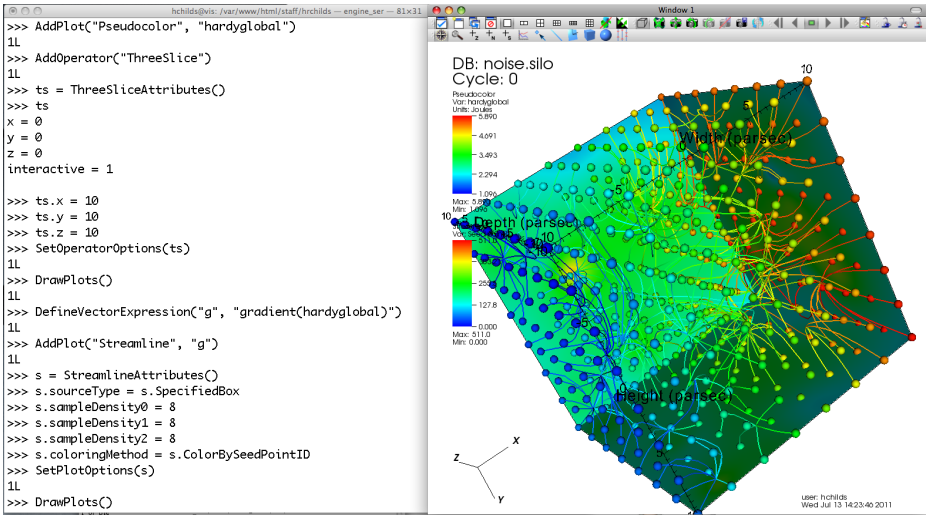
VisIt Python Interface Manual



Version 2.5.2

Contents

Chapter 1

Introduction to VisIt

1.1 Overview

VisIt is a distributed, parallel, visualization tool for visualizing data defined on two and three-dimensional structured and unstructured meshes. VisIt's distributed architecture allows it to leverage both the compute power of a large parallel computer and the graphics acceleration hardware of a local workstation. Another benefit of the distributed architecture is that VisIt can visualize the data where it is generated, eliminating the need to move data. VisIt can be controlled by a Graphical User Interface (GUI) or through the Python scripting language. More information about VisIt's Graphical User Interface can be found in the *VisIt User's Manual*.

1.2 Manual chapters

This manual is broken down into the following chapters:

Chapter title	Chapter description
Introduction to VisIt	This chapter.
Python	Describes the basic features of the Python programming language.
Quick Recipes	Describes common patterns for scripting using the VisIt Python Interface.
Functions	Describes functions in the VisIt Python Interface.
Attributes References	Describes attributes for setting common operations, as well as for VisIt's plugins
CLI Events	Describes possible events for callbacks.

1.3 Understanding how VisIt works

VisIt visualizes data by creating one or more plots in a visualization window, also known as a vis window. Examples of plots include Mesh plots, Contour plots and Pseudocolor plots. Plots take as input one or more mesh, material, scalar, or tensor variables. It is possible to modify the variables by applying one or

more operators to the variables before passing them to a plot. Examples of operators include arithmetic operations or taking slices through the mesh. It is also possible to restrict the visualization of the data to subsets of the mesh. VisIt provides Python bindings to all of its plots and operators so they may be controlled through scripting. Each plot or operator plugin provides a function, which is added to the VisIt namespace, to create the right type of plot or operator attributes. The attribute object can then be modified by setting its fields and then it can be passed to a general-purpose function to set the plot or operator attributes. To display a complete list of functions in the VisIt Python Interface, you can type dir() at the Python prompt. Similarly, to inspect the contents of any object, you can type its name at the Python prompt. VisIt supports up to 16 visualization windows, also called vis windows. Each vis window is independent of the other vis windows and VisIt Python functions generally apply only to the currently active vis window. This manual explains how to use the VisIt Python Interface which is a Python extension module that controls VisIt's viewer. In that way, the VisIt Python Interface fulfills the same role as VisIt's GUI. The difference is that the viewer is totally controlled through Python scripting, which makes it easy to write scripts to create visualizations and even movies. Since the VisIt module controls VisIt's viewer, the Python interpreter currently has no direct mechanism for passing data to the compute engine (see Figure 1.1). If you want to write a script that generates simulation data and have that script pass data to the compute engine, you must pass the data through a file on disk. The VisIt Python Interface comes packaged in two varieties: the extension module and the Command Line Interface (CLI). The extension module version of the VisIt Python Interface is imported into a standard Python interpreter using the import directive. VisIt's command line interface (CLI) is essentially a Python interpreter where the VisIt Python Interface is built-in. The CLI is provided to simplify the process of running VisIt Python scripts.

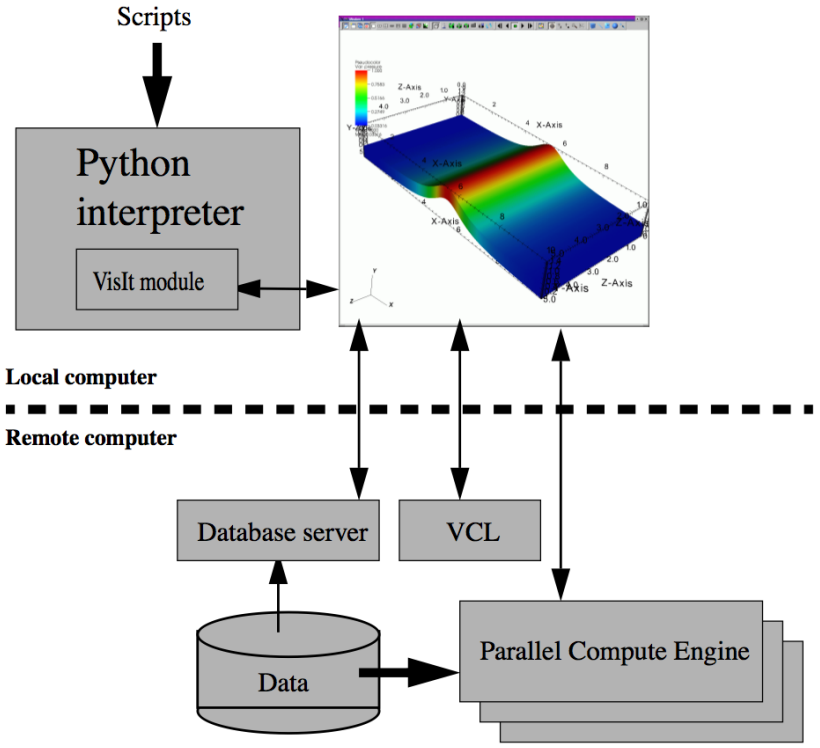


Figure 1.1: VisIt's architecture

1.4 Starting VisIt

You can invoke VisIt's command line interface from the command line by typing:

```
visit -cli
```

VisIt provides a separate Python module if you instead wish to include VisIt functions in an existing Python script. In that case, you must first import the VisIt module into Python and then call the Launch() function to make VisIt launch and dynamically load the rest of the VisIt functions into the Python namespace. VisIt adopts this somewhat unusual approach to module loading since the lightweight "visit" front-end module can be installed as one of your Python's site packages yet still dynamically load the real control functions from different versions of VisIt selected by the user.

If you do not install the visit.so module as a Python site package, you can tell the Python interpreter where it is located by appending a new path to the sys.path variable. Be sure to substitute the correct path to visit.so on your system.

```
import sys
sys.path.append("/path/to/visit/<version>/<architecture>/lib")
```

Here is how to import all functions into the global Python namespace:

```
from visit import *
Launch()
```

Here is how to import all functions into a "visit" module namespace:

```
import visit
visit.Launch()
```

1.5 Getting started

VisIt is a tool for visualizing 2D and 3D scientific databases. The first thing to do when running VisIt is select databases to visualize. To select a database, you must first open the database using the OpenDatabase function. After a window has an open database, any number of plots and operators can be added. To create a plot, use the AddPlot function. After adding a plot, call the DrawPlots function to make sure that all of the new plots are drawn.

Example:

```
OpenDatabase("/usr/local/visit/data/multi_curv3d.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
```

To see a list of the available plots and operators when you use the VisIt Python Interface, use the Operator Plugins and Plot Plugins functions. Each of those functions returns a tuple of strings that contain the names of the currently loaded plot or operator plugins. Each plot and operator plugin provides a function for creating an attributes object to set the plot or operator attributes. The name of the function is the name of the plugin in the tuple returned by the OperatorPlugins or PlotPlugins functions plus the word "Attributes". For example, the "Pseudocolor" plot provides a function called PseudocolorAttributes. To set the plot attributes or the operator attributes, first use the attributes creation function to create an attributes object. Assign the newly created object to a variable name and set the fields in the object. Each object has its own set of fields. To see the available fields in an object, print the name of the variable at the Python prompt and press the Enter key. This will print the contents of the object so you can see the fields contained by the object. After setting the

appropriate fields, pass the object to either the SetPlotOptions function or the SetOperatorAttributes function.

Example:

```
OpenDatabase("/usr/local/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
AddOperator("Slice")
p = PseudocolorAttributes()
p.colorTableName = "rainbow"
p.opacity = 0.5
SetPlotOptions(p)
a = SliceAttributes()
a.originType = a.Point
a.normal, a.upAxis = (1,1,1), (-1,1,-1)
SetOperatorOptions(a)
DrawPlots()
```

That's all there is to creating a plot using VisIt's Python Interface. For more information on creating plots and performing specific actions in VisIt, refer to the documentation for each function later in this manual.

Chapter 2

Python

2.1 Overview

Python is a general purpose, interpreted, extensible, object-oriented scripting language that was chosen for VisIt's scripting language due to its ease of use and flexibility. VisIt's Python interface was implemented as Python module and it allows you to enhance your Python scripts with coding to control VisIt. This chapter explains some of Python's syntax so it will be more familiar when you examine the examples found in this document. For more information on programming in Python, there are a number of good references, including on the Internet at http://www.python.org.

2.2 Indentation

One of the most obvious features of Python is its use of indentation for new scopes. You must take special care to indent all program logic consistently or else the Python interpreter may halt with an error, or worse, not do what you intended. You must increase indentation levels when you define a function, use an if/elif/else statement, or use any loop construct.

Note the different levels of indentation:

```
def example_function(n):
  v = 0
  if n > 2:
    print "n greater than 2."
  else:
    v = n * n
  return v
```

2.3 Comments

Like all good programming languages, Python supports the addition of comments in the code. Comments begin with a pound character (#) and continue to the end of the line.

```
# This is a comment
a = 5 * 5
```

2.4 Identifiers

The Python interpreter accepts any identifier that contains letters 'A'-'Z', 'a'-'z' and numbers '0'-'9' as long as the identifier does not begin with a number. The Python interpreter is case-sensitive so the identifier "case" would not be the same identifier as "CASE". Be sure to case consistently throughout your Python code since the Python interpreter will instantiate any identifier that it has not seen before and mixing case would cause the interpreter to use multiple identifiers and cause problems that you might not expect. Identifiers can be used to refer to any type of object since Python is flexible in its treatment of types.

2.5 Data types

Python supports a wide variety of data types and allows you to define your own data types readily. Most types are created from a handful of building-block types such as integers, floats, strings, tuples, lists, and dictionaries.

2.5.1 Strings

Python has built-in support for strings and you can create them using single quotes or double quotes. You can even use both types of quotes so you can create strings that include quotes in case quotes are desired in the output. Strings are sequence objects and support operations that can break them down into characters.

```
s = 'using single quotes'
s2 = "using double quotes"
s3 = 'nesting the "spiffy" double quotes'
```

2.5.2 Tuples

Python supports tuples, which can be thought of as a read-only set of objects. The members of a tuple can be of different types. Tuples are commonly used to group multiple related items into a single object that can be passed around more easily. Tuples support a number of operations. You can subscript a tuple like an array to access its individual members. You can easily determine whether an object is a member of a tuple. You can iterate over a tuple. There are many more uses for tuples. You can create tuples by enclosing a comma-separated list of objects in parenthesis.

```
# Create a tuple
a = (1,2,3,4,5)
print "The first value in a is:", a[0]
# See if 3 is in a using the "in" operator.
print "3 is in a:", 3 in a
# Create another tuple and add it to the first one to create c.
b = (6,7,8,9)
c = a + b
# Iterate over the items in the tuple
for value in c:
  print "value is: ", value
```

2.5.3 Lists

Lists are just like tuples except they are not read-only and they use square brackets [] to enclose the items in the list instead of using parenthesis.

```
# Start with an empty list.
L = []
for i in range(10):
  # Add i to the list L
  L = L + [i]
print L
# Assign a value into element 6
L[5] = 1000
print L
```

2.5.4 Dictionaries

Dictionaries are Python containers that allow you to store a value that is associated with a key. Dictionaries are convenient for mapping 1 set to another set since they allow you to perform easy lookups of values. Dictionaries are declared using curly braces and each item in the dictionary consists of a key: value pair with the key and values being separated by a colon. To perform a lookup using a dictionary, provide the key whose value you want to look up to the subscript [] operator.

```
colors = {"red" : "rouge", "orange" : "orange", \
"yellow" : "jaune", "green" : "vert", "blue" : "bleu"}
# Perform lookups using the keys.
for c in colors.keys():
   print "%s in French is: %s" % (c, colors[c])
```

2.6 Control flow

Python, like other general-purpose programming languages provides keywords that implement control flow. Control flow is an important feature to have in a programming language because it allows complex behavior to be created using a minimum amount of scripting.

2.6.1 if/elif/else

Python provides if/elif/else for conditional branching. The if statement takes any expression that evaluates to an integer and it takes the if branch if the integer value is 1 other wise it takes the else branch if it is present.

```
# Example 1
if condition:
   do_something()

# Example 2
if condition:
   do_something()
else:
   do_something_else()
```

```
# Example 3
if condition:
   do_domething()
elif conditionn:
   do_something_n()
...
else:
   do_something_else()
```

2.6.2 For loop

Python provides a for loop that allows you to iterate over all items stored in a sequence object (tuples, lists, strings). The body of the for loop executes once for each item in the sequence object and allows you to specify the name of an identifier to use in order to reference the current item.

```
# Iterating through the characters of a string
for c in "characters":
   print c

# Iterating through a tuple
for value in ("VisIt", "is", "coolness", "times", 100):
   print value

# Iterating through a list
for value in ["VisIt", "is", "coolness", "times", 100]:
   print value

# Iterating through a range of numbers [0,N) created with range(N).
N = 100
for i in range(N):
   print i, i*i
```

2.6.3 While loop

Python provides a while loop that allows you to execute a loop body indefinitely based on some condition. The while loop can be used for iteration but can also be used to execute more complex types of loops.

```
token = get_next_token()
while token != "":
  do_something(token)
  token = get_next_token()
```

2.7 Functions

Python comes with many built-in functions and modules that implement additional functions. Functions can be used to execute bodies of code that are meant to be re-used. Functions can optionally take arguments and can optionally return values. Python provides the def keyword, which allows you to

define a function. The def keyword is followed by the name of the function and its arguments, which should appear as a tuple next to the name of the function.

```
# Define a function with no arguments and no return value.
def my_function():
   print "my function prints this..."

# Define a function with arguments and a return value.
def n_to_the_d_power(n, d):
  value = 1
  if d > 0:
  for i in range(d):
    value = value * n
  elif d < 0:
    value = 1. / float(n_to_the_d_power(n, -d))

return value
```

Chapter 3

Quick Recipes

3.1 Overview

This manual contains documentation for over two hundred functions and several dozen extension object types. Learning to combine the right functions in order to accomplish a visualization task without guidance would involve hours of trial and error. To maximize productivity and start creating visualizations using Visit's Python Interface as fast as possible, this chapter provides some common patterns, or "quick recipes" that you can combine to quickly create complex scripts.

3.2 How to start

The most important question when developing a script is: "Where do I start?". You can either use session files that you used to save the state of your visualization to initialize the plots before you start scripting or you can script every aspect of plot initialization.

3.2.1 Using session files

VisIt's session files contain all of the information required to recreate plots that have been set up in previous interactive VisIt sessions. Since session files contain all of the information about plots, etc., they are natural candidates to make scripting easier since they can be used to do the hard part of setting up the complex visualization, leaving the bulk of the script to animate through time or alter the plots in some way. To use session files within a script, use the RestoreSession function.

```
# Import a session file from the current working directory.
RestoreSesssion("my_visualization.session", 0)
# Now that VisIt has restored the session, animate through time.
for states in range(TimeSliderGetNStates()):
  SetTimeSliderState(state)
  SaveWindow()
```

3.2.2 Getting something on the screen

If you don't want to use a session file to begin the setup for your visualization then you will have to dive into opening databases, creating plots, and animating through time. This is where all of hand-crafted scripts begin. The first step in creating a visualization is opening a database. VisIt provides the OpenDatabase function to open a database. Once a database has been opened, you can create plots from its variables using the AddPlot function. The AddPlot function takes a plot plugin name and the name of a variable from the open database. Once you've added a plot, it is in the new state, which means that it has not yet been submitted to the compute engine for processing. To make sure that the plot gets drawn, call the DrawPlots function.

```
# Step 1: Open a database
OpenDatabase("/usr/local/visit/data/wave.visit")

# Step 2: Add plots
AddPlot("Pseudocolor", "pressure")
AddPlot("Mesh", "quadmesh")

# Step 3: Draw the plots
DrawPlots()

# Step 4: Animate through time and save images
for states in range(TimeSliderGetNStates()):
  SetTimeSliderState(state)
  SaveWindow()
```

3.3 Saving images

Much of the time, the entire purpose of using VisIt's Python Interface is to create a script that can save out images of a time-varying database for the purpose of making movies. Saving images using VisIt's Python Interface is a straight-forward process, involving just a few functions.

3.3.1 Setting the output image characteristics

VisIt provides a number of options for saving files, including: fileformat, filename, and image size, to name a few. These attributes are grouped into the SaveWindowAttributes object. To set the options that VisIt uses to save files, you must create a SaveWindowAttributes object, change the necessary attributes, and call the SetSaveWindowAttributes function. Note that if you want to create images using a specific image resolution, the best way is to use the *-geometry* command line argument with VisIt's Command Line Interface and tell VisIt to use screen capture. If you instead require your script to be capable of saving everal different image sizes then you can turn off screen capture and set the image resolution in the SaveWindowAttributes object.

```
# Save a BMP file at 1024x768 resolution
s = SaveWindowAttributes()
s.format = s.BMP
s.fileName = "mybmpfile"
s.width, s.height = 1024,768
s.screenCapture = 0
SetSaveWindowAttributes(s)
```

3.3.2 Saving an image

Once you have set the SaveWindowAttributes to your liking, you can call the SaveWindow function to save an image. The SaveWindow function returns the name of the image that is saved so you can use that for other purposes in your script.

```
# Save images of all timesteps and add each image filename to a list.
names = []
for state in range(TimeSliderGetNStates()):
  SetTimeSliderState(state)
  # Save the image
  n = SaveWindow()
  names = names + [n]
print names
```

3.4 Working with databases

VisIt allows you to open a wide array of databases both in terms of supported file formats and in terms how databases treat time. Databases can have a single time state or can have multiple time states. Databases can natively support multiple time states or sets of single time states files can be grouped into time-varying databases using .visit files or using virtual databases. Working with databases gets even trickier if you are using VisIt to visualize a database that is still being generated by a simulation. This section describes how to interact with databases.

3.4.1 Opening a database

Opening a database is a relatively simple operation - most complexities arise in how the database treats time. If you only want to visualize a single time state or if your database format natively supports multiple timestates per file then opening a database requires just a single call to the OpenDatabase function.

```
# Open a database at time state 0
OpenDatabase("/usr/local/visit/data/allinone00.pdb")
```

3.4.2 Opening a database at late time

Opening a database at a later timestate is done just the same as opening a database at time state zero except that you must specify the time state at which you want to open the database. There are a number of reasons for opening a database at a later time state. The most common reason for doing so, as opposed to just changing time states later, is that VisIt uses the metadata from the first opened time state to describe the contents of the database for all timestates (except for certain file formats that don't do this, i.e. SAMRAI). This means that the list of variables found for the first time state that you open is used for all timestates. If your database contains a variable at a later timestate that does not exist at earlier time states, you must open the database at a later time state to gain access to the transient variable.

```
# Open a database at a later time state to pick up transient variables
OpenDatabase("/usr/local/visit/data/wave.visit", 17)
```

3.4.3 Opening a virtual database

VisIt provides two ways for accessing a set of single time-state files as a single time- varying database. The first method is a .visit file, which is a simple text file that contains the names of each file to be used as a time state in the time-varying database. The second method uses "virtual databases", which allow VisIt to exploit the file naming conventions that are often employed by simulation codes when they create their dumps. In many cases, VisIt can scan a specified directory and determine which filenames look related. Filenames with close matches are grouped as individual time states into a virtual database whose name is based on the more abstract pattern used to create the filenames.

```
# Opening first file in series wave0000.silo, wave0010.silo, ...
OpenDatabase("/usr/local/visit/data/wave0000.silo")

# Opening a virtual database representing all wave*.silo files.
OpenDatabase("/usr/local/visit/data/wave*.silo database.)
```

3.4.4 Opening a remote database

VisIt supports running the client on a local computer while also allowing you to process data in parallel on a remote computer. If you want to access databases on a remote computer using VisIt's Python Interface, the only difference to accessing a database on a local computer is that you must specify a host name as part of the database name.

```
# Opening a file on a remote computer by giving a host name
# Also, open the database to a later time slice (17)
OpenDatabase("thunder:/usr/local/visit/data/wave.visit", 17)
```

3.4.5 Opening a compute engine

Sometimes it is advantageous to open a compute engine before opening a database. When you tell VisIt to open a database using the OpenDatabase function, VisIt also launches a compute engine and tells the compute engine to open the specified database. When the VisIt Python Interface is run with a visible window, the **Engine Chooser Window** will present itself so you can select a host profile. If you want to design a script that must specify parallel options, etc in batch mode where there is no **Engine ChooserWindow** then you have few options other than to open a compute engine before opening a database. To open a compute engine, use the OpenComputeEngine function. You can pass the name of the host on which to run the compute engine and any arguments that must be used to launch the engine such as the number of processors.

```
# Open a remote, parallel compute engine before opening a database
OpenComputeEngine("mcr", ("-np", "4", "-nn", "2"))
OpenDatabase("mcr:/usr/local/visit/data/multi_ucd3d.silo")
```

3.5 Working with plots

Plots are viewable objects, created from a database, that can be displayed in a visualization window. VisIt provides several types of plots and each plot allows you to view data using different visualization techniques. For example, the Pseudocolor plot allows you to see the general shape of a simulated object while painting colors on it according to the values stored in a variable's scalar field. The most important functions for interacting with plots are covered in this section.

3.5.1 Creating a plot

The function for adding a plot in VisIt is: AddPlot. The AddPlot function takes the name of a plot type and the name of a variable that is to be plotted and creates a new plot and adds it to the plot list. The name of the plot to be created corresponds to the name of one of VisIt's plot plugins, which can be queried using the PlotPlugins function. The variable that you pass to the AddPlot function must be a valid variable for the opend atabase. New plots are not realized, meaning that they have not been submitted to the compute engine for processing. If you want to force VisIt to process the new plot you must call the DrawPlots function.

```
# Names of all available plot plugins
print PlotPlugins()
# Create plots
AddPlot("Pseudocolor", "pressure")
AddPlot("Mesh", "quadmesh")
# Draw the plots
DrawPlots()
```

3.5.2 Plotting materials

Plotting materials is a common operation in VisIt. The Boundary and FilledBoundary plots enable you to plot material boundaries and materials, respectively.

```
# Plot material boundaries
AddPlot("Boundary", "mat1")
# Plot materials
AddPlot("FilledBoundary", "mat1")
```

3.5.3 Setting plot attributes

Each plot type has an attributes object that controls how the plot generates its data or how it looks in the visualization window. The attributes object for each plot contains different fields. You can view the individual object fields by printing the object to the console. Each plot type provides a function that creates a new instance of one of its attribute objects. The function name is always of the form: plotname + "Attributes". For example, the attributes object creation function for the Pseudocolor plot would be: PseudocolorAttributes. To change the attributes for a plot, you create an attributes object using the appropriate function, set the properties in the returned object, and tell VisIt to use the new plot attributes by passing the object to the SetPlotOptions function. Note that you should set a plot's attributes before calling the DrawPlots method to realize the plot since setting a plot's attributes can cause the compute engine to recalculate the plot.

```
# Creating a Pseudocolor plot and setting min/max values.
AddPlot("Pseudocolor", "pressure")
p = PseudocolorAttributes()
# Look in the object
print p
# Set the min/max values
p.min, p.minFlag = 1, 0.0
p.max, p.maxFlag = 1, 10.0
SetPlotOptions(p)
```

3.5.4 Working with multiple plots

When you work with more than one plot, it is sometimes necessary to set the active plots because some of VisIt's functions apply to all of the active plots. The active plot is usually the last plot that was created unless you've changed the list of active plots. Changing which plots are active is useful when you want to delete or hide certain plots or set their plot attributes independently. When you want to set which plots are active, use the SetActivePlots function. If you want to list the plots that you've created, call the ListPlots function.

```
# Create more than 1 plot of the same type
AddPlot("Pseudocolor", "pressure")
AddPlot("Pseudocolor", "density")

# List the plots. The second plot should be active.
ListPlots()

# Draw the plots
DrawPlots()

# Hide the first plot
SetActivePlots(0)
HideActivePlots()

# Set both plots' color table to "hot"
p = PseudocolorAttributes()
p.colorTableName = "hot"
SetActivePlots((0,1))
SetPlotOptions(p)

# Show the first plot again.
SetActivePlots(0)
HideActivePlots()

# Delete the second plot
SetActivePlots(1)
DeleteActivePlots()
ListPlots()
```

3.5.5 Plots in the error state

When VisIt's compute engine cannot process a plot, the plot is put into the error state. Once a plot is in the error state, it no longer is displayed in the visualization window. If you are generating a movie, plots entering the error state can be a serious problem because you most often want all of the plots that you have created to animate through time and not disappear in the middle of the animation. You can add extra code to your script to prevent plots from disappearing (most of the time) due to error conditions by adding a call to the DrawPlots function.

```
# Save an image and take care of plots that entered the error state.
drawThePlots = 0
for state in range(TimeSliderGetNStates()):
  if SetTimeSliderState(state) == 0:
    drawThePlots = 1
```

```
if drawThePlots == 1:
  if DrawPlots() == 0:
    print "VisIt could not draw plots for state: %d" % state
  else:
    drawThePlots = 0
SaveWindow()
```

3.6 Operators

Operators are filters that are applied to database variables before the compute engine uses them to create plots. Operators can be linked one after the other to form chains of operators that can drastically transform the data before plotting it.

3.6.1 Adding operators

Adding an operator is similar to adding a plot in that you call a function with the name of the operator to be added. The list of available operators is returned by the OperatorPlugins function. Any of the names returned in that plugin can be used to add an operator using the AddOperator function. Operators are added to the active plots by default but you can also force VisIt to add them to all plots in the plot list.

```
# Print available operators
print OperatorPlugins()
# Create a plot
AddPlot("Pseudocolor")
# Add an Isovolume operator and a Slice operator
AddOperator("Isovolume")
AddOperator("Slice")
DrawPlots()
```

3.6.2 Setting operator attributes

Each plot gets its own instance of an operator which means that you can set each plot's operator attributes independently. Like plots, operators use objects to set their attributes. These objects are returned by functions whose names are of the form: operatorname + "Attributes". Once you have created an operator attributes object, you can pass it to the SetOperatorOptions to set the options for an operator. Note that setting the attributes for an operator nearly always causes the compute engine to recalculate the operator. You can use the power of VisIt's Python Interface to create complex operator behavior such as in the following code example, which moves slice planes through a Pseudocolor plot.

```
OpenDatabase("/usr/local/visit/data/noise.silo")
AddPlot("Pseudocolor", "hardyglobal")
AddOperator("Slice")
s = SliceAttributes()
s.originType = s.Percent
s.project2d = 0
SetOperatorOptions(s)
DrawPlots()
```

```
nSteps = 20
for axis in (0,1,2):
  s.axisType = axis
  for step in range(nSteps):
  t = float(step) / float(nSteps - 1)
  s.originPercent = t * 100.
  SetOperatorOptions(s)
  SaveWindow()
```

3.7 Quantitative operations

This section focuses on some of the operations that allow you to examine your data more quantitatively.

3.7.1 Defining expressions

VisIt allows you to create derived variables using its powerful expressions language. You can plot or query variables created using expressions just as you would if they were read from a database. VisIt's Python Interface allows you to create new scalar, vector, tensor variables using the DefineScalarExpression, DefineVectorExpression, and DefineTensorExpression functions.

```
# Creating a new expression
OpenDatabase("/usr/local/visit/data/noise.silo")
AddPlot("Pseudocolor", "hardyglobal")
DrawPlots()
DefineScalarExpression("newvar", "sin(hardyglobal) + cos(shepardglobal")
ChangeActivePlotsVar("newvar")
```

3.7.2 Pick

VisIt allows you to pick on cells, nodes, and points within a database and reutrn information for the item of interest. To that end, VisIt provides several pick functions. Once a pick function has been called, you can call the GetPickOutput function to get a string that contains the pick information. The information in the string could be used for a multitude of uses such as building a test suite for a simulation code.

```
OpenDatabase("/usr/local/visit/data/noise.silo")
AddPlot("Pseudocolor", "hgslice")
DrawPlots()
s = []
# Pick by a node id
PickbyNode(300)
s = s + [GetPickOutput()]
# Pick by a cell id
PickByZone(250)
s = s + [GetPickOutput()]
# Pick on a cell using a 3d point
Pick((-2., 2., 0.))
s = s + [GetPickOutput()]
# Pick on the node closest to (-2,2,0)
```

```
NodePick((-2,2,0))
s = s + [GetPickOutput()]
# Print all pick results
print s
```

3.7.3 Lineout

VisIt allows you to extract data along a line, called a lineout, and plot the data using a Curve plot.

```
OpenDatabase("/usr/local/visit/data/noise.silo")
AddPlot("Pseudocolor", "hgslice")
DrawPlots()
Lineout((-5,-3), (5,8))
# Specify a number of sample points
Lineout((-5,-4), (5,7))
```

3.7.4 Query

VisIt can perform a number of different queries based on values calculated about plots or their originating database.

```
OpenDatabase("/usr/local/visit/data/noise.silo")
AddPlot("Pseudocolor", "hardyglobal")
DrawPlots()
Query("NumNodes")
print "The float value is: %g" % GetQueryOutputValue()
Query("NumNodes")
```

3.7.5 Finding the min and the max

A common operation in debugging a simulation code is examining the min and max values. Here is a pattern that allows you to print out the min and the max values and their locations in the database and also see them visually.

```
# Define a helper function to get the id's of the MinMax query.
def GetMinMaxIds():
  Query("MinMax")
  import string
  s = string.split(GetQueryOutputString(), " ")
  retval = []
  nextGood = 0
  idType = 0
  for token in s:
    if token == "(zone" or token == "(cell":
      idType = 1
      nextGood = 1
      continue
    elif token == "(node":
      idType = 0
      nextGood = 1
      continue
```

```
     if nextGood == 1:
        nextGood = 0
        retval = retval + [(idType, int(token))]
  return retval

# Set up a plot
OpenDatabase("/usr/local/visit/data/noise.silo")
AddPlot("Pseudocolor", "hgslice")
DrawPlots()

# Do picks on the ids that were returned by MinMax.
for ids in GetMinMaxIds():
  idType = ids[0]
  id = ids[1]
  if idType == 0:
    PickByNode(id)
  else:
  PickByZone(id)
```

3.8 Subsetting

VisIt allows the user to turn off subsets of the visualization using a number of different methods. Databases can be divided up any number of ways: domains, materials, etc. This section provides some details on how to remove materials and domains from your visualization.

3.8.1 Turning off domains

VisIt's Python Interface provides the TurnDomainsOn and TurnDomainsOff functions to make it easy to turn domains on and off.

```
OpenDatabase("/usr/local/visit/data/multi_rect2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
# Turning off all but the last domain
d = GetDomains()
for dom in d[:-1]:
TurnDomainsOff(dom)
# Turn all domains off
TurnDomainsOff()
# Turn on domains 3,5,7
TurnDomainsOn((d[3], d[5], d[7]))
```

3.8.2 Turning off materials

VisIt's Python Interface provides the TurnMaterialsOn and TurnMaterialsOff functions to make it easy to turn materials on and off.

```
OpenDatabase("/usr/local/visit/data/multi_rect2d.silo")
AddPlot("FilledBoundary", "mat1")
DrawPlots()
```

```
# Print the materials are:
GetMaterials()
# Turn off material 2
TurnMaterialsOff("2")
```

3.9 View

Setting up the view in your Python script is one of the most important things you can do to ensure the quality of your visualization because the view concentrates attention on an object or inferest. VisIt provides different methods for setting the view, depending on the dimensionality of the plots in the visualization window but despite differences in how the view is set, the general procedure is basically the same.

3.9.1 Setting the 2D view

The 2D view consists of a rectangular window in 2D space and a 2D viewport in the visualization window. The window in 2D space determines what parts of the visualization you will look at while the viewport determines where the images will appear in the visualization window. It is not necessary to change the viewport most of the time.

```
OpenDatabase("/usr/local/visit/data/noise.silo")
AddPlot("Pseudocolor", "hgslice")
AddPlot("Mesh", "Mesh2D")
AddPlot("Label", "hgslice")
DrawPlots()
print "The current view is:", GetView2D()
# Get an initialized 2D view object.
v = GetView2D()
v.windowCoords = (-7.67964, -3.21856, 2.66766, 7.87724)
SetView2D(v)
```

3.9.2 Setting the 3D view

The 3D view is much more complex than the 2D view. For information on the actual meaning of the fields in the View3DAttributes object, refer to page 214 or the VisIt User's Manual. VisIt automatically computes a suitable view for 3D objects and it is best to initialize new View3DAttributes objects using the GetView3D function so most of the fields will already be initialized. The best way to get new views to use in a script is to interactively create the plot and repeatedly call GetView3D() after you finish rotating the plots with the mouse. You can paste the printed view information into your script and modify it slightly to create sophisticated view transitions.

```
OpenDatabase("/usr/local/visit/data/noise.silo")
AddPlot("Pseudocolor", "hardyglobal")
AddPlot("Mesh", "Mesh")
DrawPlots()
v = GetView3D()
print "The view is: ", v
v.viewNormal = (-0.571619, 0.405393, 0.713378)
v.viewUp = (0.308049, 0.911853, -0.271346)
SetView3D(v)
```

3.9.3 Flying around plots

Flying around plots is a commonly requested feature when making movies. Fortunately, this is easy to script. The basic method used for flying around plots is interpolating the view. VisIt provides a number of functions that can interpolate View2DAttributes and View3DAttributes objects. The most useful of these functions is the EvalCubicSpline function. The EvalCubicSpline function uses piece-wise cubic polynomials to smoothly interpolate between a tuple of N like items. Scripting smooth view changes using EvalCubicSpline is rather like keyframing in that you have a set of views that are mapped to some distance along the parameterized space [0., 1.]. When the parameterized space is sampled with some number of samples, VisIt calculates the view for the specified parameter value and returns a smoothly interpolated view. One benefit over keyframing, in this case, is that you can use cubic interpolation whereas VisIt's keyframing mode currently uses linear interpolation.

```
# Do a pseudocolor plot of u.
OpenDatabase("/usr/local/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()

# Create the control points for the views.
c0 = View3DAttributes()
c0.viewNormal = (0, 0, 1)
c0.focus = (0, 0, 0)
c0.viewUp = (0, 1, 0)
c0.viewAngle = 30
c0.parallelScale = 17.3205
c0.nearPlane = 17.3205
c0.farPlane = 81.9615
c0.perspective = 1

c1 = View3DAttributes()
c1.viewNormal = (-0.499159, 0.475135, 0.724629)
c1.focus = (0, 0, 0)
c1.viewUp = (0.196284, 0.876524, -0.439521)
c1.viewAngle = 30
c1.parallelScale = 14.0932
c1.nearPlane = 15.276
c1.farPlane = 69.917
c1.perspective = 1

c2 = View3DAttributes()
c2.viewNormal = (-0.522881, 0.831168, -0.189092)
c2.focus = (0, 0, 0)
c2.viewUp = (0.783763, 0.556011, 0.27671)
c2.viewAngle = 30
c2.parallelScale = 11.3107
c2.nearPlane = 14.8914
c2.farPlane = 59.5324
c2.perspective = 1

c3 = View3DAttributes()
c3.viewNormal = (-0.438771, 0.523661, -0.730246)
c3.focus = (0, 0, 0)
c3.viewUp = (-0.0199911, 0.80676, 0.590541)
```

```
c3.viewAngle = 30
c3.parallelScale = 8.28257
c3.nearPlane = 3.5905
c3.farPlane = 48.2315
c3.perspective = 1

c4 = View3DAttributes()
c4.viewNormal = (0.286142, -0.342802, -0.894768)
c4.focus = (0, 0, 0)
c4.viewUp = (-0.0382056, 0.928989, -0.36813)
c4.viewAngle = 30
c4.parallelScale = 10.4152
c4.nearPlane = 1.5495
c4.farPlane = 56.1905
c4.perspective = 1

c5 = View3DAttributes()
c5.viewNormal = (0.974296, -0.223599, -0.0274086)
c5.focus = (0, 0, 0)
c5.viewUp = (0.222245, 0.97394, -0.0452541)
c5.viewAngle = 30
c5.parallelScale = 1.1052
c5.nearPlane = 24.1248
c5.farPlane = 58.7658
c5.perspective = 1

c6 = c0

# Create a tuple of camera values and x values. The x values
# determine where in [0,1] the control points occur.
cpts = (c0, c1, c2, c3, c4, c5, c6)
x=[]
for i in range(7):
  x = x + [float(i) / float(6.)]

# Animate the view using EvalCubicSpline.
nsteps = 100
for i in range(nsteps):
  t = float(i) / float(nsteps - 1)
  c = EvalCubicSpline(t, x, cpts)
  c.nearPlane = -34.461
  c.farPlane = 34.461
  SetView3D(c)
```

3.10 Working with annotations

Adding annotations to your visualization improve the quality of the final visualization in that you can refine the colors that you use, add logos, or highlight features of interest in your plots. This section provides some recipes for creating annotations using scripting.

3.10.1 Using gradient background colors

VisIt's default white background is not necessarily the best looking background color for presentations. Adding a gradient background under your plots is an easy way to add a small professional touch to your visualizations. VisIt provides a few different styles of gradient background: radial, top to bottom, bottom to top, left to right, and right to left. The gradient style is set using the *gradientBackgroundStyle* member of the AnnotationAttributes object. The before and after results are shown in Figure 3.1.

```
# Set a blue/black, radial, gradient background.
a = AnnotationAttributes()
a.backgroundMode = a.Gradient
a.gradientBackgroundStyle = a.Radial
a.gradientColor1 = (0,0,255,255) # Blue
a.gradientColor2 = (0,0,0,255) # Black
SetAnnotationAttributes(a)
```

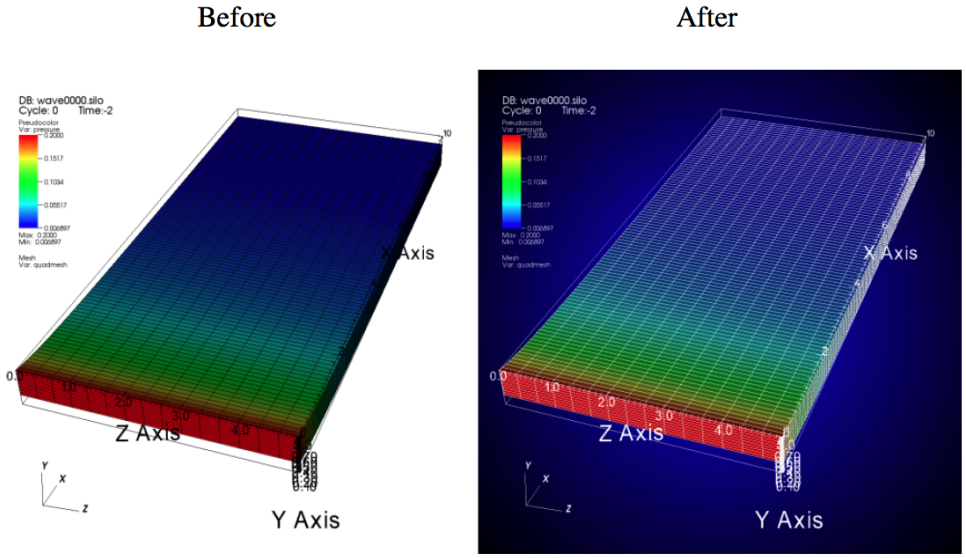


Figure 3.1: Before and after image of adding a gradient background.

3.10.2 Adding a banner

Banners are useful for providing titles for a visualization or for marking its content (see Figure 3.2). To add an "Unclassified" banner to a visualization, use the following bit of Python code:

```
# Create a text object that we'll use to indicate that our
# visualization is unclassified.
banner = CreateAnnotationObject("Text2D")
banner.text = "Unclassified"
banner.position = (0.37, 0.95)
banner.fontBold = 1
# print the attributes that you can set in the banner object.
print banner
```

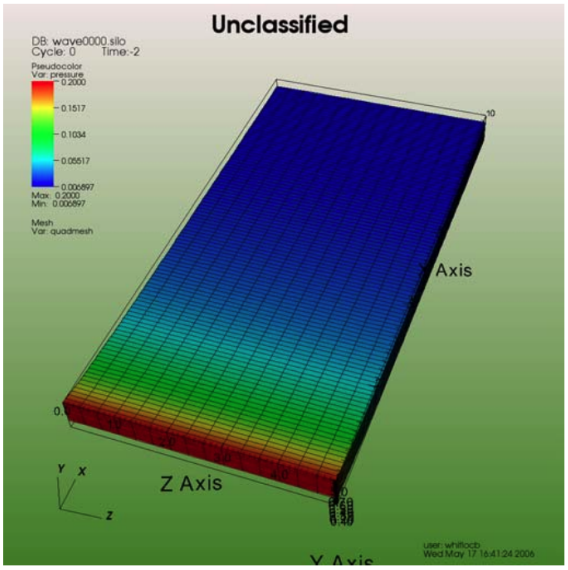


Figure 3.2: Adding a banner

3.10.3 Adding a time slider

Time sliders are important annotations for movies since they convey how much progress an animation has made as well as how many more frames have yet to be seen. The time slider is also important for showing the simulation time as the animation progresses so users can get a sense of when in the simulation important events occur. VisIt's time slider annotation object is shown in Figure 3.3.

```
# Add a time slider in the lower left corner
slider = CreateAnnotationObject("TimeSlider")
slider.height = 0.07
# Print the options that are available in the time slider object
print slider
```

3.10.4 Adding a logo

Adding a logo to a visualization is an important part of project identification for movies and other visualizations created with VisIt. If you have a logo image file stored in TIFF, JPEG, BMP, or PPM format then you can use it with VisIt as an image annotation (see Figure 3.4). Note that this approach can also be used to insert images of graphs, plots, portraits, diagrams, or any other form of image data into a visualization.

```
# Incorporate LLNL logo image (llnl.jpeg) as an annotation
image = CreateAnnotationObject("Image")
image.image = "llnl.jpeg"
image.position = (0.02, 0.02)
# Print the other image annotation options
print image
```

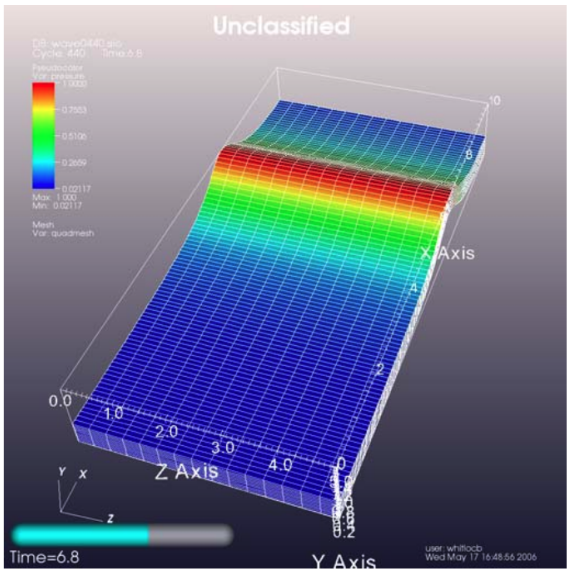


Figure 3.3: Time slider annotation in the lower left corner

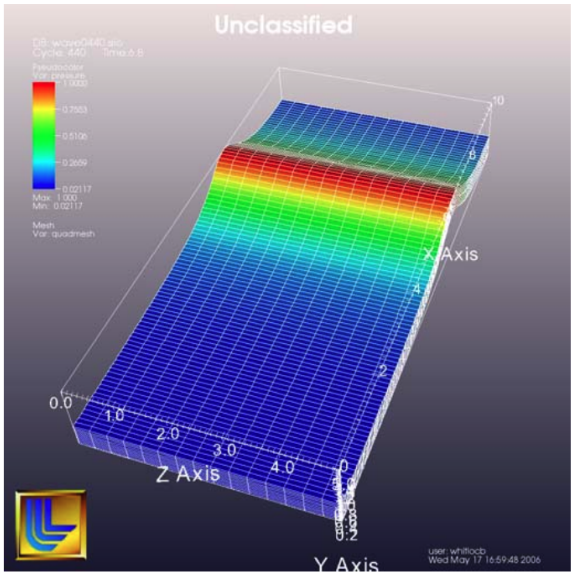


Figure 3.4: Image annotation used to incorporate LLNL logo

Chapter 4

Functions

ActivateDatabase: activates a database that has been previously opened.

Synopsis:
`ActivateDatabase(argument) -> integer`

Arguments:

`argument`	A string object containing the name of the database to be activated.

Returns:
ActivateDatabase returns 1 on success and 0 on failure.

Description:
The ActivateDatabase function is used to set the active database to a database that has been previously opened. The ActivateDatabase function only works when you are using it to activate a database that you have previously opened. You do not need to use this function unless you frequently toggle between more than one database when making plots or changing time states. While the OpenDatabase function can also be used to set the active database, the ActivateDatabase function does not have any side effects that would cause the time state for the new active database to be changed.

Example:
```
% visit -cli
dbs = ("/usr/gapps/visit/data/wave.visit", \
"/usr/gapps/visit/data/curv3d.silo")
OpenDatabase(dbs[0], 17)
AddPlot("Pseudocolor", "u")
DrawPlots()
OpenDatabase(dbs[1])
AddPlot("Pseudocolor", "u")
DrawPlots()
# Let's add another plot from the first database.
ActivateDatabase(dbs[0])
AddPlot("Mesh", "quadmesh")
DrawPlots()
```

AddArgument: add an argument to the viewer's command line argument list.

