a component configuration.

UI Access	Hover mouse over a schematic component showing a warning indicator.
Parameters	*<componentName>* – The component that will be queried for warnings
Return Value	The warnings associated with the component. An empty string (“”) is returned if there are no warnings.

Python Syntax	GetComponentWarnings(<*componentName*>)
Python Example	`oDesign.GetComponentWarnings("Radio")`

GetCouplingWarnings [EMIT]

Get the warning messages associated with the EMIT coupling configuration.

UI Access	Click the Coupling node in the project manager or click an individual coupling link to observe the warnings in the status bar.
Parameters	<*optionalLinkName*> – Warnings for the link are returned if a link is specified. If not specified or an empty string (“”), the general coupling warnings are returned.
Return Value	The warnings associated with a specific coupling link or general coupling warnings.

Python Syntax	GetCouplingWarnings(<*optionalLinkName*>)
Python Example	`link_warnings = oDesign.GetCouplingWarnings("HFSSDesign1")` `general_warnings = oDesign.GetCouplingWarnings("")`

GetCurrentResult

Get the name of the current EMIT result (result with revision matching the current design revision).

UI Access	N/A
Parameters	None.
Return Value	The name of the current result, if present. An empty string is returned if there is no current result.

Python Syntax	GetCurrentResult()
Python Example	`oDesign.GetCurrentResult()` `print("Name of current result: '{}'".format(currentResult))`

GetDesignType

Returns the design type of the active design.

UI Access	N/A
Parameters	None.
Return Value	String indicating the design type of the active design ("Circuit Design", "Circuit Netlist","EMIT", "HFSS 3D Layout Design", "HFSS", "HFSS-IE", "Icepak", "Maxwell 2D", "Maxwell 3D", "Q2D Extractor", "Q3D Extractor", "RMxprt", or "Twin Builder").

Python Syntax	GetDesignType()
Python Example	`oDesign = oProject.GetActiveDesign()` `oDesign.GetDesignType()`

VB Syntax	GetDesignType
VB Example	`Set oDesign = oProject.GetActiveDesign` `oDesign.GetDesignType`

GetLinkNames [EMIT]

Get the names of all coupling links in the EMIT design.

UI Access	The nodes visible under “Coupling” in the Project Manager.
Parameters	None.
Return Value	The list of link names.

Python Syntax	GetLinkNames()
Python Example	`links = oDesign.GetLinkNames()`

GetManagedFilesPath

Get the path to the project's results folder.

UI Access	N/A
Parameters	None.
Return Value	String containing path where project results are located.

Python Syntax	oDesign.GetManagedFilesPath()
Python Example	`oDesign.GetManagedFilesPath()`

VB Syntax	oDesign.GetManagedFilesPath

VB Example	`oDesign.GetManagedFilesPath`

GetName

Returns the design name of the active design, in that order separated by a semicolon.

UI Access	N/A
Parameters	None.
Return Value	String indicating the name of the active design.

Python Syntax	GetName()
Python Example	design_name = oDesign.GetName()

VB Syntax	GetName
VB Example	`design_name = oDesign.GetName`

GetRadioNames [EMIT]

Get a list of all excitations and receivers in the EMIT library.

UI Access	N/A.

Parameters	None.
Return Value	The list of excitations and receivers in the EMIT library.

Python Syntax	GetRadioNames()
Python Example	`radios = oDesign.GetRadioNames()`

GetResultList

Get the list of result names for the EMIT design.

UI Access	N/A
Parameters	None.
Return Value	A list of the names of all results.

Python Syntax	GetResultList()

Python Example	`resultNames = oDesign.GetResultList()`
	`for i, name in enumerate(resultNames):`
	`  print("Result {} is named '{}'".format(i+1, name))`

GetResultNotes [EMIT]

Returns the notes for the specified result.

UI Access	N/A
Parameters	*<resultName>* - Name of the result node. If no result with the given name exists, an error is shown in the message manager.
Return Value	String containing the result's notes.

Python Syntax	GetResultNotes(*<resultName>*)
Python Example	`notes = oDesign.GetResultNotes("Revision 1")`

GetResultProperties

Get the properties for the specified result.

UI Access	Select the result node in the Project Manager. The properties are shown in the Properties window.
Parameters	*<resultName>* – Name of the result node. If no result with the given name exists, an error is shown in the message manager.
Return Value	A list containing 'key=value' entries for each property of the specified result.

Python Syntax	GetResultProperties("*<resultName>*")

Python Example	`resultProperties = oDesign.GetResultProperties("Revision 2")`
	`for property in resultProperties:`
	`  print(property)`

GetRevision

Get the revision of the EMIT design.

UI Access	Select the EMIT design node in the Project Manager. The revision is shown in the Properties window.
Parameters	None.
Return Value	The revision of the EMIT design.

Python Syntax	GetRevision()
Python Example	`designRevision = oDesign.GetRevision()`

Redo [EMIT]

Redo the last command undo performed on the design.

UI Access	**EMIT** > **Redo**.
Parameters	None.
Return Value	None.

Python Syntax	Redo()
Python Example	`oDesign.Redo() # Redo and restore state prior to undo`

RenameDesign [EMIT]

Rename the EMIT design represented by the design object.

UI Access	**Edit** > **Rename**
Parameters	*<newName>* – The new name to assign to the design.
Return Value	None.

Python Syntax	Rename(*<newName>*)
Python Example	`oDesign.Rename("New Design Name")`

RenameResult

Rename an EMIT result.

UI Access	Right-click on result node, and click **Rename**.
Parameters	*<resultName>* – Name of the result node to rename. If no result with the given name exists, an error is shown in the message manager. *<newResutName>* – The new name for the result node. If a result with the given name already exists, an error is shown in the message manager.
Return Value	None.

Python Syntax	RenameResult(<*resultName*>, <*newResultName*>
Python Example	`oDesign.RenameResult("Revision 2", "Mitigated – Filter Added")`

RunToolkit [EMIT]

Run a toolkit script installed in the Toolkits directory and available from the toolkit menu.

UI Access	**EMIT**> **Toolkit**
Parameters	<*libraryType*> – Library containing the toolkit (for example, 'syslib', 'userlib', 'personallib') <*toolkitName*> – The name of the toolkit to be run. <*toolkitArgs*> – Arguments to pass to the toolkit.
Return Value	None.

Python Syntax	RunToolkit(<*libraryType*>, *toolkitName*>, <*toolkitArgs*>)
Python Example	`oDesign.RunToolkit("userlib", "CustomImportCSV", "C:\\temp\\datafile.csv")`

SetActiveEditor [EMIT]

Activate the EMIT Schematic Editor and get the SchematicEditor object for script-based manipulation.

UI Access	**NA**.
Parameters	<*editorName*> – The name of the editor. The only editor currently supported by EMIT is "SchematicEditor".
Return Value	The SchematicEditor (oEditor) object associated with the EMIT design.

Python Syntax	SetActiveEditor(<*editorName*>)
Python Example	`oEditor = oDesign.SetActiveEditor("SchematicEditor")` `component_list = oEditor.GetAllComponents()`

SetResultNotes [EMIT]

Sets the notes for the specified result.

UI Access	N/A
Parameters	<*resultName*> - Name of the result node. If no result with the given name exists, an error is shown in the message manager. <*newNotes*> - A string containing the new desired notes for the specified result.
Return Value	N/A

Python Syntax	SetResultNotes(<*resultName*>, <*newNotes*>)
Python Example	`oDesign.SetResultNotes("Revision 1", "New notes here.")`

ShowAnalysisAndResults [EMIT]

Display the EMIT Analysis & Results dialog box.

UI Access	**EMIT** > **Analyze** or **Analysis & Results** on the Simulation ribbon..
Parameters	<*optionalScriptToRun*> – A Python script to run within the Analysis & Results dialog box. If this argument is omitted or an empty string (“”) is passed, no script is run.

Return Value	None.

Python Syntax	ShowAnalysisAndResults()
Python Example	`oDesign.ShowAnalysisAndResults()`

ShowCouplingDialog [EMIT]

Display the EMIT Coupling Editor dialog.