Synopsis:
`AddArgument(argument)`

Arguments:

`argument`	A string object that is added to the viewer's command line argument list.

Returns:
AddArgument does not return a value.

Description:
The AddArgument function is used to add extra command line arguments to VisIt's viewer. This is only useful when VisIt's Python interface is imported into a stand−alone Python interpreter because the AddArgument function must be called before the viewer is launched. The AddArgument function has no effect when used in VisIt's cli program because the viewer is automatically launched before any commands are processed.

Example:
```
import visit
visit.AddArgument("-nowin") # Add the -nowin argument to the viewer.
```

AddMachineProfile: ¡description¿

Synopsis:
`AddMachineProfile(MachineProfile) -> integer`

Arguments:
`MachineProfile`

Description:
Sets the input machine profile in the HostProfileList, replaces if one already exists Otherwise adds to the list

AddOperator: adds the named operator to the active plots.

Synopsis:
```
AddOperator(operator) -> integer
AddOperator(operator, all) -> integer
```

Arguments:

`operator`	This is a string containing the name of the operator to be applied.
`all`	This is an optional integer argument that applies the operator to allplots if the value of the argument is not zero.

Returns:
The AddOperator function returns an integer value of 1 for success and 0 for failure.

Description:
The AddOperator function adds a VisIt operator to the active plots. The operator argument is a string containing the name of the operator to be added to the active plots. The operatore name must be a valid operator plugin name that is a member of the tuple returned by the OperatorPlugins function. The all argument is an integer that determines whether or not the operator is applied to all plots. If the all argument is not provided, the operator is only added to active plots. Once the AddOperator function is called, the desired operator is added to all active plots unless the all argument is a non−zero value. When the all argument is a non−zero value, the operator is applied to all plots regardless of whether or not they are selected. Operator attributes are set through the SetOperatorOptions function.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
AddPlot("Mesh", "mesh1")
AddOperator("Slice", 1) # Slice both plots
DrawPlots()
```

AddPlot: creates a new plot.

Synopsis:
```
AddPlot(plotType, variableName) -> integer
AddPlot(plotType, variableName, inheritSIL) -> integer
AddPlot(plotType, variableName, inheritSIL,\
     applyOperators) -> integer
```

Arguments:

`plotType`	This is a string containing the name of a valid plot plugin type.
`variableName`	This is a string containing a valid variable name for the open database.
`inheritSIL`	This is an integer flag indicating whether the plot should inherit theactive plot's SIL restriction.
`applyOperators`	This is an integer flag indicating whether the operators from the activeplot should be applied to the new plot.

Returns:
The AddPlot function returns an integer value of 1 for success and 0 for failure.

Description:
The AddPlot function creates a new plot of the specified type using a variable from the open database. The plotType argument is a string that contains the name of a valid plot plugin type which must be a member of the string tuple that is returned by the PlotPlugins function. The variableName argument is a string that contains the name of a variable in the open database. After the AddPlot function is called, a new plot is created and it is made the sole active plot.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Subset", "mat1") # Create a subset plot
DrawPlots()
```

AddWindow: creates a new visualization window.

Synopsis:
`AddWindow()`

Returns:
The AddWindow function does not a return value.

Description:
The AddWindow function creates a new visualization window and makes it the active window. This function can be used to create up to 16 visualization windows. After that, the AddWindow function has no effect.

Example:
```
import visit
visit.Launch()
visit.AddWindow() # Create window #2
visit.AddWindow() # Create window #3
```

AlterDatabaseCorrelation: alters a specific database correlation.

Synopsis:
`AlterDatabaseCorrelation(name, databases, method) -> integer`

Arguments:

`name`	The name argument must be a string object containing the name of thedatabase correlation to be altered.
`databases`	The databases argument must be a tuple or list of strings containing thefully qualified database names to be used in the database correlation.
`method`	The method argument must be an integer in the range [0,3].

Returns:
The AlterDatabaseCorrelation function returns 1 on success and 0 on failure.

Description:
The AlterDatabaseCorrelation method alters an existing database correlation. A database correlation is a VisIt construct that relates the time states for two or more databases in some way. You would use the AlterDatabaseCorrelation function if you wanted to change the list of databases used in a database correlation or if you wanted to change how the databases are related − the correlation method. The name argument is a string that is the name of the database correlation to be altered. If the name that you pass is not a valid database correlation then the AlterDatabaseCorrelation function fails. The databases argument is a list or tuple of string objects containing the fully−qualified (host:/path/filename) names of the databases to be involved in the database query. The method argument allows you to specify a database correlation method.

Correlation method	Value
IndexForIndexCorrelation	0
StretchedIndexCorrelation	1
TimeCorrelation	2
CycleCorrelation	3

Example:
```
dbs = ("/usr/gapps/visit/data/wave.visit", \
"/usr/gapps/visit/data/wave*.silo database")
OpenDatabase(dbs[0])
AddPlot("Pseudocolor", "pressure")
OpenDatabase(dbs[1])
AddPlot("Pseudocolor", "d")
# VisIt created an index for index database correlation but we
# want a cycle correlation.
AlterDatabaseCorrelation("Correlation01", dbs, 3)
```

ApplyNamedSelection: applies a named selection to the active plot.

Synopsis:
`ApplyNamedSelection(name) -> integer`

Arguments:
`name` The name of a named selection. (This should have been previously createdwith a CreateNamedSelection call.)

Returns:
The ApplyNamedSelection function returns 1 for success and 0 for failure.

Description:
Named Selections allow you to select a group of elements (or particles). One typically creates a named selection from a group of elements and then later applies the named selection to another plot (thus reducing the set of elements displayed to the ones from when the named selection was created).

Example:
```
% visit -cli
db = "/usr/gapps/visit/data/wave*.silo database"
OpenDatabase(db)
AddPlot("Pseudocolor", "pressure")
AddOperator("Clip")
c = ClipAttributes()
c.plane1Origin = (0,0.6,0)
c.plane1Normal = (0,-1,0)
SetOperatorOption(c)
DrawPlots()
CreateNamedSelection("els_above_at_time_0")
SetTimeSliderState(40)
RemoveLastOperator()
ApplyNamedSelection("els_above_at_time_0")
```

ChangeActivePlotsVar: changes the variable for the active plots

Synopsis:
`ChangeActivePlotsVar(variableName) -> integer`

Arguments:
`variableName` The name of the new plot variable.

Returns:
The ChangeActivePlotsVar function returns an integer value of 1 for success and 0 for failure.

Description:
The ChangeActivePlotsVar function changes the plotted variable for the active plots. This is a useful way to change what is being visualized without having to delete and recreate the current plots. The variableName argument is a string that contains the name of a variable in the open database.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
SaveWindow()
ChangeActivePlotsVar("v")
```

CheckForNewStates: checks the specified database for new time states.

Synopsis:
`CheckForNewStates(name) -> integer`

Arguments:
`name` The name argument must be a string that contains the name of a databasethat has been opened previously.

Returns:
The CheckForNewStates function returns 1 for success and 0 for failure.

Description:
Calculations are often run at the same time as some of the preliminary visualization work is being performed. That said, you might be visualizing the leading time states of a database that is still being created. If you want to force VisIt to add any new time states that were added since you opened the database, you can use the CheckForNewStates function. The name argument must contain the name of a database that has been opened before.

Example:
```
% visit -cli
db = "/usr/gapps/visit/data/wave*.silo database"
OpenDatabase(db)
AddPlot("Pseudocolor", "pressure")
DrawPlots()
SetTimeSliderState(TimeSliderGetNStates() - 1)
# More files appear on disk
CheckForNewStates(db)
SetTimeSliderState(TimeSliderGetNStates() - 1)
```

ChooseCenterOfRotation: allows you to interactively pick a new center of rotation.

Synopsis:
```
ChooseCenterOfRotation() -> integer
ChooseCenterOfRotation(screenX, screenY) -> integer
```

Arguments:

`screenX`	The X coordinate of the pick point in normalized [0,1] screen space.
`screenY`	The Y cooridinate of the pick point in normalized [0,1] screen space.

Returns:
The ChooseCenterOfRotation function returns 1 if successful and 0 if it fails.

Description:
The ChooseCenterOfRotation function allows you to pick a new center of rotation, which is the point about which plots are rotated when you interactively rotate plots. The function can either take zero arguments, in which case you must interactively pick on plots, or it can take two arguments that correspond to the X and Y coordinates of the desired pick point in normalized screen space. When using the two argument version of the ChooseCenterOfRotation function, the X and Y values are floating point values in the range [0,1]. If the ChooseCenterOfRotation function is able to actually pick on plots, yes there must be plots in the vis window, then the center of rotation is updated and the new value is printed to the console.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlots("Pseudocolor", "u")
DrawPlots()
# Interactively choose the center of rotation
ChooseCenterOfRotation()
# Choose a center of rotation using normalized screen
# coordinates and print the value.
ResetView()
ChooseCenterOfRotation(0.5, 0.3)
print "The new center of rotation is:", GetView3D().centerOfRotation
```

ClearAllWindows: clears visualization windows of plots.

Synopsis:
```
ClearAllWindows() -> integer
ClearWindow() -> integer
```

Returns:
1 on success, 0 on failure.

Description:
The ClearWindow function is used to clear out the plots from the active visualization window. The plots are removed from the visualization window but are left in the plot list so that subsequent calls to the DrawPlots function regenerate the plots in the plot list. The ClearAllWindows function preforms the same work as the ClearWindow function except that all windows are cleared of their plots.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
AddWindow()
SetActiveWindow(2) # Make window 2 active
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Subset", "mat1")
DrawPlots()
ClearWindow() # Clear the plots in window 2.
DrawPlots() # Redraw the plots in window 2.
ClearAllWindows() # Clear the plots from all windows.
```

ClearCache: clears the compute engine's network cache.

Synopsis:
```
ClearCache(host) -> integer
ClearCache(host, simulation) -> integer
ClearCacheForAllEngines() -> integer
```

Arguments:

`host`	The name of the computer where the compute engine is running.
`simulation`	The name of the simulation being processed by the compute engine.

Returns:
The ClearCache and ClearCacheForAllEngines functions return 1 on success and 0 on failure.

Description:
Sometimes during extended VisIt runs, you might want to periodically clear the compute engine's network cache to reduce the amount of memory being used by the compute engine. Clearing the network cache is also useful when you want to change what the compute engine is working on. For example, you might process a large database and then decide to process another large database. Clearing the network cache beforehand will free up more resources for the compute engine so it can more efficiently process the new database. The host argument is a string object containing the name of the computer on which the compute engine is running. The simulation argument is optional and only applies to when you want to instruct a simulation that is acting as a VisIt compute engine to clear its network cache. If you want to tell more than one compute engine to clear its cache without having to call ClearCache multiple times, you can use the ClearCacheForAllEngines function.

Example:
```
%visit -cli
OpenDatabase("localhost:very_large_database")
# Do a lot of work
ClearCache("localhost")
OpenDatabase(localhost:another_large_database")
# Do more work
OpenDatabase("remotehost:yet_another_database")
# Do more work
ClearCacheForAllEngines()
```

ClearCacheForAllEngines: clears the compute engine's network cache.

Synopsis:
```
ClearCache(host) -> integer
ClearCache(host, simulation) -> integer
ClearCacheForAllEngines() -> integer
```

Arguments:

`host`	The name of the computer where the compute engine is running.
`simulation`	The name of the simulation being processed by the compute engine.

Returns:
The ClearCache and ClearCacheForAllEngines functions return 1 on success and 0 on failure.

Description:
Sometimes during extended VisIt runs, you might want to periodically clear the compute engine's network cache to reduce the amount of memory being used by the compute engine. Clearing the network cache is also useful when you want to change what the compute engine is working on. For example, you might process a large database and then decide to process another large database. Clearing the network cache beforehand will free up more resources for the compute engine so it can more efficiently process the new database. The host argument is a string object containing the name of the computer on which the compute engine is running. The simulation argument is optional and only applies to when you want to instruct a simulation that is acting as a VisIt compute engine to clear its network cache. If you want to tell more than one compute engine to clear its cache without having to call ClearCache multiple times, you can use the ClearCacheForAllEngines function.

Example:
```
%visit -cli
OpenDatabase("localhost:very_large_database")
# Do a lot of work
ClearCache("localhost")
OpenDatabase(localhost:another_large_database")
# Do more work
OpenDatabase("remotehost:yet_another_database")
# Do more work
ClearCacheForAllEngines()
```

ClearMacros: clear the macros that have been registered using RegisterMacros.

Synopsis:
`ClearMacros()`

Arguments:
`none`

Returns:
The ClearMacros function does not return a value.

Description:
The ClearMacros function clears out the list of registered macros and sends a message to the gui to clear the buttons from the Macros window.

Example:
`ClearMacros()`

ClearPickPoints: clears pick points from the visualization window

Synopsis:
`ClearPickPoints()`

Returns:
The ClearPickPoints function does not return a value.

Description:
The ClearPickPoints function removes pick points from the active visualization window. Pick points are the letters that are added to the visualization window where the mouse is clicked when the visualization window is in pick mode.

Example:
```
% visit -cli
# Put the visualization window into pick mode using the popup
# menu and add some pick points.
# Clear the pick points.
ClearPickPoints()
```

ClearReferenceLines: clears reference lines from the visualization window.

Synopsis:
`ClearReferenceLines()`

Returns:
The ClearReferenceLines function does not return a value.

Description:
The ClearReferenceLines function removes reference lines from the active visualization window. Reference lines are the lines that are drawn on a plot to show where you have performed lineouts.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/curv2d.silo")
AddPlot("Pseudocolor", "d")
Lineout((-3.0, 2.0), (2.0, 4.0), ("default", "u", "v"))
ClearReferenceLines()
```

ClearViewKeyframes: clears any view keyframes that have been set.

Synopsis:
`ClearViewKeyframes() -> integer`

Returns:
The ClearViewKeyframes function returns 1 on success and 0 on failure.

Description:
The ClearViewKeyframes function clears any view keyframes that may have been set. View keyframes are used to create complex view behavior such as fly−throughs when VisIt is in keyframing mode.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
k = KeyframeAttributes()
k.enabled, k.nFrames, k.nFramesWasUserSet = 1,10,1
SetKeyframeAttributes(k)
DrawPlots()
SetViewKeyframe()
v1 = GetView3D()
v1.viewNormal = (-0.66609, 0.337227, 0.665283)
v1.viewUp = (0.157431, 0.935425, -0.316537)
SetView3D(v1)
SetTimeSliderState(9)
SetViewKeyframe()
ToggleCameraViewMode()
for i in range(10):
SetTimeSliderState(i)
ClearViewKeyframes()
```

ClearWindow: clears visualization windows of plots.

Synopsis:
```
ClearAllWindows() -> integer
ClearWindow() -> integer
```

Returns:
1 on success, 0 on failure.

Description:
The ClearWindow function is used to clear out the plots from the active visualization window. The plots are removed from the visualization window but are left in the plot list so that subsequent calls to the DrawPlots function regenerate the plots in the plot list. The ClearAllWindows function preforms the same work as the ClearWindow function except that all windows are cleared of their plots.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
AddWindow()
SetActiveWindow(2) # Make window 2 active
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Subset", "mat1")
DrawPlots()
ClearWindow() # Clear the plots in window 2.
DrawPlots() # Redraw the plots in window 2.
ClearAllWindows() # Clear the plots from all windows.
```

CloneWindow: creates a new window that has the same plots, annotations, lights, and

Synopsis:
`CloneWindow() -> integer`

Returns:
The CloneWindow function returns an integer value of 1 for success and 0 for failure.

Description:
The CloneWindow function tells the viewer to create a new window, based on the active window, that contains the same plots, annotations, lights, and view as the active window. This function is useful for when you have a window set up like you want and then want to do the same thing in another window using a different database. You can first clone the window and then replace the database.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
v = ViewAttributes()
v.camera = (-0.505893, 0.32034, 0.800909)
v.viewUp = (0.1314, 0.946269, -0.295482)
v.parallelScale = 14.5472
v.nearPlane = -34.641
v.farPlane = 34.641
v.perspective = 1
SetView3D() # Set the view
a = AnnotationAttributes()
a.backgroundColor = (0, 0, 255, 255)
SetAnnotationAttributes(a) # Set the annotation properties
CloneWindow() # Create a clone of the active window
DrawPlots() # Make the new window draw its plots
```

Close: closes the viewer.

Synopsis:
`Close()`

Arguments:
`none`

Returns:
The Close function does not return a value.

Description:
The Close function terminates VisIt's viewer. This is useful for Python scripts that only need access to VisIt's capabilties for a short time before closing VisIt.

Example:
```
import visit
visit.Launch()
visit.Close() # Close the viewer
```

CloseComputeEngine: closes the compute engine running on a specified host.

Synopsis:
```
CloseComputeEngine() -> integer
CloseComputeEngine(hostName) -> integer
CloseComputeEngine(hostName, simulation) -> integer
```

Arguments:

`hostName`	Optional name of the computer on which the compute engine is running.
`simulation`	Optional name of a simulation.

Returns:
The CloseComputeEngine function returns an integer value of 1 for success and 0 for failure.

Description:
The CloseComputeEngine function tells the viewer to close the compute engine running a specified host. The hostName argument is a string that contains the name of the computer where the compute engine is running. The hostName argument can also be the name "localhost" if you want to close the compute engine on the local machine without having to specify its name. It is not necessary to provide the hostName argument. If the argument is omitted, the first compute engine in the engine list will be closed. The simulation argument can be provided if you want to close a connection to a simulation that is acting as a VisIt compute engine. A compute engine can be launched again by creating a plot or by calling the OpenComputeEngine function.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo") # Launches an engine
AddPlot("Pseudocolor", "u")
DrawPlots()
CloseComputeEngine() # Close the compute engine
```

CloseDatabase: closes the specified database and frees up resources associated with it.

Synopsis:
`CloseDatabase(name) -> integer`

Arguments:
`name` A string object containing the name of the database to close.

Returns:
The CloseDatabase function returns 1 on success and 0 on failure.

Description:
The CloseDatabase function is used to close a specified database and free all resources that were devoted to keeping the database open. This function has an effect similar to ClearCache but it does more in that in addition to clearing the compute engine's cache, which it only does for the specified database, it also removes all references to the specified database from tables of cached metadata, etc. Note that the CloseDatabase function will fail and the database will not be closed if any plots reference the specified database.

Example:
```
% visit -cli
db = "/usr/gapps/visit/data/globe.silo"
OpenDatabase(db)
AddPlot("Pseudocolor", "u")
DrawPlots()
print "This won't work: retval = %d" % CloseDatabase(db)
DeleteAllPlots()
print "Now it works: retval = %d" % CloseDatabase(db)
```

ColorTableNames: returns a tuple of color table names.

Synopsis:
`ColorTableNames() -> tuple`

Returns:
The ColorTableNames function returns a tuple of strings containing the names of the color tables that have been defined.

Description:
The ColorTableNames function returns a tuple of strings containing the names of the color tables that have been defined. This method can be used in case you want to iterate over several color tables.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/curv2d.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
for ct in ColorTableNames():
p = PseudocolorAttributes()
p.colorTableName = ct
SetPlotOptions(p)
```

ConstructDataBinning: constructs a data binning

Synopsis:
`ConstructDataBinning(i) -> integer`

Arguments:
`i` An object of type ConstructDataBinningAttributes. This object specifies the optionsfor constructing a data binning.

Returns:
Returns 1 on success, 0 on failure.

Description:
The ConstructDataBinning function creates a data binning function for the active plot. Data Binnings place data from a data set into bins and reduce that data. They are used to either be incorporated with expressions to make new derived quantities or to be directly visualized.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/curv3d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
# Set the construct data binning attributes.
i = ConstructDataBinningAttributes()
i.name = "db1"
i.binningScheme = i.Uniform
i.varnames = ("u", "w")
i.binBoundaries = (-1, 1, -1, 1) # minu, maxu, minw, maxw
i.numSamples = (25, 25)
i.reductionOperator = i.Average
i.varForReductionOperator = "v"
ConstructDataBinning(i)
# Example of binning using spatial coordinates
i.varnames = ("X", "u") # X is added as a placeholder to maintain indexing
i.binType = (1, 0) # 1 = X, 2 = Y, 3 = Z, 0 = variable
```

CopyAnnotationsToWindow: copies attributes from one visualization window to another visualization

Synopsis:
```
CopyAnnotationsToWindow(source, dest) -> integer
CopyLightingToWindow(source, dest) -> integer
CopyViewTowindow(source, dest) -> integer
CopyPlotsToWindow(source, dest) -> integer
```

Arguments:

`source`	The index (an integer from 1 to 16) of the source window.
`dest`	The index (an integer from 1 to 16) of the destination window.

Returns:
The Copy functions return an integer value of 1 for success and 0 for failure.

Description:
The Copy functions copy attributes from one visualization window to another visualization window. The CopyAnnotationsToWindow function copies the annotations from a source visualization window to a destination visualization window while the CopyLightingAttributes function copies lighting and the CopyViewToWindow function copies the view. The CopyPlotsToWindow function copies the plots from one visualization window to another visualization window but does not also force plots to generate so after copying plots with the CopyPlotsToWindow function, you should also call the DrawPlots function.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
AddWindow()
SetActiveWindow(2)
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Mesh", "mesh1")
# Copy window 1's Pseudocolor plot to window 2.
CopyPlotsToWindow(1, 2)
DrawPlots() # Window 2 will have 2 plots
# Spin the plots around in window 2 using the mouse.
CopyViewToWindow(2, 1) # Copy window 2's view to window 1.
```

CopyLightingToWindow: copies attributes from one visualization window to another visualization

Synopsis:
```
CopyAnnotationsToWindow(source, dest) -> integer
CopyLightingToWindow(source, dest) -> integer
CopyViewTowindow(source, dest) -> integer
CopyPlotsToWindow(source, dest) -> integer
```

Arguments:

`source`	The index (an integer from 1 to 16) of the source window.
`dest`	The index (an integer from 1 to 16) of the destination window.

Returns:
The Copy functions return an integer value of 1 for success and 0 for failure.

Description:
The Copy functions copy attributes from one visualization window to another visualization window. The CopyAnnotationsToWindow function copies the annotations from a source visualization window to a destination visualization window while the CopyLightingAttributes function copies lighting and the CopyViewToWindow function copies the view. The CopyPlotsToWindow function copies the plots from one visualization window to another visualization window but does not also force plots to generate so after copying plots with the CopyPlotsToWindow function, you should also call the DrawPlots function.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
AddWindow()
SetActiveWindow(2)
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Mesh", "mesh1")
# Copy window 1's Pseudocolor plot to window 2.
CopyPlotsToWindow(1, 2)
DrawPlots() # Window 2 will have 2 plots
# Spin the plots around in window 2 using the mouse.
CopyViewToWindow(2, 1) # Copy window 2's view to window 1.
```

CopyPlotsToWindow: copies attributes from one visualization window to another visualization

Synopsis:
```
CopyAnnotationsToWindow(source, dest) -> integer
CopyLightingToWindow(source, dest) -> integer
CopyViewTowindow(source, dest) -> integer
CopyPlotsToWindow(source, dest) -> integer
```

Arguments:

`source`	The index (an integer from 1 to 16) of the source window.
`dest`	The index (an integer from 1 to 16) of the destination window.

Returns:
The Copy functions return an integer value of 1 for success and 0 for failure.

Description:
The Copy functions copy attributes from one visualization window to another visualization window. The CopyAnnotationsToWindow function copies the annotations from a source visualization window to a destination visualization window while the CopyLightingAttributes function copies lighting and the CopyViewToWindow function copies the view. The CopyPlotsToWindow function copies the plots from one visualization window to another visualization window but does not also force plots to generate so after copying plots with the CopyPlotsToWindow function, you should also call the DrawPlots function.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
AddWindow()
SetActiveWindow(2)
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Mesh", "mesh1")
# Copy window 1's Pseudocolor plot to window 2.
CopyPlotsToWindow(1, 2)
DrawPlots() # Window 2 will have 2 plots
# Spin the plots around in window 2 using the mouse.
CopyViewToWindow(2, 1) # Copy window 2's view to window 1.
```

CopyViewToWindow: copies attributes from one visualization window to another visualization

Synopsis:
```
CopyAnnotationsToWindow(source, dest) -> integer
CopyLightingToWindow(source, dest) -> integer
CopyViewTowindow(source, dest) -> integer
CopyPlotsToWindow(source, dest) -> integer
```

Arguments:

`source`	The index (an integer from 1 to 16) of the source window.
`dest`	The index (an integer from 1 to 16) of the destination window.

Returns:
The Copy functions return an integer value of 1 for success and 0 for failure.

Description:
The Copy functions copy attributes from one visualization window to another visualization window. The CopyAnnotationsToWindow function copies the annotations from a source visualization window to a destination visualization window while the CopyLightingAttributes function copies lighting and the CopyViewToWindow function copies the view. The CopyPlotsToWindow function copies the plots from one visualization window to another visualization window but does not also force plots to generate so after copying plots with the CopyPlotsToWindow function, you should also call the DrawPlots function.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
AddWindow()
SetActiveWindow(2)
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Mesh", "mesh1")
# Copy window 1's Pseudocolor plot to window 2.
CopyPlotsToWindow(1, 2)
DrawPlots() # Window 2 will have 2 plots
# Spin the plots around in window 2 using the mouse.
CopyViewToWindow(2, 1) # Copy window 2's view to window 1.
```

CreateAnnotationObject: creates an annotation object that can directly manipulate annotations in

Synopsis:
`CreateAnnotationObject(annotType) -> annotation object`

Arguments:

`annotType`	A string containing the name of the type of annotation object to create.

Returns:
CreateAnnotationObject is a factory function that creates annotation objects of different types. The return value, if a valid annotation type is provided, is an annotation object. If the function fails, VisItException is raised.

Description:
CreateAnnotationObject is a factory function that creates different kinds of annotation objects. The annotType argument is a string containing the name of the type of annotation object to create. Each type of annotation object has different properties that can be set. Setting the different properties of an Annotation objects directly modifes annotations in the vis window. Currently there are 5 types of annotation objects:

Annotation type	String
2D text annotation	"Text2D"
3D text annotation	"Text3D"
Time slider annotation	"TimeSlider"
Image annotation	"Image"
Line/arrow annotation	"Line2D"

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/wave.visit", 17)
AddPlot("Pseudocolor", "pressure")
DrawPlots()
slider = CreateAnnotationObject("TimeSlider")
print slider
slider.startColor = (255,0,0,255)
slider.endColor = (255,255,0,255)
```

CreateDatabaseCorrelation: creates a database correlation.

Synopsis:
`CreateDatabaseCorrelation(name, databases, method) -> integer`

Arguments:

`name`	String object containing the name of the database correlation to becreated.
`databases`	Tuple or list of string objects containing the names of the databases toinvolve in the database correlation.
`method`	An integer in the range [0,3] that determines the correlation method.

Returns:
The CreateDatabaseCorrelation function returns 1 on success and 0 on failure.

Description:
The CreateDatabaseCorrelation function creates a database correlation, which is a VisIt construct that relates the time states for two or more databases in some way. You would use the CreateDatabaseCorrelation function if you wanted to put plots from more than one time−varying database in the same vis window and then move them both through time in some synchronized way. The name argument is a string that is the name of the database correlation to be created. You will use the name of the database correlation to set the active time slider later so that you can change time states. The databases argument is a list or tuple of string objects containing the fully−qualified (host:/path/filename) names of the databases to be involved in the database query. The method argument allows you to specify a database correlation method.

Correlation method	Value
IndexForIndexCorrelation	0
StretchedIndexCorrelation	1
TimeCorrelation	2
CycleCorrelation	3

Each database correlation has its own time slider that can be used to set the time state of databases that are part of a database correlation. Individual time−varying databases have their own trivial database correlation, consisting of only 1 database. When you call the CreateDatabaseCorrelation function, VisIt creates a new time slider with the same name as the database correlation and makes it be the active time slider.

Example:
```
% visit -cli
dbs = ("/usr/gapps/visit/data/dbA00.pdb",
"/usr/gapps/visit/data/dbB00.pdb")
OpenDatabase(dbs[0])
AddPlot("FilledBoundary", "material(mesh)")
OpenDatabase(dbs[1])
AddPlot("FilledBoundary", "material(mesh)")
DrawPlots()
CreateDatabaseCorrelation("common", dbs, 1)
# Creating a new database correlation also creates a new time
# slider and makes it be active.
w = GetWindowInformation()
print "Active time slider: %s" % w.timeSliders[w.activeTimeSlider]
```

```
# Animate through time using the "common" database correlation's
# time slider.
for i in range(TimeSliderGetNStates()):
SetTimeSliderState(i)
```

CreateNamedSelection: creates a named selection.

Synopsis:
```
CreateNamedSelection(name) -> integer
CreateNamedSelection(name, properties) -> integer
```

Arguments:

`name`	The name of a named selection.
`properties`	This optional argument lets you pass a SelectionProperties object containingthe properties that will be used to create the named selection. When thisargument is omitted, the named selection will always be associated withthe active plot. You can use this argument to set up more complex namedselections that may be associated with plots or databases.

Returns:
The CreateNamedSelection function returns 1 for success and 0 for failure.

Description:
Named Selections allow you to select a group of elements (or particles). One typically creates a named selection from a group of elements and then later applies the named selection to another plot (thus reducing the set of elements displayed to the ones from when the named selection was created).

Example:
```
% visit -cli
db = "/usr/gapps/visit/data/wave*.silo database"
OpenDatabase(db)
AddPlot("Pseudocolor", "pressure")
AddOperator("Clip")
c = ClipAttributes()
c.plane1Origin = (0,0.6,0)
c.plane1Normal = (0,-1,0)
SetOperatorOption(c)
DrawPlots()
CreateNamedSelection("els_above_at_time_0")
SetTimeSliderState(40)
RemoveLastOperator()
ApplyNamedSelection("els_above_at_time_0")
```

DeIconifyAllWindows: unhides all of the hidden visualization windows.

Synopsis:
`DeIconifyAllWindows()`

Returns:
The DeIconifyAllWindows function does not return a value.

Description:
The DeIconifyAllWindows function unhides all of the hidden visualization windows. This function is usually called after IconifyAllWindows as a way of making all of the hidden visualization windows visible.

Example:
```
% visit -cli
SetWindowLayout(4) # Have 4 windows
IconifyAllWindows()
DeIconifyAllWindows()
```

DefineArrayExpression: creates a expression variable.

Synopsis:
```
DefineMaterialExpression(variableName, expression) -> integer
DefineMeshExpression(variableName, expression) -> integer
DefineScalarExpression(variableName, expression) -> integer
DefineSpeciesExpression(variableName, expression) -> integer
DefineTensorExpression(variableName, expression) -> integer
DefineVectorExpression(variableName, expression) -> integer
DefineArrayExpression(variableName, expression) -> integer
DefineCurveExpression(variableName, expression) -> integer
```

Arguments:

`variableName`	The name of the variable to be created.
`expression`	The expression definition.

Returns:
The DefineExpression functions return 1 on success and 0 on failure.

Description:
The DefineScalarExpression function creates a new scalar variable based on other variables from the open database. Likewise, the DefineMaterialExpression function creates new material variables, DefineMeshExpression creates new mesh variables, DefineSpeciesExpression creates new species variables, DefineVectorExpression creates new vector variables, DefineTensorExpression creates new tensor variables, and DefineArrayExpression creates new array variables. Expression variables can be plotted like any other variable. The variableName argument is a string that contains the name of the new variable. You can pass the name of an existing expression if you want to provide a new expression definition. The expression argument is a string that contains the definition of the new variable in terms of math operators and pre−existing variable names Reference the VisIt User's Manual if you want more information on creating expressions, such as expression syntax, or a list of built−in expression functions.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
DefineScalarExpression("myvar", "sin(u) + cos(w)")
# Plot the scalar expression variable.
AddPlot("Pseudocolor", "myvar")
DrawPlots()
# Plot a vector expression variable.
DefineVectorExpression("myvec", "{u,v,w}")
AddPlot("Vector", "myvec")
DrawPlots()
```

DefineCurveExpression: creates a expression variable.

Synopsis:

```
DefineMaterialExpression(variableName, expression) -> integer
DefineMeshExpression(variableName, expression) -> integer
DefineScalarExpression(variableName, expression) -> integer
DefineSpeciesExpression(variableName, expression) -> integer
DefineTensorExpression(variableName, expression) -> integer
DefineVectorExpression(variableName, expression) -> integer
DefineArrayExpression(variableName, expression) -> integer
DefineCurveExpression(variableName, expression) -> integer
```

Arguments:

`variableName`	The name of the variable to be created.
`expression`	The expression definition.

Returns:
The DefineExpression functions return 1 on success and 0 on failure.

Description:
The DefineScalarExpression function creates a new scalar variable based on other variables from the open database. Likewise, the DefineMaterialExpression function creates new material variables, DefineMeshExpression creates new mesh variables, DefineSpeciesExpression creates new species variables, DefineVectorExpression creates new vector variables, DefineTensorExpression creates new tensor variables, and DefineArrayExpression creates new array variables. Expression variables can be plotted like any other variable. The variableName argument is a string that contains the name of the new variable. You can pass the name of an existing expression if you want to provide a new expression definition. The expression argument is a string that contains the definition of the new variable in terms of math operators and pre−existing variable names Reference the VisIt User's Manual if you want more information on creating expressions, such as expression syntax, or a list of built−in expression functions.

Example:

```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
DefineScalarExpression("myvar", "sin(u) + cos(w)")
# Plot the scalar expression variable.
AddPlot("Pseudocolor", "myvar")
DrawPlots()
# Plot a vector expression variable.
DefineVectorExpression("myvec", "{u,v,w}")
AddPlot("Vector", "myvec")
DrawPlots()
```

DefineMaterialExpression: creates a expression variable.

Synopsis:
```
DefineMaterialExpression(variableName, expression) -> integer
DefineMeshExpression(variableName, expression) -> integer
DefineScalarExpression(variableName, expression) -> integer
DefineSpeciesExpression(variableName, expression) -> integer
DefineTensorExpression(variableName, expression) -> integer
DefineVectorExpression(variableName, expression) -> integer
DefineArrayExpression(variableName, expression) -> integer
DefineCurveExpression(variableName, expression) -> integer
```

Arguments:

`variableName`	The name of the variable to be created.
`expression`	The expression definition.

Returns:
The DefineExpression functions return 1 on success and 0 on failure.

Description:
The DefineScalarExpression function creates a new scalar variable based on other variables from the open database. Likewise, the DefineMaterialExpression function creates new material variables, DefineMeshExpression creates new mesh variables, DefineSpeciesExpression creates new species variables, DefineVectorExpression creates new vector variables, DefineTensorExpression creates new tensor variables, and DefineArrayExpression creates new array variables. Expression variables can be plotted like any other variable. The variableName argument is a string that contains the name of the new variable. You can pass the name of an existing expression if you want to provide a new expression definition. The expression argument is a string that contains the definition of the new variable in terms of math operators and pre−existing variable names Reference the VisIt User's Manual if you want more information on creating expressions, such as expression syntax, or a list of built−in expression functions.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
DefineScalarExpression("myvar", "sin(u) + cos(w)")
# Plot the scalar expression variable.
AddPlot("Pseudocolor", "myvar")
DrawPlots()
# Plot a vector expression variable.
DefineVectorExpression("myvec", "{u,v,w}")
AddPlot("Vector", "myvec")
DrawPlots()
```

DefineMeshExpression: creates a expression variable.

Synopsis:
```
DefineMaterialExpression(variableName, expression) -> integer
DefineMeshExpression(variableName, expression) -> integer
DefineScalarExpression(variableName, expression) -> integer
DefineSpeciesExpression(variableName, expression) -> integer
DefineTensorExpression(variableName, expression) -> integer
DefineVectorExpression(variableName, expression) -> integer
DefineArrayExpression(variableName, expression) -> integer
DefineCurveExpression(variableName, expression) -> integer
```

Arguments:

`variableName`	The name of the variable to be created.
`expression`	The expression definition.

Returns:
The DefineExpression functions return 1 on success and 0 on failure.

Description:
The DefineScalarExpression function creates a new scalar variable based on other variables from the open database. Likewise, the DefineMaterialExpression function creates new material variables, DefineMeshExpression creates new mesh variables, DefineSpeciesExpression creates new species variables, DefineVectorExpression creates new vector variables, DefineTensorExpression creates new tensor variables, and DefineArrayExpression creates new array variables. Expression variables can be plotted like any other variable. The variableName argument is a string that contains the name of the new variable. You can pass the name of an existing expression if you want to provide a new expression definition. The expression argument is a string that contains the definition of the new variable in terms of math operators and pre−existing variable names Reference the VisIt User's Manual if you want more information on creating expressions, such as expression syntax, or a list of built−in expression functions.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
DefineScalarExpression("myvar", "sin(u) + cos(w)")
# Plot the scalar expression variable.
AddPlot("Pseudocolor", "myvar")
DrawPlots()
# Plot a vector expression variable.
DefineVectorExpression("myvec", "{u,v,w}")
AddPlot("Vector", "myvec")
DrawPlots()
```

DefinePythonExpression: defines a new Python Filter Expression.

Synopsis:

```
DefinePythonExpression("myvar",[args],\
    source='python filter source ...')
DefinePythonExpression("myvar",[args],\
        source='python filter source ...',type='scalar')
DefinePythonExpression("myvar",[args],\
    file='path/to/python_filter_script.py')
```

Arguments:

`name`	The name of the variable to be created.
`args`	A tuple (or list) of strings providing the variable names of thearguments to the Python Expression.
`source`	A string containing the source code for a Python Expression Filter .
`file`	A string containing the path to a Python Expression Filter script file.
`type`	An optional string defining the output type of the expression.Default type: 'scalar'Avalaible types: 'scalar','vector','tensor','array','curve'Note: Use only one of the 'source' or 'file' arguments.If both are used the 'source' argument overrides 'file'.

Returns:
The DefineExpression functions do not return a value.

Description:
Used to define a Python Filter Expression.

DefineScalarExpression: creates a expression variable.

Synopsis:
```
DefineMaterialExpression(variableName, expression) -> integer
DefineMeshExpression(variableName, expression) -> integer
DefineScalarExpression(variableName, expression) -> integer
DefineSpeciesExpression(variableName, expression) -> integer
DefineTensorExpression(variableName, expression) -> integer
DefineVectorExpression(variableName, expression) -> integer
DefineArrayExpression(variableName, expression) -> integer
DefineCurveExpression(variableName, expression) -> integer
```

Arguments:

`variableName`	The name of the variable to be created.
`expression`	The expression definition.

Returns:
The DefineExpression functions return 1 on success and 0 on failure.

Description:
The DefineScalarExpression function creates a new scalar variable based on other variables from the open database. Likewise, the DefineMaterialExpression function creates new material variables, DefineMeshExpression creates new mesh variables, DefineSpeciesExpression creates new species variables, DefineVectorExpression creates new vector variables, DefineTensorExpression creates new tensor variables, and DefineArrayExpression creates new array variables. Expression variables can be plotted like any other variable. The variableName argument is a string that contains the name of the new variable. You can pass the name of an existing expression if you want to provide a new expression definition. The expression argument is a string that contains the definition of the new variable in terms of math operators and pre−existing variable names Reference the VisIt User's Manual if you want more information on creating expressions, such as expression syntax, or a list of built−in expression functions.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
DefineScalarExpression("myvar", "sin(u) + cos(w)")
# Plot the scalar expression variable.
AddPlot("Pseudocolor", "myvar")
DrawPlots()
# Plot a vector expression variable.
DefineVectorExpression("myvec", "{u,v,w}")
AddPlot("Vector", "myvec")
DrawPlots()
```

DefineSpeciesExpression: creates a expression variable.

Synopsis:
```
DefineMaterialExpression(variableName, expression) -> integer
DefineMeshExpression(variableName, expression) -> integer
DefineScalarExpression(variableName, expression) -> integer
DefineSpeciesExpression(variableName, expression) -> integer
DefineTensorExpression(variableName, expression) -> integer
DefineVectorExpression(variableName, expression) -> integer
DefineArrayExpression(variableName, expression) -> integer
DefineCurveExpression(variableName, expression) -> integer
```

Arguments:

`variableName`	The name of the variable to be created.
`expression`	The expression definition.

Returns:
The DefineExpression functions return 1 on success and 0 on failure.

Description:
The DefineScalarExpression function creates a new scalar variable based on other variables from the open database. Likewise, the DefineMaterialExpression function creates new material variables, DefineMeshExpression creates new mesh variables, DefineSpeciesExpression creates new species variables, DefineVectorExpression creates new vector variables, DefineTensorExpression creates new tensor variables, and DefineArrayExpression creates new array variables. Expression variables can be plotted like any other variable. The variableName argument is a string that contains the name of the new variable. You can pass the name of an existing expression if you want to provide a new expression definition. The expression argument is a string that contains the definition of the new variable in terms of math operators and pre−existing variable names Reference the VisIt User's Manual if you want more information on creating expressions, such as expression syntax, or a list of built−in expression functions.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
DefineScalarExpression("myvar", "sin(u) + cos(w)")
# Plot the scalar expression variable.
AddPlot("Pseudocolor", "myvar")
DrawPlots()
# Plot a vector expression variable.
DefineVectorExpression("myvec", "{u,v,w}")
AddPlot("Vector", "myvec")
DrawPlots()
```

DefineTensorExpression: creates a expression variable.

Synopsis:
```
DefineMaterialExpression(variableName, expression) -> integer
DefineMeshExpression(variableName, expression) -> integer
DefineScalarExpression(variableName, expression) -> integer
DefineSpeciesExpression(variableName, expression) -> integer
DefineTensorExpression(variableName, expression) -> integer
DefineVectorExpression(variableName, expression) -> integer
DefineArrayExpression(variableName, expression) -> integer
DefineCurveExpression(variableName, expression) -> integer
```

Arguments:

`variableName`	The name of the variable to be created.
`expression`	The expression definition.

Returns:
The DefineExpression functions return 1 on success and 0 on failure.

Description:
The DefineScalarExpression function creates a new scalar variable based on other variables from the open database. Likewise, the DefineMaterialExpression function creates new material variables, DefineMeshExpression creates new mesh variables, DefineSpeciesExpression creates new species variables, DefineVectorExpression creates new vector variables, DefineTensorExpression creates new tensor variables, and DefineArrayExpression creates new array variables. Expression variables can be plotted like any other variable. The variableName argument is a string that contains the name of the new variable. You can pass the name of an existing expression if you want to provide a new expression definition. The expression argument is a string that contains the definition of the new variable in terms of math operators and pre−existing variable names Reference the VisIt User's Manual if you want more information on creating expressions, such as expression syntax, or a list of built−in expression functions.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
DefineScalarExpression("myvar", "sin(u) + cos(w)")
# Plot the scalar expression variable.
AddPlot("Pseudocolor", "myvar")
DrawPlots()
# Plot a vector expression variable.
DefineVectorExpression("myvec", "{u,v,w}")
AddPlot("Vector", "myvec")
DrawPlots()
```

DefineVectorExpression: creates a expression variable.

Synopsis:
```
DefineMaterialExpression(variableName, expression) -> integer
DefineMeshExpression(variableName, expression) -> integer
DefineScalarExpression(variableName, expression) -> integer
DefineSpeciesExpression(variableName, expression) -> integer
DefineTensorExpression(variableName, expression) -> integer
DefineVectorExpression(variableName, expression) -> integer
DefineArrayExpression(variableName, expression) -> integer
DefineCurveExpression(variableName, expression) -> integer
```

Arguments:

`variableName`	The name of the variable to be created.
`expression`	The expression definition.

Returns:
The DefineExpression functions return 1 on success and 0 on failure.

Description:
The DefineScalarExpression function creates a new scalar variable based on other variables from the open database. Likewise, the DefineMaterialExpression function creates new material variables, DefineMeshExpression creates new mesh variables, DefineSpeciesExpression creates new species variables, DefineVectorExpression creates new vector variables, DefineTensorExpression creates new tensor variables, and DefineArrayExpression creates new array variables. Expression variables can be plotted like any other variable. The variableName argument is a string that contains the name of the new variable. You can pass the name of an existing expression if you want to provide a new expression definition. The expression argument is a string that contains the definition of the new variable in terms of math operators and pre−existing variable names Reference the VisIt User's Manual if you want more information on creating expressions, such as expression syntax, or a list of built−in expression functions.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
DefineScalarExpression("myvar", "sin(u) + cos(w)")
# Plot the scalar expression variable.
AddPlot("Pseudocolor", "myvar")
DrawPlots()
# Plot a vector expression variable.
DefineVectorExpression("myvec", "{u,v,w}")
AddPlot("Vector", "myvec")
DrawPlots()
```

DeleteActivePlots: deletes plots from the active window's plot list.

Synopsis:
```
DeleteActivePlots() -> integer
DeleteAllPlots() -> integer
```

Returns:
The Delete functions return an integer value of 1 for success and 0 for failure.

Description:
The Delete functions delete plots from the active window's plot list. The DeleteActivePlots function deletes all of the active plots from the plot list. There is no way to retrieve a plot once it has been deleted from the plot list. The active plots are set using the SetActivePlots function. The DeleteAllPlots function deletes all plots from the active window's plot list regardless of whether or not they are active.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/curv2d.silo")
AddPlot("Pseudocolor", "d")
AddPlot("Contour", "u")
AddPlot("Mesh", "curvmesh2d")
DrawPlots()
DeleteActivePlots() # Delete the mesh plot
DeleteAllPlots() # Delete the pseudocolor and contour plots.
```

DeleteAllPlots: deletes plots from the active window's plot list.

Synopsis:
```
DeleteActivePlots() -> integer
DeleteAllPlots() -> integer
```

Returns:
The Delete functions return an integer value of 1 for success and 0 for failure.

Description:
The Delete functions delete plots from the active window's plot list. The DeleteActivePlots function deletes all of the active plots from the plot list. There is no way to retrieve a plot once it has been deleted from the plot list. The active plots are set using the SetActivePlots function. The DeleteAllPlots function deletes all plots from the active window's plot list regardless of whether or not they are active.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/curv2d.silo")
AddPlot("Pseudocolor", "d")
AddPlot("Contour", "u")
AddPlot("Mesh", "curvmesh2d")
DrawPlots()
DeleteActivePlots() # Delete the mesh plot
DeleteAllPlots() # Delete the pseudocolor and contour plots.
```

DeleteDatabaseCorrelation: deletes a database correlation.