UI Access	**EMIT>Coupling Editor**
Parameters	<*optionalNodeToSelect*> – Optionally, a property node can be specified and this node will be initially selected when the dialog is shown. <*optionalScriptToRun*> – A Python script to run within the Coupling Editor dialog box. If this argument is omitted or an empty string (“”) is passed, no script is run.
Return Value	None

Python Syntax	ShowCouplingDialog(<optionalNodeToSelect>, <*optionalScriptToRun*>)
Python Example	`oDesign.ShowCouplingDialog('NODE-*-Scene-*-Antenna’) # Show the coupling dialog with ‘Antenna’ selected`

ShowEditDialog [EMIT]

Display the EMIT Component Editor dialog for a specified component.

UI Access	**Edit>Configure** when a component is selected in the schematic
Parameters	<*componentName*> – The component to select. May be any component type in the schematic. <*optionalScriptToRun*> – A Python script to run within the Edit dialog box. If this argument is omitted or an empty string (“”) is passed, no script is run.
Return Value	None

Python Syntax	ShowEditDialog(<*componentName*>, <*optionalScriptToRun*>)
Python Example	`oDesign.ShowEditDialog("Radio")`

ShowResultWindow

Show the result window for the specified result.

UI Access	Right-click the result node in the Project Manager, and select **Show Result**.
Parameters	<*resultName*> – Name of the result to show. If no result with the given name exists, an error is shown in the message manager. <*ScriptToRun*> - A Python script to run within the result window. If this argument is omitted or an empty string (“”) is passed, no script is run. Optional.
Return Value	None.

Python Syntax	ShowResultWindow(<*resultName*>, <*ScriptToRun*>
Python Example	`oDesign.ShowResultWindow("Mitigated – Filter Added")`

TransferToEmit

This method is no longer supported and should not be used.

Undo [Design]

Cancels the last design-level command.

UI Access	**Edit** > **Undo**
Parameters	None.
Return Value	None.

Python Syntax	Undo()
Python Example	`oDesign.Undo()`

VB Syntax	Undo
VB Example	`oDesign.Undo`

UpdateLink [EMIT]

Update the coupling link, retrieving any changes to the port list, setups, sweeps, and associated coupling data from the linked design.

UI Access	Right-click on the link and click **Update Link.**
Parameters	*<name>* – Name of the HFSS or HFSS 3D Layout design link to update
Return Value	None.

Python Syntax	UpdateLink(*<name>*)
Python Example	`oDesign.UpdateLink("HFSSDesign1")`

8 - Core Global Script Context Commands

To run these commands:

```
import CoreGlobalScriptContextFunctions

CoreGlobalScriptContextFunctions.[CommandName]
```

The following are general script commands recognized by the **CoreGlobalScriptContextFunctions** object:

AddErrorMessage

Adds an error message to the **Message Manager** window. AddErrorMessageis a function of CoreGlobalScriptContextFunctions.

UI Access	N/A
Parameters	Name | Type | Description <*message*> | String | Error message.
Return Value	None.

Python Syntax	AddErrorMessage(<*message*>)
Python Example	`import CoreGlobalScriptContextFunctions` `CoreGlobalScriptContextFunctions.AddErrorMessage('My error message.')`

VB Syntax	N/A
VB Example	N/A. Script cannot be run in VBScript.

AddFatalMessage

Adds a fatal error message to the **Message Manager** window. AddFatalMessageis a function of CoreGlobalScriptContextFunctions.

UI Access	N/A
Parameters	Name: *<message>* — Type: String — Description: Error message.
Return Value	None.

Python Syntax	AddFatalMessage(<*message*>)
Python Exampl-e	`import CoreGlobalScriptContextFunctions` `CoreGlobalScriptContextFunctions.AddFatalMessage('My fatal error message.')`

VB Syntax	N/A
VB Example	N/A. Script cannot be run in VBScript.

AddInfoMessage

Adds an informational message to the **Message Manager** window. AddInfoMessageis a function of CoreGlobalScriptContextFunctions.

UI Access	N/A
Parameters	Name: *<message>* — Type: String — Description: Informational message.
Return Value	None.

Python Syntax	AddInfoMessage(<*message*>)
Python Example	`import CoreGlobalScriptContextFunctions` `CoreGlobalScriptContextFunctions.AddInfoMessage('My info.')`

VB Syntax	N/A
VB Example	N/A. Script cannot be run in VBScript.

AddWarningMessage

Adds a warning message to the **Message Manager** window. AddWarningMessageis a function of CoreGlobalScriptContextFunctions.

UI Access	N/A
Parameters	Name | Type | Description <*message*> | String | Warning message.
Return Value	None.

Python Syntax	AddWarningMessge(<*message*>)
Python Example	`import CoreGlobalScriptContextFunctions` `CoreGlobalScriptContextFunctions.AddWarningMessage('My warning.')`

VB Syntax	N/A
VB Example	N/A. Script cannot be run in VBScript.

LogDebug

Adds a debug line to the log specified at **Tools** > **Debug Logging**. LogDebugis a function of CoreGlobalScriptContextFunctions.

UI Access	N/A
Parameters	Name | Type | Description <*message*> | String | Debug message.
Return Value	None.

Python Syntax	LogDebug(<*message*>)

Python Example	`import CoreGlobalScriptContextFunctions` `CoreGlobalScriptContextFunctions.LogDebug('My debug message.')`

VB Syntax	N/A
VB Example	N/A. Script cannot be run in VBScript.

LogError

Adds an error line to the log specified at **Tools** > **Debug Logging**. LogErroris a function of CoreGlobalScriptContextFunctions.

UI Access	N/A
Parameters	Name | Type | Description <*error*> | String | Error to log.
Return Value	None.

Python Syntax	LogError(<*error*>)
Python Example	`import CoreGlobalScriptContextFunctions` `CoreGlobalScriptContextFunctions.LogError('My error.')`

VB Syntax	N/A
VB Example	N/A. Script cannot be run in VBScript.

9 - Library Management Script Commands

Library Management script commands provide management functions for libraries in Personal Libraries. These functions are accessed via the Component Manager, which is accessed via the Definition Manager.

```
Set oDefinitionManager = oProject.GetDefinitionManager()

Set oComponentManager = oDefinitionManager.GetManager("Component")
```

The topics for this section include:

DeleteLibraryComponent

Deletes a component in a specified library.

UI Access	From the **Component Libraries** window, right-click a component displayed under **Personal Libraries** and select **Delete Library Component**.
Parameters	<table><tr><td>Name</td><td>Type</td><td>Description</td></tr><tr><td><*compName*></td><td>String</td><td>Name of the component to be deleted.</td></tr><tr><td><*libName*></td><td>String</td><td>Library name of the component to be deleted.</td></tr></table>
Return Value	None.

Python Syntax	DeleteLibraryComponent(<*compName*>,<*libName*>)
Python Example	`oComponentManager.DeleteLibraryComponent("Radio1", "Library1")`

VB Syntax	DeleteLibraryComponent <*compName*>, <*libName*>
VB Example	`oComponentManager.DeleteLibraryComponent "Radio1", "Library1"`

RenameLibraryComponent

Renames a component in a specified library.

<table>
<tr><td>UI Access</td><td>From the Component Libraries window, right-click a component displayed under Personal Libraries and select Rename Library Component.</td></tr>
<tr><td>Parameters</td><td>
<table>
<tr><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td><origCompName></td><td>String</td><td>Name of the component to be renamed.</td></tr>
<tr><td><libName></td><td>String</td><td>Library name of the component to be renamed.</td></tr>
<tr><td><newCompName></td><td>String</td><td>New name for the component.</td></tr>
</table>
</td></tr>
<tr><td>Return Value</td><td>None.</td></tr>
</table>

Python Syntax	RenameLibraryComponent(<*origCompName*>,<*libName*>,<*newCompName*>)
Python Example	`oComponentManager.RenameLibraryComponent("Radio1", "Library1", "NewRadio1")`

VB Syntax	RenameLibraryComponent <*origCompName*>, <*libName*>, <*newCompName*>
VB Example	`oComponentManager.RenameLibraryComponent "Radio1", "Library1", "NewRadio1"`

10 - Schematic Scripting for EMIT

The Schematic scripting interface is a set of commands that match the data changing methods available in the UI of the Schematic, plus some selection and query methods. Examples of the commands available through the scripting interface are adding items, removing items, and modifying items on the Schematic.

BringToFront

ChangeProperty

ClearSelections

Copy

CreateArc

CreateCircle

CreateComponent

CreateComponentAt

CreateCurve

CreateEllipse

CreateImage

CreateLine

CreatePolygon

CreateRectangle

CreateText

Cut

Delete

Editor Scripting IDs (EMIT)

Objects in the schematic are identified by text IDs. These IDs are used in scripts to perform actions on objects. These IDs are returned by all methods that add objects, and by the FindElements and GetSelections methods. The IDs are then passed into commands to modify or remove Schematic objects.

BringToFront (Schematic Editor)

Bring the selected object to the front of the view

UI Access	Draw>Bring to Front

Parameters	Name	Type	Description
	<Selections>	Array	["NAME: Selections", "Selections:=",[<Selected Components>]] <Selected Components> CompInst@DefName; ID, schematic ID , CompInst@DefName; ID, schematic ID CompInst@R;1,2
Return Value	None		

Note:

For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

Python Syntax	BringToFront( )
Python Example	oEditor.BringToFront(["NAME:Selections", "Selections:=", ["SchObj@1"]])

VB Syntax	BringToFront( )
VB Example	oEditor.BringToFront Array("NAME:Selections", "Selections:=", Array("SchObj@1"))

ChangeProperty [EMIT]

Change the value of a property for one of the primitives (such as. Line, Rectangle) in the Schematic.

UI Access	Properties in the Property Window
Parameters	*<args>* – The structure specifying the tab, item, and property to be changed and the new value(s) to apply.
Return Value	None.

Python Syntax	ChangeProperty(<args>)

Python Example	`oEditor.ChangeProperty(["NAME:AllTabs",`
	`["NAME:BaseElementTab", ["NAME:PropServers", "SchObj@26"],`
	`["NAME:ChangedProps", ["NAME:Color","R:=", 0,"G:=", 0,"B:=", 255]]`
	]])

ClearSelections

Clear all selections in the schematic.

UI Access	Click in an empty region in the schematic.
Parameters	None
Return Value	None

Python Syntax	ClearSelections()

Python Example	`oEditor.ClearSelections()`

VB Syntax	DeleteDesign
VB Example	`oDesign.DeleteDesign`

Copy [EMIT]

Copy items in the schematic for pasting.

UI Access	**Edit>Copy** or **Copy** button on Schematic ribbon
Parameters	*<elementNames>* – A list of component, wire, or primitive names to copy.
Return Value	None.

Python Syntax	Copy(*<elementNames>*)
Python Example	`oEditor.Copy(["Radio", "Antenna"])` `oEditor.Paste(-0.00127, 0.01651)`

CreateArc (Schematic Editor)

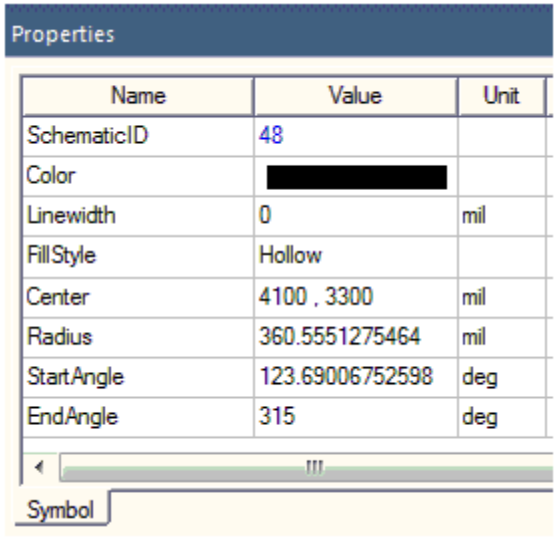
Create an arc

UI Access	**Draw>Primitive>Arc**

<table>
<tr><td>Parameters</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td><ArcData></td><td>Array</td><td>Array("NAME:ArcData" _
"x:=", double, _ // X position of the object
"y:=", double, _ // Y position of the object
"Radius:=", double, _ // Radius of the circle
"StartAng:=", double,_ // Start angle of the arc (radians)
"EndAng:=", double, _ // End angle of the arc (radians)
"LineWidth:=", double, // the width of the line, in meters
"Color:=", int, // the RBG value of the arc color
"Id:=", int), // [Opt=New id] Id for this item</td></tr>
<tr><td><Attributes></td><td>Array</td><td>Array("NAME:Attributes", _
"Page:=", int) _ // [Opt=1] Page number (one-based)
[out,retval] string id)</td></tr>
</table>
</td></tr>
<tr><td>Return Value</td><td>String
Unique id</td></tr>
</table>

Arc is created with the format SchObj@<schematicID>

The Schematic ID can be found on the **Symbols** tab of the **Properties** window when you select the arc.



Python Syntax	CreateArc()

Python Example	`oEditor.CreateArc`
	`(["NAME:ArcData", _`
	`"X:=", -0.004318, "Y:=", -0.00127, _`
	`"Radius:=", 0.00297299377732279, _`
	`"StartAng:=", 1.9195673303788, _`
	`"EndAng:=", 3.32144615338227, _`

	"Id:=", 10], _ ["NAME:Attributes", "Page:=", 1])

VB Syntax	CreateArc()
VB Example	oEditor.CreateArc Array("NAME:ArcData", _ "X:=", -0.004318, "Y:=", -0.00127, _ "Radius:=", 0.00297299377732279, _ "StartAng:=", 1.9195673303788, _ "EndAng:=", 3.32144615338227, _ "Id:=", 10), _ Array("NAME:Attributes", "Page:=", 1)