Synopsis:
`DeleteDatabaseCorrelation(name) -> integer`

Arguments:

`name`	A string object containing the name of the database correlation to delete.

Returns:
The DeleteDatabaseCorrelation function returns 1 on success and 0 on failure.

Description:
The DeleteDatabaseCorrelation function deletes a specific database correlation and its associated time slider. If you delete a database correlation whose time slider is being used for the current time slider, the time slider will be reset to the time slider of the next best suited database correlation. You can use the DeleteDatabaseCorrelation function to remove database correlations that you no longer need such as when you choose to examine databases that have nothing to do with your current databases.

Example:
```
% visit -cli
dbs = ("dbA00.pdb", "dbB00.pdb")
OpenDatabase(dbs[0])
AddPlot("FilledBoundary", "material(mesh)")
OpenDatabase(dbs[1])
AddPlot("FilledBoundary", "material(mesh)")
DrawPlots()
CreateDatabaseCorrelation("common", dbs, 1)
SetTimeSliderState(10)
DeleteAllPlots()
DeleteDatabaseCorrelation("common")
CloseDatabase(dbs[0])
CloseDatabase(dbs[1])
```

DeleteExpression: deletes an expression variable from the expression list.

Synopsis:
`DeleteExpression(variableName) -> integer`

Arguments:
`variableName` The name of the expression variable to be deleted.

Returns:
The DeleteExpression function returns 1 on success and 0 on failure.

Description:
The DeleteExpression function deletes the definition of an expression. The variableName argument is a string containing the name of the variable expression to be deleted. Any plot that uses an expression that has been deleted will fail to regenerate if its attributes are changed.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
DefineScalarExpression("myvar", "sin(u) + cos(w)")
AddPlot("Pseudocolor", "myvar") # Plot the expression variable.
DrawPlots()
DeleteExpression("myvar") # Delete the expression variable myvar.
```

DeleteNamedSelection: deletes knowledge of a named selection.

Synopsis:
`DeleteNamedSelection(name) -> integer`

Arguments:
`name` The name of a named selection.

Returns:
The DeleteNamedSelection function returns 1 for success and 0 for failure.

Description:
Named Selections allow you to select a group of elements (or particles). One typically creates a named selection from a group of elements and then later applies the named selection to another plot (thus reducing the set of elements displayed to the ones from when the named selection was created). If you have created a named selection that you are no longer interested in, you can delete it with the DeleteNamedSelection function.

Example:
```
% visit -cli
db = "/usr/gapps/visit/data/wave*.silo database"
OpenDatabase(db)
AddPlot("Pseudocolor", "pressure")
AddOperator("Clip")
c = ClipAttributes()
c.plane1Origin = (0,0.6,0)
c.plane1Normal = (0,-1,0)
SetOperatorOption(c)
DrawPlots()
CreateNamedSelection("els_above_y")
SetTimeSliderState(40)
DeleteNamedSelection("els_above_y")
CreateNamedSelection("els_above_y")
```

DeletePlotDatabaseKeyframe: deletes a database keyframe for a plot.

Synopsis:
`DeletePlotDatabaseKeyframe(plotIndex, frame)`

Arguments:

`plotIndex`	A zero−based integer value corresponding to a plot's index in the plotlist.
`frame`	A zero−based integer value corresponding to a database keyframe at aparticular animation frame.

Returns:
DeletePlotDatabaseKeyframe does not return a value.

Description:
The DeletePlotDatabaseKeyframe function removes a database keyframe from a specific plot. A database keyframe represents the database time state that will be used at a given animation frame when VisIt's keyframing mode is enabled. The plotIndex argument is a zero−based integer that is used to identify a plot in the plot list. The frame argument is a zero−based integer that is used to identify the frame at which a database keyframe is to be removed for the specified plot.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/wave.visit")
k = GetKeyframeAttributes()
k.enabled,k.nFrames,k.nFramesWasUserSet = 1,20,1
SetKeyframeAttributes(k)
AddPlot("Pseudocolor", "pressure")
SetPlotDatabaseState(0, 0, 60)
# Repeat time state 60 for the first few animation frames by adding a
keyframe at frame 3.
SetPlotDatabaseState(0, 3, 60)
SetPlotDatabaseState(0, 19, 0)
DrawPlots()
ListPlots()
# Delete the database keyframe at frame 3.
DeletePlotDatabaseKeyframe(0, 3)
ListPlots()
```

DeletePlotKeyframe: deletes a plot keyframe at a specified frame.

Synopsis:
`DeletePlotKeyframe(plotIndex, frame)`

Arguments:

`plotIndex`	A zero−based integer value corresponding to a plot's index in the plotlist.
`frame`	A zero−based integer value corresponding to a plot keyframe at aparticular animation frame.

Returns:
DeletePlotKeyframe does not return a value.

Description:
The DeletePlotKeyframe function removes a plot keyframe from a specific plot. A plot keyframe is the set of plot attributes at a specified frame. Plot keyframes are used to determine what plot attributes will be used at a given animation frame when VisIt's keyframing mode is enabled. The plotIndex argument is a zero−based integer that is used to identify a plot in the plot list. The frame argument is a zero−based integer that is used to identify the frame at which a keyframe is to be removed.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/wave.visit")
k = GetKeyframeAttributes()
k.enabled,k.nFrames,k.nFramesWasUserSet = 1,20,1
SetKeyframeAttributes(k)
AddPlot("Pseudocolor", "pressure")
# Set up plot keyframes so the Pseudocolor plot's min will change
# over time.
p0 = PseudocolorAttributes()
p0.minFlag,p0.min = 1,0.0
p1 = PseudocolorAttributes()
p1.minFlag,p1.min = 1, 0.5
SetPlotOptions(p0)
SetTimeSliderState(19)
SetPlotOptions(p1)
SetTimeSliderState(0)
DrawPlots()
ListPlots()
# Iterate over all animation frames and wrap around to the first one.
for i in list(range(TimeSliderGetNStates())) + [0]:
SetTimeSliderState(i)
# Delete the plot keyframe at frame 19 so the min won't
# change anymore.
DeletePlotKeyframe(19)
ListPlots()
SetTimeSliderState(10)
```

DeleteViewKeyframe: deletes a view keyframe at a specified frame.

Synopsis:
`DeleteViewKeyframe(frame)`

Arguments:

`frame`	A zero−based integer value corresponding to a view keyframe at aparticular animation frame.

Returns:
DeleteViewKeyframe returns 1 on success and 0 on failure.

Description:
The DeleteViewKeyframe function removes a view keyframe at a specified frame. View keyframes are used to determine what view will be used at a given animation frame when VisIt's keyframing mode is enabled. The frame argument is a zero−based integer that is used to identify the frame at which a keyframe is to be removed.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
k = KeyframeAttributes()
k.enabled, k.nFrames, k.nFramesWasUserSet = 1,10,1
SetKeyframeAttributes(k)
AddPlot("Pseudocolor", "u")
DrawPlots()
# Set some view keyframes
SetViewKeyframe()
v1 = GetView3D()
v1.viewNormal = (-0.66609, 0.337227, 0.665283)
v1.viewUp = (0.157431, 0.935425, -0.316537)
SetView3D(v1)
SetTimeSliderState(9)
SetViewKeyframe()
ToggleCameraViewMode()
# Iterate over the animation frames to watch the view change.
for i in list(range(10)) + [0]:
SetTimeSliderState(i)
# Delete the last view keyframe, which is on frame 9.
DeleteViewKeyframe(9)
# Iterate over the animation frames again. The view should stay
# the same.
for i in range(10):
SetTimeSliderState(i)
```

DeleteWindow: deletes the active visualization window.

Synopsis:
```
DeleteWindow() -> integer
```

Returns:
The DeleteWindow function returns an integer value of 1 for success and 0 for failure.

Description:
The DeleteWindow function deletes the active visualization window and makes the visualization window with the smallest window index the new active window. This function has no effect when there is only one remaining visualization window.

Example:
```
% visit -cli
DeleteWindow() # Does nothing since there is only one window
AddWindow()
DeleteWindow() # Deletes the new window.
```

DemoteOperator: moves an operator closer to the database in the visualization pipeline.

Synopsis:
```
DemoteOperator(opIndex) -> integer
DemoteOperator(opIndex, applyToAllPlots) -> integer
```

Arguments:

`opIndex`	A zero−based integer corresponding to the operator that should be demoted.
`applyAll`	An integer flag that causes all plots in the plot list to be affected whenit is non−zero.

Returns:
DemoteOperator returns 1 on success and 0 on failure.

Description:
The DemoteOperator function moves an operator closer to the database in the visualization pipeline. This allows you to change the order of operators that have been applied to a plot without having to remove them from the plot. For example, consider moving a Slice to before a Reflect operator when it had been the other way around. Changing the order of operators can result in vastly different results for a plot. The opposite function is PromoteOperator.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Pseudocolor", "hardyglobal")
AddOperator("Slice")
s = SliceAttributes()
s.project2d = 0
s.originPoint = (0,5,0)
s.originType=s.Point
s.normal = (0,1,0)
s.upAxis = (-1,0,0)
SetOperatorOptions(s)
AddOperator("Reflect")
DrawPlots()
# Now reflect before slicing. We'll only get 1 slice plane
# instead of 2.
DemoteOperator(1)
DrawPlots()
```

DisableRedraw: prevents the active visualization window from redrawing itself.

Synopsis:
`DisableRedraw()`

Returns:
The DisableRedraw function does not return a value.

Description:
The DisableRedraw function prevents the active visualization window from ever redrawing itself. This is a useful function to call when performing many operations that would cause unnecessary redraws in the visualization window. The effects of this function are undone by calling the RedrawWindow function.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Contour", "u")
AddPlot("Pseudocolor", "w")
DrawPlots()
DisableRedraw()
AddOperator("Slice")
# Set the slice operator attributes
# Redraw now that th operator attributes are set. This will
# prevent 1 redraw.
RedrawWindow()
```

DrawPlots: draws any new plots

Synopsis:
`DrawPlots() -> integer`

Returns:
The DrawPlots function returns an integer value of 1 for success and 0 for failure.

Description:
The DrawPlots function forces all new plots in the plot list to be drawn. Plots are added and then their attributes are modified. Finally, the DrawPlots function is called to make sure all of the new plots draw themselves in the visualization window. This function has no effect if all of the plots in the plot list are already drawn.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots() # Draw the new pseudocolor plot.
```

EnableTool: sets the enabled state of an interactive tool in the active visualization

Synopsis:
`EnabledTool(toolIndex, activeFlag)`

Arguments:

`toolIndex`	This is an integer that corresponds to an interactive tool. (Plane tool = 0, Line tool = 1, Plane tool = 2, Box tool = 3, Sphere tool = 4, Axis Restriction tool = 5)
`activeFlag`	A value of 1 enables the tool while a value of 0 disables the tool.

Returns:
The EnableTool function returns 1 on success and 0 on failure.

Description:
The EnableTool function is used to set the enabled state of an interactive tool in the active visualization window. The toolIndex argument is an integer index that corresponds to a certain tool. The activeFlag argument is an integer value (0 or 1) that indicates whether to turn the tool on or off.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
EnableTool(0, 1) # Turn on the line tool.
EnableTool(1,1) # Turn on the plane tool.
EnableTool(2,1) # Turn on the sphere tool.
EnableTool(2,0) # Turn off the sphere tool.
```

ExecuteMacro: execute a macro that has been registered using RegisterMacro.

Synopsis:
`ExecuteMacro(name) -> value`

Arguments:
`name` The name of the macro to execute.

Returns:
The ExecuteMacro function returns the value returned from the user's macro function.

Description:
The ExecuteMacro function lets you call a macro function that was previously registered using the RegisterMacro method. Once macros are registered with a name, this function can be called whenever the macro function associated with that name needs to be called. The VisIt gui uses this function to tell the Python interface when macros need to be executed in response to user button clicks.

Example:
```
def SetupMyPlots():
    OpenDatabase('noise.silo')
    AddPlot('Pseudocolor', 'hardyglobal')
    DrawPlots()
RegisterMacro('Setup My Plots', SetupMyPlots)
ExecuteMacro('Setup My Plots')
```

ExportDatabase: export a database

Synopsis:
```
ExportDatabase(e) -> integer
ExportDatabase(e, o) -> integer
```

Arguments:

`e`	An object of type ExportDBAttributes. This object specifies the optionsfor exporting the database.
`o (optional)`	A dictionary containing a key/value mapping to set options needed by thedatabase exporter. The default values can be obtained in the appropriateformat using GetExportOptions('plugin').

Returns:
Returns 1 on success, 0 on failure.

Description:
The ExportDatabase function exports the active plot for the current window to a file. The format of the file, name, and variables to be saved are specified using the ExportDBAttributes argument. Note that this functionality is distinct from the geometric formats of SaveWindow, such as STL. SaveWindow can only save surfaces (triangle meshes), while ExportDatabase can export an entire three dimensional data set.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/curv3d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
# Set the export database attributes.
e = ExportDBAttributes()
e.db_type = "Silo"
e.variables = ("u", "v")
e.filename = "test_ex_db"
ExportDatabase(e)
```

Expressions: returns a tuple of expression names and definitions.

Synopsis:
`Expressions() -> tuple of expression tuples`

Returns:
The Expressions function returns a tuple of tuples that contain two strings that give the expression name and definition.

Description:
The Expressions function returns a tuple of tuples that contain two strings that give the expression name and definition. This function is useful for listing the available expressions or for iterating through a list of expressions in order to create plots.

Example:
```
% visit -cli
SetWindowLayout(4)
DefineScalarExpression("sin_u", "sin(u)")
DefineScalarExpression("cos_u", "cos(u)")
DefineScalarExpression("neg_u", "-u")
DefineScalarExpression("bob", "sin_u + cos_u")
for i in range(1,5):
SetActiveWindow(i)
OpenDatabase("/usr/gapps/visit/data/globe.silo")
exprName = Expressions()[i-1][0]
AddPlot("Pseudocolor", exprName)
DrawPlots()
```

GetActiveContinuousColorTable: returns the name of the active color table.

Synopsis:
```
GetActiveContinuousColorTable() -> string
GetActiveDiscreteColorTable() -> string
```

Returns:
Both functions return a string object containing the name of a color table.

Description:
A color table is a set of color values that are used as the colors for plots. VisIt supports two flavors of color table: continuous and discrete. A continuous color table is defined by a small set of color control points and the colors specified by the color control points are interpolated smoothly to fill in any gaps. Continuous color tables are used for plots that need to be colored smoothly by a variable (e.g. Pseudocolor plot). A discrete color table is a set of color control points that are used to color distinct regions of a plot (e.g. Subset plot). VisIt supports the notion of default continuous and default discrete color tables so plots can just use the "default" color table. This lets you change the color table used by many plots by just changing the "default" color table. The GetActiveContinuousColorTable function returns the name of the default continuous color table. The GetActiveDiscreteColorTable function returns the name of the default discrete color table.

Example:
```
% visit -cli
print "Default continuous color table: %s" % \
GetActiveContinuousColorTable()
print "Default discrete color table: %s" % \
GetActiveDiscreteColorTable()
```

GetActiveDiscreteColorTable: returns the name of the active color table.

Synopsis:
```
GetActiveContinuousColorTable() -> string
GetActiveDiscreteColorTable() -> string
```

Returns:
Both functions return a string object containing the name of a color table.

Description:
A color table is a set of color values that are used as the colors for plots. VisIt supports two flavors of color table: continuous and discrete. A continuous color table is defined by a small set of color control points and the colors specified by the color control points are interpolated smoothly to fill in any gaps. Continuous color tables are used for plots that need to be colored smoothly by a variable (e.g. Pseudocolor plot). A discrete color table is a set of color control points that are used to color distinct regions of a plot (e.g. Subset plot). VisIt supports the notion of default continuous and default discrete color tables so plots can just use the "default" color table. This lets you change the color table used by many plots by just changing the "default" color table. The GetActiveContinuousColorTable function returns the name of the default continuous color table. The GetActiveDiscreteColorTable function returns the name of the default discrete color table.

Example:
```
% visit -cli
print "Default continuous color table: %s" % \
GetActiveContinuousColorTable()
print "Default discrete color table: %s" % \
GetActiveDiscreteColorTable()
```

GetActiveTimeSlider: returns the name of the active time slider.

Synopsis:
`GetActiveTimeSlider() -> string`

Returns:
The GetActiveTimeSlider function returns a string containing the name of the active time slider.

Description:
VisIt can support having multiple time sliders when you have opened more than one time−varying database. You can then use each time slider to independently change time states for each database or you can use a database correlation to change time states for all databases simultaneously. Every time−varying database has a database correlation and every database correlation has its own time slider. If you want to query to determine which time slider is currently the active time slider, you can use the GetActiveTimeSlider function.

Example:
```
% visit -cli
OpenDatabase("dbA00.pdb")
AddPlot("FilledBoundary", "material(mesh)")
OpenDatabase("dbB00.pdb")
AddPlot("FilledBoundary", "materials(mesh)")
print "Active time slider: %s" % GetActiveTimeSlider()
CreateDatabaseCorrelation("common", ("dbA00.pdb", "dbB00.pdb"), 2)
print "Active time slider: %s" % GetActiveTimeSlider()
```

GetAnimationAttributes: return a copy of the current animation attributes.

Synopsis:
```
GetAnimationAttributes() -> AnimationAttributes object
```

Arguments:
```
 none
```

Returns:
The GetAnimationAttributes function returns an AnimationAttributes object.

Description:
This function returns the current animation attributes, which contain the animation mode, increment, and playback speed.

Example:
```
a = GetAnimationAttributes()
print a
```

GetAnimationTimeout: returns the animation timeout in milliseconds.

Synopsis:
```
GetAnimationTimeout() -> integer
```

Returns:
The GetAnimationTimeout function returns an integer that contains the time interval, measured in milliseconds, between the rendering of animation frames.

Description:
The GetAnimationTimeout returns an integer that contains the time interval, measured in milliseconds, between the rendering of animation frames.

Example:
```
% visit -cli
print "Animation timeout = %d" % GetAnimationTimeout()
```

GetAnnotationAttributes: returns an object containing the active visualization window's annotation

Synopsis:
`GetAnnotationAttributes() -> AnnotationAttributes object`

Returns:
The GetAnnotationAttributes function returns an AnnotationAttributes object that contains the annotation settings for the active visualization window.

Description:
The GetAnnotationAttributes function returns an AnnotationAttributes object that contains the annotation settings for the active visualization window. It is often useful to retrieve the annotation settings and modify them to suit the visualization.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
a = GetAnnotationAttributes()
print a
a.backgroundMode = a.BACKGROUNDMODE_GRADIENT
a.gradientColor1 = (0, 0, 255)
SetAnnotationAttributes(a)
```

GetAnnotationObject: returns a reference to the annotation object at the specified index in

Synopsis:
`GetAnnotationObject(string) -> Annotation object`

Arguments:

`string`	The name of the annotation object as returned by GetAnnotationObjectNames.

Returns:
GetAnnotationObject returns a reference to an annotation object that was created using the CreateAnnotationObject function.

Description:
GetAnnotationObject returns a reference to an annotation object that was created using the CreateAnnotationObject function. The string argument specifies the name of the desired annotation object. It must be one of the names returned by GetAnnotationObjectNames. This function is not currently necessary unless the annotation object that you used to create an annotation has gone out of scope and you need to create another reference to the object to set its properties. Also note that although this function will apparently also accept an integer index, that mode of access is not reliably and should be avoided.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/wave.visit")
AddPlot("Pseudocolor", "pressure")
DrawPlots()
a = CreateAnnotationObject("TimeSlider")
GetAnnotationObjectNames()
["plot0000", "TimeSlider1"]
ref = GetAnnotationObject("TimeSlider1")
print ref
```

GetAnnotationObjectNames: returns a tuple of names of all currently defined annotation objects

Synopsis:
```
GetAnnotationObjectNames() -> tuple of strings
```

Returns:
GetAnnotationObjectNames returns a tuple of strings of the names of all annotation objects defined for the currently active window.

Example:
```
names = GetAnnotationObjectNames()
names
["plot0000", "Line2D1", "TimeSlider1"]
```

GetCallbackArgumentCount: ¡description¿

Synopsis:
`GetCallbackArgumentCount(callbackName) -> integer`

Arguments:

`callbackName`	The name of a callback function. This name is a member of the tuple returnedby GetCallbackNames().

Returns:
The GetCallbackArgumentCount function returns the number of arguments associated with a particular callback function.

Description:
None

Example:
```
cbName = 'OpenDatabaseRPC'
count = GetCallbackArgumentCount(cbName)
print 'The number of arguments for %s is: %d
' % (cbName, count)
```

GetCallbackNames: returns a list of allowable callback names for use with RegisterCallback().

Synopsis:
`GetCallbackNames() -> tuple of string objects`

Returns:
GetCallbackNames returns a tuple containing the names of valid callback function identifiers for use in RegisterCallback().

Description:
The GetCallbackNames function returns a tuple containing the names of valid callback function identifiers for use in RegisterCallback().

Example:
```
import visit
print visit.GetCallbackNames()
```

GetDatabaseNStates: returns the number of time states in the active database.

Synopsis:
`GetDatabaseNStates() -> integer`

Returns:
Returns the number of time states in the active database or 0 if there is no active database.

Description:
GetDatabaseNStates returns the number of time states in the active database, which is not the same as the number of states in the active time slider. Time sliders can have different lengths due to database correlations and keyframing. Use this function when you need the actual number of time states in the active database.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/wave*.silo database")
print "Number of time states: %d" % GetDatabaseNStates()
```

GetDebugLevel: set or Get the VisIt module's debug level.

Synopsis:
```
GetDebugLevel() -> integer
SetDebugLevel(level)
```

Arguments:

`level`	A string '1', '2', '3', '4', '5' with an optional 'b' suffix to indicatewhether the output should be buffered. A value of '1' is a low debug level, which should be used to produce little output while a value of 5 should produce a lot of debug output.

Returns:
The GetDebugLevel function returns the debug level of the VisIt module.

Description:
The GetDebugLevel and SetDebugLevel functions are used when debugging VisIt Python scripts. The SetDebugLevel function sets the debug level for VisIt's viewer thus it must be called before a Launch method. The debug level determines how much detail is written to VisIt's execution logs when it executes. The GetDebugLevel function can be used in Python scripts to alter the behavior of the script. For instance, the debug level can be used to selectively print values to the console.

Example:
```
% visit -cli -debug 2
print "VisIt's debug level is: %d" % GetDebugLevel()
```

GetDefaultFileOpenOptions: returns the current default options used when opening files for a

Synopsis:
`GetDefaultFileOpenOptions(pluginName) -> Dictionary`

Arguments:
`pluginName` The name of a plugin.

Returns:
Returns a dictionary containing the options.

Description:
GetDefaultFileOpenOptions returns the current options used to open new files when a specific plugin is triggered.

Example:
```
% visit -cli
OpenMDServer()
opts = GetDefaultFileOpenOptions("VASP")
opts["Allow multiple timesteps"] = 1
SetDefaultFileOpenOptions("VASP", opts)
OpenDatabase("CHGCAR")
```

GetDomains: returns a tuple containing the names of all of the domain subsets for the

Synopsis:
`GetDomains() -> tuple of strings`

Returns:
GetDomains returns a tuple of strings.

Description:
GetDomains returns a tuple containing the names of all of the domain subsets for a plot that was created using a database with multiple domains. This function can be used in specialized logic that iterates over domains to turn them on or off in some programmed way.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/multi_ucd3d.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
doms = GetDomains()
print doms
# Turn off all but the last domain, one after the other.
for d in doms[:-1]:
TurnDomainsOff(d)
```

GetEngineList: returns a tuple containing the names of the compute engines.

Synopsis:
`GetEngineList() -> tuple of strings`

Returns:
GetEngineList returns a tuple of strings that contain the names of the computers on which compute engines are running.

Description:
The GetEngineList function returns a tuple of strings containing the names of the computers on which compute engines are running. This function can be useful if engines are going to be closed and opened explicitly in the Python script. The contents of the tuple can be used to help determine which compute engines should be closed or they can be used to determine if a compute engine was successfully launched.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
OpenDatabase("mcr:/usr/gapps/visit/data/globe.silo")
AddPlot("Mesh", "mesh1")
DrawPlots()
for name in GetEngineList():
print "VisIt has a compute engine running on %s" % name
CloseComputeEngine(GetEngineList()[1])
```

GetGlobalAttributes: returns a GlobalAttributes object.

Synopsis:
`GetGlobalAttributes() -> GlobalAttributes object`

Returns:
Returns a GlobalAttributes object that has been initialized.

Description:
The GetGlobalAttributes function returns a GlobalAttributes object that has been initialized with the current state of the viewer proxy's GlobalAttributes object. The GlobalAttributes object contains read−only information about the list of sources, the list of windows, and various flags that can be queried.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
g = GetGlobalAttributes()
print g
```

GetGlobalLineoutAttributes: returns a GlobalLineoutAttributes object.

Synopsis:
`GetGlobalLineoutAttributes() -> GlobalLineoutAttributes object`

Returns:
Returns an initialized GlobalLineoutAttributes object.

Description:
The GetGlobalLineoutAttributes function returns an initialized GlobalLineoutAttributes object. The GlobalLineoutAttributes, as suggested by its name, contains global properties that apply to all lineouts. You can use the GlobalLineoutAttributes object to turn on lineout sampling, specify the destination window, etc. for curve plots created as a result of performing lineouts. Once you make changes to the object by setting its properties, use the SetGlobalLineoutAttributes function to make VisIt use the modified global lineout attributes.

Example:
```
% visit -cli
SetWindowLayout(4)
OpenDatabase("/usr/gapps/visit/data/curv2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
g = GetGlobalLineoutAttributes()
print g
g.samplingOn = 1
g.windowId = 4
g.createWindow = 0
g.numSamples = 100
SetGlobalLineoutAttributes(g)
Lineout((-3,2),(3,3),("default"))
```

GetInteractorAttributes: returns an InteractorAttributes object.

Synopsis:
`GetInteractorAttributes() -> InteractorAttributes object`

Returns:
Returns an initialized InteractorAttributes object.

Description:
The GetInteractorAttributes function returns an initialized InteractorAttributes object. The InteractorAttributes object can be used to set certain interactor properties. Interactors, can be thought of as how mouse clicks and movements are translated into actions in the vis window. To set the interactor attributes, first get the interactor attributes using the GetInteractorAttributes function. Once you've set the object's properties, call the SetInteractorAttributes function to make VisIt use the new interactor attributes.

Example:
```
% visit -cli
ia = GetInteractorAttributes()
print ia
ia.showGuidelines = 0
SetInteractorAttributes(ia)
```

GetKeyframeAttributes: returns an initialized KeyframeAttributes object.

Synopsis:
`GetKeyframeAttributes() -> KeyframeAttributes object`

Returns:
GetKeyframeAttributes returns an initialized KeyframeAttributes object.

Description:
Use the GetKeyframeAttributes function when you want to examine a KeyframeAttributes object so you can determine VisIt's state when it is in keyframing mode. The KeyframeAttributes object allows you to see whether VisIt is in keyframing mode and, if so, how many animation frames are in the current keyframe animation.

Example:
```
% visit -cli
k = GetKeyframeAttributes()
print k
k.enabled,k.nFrames,k.nFramesWasUserSet = 1, 100, 1
SetKeyframeAttributes(k)
```

GetLastError: returns a string containing the last error message that VisIt issued.

Synopsis:
`GetLastError() -> string`

Returns:
GetLastError returns a string containing the last error message that VisIt issued.

Description:
The GetLastError function returns a string containing the last error message that VisIt issued.

Example:
```
% visit -cli
OpenDatabase("/this/database/does/not/exist")
print "VisIt Error: %s" % GetLastError()
```

GetLight: returns a light object containing the attributes for a specified light.

Synopsis:
`GetLight(index) -> LightAttributes object`

Arguments:

`index`	A zero−based integer index into the light list. Index can be in the range[0,7].

Returns:
GetLight returns a LightAttributes object.

Description:
The GetLight function returns a LightAttributes object containing the attributes for a specific light. You can use the LightAttributes object that GetLight returns to set light properties and then you can pass the object to SetLight to make VisIt use the light properties that you've set.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "w")
p = PseudocolorAttributes()
p.colorTableName = "xray"
SetPlotOptions(p)
DrawPlots()
InvertBackgroundColor()
light = GetLight(0)
print light
light.enabledFlag = 1
light.direction = (0,-1,0)
light.color = (255,0,0,255)
SetLight(0, light)
light.color,light.direction = (0,255,0,255), (-1,0,0)
SetLight(1, light)
```

GetLocalHostName: gets the local user or host name.

Synopsis:
```
GetLocalHostName() -> string
GetLocalUserName() -> string
```

Returns:
Both functions return a string.

Description:
These functions are useful for determining the name of the local computer or the account name of the user running VisIt. The GetLocalHostName function returns a string that contains the name of the local computer. The GetLocalUserName function returns a string containing the name of the user running VisIt.

Example:
```
% visit -cli
print "Local machine name is: %s" % GetLocalHostName()
print "My username: %s" % GetLocalUserName()
```

GetLocalUserName: gets the local user or host name.

Synopsis:
```
GetLocalHostName() -> string
GetLocalUserName() -> string
```

Returns:
Both functions return a string.

Description:
These functions are useful for determining the name of the local computer or the account name of the user running VisIt. The GetLocalHostName function returns a string that contains the name of the local computer. The GetLocalUserName function returns a string containing the name of the user running VisIt.

Example:
```
% visit -cli
print "Local machine name is: %s" % GetLocalHostName()
print "My username: %s" % GetLocalUserName()
```

GetMachineProfile: ¡description¿

Synopsis:
`GetMachineProfile(hostname) -> MachineProfile`

Arguments:
`hostname`

Returns:
MachineProfile for hostname

Description:
Gets the MachineProfile for a given hostname

GetMachineProfileNames: ¡description¿

Synopsis:
`GetMachineProfileNames() -> [hostname1, hostname2, ...]`

Returns:
List of MachineProfile hostnames

Description:
Returns a list of hostnames that can be used to get a specific MachineProfile

GetMaterialAttributes: returns a MaterialAttributes object containing VisIt's current material

Synopsis:
`GetMaterialAttributes() -> MaterialAttributes object`

Returns:
Returns a MaterialAttributes object.

Description:
The GetMaterialAttributes function returns a MaterialAttributes object that contains VisIt's current material interface reconstruction settings. You can set properties on the MaterialAttributes object and then pass it to SetMaterialAttributes to make VisIt use the new material attributes that you've specified:

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/allinone00.pdb")
AddPlot("Pseudocolor", "mesh/mixvar")
p = PseudocolorAttributes()
p.min,p.minFlag = 4.0, 1
p.max,p.maxFlag = 13.0, 1
SetPlotOptions(p)
DrawPlots()
# Tell VisIt to always do material interface reconstruction.
m = GetMaterialAttributes()
m.forceMIR = 1
SetMaterialAttributes(m)
ClearWindow()
# Redraw the plot forcing VisIt to use the mixed variable information.
DrawPlots()
```

GetMaterials: returns a string tuple of material names for the current plot's database.

Synopsis:
`GetMaterials() -> tuple of strings`

Returns:
The GetMaterials function returns a tuple of strings.

Description:
The GetMaterials function returns a tuple of strings containing the names of the available materials for the current plot's database. Note that the active plot's database must have materials for this function to return a tuple that has any string objects in it. Also, you must have at least one plot. You can use the materials returned by the GetMaterials function for a variety of purposes including turning materials on or off.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/allinone00.pdb")
AddPlot("Pseudocolor", "mesh/mixvar")
DrawPlots()
mats = GetMaterials()
for m in mats[:-1]:
TurnMaterialOff(m)
```

GetMeshManagementAttributes: returns a MeshManagementAttributes object containing VisIt's current mesh

Synopsis:
`GetMeshmanagementAttributes() -> MeshmanagementAttributes object`

Returns:
Returns a MeshmanagementAttributes object.

Description:
The GetMeshmanagementAttributes function returns a MeshmanagementAttributes object that contains VisIt's current mesh discretization settings. You can set properties on the MeshManagementAttributes object and then pass it to SetMeshManagementAttributes to make VisIt use the new material attributes that you've specified:

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/csg.silo")
AddPlot("Mesh", "csgmesh")
DrawPlots()
# Tell VisIt to always do material interface reconstruction.
mma = GetMeshManagementAttributes()
mma.discretizationTolernace = (0.01, 0.025)
SetMeshManagementAttributes(mma)
ClearWindow()
# Redraw the plot forcing VisIt to use the mixed variable information.
DrawPlots()
```

GetMetaData: return a metadata object for the specified database without making it be the active database.

Synopsis:
```
GetMetaData(db) -> avtDatabaseMetaData object
GetMetaData(db, ts) -> avtDatabaseMetaData object
```

Arguments:

`db`	The name of the database for which to return metadata.
`ts`	An optional integer indicating the time state at which to open the database.

Returns:
The GetMetaData function returns an avtDatabaseMetaData object.

Description:
VisIt relies on metadata to populate its variable menus and make important decisions. Metadata can be used to create complex scripts whose behavior adapts based on the contents of the database.

Example:
```
md = GetMetaData('noise.silo')
for i in xrange(md.GetNumScalars()):
    AddPlot('Pseudocolor', md.GetScalars(i).name)
DrawPlots()
```

GetNumPlots: returns the number of plots in the active window.

Synopsis:
```
GetNumPlots() -> integer
```

Returns:
Returns the number of plots in the active window.

Description:
The GetNumPlots function returns the number of plots in the active window.

Example:
```
% visit -cli
print "Number of plots", GetNumPlots()
OpenDatabase("/usr/gapps/visit/data/curv2d.silo")
AddPlot("Pseudocolor", "d")
print "Number of plots", GetNumPlots()
AddPlot("Mesh", "curvmesh2d")
DrawPlots()
print "Number of plots", GetNumPlots()
```

GetOperatorOptions: return a copy of the operator attributes for the i'th operator in the first

Synopsis:
`GetOperatorOptions(index) -> operator attributes object`

Arguments:
`index` The index of the operator within the plot's list of operators.

Returns:
The GetOperatorOptions function returns an operator attributes object.

Description:
This function is provided to make it easy to probe the current attributes for a specific operator on the active plot.

Example:
```
AddPlot('Pseudocolor', 'temperature')
AddOperator('Transform')
AddOperator('Transform')
t = GetOperatorOptions(1)
print 'Attributes for the 2nd Transform operator:', t
```

GetPickAttributes: returns the current pick attributes.

Synopsis:
`GetPickAttributes() -> PickAttributes object`

Returns:
GetPickAttributes returns a PickAttributes object.

Description:
The GetPickAttributes object returns the pick settings that VisIt is currently using when it performs picks. These settings mainly determine which pick information is displayed when pick results are printed out but they can also be used to select auxiliary variables and generate time curves. You can examing the settings and you can set properties on the returned object. Once you've changed pick settings by setting properties on the object, you can pass the altered object to the SetPickAttributes function to force VisIt to use the new pick settings.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/allinone00.pdb")
AddPlot("Pseudocolor", "mesh/ireg")
DrawPlots()
p = GetPickAttributes()
print p
p.variables = ("default", "mesh/a", "mesh/mixvar")
SetPickAttributes(p)
# Now do some interactive picks and you'll see pick information
# for more than 1 variable.
p.doTimeCurve = 1
SetPickAttributes(p)
# Now do some interactive picks and you'll get time-curves in
# a new window.
```

GetPickOutput: returns a string containing the output from the last pick.

Synopsis:
```
GetPickOutput() -> string
GetPickOutputObject() -> dictonary
```

Returns:
GetPickOutput returns a string containing the output from the last pick. GetPickOutputObject returns a dictionary produced by the last pick.

Description:
The GetPickOutput returns a string object that contains the output from the last pick. GetPickOutputObject returns a dictionary object containing output from the last pick

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/rect2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
ZonePick((0.4, 0.6), ("default", "u", "v"))
s = GetPickOutput()
print s
o = GetPickOutputObject()
print o
```

GetPickOutputObject: returns a string containing the output from the last pick.

Synopsis:
```
GetPickOutput() -> string
GetPickOutputObject() -> dictonary
```

Returns:
GetPickOutput returns a string containing the output from the last pick. GetPickOutputObject returns a dictionary produced by the last pick.

Description:
The GetPickOutput returns a string object that contains the output from the last pick. GetPickOutputObject returns a dictionary object containing output from the last pick

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/rect2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
ZonePick((0.4, 0.6), ("default", "u", "v"))
s = GetPickOutput()
print s
o = GetPickOutputObject()
print o
```

GetPipelineCachingMode: returns if pipelines are cached in the viewer.

Synopsis:
`GetPipelineCachingMode() -> integer`

Returns:
The GetPipelineCachingMode function returns 1 if pipelines are being cached and 0 otherwise.

Description:
The GetPipelineCachingMode function returns whether or not pipelines are being cached in the viewer. For animations of long time sequences, it is often useful to turn off pipeline caching so the viewer does not run out of memory.

Example:
```
%visit -cli
offon = ("off", "on")
print "Pipeline caching is %s" % offon[GetPipelineCachingMode()]
```

GetPlotInformation: returns a dictionary of plot information for the active plot.

Synopsis:
`GetPlotInformation() -> dictionary`

Returns:
GetPlotInformation returns a dictionary.

Description:
The GetPlotInformation function returns information about the active plot. For example, a Curve plot will return the xy pairs that comprise the curve. The tuple is arranged <x1, y1, x2, y2, ..., xn, yn>.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/rect2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
Lineout((0, 0), (1, 1))
SetActiveWindow(2)
info = GetPlotInformation()
lineout = info["Curve"]
print "The first lineout point is: [%g, %g] " % lineout[0], lineout[1]
```

GetPlotList: return a copy of the plot list object.

Synopsis:
`GetPlotList() -> PlotList object`

Arguments:
`none`

Returns:
The GetPlotList function returns a PlotList object.

Description:
The GetPlotList function returns a copy of the plot list that gets exchanged between VisIt's viewer and its clients. The plot list object contains the list of plots, along with the databases, and any operators that are applied to each plot. Changing this object has NO EFFECT but it can be useful when writing complex functions that need to know about the plots and operators that exist within a visualization window

Example:
```
# Copy plots (without operators to window 2)
pL = GetPlotList()
AddWindow()
for i in xrange(pL.GetNumPlots()):
    AddPlot(PlotPlugins()[pL.GetPlots(i).plotType], pL.GetPlots(i).plotVar)
DrawPlots()
```

GetPlotOptions: return a copy of the plot options for the first active plot of the first plot

Synopsis:
`GetPlotOptions() -> plot attributes object`

Arguments:
`none`

Returns:
The GetPlotOptions function returns a plot attributes object whose type varies depending the selected plots.

Description:
This function is provided to make it easy to probe the current attributes for the selected plot.

Example:
```
pc = GetPlotOptions()
pc.legend = 0
SetPlotOptions(pc)
```

GetPreferredFileFormats: gets the list of preferred file format IDs.

Synopsis:
```
GetPreferredFileFormats() -> tuple of strings
Arguments: none
```

Returns:
The GetPreferredFileFormats returns the current list of preferred plugins.

Description:
The GetPreferredFileFormats method is a way to get the list of file format reader plugins which are tried before any others. These IDs are full IDs, not just names, and are tried in order.

Example:
```
GetPreferredFileFormats()
# returns ('Silo_1.0',)
```

GetQueryOutputObject: returns information about the last query.

Synopsis:
```
GetQueryOutputString() -> string
GetQueryOutputValue() -> double, tuple of doubles
GetQueryOutputXML() -> string
GetQueryOutputObject() -> dictonary or value
```

Returns:
GetQueryOutputString returns a string. GetQueryOutputValue returns a single double precision number or a tuple of double precision numbers. GetQueryOutputXML returns an xml string produced by the last query. GetQueryOutputObject returns an xml string produced by the last query.

Description:
Both the GetQueryOutputString and GetQueryOutputValue functions return information about the last query to be executed but the type of information returns differs. GetQueryOutputString returns a string containing the output of the last query. GetQueryOutputValue returns a single number or tuple of numbers, depending on the nature of the last query to be executed. GetQueryOutputXML and GetQueryOutputObject expose more complex query output.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/rect2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
Query("MinMax")
print GetQueryOutputString()
print "The min is: %g and the max is: %g" % GetQueryOutputValue()
```

GetQueryOutputString: returns information about the last query.

Synopsis:
```
GetQueryOutputString() -> string
GetQueryOutputValue() -> double, tuple of doubles
GetQueryOutputXML() -> string
GetQueryOutputObject() -> dictonary or value
```

Returns:
GetQueryOutputString returns a string. GetQueryOutputValue returns a single double precision number or a tuple of double precision numbers. GetQueryOutputXML returns an xml string produced by the last query. GetQueryOutputObject returns an xml string produced by the last query.

Description:
Both the GetQueryOutputString and GetQueryOutputValue functions return information about the last query to be executed but the type of information returns differs. GetQueryOutputString returns a string containing the output of the last query. GetQueryOutputValue returns a single number or tuple of numbers, depending on the nature of the last query to be executed. GetQueryOutputXML and GetQueryOutputObject expose more complex query output.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/rect2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
Query("MinMax")
print GetQueryOutputString()
print "The min is: %g and the max is: %g" % GetQueryOutputValue()
```

GetQueryOutputValue: returns information about the last query.

Synopsis:
```
GetQueryOutputString() -> string
GetQueryOutputValue() -> double, tuple of doubles
GetQueryOutputXML() -> string
GetQueryOutputObject() -> dictonary or value
```

Returns:
GetQueryOutputString returns a string. GetQueryOutputValue returns a single double precision number or a tuple of double precision numbers. GetQueryOutputXML returns an xml string produced by the last query. GetQueryOutputObject returns an xml string produced by the last query.

Description:
Both the GetQueryOutputString and GetQueryOutputValue functions return information about the last query to be executed but the type of information returns differs. GetQueryOutputString returns a string containing the output of the last query. GetQueryOutputValue returns a single number or tuple of numbers, depending on the nature of the last query to be executed. GetQueryOutputXML and GetQueryOutputObject expose more complex query output.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/rect2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
Query("MinMax")
print GetQueryOutputString()
print "The min is: %g and the max is: %g" % GetQueryOutputValue()
```

GetQueryOutputXML: returns information about the last query.

Synopsis:
```
GetQueryOutputString() -> string
GetQueryOutputValue() -> double, tuple of doubles
GetQueryOutputXML() -> string
GetQueryOutputObject() -> dictonary or value
```

Returns:
GetQueryOutputString returns a string. GetQueryOutputValue returns a single double precision number or a tuple of double precision numbers. GetQueryOutputXML returns an xml string produced by the last query. GetQueryOutputObject returns an xml string produced by the last query.

Description:
Both the GetQueryOutputString and GetQueryOutputValue functions return information about the last query to be executed but the type of information returns differs. GetQueryOutputString returns a string containing the output of the last query. GetQueryOutputValue returns a single number or tuple of numbers, depending on the nature of the last query to be executed. GetQueryOutputXML and GetQueryOutputObject expose more complex query output.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/rect2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
Query("MinMax")
print GetQueryOutputString()
print "The min is: %g and the max is: %g" % GetQueryOutputValue()
```

GetQueryOverTimeAttributes: returns a QueryOverTimeAttributes object containing the settings that

Synopsis:
`GetQueryOverTimeAttributes() -> QueryOverTimeAttributes object`

Returns:
GetQueryOverTimeAttributes returns a QueryOverTimeAttributes object.

Description:
The GetQueryOverTimeAttributes function returns a QueryOverTimeAttributes object containing the settings that VisIt currently uses for query over time. You can use the returned object to change those settings by first setting object properties and then by passing the modified object to the SetQueryOverTimeAttributes function.

Example:
```
% visit -cli
SetWindowLayout(4)
OpenDatabase("/usr/gapps/visit/data/allinone00.pdb")
AddPlot("Pseudocolor", "mesh/mixvar")
DrawPlots()
qot = GetQueryOverTimeAttributes()
print qot
# Make queries over time go to window 4.
qot.createWindow,q.windowId = 0, 4
SetQueryOverTimeAttributes(qot)
QueryOverTime("Min")
# Make queries over time only use half of the number of time states.
endTime = GetDatabaseNStates() / 2
QueryOverTime("Min", end_time=endTime)
ResetView()
```

GetQueryParameters: returns information about the default parameters used by the named query.

Synopsis:
`GetQueryParameters(name) -> python dictonary`

Returns:
A python dictionary.

Description:
The GetQueryParameters function returns a Python dictionary containing the default parameters for the named query, or None if the query does not accept additional parameters. The returned dictionary (if any) can then be modified if necessary and passed back as an argument to the Query function.

Example:
```
% visit -cli
minMaxInput = GetQueryParameters("MinMax")
minMaxInput["use_actual_data"] = 1
Query("MinMax", minMaxInput)
xrayInput = GetQueryParameters("XRay Image")
xrayInput["origin"]=(0.5, 2.5, 0.)
xrayInput["image_size"]=(300,300)
xrayInput["vars"]=("p", "d")
Query("XRay Image", xrayInput)
```

GetRenderingAttributes: returns a RenderingAttributes object containing VisIt's current rendering

Synopsis:
`GetRenderingAttributes() -> RenderingAttributes object`

Returns:
Returns a RenderingAttributes object.

Description:
The GetRenderingAttributes function returns a RenderingAttributes object that contains the rendering settings that VisIt currently uses. The RenderingAttributes object contains information related to rendering such as whether or not specular highlights or shadows are enabled. The RenderingAttributes object also contains information scalable rendering such as whether or not it is currently in use and the scalable rendering threshold.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Surface", "hgslice")
DrawPlots()
v = GetView3D()
v.viewNormal = (-0.215934, -0.454611, 0.864119)
v.viewUp = (0.973938, -0.163188, 0.157523)
v.imageZoom = 1.64765
SetView3D(v)
light = GetLight(0)
light.direction = (0,1,-1)
SetLight(0, light)
r = GetRenderingAttributes()
r.scalableActivationMode = r.Always
r.doShadowing = 1
SetRenderingAttributes(r)
```

GetSaveWindowAttributes: returns an object that contains the attributes used to save windows.

Synopsis:
`GetSaveWindowAttributes() -> SaveWindowAttributes object`

Returns:
This function returns a VisIt SaveWindowAttributes object that contains the attributes used in saving windows.

Description:
The GetSaveWindowAttributes function returns a SaveWindowAttributes object that is a structure containing several fields which determine how windows are saved to files. The object that us returned can be modified and used to set the save window attributes.

Example:
```
% visit -cli
s = GetSaveWindowAttributes()
print s
s.width = 600
s.height = 600
s.format = s.RGB
print s
```

GetSelection: return the selection properties for the specified selection name.

Synopsis:
`GetSelection(name) -> SelectionProperties object`

Arguments:

`name`	The name of the selection whose properties we want to retrieve.

Returns:
The GetSelection function returns a SelectionProperties object.

Description:
Named selections have properties that describe how the selection is defined. This function lets you query those selection properties.

Example:
```
CreateNamedSelection('selection1')
s = GetSelection('selection1')
s.selectionType = s.CumulativeQuerySelection
s.histogramType = s.HistogramMatches
s.combineRule = s.CombineOr
s.variables = ('temperature',)
s.variableMins = (2.9,)
s.variableMaxs = (3.1,)
UpdateNamedSelection('selection1', s)
```

GetSelectionList: return a copy of the selection list.

Synopsis:
`GetSelectionList() -> SelectionList object`

Arguments:
`none`

Returns:
The GetSelectionList function returns a SelectionList object.

Description:
VisIt maintains a list of named selections, which are sets of cells that are used to restrict the cells processed by other plots. This function returns a list of the selections that VisIt knows about, including their properties.

Example:
```
s = GetSelectionList()
```

GetSelectionSummary: return the selection summary for the specified selection name.

Synopsis:
`GetSelectionSummary(name) -> SelectionSummary object`

Arguments:

`name`	The name of the selection whose summary we want to retrieve.

Returns:
The GetSelectionSummary function returns a SelectionSummary object.

Description:
Named selections have both properties, which describe how the selection is defined, and a summary that desribes the data that was processed while creating the selection. The selection summary object contains some statistics about the selection such as how many cells it contains and histograms of the various variables that were used in creating the selection.

Example:

```
print GetSelectionSummary('selection1')
```

GetTimeSliders: returns a tuple containing the names of all of the available time sliders.

Synopsis:
`GetTimeSliders() -> tuple of strings`

Returns:
GetTimeSliders returns a tuple of strings.

Description:
The GetTimeSliders function returns a tuple of strings containing the names of each of the available time sliders. The list of time sliders contains the names of any open time−varying database, all database correlations, and the keyframing time slider if VisIt is in keyframing mode.

Example:
```
% visit -cli
path = "/usr/gapps/visit/data/"
dbs = (path + "/dbA00.pdb", path + "dbB00.pdb", path + "dbC00.pdb")
for db in dbs:
OpenDatabase(db)
AddPlot("FilledBoundary", "material(mesh)")
DrawPlots()
CreateDatabaseCorrelation("common", dbs, 1)
print "The list of time sliders is: ", GetTimeSliders()
```

GetUltraScript: return the file to be used by the LoadUltra function.

Synopsis:
`GetUltraScript() -> string`

Arguments:
`none`

Returns:
The GetUltraScript function returns a filename.

Description:
Return the name of the file in use by the LoadUltra function. Normal users do not need to use this function.

GetView2D: return an object containing the current view.

Synopsis:
```
GetViewCurve() -> ViewCurveAttributes object
GetView2D() -> View2DAttributes object
GetView3D() -> View3DAttributes object
GetViewAxisArray() -> ViewAxisArrayAttributes object
```

Returns:
Both functions return objects that represent the curve, 2D, or 3D view information.

Description:
The GetView functions return ViewAttributes objects which describe the current camera location. The GetView2D function should be called if the active visualization window contains 2D plots. The GetView3D function should be called to get the view if the active visualization window contains 3D plots. The GetViewCurve function should be called if the active visualization window contains 1D curve plots. The GetViewAxisArray function should be called if the active visualization window contains axis−array plots.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
# Change the view interactively using the mouse.
v0 = GetView3D()
# Change the view again using the mouse
v1 = GetView3D()
print v0
for i in range(0,20):
t = float(i) / 19.
v2 = (1. - t) * v1 + t * v0
SetView3D(v2) # Animate the view back to the first view.
```

GetView3D: return an object containing the current view.

Synopsis:
```
GetViewCurve() -> ViewCurveAttributes object
GetView2D() -> View2DAttributes object
GetView3D() -> View3DAttributes object
GetViewAxisArray() -> ViewAxisArrayAttributes object
```

Returns:
Both functions return objects that represent the curve, 2D, or 3D view information.

Description:
The GetView functions return ViewAttributes objects which describe the current camera location. The GetView2D function should be called if the active visualization window contains 2D plots. The GetView3D function should be called to get the view if the active visualization window contains 3D plots. The GetViewCurve function should be called if the active visualization window contains 1D curve plots. The GetViewAxisArray function should be called if the active visualization window contains axis−array plots.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
# Change the view interactively using the mouse.
v0 = GetView3D()
# Change the view again using the mouse
v1 = GetView3D()
print v0
for i in range(0,20):
t = float(i) / 19.
v2 = (1. - t) * v1 + t * v0
SetView3D(v2) # Animate the view back to the first view.
```

GetViewAxisArray: return an object containing the current view.

Synopsis:
```
GetViewCurve() -> ViewCurveAttributes object
GetView2D() -> View2DAttributes object
GetView3D() -> View3DAttributes object
GetViewAxisArray() -> ViewAxisArrayAttributes object
```

Returns:
Both functions return objects that represent the curve, 2D, or 3D view information.

Description:
The GetView functions return ViewAttributes objects which describe the current camera location. The GetView2D function should be called if the active visualization window contains 2D plots. The GetView3D function should be called to get the view if the active visualization window contains 3D plots. The GetViewCurve function should be called if the active visualization window contains 1D curve plots. The GetViewAxisArray function should be called if the active visualization window contains axis−array plots.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
# Change the view interactively using the mouse.
v0 = GetView3D()
# Change the view again using the mouse
v1 = GetView3D()
print v0
for i in range(0,20):
t = float(i) / 19.
v2 = (1. - t) * v1 + t * v0
SetView3D(v2) # Animate the view back to the first view.
```

GetViewCurve: return an object containing the current view.

Synopsis:
```
GetViewCurve() -> ViewCurveAttributes object
GetView2D() -> View2DAttributes object
GetView3D() -> View3DAttributes object
GetViewAxisArray() -> ViewAxisArrayAttributes object
```

Returns:
Both functions return objects that represent the curve, 2D, or 3D view information.

Description:
The GetView functions return ViewAttributes objects which describe the current camera location. The GetView2D function should be called if the active visualization window contains 2D plots. The GetView3D function should be called to get the view if the active visualization window contains 3D plots. The GetViewCurve function should be called if the active visualization window contains 1D curve plots. The GetViewAxisArray function should be called if the active visualization window contains axis−array plots.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
# Change the view interactively using the mouse.
v0 = GetView3D()
# Change the view again using the mouse
v1 = GetView3D()
print v0
for i in range(0,20):
t = float(i) / 19.
v2 = (1. - t) * v1 + t * v0
SetView3D(v2) # Animate the view back to the first view.
```

GetWindowInformation: returns a WindowInformation object that contains information about the

Synopsis:
`GetWindowInformation() -> WindowInformation object`

Returns:
The GetWindowInformation object returns a WindowInformation object.

Description:
The GetWindowInformation object returns a WindowInformation object that contains information about the active visualization window. The WindowInformation object contains the name of the active source, the active time slider index, the list of available time sliders and their current states, as well as certain window flags that determine whether a window's view is locked, etc. Use the WindowInformation object if you need to query any of these types of information in your script to influence how it behaves.

Example:
```
path = "/usr/gapps/visit/data/"
dbs = (path + "dbA00.pdb", path + "dbB00.pdb", path + "dbC00.pdb")
for db in dbs:
OpenDatabase(db)
AddPlot("FilledBoundary", "material(mesh)")
DrawPlots()
CreateDatabaseCorrelation("common", dbs, 1)
# Get the list of available time sliders.
tsList = GetWindowInformation().timeSliders
# Iterate through "time" on each time slider.
for ts in tsList:
SetActiveTimeSlider(ts)
for state in range(TimeSliderGetNStates()):
SetTimeSliderState(state)
# Print the window information to examine the other attributes
# that are available.
GetWindowInformation()
```

HideActivePlots: hides the active plots in the active visualization window.

Synopsis:
`HideActivePlots() -> integer`

Returns:
The HideActivePlots function returns an integer value of 1 for success and 0 for failure.

Description:
The HideActivePlots function tells the viewer to hide the active plots in the active visualization window.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
AddPlot("Mesh", "mesh1")
DrawPlots()
SetActivePlots(0)
HideActivePlots()
AddPlot("FilledBoundary", "mat1")
DrawPlots()
```

HideToolbars: hides the visualization window's toolbars.

Synopsis:
```
HideToolbars() -> integer
HideToolbars(allWindows) ->integer
```

Arguments:

`allWindows`	An optional integer value that tells VisIt to hide the toolbars for allwindows when it is non−zero.

Returns:
The HideToolbars function returns 1 on success and 0 on failure.

Description:
The HideToolbars function tells VisIt to hide the toolbars for the active visualization window or for all visualization windows when the optional allWindows argument is provided and is set to a non−zero value.

Example:
```
% visit -cli
SetWindowLayout(4)
HideToolbars()
ShowToolbars()
# Hide the toolbars for all windows.
HideToolbars(1)
```

IconifyAllWindows: minimizes all of the visualization windows.

Synopsis:
`IconifyAllWindows()`

Returns:
The IconifyAllWindows function does not return a value.

Description:
The IconifyAllWindows function minimizes all of the hidden visualization windows to get them out of the way.

Example:
```
% visit -cli
SetWindowLayout(4) # Have 4 windows
IconifyAllWindows()
DeIconifyAllWindows()
```

InitializeNamedSelectionVariables: initialize a named selection's properties with variables from compatible plots and operators.

Synopsis:
`InitializeNamedSelectionVariables(name) -> integer`

Arguments:
`name` The name of the named selection to initialize.

Returns:
The InitializeNamedSelectionVariables function returns 1 on success and 0 on failure.

Description:
Complex thresholds are often defined using the Parallel Coordinates plot or the Threshold operator. This function can copy variable ranges from compatible plots and operators into the specified named selection's properties. This can be useful when setting up Cumulative Query selections.

Example:
`InitializeNamedSelectionVariables('selection1')`

InvertBackgroundColor: swaps the background and foreground colors in the active visualization

Synopsis:
`InvertBackgroundColor()`

Returns:
The InvertBackgroundColor function does not return a value.

Description:
The InvertBackgroundColor function swaps the background and foreground colors in the active visualization window. This function is a cheap alternative to setting the foreground and background colors though the AnnotationAttributes in that it is a simple no−argument function call. It is not adequate to set new colors for the background and foreground, but in the event where the two colors can be exchanged favorably, it is a good function to use. An example of when this function is used is after the creation of a Volume plot.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Volume", "u")
DrawPlots()
InvertBackgroundColor()
```

Launch: launches VisIt's viewer

Synopsis:
```
Launch() -> integer
Launch(program) -> integer
LaunchNowin() -> integer
LaunchNowin(program) -> integer
```

Arguments:
`program` The complete path to the top level 'visit' script.

Returns:
The Launch functions return 1 for success and 0 for failure

Description:
The Launch function is used to launch VisIt's viewer when the VisIt module is imported into a stand−alone Python interpreter. The Launch function has no effect when a viewer already exists. The difference between Launch and LaunchNowin is that LaunchNowin prevents the viewer from ever creating onscreen visualization windows. The LaunchNowin function is primarily used in Python scripts that want to generate visualizations using VisIt without the use of a display such as when generating movies.

Example 1:
```
import visit
visit.AddArgument("-geometry")
visit.AddArgument("1024x1024")
visit.LaunchNowin()
```

Example 2:
```
import visit
visit.AddArgument("-nowin")
visit.Launch()
```

LaunchNowin: launches VisIt's viewer

Synopsis:
```
Launch() -> integer
Launch(program) -> integer
LaunchNowin() -> integer
LaunchNowin(program) -> integer
```

Arguments:
`program` The complete path to the top level 'visit' script.

Returns:
The Launch functions return 1 for success and 0 for failure

Description:
The Launch function is used to launch VisIt's viewer when the VisIt module is imported into a stand−alone Python interpreter. The Launch function has no effect when a viewer already exists. The difference between Launch and LaunchNowin is that LaunchNowin prevents the viewer from ever creating onscreen visualization windows. The LaunchNowin function is primarily used in Python scripts that want to generate visualizations using VisIt without the use of a display such as when generating movies.

Example 1:
```
import visit
visit.AddArgument("-geometry")
visit.AddArgument("1024x1024")
visit.LaunchNowin()
```

Example 2:
```
import visit
visit.AddArgument("-nowin")
visit.Launch()
```

Lineout: performs a lineout.

Synopsis:
```
Lineout(start, end) -> integer
Lineout(start, end, variables) -> integer
Lineout(start, end, samples) -> integer
Lineout(start, end, variables, samples) -> integer
```

Arguments:

`start`	A 2 or 3 item tuple containing the coordinates of the starting point.
`end`	A 2 or 3 item tuple containing the coordinates of the end point.
`variables`	A tuple of strings containing the names of the variables for whichlineouts should be created.
`samples`	An integer value containing the number of sample points along the lineout.

Returns:
The Lineout function returns 1 on success and 0 on failure.

Description:
The Lineout function extracts data along a given line segment and creates curves from it in a new visualization window. The start argument is a tuple of numbers that make up the coordinate of the lineout's starting location. The end argument is a tuple of numbers that make up the coordinate of the lineout's ending location. The optional variables argument is a tuple of strings that contain the variables that should be sampled to create lineouts. The optional samples argument is used to determine the number of sample points that should be taken along the specified line. If the samples argument is not provided then VisIt will sample the mesh where it intersects the specified line instead of using the number of samples to compute a list of points to sample.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/rect2d.silo")
AddPlot("Pseudocolor", "ascii")
DrawPlots()
Lineout((0.2,0.2), (0.8,1.2))
Lineout((0.2,1.2), (0.8,0.2), ("default", "d", "u"))
Lineout((0.6, 0.1), (0.6, 1.2), 100)
```

ListDomains: lists the members of a SIL restriction category.

Synopsis:
```
ListDomains()
ListMaterials()
```

Returns:
The List functions do not return a value.

Description:
The List functions: ListDomains, and List Materials prints a list of the domains and the materials for the active plots, which indicates which domains or materials are on and off. The list functions are used mostly to print the results of restricting the SIL.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
TurnMaterialsOff("4") # Turn off material 4
ListMaterials() # List the materials in the SIL restriction
```

ListMaterials: lists the members of a SIL restriction category.

Synopsis:
```
ListDomains()
ListMaterials()
```

Returns:
The List functions do not return a value.

Description:
The List functions: ListDomains, and List Materials prints a list of the domains and the materials for the active plots, which indicates which domains or materials are on and off. The list functions are used mostly to print the results of restricting the SIL.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
TurnMaterialsOff("4") # Turn off material 4
ListMaterials() # List the materials in the SIL restriction
```

ListPlots: lists the plots in the active visualization window's plot list.

Synopsis:
```
ListPlots() -> string
ListPlots(stringOnly) -> string
```

Returns:
The ListPlots function returns a string containing a representation of the. plot list.

Description:
Sometimes it is difficult to remember the order of the plots in the active visualization window's plot list. The ListPlots function prints the contents of the plot list to the output console and returns that string as well.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/curv2d.silo")
AddPlot("Pseudocolor", "u")
AddPlot("Contour", "d")
DrawPlots()
ListPlots()
```

LoadAttribute: loads or saves a single attribute to/from an XML file.

Synopsis:
```
LoadAttribute(filename, object)
SaveAttribute(filename, object)
```

Returns:
success or failure

Description:
The LoadAttribute and SaveAttribute methods save a single attribute, such as a current plot or operator python object, to a standalone XML file. Note that LoadAttribute requires that the target attribute already be created by other means; it fills, but does not create, the attribute.

Example:
```
% visit -cli
a = MeshPlotAttributes()
SaveAttribute('mesh.xml', a)
b = MeshPlotAttributes()
LoadAttribute('mesh.xml', b)
```

LoadNamedSelection: loads a named selection from a file.

Synopsis:
```
LoadNamedSelection(name) -> integer
LoadNamedSelection(name, engineName) -> integer
LoadNamedSelection(name, engineName, simName) -> integer
```

Arguments:

`name`	The name of a named selection.
`engineName`	(optional) The name of the engine where the selection was saved.
`simName`	(optional) The name of the simulation that saved the selection.

Returns:
The LoadNamedSelection function returns 1 for success and 0 for failure.

Description:
Named Selections allow you to select a group of elements (or particles). One typically creates a named selection from a group of elements and then later applies the named selection to another plot (thus reducing the set of elements displayed to the ones from when the named selection was created). Named selections only last for the current session. However, if you find a named selection that is particularly interesting, you can save it to a file for use in later sessions. You would use LoadNamedSelection to do the loading.

Example:
```
% visit -cli
db = "/usr/gapps/visit/data/wave*.silo database"
OpenDatabase(db)
AddPlot("Pseudocolor", "pressure")
LoadNamedSelection("selection_from_previous_session")
ApplyNamedSelection("selection_from_previous_session")
```

LoadUltra: loads the Ultra command parser.

Synopsis:
`LoadUltra()`

Arguments:
`none`

Returns:
LoadUltra does not return a value.

Description:
LoadUltra launches the Ultra command parser, allowing you to enter Ultra commands and have VisIt process them. A new command prompt is presented, and only Ultra commands will be allowed until 'end' or 'quit' is entered, at which time, you will be returned to VisIt's cli prompt. For information on currently supported commands, type 'help' at the Ultra prompt Please note that filenames/paths must be surrounded by quotes, unlike with Ultra.

Example:
```
% visit -cli
>>> LoadUltra()
U-> rd "../../data/distribution.ultra"
U-> select 1
U-> end
>>>
```

LocalNameSpace: tells the VisIt module to import plugins into the global namespace.

Synopsis:
`LocalNamespace()`

Arguments:
`none`

Returns:
The LocalNamespace function does not return a value.

Description:
The LocalNamespace function tells the VisIt module to add plugin functions to the global namespace when the VisIt module is imported into a stand−alone Python interpreter. This is the default behavior when using VisIt's cli program.

Example:
```
import visit
visit.LocalNamespace()
visit.Launch()
```

LongFileName: returns the long filename for a short WIN32 filename.

Synopsis:
`LongFileName(filename) -> string`

Arguments:
`filename` A string object containing the short filename to expand.

Returns:
The LongFileName function returns a string. Notes: This function returns the input argument unless you are on the Windows platform.

Description:
On Windows, filenames can have two different sizes: traditional 8.3 format, and long format. The long format, which lets you name files whatever you want, is implemented using the traditional 8.3 format under the covers. Sometimes filenames are given to VisIt in the traditional 8.3 format and must be expanded to long format before it is possible to open them. If you ever find that you need to do this conversion, such as when you process command line arguments, then you can use the LongFileName function to return the longer filename.

MoveAndResizeWindow: moves and resizes a vis window.

Synopsis:
`MoveAndResizeWindow(win, x, y, w, h) -> integer`

Arguments:

`win`	The id of the window to be moved [1..16].
`x`	The new x location for the window being moved.
`y`	The new y location for the window being moved.
`w`	The new width for the window being moved.
`h`	The new height for the window being moved.

Returns:
MoveAndResizeWindow returns 1 on success and 0 on failure.

Description:
MoveAndResizeWindow moves and resizes a visualization window.

Example:
```
% visit -cli
MoveAndResizeWindow(1, 100, 100, 300, 600)
```

MovePlotDatabaseKeyframe: moves a database keyframe for a plot.

Synopsis:
`MovePlotDatabaseKeyframe(index, oldFrame, newFrame)`

Arguments:

`index`	An integer representing the index of the plof in the plot list.
`oldFrame`	The old animation frame where the keyframe is located.
`newFrame`	The new animation frame where the keyframe will be moved.

Returns:
MovePlotDatabaseKeyframe does not return a value.

Description:
MovePlotDatabaseKeyframe moves a database keyframe for a specified plot to a new animation frame, which changes the list of database time states that are used for each animation frame when VisIt is in keyframing mode.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/wave.visit")
k = GetKeyframeAttributes()
nFrames = 20
k.enabled, k.nFrames, k.nFramesWasUserSet = 1, nFrames, 1
AddPlot("Pseudocolor", "pressure")
SetPlotFrameRange(0, 0, nFrames-1)
SetPlotDatabaseKeyframe(0, 0, 70)
SetPlotDatabaseKeyframe(0, nFrames/2, 35)
SetPlotDatabaseKeyframe(0, nFrames-1, 0)
DrawPlots()
for state in list(range(TimeSliderGetNStates())) + [0]:
SetTimeSliderState(state)
MovePlotDatabaseKeyframe(0, nFrames/2, nFrames/4)
for state in list(range(TimeSliderGetNStates())) + [0]:
SetTimeSliderState(state)
```

MovePlotKeyframe: moves a keyframe for a plot.

Synopsis:
`MovePlotKeyframe(index, oldFrame, newFrame)`

Arguments:

`index`	An integer representing the index of the plof in the plot list.
`oldFrame`	The old animation frame where the keyframe is located.
`newFrame`	The new animation frame where the keyframe will be moved.

Returns:
MovePlotKeyframe does not return a value.

Description:
MovePlotKeyframe moves a keyframe for a specified plot to a new animation frame, which changes the plot attributes that are used for each animation frame when VisIt is in keyframing mode.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Contour", "hgslice")
DrawPlots()
k = GetKeyframeAttributes()
nFrames = 20
k.enabled, k.nFrames, k.nFramesWasUserSet = 1, nFrames, 1
SetKeyframeAttributes(k)
SetPlotFrameRange(0, 0, nFrames-1)
c = ContourAttributes()
c.contourNLevels = 5
SetPlotOptions(c)
SetTimeSliderState(nFrames/2)
c.contourNLevels = 10
SetPlotOptions(c)
c.contourLevels = 25
SetTimeSliderState(nFrames-1)
SetPlotOptions(c)
for state in range(TimeSliderGetNStates()):
SetTimeSliderState(state)
SaveWindow()
temp = nFrames-2
MovePlotKeyframe(0, nFrames/2, temp)
MovePlotKeyframe(0, nFrames-1, nFrames/2)
MovePlotKeyframe(0, temp, nFrames-1)
for state in range(TimeSliderGetNStates()):
SetTimeSliderState(state)
SaveWindow()
```

MovePlotOrderTowardFirst: movethe i'th plot towards the start of the plot list.

Synopsis:
`MovePlotOrderTowardFirst(index) -> integer`

Arguments:

`index`	The index of the plot that will be moved within the plot list.

Returns:
The MovePlotOrderTowardFirst function returns 1 on success and 0 on failure.

Description:
This function shifts the specified plot one slot towards the start of the plot list.

Example:
`MovePlotOrderTowardFirst(2)`

MovePlotOrderTowardLast: move the i'th plot towards the end of the plot list.

Synopsis:
`MovePlotOrderTowardLast(index) -> integer`

Arguments:

`index`	The index of the plot that will be moved within the plot list.

Returns:
The MovePlotOrderTowardLast function returns 1 on success and 0 on failure.

Description:
This function shifts the specified plot one slot towards the end of the plot list.

Example:
```
MovePlotOrderTowardLast(0)
```

MoveViewKeyframe: moves a view keyframe.

Synopsis:
`MoveViewKeyframe(oldFrame, newFrame) -> integer`

Arguments:

`oldFrame`	The old animation frame where the keyframe is located.
`newFrame`	The new animation frame where the keyframe will be moved.

Returns:
MoveViewKeyframe returns 1 on success and 0 on failure.

Description:
MoveViewKeyframe moves a view keyframe to a new animation frame, which changes the view that is used for each animation frame when VisIt is in keyframing mode.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Contour", "hardyglobal")
DrawPlots()
k = GetKeyframeAttributes()
nFrames = 20
k.enabled, k.nFrames, k.nFramesWasUserSet = 1, nFrames, 1
SetKeyframeAttributes(k)
SetViewKeyframe()
SetTimeSliderState(nFrames/2)
v = GetView3d()
v.viewNormal = (-0.616518, 0.676972, 0.402014)
v.viewUp = (0.49808, 0.730785, -0.466764)
SetViewKeyframe()
SetTimeSliderState(0)
# Move the view keyframe to the last animation frame.
MoveViewKeyframe(nFrames/2, nFrames-1)
```

MoveWindow: moves a vis window.

Synopsis:
`MoveWindow(win, x, y) -> integer`

Arguments:

`win`	The id of the window to be moved [1..16].
`x`	The new x location for the window being moved.
`y`	The new y location for the window being moved.

Returns:
MoveWindow returns 1 on success and 0 on failure.

Description:
MoveWindow moves a visualization window.

Example:
```
% visit -cli
MoveWindow(1, 100, 100)
```

NodePick: performs a nodal pick on a plot.

Synopsis:
```
NodePick(point) -> integer
NodePick(point, variables) -> integer
NodePick(sx, sy) -> integer
NodePick(sx, sy, variables) -> integer
```

Arguments:

`point`	A tuple of values that describe the coordinate of where we want to performthe nodal pick.
`variables`	An optional tuple of strings containing the names of the variables forwhich we want information. The tuple can contain the name "default" ifyou want information for the plotted variable.
`sx`	A screen X location (in pixels) offset from the left side of thevisualization window.
`sy`	A screen Y location (in pixels) offset from the bottom of thevisualization window.

Returns:
The NodePick function prints pick information for the node closest to the specified point. The point can be specified as a 2D or 3D point in world space or it can be specified as a pixel location in screen space. If the point is specified as a pixel location then VisIt finds the node closest to a ray that is projected into the mesh. Once the nodal pick has been calculated, you can use the GetPickOutput function to retrieve the printed pick output as a string which can be used for other purposes.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Pseudocolor", "hgslice")
DrawPlots()
# Perform node pick in screen space
NodePick(300,300)
# Perform node pick in world space.
NodePick((-5.0, 5.0))
```

NumColorTableNames: returns the number of color tables that have been defined.

Synopsis:
`NumColorTableNames() -> integer`

Returns:
The NumColorTableNames function return an integer.

Description:
The NumColorTableNames function returns the number of color tables that have been defined.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
p = PseudocolorAttributes()
p.colorTableName = "default"
SetPlotOptions(p)
DrawPlots()
print "There are %d color tables." % NumColorTableNames()
for ct in ColorTableNames():
SetActiveContinuousColorTable(ct)
SaveWindow()
```

NumOperatorPlugins: returns the number of available operator plugins.

Synopsis:
`NumOperatorPlugins() -> integer`

Returns:
The NumOperatorPlugins function returns an integer.

Description:
The NumOperatorPlugins function returns the number of available operator plugins.

Example:
```
% visit -cli
print "The number of operator plugins is: ", NumOperatorPlugins()
print "The names of the plugins are: ", OperatorPlugins()
```

NumPlotPlugins: returns the number of available plot plugins.

Synopsis:
`NumPlotPlugins() -> integer`

Returns:
The NumPlotPlugins function returns an integer.

Description:
The NumPlotPlugins function returns the number of available plot plugins.

Example:
```
% visit -cli
print "The number of plot plugins is: ", NumPlotPlugins()
print "The names of the plugins are: ", PlotPlugins()
```

OpenComputeEngine: opens a compute engine on the specified computer

Synopsis:
```
OpenComputeEngine() -> integer
OpenComputeEngine(hostName) -> integer
OpenComputeEngine(hostName, simulation) -> integer
OpenComputeEngine(hostName, args) -> integer
OpenComputeEngine(MachineProfile) -> integer
```

Arguments:

`hostName`	The name of the computer on which to start the engine.
`args`	Optional tuple of command line arguments for the engine.
`Alternative arguments:`	MachineProfile object to load with OpenComputeEngine call

Returns:
The OpenComputeEngine function returns an integer value of 1 for success and 0 for failure.

Description:
The OpenComputeEngine function is used to explicitly open a compute engine with certain properties. When a compute engine is opened implicitly, the viewer relies on sets of attributes called host profiles. Host profiles determine how compute engines are launched. This allows compute engines to be easily launched in parallel. Since the VisIt Python Interface does not expose VisIt's host profiles, it provides the OpenComputeEngine function to allow users to launch compute engines. The OpenComputeEngine function must be called before opening a database in order to prevent any latent host profiles from taking precedence.

Example:
```
% visit -cli
# Launch parallel compute engine remotely.
args = ("-np", "16", "-nn", "4")
OpenComputeEngine("thunder", args)
OpenDatabase("thunder:/usr/gapps/visit/data/multi_ucd3d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
```

OpenDatabase: opens a database for plotting.

Synopsis:
```
OpenDatabase(databaseName) -> integer
OpenDatabase(databaseName, timeIndex) -> integer
OpenDatabase(databaseName, timeIndex, dbPluginName) -> integer
```

Arguments:

`databaseName`	A string containing the name of the database to open.
`timeIndex`	This is an optional integer argument indicating the time index at which toopen the database. If it is not specified, a time index of zero is assumed.
`dbPluginIndex`	An optional string containing the name of the plugin to use. Note that thisstring must also include the plugin's version number (with few exceptions,almost all plugins' version numbers are 1.0). Note also that you mustcapitalize the spelling identically to what the plugin'sGeneralPluginInfo::GetName() method returns. For example, "XYZ_1.0"is the string you would use for the XYZ plugin.

Returns:
The OpenDatabase function returns an integer value of 1 for success and 0 for failure.

Description:
The OpenDatabase function is one of the most important functions in the VisIt Python Interface because it opens a database so it can be plotted. The databaseName argument is a string containing the full name of the database to be opened. The database name is of the form: computer:/path/filename. The computer part of the filename can be omitted if the database to be opened resides on the local computer.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
OpenDatabase("mcr:/usr/gapps/visit/data/multi_ucd3d.silo")
OpenDatabase("file.visit")
OpenDatabase("file.visit", 4)
OpenDatabase("mcr:/usr/gapps/visit/data/multi_ucd3d.silo",0,"Silo_1.0")
```

OpenMDServer: explicitly opens a metadata server.

Synopsis:
```
OpenMDServer() -> integer
OpenMDServer(host) -> integer
OpenMDServer(host, args) -> integer
OpenMDServer(MachineProfile) -> integer
```

Arguments:

`host`	The optional host argument determines the host on which the metadataserver is to be launched. If this argument is not provided, "localhost"is assumed.
`args`	A tuple of strings containing command line flags for the metadata server.
`Alternative arguments:`	MachineProfile object to load with OpenMDServer call

Returns:
The OpenMDServer function returns 1 on success and 0 on failure.

Description:
The OpenMDServer explicitly launches a metadata server on a specified host. This allows you to provide command line options that influence how the metadata server will run.

Argument	Description
`-debug #`	The −debug argument allows you to specify a debug level in therange [1,5] that VisIt uses to write debug logs to disk.
`-dir visitdir`	The −dir argument allows you to specify where VisIt islocated on a remote computer. This allows you to successfully connect to a remote computer in the absence of host profiles. It also allows you to debug VisIt in distributed mode.
`-fallback_format <format>`	The −fallback_format argument allows you to specify thedatabase plugin that will be used to open files if allother guessing failed. This is useful when the filesthat you want to open do not have file extensions.
`-assume_format <format>`	The −assume_format argument allows you to specify thedatabase plugin that will be used FIRST when attemptingto open files. This is useful when the files that you want to open have a file extension which may matchmultiple file format readers.

Example:
```
-assume_format PDB
```

Example:
```
% visit -cli
args = ("-dir", "/my/private/visit/version/", "-assume_format", \
"PDB", "-debug", "4")
# Open a metadata server before the call to OpenDatabase so we
# can launch it how we want.
OpenMDServer("thunder", args)
OpenDatabase("thunder:/usr/gapps/visit/data/allinone00.pdb")
# Open a metadata server on localhost too.
OpenMDServer()
```

OperatorPlugins: returns a tuple of operator plugin names.

Synopsis:
```
OperatorPlugins() -> tuple of strings
```

Returns:
The OperatorPlugins function returns a tuple of strings.

Description:
The OperatorPlugins function returns a tuple of strings that contain the names of the loaded operator plugins. This can be useful for the creation of scripts that alter their behavior based on the available operator plugins.

Example:
```
% visit -cli
for plugin in OperatorPlugins():
print "The %s operator plugin is loaded." % plugin
```

OverlayDatabase: creates new plots based on current plots but with a new database.

Synopsis:
```
OverlayDatabase(databaseName) -> integer
OverlayDatabase(databaseName, state) -> integer
```

Arguments:

`databaseName`	A string containing the name of the new plot database.
`state`	The time state at which to open the database.

Returns:
The OverlayDatabase function returns an integer value of 1 for success and 0 for failure.

Description:
VisIt has the concept of overlaying plots which, in the nutshell, means that the entire plot list is copied and a new set of plots with exactly the same attributes but a different database is appended to the plot list of the active window. The OverlayDatabase function allows the VisIt Python Interface to overlay plots. OverlayDatabase takes a single string argument which contains the name of the database. After calling the OverlayDatabase function, the plot list is larger and contains plots of the specified overlay database.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
OverlayDatabase("riptide:/usr/gapps/visit/data/curv3d.silo")
```

Pick: performs a zonal pick on a plot.

Synopsis:
```
ZonePick(point) -> integer
ZonePick(point, variables) -> integer
ZonePick(sx, sy) -> integer
ZonePick(sx, sy, variables) -> integer
```

Arguments:

`point`	A tuple of values that describe the coordinate of where we want to performthe zonal pick.
`variables`	An optional tuple of strings containing the names of the variables forwhich we want information. The tuple can contain the name "default" ifyou want information for the plotted variable.
`sx`	A screen X location (in pixels) offset from the left side of thevisualization window.
`sy`	A screen Y location (in pixels) offset from the bottom of thevisualization window.

Returns:
The ZonePick function prints pick information for the cell (a.k.a zone) that contains the specified point. The point can be specified as a 2D or 3D point in world space or it can be specified as a pixel location in screen space. If the point is specified as a pixel location then VisIt finds the zone that contains the intersection of a cell and a ray that is projected into the mesh. Once the zonal pick has been calculated, you can use the GetPickOutput function to retrieve the printed pick output as a string which can be used for other purposes.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Pseudocolor", "hgslice")
DrawPlots()
# Perform zone pick in screen space
ZonePick(300,300)
# Perform zone pick in world space.
ZonePick((-5.0, 5.0))
```

PickByGlobalNode: performs pick operation for a specific node using a global node id.

Synopsis:
```
PickByGlobalNode(node) -> integer
PickByGlobalNode(node, variables) -> integer
```

Arguments:

`node`	Integer index of the global node for which we want pick information.
`variables`	A tuple of strings containing the names of the variables for which we wantpick information.

Returns:
PickByGlobalNode returns 1 on success and 0 on failure.

Description:
The PickByGlobalNode function tells VisIt to perform pick using a specific global node index for the entire problem. Some meshes are broken up into smaller "domains" and then these smaller domains can employ a global indexing scheme to make it appear as though the mesh was still one large mesh. Not all meshes that have been decomposed into domains provide sufficient information to allow global node indexing. You can use the GetPickOutput function to retrieve a string containing the pick information once you've called PickByGlobalNode.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/multi_curv2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
# Pick on node 200 in the first domain.
PickByGlobalNode(200)
print "Last pick = ", GetPickOutput()
```

PickByGlobalZone: performs pick operation for a specific cell using a global cell id.

Synopsis:
```
PickByGlobalZone(id) -> integer
PickByGlobalZone(id, variables) -> integer
```

Arguments:

`id`	Integer index of the global cell for which we want pick information.
`variables`	A tuple of strings containing the names of the variables for which we wantpick information.

Returns:
PickByGlobalZone returns 1 on success and 0 on failure.

Description:
The PickByGlobalZone function tells VisIt to perform pick using a specific global cell index for the entire problem. Some meshes are broken up into smaller "domains" and then these smaller domains can employ a global indexing scheme to make it appear as though the mesh was still one large mesh. Not all meshes that have been decomposed into domains provide sufficient information to allow global cell indexing. You can use the GetPickOutput function to retrieve a string containing the pick information once you've called PickByGlobalZone.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/multi_curv2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
# Pick on cell 200 in the first domain.
PickByGlobalZone(200)
print "Last pick = ", GetPickOutput()
```

PickByNode: performs pick operation for a specific node in a given domain.

Synopsis:
```
PickByNode(node) -> integer
PickByNode(node, variables) -> integer
PickByNode(node, domain) -> integer
PickByNode(node, domain, variables) -> integer
```

Arguments:

`node`	Integer index of the node for which we want pick information.
`domain`	An integer representing the index of the domain that contains the node forwhich we want pick information. Note that if the first domain is"domain1" then the first valid index is 1.
`variables`	A tuple of strings containing the names of the variables for which we wantpick information.

Returns:
PickByNode returns 1 on success and 0 on failure.

Description:
The PickByNode function tells VisIt to perform pick using a specific node index in a given domain. Other pick by node variants first determine the node that is closest to some user−specified 3D point but the PickByNode functions cuts out this step and allows you to directly pick on the node of your choice. You can use the GetPickOutput function to retrieve a string containing the pick information once you've called PickByNode.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/multi_curv2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
# Pick on node 200 in the first domain.
PickByNode(200)
# Pick on node 200 in the second domain.
PickByNode(200, 2)
# Pick on node 100 in domain 5 and return information for two additional
variables.
PickByNode(100, 5, ("default", "u", "v"))
print "Last pick = ", GetPickOutput()
```

PickByZone: performs pick operation for a specific cell in a given domain.

Synopsis:
```
PickByZone(cell) -> integer
PickByZone(cell, variables) -> integer
PickByZone(cell, domain) -> integer
PickByZone(cell, domain, variables) -> integer
```

Arguments:

`cell`	Integer index of the cell for which we want pick information.
`domain`	An integer representing the index of the domain that contains the cell forwhich we want pick information. Note that if the first domain is"domain1" then the first valid index is 1.
`variables`	A tuple of strings containing the names of the variables for which we wantpick information.

Returns:
PickByZone returns 1 on success and 0 on failure.

Description:
The PickByZone function tells VisIt to perform pick using a specific cell index in a given domain. Other pick by zone variants first determine the cell that contains some user−specified 3D point but the PickByZone functions cuts out this step and allows you to directly pick on the cell of your choice. You can use the GetPickOutput function to retrieve a string containing the pick information once you've called PickByZone.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/multi_curv2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
# Pick on cell 200 in the first domain.
PickByZone(200)
# Pick on cell 200 in the second domain.
PickByZone(200, 2)
# Pick on cell 100 in domain 5 and return information for two additional
variables.
PickByZone(100, 5, ("default", "u", "v"))
print "Last pick = ", GetPickOutput()
```

PlotPlugins: returns a tuple of plot plugin names.

Synopsis:
```
PlotPlugins() -> tuple of strings
```

Returns:
The PlotPlugins function returns a tuple of strings.

Description:
The PlotPlugins function returns a tuple of strings that contain the names of the loaded plot plugins. This can be useful for the creation of scripts that alter their behavior based on the available plot plugins.

Example:
```
% visit -cli
for plugin in PluginPlugins():
print "The %s plot plugin is loaded." % plugin
```

PointPick: performs a nodal pick on a plot.

Synopsis:
```
NodePick(point) -> integer
NodePick(point, variables) -> integer
NodePick(sx, sy) -> integer
NodePick(sx, sy, variables) -> integer
```

Arguments:

`point`	A tuple of values that describe the coordinate of where we want to performthe nodal pick.
`variables`	An optional tuple of strings containing the names of the variables forwhich we want information. The tuple can contain the name "default" ifyou want information for the plotted variable.
`sx`	A screen X location (in pixels) offset from the left side of thevisualization window.
`sy`	A screen Y location (in pixels) offset from the bottom of thevisualization window.

Returns:
The NodePick function prints pick information for the node closest to the specified point. The point can be specified as a 2D or 3D point in world space or it can be specified as a pixel location in screen space. If the point is specified as a pixel location then VisIt finds the node closest to a ray that is projected into the mesh. Once the nodal pick has been calculated, you can use the GetPickOutput function to retrieve the printed pick output as a string which can be used for other purposes.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Pseudocolor", "hgslice")
DrawPlots()
# Perform node pick in screen space
NodePick(300,300)
# Perform node pick in world space.
NodePick((-5.0, 5.0))
```

PrintWindow: prints the active visualization window.

Synopsis:
```
PrintWindow() -> integer
```

Returns:
The PrintWindow function returns an integer value of 1 for success and 0 for failure.

Description:
The PrintWindow function tells the viewer to print the image in the active visualization window using the current printer settings.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/curv2d.silo")
AddPlot("Pseudocolor", "d")
AddPlot("Contour", "u")
DrawPlots()
PrintWindow()
```

PromoteOperator: moves an operator closer to the end of the visualization pipeline.

Synopsis:
```
PromoteOperator(opIndex) -> integer
PromoteOperator(opIndex, applyToAllPlots) -> integer
```

Arguments:

`opIndex`	A zero−based integer corresponding to the operator that should be promoted.
`applyAll`	An integer flag that causes all plots in the plot list to be affected whenit is non−zero.

Returns:
PromoteOperator returns 1 on success and 0 on failure.

Description:
The PromoteOperator function moves an operator closer to the end of the visualization pipeline. This allows you to change the order of operators that have been applied to a plot without having to remove them from the plot. For example, consider moving a Slice to after a Reflect operator when it had been the other way around. Changing the order of operators can result in vastly different results for a plot. The opposite function is DemoteOperator.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Pseudocolor", "hardyglobal")
AddOperator("Slice")
s = SliceAttributes()
s.project2d = 0
s.originPoint = (0,5,0)
s.originType=s.Point
s.normal = (0,1,0)
s.upAxis = (-1,0,0)
SetOperatorOptions(s)
AddOperator("Reflect")
DrawPlots()
# Now slice after reflect. We'll only get 1 slice plane instead of 2.
PromoteOperator(0)
DrawPlots()
```

PythonQuery: executes a Python Filter Query.

Synopsis:
```
PythonQuery(source='python filter source ...') -> integer
PythonQuery(file='path/to/python_filter_script.py') -> integer
```

Arguments:

`source`	A string containing the source code for a Python Query Filter .
`file`	A string containing the path to a Python Query Filter script file.Note: Use only one of the 'source' or 'file' arguments.If both are used the 'source' argument overrides 'file'.

Returns:
The PythonQuery function returns 1 on success and 0 on failure.

Description:
Used to execute a Python Filter Query.

Queries: returns a tuple containing the names of all supported queries.

Synopsis:
```
Queries() -> tuple of strings
```

Returns:
The Queries function returns a tuple of strings.

Description:
The Queries function returns a tuple of strings that contain the names of all of VisIt's supported queries.

Example:
```
% visit -cli
print "supported queries: ", Queries()
```

QueriesOverTime: returns a tuple of containing the names of all supported queries that can

Synopsis:
`QueriesOverTime() -> tuple of strings`

Returns:
Returns a tuple of strings.

Description:
The QueriesOverTime function returns a tuple of strings that contains the names of all of the VisIt queries that can be executed over time.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/allineone00.pdb")
AddPlot("Pseudocolor", "mesh/mixvar")
DrawPlots()
# Execute each of the queries over time on the plots.
for q in QueriesOverTime():
QueryOverTime(q)
You can control timestates used in the query via start_time,
end_time, and stride as follows:
QueryOverTime("Volume", start_time=5, end_time=250, stride=5)
(Defaults used if not specified are 0, nStates, 1)
```

Query: executes one of VisIt's queries.

Synopsis:
```
Query(name) -> integer
Query(name, dict) -> integer
Query(name, namedarg1=arg1,namedarg2=arg2, ...) -> integer
```

Arguments:

`name`	A string containing the name of the query to execute.
`dict`	An optional dictionary containing additional query arguments.
`namedarg1, namedarg2,...`	An optional list of named arguments supplying additional query parameters.

Returns:
The Query function returns 1 on success and 0 on failure.

Description:
The Query function is used to execute any of VisIt's predefined queries. The list of queries can be found in theVisIt User's Manual in the Quantitative Analysis chapter. You can get also get a list of queries using 'Queries' function. Since queries can take a wide array of arguments, the Query function takes either a python dictorary or a list of named arguments specific to the given query. To obtain the possible options for a given query, use the GetQueryParameters(name) function. If the query accepts additional arguments beyond its name, this function will return a python dictionary containing the needed variables and their default values. This can be modified and passed back to the Query method, or named arguments can be used instead.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/wave.visit")
AddPlot("Pseudocolor", "pressure")
DrawPlots()
Query("Volume")
Query("MinMax")
Query("MinMax", use_actual_data=1)
hohlraumArgs = GetQueryParameters("Hohlraum Flux")
hohlraumArgs["ray_center"]=(0.5,0.5,0)
hohlraumArgs["vars"]=("a1", "e1")
Query("Hohlraum Flux", hohlraumArgs)
```

QueryOverTime: executes one of VisIt's queries over time to produce a curve.

Synopsis:
```
QueryOverTime(name) -> integer
QueryOverTime(name, dict) -> integer
QueryOverTime(name, namedarg1=val1,namedarg2=val2,\
     ...) -> integer
```

Arguments:

`name`	A string containing the name of the query to execute.
`dict`	An optional dictionary containing additional query arguments.
`namedarg1, namedarg2,...`	An optional list of named arguments supplying additional query parameters.

Returns:
The QueryOverTime function returns 1 on success and 0 on failure.

Description:
The QueryOverTime function is used to execute any of VisIt's predefined queries. The list of queries can be found in the VisIt User's Manual in the Quantitative Analysis chapter. You can get also get a list of queries that can be executed over time using 'QueriesOverTime' function. Since queries can take a wide array of arguments, the Query function takes either a python dictorary or a list of named arguments specific to the given query. To obtain the possible options for a given query, use the GetQueryParameters(name) function. If the query accepts additional arguments beyond its name, this function will return a python dictionary containing the needed variables and their default values. This can be modified and passed back to the QueryOverTime method, or named arguments can be used instead.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/wave.visit")
AddPlot("Pseudocolor", "pressure")
DrawPlots()
for q in QueriesOverTime():
    QueryOverTime(q)
ResetView()
```

ReOpenDatabase: reopens a database for plotting.

Synopsis:
`ReOpenDatabase(databaseName) -> integer`

Arguments:
`databaseName` A string containing the name of the database to open.

Returns:
The ReOpenDatabase function returns an integer value of 1 for success and 0 for failure.

Description:
The ReOpenDatabase function reopens a database that has been opened previously with the OpenDatabase function. The ReOpenDatabase function is primarily used for regenerating plots whose database has been rewritten on disk. ReOpenDatabase allows VisIt to access new variables and new time states that have been added since the database was opened using the OpenDatabase function. Note that ReOpenDatabase is expensive since it causes all plots that use the specified database to be regenerated. If you want to ensure that a time−varying database has all of its time states as they are being created by a simulation, try the CheckForNewStates function instead. The databaseName argument is a string containing the full name of the database to be opened. The database name is of the form: host:/path/filename. The host part of the filename can be omitted if the database to be reopened resides on the local computer.