VB Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
```

```
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project4")
Set oDesign = oProject.SetActiveDesign("Circuit1")
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
Dim ArcID
ArcID = oEditor.CreateArc(Array("NAME:ArcData", "X:=", 0.10414, "Y:=", 0.08382, "Radius:=", _
0.00915810023967853, "StartAng:=", 2.15879893034246, "EndAng:=", _
5.49778714378214, "LineWidth:=", 0, "Color:=", 0, "Id:=", 48), Array("NAME:Attributes", "Page:=",
_
1))
MsgBox "Arc ID = " & ArcID
```

CreateCircle (Schematic Editor)

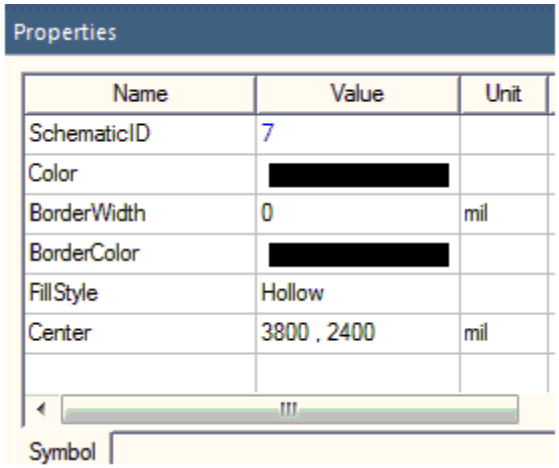
Create a circle

UI Access	**Draw>Primitive>Circle**		
Parameters	Name	Type	Description
	<CircleData>	Array	`Array("NAME:CircleData" _` "x:=", double, _ // X position of the object "y:=", double, _ // Y position of the object

<table>
<tr><td></td><td>
<table>
<tr><td></td><td></td><td>"Radius:=", double, _ // Radius of the circle
"LineWidth:=", double, // the width of the circle border, in meters
"Bordercolor:=", int, // the RBG value of the border color
"Fill:=", int, // the fill pattern id, 0 = hollow, 1 = solid, 2 = NEDiagonal,
3 = OrthoCross, 4 = DiagCross, 5 = NWDiagonal, 6 = Horizontal, 7 = Vertical
"Color:=", int, // the RBG value of the circle fill color
"Id:=", int), // [Opt=New id] Id for this item</td></tr>
<tr><td><Attributes></td><td>Array</td><td>Array("NAME:Attributes", _
"Page:=", int) _ // [Opt=1] Page number (one-based)
[out,retval] string id)</td></tr>
</table>
</td></tr>
<tr><td>Return Value</td><td>String
Unique id</td></tr>
</table>

Circle is created and the returned value has the format SchObj@<schematicID>

The schematic ID of the circle can be located in the **Properties** window on the **Symbols** tab.



Python Syntax	CreateCircle()

Python Example	`oEditor.CreateCircle(["NAME:CircleData", _`
	`"X:=", -0.004572, "Y:=", -0.000508, _`
	`"Radius:=", 0.001778, "Id:=", 12], _`
	`["NAME:Attributes", "Page:=", 1])`

VB Syntax	CreateCircle()

VB Example	oEditor.CreateCircle Array("NAME:CircleData", _ "X:=", -0.004572, "Y:=", -0.000508, _ "Radius:=", 0.001778, "Id:=", 12), _ Array("NAME:Attributes", "Page:=", 1)

CreateComponent [EMIT]

Create a new component of the given type in the schematic

UI Access	Component buttons on Schematic ribbon or dragging a component from the Component Libraries.
Parameters	*<desiredName>* – The desired name to assign to the new component. If an empty string (“”) is given, a name is assigned automatically. *<componentType>* – The type (for example, Cable) of component to create. *<libraryName [optional]>* – The library file (*.aclb) containing the specified component. If not specified or an empty string (“”), the syslib libraries are used.
Return Value	The name of the new component.

Python Syntax	CreateComponent(*<desiredName>, <componentType>*)
Python Example	oEditor.CreateComponent("", "New Radio") oEditor.CreateComponent("Cable C04", "Cable") oEditor.CreateComponent("Horn A01", "Antenna")

CreateComponentAt [EMIT]

Create a new component of the given type in the schematic

UI Access	Component buttons on Schematic ribbon or dragging a component from the Component Libraries.
Parameters	*<desiredName>* – The desired name to assign to the new component. If an empty string (“”) is given, a name is assigned automatically. *<xPosition>* – The x-location where the new component will be located on the schematic. *<yPosition>* – The x-location where the new component will be located on the schematic. *<componentType>* – The type (for example., Cable) of component to create. *<libraryName>* [optional] – The library file (*.aclb) containing the specified component. If not specified or an empty string (“”), the syslib libraries are used.
Return Value	The name of the new component.

Python Syntax	CreateComponentAt(*<desiredName>, <xPosition>*, *<yPosition>*, *<componentType>*)

Python Example	`oEditor.CreateEmitComponentAt("", -0.02032, -0.0254, "New Radio")`
	`oEditor.CreateEmitComponentAt("", -0.00762, -0.0254, "Cable")`
	`oEditor.CreateEmitComponentAt("", 0.00508, -0.02032, "Antenna")`

CreateCurve [Schematic]

Creates a Bezier curve.

<table>
<tr><td>UI Access</td><td>Draw > Primitive > Curve</td></tr>
<tr><td>Parameters</td><td>
<table>
<tr><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td><curve data></td><td>Array</td><td>("NAME:CurveData",
"Points:=", Array(), //Control Points of curve
"LineWidth:=", 0, //Linewidth of curve
"Color:=", 0, //Color
"Id:=", 1
)</td></tr>
<tr><td><attributes></td><td>Array</td><td>Array("NAME:Attributes", _
"Page:=", int) _ // [Opt=1] Page number
(one-based)</td></tr>
</table>
</td></tr>
<tr><td>Return Value</td><td>
<table>
<tr><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td><id></td><td>String</td><td>Unique ID of this object. Use it to perform actions (such as select and move) on object.</td></tr>
</table>
</td></tr>
</table>

Python Syntax	oEditor.CreateCurve([<*CurveData*>], [<*attributes*>])

Python Example	`curveObjId = oEditor.CreateCurve(`
	`  [`
	`  "NAME:CurveData",`
	`  "Points:=" , ["(-0.149860, 0.233680)","(-0.139700, 0.254000)","(-0.121920, 0.238760)","(-0.147320, 0.226060)","(-0.139700, 0.218440)","(-0.121920, 0.220980)"],`
	`  "LineWidth:=" , 0,`
	`  "Color:=" , 0,`

	"Id:=" , 7], ["NAME:Attributes", "Page:=" , 1])

VB Syntax	oEditor.CreateCurve Array(<CurveData>),Array(<attributes>)
VB Example	oEditor.CreateCurve Array("NAME:CurveData", "Points:=", Array("(0.048260, 0.050800)", _ "(0.076200, 0.073660)", "(0.091440, 0.033020)", "(0.060960, 0.038100)"), "LineWidth:=", _ 0, "Color:=", 0, "Id:=", 1), Array("NAME:Attributes", "Page:=", 1) or curveObjId = oEditor.CreateCurve(Array("NAME:CurveData", "Points:=", Array("(0.048260, 0.050800)", _ "(0.076200, 0.073660)", "(0.091440, 0.033020)", "(0.060960, 0.038100)"), "LineWidth:=", _ 0, "Color:=", 0, "Id:=", 1), Array("NAME:Attributes", "Page:=", 1))

CreateEllipse

Creates an ellipse.

UI Access	**Draw** > **Ellipse**.
Parameters	See table below
Return Value	None.

Name	Type	Description

<Parameters>	Array	Structured array.
		`Array("NAME:EllipseParameters",`
		`"IsCovered:=", <string>,`
		`"XCenter:=", <string>,`
		`"YCenter:=", <string>,`
		`"ZCenter:=", <string>,`
		`"MajRadius:=", <string>,`
		`"Ratio:=", <string>,`
		`"WhichAxis:=", <string>,`
		`"NumSegments:=", <string>)`

<AttributesArray>	Array	Structured array. See: AttributesArray.

Python Syntax	CreateEllipse(*<Parameters>*, *<Attributes>*)

Python Example	`oEditor.CreateEllipse(`
	`["NAME:EllipseParameters",`
	`"IsCovered:=" , True,`

```
  "XCenter:="             , "0.6mm",
  "YCenter:="             , "-0.6mm",
  "ZCenter:="             , "0mm",
  "MajRadius:="           , "0.2mm",
  "Ratio:="               , "7",
  "WhichAxis:="           , "Z",
  "NumSegments:="         , "0"
],
["NAME:Attributes",
  "Name:="                , "Ellipse1",
  "Flags:="               , "",
  "Color:="               , "(143 175 143)",
  "Transparency:="        , 0,
  "PartCoordinateSystem:=", "Global",
  "UDMId:="               , "",
  "MaterialValue:="       , "\"copper\"",
  "SurfaceMaterialValue:=", "\"\"",
  "SolveInside:="         , False,
  "ShellElement:="        , False,
  "ShellElementThickness:=", "0mm",
```

	"IsMaterialEditable:=" , True, "UseMaterialAppearance:=", False, "IsLightweight:=" , False])

VB Syntax	CreateEllipse <*Parameters*>, <*Attributes*>
VB Example	oEditor.CreateEllipse Array("NAME:EllipseParameters", "IsCovered:=", true, "XCenter:=", _ "-0.4mm", "YCenter:=", "-3.2mm", "ZCenter:=", "0mm", "MajRadius:=", "0.4mm", "Ratio:=", _ "0.5", "WhichAxis:=", "Z", "NumSegments:=", "0"), Array("NAME:Attributes", "Name:=", _ "Ellipse2", "Flags:=", "", "Color:=", "(143 175 143)", "Transparency:=", 0, "PartCoordinateSystem:=", _ "Global", "UDMId:=", "", "MaterialValue:=", "" & Chr(34) & "copper" & Chr(34) & "", "SurfaceMaterialValue:=", _ "" & Chr(34) & "" & Chr(34) & "", "SolveInside:=", false, "ShellElement:=", _ false, "ShellElementThickness:=", "0mm", "IsMaterialEditable:=", true, "UseMaterialAppearance:=", _ false, "IsLightweight:=", false)

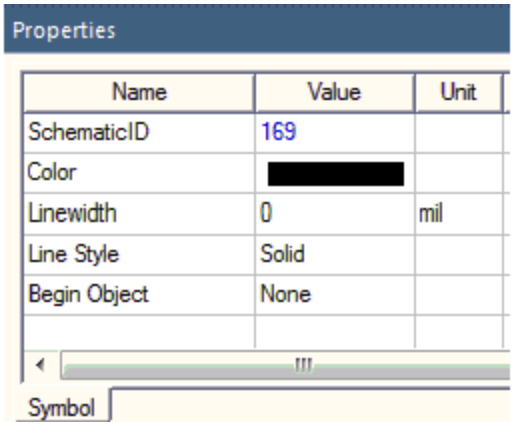
CreateLine (Schematic Editor)

Creates a line.

<table>
<tr><td>UI Access</td><td>Draw>Primitive>Line</td></tr>
<tr><td>Parameters</td><td>
<table>
<tr><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td><LineData></td><td>Array</td><td>Array("NAME:LineData" _
"Points:=", Array of points, _
"LineWidth:=", double, // the width of the line, in meters
"Color:=", int, // the RBG value of the line color
"Id:=", int), // [Opt=New id] Id for this item</td></tr>
<tr><td><Attributes></td><td>Array</td><td>Array("NAME:Attributes", _
"Page:=", int) _ // [Opt=1] Page number (one-based)
[out,retval] string id)</td></tr>
</table>
</td></tr>
<tr><td>Return Value</td><td>String
Unique id</td></tr>
</table>

Line is created and the returned value has the format SchObj@<schematicID>

The Schematic ID can be found on the **Symbol** tab of the **Properties** window when you select the line.



Python Syntax	CreateLine()

Python Example	`oEditor.CreateLine (["NAME:LineData",_`
	`"Points:=", ["(-0.055652, 0.020669)", _`
	`"(-0.036923, 0.011257)", "(-0.023144, 0.018049)"],_`
	`"LineWidth:=", 0, "Color:=", 0, "Id:=", 22)], _`
	`["NAME:Attributes", "Page:=", 1])`

VB Syntax	CreateLine()
VB Example	`oEditor.CreateLine Array("NAME:LineData",_`

	"Points:=", Array("(-0.055652, 0.020669)", _ "(-0.036923, 0.011257)", "(-0.023144, 0.018049)"),_ "LineWidth:=", 0, "Color:=", 0, "Id:=", 22), _ Array("NAME:Attributes", "Page:=", 1)

VB Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project4")
Set oDesign = oProject.SetActiveDesign("Circuit1")
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
LineID = oEditor.CreateLine(Array("NAME:LineData", "Points:=", Array("(0.050800, 0.106680)", _
"(0.124460, 0.106680)", "(0.124460, 0.106680)"), "LineWidth:=", 0, "Color:=", _
0, "Id:=", 169), Array("NAME:Attributes", "Page:=", 1))
```

```
MsgBox "LineID = " & LineID
```