Example:
```
% visit -cli
OpenDatabase("edge:/usr/gapps/visit/data/wave*.silo database")
AddPlot("Pseudocolor", "pressure")
DrawPlots()
last = TimeSliderGetNStates()
for state in range(last):
SetTimeSliderState(state)
SaveWindow()
ReOpenDatabase("edge:/usr/gapps/visit/data/wave*.silo database")
for state in range(last, TimeSliderGetNStates()):
SetTimeSliderState(state)
SaveWindow()
```

RecenterView: recalculates the view for the active visualization window so that its

Synopsis:
`RecenterView() -> integer`

Returns:
The RecenterView function returns 1 on success and 0 on failure.

Description:
After adding plots to a visualization window or applying operators to those plots, it is sometimes necessary to recenter the view. When the view is recentered, the orientation does not change but the view is shifted to make better use of the screen.

Example:
```
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
OpenDatabase("/usr/gapps/visit/data/curv3d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
RecenterView()
```

RedoView: restores a view that has been undone with UndoView.

Synopsis:
`RedoView() -> integer`

Returns:
The RedoView function returns 1 on success and 0 on failure.

Description:
When the view changes in the visualization window, it puts the old view on a stack of views. VisIt provides the UndoView function that lets you undo view changes. The RedoView function re−applies any views that have been undone by the UndoView function.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/curv2d.silo")
AddPlot("Subset", "mat1")
DrawPlots()
v = GetView2D()
v.windowCoords = (-2.3,2.4,0.2,4.9)
SetView2D(v)
UndoView()
RedoView()
```

RedrawWindow: enables redraws and forces the active visualization window to redraw.

Synopsis:
`RedrawWindow() -> integer`

Returns:
The RedrawWindow function returns 1 on success and 0 on failure.

Description:
The RedrawWindow function allows a visualization window to redraw itself and then forces the window to redraw. This function does the opposite of the DisableRedraw function and is used to recover from it.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Contour", "u")
AddPlot("Pseudocolor", "w")
DrawPlots()
DisableRedraw()
AddOperator("Slice")
# Set the slice operator attributes
# Redraw now that the operator attributes are set. This will
# prevent 1 redraw.
RedrawWindow()
```

RegisterCallback: register a user-defined callback function that gets called when a state object changes or when an rpc is called.

Synopsis:
`RegisterCallback(callbackname, callback) --> integer`

Arguments:

`callbackname`	A string object designating the callback that we're installing. Allowable values are returned by the GetCallbackNames() function.
`callback`	A Python function, typically with one argument by which VisIt passes the object that caused the callback to be called.

Returns:
RegisterCallback returns 1 on success.

Description:
The RegisterCallback function is used to associate a user−defined callback function with the updating of a state object or execution of a particular rpc

Example:
```
import visit
def print_sliceatts(atts):
    print "SLICEATTS=", atts
visit.RegisterCallback("SliceAttributes", print_sliceatts)
```

RegisterMacro: associates a user-defined Python function with a macro name.

Synopsis:
`RegisterMacro(name, callable)`

Arguments:

`name`	A string containing the name of the macro.
`callable`	A Python function that will be associated with the macro name.

Returns:
The RegisterMacro function does not return a value.

Description:
The RegisterMacro function lets you associate a Python function with a name so when VisIt's gui calls down into Python to execute a macro, it ends up executing the registered Python function. Macros let users define complex new behaviors using Python functions yet still call them simply by clicking a button within VisIt's gui. When a new macro function is registered, a message is sent to the gui that adds the known macros as buttons in the Macros window.

Example:
```
def SetupMyPlots():
    OpenDatabase('noise.silo')
    AddPlot('Pseudocolor', 'hardyglobal')
    DrawPlots()
RegisterMacro('Setup My Plots', SetupMyPlots)
```

RemoveAllOperators: removes operators from plots.

Synopsis:
```
RemoveAllOperators() -> integer
RemoveAllOperators(all) -> integer
RemoveLastOperator() -> integer
RemoveLastOperator(all) -> integer
RemoveOperator(index) -> integer
RemoveOperator(index, all) -> integer
```

Arguments:

`all`	An optional integer argument that tells the function to ignore theactive plots and use all plots in the plot list if the value of theargument is non−zero.
`index`	The zero−based integer index into a plot's operator list that specifieswhich operator is to be deleted.

Returns:
All functions return an integer value of 1 for success and 0 for failure.

Description:
The RemoveOperator functions allow operators to be removed from plots. The RemoveLastOperator function removes the operator that was last applied to the active plots. The RemoveAllOperators function removes all operators from the active visualization window. If the all argument is provided and contains a non−zero value, all plots in the active visualization window are affected. If the value is zero or if the argument is not provided, only the active plots are affected.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
AddOperator("Threshold")
AddOperator("Slice")
AddOperator("SphereSlice")
DrawPlots()
RemoveLastOperator() # Remove SphereSlice
RemoveOperator(0) # Remove Threshold
RemoveAllOperators() # Remove the rest of the operators
```

RemoveLastOperator: removes operators from plots.

Synopsis:
```
RemoveAllOperators() -> integer
RemoveAllOperators(all) -> integer
RemoveLastOperator() -> integer
RemoveLastOperator(all) -> integer
RemoveOperator(index) -> integer
RemoveOperator(index, all) -> integer
```

Arguments:

`all`	An optional integer argument that tells the function to ignore theactive plots and use all plots in the plot list if the value of theargument is non−zero.
`index`	The zero−based integer index into a plot's operator list that specifieswhich operator is to be deleted.

Returns:
All functions return an integer value of 1 for success and 0 for failure.

Description:
The RemoveOperator functions allow operators to be removed from plots. The RemoveLastOperator function removes the operator that was last applied to the active plots. The RemoveAllOperators function removes all operators from the active visualization window. If the all argument is provided and contains a non−zero value, all plots in the active visualization window are affected. If the value is zero or if the argument is not provided, only the active plots are affected.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
AddOperator("Threshold")
AddOperator("Slice")
AddOperator("SphereSlice")
DrawPlots()
RemoveLastOperator() # Remove SphereSlice
RemoveOperator(0) # Remove Threshold
RemoveAllOperators() # Remove the rest of the operators
```

RemoveMachineProfile: ¡description¿

Synopsis:
`RemoveMachineProfile(hostname) -> integer`

Arguments:
`hostname`

Description:
Removes machine profile with hostname from HostProfileList

RemoveOperator: removes operators from plots.

Synopsis:
```
RemoveAllOperators() -> integer
RemoveAllOperators(all) -> integer
RemoveLastOperator() -> integer
RemoveLastOperator(all) -> integer
RemoveOperator(index) -> integer
RemoveOperator(index, all) -> integer
```

Arguments:

`all`	An optional integer argument that tells the function to ignore theactive plots and use all plots in the plot list if the value of theargument is non−zero.
`index`	The zero−based integer index into a plot's operator list that specifieswhich operator is to be deleted.

Returns:
All functions return an integer value of 1 for success and 0 for failure.

Description:
The RemoveOperator functions allow operators to be removed from plots. The RemoveLastOperator function removes the operator that was last applied to the active plots. The RemoveAllOperators function removes all operators from the active visualization window. If the all argument is provided and contains a non−zero value, all plots in the active visualization window are affected. If the value is zero or if the argument is not provided, only the active plots are affected.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
AddOperator("Threshold")
AddOperator("Slice")
AddOperator("SphereSlice")
DrawPlots()
RemoveLastOperator() # Remove SphereSlice
RemoveOperator(0) # Remove Threshold
RemoveAllOperators() # Remove the rest of the operators
```

RenamePickLabel: let the user rename the pick label to provide a more descriptive string.

Synopsis:
`RenamePickLabel(oldLabel, newLabel) -> integer`

Arguments:

`oldLabel`	The old pick label to replace. (e.g. 'A', 'B').
`newLabel`	A new label to display in place of the old label.

Returns:
The RenamePickLabel function returns 1 on success and 0 on failure.

Description:
The RenamePickLabel function can be used to replace an automatically generated pick label such as 'A' with a user−defined string.

Example:
```
RenamePickLabel('A', 'Point of interest')
```

ReplaceDatabase: replaces the database in the current plots with a new database.

Synopsis:
```
ReplaceDatabase(databaseName) -> integer
ReplaceDatabase(databaseName, timeState) -> integer
```

Arguments:

`databaseName`	A string containing the name of the new database.
`timeState`	A zero−based integer containing the time state that should be made activeonce the database has been replaced.

Returns:
The ReplaceDatabase function returns an integer value of 1 for success and 0 for failure.

Description:
The ReplaceDatabase function replaces the database in the current plots with a new database. This is one way of switching timesteps if no ".visit" file was ever created. If two databases have the same variable name then replace is usually a success. In the case where the new database does not have the desired variable, the plot with the variable not contained in the new database does not get regenerated with the new database.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo)
AddPlot("Pseudocolor", "u")
DrawPlots()
ReplaceDatabase("/usr/gapps/visit/data/curv3d.silo")
SaveWindow()
# Replace with a time-varying database and change the time
# state to 17.
ReplaceDatabase("/usr/gapps/visit/data/wave.visit", 17)
```

ResetLineoutColor: resets the color used by lineout to the first color.

Synopsis:
`ResetLineoutColor() -> integer`

Returns:
ResetLineoutColor returns 1 on success and 0 on failure.

Description:
Lineouts on VisIt cause reference lines to be drawn over the plot where the lineout was being extracted. Each reference line uses a different color in a discrete color table. Once the colors in the discrete color table are used up, the reference lines start using the color from the start of the discrete color table and so on. ResetLineoutColor forces reference lines to start using the color at the start of the discrete color table again thus resetting the lineout color.

ResetOperatorOptions: resets operator attributes back to the default values.

Synopsis:
```
ResetOperatorOptions(operatorType) -> integer
ResetOperatorOptions(operatorType, all) -> integer
```

Arguments:

`operatorType`	A string containing the name of a valid operator type.
`all`	An optional integer argument that tells the function to reset the operatoroptions for all plots regardless of whether or not they are active.

Returns:
The ResetOperatorOptions function returns an integer value of 1 for success and 0 for failure.

Description:
The ResetOperatorOptions function resets the operator attributes of the specified operator type for the active plots back to the default values. The operatorType argument is a string containing the name of the type of operator whose attributes are to be reset. The all argument is an optional flag that tells the function to reset the operator attributes for the indicated operator in all plots regardless of whether the plots are active. When non−zero values are passed for the all argument, all plots are reset. When the all argument is zero or not provided, only the operators on active plots are modified.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
AddOperator("Slice")
a = SliceAttributes()
a.normal,a.upAxis = (0,0,1),(0,1,0)
SetOperatorOptions(a)
ResetOperatorOptions("Slice")
```

ResetPickLetter: resets the pick marker letter back to "A".

Synopsis:
`ResetPickLetter() -> integer`

Returns:
ResetPickLetter returns 1 on success and 0 on failure.

Description:
The ResetPickLetter function resets the pick marker back to "A" so that the next pick will use "A" as the pick letter and then "B" and so on.

ResetPlotOptions: resets plot attributes back to the default values.

Synopsis:
`ResetPlotOptions(plotType) -> integer`

Arguments:
`plotType` A string containing the name of the plot type.

Returns:
The ResetPlotOptions function returns an integer value of 1 for success and 0 for failure.

Description:
The ResetPlotOptions function resets the plot attributes of the specified plot type for the active plots back to the default values. The plotType argument is a string containing the name of the type of plot whose attributes are to be reset.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
p = PseudocolorAttributes()
p.colorTableName = "calewhite"
p.minFlag,p.maxFlag = 1,1
p.min,p.max = -5.0, 8.0
SetPlotOptions(p)
ResetPlotOptions("Pseudocolor")
```

ResetView: resets the view to the initial view

Synopsis:
`ResetView() -> integer`

Returns:
The ResetView function returns 1 on success and 0 on failure.

Description:
The ResetView function resets the camera to the initial view.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/curv3d.silo")
AddPlot("Mesh", "curvmesh3d")
v = ViewAttributes()
v.camera = (-0.45396, 0.401908, 0.79523)
v.focus = (0, 2.5, 15)
v.viewUp = (0.109387, 0.910879, -0.397913)
v.viewAngle = 30
v.setScale = 1
v.parallelScale = 16.0078
v.nearPlane = -32.0156
v.farPlane = 32.0156
v.perspective = 1
SetView3D(v) # Set the 3D view
DrawPlots()
ResetView()
```

ResizeWindow: resizes a vis window.

Synopsis:
`ResizeWindow(win, w, h) -> integer`

Arguments:

`win`	The id of the window to be moved [1..16].
`w`	The new width for the window.
`h`	The new height for the window.

Returns:
ResizeWindow returns 1 on success and 0 on failure.

Description:
ResizeWindow resizes a visualization window.

Example:
```
% visit -cli
ResizeWindow(1, 300, 600)
```

RestoreSession: restores a VisIt session.

Synopsis:
```
RestoreSession(filename, visitDir) -> integer
RestoreSessionWithDifferentSources(filename, visitDir,\
     tuple of strings) -> integer
```

Arguments:

`filename`	The name of the session file to restore.
`visitDir`	An integer flag that indicates whether the filename to be restored islocated in the user's VisIt directory. If the flag is set to 1 then thesession file is assumed to be located in the user's VisIt directoryotherwise the filename must contain an absolute path.tuple of stringsA tuple of strings representing the maping from sources as specifiedin the original session file to new sources. Sources in the originalsession file are numbered starting from 0. So, this tuple of stringssimply contains the new names for each of the sources, in order.

Returns:
RestoreSession returns 1 on success and 0 on failure.

Description:
The RestoreSession function is important for setting up complex visualizations because you can design a VisIt session file, which is an XML file that describes exactly how plots are set up, using the VisIt GUI and then use that same session file in the CLI to generate movies in batch. The RestoreSession function takes 2 arguments. The first argument specifies the filename that contains the VisIt session to be restored. The second argument determines whether the session file is assumed to be in the user's VisIt directory. If the visitDir argument is set to 0 then the filename argument must contain the absolute path to the session file.

Example:
```
% visit -cli
# Restore my session file for a time-varying database from
# my .visit directory.
RestoreSessionFile("visit.session", 1)
for state in range(TimeSliderGetNStates()):
SetTimeSliderState(state)
SaveWindow()
```

RestoreSessionWithDifferentSources: restores a VisIt session.

Synopsis:
```
RestoreSession(filename, visitDir) -> integer
RestoreSessionWithDifferentSources(filename, visitDir,\
     tuple of strings) -> integer
```

Arguments:

`filename`	The name of the session file to restore.
`visitDir`	An integer flag that indicates whether the filename to be restored islocated in the user's VisIt directory. If the flag is set to 1 then thesession file is assumed to be located in the user's VisIt directoryotherwise the filename must contain an absolute path.tuple of stringsA tuple of strings representing the maping from sources as specifiedin the original session file to new sources. Sources in the originalsession file are numbered starting from 0. So, this tuple of stringssimply contains the new names for each of the sources, in order.

Returns:
RestoreSession returns 1 on success and 0 on failure.

Description:
The RestoreSession function is important for setting up complex visualizations because you can design a VisIt session file, which is an XML file that describes exactly how plots are set up, using the VisIt GUI and then use that same session file in the CLI to generate movies in batch. The RestoreSession function takes 2 arguments. The first argument specifies the filename that contains the VisIt session to be restored. The second argument determines whether the session file is assumed to be in the user's VisIt directory. If the visitDir argument is set to 0 then the filename argument must contain the absolute path to the session file.

Example:
```
% visit -cli
# Restore my session file for a time-varying database from
# my .visit directory.
RestoreSessionFile("visit.session", 1)
for state in range(TimeSliderGetNStates()):
SetTimeSliderState(state)
SaveWindow()
```

SaveAttribute: loads or saves a single attribute to/from an XML file.

Synopsis:
```
LoadAttribute(filename, object)
SaveAttribute(filename, object)
```

Returns:
success or failure

Description:
The LoadAttribute and SaveAttribute methods save a single attribute, such as a current plot or operator python object, to a standalone XML file. Note that LoadAttribute requires that the target attribute already be created by other means; it fills, but does not create, the attribute.

Example:
```
% visit -cli
a = MeshPlotAttributes()
SaveAttribute('mesh.xml', a)
b = MeshPlotAttributes()
LoadAttribute('mesh.xml', b)
```

SaveNamedSelection: saves a named selection from the active plot to a file.

Synopsis:
`SaveNamedSelection(name) -> integer`

Arguments:
`name` The name of a named selection.

Returns:
The SaveNamedSelection function returns 1 for success and 0 for failure.

Description:
Named Selections allow you to select a group of elements (or particles). One typically creates a named selection from a group of elements and then later applies the named selection to another plot (thus reducing the set of elements displayed to the ones from when the named selection was created). Named selections only last for the current session. If you create a named selection that you want to use over and over, you can save it to a file with the SaveNamedSelection function.

Example:
```
% visit -cli
db = "/usr/gapps/visit/data/wave*.silo database"
OpenDatabase(db)
AddPlot("Pseudocolor", "pressure")
AddOperator("Clip")
c = ClipAttributes()
c.plane1Origin = (0,0.6,0)
c.plane1Normal = (0,-1,0)
SetOperatorOption(c)
DrawPlots()
CreateNamedSelection("els_above_at_time_0")
SaveNamedSelection("els_above_at_time_0")
```

SaveSession: tells VisIt to save a session file describing the current visualization.

Synopsis:
`SaveSession(filename) -> integer`

Arguments:

`filename`	The filename argument is the filename that is used to save the sessionfile. The filename is relative to the current working directory.

Returns:
The SaveSession function returns 1 on success and 0 on failure.

Description:
The SaveSession function tells VisIt to save an XML session file that describes everything about the current visualization. Session files are very useful for creating movies and also as shortcuts for setting up complex visualizations.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/noise.silo")
# Set up a keyframe animation of view and save a session file of it.
k = GetKeyframeAttributes()
k.enabled,k.nFrames,k.nFramesWasUserSet = 1,20,1
SetKeyframeAttributes(k)
AddPlot("Surface", "hgslice")
DrawPlots()
v = GetView3D()
v.viewNormal = (0.40823, -0.826468, 0.387684)
v.viewUp, v.imageZoom = (-0.261942, 0.300775, 0.917017), 1.60684
SetView3D(v)
SetViewKeyframe()
SetTimeSliderState(TimeSliderGetNStates() - 1)
v.viewNormal = (-0.291901, -0.435608, 0.851492)
v.viewUp = (0.516969, 0.677156, 0.523644)
SetView3D(v)
SetViewKeyframe()
ToggleCameraViewMode()
SaveSession("~/.visit/keyframe.session")
```

SaveWindow: save the contents of the active window

Synopsis:
`SaveWindow() -> string`

Returns:
The SaveWindow function returns a string containing the name of the file that was saved.

Description:
The SaveWindow function saves the contents of the active visualization window. The format of the saved window is dictated by the SaveWindowAttributes which can be set using the SetSaveWindowAttributes function. The contents of the active visualization window can be saved as TIFF, JPEG, RGB, PPM, PNG images or they can be saved as curve, Alias Wavefront Obj, or VTK geometry files.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/curv3d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
# Set the save window attributes.
s = SaveWindowAttributes()
s.fileName = "test"
s.format = s.JPEG
s.progressive = 1
s.fileName = "test"
SetSaveWindowAttributes(s)
name = SaveWindow()
print "name = %s" % name
```

SendSimulationCommand: sends a command to an active simulation.

Synopsis:
```
SendSimulationCommand(host, simulation, command)
SendSimulationCommand(host, simulation, command, argument)
```

Arguments:

`host`	The name of the computer where the simulation is running.
`simulation`	The name of the simulation being processed at the specified host.
`command`	The command to send to the simulation.
`argument`	An argument to the command.

Returns:
The SendSimulationCommand method does not return a value.

Description:
The SendSimulationCommand method tells the viewer to send a command to a simulation that is running on the specified host. The host argument is a string that contains the name of the computer where the simulation is running. The simulation argument is a string that contains the name of the simulation to send the command to.

SetActiveContinuousColorTable: sets the color table that is used by plots that use a "default" color

Synopsis:
```
SetActiveContinuousColorTable(name) -> integer
SetActiveDiscreteColorTable(name) -> integer
```

Arguments:

`name`	The name of the color table to use for the active color table. The namemust be present in the tuple returned by the ColorTableNames function.

Returns:
Both functions return 1 on success and 0 on failure.

Description:
VisIt supports two flavors of color tables: continuous and discrete. Both types of color tables have the same underlying representation but each type of color table is used a slightly different way. Continuous color tables are made of a small number of color control points and the gaps in the color table between two color control points are filled by interpolating the colors of the color control points. Discrete color tables do not use any kind of interpolation and like continuous color tables, they are made up of control points. The color control points in a discrete color table repeat infinitely such that if we have 4 color control points: A, B, C, D then the pattern of repetition is: ABCDABCDABCD... Discrete color tables are mainly used for plots that have a discrete set of items to display (e.g. Subset plot). Continuous color tables are used in plots that display a continuous range of values (e.g. Pseudocolor).

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Contour", "hgslice")
DrawPlots()
SetActiveDiscreteColorTable("levels")
```

SetActiveDiscreteColorTable: sets the color table that is used by plots that use a "default" color

Synopsis:
```
SetActiveContinuousColorTable(name) -> integer
SetActiveDiscreteColorTable(name) -> integer
```

Arguments:

`name`	The name of the color table to use for the active color table. The namemust be present in the tuple returned by the ColorTableNames function.

Returns:
Both functions return 1 on success and 0 on failure.

Description:
VisIt supports two flavors of color tables: continuous and discrete. Both types of color tables have the same underlying representation but each type of color table is used a slightly different way. Continuous color tables are made of a small number of color control points and the gaps in the color table between two color control points are filled by interpolating the colors of the color control points. Discrete color tables do not use any kind of interpolation and like continuous color tables, they are made up of control points. The color control points in a discrete color table repeat infinitely such that if we have 4 color control points: A, B, C, D then the pattern of repetition is: ABCDABCDABCD... Discrete color tables are mainly used for plots that have a discrete set of items to display (e.g. Subset plot). Continuous color tables are used in plots that display a continuous range of values (e.g. Pseudocolor).

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Contour", "hgslice")
DrawPlots()
SetActiveDiscreteColorTable("levels")
```

SetActivePlots: sets the active plots in the plot list.

Synopsis:
`SetActivePlots(plots) -> integer`

Arguments:
`plots` A tuple of integer plot indices starting at zero. A single integer is also accepted

Returns:
The SetActivePlots function returns an integer value of 1 for success and 0 for failure.

Description:
Any time VisIt sets the attributes for a plot, it only sets the attributes for plots which are active. The SetActivePlots function must be called to set the active plots. The function takes one argument which is a tuple of integer plot indices that start at zero. If only one plot is being selected, the plots argument can be an integer instead of a tuple.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Subset", "mat1")
AddPlot("Mesh", "mesh1")
AddPlot("Contour", "u")
DrawPlots()
SetActivePlots((0,1,2)) # Make all plots active
SetActivePlots(0) # Make only the Subset plot active
```

SetActiveTimeSlider: sets the active time slider.

Synopsis:
`SetActiveTimeSlider(tsName) -> integer`

Arguments:

`tsName`	A string containing the name of the time slider that should be made active.

Returns:
SetActiveTimeSlider returns 1 on success and 0 on failure.

Description:
Sets the active time slider, which is the time slider that is used to change time states.

Example:
```
% visit -cli
path = "/usr/gapps/visit/data/"
dbs = (path + "dbA00.pdb", path + "dbB00.pdb", path + "dbC00.pdb")
for db in dbs:
    OpenDatabase(db)
    AddPlot("FilledBoundary", "material(mesh)")
    DrawPlots()
CreateDatabaseCorrelation("common", dbs, 1)
tsNames = GetWindowInformation().timeSliders
for ts in tsNames:
    SetActiveTimeSlider(ts)
for state in list(range(TimeSliderGetNStates())) + [0]:
    SetTimeSliderState(state)
```

SetActiveWindow: sets the active visualization window.

Synopsis:
```
SetActiveWindow(windowIndex) -> integer
SetActiveWindow(windowIndex, raiseWindow) -> integer
```

Arguments:

`windowIndex`	An integer window index starting at 1.
`raiseWindow`	This is an optional integer argument that raises and activates the window ifset to 1. If omitted, the default behavior is to raise and activate the window.

Returns:
The SetActiveWindow function returns an integer value of 1 for success and 0 for failure.

Description:
Most of the functions in the VisIt Python Interface operate on the contents of the active window. If there is more than one window, it is very important to be able to set the active window. To set the active window, use the SetActiveWindow function. The SetActiveWindow function takes a single integer argument which is the index of the new active window. The new window index must be an integer greater than zero and less than or equal to the number of open windows.

Example:
```
% visit -cli
SetWindowLayout(2)
SetActiveWindow(2)
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Mesh", "mesh1")
DrawPlots()
```

SetAnimationTimeout: sets the speed at which animations play.

Synopsis:
`SetAnimationTimeout(milliseconds) -> integer`

Returns:
The SetAnimationTimeout function returns 1 for success and 0 for failure.

Description:
The SetAnimationTimeout function sets the animation timeout which is a value that governs how fast animations play. The timeout is specified in milliseconds and has a default value of 1 millisecond. Larger timeout values decrease the speed at which animations play.

Example:
```
%visit -cli
# Play a new frame every 5 seconds.
SetAnimationTimeout(5000)
OpenDatabase("/usr/gapps/visit/data/wave.visit")
AddPlot("Pseudocolor", "pressure")
DrawPlots()
# Click the play button in the toolbar
```

SetAnnotationAttributes: sets the annotation attributes for the active window.

Synopsis:
```
SetAnnotationAttributes(atts) -> integer
SetDefaultAnnotationAttributes(atts) -> integer
```

Arguments:
`atts` An AnnotationAttributes object containing the annotation settings.

Returns:
Both functions return 1 on success and 0 on failure.

Description:
The annotation settings control what bits of text are drawn in the visualization window. Among the annotations are the plot legends, database information, user information, plot axes, triad, and the background style and colors. Setting the annotation attributes is important for producing quality visualizations. The annotation settings are stored in AnnotationAttributes objects. To set the annotation attributes, first create an AnnotationAttributes object using the AnnotationAttributes function and then pass the object to the SetAnnotationAttributes function. To set the default annotation attributes, also pass the object to the SetDefaultAnnotationAttributes function.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/wave.visit")
AddPlot("Pseudocolor", "pressure")
DrawPlots()
a = AnnotationAttributes()
a.gradientBackgroundStyle = a.GRADIENTSTYLE_RADIAL
a.gradientColor1 = (0,255,255)
a.gradientColor2 = (0,0,0)
a.backgroundMode = a.BACKGROUNDMODE_GRADIENT
SetAnnotationAttributes(a)
```

SetCenterOfRotation: sets the center of rotation for plots in a 3D vis window.

Synopsis:
`SetCenterOfRotation(x,y,z) -> integer`

Arguments:
- `x` The x component of the center of rotation.
- `y` The y component of the center of rotation.
- `z` The z component of the center of rotation.

Returns:
The SetCenterOfRotation function returns 1 on success and 0 on failure.

Description:
The SetCenterOfRotation function sets the center of rotation for plots in a 3D visualization window. The center of rotation, is the point about which plots are rotated when you interactively spin the plots using the mouse. It is useful to set the center of rotation if you've zoomed in on any 3D plots so in the event that you rotate the plots, the point of interest remains fixed on the screen.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
AddPlot("Mesh", "mesh1")
DrawPlots()
v = GetView3D()
v.viewNormal = (-0.409139, 0.631025, 0.6591)
v.viewUp = (0.320232, 0.775678, -0.543851)
v.imageZoom = 4.8006
SetCenterOfRotation(-4.755280, 6.545080, 5.877850)
# Rotate the plots interactively.
```

SetColorTexturingEnabled: sets whether nodal values will be drawn using hardware texturing instead of

Synopsis:
`SetColorTexturingEnabled(enabled) -> integer`

Arguments:

`enabled`	A integer value. Non−zero values enable color texturing and zero disables it.

Returns:
The SetColorTexturingEnabled function returns 1 on success and 0 on failure.

Description:
Node−centered variables are drawn on plots such as the Pseudocolor plot such that the nodal value looks interpolated throughout the zone. This can be done by interpolating colors, which can produce some colors that do not appear in a color table. Alternatively, the nodal values can be mapped to a texture coordinate in a 1D texture and those values can be interpolated, with colors being selected after interpolating the texture coordinate. This method always uses colors that are defined in the color table.

Example:
`SetColorTexturingEnabled(1)`

SetCreateMeshQualityExpressions: sets global attribute boolean to automatically create Mesh Quality

Synopsis:
`SetCreateMeshQualityExpressions(val) -> integer`

Arguments:
`val` Either a zero (false) or non−zero (true) integer value to indicate ifMesh Quality expressions should be automatically created when a database is opened.

Returns:
The SetCreateMeshQualityExpressions function returns 1 on success and 0 on failure.

Description:
The SetCreateMeshQualityExpressions function sets a boolean in the global attributes indicating whether or not Mesh Quality expressions should be automatically created. The default behavior is for the expressions to be created, which may slow down VisIt's performance if there is an extraordinary large number of meshes. Turning this feature off tells VisIt to skip automatic creation of the Mesh Quality expressions.

Example:
```
% visit -cli
SetCreateMeshQualityExpressions(1) # turn this feature on
SetCreateMeshQualityExpressions(0) # turn this feature off
```

SetCreateTimeDerivativeExpressions: sets global attribute boolean to automatically create Time Derivative

Synopsis:
`SetCreateTimeDerivativeExpressions(val) -> integer`

Arguments:
`val` Either a zero (false) or non−zero (true) integer value to indicate ifTime Derivative expressions should be automatically created when a database is opened.

Returns:
The SetCreateTimeDerivativeExpressions function returns 1 on success and 0 on failure.

Description:
The SetCreateTimeDerivativeExpressions function sets a boolean in the global attributes indicating whether or not Time Derivative expressions should be automatically created. The default behavior is for the expressions to be created, which may slow down VisIt's performance if there is an extraordinary large number of variables. Turning this feature off tells VisIt to skip automatic creation of the Time Derivative expressions.

Example:
```
% visit -cli
SetCreateTimeDerivativeExpressions(1) # turn this feature on
SetCreateTimeDerivativeExpressions(0) # turn this feature off
```

SetCreateVectorMagnitudeExpressions: sets global attribute boolean to automatically create Vector Magnitude

Synopsis:
`SetCreateVectorMagnitudeExpressions(val) -> integer`

Arguments:
`val` Either a zero (false) or non−zero (true) integer value to indicate ifVector magnitude expressions should be automatically created when a database is opened.

Returns:
The SetCreateVectorMagnitudeExpressions function returns 1 on success and 0 on failure.

Description:
The SetCreateVectorMagnitudeExpressions function sets a boolean in the global attributes indicating whether or not vector magnitude expressions should be automatically created. The default behavior is for the expressions to be created, which may slow down VisIt's performance if there is an extraordinary large number of vector variables. Turning this feature off tells VisIt to skip automatic creation of the vecotr magnitude expressions. expressions.

Example:
```
% visit -cli
SetCreateVectorMagnitudeExpressions(1) # turn this feature on
SetCreateVectorMagnitudeExpressions(0) # turn this feature off
```

SetDatabaseCorrelationOptions: sets the global options for database correlations.

Synopsis:
`SetDatabaseCorrelationOptions(method, whenToCreate) -> integer`

Arguments:

`method`	An integer that tells VisIt what default method to use when automaticallycreating a database correlation. The value must be in the range [0,3].
`whenToCreate`	An integer that tells VisIt when to automatically create databasecorrelations.

Returns:
SetDatabaseCorrelationOptions returns 1 on success and 0 on failure.

Description:
VisIt provides functions to explicitly create and alter database correlations but there are also a number of occasions where VisIt can automatically create a database correlation. The SetDatabaseCorrelationOptions function allows you to tell VisIt the default correlation method to use when automatically creating a new database correlation and it also allows you to tell VisIt when database correlations can be automatically created.

`method`	Description
`0`	IndexForIndexCorrelation
`1`	StretchedIndexCorrelation
`2`	TimeCorrelation
`3`	CycleCorrelation

`whenToCreate`	Description
`0`	Always create database correlation
`1`	Never create database correlation
`2`	Create database correlation only if the new time−varying database hasthe same length as another time−varying database already being used in aplot.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/dbA00.pdb")
AddPlot("FilledBoundary", "material(mesh)")
DrawPlots()
# Always create a stretched index correlation.
SetDatabaseCorrelationOptions(1, 0)
OpenDatabase("/usr/gapps/visit/data/dbB00.pdb")
AddPlot("FilledBoundary", "material(mesh)")
# The AddPlot caused a database correlation to be created.
DrawPlots()
wi = GetWindowInformation()
print "Active time slider: " % wi.timeSliders[wi.activeTimeSlider]
# This will set time for both databases since the database correlation is
the active time slider.
SetTimeSliderState(5)
```

SetDebugLevel: set or Get the VisIt module's debug level.

Synopsis:
```
GetDebugLevel() -> integer
SetDebugLevel(level)
```

Arguments:

`level`	A string '1', '2', '3', '4', '5' with an optional 'b' suffix to indicatewhether the output should be buffered. A value of '1' is a low debug level, which should be used to produce little output while a value of 5 should produce a lot of debug output.

Returns:
The GetDebugLevel function returns the debug level of the VisIt module.

Description:
The GetDebugLevel and SetDebugLevel functions are used when debugging VisIt Python scripts. The SetDebugLevel function sets the debug level for VisIt's viewer thus it must be called before a Launch method. The debug level determines how much detail is written to VisIt's execution logs when it executes. The GetDebugLevel function can be used in Python scripts to alter the behavior of the script. For instance, the debug level can be used to selectively print values to the console.

Example:
```
% visit -cli -debug 2
print "VisIt's debug level is: %d" % GetDebugLevel()
```

SetDefaultAnnotationAttributes: sets the annotation attributes for the active window.

Synopsis:
```
SetAnnotationAttributes(atts) -> integer
SetDefaultAnnotationAttributes(atts) -> integer
```

Arguments:
`atts` An AnnotationAttributes object containing the annotation settings.

Returns:
Both functions return 1 on success and 0 on failure.

Description:
The annotation settings control what bits of text are drawn in the visualization window. Among the annotations are the plot legends, database information, user information, plot axes, triad, and the background style and colors. Setting the annotation attributes is important for producing quality visualizations. The annotation settings are stored in AnnotationAttributes objects. To set the annotation attributes, first create an AnnotationAttributes object using the AnnotationAttributes function and then pass the object to the SetAnnotationAttributes function. To set the default annotation attributes, also pass the object to the SetDefaultAnnotationAttributes function.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/wave.visit")
AddPlot("Pseudocolor", "pressure")
DrawPlots()
a = AnnotationAttributes()
a.gradientBackgroundStyle = a.GRADIENTSTYLE_RADIAL
a.gradientColor1 = (0,255,255)
a.gradientColor2 = (0,0,0)
a.backgroundMode = a.BACKGROUNDMODE_GRADIENT
SetAnnotationAttributes(a)
```

SetDefaultFileOpenOptions: sets the current default options used when opening files for a

Synopsis:
`GetDefaultFileOpenOptions(pluginName, options) -> integer`

Arguments:

`pluginName`	The name of a plugin.
`options`	A dictionary containing the new default options for that plugin.

Returns:
The SetDefaultFileOpenOptions function returns 1 on success and 0 on failure.

Description:
SetDefaultFileOpenOptions sets the current options used to open new files when a specific plugin is triggered.

Example:
```
% visit -cli
OpenMDServer()
opts = GetDefaultFileOpenOptions("VASP")
opts["Allow multiple timesteps"] = 1
SetDefaultFileOpenOptions("VASP", opts)
OpenDatabase("CHGCAR")
```

SetDefaultInteractorAttributes: sets VisIt's interactor attributes.

Synopsis:
```
SetInteractorAttributes(atts) -> integer
SetDefaultInteractorAttributes(atts) -> integer
```

Arguments:

`atts`	An InteractorAttributes object that contains the new interactor attributesthat you want to use.

Returns:
SetInteractorAttributes returns 1 on success and 0 on failure.

Description:
The SetInteractorAttributes function is used to set certain interactor properties. Interactors, can be thought of as how mouse clicks and movements are translated into actions in the vis window. To set the interactor attributes, first get the interactor attributes using the GetInteractorAttributes function. Once you've set the object's properties, call the SetInteractorAttributes function to make VisIt use the new interactor attributes. The SetDefaultInteractorAttributes function sets the default interactor attributes, which are used for new visualization windows. The default interactor attributes can also be saved to the VisIt configuration file to ensure that future VisIt sessions have the right default interactor attributes.

Example:
```
% visit -cli
ia = GetInteractorAttributes()
print ia
ia.showGuidelines = 0
SetInteractorAttributes(ia)
```

SetDefaultMaterialAttributes: sets VisIts material interface reconstruction options.

Synopsis:
```
SetMaterialAttributes(atts) -> integer
SetDefaultMaterialAttributes(atts) -> integer
```

Arguments:
`atts` A MaterialAttributes object containing the new settings.

Returns:
Both functions return 1 on success and 0 on failure.

Description:
The SetMaterialAttributes function takes a MaterialAttributes object and makes VisIt use the material settings that it contains. You use the SetMaterialAttributes function when you want to change how VisIt performs material interface reconstruction. The SetDefaultMaterialAttributes function sets the default material attributes, which are saved to the config file and are also used by new visualization windows.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/allinone00.pdb")
AddPlot("Pseudocolor", "mesh/mixvar")
p = PseudocolorAttributes()
p.min,p.minFlag = 4.0, 1
p.max,p.maxFlag = 13.0, 1
SetPlotOptions(p)
DrawPlots()
# Tell VisIt to always do material interface reconstruction.
m = GetMaterialAttributes()
m.forceMIR = 1
SetMaterialAttributes(m)
ClearWindow()
# Redraw the plot forcing VisIt to use the mixed variable information.
DrawPlots()
```

SetDefaultMeshManagementAttributes: returns a MeshManagementAttributes object containing VisIt's current mesh

Synopsis:
`GetMeshmanagementAttributes() -> MeshmanagementAttributes object`

Returns:
Returns a MeshmanagementAttributes object.

Description:
The GetMeshmanagementAttributes function returns a MeshmanagementAttributes object that contains VisIt's current mesh discretization settings. You can set properties on the MeshManagementAttributes object and then pass it to SetMeshManagementAttributes to make VisIt use the new material attributes that you've specified:

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/csg.silo")
AddPlot("Mesh", "csgmesh")
DrawPlots()
# Tell VisIt to always do material interface reconstruction.
mma = GetMeshManagementAttributes()
mma.discretizationTolernace = (0.01, 0.025)
SetMeshManagementAttributes(mma)
ClearWindow()
# Redraw the plot forcing VisIt to use the mixed variable information.
DrawPlots()
```

SetDefaultOperatorOptions: sets the attributes for an operator.

Synopsis:
```
SetOperatorOptions(atts) -> integer
SetOperatorOptions(atts, operatorIndex) -> integer
SetOperatorOptions(atts, operatorIndex, all) -> integer
SetDefaultOperatorOptions(atts) -> integer
```

Arguments:

`atts`	Any type of operator attributes object.
`operatorIndex`	An optional zero−based integer that serves as an index into the activeplot's operator list. Use this argument if you want to set the operatorattributes for a plot that has multiple instances of the same type ofoperator. For example, if the active plot had a Transform operatorfollowed by a Slice operator followed by another Transform operator andyou wanted to adjust the attributes of the second Transform operator,you would pass an operatorIndex value of 2.
`all`	An optional integer argument that tells the function to apply the operatorattributes to all plots containing the specified operator if the value ofthe argument is non−zero.

Returns:
All functions return an integer value of 1 for success and 0 for failure.

Description:
Each operator in VisIt has a group of attributes that controls the operator. To set the attributes for an operator, first create an operator attributes object. This is done by calling a function which is the name of the operator plus the word "Attributes". For example, a Slice operator's operator attributes object is created and returned by the SliceAttributes function. Assign the new operator attributes object into a variable and set its fields. After setting the desired fields in the operator attributes object, pass the object to the SetOperatorOptions function. The SetOperatorOptions function determines the type of operator to which the operator attributes object applies and sets the attributes for that operator type. To set the default plot attributes, use the SetDefaultOperatorOptions function. Setting the default attributes ensures that all future instances of a certain operator are initialized with the new default values. Note that there is no SetOperatorOptions(atts, all) variant of this call. To set operator options for all plots that have a an instance of the associated operator, you must first make all plots active with SetActivePlots() and then use the SetOperatorOptions(atts) variant.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
AddPlot("Mesh", "mesh1")
AddOperator("Slice", 1) # Add the operator to both plots
a = SliceAttributes()
a.normal, a.upAxis = (0,0,1), (0,1,0)
# Only set the attributes for the active plot.
SetOperatorOptions(a)
DrawPlots()
```

SetDefaultPickAttributes: changes the pick settings that VisIt uses when picking on plots.

Synopsis:
```
SetPickAttributes(atts) -> integer
SetDefaultPickAttributes(atts) -> integer
ResetPickAttributes() -> integer
```

Arguments:
`atts` A PickAttributes object containing the new pick settings.

Returns:
All functions return 1 on success and 0 on failure.

Description:
The SetPickAttributes function changes the pick attributes that are used when VisIt picks on plots. The pick attributes allow you to format your pick output in various ways and also allows you to select auxiliary pick variables.

Example:
```
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Pseudocolor", "hgslice")
DrawPlots()
ZonePick((-5,5))
p = GetPickAttributes()
p.showTimeStep = 0
p.showMeshName = 0
p.showZoneId = 0
SetPickAttributes(p)
ZonePick((0,5))
```

SetDefaultPlotOptions: sets plot attributes for the active plots.

Synopsis:
```
SetPlotOptions(atts) -> integer
SetDefaultPlotOptions(atts) -> integer
```

Arguments:
`atts` Any type of plot attributes object.

Returns:
All functions return an integer value of 1 for success and 0 for failure.

Description:
Each plot in VisIt has a group of attributes that controls the appearance of the plot. To set the attributes for a plot, first create a plot attributes object. This is done by calling a function which is the name of the plot plus the word "Attributes". For example, a Pseudocolor plot's plotattributes object is created and returned by the PseudocolorAttributes function. Assign the new plot attributes object into a variable and set its fields. After setting the desired fields in the plot attributes object, pass the object to the SetPlotOptions function. The SetPlotOptions function determines the type of plot to which the plot attributes object applies and sets the attributes for that plot type. To set the default plot attributes, use the SetDefaultPlotOptions function. Setting the default attributes ensures that all future instances of a certain plot are initialized with the new default values.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
p = PseudocolorAttributes()
p.colorTableName = "calewhite"
p.minFlag,p.maxFlag = 1,1
p.min,p.max = -5.0, 8.0
SetPlotOptions(p)
DrawPlots()
```

SetGlobalLineoutAttributes: sets global lineout attributes that are used for all lineouts.

Synopsis:
`SetGlobalLineoutAttributes(atts) -> integer`

Arguments:
`atts` A GlobalLineoutAttributes object that contains the new settings.

Returns:
The SetGlobalLineoutAttributes function returns 1 on success and 0 on failure.

Description:
The SetGlobalLineoutAttributes function allows you to set global lineout options that are used in the creation of all lineouts. You can, for example, specify the destination window and the number of sample points for lineouts.

Example:
```
% visit -cli
SetWindowLayout(4)
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Pseudocolor", "hgslice")
DrawPlots()
gla = GetGlobalLineoutAttributes()
gla.createWindow = 0
gla.windowId = 4
gla.samplingOn = 1
gla.numSamples = 150
SetGlobalLineoutAttributes(gla)
Lineout((-5,-8), (-3.5, 8))
```

SetInteractorAttributes: sets VisIt's interactor attributes.

Synopsis:
```
SetInteractorAttributes(atts) -> integer
SetDefaultInteractorAttributes(atts) -> integer
```

Arguments:

`atts`	An InteractorAttributes object that contains the new interactor attributesthat you want to use.

Returns:
SetInteractorAttributes returns 1 on success and 0 on failure.

Description:
The SetInteractorAttributes function is used to set certain interactor properties. Interactors, can be thought of as how mouse clicks and movements are translated into actions in the vis window. To set the interactor attributes, first get the interactor attributes using the GetInteractorAttributes function. Once you've set the object's properties, call the SetInteractorAttributes function to make VisIt use the new interactor attributes. The SetDefaultInteractorAttributes function sets the default interactor attributes, which are used for new visualization windows. The default interactor attributes can also be saved to the VisIt configuration file to ensure that future VisIt sessions have the right default interactor attributes.

Example:
```
% visit -cli
ia = GetInteractorAttributes()
print ia
ia.showGuidelines = 0
SetInteractorAttributes(ia)
```

SetKeyframeAttributes: sets VisIt's keyframing attributes.

Synopsis:
`SetKeyframeAttributes(kfAtts) -> integer`

Arguments:

`kfAtts`	A KeyframeAttributes object that contains the new keyframing attributes touse.

Returns:
SetKeyframeAttributes returns 1 on success and 0 on failure.

Description:
Use the SetKeyframeAttributes function when you want to change VisIt's keyframing settings. You must pass a KeyframeAttributes object, which you can create using the GetKeyframeAttributes function. The KeyframeAttributes object must contain the keyframing settings that you want VisIt to use. For example, you would use the SetKeyframeAttributes function if you wanted to turn on keyframing mode and set the number of animation frames.

Example:
```
% visit -cli
k = GetKeyframeAttributes()
print k
k.enabled,k.nFrames,k.nFramesWasUserSet = 1, 100, 1
SetKeyframeAttributes(k)
```

SetLight: returns a light object containing the attributes for a specified light.

Synopsis:
`SetLight(index, light) -> integer`

Arguments:

`index`	A zero−based integer index into the light list. Index can be in the range[0,7].
`light`	A LightAttributes object containing the properties to use for thespecified light.

Returns:
SetLight returns 1 on success and 0 on failure.