CreatePolygon (Schematic Editor)

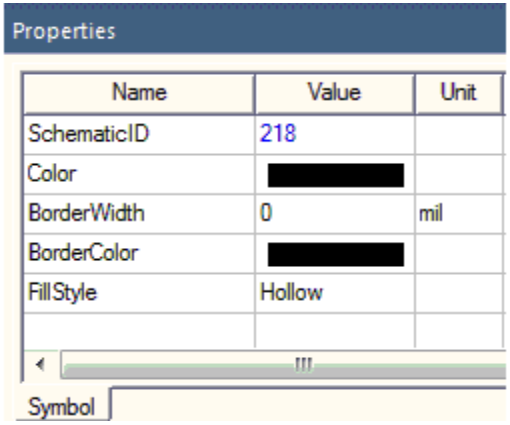
Create a polygon

<table>
<tr><td>UI Access</td><td colspan="3">Draw>Primitive>Polygon</td></tr>
<tr><td>Parameters</td><td colspan="3"><table>
<tr><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td><PolygonData></td><td>Array</td><td>Array("NAME:PolygonData" _
"Points:=", Array of points, _
"LineWidth:=", double, // the width of the Polygon border, in meters
"Bordercolor:=", int, // the RBG value of the border color
"Fill:=", int, // the fill pattern id, 0 = hollow, 1 = solid, 2 = NEDiagonal,
3 = OrthoCross, 4 = DiagCross, 5 = NWDiagonal, 6 = Horizontal, 7 = Vertical
"Color:=", int, // the RBG value of the Polygon fill color
"Id:=", int), // [Opt=New id] Id for this item</td></tr>
<tr><td><Attributes></td><td>Array</td><td>Array("NAME:Attributes", _
"Page:=", int) _ // [Opt=1] Page number (one-based)
[out,retval] string id)</td></tr>
</table></td></tr>
<tr><td>Return Value</td><td colspan="3">String
Unique id</td></tr>
</table>

Polygon is created and the return value has the format

SchObj@<schematicID>

The Schematic ID can be found in the **Properties** window on the **Symbol** tab when you select the polygon.



Python Syntax	CreatePolygon()

Python Example	`oEditor.CreatePolygon(`
	`["NAME:PolygonData", "Points:=", [ _`
	`"(-0.058951, 0.006308)", "(-0.046142, 0.010480)",_`
	`"(-0.046239, 0.001067)"), "LineWidth:=", _`
	`0, "BorderColor:=", 0, "Fill:=", 0, "Color:=",_`
	`0, "Id:=", 27], ["NAME:Attributes", "Page:=", 1]])`

VB Syntax	CreatePolygon()
VB Example	oEditor.CreatePolygon Array("NAME:PolygonData", "Points:=", Array(_ "(-0.058951, 0.006308)", "(-0.046142, 0.010480)",_ "(-0.046239, 0.001067)"), "LineWidth:=", _ 0, "BorderColor:=", 0, "Fill:=", 0, "Color:=",_ 0, "Id:=", 27), Array("NAME:Attributes", "Page:=", 1)

VB Example:

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project4")
Set oDesign = oProject.SetActiveDesign("Circuit1")
```

```
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
polygonID = oEditor.CreatePolygon(Array("NAME:PolygonData", "Points:=", Array( _
"(0.101600, 0.058420)", "(0.101600, 0.060960)", "(0.101600, 0.058420)", _
"(0.099060, 0.060960)", "(0.101600, 0.060960)", "(0.101600, 0.058420)", _
"(0.101600, 0.060960)", "(0.101600, 0.060960)"), "LineWidth:=", 0, "BorderColor:=", _
0, "Fill:=", 0, "Color:=", 0, "Id:=", 218), Array("NAME:Attributes", "Page:=", 1))
MsgBox "Polygon ID = " & polygonID
```

CreateRectangle (Schematic Editor)

Create a rectangle

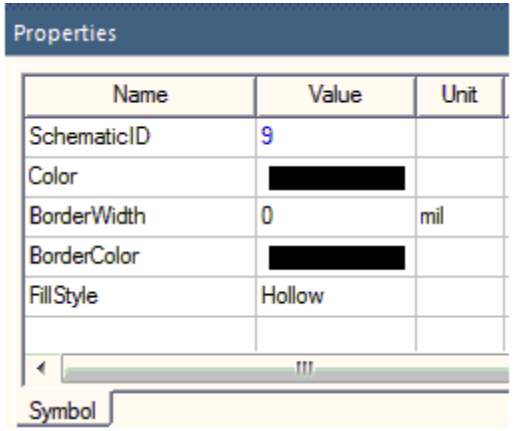
<table>
<tr><td>UI Access</td><td>Draw>Primitive>Rectangle</td></tr>
<tr><td>Parameters</td><td>
<table>
<tr><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td><RectData></td><td>Array</td><td>Array("NAME:RectData" _
"x1:=", double, _ // X position of the upper left
"y1:=", double, _ // Y position of the upper left
"x2:=", double, _ // X position of the lower right
"y2:=", double",_ // Y position of the lower right
"LineWidth:=", double, // the width of the Rectangle border, in meters
"Bordercolor:=", int, // the RBG value of the border color
"Fill:=", int, // the fill pattern id, 0 = hollow, 1 = solid, 2 = NEDiagonal,
3 = OrthoCross, 4 = DiagCross, 5 = NWDiagonal, 6 = Horizontal, 7 = Vertical</td></tr>
</table>
</td></tr>
</table>

			"Color:=", int, // the RBG value of the Rectangle fill color "Id:=", int), // [Opt=New id] Id for this item
	<Attributes>	Array	`Array("NAME:Attributes", _` "Page:=", int) _ // [Opt=1] Page number (one-based) [out,retval] string id)
Return Value	String Unique id		

Rectangle is created and the return value has the format

SchObj@<schematicID>

The Schematic ID can be found in the **Properties** window on the **Symbol** tab when you select the rectangle.



Python Syntax	CreateRectangle()
Python Example	oEditor.CreateRectangle (["NAME:RectData", "X1:=", -0.0755449856733525, "Y1:=", _ 0.0335755491881566, "X2:=", -0.0611831900668577, "Y2:=", _ 0.0254242597898758, "LineWidth:=", _ 0, "BorderColor:=", 0, "Fill:=", 0, "Color:=", 0, "Id:=", 31], _ ["NAME:Attributes", "Page:=", 1])

VB Syntax	CreateRectangle()
VB Example	oEditor.CreateRectangle Array("NAME:RectData", "X1:=", -0.0755449856733525, "Y1:=", _ 0.0335755491881566, "X2:=", -0.0611831900668577, "Y2:=", _ 0.0254242597898758, "LineWidth:=", _ 0, "BorderColor:=", 0, "Fill:=", 0, "Color:=", 0, "Id:=", 31), _ Array("NAME:Attributes", "Page:=", 1)

VB Example:

```
Dim oAnsoftApp
Dim oDesktop
```

```
Dim oProject

Dim oDesign

Dim oEditor

Dim oModule

Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")

Set oDesktop = oAnsoftApp.GetAppDesktop()

oDesktop.RestoreWindow

Set oProject = oDesktop.SetActiveProject("Project4")

Set oDesign = oProject.SetActiveDesign("Circuit1")

Set oEditor = oDesign.SetActiveEditor("SchematicEditor")

rectangleID = oEditor.CreateRectangle(Array("NAME:RectData", "X1:=", 0.1016, "Y1:=", 0.0635,
"X2:=", _

0.10414, "Y2:=", 0.06096, "LineWidth:=", 0, "BorderColor:=", 0, "Fill:=", 0, "Color:=", _

0, "Id:=", 9), Array("NAME:Attributes", "Page:=", 1))

MsgBox "Rectangle ID = " & rectangleID
```

CreateText (Schematic Editor)

Create text

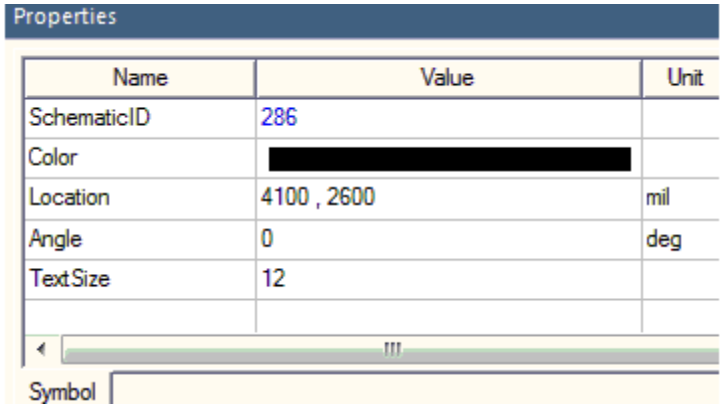
<table>
<tr><td>UI Access</td><td colspan="3">Draw>Primitive>Text</td></tr>
<tr><td>Parameters</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td></td><td><TextData></td><td>Array</td><td>Array("NAME:TextData" _
"x:=", double, _ // X position of the object</td></tr>
</table>

			"y:=", double, _ // Y position of the object
			"Text:=", string, _ // Text to display
			"Color:=", int, // the RBG value of the text color
			"Id:=", int), // Id for this item
			"ShowRect:=", bool, // true or false, whether to show the highlight rectangle for the text
			"X1:=", double, // the text rectangle left X value, in meters
			"Y1:=", double, // the text rectangle upper Y value, in meters
			"X2:=", double, // the text rectangle right X value, in meters
			"Y2:=", double, // the text rectangle lower Y value, in meters
			"RectLineWidth:=", double, // the width of the rectangle border, in meters
			"RectBordercolor:=", int, // the RBG value of the rectangle border color
			"RectFill:=", int, // the rectangle fill pattern id, 0 = hollow, 1 = solid,
			2 = NEDiagonal, 3 = OrthoCross, 4 = DiagCross, 5 = NWDiagonal, 6 = Horizontal, 7 = Vertical
			"RectColor:=", int, // the RBG value of the rectangle fill color

	<Attributes>	Array	`Array("NAME:Attributes", _` "Page:=", int) _ // [Opt=1] Page number (one-based) [out,retval] string id)
Return Value	String Unique id		

Text is created and the return value has the format: SchObj@<SchematicID>

The Schematic ID can be found in the **Properties** window on the **Symbols** tab when you select the text.



Python Syntax	CreateText()
Python Example	oEditor.CreateText(["NAME:TextData", "X:=", -0.0764183381088826, "Y:=", _ 0.040174212034384, "Size:=", 12, "Angle:=",_ 0, "Text:=", "Control Circuit", "Color:=", _ 0, "Id:=", 34, "ShowRect:=", false, "X1:=", _ -0.0793287547755609, "Y1:=", _ 0.0407033787010528, "X2:=", -0.0502245881087778,_

	"Y2:=", 0.035411712034365, "RectLineWidth:=", _ 0, "RectBorderColor:=", 0, "RectFill:=", 0, "RectColor:=", 0], ["NAME:Attributes", "Page:=", 1])

VB Syntax	CreateText()
VB Example	oEditor.CreateText Array("NAME:TextData", "X:=", -0.0764183381088826, "Y:=", _ 0.040174212034384, "Size:=", 12, "Angle:=",_ 0, "Text:=", "Control Circuit", "Color:=", _ 0, "Id:=", 34, "ShowRect:=", false, "X1:=", _ -0.0793287547755609, "Y1:=", _ 0.0407033787010528, "X2:=", -0.0502245881087778,_ "Y2:=", 0.035411712034365, "RectLineWidth:=", _ 0, "RectBorderColor:=", 0, "RectFill:=", 0, "RectColor:=", 0), Array("NAME:Attributes", "Page:=", 1)

VB Example:

```
Dim oAnsoftApp
Dim oDesktop
```

```
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Set oAnsoftApp = CreateObject("Ansoft.ElectronicsDesktop")
Set oDesktop = oAnsoftApp.GetAppDesktop()
oDesktop.RestoreWindow
Set oProject = oDesktop.SetActiveProject("Project4")
Set oDesign = oProject.SetActiveDesign("Circuit1")
Set oEditor = oDesign.SetActiveEditor("SchematicEditor")
textobj = oEditor.CreateText(Array("NAME:TextData", "X:=", 0.10414, "Y:=", 0.06604, "Size:=", _
12, "Angle:=", 0, "Text:=", "CreateDefaultTectxt" & Chr(13) & Chr(10) & "", "Color:=", _
0, "Id:=", 286, "ShowRect:=", false, "X1:=", 0.100141851851836, "Y1:=", _
0.0670983333333376, "X2:=", 0.140123333333477, "Y2:=", 0.0565149999999619, "RectLineWidth:=", _
0, "RectBorderColor:=", 0, "RectFill:=", 0, "RectColor:=", 0), Array("NAME:Attributes",
"Page:=", _
1))
MsgBox "text = " & textobj
```

Cut [EMIT]

Cut items in the schematic for pasting.

UI Access	**Edit>Cut** or **Cut** button on Schematic ribbon.

Parameters	*<elementNames>* – A list of component, wire, or primitive names to cut
Return Value	None.

Python Syntax	Cut(*<elementNames>, <elementNames>*)
Python Example	`oEditor.Cut(["Radio", "Antenna"])` `oEditor.Paste(-0.00127, 0.01651)`

Delete [EMIT]

Delete items from the schematic.

UI Access	**Edit>Delete** or **Delete** button on Schematic ribbon.
Parameters	*<elementNames>* – A list of component, wire, or primitive names to delete.
Return Value	The name of the new component.

Python Syntax	Delete(*<elementNames>, <elementNames>*)
Python Example	`oEditor.Delete(["Radio", "Antenna"])`

DisableEmitComponents [EMIT]

Disables/enables specified EMIT component(s). This command only works for emitters and radios.

UI Access	**Edit** > **Disable** or **Edit** > **Enable**
Parameters	<*componentNames*> – String component name, or Array of strings containing the components to select. <*Disable/Enable*> – Boolean True to disable; False to enable.
Return Value	None.

Python Syntax	DisableEmitComponents(<*componentNames*>)
Python Example	`oEditor.DisableEmitComponents("Radio1", True)` `oEditor.DisableEmitComponents(["Radio1","Radio2","Radio3"], False)`

VB Syntax	DisableEmitComponents <*componentNames*>
VB Example	`oEditor.DisableEmitComponents "Radio1", True` `oEditor.DisableEmitComponents Array("Radio1","Radio2","Radio3"), False`

ExportImage (EMIT)

To export a picture for a specified page of the current design to a file. The image size can also be specified. The filename extension determines the type of image exported.

UI Access	None.
Parameters	

Name	Type	Description
<filename>	String	The name of the file, with format-specific extension. Extensions supported are: bmp, gif, jpg, jpeg, png, tif, tiff.
<dx>	Integer	The width of the image. If `dx` is less than 160, 160 will be used for the width.
<dy>	Integer	The height of the image. If `dy` is less than 160, 160 will be used for the height.

Return Value	None

> **Note:**
>
> For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs.

Python Syntax	ExportImage (<filename>, <dx>, <dy>)
Python Example	`oEditor.ExportImage ("c:\mysch.png", 800, 400)`

VB Syntax	ExportImage (<filename>, <dx>, <dy>)
VB Example	`oEditor.ExportImage "c:\mysch.png", 800, 400`

ExportToLibrary [EMIT]

Export a component from the schematic to a new or existing library. If the component already exists in the specified library, update it.

UI Access	Right-click on a component in the schematic and select **Export to Library**.
Parameters	<*componentName*> – The name of the component to export. <*libraryType*> – The type of library to create or update: “PersonalLib” or “UserLib”. <*libraryFile*> – The relative path to the library files within the PersonalLib or UserLib directory. If a path to a sub-directory is specified, the directory must exist.

Return Value	None.

Python Syntax	ExportToLibrary(<*componentName*>, <*libraryType*>, <*libraryFile*>)
Python Example	`oEditor.ExportToLibrary("Radio1", "PersonalLib", "Library1")`

GetAllComponents [EMIT]

Get a list of all components in the schematic.

UI Access	N/A
Parameters	None.
Return Value	A list of component names.

Python Syntax	GetAllComponents()
Python Example	`for component_name in oEditor.GetAllComponents():` `  print(component_name)`

GetAllElements [EMIT]

Get a list of all element (such as component, wire, or primitive) names in the schematic.

UI Access	N/A
Parameters	None.
Return Value	A list of element names.

Python Syntax	GetAllComponents()
Python Example	`for component_name in oEditor.GetAllElements():` `  print(element_name)`

GetComponentBoundingBox [EMIT]

Get the location and extents of a component within the schematic.

UI Access	N/A
Parameters	*<componentName>* – The name of the component in the schematic.
Return Value	A list containing a pair of points that specify the bottom-left corner of the component and the component width and height: *[[xBottom, yBottom], [width, height]]*

Python Syntax	GetComponentBoundingBox(*<componentName>*)
Python Example	`oEditor.GetComponentBoundingBox("Radio")`

GetComponentLocation [EMIT]

Get the location of a component within the schematic.

UI Access	N/A
Parameters	*<componentName>* – The name of the component in the schematic.

Return Value	A list containing the x and y location of the component. The location returned is the component origin, typically the center of the component. *[xLocation, yLocation]*

Python Syntax	GetComponentLocation(<*componentName*>)
Python Example	`oEditor.GetComponentLocation("Radio")`

GetComponentOrientation [EMIT]

Get the orientation of a component within the schematic.

UI Access	N/A
Parameters	<*componentName*> – The name of the component in the schematic.
Return Value	An integer describing the orientation state of the component. The first orientation state is 0 and this corresponds to the default orientation of the component – the orientation when it is first placed in the schematic. The 'Reorient' button in the Schematic ribbon cycles through these states.

Python Syntax	GetComponentOrientation(<*componentName*>)
Python Example	`oEditor.GetComponentOrientation("Multiplexer")`

GetComponentPortLocation [EMIT]

Get the location of a component port within the schematic.

UI Access	N/A
Parameters	<*componentName*> – The name of the component in the schematic.

	<*portName*> – The name of the component port.
Return Value	A list containing the x and y location of the component port (that is., the point where wiring connections can be made). *[xLocation, yLocation]*

Python Syntax	GetComponentPortLocation(<*componentName*>, <*portName*>)
Python Example	`oEditor.GetComponentPortLocation("Cable", "n2")`

GetComponentPorts [EMIT]

Get the names of all ports for a component within the schematic.

UI Access	N/A
Parameters	<*componentName*> – The name of the component in the schematic.
Return Value	A list containing the names of all component ports. *["n1", "n2"]*

Python Syntax	GetComponentPorts(<*componentName*>)
Python Example	`oEditor.GetComponentPorts("Cable")`

GetEditorName (Schematic Editor)

Get the name of the schematic editor.

UI Access	N/A

Parameters	None
Return Value	String containing the name of the schematic editor. “SchematicEditor”

Python Syntax	GetEditorName()
Python Example	`oEditor.GetEditorName()`

VB Syntax	GetEditorName([out, retval] string) // name of the editor
VB Example	`dim info` `info = oEditor.GetEditorName`

GetNumComponentOrientations [EMIT]

Get the number of possible orientations for a component within the schematic.

UI Access	N/A
Parameters	*<componentName> – The name of the component in the schematic*.
Return Value	An integer describing the number of possible orientation states of the component. The ‘Reorient’ button in the Schematic ribbon cycles through these states. Some components cannot be re-oriented and thus have only 1 possible orientation state.

Python Syntax	GetNumComponenetOrientations(*<componentName>*)
Python Example	`oEditor.GetNumComponentOrientations("Multiplexer")`

GetPropertyValue [EMIT]

Get the value of a property for one of the primitives (for example. Line, Rectangle, etc.) in the schematic.

UI Access	Properties in the Property Window.
Parameters	*<tabArg>* – The structure specifying the tab, item, and property to be changed and the new value(s) to apply. *<serverArg>* – The name of the element (see GetAllElements and GetSelections). Only primitive-type elements (for example., Rectangle, Line) may be used. Components are not supported. *<propArg>* – The name of the property.
Return Value	The current value for the property. The return type varies depending on the type of property.

Python Syntax	GetPropertyValue(*<tabArg>*, *<serverArg>*, *<propArg>*)
Python Example	`line = self.oEditor.GetSelections()[0]` `oEditor.GetPropertyValue("BaseElementTab", line, "LineWidth")`

GetSelections [EMIT]

Get a list of all selected element (component, wire, or primitive) names in the schematic.

UI Access	NA.
Parameters	None.
Return Value	A list of selected element names.

Python Syntax	GetSelections()
Python Example	`for element_name in oEditor.GetSelections():` `  print(element_name)`

GetWireAtPort [EMIT]

Get the name of the wire connected to a component port within the schematic.

UI Access	N/A.
Parameters	<*componentName*> – The name of the component in the schematic. <*portName*> – The name of the component port
Return Value	The name of the wire connected to the component port (that is, the point where wiring connections can be made). An empty string (“”) is returned if the port has no wire connected to it. .

Python Syntax	GetWireAtPort<*componentName*>, <*portName*>)
Python Example	`oEditor.GetWireAtPort("Cable", "n2")`

GetWireConnections [EMIT]

Get the name of the components and ports connected by the specified wire.

UI Access	N/A.
Parameters	<*wireElementName*> – The name of the wire in the schematic. See GetAllElements, GetSelections, and GetWireAtPort for getting wire names.