Description:
The SetLight function sets the attributes for a specific light.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "w")
p = PseudocolorAttributes()
p.colorTableName = "xray"
SetPlotOptions(p)
DrawPlots()
InvertBackgroundColor()
light = GetLight(0)
print light
light.enabledFlag = 1
light.direction = (0,-1,0)
light.color = (255,0,0,255)
SetLight(0, light)
light.color,light.direction = (0,255,0,255), (-1,0,0)
SetLight(1, light)
```

SetMachineProfile: ¡description¿

Synopsis:
`SetMachineProfile(MachineProfile) -> integer`

Arguments:
`MachineProfile`

Description:
Sets the input machine profile in the HostProfileList, replaces if one already exists Otherwise adds to the list

SetMaterialAttributes: sets VisIts material interface reconstruction options.

Synopsis:
```
SetMaterialAttributes(atts) -> integer
SetDefaultMaterialAttributes(atts) -> integer
```

Arguments:
`atts` A MaterialAttributes object containing the new settings.

Returns:
Both functions return 1 on success and 0 on failure.

Description:
The SetMaterialAttributes function takes a MaterialAttributes object and makes VisIt use the material settings that it contains. You use the SetMaterialAttributes function when you want to change how VisIt performs material interface reconstruction. The SetDefaultMaterialAttributes function sets the default material attributes, which are saved to the config file and are also used by new visualization windows.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/allinone00.pdb")
AddPlot("Pseudocolor", "mesh/mixvar")
p = PseudocolorAttributes()
p.min,p.minFlag = 4.0, 1
p.max,p.maxFlag = 13.0, 1
SetPlotOptions(p)
DrawPlots()
# Tell VisIt to always do material interface reconstruction.
m = GetMaterialAttributes()
m.forceMIR = 1
SetMaterialAttributes(m)
ClearWindow()
# Redraw the plot forcing VisIt to use the mixed variable information.
DrawPlots()
```

SetMeshManagementAttributes: returns a MeshManagementAttributes object containing VisIt's current mesh

Synopsis:
`GetMeshmanagementAttributes() -> MeshmanagementAttributes object`

Returns:
Returns a MeshmanagementAttributes object.

Description:
The GetMeshmanagementAttributes function returns a MeshmanagementAttributes object that contains VisIt's current mesh discretization settings. You can set properties on the MeshManagementAttributes object and then pass it to SetMeshManagementAttributes to make VisIt use the new material attributes that you've specified:

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/csg.silo")
AddPlot("Mesh", "csgmesh")
DrawPlots()
# Tell VisIt to always do material interface reconstruction.
mma = GetMeshManagementAttributes()
mma.discretizationTolernace = (0.01, 0.025)
SetMeshManagementAttributes(mma)
ClearWindow()
# Redraw the plot forcing VisIt to use the mixed variable information.
DrawPlots()
```

SetNamedSelectionAutoApply: set whether plots that create named selections should cause their named

Synopsis:
`SetNamedSelectionAutoApply(flag) -> integer`

Arguments:
`flag` Non−zero values turn on selection auto apply mode.

Returns:
The SetNamedSelectionAutoApply function returns 1 on success and 0 on failure.

Description:
Named selections are often associated with plots for their data source. When those plots update, their named selections can be updated, which in turn will update any plots that use the named selection. When this mode is enabled, changes to a named selection's originating plot will cause the selection to be updated automatically.

Example:
```
SetNamedSelectionAutoApply(1)
```

SetOperatorOptions: sets the attributes for an operator.

Synopsis:
```
SetOperatorOptions(atts) -> integer
SetOperatorOptions(atts, operatorIndex) -> integer
SetOperatorOptions(atts, operatorIndex, all) -> integer
SetDefaultOperatorOptions(atts) -> integer
```

Arguments:

`atts`	Any type of operator attributes object.
`operatorIndex`	An optional zero−based integer that serves as an index into the activeplot's operator list. Use this argument if you want to set the operatorattributes for a plot that has multiple instances of the same type ofoperator. For example, if the active plot had a Transform operatorfollowed by a Slice operator followed by another Transform operator andyou wanted to adjust the attributes of the second Transform operator,you would pass an operatorIndex value of 2.
`all`	An optional integer argument that tells the function to apply the operatorattributes to all plots containing the specified operator if the value ofthe argument is non−zero.

Returns:
All functions return an integer value of 1 for success and 0 for failure.

Description:
Each operator in VisIt has a group of attributes that controls the operator. To set the attributes for an operator, first create an operator attributes object. This is done by calling a function which is the name of the operator plus the word "Attributes". For example, a Slice operator's operator attributes object is created and returned by the SliceAttributes function. Assign the new operator attributes object into a variable and set its fields. After setting the desired fields in the operator attributes object, pass the object to the SetOperatorOptions function. The SetOperatorOptions function determines the type of operator to which the operator attributes object applies and sets the attributes for that operator type. To set the default plot attributes, use the SetDefaultOperatorOptions function. Setting the default attributes ensures that all future instances of a certain operator are initialized with the new default values. Note that there is no SetOperatorOptions(atts, all) variant of this call. To set operator options for all plots that have a an instance of the associated operator, you must first make all plots active with SetActivePlots() and then use the SetOperatorOptions(atts) variant.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
AddPlot("Mesh", "mesh1")
AddOperator("Slice", 1) # Add the operator to both plots
a = SliceAttributes()
a.normal, a.upAxis = (0,0,1), (0,1,0)
# Only set the attributes for the active plot.
SetOperatorOptions(a)
DrawPlots()
```

SetPickAttributes: changes the pick settings that VisIt uses when picking on plots.

Synopsis:
```
SetPickAttributes(atts) -> integer
SetDefaultPickAttributes(atts) -> integer
ResetPickAttributes() -> integer
```

Arguments:
`atts` A PickAttributes object containing the new pick settings.

Returns:
All functions return 1 on success and 0 on failure.

Description:
The SetPickAttributes function changes the pick attributes that are used when VisIt picks on plots. The pick attributes allow you to format your pick output in various ways and also allows you to select auxiliary pick variables.

Example:
```
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Pseudocolor", "hgslice")
DrawPlots()
ZonePick((-5,5))
p = GetPickAttributes()
p.showTimeStep = 0
p.showMeshName = 0
p.showZoneId = 0
SetPickAttributes(p)
ZonePick((0,5))
```

SetPipelineCachingMode: sets the pipeline caching mode.

Synopsis:
`SetPipelineCachingMode(mode) -> integer`

Returns:
The SetPipelineCachingMode function returns 1 for success and 0 for failure.

Description:
The SetPipelineCachingMode function turns pipeline caching on or off in the viewer. When pipeline caching is enabled, animation timesteps are cached for fast playback. This can be a disadvantage for large databases or for plots with many timesteps because it increases memory consumption. In those cases, it is often useful to disable pipeline caching so the viewer does not use as much memory. When the viewer does not cache pipelines, each plot for a timestep must be recalculated each time the timestep is visited.

Example:
```
% visit -cli
SetPipelineCachingMode(0) # Disable caching
OpenDatabase("/usr/gapps/visit/data/wave.visit")
AddPlot("Pseudocolor", "pressure")
AddPlot("Mesh", "quadmesh")
DrawPlots()
for state in range(TimeSliderGetNStates()):
    SetTimeSliderState(state)
```

SetPlotDatabaseState: sets a database keyframe for a specific plot.

Synopsis:
`SetPlotDatabaseState(index, frame, state)`

Arguments:

`index`	A zero−based integer index that is the plot's location in the plot list.
`frame`	A zero−baed integer index representing the animation frame for which we'regoing to add a database keyframe.
`state`	A zero−based integer index representating the database time state thatwe're going to use at the specified animation frame.

Returns:
The SetPlotDatabaseState function does not return a value.

Description:
The SetPlotDatabaseState function is used when VisIt is in keyframing mode to add a database keyframe for a specific plot. VisIt uses database keyframes to determine which database state is to be used for a given animation frame. Database keyframes can be used to stop "database time" while "animation time" continues forward and they can also be used to make "database time" go in reverse, etc.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/wave.visit")
k = GetKeyframeAttributes()
nFrames = 20
k.enabled, k.nFrames, k.nFramesWasUserSet = 1, nFrames, 1
SetKeyframeAttributes(k)
AddPlot("Pseudocolor", "pressure")
AddPlot("Mesh", "quadmesh")
DrawPlots()
# Make "database time" for the Pseudocolor plot go in reverse
SetPlotDatabaseState(0, 0, 70)
SetPlotDatabaseState(0, nFrames-1, 0)
# Animate through the animation frames since the "Keyframe animation"
time slider is active.
for state in range(TimeSliderGetNStates()):
    SetTimeSliderState(state)
```

SetPlotDescription: provide text that will be shown in the plot list instead of a plot's usual

Synopsis:
`SetPlotDescription(index, description) -> integer`

Arguments:

`index`	The index of the plot within the plot list.
`description`	A new description that will be shown in the plot list so the plot canbe identified readily.

Returns:
The SetPlotDescription function returns 1 on success and 0 on failure.

Description:
Managing many related plots can be a complex task. This function lets users provide meaningful descriptions for each plot so they can more easily be identified in the plot list.

Example:
```
SetPlotDescription(0, 'Mesh for reflected pressure plot')
```

SetPlotFollowsTime: set whether the active plots follow their time slider.

Synopsis:
`SetPlotFollowsTime(val) -> integer`

Arguments:
`val` An optional flag indicating whether the plot should follow the time slider.The default behavior is for the plot to follow the time slider.

Returns:
The function returns 1 on success and 0 on failure.

Description:
SetPlotFollowsTime can let you set whether the active plot follows the time slider.

Example:
```
SetPlotFollowsTime()
```

SetPlotFrameRange: sets the range of animation frames over which a plot is valid.

Synopsis:
`SetPlotFrameRange(index, start, end)`

Arguments:

`index`	A zero−based integer representing an index into the plot list.
`start`	A zero−based integer representing the animation frame where the plot firstappears in the visualization.
`end`	A zero−based integer representing the animation frame where the plotdisappears from the visualization.

Returns:
The SetPlotFrameRange function does not return a value.

Description:
The SetPlotFrameRange function sets the start and end frames for a plot when VisIt is in keyframing mode. Outside of this frame range, the plot does not appear in the visualization. By default, plots are valid over the entire range of animation frames when they are first created. Frame ranges allow you to construct complex animations where plots appear and disappear dynamically.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/wave.visit")
k = GetKeyframeAttributes()
nFrames = 20
k.enabled, k.nFrames, k.nFramesWasUserSet = 1, nFrames, 1
SetKeyframeAttributes(k)
AddPlot("Pseudocolor", "pressure")
AddPlot("Mesh", "quadmesh")
DrawPlots()
# Make the Pseudocolor plot take up the first half of the animation frames
before it disappears.
SetPlotFrameRange(0, 0, nFrames/2-1)
# Make the Mesh plot take up the second half of the animation frames.
SetPlotFrameRange(1, nFrames/2, nFrames-1)
for state in range(TimeSliderGetNStates())
SetTimeSliderState(state)
SaveWindow()
```

SetPlotOptions: sets plot attributes for the active plots.

Synopsis:
```
SetPlotOptions(atts) -> integer
SetDefaultPlotOptions(atts) -> integer
```

Arguments:
`atts` Any type of plot attributes object.

Returns:
All functions return an integer value of 1 for success and 0 for failure.

Description:
Each plot in VisIt has a group of attributes that controls the appearance of the plot. To set the attributes for a plot, first create a plot attributes object. This is done by calling a function which is the name of the plot plus the word "Attributes". For example, a Pseudocolor plot's plotattributes object is created and returned by the PseudocolorAttributes function. Assign the new plot attributes object into a variable and set its fields. After setting the desired fields in the plot attributes object, pass the object to the SetPlotOptions function. The SetPlotOptions function determines the type of plot to which the plot attributes object applies and sets the attributes for that plot type. To set the default plot attributes, use the SetDefaultPlotOptions function. Setting the default attributes ensures that all future instances of a certain plot are initialized with the new default values.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
p = PseudocolorAttributes()
p.colorTableName = "calewhite"
p.minFlag,p.maxFlag = 1,1
p.min,p.max = -5.0, 8.0
SetPlotOptions(p)
DrawPlots()
```

SetPlotOrderToFirst: move the i'th plot in the plot list to the start of the plot list.

Synopsis:
`SetPlotOrderToFirst(index) -> integer`

Arguments:
`index` The index of the plot within the plot list.

Returns:
The SetPlotOrderToFirst function returns 1 on success and 0 on failure.

Description:
Move the i'th plot in the plot list to the start of the plot list.

Example:
```
AddPlot('Mesh', 'mesh')
AddPlot('Pseudocolor', 'pressure')
# Make the Pseudocolor plot first in the plot list
SetPlotOrderToFirst(1)
```

SetPlotOrderToLast: move the i'th plot in the plot list to the end of the plot list.

Synopsis:
`SetPlotOrderToLast(index) -> integer`

Arguments:
`index` The index of the plot within the plot list.

Returns:
The SetPlotOrderToLast function returns 1 on success and 0 on failure.

Description:
Move the i'th plot in the plot list to the end of the plot list.

Example:
```
AddPlot('Mesh', 'mesh')
AddPlot('Pseudocolor', 'pressure')
# Make the Mesh plot last in the plot list
SetPlotOrderToLast(0)
```

SetPlotSILRestriction: set the SIL restriction for the active plots.

Synopsis:
```
SetPlotSILRestriction(silr) -> integer
SetPlotSILRestriction(silr, all) -> integer
```

Arguments:

`silr`	A SIL restriction object.
`all`	An optional argument that tells the function if the SIL restriction shouldbe applied to all plots in the plot list.

Returns:
The SetPlotSILRestriction function returns an integer value of 1 for success and 0 for failure.

Description:
VisIt allows the user to select subsets of databases. The description of the subset is called a Subset Inclusion Lattice Restriction, or SIL restriction. The SIL restriction allows databases to be subselected in several different ways. The VisIt Python Interface provides the SetPlotSILRestriction function to allow Python scripts to turn off portions of the plotted database. The SetPlotSILRestriction function accepts a SILRestriction object that contains the SIL restriction for the active plots. The optional all argument is an integer that tells the function to apply the SIL restriction to all plots when the value of the argument is non−zero. If the all argument is not supplied, then the SIL restriction is only applied to the active plots.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/multi_curv2d.silo")
AddPlot("Subset", "mat1")
silr = SILRestriction()
silr.TurnOffSet(silr.SetsInCategory('mat1')[1])
SetPlotSILRestriction(silr)
DrawPlots()
```

SetPreferredFileFormats: sets the list of preferred file format IDs.

Synopsis:
`SetPreferredFileFormats(pluginIDs) -> integer`

Arguments:

`pluginIDs`	A tuple of plugin IDs to be attempted first when opening files.

Returns:
The SetPreferredFileFormats method does not return a value.

Description:
The SetPreferredFileFormats method is a way to set the list of file format reader plugins which are tried before any others. These IDs must be full IDs, not just names, and are tried in order.

Example:

```
SetPreferredFileFormats('Silo_1.0')
SetPreferredFileFormats(('Silo_1.0','PDB_1.0'))
```

SetPrinterAttributes: sets the printer attributes.

Synopsis:
`SetPrinterAttributes(atts)`

Arguments:
`atts` A PrinterAttributes object.

Returns:
The SetPrinterAttributes function does not return a value.

Description:
The SetPrinterAttributes function sets the printer attributes. VisIt uses the printer attributes to determine how the active visualization window should be printed. The function accepts a single argument which is a PrinterAttributes object containing the printer attributes to use for future printing. VisIt allows images to be printed to a network printer or to a PostScript file that can be printed later by other applications.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/curv2d.silo")
AddPlot("Surface", "v")
DrawPlots()
# Make it print to a file.
p = PrinterAttributes()
p.outputToFile = 1
p.outputToFileName = "printfile"
SetPrinterAttributes(p)
PrintWindow()
```

SetQueryFloatFormat: sets the floating point format string used to generate query output.

Synopsis:
`SetQueryFloatFormat(format_string)`

Arguments:

`format_string`	A string object that provides a printf style floating point format.

Returns:
The SetQueryFloatFormat does not return a value.

Description:
The SetQueryFloatFormat method sets a printf style format string that isused by VisIt's querys to produce textual output.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/rect2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
# Set floating point format string.
SetQueryFloatFormat("%.1f")
Query("MinMax")
# Set format back to default "%g".
SetQueryFloatFormat("%g")
Query("MinMax")
```

SetQueryOverTimeAttributes: changes the settings that VisIt uses for queries over time.

Synopsis:
```
SetQueryOverTimeAttributes(atts) -> integer
SetDefaultQueryOverTimeAttributes(atts) -> integer
ResetQueryOverTimeAttributes() -> integer
```

Arguments:

`atts`	A QueryOverTimeAttributes object containing the new settings to use forqueries over time.

Returns:
All functions return 1 on success and 0 on failure.

Description:
The SetQueryOverTimeAttributes function changes the settings that VisIt uses for query over time. The SetDefaultQueryOverTimeAttributes function changes the settings that new visualization windows inherit for doing query over time. Finally, the ResetQueryOverTimeAttributes function forces VisIt to use the stored default query over time attributes instead of the previous settings.

Example:
```
% visit -cli
SetWindowLayout(4)
OpenDatabase("/usr/gapps/visit/data/allinone00.pdb")
AddPlot("Pseudocolor", "mesh/mixvar")
DrawPlots()
qot = GetQueryOverTimeAttributes()
# Make queries over time go to window 4.
qot.createWindow,q.windowId = 0, 4
SetQueryOverTimeAttributes(qot)
QueryOverTime("Min")
# Make queries over time only use half of the number of time states.
qot.endTimeFlag,qot.endTime = 1, GetDatabaseNStates() / 2
SetQueryOverTimeAttributes(qot)
QueryOverTime("Min")
ResetView()
```

SetRenderingAttributes: sets global rendering attributes that control the look and feel of the

Synopsis:
`SetRenderingAttributes(atts) -> integer`

Arguments:
`atts` A RenderingAttributes object that contains the rendering attributes thatwe want to make VisIt use.

Returns:
The SetRenderingAttributes function returns 1 on success and 0 on failure.

Description:
The SetRenderingAttributes makes VisIt use the rendering attributes stored in the specified RenderingAttributes object. The RenderingAttributes object stores rendering attributes such as: scalable rendering options, shadows, specular highlights, display lists, etc.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Surface", "hgslice")
DrawPlots()
v = GetView2D()
v.viewNormal = (-0.215934, -0.454611, 0.864119)
v.viewUp = (0.973938, -0.163188, 0.157523)
v.imageZoom = 1.64765
SetView3D(v)
light = GetLight(0)
light.direction = (0,1,-1)
SetLight(0, light)
r = GetRenderingAttributes()
print r
r.scalableActivationMode = r.Always
r.doShadowing = 1
SetRenderingAttributes(r)
```

SetSaveWindowAttributes: set the attributes used to save windows.

Synopsis:
`SetSaveWindowAttributes(atts)`

Arguments:
`atts` A SaveWindowAttributes object.

Returns:
The SetSaveWindowAttributes object does not return a value.

Description:
The SetSaveWindowAttributes function sets the format and filename that are used to save windows when the SaveWindow function is called. The contents of the active visualization window can be saved as TIFF, JPEG, RGB, PPM, PNG images or they can be saved as curve, Alias Wavefront Obj, or VTK geometry files. To set the SaveWindowAttributes, create a SaveWindowAttributes object using the SaveWindowAttributes function and assign it into a variable. Set the fields in the object and pass it to the SetSaveWindowAttributes function.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/curv3d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
# Set the save window attributes
s = SaveWindowAttributes()
s.fileName = "test"
s.format = s.JPEG
s.progressive = 1
s.fileName = "test"
SetSaveWindowAttributes(s)
# Save the window
SaveWindow()
```

SetTimeSliderState: sets the time state for the active time slider.

Synopsis:
`SetTimeSliderState(state) -> integer`

Arguments:
`state` A zero−based integer containing the time state that we want to make active.

Returns:
The SetTimeSliderState function returns 1 on success and 0 on failure.

Description:
The SetTimeSliderState function sets the time state for the active time slider. This is the function to use if you want to animate through time or change the current keyframe frame.

Example:
```
% visit -cli
path = "/usr/gapps/visit/data/"
dbs = (path + "dbA00.pdb", path + "dbB00.pdb", path + "dbC00.pdb")
for db in dbs:
OpenDatabase(db)
AddPlot("FilledBoundary", "material(mesh)")
DrawPlots()
CreateDatabaseCorrelation("common", dbs, 1)
tsNames = GetWindowInformation().timeSliders
for ts in tsNames:
SetActiveTimeSlider(ts)
for state in list(range(TimeSliderGetNStates())) + [0]:
SetTimeSliderState(state)
```

SetTreatAllDBsAsTimeVarying: sets global attribute boolean to treat all databases as time varying.

Synopsis:
`SetTreatAllDBsAsTimeVarying(val) -> integer`

Arguments:
`val` Either a zero (false) or non−zero (true) integer value to indicate ifall databases should be treated as time varying (true) or not (false).

Returns:
The SetTreatAllDBsAsTimeVarying function returns 1 on success and 0 on failure.

Description:
The SetTreatAllDBsAsTimeVarying function sets a boolean in the global attributes indicating if all databases should be treated as time varying or not. Ordinarily, VisIt tries to minimize file I/O and database interaction by avoiding re−reading metadata that is 'time−invariant' and, therefore, assumed to be the same in a database from one time step to the next. However, sometimes, portions of the metadata, such as the list of variable names and/or number of domains, does in fact vary. In this case, VisIt can actually fail to acknowledge the existence of new variables in the file. Turning this feature on forces VisIt to re−read metadata each time the time−state is changed.

Example:
```
% visit -cli
SetTreatAllDBsAsTimeVarying(1) # turn this feature on
SetTreatAllDBsAsTimeVarying(0) # turn this feature off
```

SetTryHarderCyclesTimes: sets global attribute boolean to force VisIt to read all cycles/times.

Synopsis:
`SetTryHarderCyclesTimes(val) -> integer`

Arguments:
`val` Either a zero (false) or non−zero (true) integer value to indicate ifVisIt read cycle/time information for all timestates when opening a database.

Returns:
The SetTryHarderCyclesTimes function returns 1 on success and 0 on failure.

Description:
For certain classes of databases, obtaining cycle/time information for all time states in the database is an expensive operation, requiring each file to be opened and queried. The cost of the operation gets worse the more time states there are in the database. Ordinarily, VisIt does not bother to query each time state for precise cycle/time information. In fact, often VisIt can guess this information from the filename(s) comprising the databse. However, turning this feature on will force VisIt to obtain accurate cycle/time information for all time states by opening and querying all file(s) in the database.

Example:
```
% visit -cli
SetTryHarderCyclesTimes(1) # Turn this feature on
SetTryHarderCyclesTimes(0) # Turn this feature off
```

SetUltraScript: set the path to the script to be used by the LoadUltra command.

Synopsis:
`SetUltraScript(filename) -> integer`

Arguments:

`filename`	The name of the file to be used as the ultra script when LoadUltra is called.

Returns:
The SetUltraScript function returns 1.

Description:
Set the path to the script to be used by the LoadUltra command. Normal users do not need to use this function.

SetView2D: sets the view for the active visualization window.

Synopsis:
```
SetViewCurve(ViewCurveAttributes) -> integer
SetView2D(View2DAttributes) -> integer
SetView3D(View3DAttributes) -> integer
SetViewAxisArray(ViewAxisArrayAttributes) -> integer
```

Arguments:
`view` A ViewAttributes object containing the view.

Returns:
All functions returns 1 on success and 0 on failure.

Description:
The view is a crucial part of a visualization since it determines which parts of the database are examined. The VisIt Python Interface provides four functions for setting the view: SetView2D, SetView3D, SetViewCurve, and SetViewAxisArray. If the visualization window contains 2D plots, use the SetView2D function. Use the SetView3D function when the visualization window contains 3D plots. Similarly for windows containing curve or axis−array based plots. To set the view, first create the appropriate ViewAttributes object and set the object's fields to set a new view. After setting the fields, pass the object to the matching SetView function. A common use of the SetView functions is to animate the view to produce simple animations where the camera appears to fly around the plots in the visualization window. A View3D object also supports the RotateAxis(int axis, double deg) method which mimics the 'rotx', 'roty' and 'rotz' view commands in the GUI.

Examples:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "v")
DrawPlots()
va = GetView3D()
va.RotateAxis(1,30.0) # rotate around the y axis 30 degrees.
SetView3D(va)
v0 = GetView3D()
v1 = GetView3D()
v1.camera,v1.viewUp = (1,1,1),(-1,1,-1)
v1.parallelScale = 10.
for i in range(0,20):
t = float(i) / 19.
v2 = (1. - t) * v0 + t * v1
SetView3D(v2) # Animate the view.
```

SetView3D: sets the view for the active visualization window.

Synopsis:
```
SetViewCurve(ViewCurveAttributes) -> integer
SetView2D(View2DAttributes) -> integer
SetView3D(View3DAttributes) -> integer
SetViewAxisArray(ViewAxisArrayAttributes) -> integer
```

Arguments:
`view` A ViewAttributes object containing the view.

Returns:
All functions returns 1 on success and 0 on failure.

Description:
The view is a crucial part of a visualization since it determines which parts of the database are examined. The VisIt Python Interface provides four functions for setting the view: SetView2D, SetView3D, SetViewCurve, and SetViewAxisArray. If the visualization window contains 2D plots, use the SetView2D function. Use the SetView3D function when the visualization window contains 3D plots. Similarly for windows containing curve or axis−array based plots. To set the view, first create the appropriate ViewAttributes object and set the object's fields to set a new view. After setting the fields, pass the object to the matching SetView function. A common use of the SetView functions is to animate the view to produce simple animations where the camera appears to fly around the plots in the visualization window. A View3D object also supports the RotateAxis(int axis, double deg) method which mimics the 'rotx', 'roty' and 'rotz' view commands in the GUI.

Examples:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "v")
DrawPlots()
va = GetView3D()
va.RotateAxis(1,30.0) # rotate around the y axis 30 degrees.
SetView3D(va)
v0 = GetView3D()
v1 = GetView3D()
v1.camera,v1.viewUp = (1,1,1),(-1,1,-1)
v1.parallelScale = 10.
for i in range(0,20):
t = float(i) / 19.
v2 = (1. - t) * v0 + t * v1
SetView3D(v2) # Animate the view.
```

SetViewAxisArray: sets the view for the active visualization window.

Synopsis:
```
SetViewCurve(ViewCurveAttributes) -> integer
SetView2D(View2DAttributes) -> integer
SetView3D(View3DAttributes) -> integer
SetViewAxisArray(ViewAxisArrayAttributes) -> integer
```

Arguments:
`view` A ViewAttributes object containing the view.

Returns:
All functions returns 1 on success and 0 on failure.

Description:
The view is a crucial part of a visualization since it determines which parts of the database are examined. The VisIt Python Interface provides four functions for setting the view: SetView2D, SetView3D, SetViewCurve, and SetViewAxisArray. If the visualization window contains 2D plots, use the SetView2D function. Use the SetView3D function when the visualization window contains 3D plots. Similarly for windows containing curve or axis−array based plots. To set the view, first create the appropriate ViewAttributes object and set the object's fields to set a new view. After setting the fields, pass the object to the matching SetView function. A common use of the SetView functions is to animate the view to produce simple animations where the camera appears to fly around the plots in the visualization window. A View3D object also supports the RotateAxis(int axis, double deg) method which mimics the 'rotx', 'roty' and 'rotz' view commands in the GUI.

Examples:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "v")
DrawPlots()
va = GetView3D()
va.RotateAxis(1,30.0) # rotate around the y axis 30 degrees.
SetView3D(va)
v0 = GetView3D()
v1 = GetView3D()
v1.camera,v1.viewUp = (1,1,1),(-1,1,-1)
v1.parallelScale = 10.
for i in range(0,20):
t = float(i) / 19.
v2 = (1. - t) * v0 + t * v1
SetView3D(v2) # Animate the view.
```

SetViewCurve: sets the view for the active visualization window.

Synopsis:
```
SetViewCurve(ViewCurveAttributes) -> integer
SetView2D(View2DAttributes) -> integer
SetView3D(View3DAttributes) -> integer
SetViewAxisArray(ViewAxisArrayAttributes) -> integer
```

Arguments:
`view` A ViewAttributes object containing the view.

Returns:
All functions returns 1 on success and 0 on failure.

Description:
The view is a crucial part of a visualization since it determines which parts of the database are examined. The VisIt Python Interface provides four functions for setting the view: SetView2D, SetView3D, SetViewCurve, and SetViewAxisArray. If the visualization window contains 2D plots, use the SetView2D function. Use the SetView3D function when the visualization window contains 3D plots. Similarly for windows containing curve or axis−array based plots. To set the view, first create the appropriate ViewAttributes object and set the object's fields to set a new view. After setting the fields, pass the object to the matching SetView function. A common use of the SetView functions is to animate the view to produce simple animations where the camera appears to fly around the plots in the visualization window. A View3D object also supports the RotateAxis(int axis, double deg) method which mimics the 'rotx', 'roty' and 'rotz' view commands in the GUI.

Examples:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "v")
DrawPlots()
va = GetView3D()
va.RotateAxis(1,30.0) # rotate around the y axis 30 degrees.
SetView3D(va)
v0 = GetView3D()
v1 = GetView3D()
v1.camera,v1.viewUp = (1,1,1),(-1,1,-1)
v1.parallelScale = 10.
for i in range(0,20):
t = float(i) / 19.
v2 = (1. - t) * v0 + t * v1
SetView3D(v2) # Animate the view.
```

SetViewExtentsType: tells VisIt how to use extents when computing the view.

Synopsis:
`SetViewExtentsType(type) -> integer`

Arguments:
`type` An integer 0, 1 or one of the strings: "original", "actual".

Returns:
SetViewExtentsType returns 1 on success and 0 on failure.

Description:
VisIt can use a plot's spatial extents in two ways when computing the view. The first way of using the extents is to use the "original" extents, which are the spatial extents before any modifications, such as subset selection, have been made to the plot. This ensures that the view will remain relatively constant for a plot. Alternatively, you can use the "actual" extents, which are the spatial extents of the pieces of the plot that remain after operations such as subset selection.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
SetViewExtentsType("actual")
AddPlot("FilledBoundary", "mat1")
DrawPlots()
v = GetView3D()
v.viewNormal = (-0.618945, 0.450655, 0.643286)
v.viewUp = (0.276106, 0.891586, -0.358943)
SetView3D(v)
mats = GetMaterials()
nmats = len(mats):
# Turn off all but the last material in sequence and watch
# the view update each time.
for i in range(nmats-1):
index = nmats-1-i
TurnMaterialsOff(mats[index])
SaveWindow()
SetViewExtentsType("original")
```

SetViewKeyframe: adds a view keyframe.

Synopsis:
`SetViewKeyframe() -> integer`

Returns:
The SetViewKeyframe function returns 1 on success and 0 on failure.

Description:
The SetViewKeyframe function adds a view keyframe when VisIt is in keyframing mode. View keyframes are used to set the view at crucial points during an animation. Frames that lie between view keyframes have an interpolated view that is based on the view keyframes. You can use the SetViewKeyframe function to create complex camera animations that allow you to fly around (or through) your visualization.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Contour", "hardyglobal")
DrawPlots()
k = GetKeyframeAttributes()
nFrames = 20
k.enabled, k.nFrames, k.nFramesWasUserSet = 1, nFrames, 1
SetKeyframeAttributes(k)
SetPlotFrameRange(0, 0, nFrames-1)
SetViewKeyframe()
SetTimeSliderState(10)
v = GetView3D()
v.viewNormal = (-0.721721, 0.40829, 0.558944)
v.viewUp = (0.294696, 0.911913, -0.285604)
SetView3D(v)
SetViewKeyframe()
SetTimeSliderState(nFrames-1)
v.viewNormal = (-0.74872, 0.423588, -0.509894)
v.viewUp = (0.369095, 0.905328, 0.210117)
SetView3D()
SetViewKeyframe()
ToggleCameraViewMode()
for state in range(TimeSliderGetNStates()):
SetTimeSliderState(state)
SaveWindow()
```

SetWindowArea: set the screen area devoted to visualization windows.

Synopsis:
`SetWindowArea(x, y, width, height) -> integer`

Arguments:

`x`	Left X coordinate in screen pixels.
`y`	Top Y coordinate in screen pixels.
`width`	Width of the window area in pixels.
`height`	Height of the window area in pixels.

Returns:
The SetWindowArea function returns 1 on success and 0 on failure.

Description:
The SetWindowArea method sets the area of the screen that can be used by VisIt's visualization windows. This is useful for making sure windows are a certain size when running a Python script.

Example:
```
import visit
visit.Launch()
visit.SetWindowArea(0, 0, 600, 600)
visit.SetWindowLayout(4)
```

SetWindowLayout: sets the window layout

Synopsis:
`SetWindowLayout(layout) -> integer`

Arguments:

`layout`	An integer that specifies the window layout. (1,2,4,8,9,16 are valid)

Returns:
The SetWindowLayout function returns an integer value of 1 for success and 0 for failure.

Description:
VisIt's visualization windows can be arranged in various tiled patterns that allow VisIt to make good use of the screen while displaying several visualization windows. The window layout determines how windows are shown on the screen. The SetWindowLayout function sets the window layout. The layout argument is an integer value equal to 1,2,4,8,9, or 16.

Example:
```
% visit -cli
SetWindowLayout(2) # switch to 1x2 layout
SetWindowLayout(4) # switch to 2x2 layout
SetWindowLayout(8) # switch to 2x4 layout
```

SetWindowMode: sets the window mode of the active visualization window.

Synopsis:
`SetWindowMode(mode) -> integer`

Arguments:
`mode` A string containing the new mode: 'navigate', 'zoom', 'lineout', 'pick','zone pick', 'node pick', 'spreadsheet pick'.

Returns:
The SetWindowMode function returns 1 on success and 0 on failure.

Description:
VisIt's visualization windows have various window modes that alter their behavior. Most of the time a visualization window is in "navigate" mode which changes the view when the mouse is moved in the window. The "zoom" mode allows a zoom rectangle to be drawn in the window for changing the view. The "pick" mode retrieves information about the plots when the mouse is clicked in the window. The "lineout" mode allows the user to draw lines which produce curve plots.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/curv2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
SetWindowMode("zoom")
# Draw a rectangle in the visualization window to zoom the plots
```

ShowAllWindows: tells VisIt to show its visualization windows.

Synopsis:
`ShowAllWindows() -> integer`

Returns:
The ShowAllWindows function returns 1 on success and 0 on failure.

Description:
The ShowAllWindows function tells VisIt's viewer to show all of its visualization windows. The command line interface calls ShowAllWindows before giving control to any user−supplied script to ensure that the visualization windows appear as expected. Call the ShowAllWindows function when using the VisIt module inside another Python interpreter so the visualization windows are made visible.

Example:
```
% python
import visit
visit.Launch()
visit.ShowAllWindows()
```

ShowToolbars: shows the visualization window's toolbars.

Synopsis:
```
ShowToolbars() -> integer
ShowToolbars(allWindows) ->integer
```

Arguments:

`allWindows`	An integer value that tells VisIt to show the toolbars for all windowswhen it is non−zero.

Returns:
The ShowToolbars function returns 1 on success and 0 on failure.

Description:
The ShowToolbars function tells VisIt to show the toolbars for the active visualization window or for all visualization windows when the optional allWindows argument is provided and is set to a non−zero value.

Example:
```
% visit -cli
SetWindowLayout(4)
HideToolbars(1)
ShowToolbars()
# Show the toolbars for all windows.
ShowToolbars(1)
```

Source: executes the specified Python script

Synopsis:
`Source(filename)`

Returns:
The Source function does not return a value.

Description:
The Source function reads in the contents of a text file and interprets it with the Python interpreter. This is a simple mechanism that allows simple scripts to be included in larger scripts. The Source function takes a single string argument that contains the name of the script to execute.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
# include another script that does some animation.
Source("Animate.py")
```

SuppressMessages: suppresses the printing of Error, Warning, Message messages

Synopsis:
`SuppressMessages(int level) ->integer`

Arguments:
`int level` An integer value of 1,2,3 or 4

Returns:
The SuppressMessages function returns the previous suppression level on success and 0 on failure.

Description:
The SuppressMessage function sets the supression level for status messages generated by VisIt. A value of 1 suppresses all types of messages. A value of 2 suppresses Warnings and Messages but does NOT suppress Errors. A value of 3 suppresses Messages but does not suppress Warnings or Errors. A value of 4 does not suppress any messages. The default setting is 4.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/rect2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
# Turn off Warning and Message messages.
SuppressMessages(2)
SaveWindow()
```

SuppressQueryOutputOff: suppresses the printing of QueryOutput

Synopsis:
```
SuppressQueryOutputOn() ->integer
SuppressQueryOutputOff() ->integer
```

Arguments:
None

Returns:
The SuppressQueryOutput function returns 1 on success and 0 on failure.

Description:
The SuppressQueryOutput function tells VisIt to turn on/off the automatic printing of query output. Query output will still be available via GetQueryOutputString and GetQueryOutputValue.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/rect2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
# Turn off automatic printing of Query output.
SuppressQueryOutputOn()
Query("MinMax")
print "The min is: %g and the max is: %g" % GetQueryOutputValue()
# Turn on automatic printing of Query output.
SuppressQueryOutputOff()
Query("MinMax")
```

SuppressQueryOutputOn: suppresses the printing of QueryOutput

Synopsis:
```
SuppressQueryOutputOn() ->integer
SuppressQueryOutputOff() ->integer
```

Arguments:
None

Returns:
The SuppressQueryOutput function returns 1 on success and 0 on failure.

Description:
The SuppressQueryOutput function tells VisIt to turn on/off the automatic printing of query output. Query output will still be available via GetQueryOutputString and GetQueryOutputValue.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/rect2d.silo")
AddPlot("Pseudocolor", "d")
DrawPlots()
# Turn off automatic printing of Query output.
SuppressQueryOutputOn()
Query("MinMax")
print "The min is: %g and the max is: %g" % GetQueryOutputValue()
# Turn on automatic printing of Query output.
SuppressQueryOutputOff()
Query("MinMax")
```

TimeSliderGetNStates: returns the number of time states for the active time slider.

Synopsis:
`TimeSliderGetNStates() -> integer`

Returns:
Returns an integer containing the number of time states for the current time slider.

Description:
The TimeSliderGetNStates function returns the number of time states for the active time slider. Remember that the length of the time slider does not have to be equal to the number of time states in a time−varying database because of database correlations and keyframing. If you want to iterate through time, use this function to determine the number of iterations that are required to reach the end of the active time slider.

Example:
```
OpenDatabase("/usr/gapps/visit/data/wave.visit")
AddPlot("Pseudocolor", "pressure")
DrawPlots()
for state in range(TimeSliderGetNStates()):
    SetTimeSliderState(state)
    SaveWindow()
```

TimeSliderNextState: advances the active time slider to the next state.

Synopsis:
`TimeSliderNextState() -> integer`

Returns:
The TimeSliderNextState function returns 1 on success and 0 on failure.

Description:
The TimeSliderNextState function advances the active time slider to the next time slider state.

Example:
```
# Assume that files are being written to the disk.
% visit -cli
OpenDatabase("dynamic*.silo database")
AddPlot("Pseudocolor", "var")
AddPlot("Mesh", "mesh")
DrawPlots()
SetTimeSliderState(TimeSliderGetNStates() - 1)
while 1:
    SaveWindow()
    TimeSliderPreviousState()
```

TimeSliderPreviousState: moves the active time slider to the previous time state.

Synopsis:
`TimeSliderPreviousState() -> integer`

Returns:
The TimeSliderPreviousState function returns 1 on success and 0 on failure.

Description:
The TimeSliderPreviousState function moves the active time slider to the previous time slider state.

Example:
```
# Assume that files are being written to the disk.
% visit -cli
OpenDatabase("dynamic*.silo database")
AddPlot("Pseudocolor", "var")
AddPlot("Mesh", "mesh")
DrawPlots()
while 1:
    TimeSliderNextState()
    SaveWindow()
```

TimeSliderSetState: sets the time state for the active time slider.

Synopsis:
`SetTimeSliderState(state) -> integer`

Arguments:
`state` A zero−based integer containing the time state that we want to make active.

Returns:
The SetTimeSliderState function returns 1 on success and 0 on failure.

Description:
The SetTimeSliderState function sets the time state for the active time slider. This is the function to use if you want to animate through time or change the current keyframe frame.

Example:
```
% visit -cli
path = "/usr/gapps/visit/data/"
dbs = (path + "dbA00.pdb", path + "dbB00.pdb", path + "dbC00.pdb")
for db in dbs:
OpenDatabase(db)
AddPlot("FilledBoundary", "material(mesh)")
DrawPlots()
CreateDatabaseCorrelation("common", dbs, 1)
tsNames = GetWindowInformation().timeSliders
for ts in tsNames:
SetActiveTimeSlider(ts)
for state in list(range(TimeSliderGetNStates())) + [0]:
SetTimeSliderState(state)
```

ToggleBoundingBoxMode: toggle a visualization window mode

Synopsis:
```
ToggleBoundingBoxMode() -> integer
ToggleCameraViewMode() -> integer
ToggleFullFrameMode() -> integer
ToggleLockTime() -> integer
ToggleLockViewMode() -> integer
ToggleMaintainDataMode() -> integer
ToggleMaintainViewMode() -> integer
ToggleSpinMode() -> integer
```

Returns:
All functions return 1 on success and 0 on failure.

Description:
The visualization window has various modes that affect its behavior and the VisIt Python Interface provides a few functions to toggle some of those modes. The ToggleBoundingBoxMode function toggles bounding box mode on and off. When the visualization window is in bounding box mode, any plots it contains are hidden while the view is being changed so the window redraws faster. The ToggleCameraViewMode function toggles camera view mode on and off. When the visualization window is in camera view mode, the view is updated using any view keyframes that have been defined when VisIt is in keyframing mode. The ToggleFullFrameMode function toggles fullframe mode on and off. When the visualization window is in fullframe mode, the viewport is stretched non−uniformly so that it covers most of the visualization window. While not maintaining a 1:1 aspect ratio, it does make better use of the visualization window. The ToggleLockTime function turns time locking on and off in a visualization window. When time locking is on in a visualization window, VisIt creates a database correlation that works for the databases in all visualization windows that are time−locked. When you change the time state using the time slider for the the afore−mentioned database correlation, it has the effect of updating time in all time−locked visualization windows. The ToggleLockViewMode function turns lock view mode on and off. When windows are in lock view mode, each view change is broadcast to other windows that are also in lock view mode. This allows windows containing similar plots to be compared easily. The ToggleMaintainDataMode and ToggleMaintainViewMode functions force the data range and the view, respectively, that was in effect when the mode was toggled to be used for all subsequent time states. The ToggleSpinMode function turns spin mode on and off. When the visualization window is in spin mode, it continues to spin along the axis of rotation when the view is changed interactively.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
# Turn on spin mode.
ToggleSpinMode()
# Rotate the plot interactively using the mouse and watch it keep spinning
after the mouse release.
# Turn off spin mode.
ToggleSpinMode()
```

ToggleCameraViewMode: toggle a visualization window mode

Synopsis:
```
ToggleBoundingBoxMode() -> integer
ToggleCameraViewMode() -> integer
ToggleFullFrameMode() -> integer
ToggleLockTime() -> integer
ToggleLockViewMode() -> integer
ToggleMaintainDataMode() -> integer
ToggleMaintainViewMode() -> integer
ToggleSpinMode() -> integer
```

Returns:
All functions return 1 on success and 0 on failure.

Description:
The visualization window has various modes that affect its behavior and the VisIt Python Interface provides a few functions to toggle some of those modes. The ToggleBoundingBoxMode function toggles bounding box mode on and off. When the visualization window is in bounding box mode, any plots it contains are hidden while the view is being changed so the window redraws faster. The ToggleCameraViewMode function toggles camera view mode on and off. When the visualization window is in camera view mode, the view is updated using any view keyframes that have been defined when VisIt is in keyframing mode. The ToggleFullFrameMode function toggles fullframe mode on and off. When the visualization window is in fullframe mode, the viewport is stretched non−uniformly so that it covers most of the visualization window. While not maintaining a 1:1 aspect ratio, it does make better use of the visualization window. The ToggleLockTime function turns time locking on and off in a visualization window. When time locking is on in a visualization window, VisIt creates a database correlation that works for the databases in all visualization windows that are time−locked. When you change the time state using the time slider for the the afore−mentioned database correlation, it has the effect of updating time in all time−locked visualization windows. The ToggleLockViewMode function turns lock view mode on and off. When windows are in lock view mode, each view change is broadcast to other windows that are also in lock view mode. This allows windows containing similar plots to be compared easily. The ToggleMaintainDataMode and ToggleMaintainViewMode functions force the data range and the view, respectively, that was in effect when the mode was toggled to be used for all subsequent time states. The ToggleSpinMode function turns spin mode on and off. When the visualization window is in spin mode, it continues to spin along the axis of rotation when the view is changed interactively.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
# Turn on spin mode.
ToggleSpinMode()
# Rotate the plot interactively using the mouse and watch it keep spinning
after the mouse release.
# Turn off spin mode.
ToggleSpinMode()
```

ToggleFullFrameMode: toggle a visualization window mode

Synopsis:
```
ToggleBoundingBoxMode() -> integer
ToggleCameraViewMode() -> integer
ToggleFullFrameMode() -> integer
ToggleLockTime() -> integer
ToggleLockViewMode() -> integer
ToggleMaintainDataMode() -> integer
ToggleMaintainViewMode() -> integer
ToggleSpinMode() -> integer
```

Returns:
All functions return 1 on success and 0 on failure.

Description:
The visualization window has various modes that affect its behavior and the VisIt Python Interface provides a few functions to toggle some of those modes. The ToggleBoundingBoxMode function toggles bounding box mode on and off. When the visualization window is in bounding box mode, any plots it contains are hidden while the view is being changed so the window redraws faster. The ToggleCameraViewMode function toggles camera view mode on and off. When the visualization window is in camera view mode, the view is updated using any view keyframes that have been defined when VisIt is in keyframing mode. The ToggleFullFrameMode function toggles fullframe mode on and off. When the visualization window is in fullframe mode, the viewport is stretched non−uniformly so that it covers most of the visualization window. While not maintaining a 1:1 aspect ratio, it does make better use of the visualization window. The ToggleLockTime function turns time locking on and off in a visualization window. When time locking is on in a visualization window, VisIt creates a database correlation that works for the databases in all visualization windows that are time−locked. When you change the time state using the time slider for the the afore−mentioned database correlation, it has the effect of updating time in all time−locked visualization windows. The ToggleLockViewMode function turns lock view mode on and off. When windows are in lock view mode, each view change is broadcast to other windows that are also in lock view mode. This allows windows containing similar plots to be compared easily. The ToggleMaintainDataMode and ToggleMaintainViewMode functions force the data range and the view, respectively, that was in effect when the mode was toggled to be used for all subsequent time states. The ToggleSpinMode function turns spin mode on and off. When the visualization window is in spin mode, it continues to spin along the axis of rotation when the view is changed interactively.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
# Turn on spin mode.
ToggleSpinMode()
# Rotate the plot interactively using the mouse and watch it keep spinning
after the mouse release.
# Turn off spin mode.
ToggleSpinMode()
```

ToggleLockTime: toggle a visualization window mode

Synopsis:
```
ToggleBoundingBoxMode() -> integer
ToggleCameraViewMode() -> integer
ToggleFullFrameMode() -> integer
ToggleLockTime() -> integer
ToggleLockViewMode() -> integer
ToggleMaintainDataMode() -> integer
ToggleMaintainViewMode() -> integer
ToggleSpinMode() -> integer
```

Returns:
All functions return 1 on success and 0 on failure.

Description:
The visualization window has various modes that affect its behavior and the VisIt Python Interface provides a few functions to toggle some of those modes. The ToggleBoundingBoxMode function toggles bounding box mode on and off. When the visualization window is in bounding box mode, any plots it contains are hidden while the view is being changed so the window redraws faster. The ToggleCameraViewMode function toggles camera view mode on and off. When the visualization window is in camera view mode, the view is updated using any view keyframes that have been defined when VisIt is in keyframing mode. The ToggleFullFrameMode function toggles fullframe mode on and off. When the visualization window is in fullframe mode, the viewport is stretched non−uniformly so that it covers most of the visualization window. While not maintaining a 1:1 aspect ratio, it does make better use of the visualization window. The ToggleLockTime function turns time locking on and off in a visualization window. When time locking is on in a visualization window, VisIt creates a database correlation that works for the databases in all visualization windows that are time−locked. When you change the time state using the time slider for the the afore−mentioned database correlation, it has the effect of updating time in all time−locked visualization windows. The ToggleLockViewMode function turns lock view mode on and off. When windows are in lock view mode, each view change is broadcast to other windows that are also in lock view mode. This allows windows containing similar plots to be compared easily. The ToggleMaintainDataMode and ToggleMaintainViewMode functions force the data range and the view, respectively, that was in effect when the mode was toggled to be used for all subsequent time states. The ToggleSpinMode function turns spin mode on and off. When the visualization window is in spin mode, it continues to spin along the axis of rotation when the view is changed interactively.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
# Turn on spin mode.
ToggleSpinMode()
# Rotate the plot interactively using the mouse and watch it keep spinning
after the mouse release.
# Turn off spin mode.
ToggleSpinMode()
```

ToggleLockTools: toggle a visualization window mode

Synopsis:
```
ToggleBoundingBoxMode() -> integer
ToggleCameraViewMode() -> integer
ToggleFullFrameMode() -> integer
ToggleLockTime() -> integer
ToggleLockViewMode() -> integer
ToggleMaintainDataMode() -> integer
ToggleMaintainViewMode() -> integer
ToggleSpinMode() -> integer
```

Returns:
All functions return 1 on success and 0 on failure.

Description:
The visualization window has various modes that affect its behavior and the VisIt Python Interface provides a few functions to toggle some of those modes. The ToggleBoundingBoxMode function toggles bounding box mode on and off. When the visualization window is in bounding box mode, any plots it contains are hidden while the view is being changed so the window redraws faster. The ToggleCameraViewMode function toggles camera view mode on and off. When the visualization window is in camera view mode, the view is updated using any view keyframes that have been defined when VisIt is in keyframing mode. The ToggleFullFrameMode function toggles fullframe mode on and off. When the visualization window is in fullframe mode, the viewport is stretched non−uniformly so that it covers most of the visualization window. While not maintaining a 1:1 aspect ratio, it does make better use of the visualization window. The ToggleLockTime function turns time locking on and off in a visualization window. When time locking is on in a visualization window, VisIt creates a database correlation that works for the databases in all visualization windows that are time−locked. When you change the time state using the time slider for the the afore−mentioned database correlation, it has the effect of updating time in all time−locked visualization windows. The ToggleLockViewMode function turns lock view mode on and off. When windows are in lock view mode, each view change is broadcast to other windows that are also in lock view mode. This allows windows containing similar plots to be compared easily. The ToggleMaintainDataMode and ToggleMaintainViewMode functions force the data range and the view, respectively, that was in effect when the mode was toggled to be used for all subsequent time states. The ToggleSpinMode function turns spin mode on and off. When the visualization window is in spin mode, it continues to spin along the axis of rotation when the view is changed interactively.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
# Turn on spin mode.
ToggleSpinMode()
# Rotate the plot interactively using the mouse and watch it keep spinning
after the mouse release.
# Turn off spin mode.
ToggleSpinMode()
```

ToggleLockViewMode: toggle a visualization window mode

Synopsis:
```
ToggleBoundingBoxMode() -> integer
ToggleCameraViewMode() -> integer
ToggleFullFrameMode() -> integer
ToggleLockTime() -> integer
ToggleLockViewMode() -> integer
ToggleMaintainDataMode() -> integer
ToggleMaintainViewMode() -> integer
ToggleSpinMode() -> integer
```

Returns:
All functions return 1 on success and 0 on failure.

Description:
The visualization window has various modes that affect its behavior and the VisIt Python Interface provides a few functions to toggle some of those modes. The ToggleBoundingBoxMode function toggles bounding box mode on and off. When the visualization window is in bounding box mode, any plots it contains are hidden while the view is being changed so the window redraws faster. The ToggleCameraViewMode function toggles camera view mode on and off. When the visualization window is in camera view mode, the view is updated using any view keyframes that have been defined when VisIt is in keyframing mode. The ToggleFullFrameMode function toggles fullframe mode on and off. When the visualization window is in fullframe mode, the viewport is stretched non−uniformly so that it covers most of the visualization window. While not maintaining a 1:1 aspect ratio, it does make better use of the visualization window. The ToggleLockTime function turns time locking on and off in a visualization window. When time locking is on in a visualization window, VisIt creates a database correlation that works for the databases in all visualization windows that are time−locked. When you change the time state using the time slider for the the afore−mentioned database correlation, it has the effect of updating time in all time−locked visualization windows. The ToggleLockViewMode function turns lock view mode on and off. When windows are in lock view mode, each view change is broadcast to other windows that are also in lock view mode. This allows windows containing similar plots to be compared easily. The ToggleMaintainDataMode and ToggleMaintainViewMode functions force the data range and the view, respectively, that was in effect when the mode was toggled to be used for all subsequent time states. The ToggleSpinMode function turns spin mode on and off. When the visualization window is in spin mode, it continues to spin along the axis of rotation when the view is changed interactively.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
# Turn on spin mode.
ToggleSpinMode()
# Rotate the plot interactively using the mouse and watch it keep spinning
after the mouse release.
# Turn off spin mode.
ToggleSpinMode()
```

ToggleMaintainViewMode: toggle a visualization window mode

Synopsis:
```
ToggleBoundingBoxMode() -> integer
ToggleCameraViewMode() -> integer
ToggleFullFrameMode() -> integer
ToggleLockTime() -> integer
ToggleLockViewMode() -> integer
ToggleMaintainDataMode() -> integer
ToggleMaintainViewMode() -> integer
ToggleSpinMode() -> integer
```

Returns:
All functions return 1 on success and 0 on failure.

Description:
The visualization window has various modes that affect its behavior and the VisIt Python Interface provides a few functions to toggle some of those modes. The ToggleBoundingBoxMode function toggles bounding box mode on and off. When the visualization window is in bounding box mode, any plots it contains are hidden while the view is being changed so the window redraws faster. The ToggleCameraViewMode function toggles camera view mode on and off. When the visualization window is in camera view mode, the view is updated using any view keyframes that have been defined when VisIt is in keyframing mode. The ToggleFullFrameMode function toggles fullframe mode on and off. When the visualization window is in fullframe mode, the viewport is stretched non−uniformly so that it covers most of the visualization window. While not maintaining a 1:1 aspect ratio, it does make better use of the visualization window. The ToggleLockTime function turns time locking on and off in a visualization window. When time locking is on in a visualization window, VisIt creates a database correlation that works for the databases in all visualization windows that are time−locked. When you change the time state using the time slider for the the afore−mentioned database correlation, it has the effect of updating time in all time−locked visualization windows. The ToggleLockViewMode function turns lock view mode on and off. When windows are in lock view mode, each view change is broadcast to other windows that are also in lock view mode. This allows windows containing similar plots to be compared easily. The ToggleMaintainDataMode and ToggleMaintainViewMode functions force the data range and the view, respectively, that was in effect when the mode was toggled to be used for all subsequent time states. The ToggleSpinMode function turns spin mode on and off. When the visualization window is in spin mode, it continues to spin along the axis of rotation when the view is changed interactively.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
# Turn on spin mode.
ToggleSpinMode()
# Rotate the plot interactively using the mouse and watch it keep spinning
after the mouse release.
# Turn off spin mode.
ToggleSpinMode()
```

ToggleSpinMode: toggle a visualization window mode

Synopsis:
```
ToggleBoundingBoxMode() -> integer
ToggleCameraViewMode() -> integer
ToggleFullFrameMode() -> integer
ToggleLockTime() -> integer
ToggleLockViewMode() -> integer
ToggleMaintainDataMode() -> integer
ToggleMaintainViewMode() -> integer
ToggleSpinMode() -> integer
```

Returns:
All functions return 1 on success and 0 on failure.

Description:
The visualization window has various modes that affect its behavior and the VisIt Python Interface provides a few functions to toggle some of those modes. The ToggleBoundingBoxMode function toggles bounding box mode on and off. When the visualization window is in bounding box mode, any plots it contains are hidden while the view is being changed so the window redraws faster. The ToggleCameraViewMode function toggles camera view mode on and off. When the visualization window is in camera view mode, the view is updated using any view keyframes that have been defined when VisIt is in keyframing mode. The ToggleFullFrameMode function toggles fullframe mode on and off. When the visualization window is in fullframe mode, the viewport is stretched non−uniformly so that it covers most of the visualization window. While not maintaining a 1:1 aspect ratio, it does make better use of the visualization window. The ToggleLockTime function turns time locking on and off in a visualization window. When time locking is on in a visualization window, VisIt creates a database correlation that works for the databases in all visualization windows that are time−locked. When you change the time state using the time slider for the the afore−mentioned database correlation, it has the effect of updating time in all time−locked visualization windows. The ToggleLockViewMode function turns lock view mode on and off. When windows are in lock view mode, each view change is broadcast to other windows that are also in lock view mode. This allows windows containing similar plots to be compared easily. The ToggleMaintainDataMode and ToggleMaintainViewMode functions force the data range and the view, respectively, that was in effect when the mode was toggled to be used for all subsequent time states. The ToggleSpinMode function turns spin mode on and off. When the visualization window is in spin mode, it continues to spin along the axis of rotation when the view is changed interactively.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
# Turn on spin mode.
ToggleSpinMode()
# Rotate the plot interactively using the mouse and watch it keep spinning
after the mouse release.
# Turn off spin mode.
ToggleSpinMode()
```

TurnDomainsOff: turns materials or domains on or off.

Synopsis:
```
TurnMaterialsOn() -> integer
TurnMaterialsOn(string) -> integer
TurnMaterialsOn(tuple of strings) -> integer
TurnMaterialsOff() -> integer
TurnMaterialsOff(string) -> integer
TurnMaterialsOff(tuple of strings) -> integer
TurnDomainsOn() -> integer
TurnDomainsOn(string) -> integer
TurnDomainsOn(tuple of strings) -> integer
TurnDomainsOff() -> integer
TurnDomainsOff(string) -> integer
TurnDomainsOff(tuple of strings) -> integer
```

Returns:
The Turn functions return an integer with a value of 1 for success or 0 for failure.

Description:
The Turn functions are provided to simplify the removal of material or domain subsets. Instead of creating a SILRestriction object, you can use the Turn functions to turn materials or domains on or off. The TurnMaterialsOn function turns materials on and the TurnMaterialsOff function turns them off. The TurnDomainsOn function turns domains on and the TurnDomainsOff function turns them off. All of the Turn functions have three possible argument lists. When you do not provide any arguments, the function applies to all subsets in the SIL so if you called the TurnMaterialsOff function with no arguments, all materials would be turned off. Calling TurnMaterialsOn with no arguments would turn all materials on. All functions can also take a string argument, which is the name of the set to modify. For example, you could turn off domain 0 by calling the TurnDomainsOff with a single argument of "domain0" (or the appropriate set name). All of the Turn functions can also be used to modify more than one set if you provide a tuple of set names. After you use the Turn functions to change the SIL restriction, you might want to call the ListMaterials or ListDomains functions to make sure that the SIL restriction was actually modified.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
TurnMaterialsOff("4") # Turn off material 4
TurnMaterialsOff(("1", "2")) # Turn off materials 1 and 2
TurnMaterialsOn() # Turn on all materials
```

TurnDomainsOn: turns materials or domains on or off.

Synopsis:
```
TurnMaterialsOn() -> integer
TurnMaterialsOn(string) -> integer
TurnMaterialsOn(tuple of strings) -> integer
TurnMaterialsOff() -> integer
TurnMaterialsOff(string) -> integer
TurnMaterialsOff(tuple of strings) -> integer
TurnDomainsOn() -> integer
TurnDomainsOn(string) -> integer
TurnDomainsOn(tuple of strings) -> integer
TurnDomainsOff() -> integer
TurnDomainsOff(string) -> integer
TurnDomainsOff(tuple of strings) -> integer
```

Returns:
The Turn functions return an integer with a value of 1 for success or 0 for failure.

Description:
The Turn functions are provided to simplify the removal of material or domain subsets. Instead of creating a SILRestriction object, you can use the Turn functions to turn materials or domains on or off. The TurnMaterialsOn function turns materials on and the TurnMaterialsOff function turns them off. The TurnDomainsOn function turns domains on and the TurnDomainsOff function turns them off. All of the Turn functions have three possible argument lists. When you do not provide any arguments, the function applies to all subsets in the SIL so if you called the TurnMaterialsOff function with no arguments, all materials would be turned off. Calling TurnMaterialsOn with no arguments would turn all materials on. All functions can also take a string argument, which is the name of the set to modify. For example, you could turn off domain 0 by calling the TurnDomainsOff with a single argument of "domain0" (or the appropriate set name). All of the Turn functions can also be used to modify more than one set if you provide a tuple of set names. After you use the Turn functions to change the SIL restriction, you might want to call the ListMaterials or ListDomains functions to make sure that the SIL restriction was actually modified.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
TurnMaterialsOff("4") # Turn off material 4
TurnMaterialsOff(("1", "2")) # Turn off materials 1 and 2
TurnMaterialsOn() # Turn on all materials
```

TurnMaterialsOff: turns materials or domains on or off.

Synopsis:
```
TurnMaterialsOn() -> integer
TurnMaterialsOn(string) -> integer
TurnMaterialsOn(tuple of strings) -> integer
TurnMaterialsOff() -> integer
TurnMaterialsOff(string) -> integer
TurnMaterialsOff(tuple of strings) -> integer
TurnDomainsOn() -> integer
TurnDomainsOn(string) -> integer
TurnDomainsOn(tuple of strings) -> integer
TurnDomainsOff() -> integer
TurnDomainsOff(string) -> integer
TurnDomainsOff(tuple of strings) -> integer
```

Returns:
The Turn functions return an integer with a value of 1 for success or 0 for failure.

Description:
The Turn functions are provided to simplify the removal of material or domain subsets. Instead of creating a SILRestriction object, you can use the Turn functions to turn materials or domains on or off. The TurnMaterialsOn function turns materials on and the TurnMaterialsOff function turns them off. The TurnDomainsOn function turns domains on and the TurnDomainsOff function turns them off. All of the Turn functions have three possible argument lists. When you do not provide any arguments, the function applies to all subsets in the SIL so if you called the TurnMaterialsOff function with no arguments, all materials would be turned off. Calling TurnMaterialsOn with no arguments would turn all materials on. All functions can also take a string argument, which is the name of the set to modify. For example, you could turn off domain 0 by calling the TurnDomainsOff with a single argument of "domain0" (or the appropriate set name). All of the Turn functions can also be used to modify more than one set if you provide a tuple of set names. After you use the Turn functions to change the SIL restriction, you might want to call the ListMaterials or ListDomains functions to make sure that the SIL restriction was actually modified.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
TurnMaterialsOff("4") # Turn off material 4
TurnMaterialsOff(("1", "2")) # Turn off materials 1 and 2
TurnMaterialsOn() # Turn on all materials
```

TurnMaterialsOn: turns materials or domains on or off.

Synopsis:
```
TurnMaterialsOn() -> integer
TurnMaterialsOn(string) -> integer
TurnMaterialsOn(tuple of strings) -> integer
TurnMaterialsOff() -> integer
TurnMaterialsOff(string) -> integer
TurnMaterialsOff(tuple of strings) -> integer
TurnDomainsOn() -> integer
TurnDomainsOn(string) -> integer
TurnDomainsOn(tuple of strings) -> integer
TurnDomainsOff() -> integer
TurnDomainsOff(string) -> integer
TurnDomainsOff(tuple of strings) -> integer
```

Returns:
The Turn functions return an integer with a value of 1 for success or 0 for failure.

Description:
The Turn functions are provided to simplify the removal of material or domain subsets. Instead of creating a SILRestriction object, you can use the Turn functions to turn materials or domains on or off. The TurnMaterialsOn function turns materials on and the TurnMaterialsOff function turns them off. The TurnDomainsOn function turns domains on and the TurnDomainsOff function turns them off. All of the Turn functions have three possible argument lists. When you do not provide any arguments, the function applies to all subsets in the SIL so if you called the TurnMaterialsOff function with no arguments, all materials would be turned off. Calling TurnMaterialsOn with no arguments would turn all materials on. All functions can also take a string argument, which is the name of the set to modify. For example, you could turn off domain 0 by calling the TurnDomainsOff with a single argument of "domain0" (or the appropriate set name). All of the Turn functions can also be used to modify more than one set if you provide a tuple of set names. After you use the Turn functions to change the SIL restriction, you might want to call the ListMaterials or ListDomains functions to make sure that the SIL restriction was actually modified.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/globe.silo")
AddPlot("Pseudocolor", "u")
DrawPlots()
TurnMaterialsOff("4") # Turn off material 4
TurnMaterialsOff(("1", "2")) # Turn off materials 1 and 2
TurnMaterialsOn() # Turn on all materials
```

UndoView: restores the previous view

Synopsis:
`UndoView()`

Returns:
The UndoView function does not return a value.

Description:
When the view changes in the visualization window, it puts the old view on a stack of views. The UndoView function restores the view on top of the stack and removes it. This allows the user to undo up to ten view changes.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/curv2d.silo")
AddPlot("Subset", "mat1")
DrawPlots()
v = GetView2D()
v.windowCoords = (-2.3,2.4,0.2,4.9)
SetView2D(v)
UndoView()
```

UpdateNamedSelection: recalculate a named selection using new selection properties.

Synopsis:
```
UpdateNamedSelection(name) -> integer
UpdateNamedSelection(name, properties) -> integer
```

Arguments:

`name`	The name of the selection to update.
`properties`	An optional SelectionProperties object that contains the selectionproperties to use when reevaluating the selection.

Returns:
The UpdateNamedSelection function returns 1 on success and 0 on failure.

Description:
This function causes VisIt to reevaluate a named selection using new selection properties. If no selection properties are provided then the selection will be reevaluated using data for the plot that was associated with the selection when it was created. This is useful if you want to change a plot in several ways before causing its associated named selection to update using the changes.

Example:
```
s = GetSelection('selection1')
s.selectionType = s.CumulativeQuerySelection
s.histogramType = s.HistogramMatches
s.combineRule = s.CombineOr
s.variables = ('temperature',)
s.variableMins = (2.9,)
s.variableMaxs = (3.1,)
UpdateNamedSelection('selection1', s)
```

Version: returns VisIt's version string.

Synopsis:
```
Version() -> string
```

Returns:
The Version function return a string that represents VisIt's version.

Description:
The Version function returns a string that represents VisIt's version. The version string can be used in Python scripts to make sure that the VisIt module is a certain version before processing the rest of the Python script.

Example:
```
% visit -cli
print "We are running VisIt version %s" % Version()
```

WriteConfigFile: tells the viewer to write its configuration file.

Synopsis:
`WriteConfigFile()`

Returns:
The WriteconfigFile function does not return a value.

Description:
The viewer maintains internal settings which determine the default values for objects like plots and operators. The viewer can save out the default values so they can be used in future VisIt sessions. The WriteConfig function tells the viewer to write out the settings to the VisIt configuration file.

Example:
```
% visit -cli
p = PseudocolorAttributes()
p.minFlag, p.min = 1, 5.0
p.maxFlag, p.max = 1, 20.0
SetDefaultPlotOptions(p)
# Save the new default Pseudocolor settings to the config file.
WriteConfig()
```

ZonePick: performs a zonal pick on a plot.

Synopsis:
```
ZonePick(point) -> integer
ZonePick(point, variables) -> integer
ZonePick(sx, sy) -> integer
ZonePick(sx, sy, variables) -> integer
```

Arguments:

`point`	A tuple of values that describe the coordinate of where we want to performthe zonal pick.
`variables`	An optional tuple of strings containing the names of the variables forwhich we want information. The tuple can contain the name "default" ifyou want information for the plotted variable.
`sx`	A screen X location (in pixels) offset from the left side of thevisualization window.
`sy`	A screen Y location (in pixels) offset from the bottom of thevisualization window.

Returns:
The ZonePick function prints pick information for the cell (a.k.a zone) that contains the specified point. The point can be specified as a 2D or 3D point in world space or it can be specified as a pixel location in screen space. If the point is specified as a pixel location then VisIt finds the zone that contains the intersection of a cell and a ray that is projected into the mesh. Once the zonal pick has been calculated, you can use the GetPickOutput function to retrieve the printed pick output as a string which can be used for other purposes.

Example:
```
% visit -cli
OpenDatabase("/usr/gapps/visit/data/noise.silo")
AddPlot("Pseudocolor", "hgslice")
DrawPlots()
# Perform zone pick in screen space
ZonePick(300,300)
# Perform zone pick in world space.
ZonePick((-5.0, 5.0))
```

Chapter 5

Attribute References

This chapter shows all the attributes that can be set to control the behavior of VisIt. The attributes themselves are not documented, but their names are usually quite explanatory. When an attribute can have values from a given list of options, the default option is printed and the other available options are printed in a column beneath in italic.

The listing is ordered in alphabetical ordering of the name of the attribute set. For each set the function that will provide you with these attributes is printed in italic.

Animation: *AnimationAttributes()*

Attribute	Default/Allowed values
animationMode	StopMode
	ReversePlayMode
	StopMode
	PlayMode
pipelineCachingMode	0
frameIncrement	1
timeout	1
playbackMode	Looping
	Looping
	PlayOnce
	Swing

Annotation: *AnnotationAttributes()*

Attribute	Default/Allowed values
axes2D.visible	1
axes2D.autoSetTicks	1
axes2D.autoSetScaling	1
axes2D.lineWidth	0
axes2D.tickLocation	axes2D.Outside
	Inside
	Outside
	Both
axes2D.tickAxes	axes2D.BottomLeft
	Off
	Bottom
	Left
	BottomLeft
	All
axes2D.xAxis.title.visible	1
axes2D.xAxis.title.font.font	axes2D.xAxis.title.font.Courier
	Arial
	Courier
	Times
axes2D.xAxis.title.font.scale	1
axes2D.xAxis.title.font.useForegroundColor	1
axes2D.xAxis.title.font.color	(0, 0, 0, 255)
axes2D.xAxis.title.font.bold	1
axes2D.xAxis.title.font.italic	1
axes2D.xAxis.title.userTitle	0
axes2D.xAxis.title.userUnits	0
axes2D.xAxis.title.title	"X−Axis"
axes2D.xAxis.title.units	""
axes2D.xAxis.label.visible	1
axes2D.xAxis.label.font.font	axes2D.xAxis.label.font.Courier
	Arial
	Courier
	Times
axes2D.xAxis.label.font.scale	1
axes2D.xAxis.label.font.useForegroundColor	1
axes2D.xAxis.label.font.color	(0, 0, 0, 255)
axes2D.xAxis.label.font.bold	1
axes2D.xAxis.label.font.italic	1
axes2D.xAxis.label.scaling	0
axes2D.xAxis.tickMarks.visible	1
axes2D.xAxis.tickMarks.majorMinimum	0
axes2D.xAxis.tickMarks.majorMaximum	1
axes2D.xAxis.tickMarks.minorSpacing	0.02
axes2D.xAxis.tickMarks.majorSpacing	0.2
axes2D.xAxis.grid	0
axes2D.yAxis.title.visible	1
axes2D.yAxis.title.font.font	axes2D.yAxis.title.font.Courier
	Arial
	Courier

Continued on next page...

... Annotation attributes continued

Attribute	Default/Allowed values
	Times
axes2D.yAxis.title.font.scale	1
axes2D.yAxis.title.font.useForegroundColor	1
axes2D.yAxis.title.font.color	(0, 0, 0, 255)
axes2D.yAxis.title.font.bold	1
axes2D.yAxis.title.font.italic	1
axes2D.yAxis.title.userTitle	0
axes2D.yAxis.title.userUnits	0
axes2D.yAxis.title.title	"Y−Axis"
axes2D.yAxis.title.units	""
axes2D.yAxis.label.visible	1
axes2D.yAxis.label.font.font	axes2D.yAxis.label.font.Courier
	Arial
	Courier
	Times
axes2D.yAxis.label.font.scale	1
axes2D.yAxis.label.font.useForegroundColor	1
axes2D.yAxis.label.font.color	(0, 0, 0, 255)
axes2D.yAxis.label.font.bold	1
axes2D.yAxis.label.font.italic	1
axes2D.yAxis.label.scaling	0
axes2D.yAxis.tickMarks.visible	1
axes2D.yAxis.tickMarks.majorMinimum	0
axes2D.yAxis.tickMarks.majorMaximum	1
axes2D.yAxis.tickMarks.minorSpacing	0.02
axes2D.yAxis.tickMarks.majorSpacing	0.2
axes2D.yAxis.grid	0
axes3D.visible	1
axes3D.autoSetTicks	1
axes3D.autoSetScaling	1
axes3D.lineWidth	0
axes3D.tickLocation	axes3D.Inside
	Inside
	Outside
	Both
axes3D.axesType	axes3D.ClosestTriad
	ClosestTriad
	FurthestTriad
	OutsideEdges
	StaticTriad
	StaticEdges
axes3D.triadFlag	1
axes3D.bboxFlag	1
axes3D.xAxis.title.visible	1
axes3D.xAxis.title.font.font	axes3D.xAxis.title.font.Arial
	Arial
	Courier
	Times
axes3D.xAxis.title.font.scale	1

Continued on next page...

... Annotation attributes continued

Attribute	Default/Allowed values
axes3D.xAxis.title.font.useForegroundColor	1
axes3D.xAxis.title.font.color	(0, 0, 0, 255)
axes3D.xAxis.title.font.bold	0
axes3D.xAxis.title.font.italic	0
axes3D.xAxis.title.userTitle	0
axes3D.xAxis.title.userUnits	0
axes3D.xAxis.title.title	"X−Axis"
axes3D.xAxis.title.units	""
axes3D.xAxis.label.visible	1
axes3D.xAxis.label.font.font	axes3D.xAxis.label.font.Arial *Arial* *Courier* *Times*
axes3D.xAxis.label.font.scale	1
axes3D.xAxis.label.font.useForegroundColor	1
axes3D.xAxis.label.font.color	(0, 0, 0, 255)
axes3D.xAxis.label.font.bold	0
axes3D.xAxis.label.font.italic	0
axes3D.xAxis.label.scaling	0
axes3D.xAxis.tickMarks.visible	1
axes3D.xAxis.tickMarks.majorMinimum	0
axes3D.xAxis.tickMarks.majorMaximum	1
axes3D.xAxis.tickMarks.minorSpacing	0.02
axes3D.xAxis.tickMarks.majorSpacing	0.2
axes3D.xAxis.grid	0
axes3D.yAxis.title.visible	1
axes3D.yAxis.title.font.font	axes3D.yAxis.title.font.Arial *Arial* *Courier* *Times*
axes3D.yAxis.title.font.scale	1
axes3D.yAxis.title.font.useForegroundColor	1
axes3D.yAxis.title.font.color	(0, 0, 0, 255)
axes3D.yAxis.title.font.bold	0
axes3D.yAxis.title.font.italic	0
axes3D.yAxis.title.userTitle	0
axes3D.yAxis.title.userUnits	0
axes3D.yAxis.title.title	"Y−Axis"
axes3D.yAxis.title.units	""
axes3D.yAxis.label.visible	1
axes3D.yAxis.label.font.font	axes3D.yAxis.label.font.Arial *Arial* *Courier* *Times*
axes3D.yAxis.label.font.scale	1
axes3D.yAxis.label.font.useForegroundColor	1
axes3D.yAxis.label.font.color	(0, 0, 0, 255)
axes3D.yAxis.label.font.bold	0
axes3D.yAxis.label.font.italic	0

Continued on next page...

... Annotation attributes continued

Attribute	Default/Allowed values
axes3D.yAxis.label.scaling	0
axes3D.yAxis.tickMarks.visible	1
axes3D.yAxis.tickMarks.majorMinimum	0
axes3D.yAxis.tickMarks.majorMaximum	1
axes3D.yAxis.tickMarks.minorSpacing	0.02
axes3D.yAxis.tickMarks.majorSpacing	0.2
axes3D.yAxis.grid	0
axes3D.zAxis.title.visible	1
axes3D.zAxis.title.font.font	axes3D.zAxis.title.font.Arial *Arial* *Courier* *Times*
axes3D.zAxis.title.font.scale	1
axes3D.zAxis.title.font.useForegroundColor	1
axes3D.zAxis.title.font.color	(0, 0, 0, 255)
axes3D.zAxis.title.font.bold	0
axes3D.zAxis.title.font.italic	0
axes3D.zAxis.title.userTitle	0
axes3D.zAxis.title.userUnits	0
axes3D.zAxis.title.title	"Z−Axis"
axes3D.zAxis.title.units	""
axes3D.zAxis.label.visible	1
axes3D.zAxis.label.font.font	axes3D.zAxis.label.font.Arial *Arial* *Courier* *Times*
axes3D.zAxis.label.font.scale	1
axes3D.zAxis.label.font.useForegroundColor	1
axes3D.zAxis.label.font.color	(0, 0, 0, 255)
axes3D.zAxis.label.font.bold	0
axes3D.zAxis.label.font.italic	0
axes3D.zAxis.label.scaling	0
axes3D.zAxis.tickMarks.visible	1
axes3D.zAxis.tickMarks.majorMinimum	0
axes3D.zAxis.tickMarks.majorMaximum	1
axes3D.zAxis.tickMarks.minorSpacing	0.02
axes3D.zAxis.tickMarks.majorSpacing	0.2
axes3D.zAxis.grid	0
axes3D.setBBoxLocation	0
axes3D.bboxLocation	(0, 1, 0, 1, 0, 1)
userInfoFlag	1
userInfoFont.font	userInfoFont.Arial *Arial* *Courier* *Times*
userInfoFont.scale	1
userInfoFont.useForegroundColor	1
userInfoFont.color	(0, 0, 0, 255)
userInfoFont.bold	0

Continued on next page...

... Annotation attributes continued

Attribute	Default/Allowed values
userInfoFont.italic	0
databaseInfoFlag	1
timeInfoFlag	1
databaseInfoFont.font	databaseInfoFont.Arial
	Arial
	Courier
	Times
databaseInfoFont.scale	1
databaseInfoFont.useForegroundColor	1
databaseInfoFont.color	(0, 0, 0, 255)
databaseInfoFont.bold	0
databaseInfoFont.italic	0
databaseInfoExpansionMode	File
	File
	Directory
	Full
	Smart
	SmartDirectory
databaseInfoTimeScale	1
databaseInfoTimeOffset	0
legendInfoFlag	1
backgroundColor	(255, 255, 255, 255)
foregroundColor	(0, 0, 0, 255)
gradientBackgroundStyle	Radial
	TopToBottom
	BottomToTop
	LeftToRight
	RightToLeft
	Radial
gradientColor1	(0, 0, 255, 255)
gradientColor2	(0, 0, 0, 255)
backgroundMode	Solid
	Solid
	Gradient
	Image
	ImageSphere
backgroundImage	""
imageRepeatX	1
imageRepeatY	1
axesArray.visible	1
axesArray.ticksVisible	1
axesArray.autoSetTicks	1
axesArray.autoSetScaling	1
axesArray.lineWidth	0
axesArray.axes.title.visible	1
axesArray.axes.title.font.font	axesArray.axes.title.font.Arial
	Arial
	Courier
	Times

Continued on next page...

... Annotation attributes continued

Attribute	**Default/Allowed values**
axesArray.axes.title.font.scale	1
axesArray.axes.title.font.useForegroundColor	1
axesArray.axes.title.font.color	(0, 0, 0, 255)
axesArray.axes.title.font.bold	0
axesArray.axes.title.font.italic	0
axesArray.axes.title.userTitle	0
axesArray.axes.title.userUnits	0
axesArray.axes.title.title	""
axesArray.axes.title.units	""
axesArray.axes.label.visible	1
axesArray.axes.label.font.font	axesArray.axes.label.font.Arial *Arial* *Courier* *Times*
axesArray.axes.label.font.scale	1
axesArray.axes.label.font.useForegroundColor	1
axesArray.axes.label.font.color	(0, 0, 0, 255)
axesArray.axes.label.font.bold	0
axesArray.axes.label.font.italic	0
axesArray.axes.label.scaling	0
axesArray.axes.tickMarks.visible	1
axesArray.axes.tickMarks.majorMinimum	0
axesArray.axes.tickMarks.majorMaximum	1
axesArray.axes.tickMarks.minorSpacing	0.02
axesArray.axes.tickMarks.majorSpacing	0.2
axesArray.axes.grid	0

Axis: *AxisAttributes()*

Attribute	Default/Allowed values
title.visible	1
title.font.font	title.font.Arial *Arial* *Courier* *Times*
title.font.scale	1
title.font.useForegroundColor	1
title.font.color	(0, 0, 0, 255)
title.font.bold	0
title.font.italic	0
title.userTitle	0
title.userUnits	0
title.title	""
title.units	""
label.visible	1
label.font.font	label.font.Arial *Arial* *Courier* *Times*
label.font.scale	1
label.font.useForegroundColor	1
label.font.color	(0, 0, 0, 255)
label.font.bold	0
label.font.italic	0
label.scaling	0
tickMarks.visible	1
tickMarks.majorMinimum	0
tickMarks.majorMaximum	1
tickMarks.minorSpacing	0.02
tickMarks.majorSpacing	0.2
grid	0

Boundary: *BoundaryAttributes()*

Attribute	Default/Allowed values
colorType	ColorByMultipleColors
	ColorBySingleColor
	ColorByMultipleColors
	ColorByColorTable
colorTableName	"Default"
invertColorTable	0
filledFlag	1
legendFlag	1
lineStyle	SOLID
	SOLID
	DASH
	DOT
	DOTDASH
lineWidth	0
singleColor	(0, 0, 0, 255)
boundaryNames	()
boundaryType	Unknown
	Domain
	Group
	Material
	Unknown
opacity	1
wireframe	0
smoothingLevel	0
pointSize	0.05
pointType	Point
	Box
	Axis
	Icosahedron
	Point
	Sphere
pointSizeVarEnabled	0
pointSizeVar	"default"
pointSizePixels	2

BoundaryOp: *BoundaryOpAttributes()*

Attribute	Default/Allowed values
smoothingLevel	0

Box: *BoxAttributes()*

Attribute	Default/Allowed values
amount	Some
	Some
	All
minx	0
maxx	1
miny	0
maxy	1
minz	0
maxz	1

Clip: *ClipAttributes()*

Attribute	Default/Allowed values
quality	Fast
	Fast
	Accurate
funcType	Plane
	Plane
	Sphere
plane1Status	1
plane2Status	0
plane3Status	0
plane1Origin	(0, 0, 0)
plane2Origin	(0, 0, 0)
plane3Origin	(0, 0, 0)
plane1Normal	(1, 0, 0)
plane2Normal	(0, 1, 0)
plane3Normal	(0, 0, 1)
planeInverse	0
planeToolControlledClipPlane	Plane1
	None
	Plane1
	Plane2
	Plane3
center	(0, 0, 0)
radius	1
sphereInverse	0

Cone: *ConeAttributes()*

Attribute	Default/Allowed values
angle	45
origin	(0, 0, 0)
normal	(0, 0, 1)
representation	Flattened
	ThreeD
	Flattened
	R_Theta
upAxis	(0, 1, 0)
cutByLength	0
length	1

ConnectedComponents: *ConnectedComponentsAttributes()*

Attribute	Default/Allowed values
EnableGhostNeighborsOptimization	1

ConstructDataBinning: *ConstructDataBinningAttributes()*

Attribute	Default/Allowed values
name	""
varnames	()
binType	()
binBoundaries	()
reductionOperator	Average
	Average
	Minimum
	Maximum
	StandardDeviation
	Variance
	Sum
	Count
	RMS
	PDF
varForReductionOperator	""
undefinedValue	0
binningScheme	Uniform
	Uniform
	Unknown
numBins	()
overTime	0
timeStart	0
timeEnd	1
timeStride	1
outOfBoundsBehavior	Clamp
	Clamp
	Discard

Contour: *ContourAttributes()*

Attribute	Default/Allowed values
defaultPalette.GetControlPoints(0).colors	(255, 0, 0, 255)
defaultPalette.GetControlPoints(0).position	0
defaultPalette.GetControlPoints(1).colors	(0, 255, 0, 255)
defaultPalette.GetControlPoints(1).position	0.034
defaultPalette.GetControlPoints(2).colors	(0, 0, 255, 255)
defaultPalette.GetControlPoints(2).position	0.069
defaultPalette.GetControlPoints(3).colors	(0, 255, 255, 255)
defaultPalette.GetControlPoints(3).position	0.103
defaultPalette.GetControlPoints(4).colors	(255, 0, 255, 255)
defaultPalette.GetControlPoints(4).position	0.138
defaultPalette.GetControlPoints(5).colors	(255, 255, 0, 255)
defaultPalette.GetControlPoints(5).position	0.172
defaultPalette.GetControlPoints(6).colors	(255, 135, 0, 255)
defaultPalette.GetControlPoints(6).position	0.207
defaultPalette.GetControlPoints(7).colors	(255, 0, 135, 255)
defaultPalette.GetControlPoints(7).position	0.241
defaultPalette.GetControlPoints(8).colors	(168, 168, 168, 255)
defaultPalette.GetControlPoints(8).position	0.276
defaultPalette.GetControlPoints(9).colors	(255, 68, 68, 255)
defaultPalette.GetControlPoints(9).position	0.31
defaultPalette.GetControlPoints(10).colors	(99, 255, 99, 255)
defaultPalette.GetControlPoints(10).position	0.345
defaultPalette.GetControlPoints(11).colors	(99, 99, 255, 255)
defaultPalette.GetControlPoints(11).position	0.379
defaultPalette.GetControlPoints(12).colors	(40, 165, 165, 255)
defaultPalette.GetControlPoints(12).position	0.414
defaultPalette.GetControlPoints(13).colors	(255, 99, 255, 255)
defaultPalette.GetControlPoints(13).position	0.448
defaultPalette.GetControlPoints(14).colors	(255, 255, 99, 255)
defaultPalette.GetControlPoints(14).position	0.483
defaultPalette.GetControlPoints(15).colors	(255, 170, 99, 255)
defaultPalette.GetControlPoints(15).position	0.517
defaultPalette.GetControlPoints(16).colors	(170, 79, 255, 255)
defaultPalette.GetControlPoints(16).position	0.552
defaultPalette.GetControlPoints(17).colors	(150, 0, 0, 255)
defaultPalette.GetControlPoints(17).position	0.586
defaultPalette.GetControlPoints(18).colors	(0, 150, 0, 255)
defaultPalette.GetControlPoints(18).position	0.621
defaultPalette.GetControlPoints(19).colors	(0, 0, 150, 255)
defaultPalette.GetControlPoints(19).position	0.655
defaultPalette.GetControlPoints(20).colors	(0, 109, 109, 255)
defaultPalette.GetControlPoints(20).position	0.69
defaultPalette.GetControlPoints(21).colors	(150, 0, 150, 255)
defaultPalette.GetControlPoints(21).position	0.724
defaultPalette.GetControlPoints(22).colors	(150, 150, 0, 255)
defaultPalette.GetControlPoints(22).position	0.759
defaultPalette.GetControlPoints(23).colors	(150, 84, 0, 255)
defaultPalette.GetControlPoints(23).position	0.793
defaultPalette.GetControlPoints(24).colors	(160, 0, 79, 255)

Continued on next page...

... Contour attributes continued

Attribute	Default/Allowed values
defaultPalette.GetControlPoints(24).position	0.828
defaultPalette.GetControlPoints(25).colors	(255, 104, 28, 255)
defaultPalette.GetControlPoints(25).position	0.862
defaultPalette.GetControlPoints(26).colors	(0, 170, 81, 255)
defaultPalette.GetControlPoints(26).position	0.897
defaultPalette.GetControlPoints(27).colors	(68, 255, 124, 255)
defaultPalette.GetControlPoints(27).position	0.931
defaultPalette.GetControlPoints(28).colors	(0, 130, 255, 255)
defaultPalette.GetControlPoints(28).position	0.966
defaultPalette.GetControlPoints(29).colors	(130, 0, 255, 255)
defaultPalette.GetControlPoints(29).position	1
defaultPalette.smoothing	defaultPalette.None *None* *Linear* *CubicSpline*
defaultPalette.equalSpacingFlag	1
defaultPalette.discreteFlag	1
defaultPalette.externalFlag	0
changedColors	()
colorType	ColorByMultipleColors *ColorBySingleColor* *ColorByMultipleColors* *ColorByColorTable*
colorTableName	"Default"
invertColorTable	0
legendFlag	1
lineStyle	SOLID *SOLID* *DASH* *DOT* *DOTDASH*
lineWidth	0
singleColor	(255, 0, 0, 255) *SetMultiColor(0, (255, 0, 0, 255))* *SetMultiColor(1, (0, 255, 0, 255))* *SetMultiColor(2, (0, 0, 255, 255))* *SetMultiColor(3, (0, 255, 255, 255))* *SetMultiColor(4, (255, 0, 255, 255))* *SetMultiColor(5, (255, 255, 0, 255))* *SetMultiColor(6, (255, 135, 0, 255))* *SetMultiColor(7, (255, 0, 135, 255))* *SetMultiColor(8, (168, 168, 168, 255))* *SetMultiColor(9, (255, 68, 68, 255))*
contourNLevels	10
contourValue	()
contourPercent	()
contourMethod	Level *Level* *Value*

Continued on next page...