Return Value	A length-two list of component name, port name pairs. *[[“Radio”, “n1”], [“Cable”, “n1”]] .*

Python Syntax	GetWireConnections*(<wireElementName>)*
Python Example	`oEditor.GetWireConnections("Wire@net_1;6")`

GetZoomArea [EMIT]

Get the location and extents of the current zoom area (visible portion) of the schematic.

UI Access	N/A.
Parameters	None.
Return Value	A list containing a pair of points that specify the bottom-left corner of the zoom area and the zoom area width and height: *[[xBottom, yBottom], [width, height]]* .

Python Syntax	GetZoomArea*()*
Python Example	`oEditor.GetZoomArea()`

IsEmitComponentDisabled [EMIT]

Checks whether a radio or emitter is disabled.

UI Access	**N/A**
Parameters	*<componentNames>* – String name of component to check. Disable/enable only works for emitters and radios.
Return Value	BOOL True if component is disabled; False if component is enabled.

Python Syntax	IsEmitComponentDisabled(<*componentNames*>)
Python Example	`oEditor.IsEmitComponentDisabled("Radio")`

VB Syntax	IsEmitComponentDisabled <*componentNames*>
VB Example	`oEditor.IsEmitComponentDisabled "Radio"`

Move [EMIT]

Move items in the schematic by a specified amount.

UI Access	Drag selected items within the schematic.
Parameters	<*elementNames*> – A list of elements to move. See GetSelected and GetAllElements for getting names of schematic items. <*xDelta*> – The distance to move the items in the x direction. <*yDelta*> – The distance to move the items in the y direction.
Return Value	None.

Python Syntax	Move*([<elementNames>], <xDelta>, <yDelta>)*
Python Example	`oEditor.Move(["Radio", "Antenna"], -0.1524, 0.0)`

Pan [EMIT]

Adjust the current zoom area (visible region) by the specified amount

UI Access	**View>Pan** or **Pan** on the View ribbon.
Parameters	*<xDelta>* – The distance to move the view in the x direction. *<yDelta>* – The distance to move the view in the y direction.
Return Value	None.

Python Syntax	Pan*(<xDelta>, <yDelta>)*
Python Example	`oEditor.Pan(-0.1524, 0.0)`

Paste [EMIT]

Paste cut or copied items from the schematic at a specified location

UI Access	**Edit>Paste** or **Paste** in the Schematic ribbon.
Parameters	*<xPosition>* – The x location to paste the items. *<yPosition> – The y location to paste the items.*.
Return Value	None.

Python Syntax	Paste*(<xPosition>, <yPosition>)*
Python Example	`oEditor.Paste(-0.00127, 0.01651)`

PlaceComponent [EMIT]

Place the specified component in a new location, such that it is connected to the next suitable port or in a new row at the bottom of the schematic.

UI Access	Reposition in the Schematic ribbon
Parameters	*<componentName>* – The component to place at the next open port or new row at the bottom of the schematic. If an empty string (“”) is passed, the selected component is used. *<componentName>* – The component to be connected to the first one, if possible; this specifies the component to attach to instead of attaching to the first component it finds. Optional.
Return Value	None.

Python Syntax	PlaceComponent*(<componentName>,<componentName>)*
Python Example	`oEditor.PlaceComponent("Antenna1","Radio 5")`

RenameComponent [EMIT]

Rename the specified component.

UI Access	Properties Window property or name label below the component in the schematic.
Parameters	*<oldName>* – The name of the component to be renamed. *<newName>* – The new name for the component.
Return Value	None.

Python Syntax	RenameComponent*(<oldName>, <newName>)*
Python Example	`oEditor.RenameComponent("Antenna", "New Antenna Name")`

ReorientComponent [EMIT]

Set the orientation of a component within the schematic.

UI Access	**Reorient** button in the Schematic ribbon.
Parameters	*<componentName>* – The name of the component in the schematic. *<orientation>* – An integer specifying the orientation state to apply to the component. The first orientation state is 0 and this corresponds to the default orientation of the component – the orientation when it is first placed in the schematic. The **Reorient** button in the Schematic ribbon cycles through these states. See GetNumComponentOrientations for valid values.
Return Value	None.

Python Syntax	ReorientComponent*(<componentName>, <orientation>)*
Python Example	`oEditor.ReorientComponent("Multiplexer", 1)`

Select [EMIT]

Select the specified element in the schematic.

UI Access	Click or Shift+Click on an element in the schematic.
Parameters	*<elementName>* – The name of the element in the schematic to select. *<clearExistingSelection>* – If True, de-select any existing selections before selecting the specified element. If

	False, preserve the existing selection(s) and additional select the specified element.
Return Value	None.

Python Syntax	Select*(<elementName>, <clearExistingSelection>)*
Python Example	`oEditor.Select("Multiplexer", True) # Select a multiplexer only` `oEditor.Select("Radio", False) # Also select a Radio`

SelectAll [EMIT]

Select all elements in the schematic.

UI Access	**Edit>Select All**.
Parameters	None.
Return Value	None.

Python Syntax	SelectAll()
Python Example	`oEditor.SelectAll()`

SendToBack(Schematic Editor)

Send the selected object to the back of the view

UI Access	Draw>Send To Back		
Parameters	Name	Type	Description

	<Object>	String	Object to send to the back.
Return Value	None		

Note:

For more information about the component name, schematic IDs, and formats, see the section Editor Scripting IDs

Python Syntax	SendToBack (["NAME:Selections", "Selections:=", [<Object>, <Object>, ...]])
Python Example	`oDefinitionEditor.SendToBack(["NAME:Selections",` `"Selections:=", ["SchObj@10"]])`

VB Syntax	SendToBack Array("NAME:Selections", "Selections:=", Array (<Object>, <Object>, ...))
VB Example	`oDefinitionEditor.SendToBack Array("NAME:Selections",` `"Selections:=", Array( "SchObj@10"))`

SetPropertyValue [EMIT]

Set the value of a property for one of the primitives (such as Line, Rectangle) in the Schematic.

UI Access	Properties in the Property Window.
Parameters	*<tabArg>* – The structure specifying the tab, item, and property to be changed and the new value(s) to apply.

	<*serverArg*> – The name of the element (see GetAllElements and GetSelections). Only primitive-type elements (such as Rectangle, Line) may be used. Components are not supported. <*propArg*> – The name of the property. <*valueArg*> – The new value for the property.
Return Value	None.

Python Syntax	SetPropertyValue(<*tabArg*>, <*serverArg*>, <*propArg*>, <*valueArg*>)

Python Example	`line = self.oEditor.GetSelections()[0]`
	
	`oEditor.SetPropertyValue("BaseElementTab", line, "LineWidth", 0.001)`

Wire [EMIT]

Create a wire between the specified ports of two components in the schematic.

UI Access	Click a port and then click a second port to create a wire. Click at additional intermediate points to customize the wire route.
Parameters	<*component1*> – The name of the first component to connect with a wire. <*port1*> – The port of the first component where one end of the new wire should connect. <*component2*> – The name of the second component to connect with a wire. <*port2*> – The port of the second component where the other end of the new wire should connect. <*segmentPointList*> – An empty list ([]) if the wire should connect directly from one port to the other (ports must be aligned horizontally). Any bends in the wire are specified by including the points (x,y location of the bend) in this list.
Return Value	The name of the new wire.

Python Syntax	Wire(<*component1*>, <*port1*>, <*component2*>, <*port2*>, [(<segmentPointList>)])
Python Example	`oEditor.Wire("Radio", "n1", "Cable", "n1")` `oEditor.Wire("Antenna", "in", "Cable", "n2", [[0.01524, -0.01016]])`

ZoomArea [EMIT]

Set the location and extents of the current zoom area (visible portion) of the schematic.

UI Access	**View>Zoom Area** or the **Zoom Area** button on the View ribbon.
Parameters	<*pt1*> – A list specifying the lower left corner of the zoom area ([left, bottom]). <*ptDelta*> – A list [width, height] specifying the size of the zoom area. If the width and height do not match the aspect ratio of the view area, the view area will be configured to be at least as large as both values.
Return Value	None.

Python Syntax	ZoomArea([<*pt1*>], [<*ptDelta*>])
Python Example	`oEditor.ZoomArea([-0.0480, -0.0356], [0.0864, 0.0489])`

ZoomIn [EMIT]

Zoom in at the specified location in the schematic.

UI Access	**View>Zoom In** or the **Zoom In** button on the View ribbon.

Parameters	<*pt1*> – The center point of the new zoom area. Optional – if not specified, the center of the view area is used. <*zoomFactor*> – The factor by which the view area should be reduced (a value greater than 1).
Return Value	None.

Python Syntax	ZoomIn([<*pt1*>], <*zoomFactor*>)
Python Example	`oEditor.ZoomIn()` `oEditor.ZoomIn([-0.0480, -0.0356], 1.1)`

ZoomOut [EMIT]

Zoom out around the specified location in the schematic.

UI Access	**View>Zoom Out** or the **Zoom Out** button on the View ribbon.
Parameters	<*pt1*> – The center point of the new zoom area. Optional – if not specified, the center of the view area is used. <*zoomFactor*> – The factor by which the view area should be increased (a value greater than 1).
Return Value	None.

Python Syntax	ZoomOut([<*pt1*>], <*zoomFactor*>)
Python Example	`oEditor.ZoomArea([-0.0480, -0.0356], [0.0864, 0.0489])` `oEditor.ZoomOut()`

ZoomPrevious [EMIT]

Zoom to the last view area, prior to the most recent zoom event.

UI Access	**View>Zoom Previous** or the **Zoom Previous** button on the View ribbon.
Parameters	None.
Return Value	None.

Python Syntax	ZoomPrevious()
Python Example	`oEditor.ZoomPrevious()`

ZoomToFit [EMIT]

Zoom so that the view area contains all elements in the schematic with a small margin around the edges.

UI Access	**View>Fit All** or the **Fit All** button on the View ribbon.
Parameters	None.
Return Value	None.

Python Syntax	ZoomToFit()
Python Example	`oEditor.ZoomToFit()`

Index

H

S