... Contour attributes continued

Attribute	**Default/Allowed values**
	Percent
minFlag	0
maxFlag	0
min	0
max	1
scaling	Linear
	Linear
	Log
wireframe	0

CoordSwap: *CoordSwapAttributes()*

Attribute	Default/Allowed values
newCoord1	Coord1
	Coord1
	Coord2
	Coord3
newCoord2	Coord2
	Coord1
	Coord2
	Coord3
newCoord3	Coord3
	Coord1
	Coord2
	Coord3

CreateBonds: *CreateBondsAttributes()*

Attribute	Default/Allowed values
elementVariable	"element"
atomicNumber1	(1, −1)
atomicNumber2	(−1, −1)
minDist	(0.4, 0.4)
maxDist	(1.2, 1.9)
maxBondsClamp	10
addPeriodicBonds	0
useUnitCellVectors	1
periodicInX	1
periodicInY	1
periodicInZ	1
xVector	(1, 0, 0)
yVector	(0, 1, 0)
zVector	(0, 0, 1)

Curve: *CurveAttributes()*

Attribute	Default/Allowed values
showLines	1
lineStyle	SOLID
	SOLID
	DASH
	DOT
	DOTDASH
lineWidth	0
showPoints	0
symbol	Point
	Point
	TriangleUp
	TriangleDown
	Square
	Circle
	Plus
	X
pointSize	5
pointFillMode	Static
	Static
	Dynamic
pointStride	1
symbolDensity	50
curveColorSource	Cycle
	Cycle
	Custom
curveColor	(0, 0, 0, 255)
showLegend	1
showLabels	1
designator	""
doBallTimeCue	0
ballTimeCueColor	(0, 0, 0, 255)
timeCueBallSize	0.01
doLineTimeCue	0
lineTimeCueColor	(0, 0, 0, 255)
lineTimeCueWidth	0
doCropTimeCue	0
timeForTimeCue	0

Cylinder: *CylinderAttributes()*

Attribute	Default/Allowed values
point1	(0, 0, 0)
point2	(1, 0, 0)
radius	1

DataBinning: *DataBinningAttributes()*

Attribute	Default/Allowed values
numDimensions	One
	One
	Two
	Three
dim1BinBasedOn	Variable
	X
	Y
	Z
	Variable
dim1Var	"default"
dim1SpecifyRange	0
dim1MinRange	0
dim1MaxRange	1
dim1NumBins	50
dim2BinBasedOn	Variable
	X
	Y
	Z
	Variable
dim2Var	"default"
dim2SpecifyRange	0
dim2MinRange	0
dim2MaxRange	1
dim2NumBins	50
dim3BinBasedOn	Variable
	X
	Y
	Z
	Variable
dim3Var	"default"
dim3SpecifyRange	0
dim3MinRange	0
dim3MaxRange	1
dim3NumBins	50
outOfBoundsBehavior	Clamp
	Clamp
	Discard
reductionOperator	Average
	Average
	Minimum
	Maximum
	StandardDeviation
	Variance
	Sum
	Count
	RMS
	PDF
varForReduction	"default"
emptyVal	0

DeferExpression: *DeferExpressionAttributes()*

Attribute	Default/Allowed values
exprs	()

Displace: *DisplaceAttributes()*

Attribute	Default/Allowed values
factor	1
variable	"default"

DualMesh: *DualMeshAttributes()*

Attribute	Default/Allowed values
mode	Auto
	Auto
	NodesToZones
	ZonesToNodes

Edge: *EdgeAttributes()*

Attribute	Default/Allowed values
dummy	1

Elevate: *ElevateAttributes()*

Attribute	Default/Allowed values
useXYLimits	0
limitsMode	OriginalData
	OriginalData
	CurrentPlot
scaling	Linear
	Linear
	Log
	Skew
skewFactor	1
minFlag	0
min	0
maxFlag	0
max	1
zeroFlag	0
variable	"default"

ExportDB: *ExportDBAttributes()*

Attribute	Default/Allowed values
db_type	""
filename	"visit_ex_db"
dirname	"."
variables	()
opts.types	()

ExternalSurface: *ExternalSurfaceAttributes()*

Attribute	Default/Allowed values
removeGhosts	0
edgesIn2D	1

Extrude: *ExtrudeAttributes()*

Attribute	Default/Allowed values
axis	(0, 0, 1)
length	1
steps	30
preserveOriginalCellNumbers	1

FFT: *FFTAttributes()*

Attribute	Default/Allowed values
dummy	0

FTLE: *FTLEAttributes()*

Attribute	Default/Allowed values
integrationTime	1
regionType	RegularGrid
	NativeResolutionOfMesh
	RegularGrid
Resolution	(10, 10, 10)
UseDataSetStart	1
StartPosition	(0, 0, 0)
UseDataSetEnd	1
EndPosition	(1, 1, 1)
direction	Forward
	Forward
	Backward
flowType	Unsteady
	Unsteady
	Steady

FilledBoundary: *FilledBoundaryAttributes()*

Attribute	Default/Allowed values
colorType	ColorByMultipleColors
	ColorBySingleColor
	ColorByMultipleColors
	ColorByColorTable
colorTableName	"Default"
invertColorTable	0
filledFlag	1
legendFlag	1
lineStyle	SOLID
	SOLID
	DASH
	DOT
	DOTDASH
lineWidth	0
singleColor	(0, 0, 0, 255)
boundaryNames	()
boundaryType	Unknown
	Domain
	Group
	Material
	Unknown
opacity	1
wireframe	0
drawInternal	0
smoothingLevel	0
cleanZonesOnly	0
mixedColor	(255, 255, 255, 255)
pointSize	0.05
pointType	Point
	Box
	Axis
	Icosahedron
	Point
	Sphere
pointSizeVarEnabled	0
pointSizeVar	"default"
pointSizePixels	2

Flux: *FluxAttributes()*

Attribute	Default/Allowed values
flowField	"default"
weight	0
weightField	"default"

Font: *FontAttributes()*

Attribute	Default/Allowed values
font	Arial
	Arial
	Courier
	Times
scale	1
useForegroundColor	1
color	(0, 0, 0, 255)
bold	0
italic	0

Global: *GlobalAttributes()*

Attribute	Default/Allowed values
sources	()
windows	(1)
activeWindow	0
iconifiedFlag	0
autoUpdateFlag	0
replacePlots	0
applyOperator	1
executing	0
windowLayout	1
makeDefaultConfirm	1
cloneWindowOnFirstRef	0
automaticallyAddOperator	0
tryHarderCyclesTimes	0
treatAllDBsAsTimeVarying	0
createMeshQualityExpressions	1
createTimeDerivativeExpressions	1
createVectorMagnitudeExpressions	1
newPlotsInheritSILRestriction	1
userDirForSessionFiles	0
saveCrashRecoveryFile	1
applySelection	1
ignoreExtentsFromDbs	0
expandNewPlots	0

Histogram: *HistogramAttributes()*

Attribute	Default/Allowed values
basedOn	ManyZonesForSingleVar
	ManyVarsForSingleZone
	ManyZonesForSingleVar
histogramType	Frequency
	Frequency
	Weighted
	Variable
weightVariable	"default"
limitsMode	OriginalData
	OriginalData
	CurrentPlot
minFlag	0
maxFlag	0
min	0
max	1
numBins	32
domain	0
zone	0
useBinWidths	1
outputType	Block
	Curve
	Block
lineStyle	SOLID
	SOLID
	DASH
	DOT
	DOTDASH
lineWidth	0
color	(200, 80, 40, 255)
dataScale	Linear
	Linear
	Log
	SquareRoot
binScale	Linear
	Linear
	Log
	SquareRoot

IndexSelect: *IndexSelectAttributes()*

Attribute	Default/Allowed values
maxDim	ThreeD
	OneD
	TwoD
	ThreeD
dim	TwoD
	OneD
	TwoD
	ThreeD
xAbsMax	−1
xMin	0
xMax	−1
xIncr	1
xWrap	0
yAbsMax	−1
yMin	0
yMax	−1
yIncr	1
yWrap	0
zAbsMax	−1
zMin	0
zMax	−1
zIncr	1
zWrap	0
useWholeCollection	1
categoryName	"Whole"
subsetName	"Whole"

InverseGhostZone: *InverseGhostZoneAttributes()*

Attribute	Default/Allowed values
requestGhostZones	1
showDuplicated	1
showEnhancedConnectivity	1
showReducedConnectivity	1
showAMRRefined	1
showExterior	1
showNotApplicable	1

Isosurface: *IsosurfaceAttributes()*

Attribute	Default/Allowed values
contourNLevels	10
contourValue	()
contourPercent	()
contourMethod	Level
	Level
	Value
	Percent
minFlag	0
min	0
maxFlag	0
max	1
scaling	Linear
	Linear
	Log
variable	"default"

Isovolume: *IsovolumeAttributes()*

Attribute	Default/Allowed values
lbound	−1e+37
ubound	1e+37
variable	"default"

Keyframe: *KeyframeAttributes()*

Attribute	Default/Allowed values
enabled	0
nFrames	1
nFramesWasUserSet	0

Label: *LabelAttributes()*

Attribute	Default/Allowed values
legendFlag	1
showNodes	0
showCells	1
restrictNumberOfLabels	1
drawLabelsFacing	Front
	Front
	Back
	FrontAndBack
labelDisplayFormat	Natural
	Natural
	LogicalIndex
	Index
numberOfLabels	200
specifyTextColor1	0
textColor1	(255, 0, 0, 0)
textHeight1	0.02
specifyTextColor2	0
textColor2	(0, 0, 255, 0)
textHeight2	0.02
horizontalJustification	HCenter
	HCenter
	Left
	Right
verticalJustification	VCenter
	VCenter
	Top
	Bottom
depthTestMode	LABEL_DT_AUTO
	LABEL_DT_AUTO
	LABEL_DT_ALWAYS
	LABEL_DT_NEVER
formatTemplate	"%g"

Lagrangian: *LagrangianAttributes()*

Attribute	Default/Allowed values
seedPoint	(0, 0, 0)
numSteps	1000
XAxisSample	Step
	Step
	Time
	ArcLength
	Speed
	Vorticity
	Variable
YAxisSample	Step
	Step
	Time
	ArcLength
	Speed
	Vorticity
	Variable
variable	"default"

Light: *LightAttributes()*

Attribute	**Default/Allowed values**
enabledFlag	1
type	Camera
	Ambient
	Object
	Camera
direction	(0, 0, −1)
color	(255, 255, 255, 255)
brightness	1

Lineout: *LineoutAttributes()*

Attribute	Default/Allowed values
point1	(0, 0, 0)
point2	(1, 1, 0)
interactive	0
ignoreGlobal	0
samplingOn	0
numberOfSamplePoints	50
reflineLabels	0

Material: *MaterialAttributes()*

Attribute	Default/Allowed values
smoothing	0
forceMIR	0
cleanZonesOnly	0
needValidConnectivity	0
algorithm	EquiZ
	EquiT
	EquiZ
	Isovolume
	PLIC
	Discrete
iterationEnabled	0
numIterations	5
iterationDamping	0.4
simplifyHeavilyMixedZones	0
maxMaterialsPerZone	3
isoVolumeFraction	0.5
annealingTime	10

Mesh: *MeshAttributes()*

Attribute	Default/Allowed values
legendFlag	1
lineStyle	SOLID
	SOLID
	DASH
	DOT
	DOTDASH
lineWidth	0
meshColor	(0, 0, 0, 255)
outlineOnlyFlag	0
errorTolerance	0.01
meshColorSource	Foreground
	Foreground
	MeshCustom
opaqueColorSource	Background
	Background
	OpaqueCustom
opaqueMode	Auto
	Auto
	On
	Off
pointSize	0.05
opaqueColor	(255, 255, 255, 255)
smoothingLevel	None
	None
	Fast
	High
pointSizeVarEnabled	0
pointSizeVar	"default"
pointType	Point
	Box
	Axis
	Icosahedron
	Point
	Sphere
showInternal	0
pointSizePixels	2
opacity	1

MeshManagement: *MeshManagementAttributes()*

Attribute	**Default/Allowed values**
discretizationTolerance	(0.02, 0.025, 0.05)
discretizationToleranceX	()
discretizationToleranceY	()
discretizationToleranceZ	()
discretizationMode	Uniform
	Uniform
	Adaptive
	MultiPass
discretizeBoundaryOnly	0
passNativeCSG	0

Molecule: *MoleculeAttributes()*

Attribute	Default/Allowed values
drawAtomsAs	SphereAtoms
	NoAtoms
	SphereAtoms
	ImposterAtoms
scaleRadiusBy	Fixed
	Fixed
	Covalent
	Atomic
	Variable
drawBondsAs	CylinderBonds
	NoBonds
	LineBonds
	CylinderBonds
colorBonds	ColorByAtom
	ColorByAtom
	SingleColor
bondSingleColor	(128, 128, 128, 255)
radiusVariable	"Default"
radiusScaleFactor	1
radiusFixed	0.3
atomSphereQuality	Medium
	Low
	Medium
	High
	Super
bondCylinderQuality	Medium
	Low
	Medium
	High
	Super
bondRadius	0.12
bondLineWidth	0
bondLineStyle	SOLID
	SOLID
	DASH
	DOT
	DOTDASH
elementColorTable	"cpk_jmol"
residueTypeColorTable	"amino_shapely"
residueSequenceColorTable	"Default"
continuousColorTable	"Default"
legendFlag	1
minFlag	0
scalarMin	0
maxFlag	0
scalarMax	1

MultiCurve: *MultiCurveAttributes()*

Attribute	Default/Allowed values
defaultPalette.GetControlPoints(0).colors	(255, 0, 0, 255)
defaultPalette.GetControlPoints(0).position	0
defaultPalette.GetControlPoints(1).colors	(0, 255, 0, 255)
defaultPalette.GetControlPoints(1).position	0.034
defaultPalette.GetControlPoints(2).colors	(0, 0, 255, 255)
defaultPalette.GetControlPoints(2).position	0.069
defaultPalette.GetControlPoints(3).colors	(0, 255, 255, 255)
defaultPalette.GetControlPoints(3).position	0.103
defaultPalette.GetControlPoints(4).colors	(255, 0, 255, 255)
defaultPalette.GetControlPoints(4).position	0.138
defaultPalette.GetControlPoints(5).colors	(255, 255, 0, 255)
defaultPalette.GetControlPoints(5).position	0.172
defaultPalette.GetControlPoints(6).colors	(255, 135, 0, 255)
defaultPalette.GetControlPoints(6).position	0.207
defaultPalette.GetControlPoints(7).colors	(255, 0, 135, 255)
defaultPalette.GetControlPoints(7).position	0.241
defaultPalette.GetControlPoints(8).colors	(168, 168, 168, 255)
defaultPalette.GetControlPoints(8).position	0.276
defaultPalette.GetControlPoints(9).colors	(255, 68, 68, 255)
defaultPalette.GetControlPoints(9).position	0.31
defaultPalette.GetControlPoints(10).colors	(99, 255, 99, 255)
defaultPalette.GetControlPoints(10).position	0.345
defaultPalette.GetControlPoints(11).colors	(99, 99, 255, 255)
defaultPalette.GetControlPoints(11).position	0.379
defaultPalette.GetControlPoints(12).colors	(40, 165, 165, 255)
defaultPalette.GetControlPoints(12).position	0.414
defaultPalette.GetControlPoints(13).colors	(255, 99, 255, 255)
defaultPalette.GetControlPoints(13).position	0.448
defaultPalette.GetControlPoints(14).colors	(255, 255, 99, 255)
defaultPalette.GetControlPoints(14).position	0.483
defaultPalette.GetControlPoints(15).colors	(255, 170, 99, 255)
defaultPalette.GetControlPoints(15).position	0.517
defaultPalette.GetControlPoints(16).colors	(170, 79, 255, 255)
defaultPalette.GetControlPoints(16).position	0.552
defaultPalette.GetControlPoints(17).colors	(150, 0, 0, 255)
defaultPalette.GetControlPoints(17).position	0.586
defaultPalette.GetControlPoints(18).colors	(0, 150, 0, 255)
defaultPalette.GetControlPoints(18).position	0.621
defaultPalette.GetControlPoints(19).colors	(0, 0, 150, 255)
defaultPalette.GetControlPoints(19).position	0.655
defaultPalette.GetControlPoints(20).colors	(0, 109, 109, 255)
defaultPalette.GetControlPoints(20).position	0.69
defaultPalette.GetControlPoints(21).colors	(150, 0, 150, 255)
defaultPalette.GetControlPoints(21).position	0.724
defaultPalette.GetControlPoints(22).colors	(150, 150, 0, 255)
defaultPalette.GetControlPoints(22).position	0.759
defaultPalette.GetControlPoints(23).colors	(150, 84, 0, 255)
defaultPalette.GetControlPoints(23).position	0.793
defaultPalette.GetControlPoints(24).colors	(160, 0, 79, 255)

Continued on next page...

... MultiCurve attributes continued

Attribute	Default/Allowed values
defaultPalette.GetControlPoints(24).position	0.828
defaultPalette.GetControlPoints(25).colors	(255, 104, 28, 255)
defaultPalette.GetControlPoints(25).position	0.862
defaultPalette.GetControlPoints(26).colors	(0, 170, 81, 255)
defaultPalette.GetControlPoints(26).position	0.897
defaultPalette.GetControlPoints(27).colors	(68, 255, 124, 255)
defaultPalette.GetControlPoints(27).position	0.931
defaultPalette.GetControlPoints(28).colors	(0, 130, 255, 255)
defaultPalette.GetControlPoints(28).position	0.966
defaultPalette.GetControlPoints(29).colors	(130, 0, 255, 255)
defaultPalette.GetControlPoints(29).position	1
defaultPalette.smoothing	defaultPalette.None
	None
	Linear
	CubicSpline
defaultPalette.equalSpacingFlag	1
defaultPalette.discreteFlag	1
defaultPalette.externalFlag	0
changedColors	()
colorType	ColorByMultipleColors
	ColorBySingleColor
	ColorByMultipleColors
singleColor	(255, 0, 0, 255)
	SetMultiColor(0, (255, 0, 0, 255))
	SetMultiColor(1, (0, 255, 0, 255))
	SetMultiColor(2, (0, 0, 255, 255))
	SetMultiColor(3, (0, 255, 255, 255))
	SetMultiColor(4, (255, 0, 255, 255))
	SetMultiColor(5, (255, 255, 0, 255))
	SetMultiColor(6, (255, 135, 0, 255))
	SetMultiColor(7, (255, 0, 135, 255))
	SetMultiColor(8, (168, 168, 168, 255))
	SetMultiColor(9, (255, 68, 68, 255))
	SetMultiColor(10, (99, 255, 99, 255))
	SetMultiColor(11, (99, 99, 255, 255))
	SetMultiColor(12, (40, 165, 165, 255))
	SetMultiColor(13, (255, 99, 255, 255))
	SetMultiColor(14, (255, 255, 99, 255))
	SetMultiColor(15, (255, 170, 99, 255))
lineStyle	SOLID
	SOLID
	DASH
	DOT
	DOTDASH
lineWidth	0
yAxisTitleFormat	"%g"
useYAxisTickSpacing	0
yAxisTickSpacing	1
displayMarkers	1

Continued on next page...

... MultiCurve attributes continued

Attribute	**Default/Allowed values**
markerVariable	"default"
displayIds	0
idVariable	"default"
legendFlag	1

MultiresControl: *MultiresControlAttributes()*

Attribute	Default/Allowed values
resolution	0
maxResolution	1
info	""

OnionPeel: *OnionPeelAttributes()*

Attribute	Default/Allowed values
adjacencyType	Node
	Node
	Face
useGlobalId	0
categoryName	"Whole"
subsetName	"Whole"
index	(1)
logical	0
requestedLayer	0
seedType	SeedCell
	SeedCell
	SeedNode

ParallelCoordinates: *ParallelCoordinatesAttributes()*

Attribute	Default/Allowed values
scalarAxisNames	()
visualAxisNames	()
extentMinima	()
extentMaxima	()
drawLines	1
linesColor	(128, 0, 0, 255)
drawContext	1
contextGamma	2
contextNumPartitions	128
contextColor	(0, 220, 0, 255)
drawLinesOnlyIfExtentsOn	1
unifyAxisExtents	0
linesNumPartitions	512
focusGamma	4
drawFocusAs	BinsOfConstantColor *IndividualLines* *BinsOfConstantColor* *BinsColoredByPopulation*

PersistentParticles: *PersistentParticlesAttributes()*

Attribute	Default/Allowed values
startIndex	0
stopIndex	1
stride	1
startPathType	Absolute
	Absolute
	Relative
stopPathType	Absolute
	Absolute
	Relative
traceVariableX	"default"
traceVariableY	"default"
traceVariableZ	"default"
connectParticles	0
showPoints	0
indexVariable	"default"

Poincare: *PoincareAttributes()*

Attribute	Default/Allowed values
opacityType	Explicit
	Explicit
	ColorTable
opacity	1
minPunctures	50
maxPunctures	500
puncturePlane	Poloidal
	Poloidal
	Toroidal
	Arbitrary
sourceType	SpecifiedPoint
	SpecifiedPoint
	SpecifiedLine
pointSource	(0, 0, 0)
lineStart	(0, 0, 0)
lineEnd	(1, 0, 0)
pointDensity	1
fieldType	Default
	Default
	M3DC12DField
	M3DC13DField
	NIMRODField
	FlashField
fieldConstant	1
velocitySource	(0, 0, 0)
integrationType	AdamsBashforth
	Euler
	Leapfrog
	DormandPrince
	AdamsBashforth
	Reserved_4
	M3DC12DIntegrator
coordinateSystem	Cartesian
	Cartesian
	Cylindrical
maxStepLength	0.1
limitMaximumTimestep	0
maxTimeStep	0.1
relTol	0.0001
absTolSizeType	FractionOfBBox
	Absolute
	FractionOfBBox
absTolAbsolute	1e−05
absTolBBox	1e−06
analysis	Normal
	None
	Normal
maximumToroidalWinding	0
overrideToroidalWinding	0

Continued on next page...

... Poincare attributes continued

Attribute	**Default/Allowed values**
overridePoloidalWinding	0
windingPairConfidence	0.9
rationalSurfaceFactor	0.1
adjustPlane	−1
overlaps	Remove
	Raw
	Remove
	Merge
	Smooth
meshType	Curves
	Curves
	Surfaces
numberPlanes	1
singlePlane	0
min	0
max	0
minFlag	0
maxFlag	0
colorType	ColorByColorTable
	ColorBySingleColor
	ColorByColorTable
singleColor	(0, 0, 0, 255)
colorTableName	"Default"
dataValue	SafetyFactorQ
	Solid
	SafetyFactorQ
	SafetyFactorP
	SafetyFactorQ_NotP
	SafetyFactorP_NotQ
	ToroidalWindings
	PoloidalWindingsQ
	PoloidalWindingsP
	FieldlineOrder
	PointOrder
	PlaneOrder
	WindingGroupOrder
	WindingPointOrder
	WindingPointOrderModulo
showOPoints	0
OPointMaxIterations	2
showXPoints	0
XPointMaxIterations	2
performOLineAnalysis	0
OLineToroidalWinding	0
OLineAxisFileName	""
showChaotic	0
showIslands	0
verboseFlag	1
show1DPlots	0

Continued on next page...

... Poincare attributes continued

Attribute	**Default/Allowed values**
showLines	1
lineWidth	0
lineStyle	SOLID
	SOLID
	DASH
	DOT
	DOTDASH
showPoints	0
pointSize	1
pointSizePixels	1
pointType	Point
	Box
	Axis
	Icosahedron
	Point
	Sphere
legendFlag	1
lightingFlag	1
streamlineAlgorithmType	LoadOnDemand
	LoadOnDemand
	ParallelStaticDomains
	MasterSlave
maxStreamlineProcessCount	10
maxDomainCacheSize	3
workGroupSize	32
forceNodeCenteredData	0

Printer: *PrinterAttributes()*

Attribute	Default/Allowed values
printerName	""
printProgram	"lpr"
documentName	"untitled"
creator	""
numCopies	1
portrait	1
printColor	1
outputToFile	0
outputToFileName	"untitled"
pageSize	2

Process: *ProcessAttributes()*

Attribute	Default/Allowed values
pids	()
ppids	()
hosts	()
isParallel	0

Project: *ProjectAttributes()*

Attribute	**Default/Allowed values**
projectionType	XYCartesian
	ZYCartesian
	XZCartesian
	XYCartesian
	XRCylindrical
	YRCylindrical
	ZRCylindrical
vectorTransformMethod	AsDirection
	None
	AsPoint
	AsDisplacement
	AsDirection

Pseudocolor: *PseudocolorAttributes()*

Attribute	Default/Allowed values
legendFlag	1
lightingFlag	1
minFlag	0
maxFlag	0
centering	Natural
	Natural
	Nodal
	Zonal
scaling	Linear
	Linear
	Log
	Skew
limitsMode	OriginalData
	OriginalData
	CurrentPlot
min	0
max	1
pointSize	0.05
pointType	Point
	Box
	Axis
	Icosahedron
	Point
	Sphere
skewFactor	1
opacity	1
colorTableName	"hot"
invertColorTable	0
smoothingLevel	0
pointSizeVarEnabled	0
pointSizeVar	"default"
pointSizePixels	2
lineStyle	SOLID
	SOLID
	DASH
	DOT
	DOTDASH
lineWidth	0
opacityType	Explicit
	Explicit
	ColorTable

Reflect: *ReflectAttributes()*

Attribute	Default/Allowed values
octant	PXPYPZ
	PXPYPZ
	NXPYPZ
	PXNYPZ
	NXNYPZ
	PXPYNZ
	NXPYNZ
	PXNYNZ
	NXNYNZ
useXBoundary	1
specifiedX	0
useYBoundary	1
specifiedY	0
useZBoundary	1
specifiedZ	0
reflections	(1, 0, 1, 0, 0, 0, 0, 0)

Rendering: *RenderingAttributes()*

Attribute	Default/Allowed values
antialiasing	0
multiresolutionMode	0
multiresolutionCellSize	0.002
geometryRepresentation	Surfaces
	Surfaces
	Wireframe
	Points
displayListMode	Auto
	Never
	Always
	Auto
stereoRendering	0
stereoType	CrystalEyes
	RedBlue
	Interlaced
	CrystalEyes
	RedGreen
notifyForEachRender	0
scalableActivationMode	Auto
	Never
	Always
	Auto
scalableAutoThreshold	2000000
specularFlag	0
specularCoeff	0.6
specularPower	10
specularColor	(255, 255, 255, 255)
doShadowing	0
shadowStrength	0.5
doDepthCueing	0
depthCueingAutomatic	1
startCuePoint	(−10, 0, 0)
endCuePoint	(10, 0, 0)
compressionActivationMode	Never
	Never
	Always
	Auto
colorTexturingFlag	1
compactDomainsActivationMode	Never
	Never
	Always
	Auto
compactDomainsAutoThreshold	256

Replicate: *ReplicateAttributes()*

Attribute	**Default/Allowed values**
useUnitCellVectors	0
xVector	(1, 0, 0)
yVector	(0, 1, 0)
zVector	(0, 0, 1)
xReplications	1
yReplications	1
zReplications	1
mergeResults	1
replicateUnitCellAtoms	0
shiftPeriodicAtomOrigin	0
newPeriodicOrigin	(0, 0, 0)

Resample: *ResampleAttributes()*

Attribute	**Default/Allowed values**
useExtents	1
startX	0
endX	1
samplesX	10
startY	0
endY	1
samplesY	10
is3D	1
startZ	0
endZ	1
samplesZ	10
tieResolver	random
	random
	largest
	smallest
tieResolverVariable	"default"
defaultValue	0
distributedResample	1
cellCenteredOutput	0

Revolve: *RevolveAttributes()*

Attribute	Default/Allowed values
meshType	Auto
	Auto
	XY
	RZ
	ZR
autoAxis	1
axis	(1, 0, 0)
startAngle	0
stopAngle	360
steps	30

SaveWindow: *SaveWindowAttributes()*

Attribute	Default/Allowed values
outputToCurrentDirectory	1
outputDirectory	"."
fileName	"visit"
family	1
format	PNG
	BMP
	CURVE
	JPEG
	OBJ
	PNG
	POSTSCRIPT
	POVRAY
	PPM
	RGB
	STL
	TIFF
	ULTRA
	VTK
	PLY
width	1024
height	1024
screenCapture	0
saveTiled	0
quality	80
progressive	0
binary	0
stereo	0
compression	PackBits
	None
	PackBits
	Jpeg
	Deflate
forceMerge	0
resConstraint	ScreenProportions
	NoConstraint
	EqualWidthHeight
	ScreenProportions
advancedMultiWindowSave	0

Scatter: *ScatterAttributes()*

Attribute	Default/Allowed values
var1	"default"
var1Role	Coordinate0
	Coordinate0
	Coordinate1
	Coordinate2
	Color
	None
var1MinFlag	0
var1MaxFlag	0
var1Min	0
var1Max	1
var1Scaling	Linear
	Linear
	Log
	Skew
var1SkewFactor	1
var2Role	Coordinate1
	Coordinate0
	Coordinate1
	Coordinate2
	Color
	None
var2	"default"
var2MinFlag	0
var2MaxFlag	0
var2Min	0
var2Max	1
var2Scaling	Linear
	Linear
	Log
	Skew
var2SkewFactor	1
var3Role	None
	Coordinate0
	Coordinate1
	Coordinate2
	Color
	None
var3	"default"
var3MinFlag	0
var3MaxFlag	0
var3Min	0
var3Max	1
var3Scaling	Linear
	Linear
	Log
	Skew
var3SkewFactor	1
var4Role	None

Continued on next page...

... Scatter attributes continued

Attribute	**Default/Allowed values**
	Coordinate0
	Coordinate1
	Coordinate2
	Color
	None
var4	"default"
var4MinFlag	0
var4MaxFlag	0
var4Min	0
var4Max	1
var4Scaling	Linear
	Linear
	Log
	Skew
var4SkewFactor	1
pointSize	0.05
pointSizePixels	1
pointType	Point
	Box
	Axis
	Icosahedron
	Point
	Sphere
scaleCube	1
colorType	ColorByForegroundColor
	ColorByForegroundColor
	ColorBySingleColor
	ColorByColorTable
singleColor	(255, 0, 0, 255)
colorTableName	"Default"
invertColorTable	0
legendFlag	1

Slice: *SliceAttributes()*

Attribute	Default/Allowed values
originType	Intercept
	Point
	Intercept
	Percent
	Zone
	Node
originPoint	(0, 0, 0)
originIntercept	0
originPercent	0
originZone	0
originNode	0
normal	(0, −1, 0)
axisType	YAxis
	XAxis
	YAxis
	ZAxis
	Arbitrary
	ThetaPhi
upAxis	(0, 0, 1)
project2d	1
interactive	1
flip	0
originZoneDomain	0
originNodeDomain	0
meshName	”default”
theta	0
phi	0

SmoothOperator: *SmoothOperatorAttributes()*

Attribute	Default/Allowed values
numIterations	20
relaxationFactor	0.01
convergence	0
maintainFeatures	1
featureAngle	45
edgeAngle	15
smoothBoundaries	0

SphereSlice: *SphereSliceAttributes()*

Attribute	Default/Allowed values
origin	(0, 0, 0)
radius	1

Spreadsheet: *SpreadsheetAttributes()*

Attribute	Default/Allowed values
subsetName	"Whole"
formatString	"%1.6f"
useColorTable	0
colorTableName	"Default"
showTracerPlane	1
tracerColor	(255, 0, 0, 150)
normal	Z
	X
	Y
	Z
sliceIndex	0
spreadsheetFont	"Courier,12,−1,5,50,0,0,0,0,0"
showPatchOutline	1
showCurrentCellOutline	0
currentPickType	0
currentPickLetter	""
pastPickLetters	()

Stagger: *StaggerAttributes()*

Attribute	Default/Allowed values
offsetX	0
offsetY	0
offsetZ	0

Streamline: *StreamlineAttributes()*

Attribute	Default/Allowed values
sourceType	SpecifiedPoint
	SpecifiedPoint
	SpecifiedPointList
	SpecifiedLine
	SpecifiedCircle
	SpecifiedPlane
	SpecifiedSphere
	SpecifiedBox
	Selection
pointSource	(0, 0, 0)
lineStart	(0, 0, 0)
lineEnd	(1, 0, 0)
planeOrigin	(0, 0, 0)
planeNormal	(0, 0, 1)
planeUpAxis	(0, 1, 0)
radius	1
sphereOrigin	(0, 0, 0)
boxExtents	(0, 1, 0, 1, 0, 1)
useWholeBox	1
pointList	(0, 0, 0, 1, 0, 0, 0, 1, 0)
sampleDensity0	2
sampleDensity1	2
sampleDensity2	2
coloringMethod	ColorByTime
	Solid
	ColorBySpeed
	ColorByVorticity
	ColorByLength
	ColorByTime
	ColorBySeedPointID
	ColorByVariable
	ColorByCorrelationDistance
colorTableName	"Default"
singleColor	(0, 0, 0, 255)
legendFlag	1
lightingFlag	1
streamlineDirection	Forward
	Forward
	Backward
	Both
maxSteps	1000
terminateByDistance	0
termDistance	10
terminateByTime	0
termTime	10
maxStepLength	0.1
limitMaximumTimestep	0
maxTimeStep	0.1
relTol	0.0001

Continued on next page...

... Streamline attributes continued

Attribute	Default/Allowed values
absTolSizeType	FractionOfBBox
	Absolute
	FractionOfBBox
absTolAbsolute	1e−06
absTolBBox	1e−06
fieldType	Default
	Default
	M3DC12DField
	M3DC13DField
	NIMRODField
	FlashField
fieldConstant	1
velocitySource	(0, 0, 0)
integrationType	DormandPrince
	Euler
	Leapfrog
	DormandPrince
	AdamsBashforth
	RK4
	M3DC12DIntegrator
streamlineAlgorithmType	VisItSelects
	LoadOnDemand
	ParallelStaticDomains
	MasterSlave
	VisItSelects
maxStreamlineProcessCount	10
maxDomainCacheSize	3
workGroupSize	32
pathlines	0
pathlinesOverrideStartingTimeFlag	0
pathlinesOverrideStartingTime	0
pathlinesCMFE	POS_CMFE
	CONN_CMFE
	POS_CMFE
coordinateSystem	AsIs
	AsIs
	CylindricalToCartesian
	CartesianToCylindrical
phiScalingFlag	0
phiScaling	1
coloringVariable	""
legendMinFlag	0
legendMaxFlag	0
legendMin	0
legendMax	1
displayBegin	0
displayEnd	1
displayBeginFlag	0
displayEndFlag	0

Continued on next page...

... Streamline attributes continued

Attribute	**Default/Allowed values**
referenceTypeForDisplay	Distance
	Distance
	Time
	Step
displayMethod	Lines
	Lines
	Tubes
	Ribbons
tubeSizeType	FractionOfBBox
	Absolute
	FractionOfBBox
tubeRadiusAbsolute	0.125
tubeRadiusBBox	0.005
ribbonWidthSizeType	FractionOfBBox
	Absolute
	FractionOfBBox
ribbonWidthAbsolute	0.125
ribbonWidthBBox	0.01
lineWidth	2
showSeeds	1
seedRadiusSizeType	FractionOfBBox
	Absolute
	FractionOfBBox
seedRadiusAbsolute	1
seedRadiusBBox	0.015
showHeads	0
headDisplayType	Sphere
	Sphere
	Cone
headRadiusSizeType	FractionOfBBox
	Absolute
	FractionOfBBox
headRadiusAbsolute	0.25
headRadiusBBox	0.02
headHeightRatio	2
opacityType	FullyOpaque
	FullyOpaque
	Constant
	Ramp
	VariableRange
opacityVariable	""
opacity	1
opacityVarMin	0
opacityVarMax	1
opacityVarMinFlag	0
opacityVarMaxFlag	0
tubeDisplayDensity	10
geomDisplayQuality	Medium
	Low

Continued on next page...

... Streamline attributes continued

Attribute	**Default/Allowed values**
	Medium
	High
	Super
sampleDistance0	10
sampleDistance1	10
sampleDistance2	10
fillInterior	1
randomSamples	0
randomSeed	0
numberOfRandomSamples	1
forceNodeCenteredData	0
issueTerminationWarnings	1
issueStiffnessWarnings	1
issueCriticalPointsWarnings	1
criticalPointThreshold	0.001
varyTubeRadius	None
	None
	Scalar
varyTubeRadiusFactor	10
varyTubeRadiusVariable	""
correlationDistanceAngTol	5
correlationDistanceMinDistAbsolute	1
correlationDistanceMinDistBBox	0.005
correlationDistanceMinDistType	FractionOfBBox
	Absolute
	FractionOfBBox
selection	""

Subset: *SubsetAttributes()*

Attribute	Default/Allowed values
colorType	ColorByMultipleColors
	ColorBySingleColor
	ColorByMultipleColors
	ColorByColorTable
colorTableName	"Default"
invertColorTable	0
filledFlag	1
legendFlag	1
lineStyle	SOLID
	SOLID
	DASH
	DOT
	DOTDASH
lineWidth	0
singleColor	(0, 0, 0, 255)
subsetNames	()
subsetType	Unknown
	Domain
	Group
	Material
	EnumScalar
	Mesh
	Unknown
opacity	1
wireframe	0
drawInternal	0
smoothingLevel	0
pointSize	0.05
pointType	Point
	Box
	Axis
	Icosahedron
	Point
	Sphere
pointSizeVarEnabled	0
pointSizeVar	"default"
pointSizePixels	2

SurfaceNormal: *SurfaceNormalAttributes()*

Attribute	Default/Allowed values
centering	Point
	Point
	Cell

Tensor: *TensorAttributes()*

Attribute	Default/Allowed values
useStride	0
stride	1
nTensors	400
scale	0.25
scaleByMagnitude	1
autoScale	1
colorByEigenvalues	1
useLegend	1
tensorColor	(0, 0, 0, 255)
colorTableName	"Default"
invertColorTable	0

ThreeSlice: *ThreeSliceAttributes()*

Attribute	Default/Allowed values
x	0
y	0
z	0
interactive	1

Threshold: *ThresholdAttributes()*

Attribute	Default/Allowed values
outputMeshType	0
listedVarNames	("default")
zonePortions	()
lowerBounds	()
upperBounds	()
defaultVarName	"default"
defaultVarIsScalar	0

Transform: *TransformAttributes()*

Attribute	Default/Allowed values
doRotate	0
rotateOrigin	(0, 0, 0)
rotateAxis	(0, 0, 1)
rotateAmount	0
rotateType	Deg
	Deg
	Rad
doScale	0
scaleOrigin	(0, 0, 0)
scaleX	1
scaleY	1
scaleZ	1
doTranslate	0
translateX	0
translateY	0
translateZ	0
transformType	Similarity
	Similarity
	Coordinate
	Linear
inputCoordSys	Cartesian
	Cartesian
	Cylindrical
	Spherical
outputCoordSys	Spherical
	Cartesian
	Cylindrical
	Spherical
m00	1
m01	0
m02	0
m03	0
m10	0
m11	1
m12	0
m13	0
m20	0
m21	0
m22	1
m23	0
m30	0
m31	0
m32	0
m33	1
invertLinearTransform	0
vectorTransformMethod	AsDirection
	None
	AsPoint
	AsDisplacement

Continued on next page...

... Transform attributes continued

Attribute	**Default/Allowed values**
	AsDirection
transformVectors	1

TriangulateRegularPoints: *TriangulateRegularPointsAttributes()*

Attribute	Default/Allowed values
useXGridSpacing	0
xGridSpacing	1
useYGridSpacing	0
yGridSpacing	1

Truecolor: *TruecolorAttributes()*

Attribute	Default/Allowed values
opacity	1
lightingFlag	1

Tube: *TubeAttributes()*

Attribute	**Default/Allowed values**
scaleByVarFlag	0
tubeRadiusType	FractionOfBBox *FractionOfBBox* *Absolute*
radiusFractionBBox	0.01
radiusAbsolute	1
scaleVariable	"default"
fineness	5
capping	0

Vector: *VectorAttributes()*

Attribute	Default/Allowed values
glyphLocation	AdaptsToMeshResolution
	AdaptsToMeshResolution
	UniformInSpace
useStride	0
stride	1
nVectors	400
lineStyle	SOLID
	SOLID
	DASH
	DOT
	DOTDASH
lineWidth	0
scale	0.25
scaleByMagnitude	1
autoScale	1
headSize	0.25
headOn	1
colorByMag	1
useLegend	1
vectorColor	(0, 0, 0, 255)
colorTableName	"Default"
invertColorTable	0
vectorOrigin	Tail
	Head
	Middle
	Tail
minFlag	0
maxFlag	0
limitsMode	OriginalData
	OriginalData
	CurrentPlot
min	0
max	1
lineStem	1
geometryQuality	Fast
	Fast
	High
stemWidth	0.08
origOnly	1
glyphType	Arrow
	Arrow
	Ellipsoid

View: *ViewAttributes()*

Attribute	Default/Allowed values
viewNormal	(0, 0, 1)
focus	(0, 0, 0)
viewUp	(0, 1, 0)
viewAngle	30
setScale	0
parallelScale	1
nearPlane	0.001
farPlane	100
imagePan	(0, 0)
imageZoom	1
perspective	1
windowCoords	(0, 0, 1, 1)
viewportCoords	(0.1, 0.1, 0.9, 0.9)
eyeAngle	2

View2D: *View2DAttributes()*

Attribute	Default/Allowed values
windowCoords	(0, 1, 0, 1)
viewportCoords	(0.2, 0.95, 0.15, 0.95)
fullFrameActivationMode	Auto
	On
	Off
	Auto
fullFrameAutoThreshold	100
xScale	LINEAR
	LINEAR
	LOG
yScale	LINEAR
	LINEAR
	LOG
windowValid	0

View3D: *View3DAttributes()*

Attribute	Default/Allowed values
viewNormal	(0, 0, 1)
focus	(0, 0, 0)
viewUp	(0, 1, 0)
viewAngle	30
parallelScale	0.5
nearPlane	−0.5
farPlane	0.5
imagePan	(0, 0)
imageZoom	1
perspective	1
eyeAngle	2
centerOfRotationSet	0
centerOfRotation	(0, 0, 0)
axis3DScaleFlag	0
axis3DScales	(1, 1, 1)
shear	(0, 0, 1)

ViewAxisArray: *ViewAxisArrayAttributes()*

Attribute	Default/Allowed values
domainCoords	(0, 1)
rangeCoords	(0, 1)
viewportCoords	(0.15, 0.9, 0.1, 0.85)

ViewCurve: *ViewCurveAttributes()*

Attribute	Default/Allowed values
domainCoords	(0, 1)
rangeCoords	(0, 1)
viewportCoords	(0.2, 0.95, 0.15, 0.95)
domainScale	LINEAR
	LINEAR
	LOG
rangeScale	LINEAR
	LINEAR
	LOG

Volume: *VolumeAttributes()*

Attribute	Default/Allowed values
legendFlag	1
lightingFlag	1
colorControlPoints.GetControlPoints(0).colors	(0, 0, 255, 255)
colorControlPoints.GetControlPoints(0).position	0
colorControlPoints.GetControlPoints(1).colors	(0, 255, 255, 255)
colorControlPoints.GetControlPoints(1).position	0.25
colorControlPoints.GetControlPoints(2).colors	(0, 255, 0, 255)
colorControlPoints.GetControlPoints(2).position	0.5
colorControlPoints.GetControlPoints(3).colors	(255, 255, 0, 255)
colorControlPoints.GetControlPoints(3).position	0.75
colorControlPoints.GetControlPoints(4).colors	(255, 0, 0, 255)
colorControlPoints.GetControlPoints(4).position	1
colorControlPoints.smoothing	colorControlPoints.Linear *None* *Linear* *CubicSpline*
colorControlPoints.equalSpacingFlag	0
colorControlPoints.discreteFlag	0
colorControlPoints.externalFlag	0
opacityAttenuation	1
opacityMode	FreeformMode *FreeformMode* *GaussianMode* *ColorTableMode* *controlPoints does not contain any GaussianControlPoint objects.*
resampleFlag	1
resampleTarget	50000
opacityVariable	"default"
compactVariable	"default"

Continued on next page...

... Volume attributes continued

Attribute	Default/Allowed values
freeformOpacity	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255)
useColorVarMin	0
colorVarMin	0
useColorVarMax	0
colorVarMax	0
useOpacityVarMin	0
opacityVarMin	0
useOpacityVarMax	0
opacityVarMax	0
smoothData	0
samplesPerRay	500
rendererType	Splatting *Splatting* *Texture3D* *RayCasting* *RayCastingIntegration* *SLIVR* *Tuvok*
gradientType	SobelOperator *CenteredDifferences* *SobelOperator*
num3DSlices	200
scaling	Linear *Linear* *Log* *Skew*
skewFactor	1

Continued on next page...

... Volume attributes continued

Attribute	**Default/Allowed values**
limitsMode	OriginalData
	OriginalData
	CurrentPlot
sampling	Rasterization
	KernelBased
	Rasterization
rendererSamples	3
	transferFunction2DWidgets does not contain any TransferFunctionWidget objects.
transferFunctionDim	1
lowGradientLightingReduction	Lower
	Off
	Lowest
	Lower
	Low
	Medium
	High
	Higher
	Highest
lowGradientLightingClampFlag	0
lowGradientLightingClampValue	1

Chapter 6

VisIt CLI Events

This chapter shows a table with all events that the VisIt GUI could potentially generate. Different plugins create different events, so the list will depend on the user configuration. The list in this section is generated from a call to the *GetCallbackNames()* function and will therefore list just the events that are applicable to the user that generates this documentation.

The list is alphabetically ordered. The left column, labeled *EventName* displays each event or callback name. The right column, labeled *ArgCount* displays the result of calling *GetCallbackArgumentCount(EventName)* for the corresponding event, which returns the number of arguments a callback function for that event should accept.

EventName	*ArgCount*
ActivateDatabaseRPC	1
AddAnnotationObjectRPC	2
AddInitializedOperatorRPC	1
AddOperatorRPC	2
AddPlotRPC	2
AddWindowRPC	0
AlterDatabaseCorrelationRPC	4
AnimationAttributes	1
AnimationPlayRPC	0
AnimationReversePlayRPC	0
AnimationSetNFramesRPC	1
AnimationStopRPC	0
AnnotationAttributes	1
ApplyNamedSelectionRPC	1
BoundaryAttributes	1
BoundaryOpAttributes	1
BoxAttributes	1
ChangeActivePlotsVarRPC	1
CheckForNewStatesRPC	1
ChooseCenterOfRotationRPC	2
ClearAllWindowsRPC	0
ClearCacheForAllEnginesRPC	0
ClearCacheRPC	2
ClearPickPointsRPC	0
ClearRefLinesRPC	0

Continued on next page...

... CLI Events continued

EventName	*ArgCount*
ClearViewKeyframesRPC	0
ClearWindowRPC	1
ClipAttributes	1
CloneWindowRPC	0
CloseComputeEngineRPC	2
CloseDatabaseRPC	1
CloseRPC	0
ColorTableAttributes	1
ConeAttributes	1
ConnectToMetaDataServerRPC	2
ConnectedComponentsAttributes	1
ConstructDataBinningAttributes	1
ConstructDataBinningRPC	0
ContourAttributes	1
CoordSwapAttributes	1
CopyActivePlotsRPC	0
CopyAnnotationsToWindowRPC	2
CopyLightingToWindowRPC	2
CopyPlotsToWindowRPC	2
CopyViewToWindowRPC	2
CreateBondsAttributes	1
CreateDatabaseCorrelationRPC	4
CreateNamedSelectionRPC	1
CurveAttributes	1
CylinderAttributes	1
DataBinningAttributes	1
DatabaseMetaData	1
DeIconifyAllWindowsRPC	0
DeferExpressionAttributes	1
DeleteActiveAnnotationObjectsRPC	0
DeleteActivePlotsRPC	0
DeleteDatabaseCorrelationRPC	1
DeleteNamedSelectionRPC	1
DeletePlotDatabaseKeyframeRPC	2
DeletePlotKeyframeRPC	2
DeleteViewKeyframeRPC	1
DeleteWindowRPC	0
DemoteOperatorRPC	1
DetachRPC	0
DisableRedrawRPC	0
DisplaceAttributes	1
DrawPlotsRPC	1
DualMeshAttributes	1
EdgeAttributes	1
ElevateAttributes	1
EnableToolRPC	2
EnableToolbarRPC	2
ExportColorTableRPC	1
ExportDBAttributes	1

Continued on next page...

... CLI Events continued

EventName	*ArgCount*
ExportDBRPC	0
ExportEntireStateRPC	1
ExpressionList	1
ExternalSurfaceAttributes	1
ExtrudeAttributes	1
FFTAttributes	1
FTLEAttributes	1
FileOpenOptions	1
FilledBoundaryAttributes	1
FluxAttributes	1
GetProcInfoRPC	3
GetQueryParametersRPC	1
GlobalAttributes	1
GlobalLineoutAttributes	1
HideActiveAnnotationObjectsRPC	0
HideActivePlotsRPC	0
HideAllWindowsRPC	0
HideToolbarsForAllWindowsRPC	0
HideToolbarsRPC	0
HistogramAttributes	1
IconifyAllWindowsRPC	0
ImportEntireStateRPC	2
ImportEntireStateWithDifferentSourcesRPC	3
IndexSelectAttributes	1
InitializeNamedSelectionVariablesRPC	1
InteractorAttributes	1
InverseGhostZoneAttributes	1
InvertBackgroundRPC	0
IsosurfaceAttributes	1
IsovolumeAttributes	1
KeyframeAttributes	1
LabelAttributes	1
LagrangianAttributes	1
LineoutAttributes	1
LoadNamedSelectionRPC	1
LowerActiveAnnotationObjectsRPC	0
MaterialAttributes	1
MenuQuitRPC	1
MeshAttributes	1
MeshManagementAttributes	1
ModelFitAtts	1
MoleculeAttributes	1
MoveAndResizeWindowRPC	5
MovePlotDatabaseKeyframeRPC	3
MovePlotKeyframeRPC	3
MovePlotOrderTowardFirstRPC	1
MovePlotOrderTowardLastRPC	1
MoveViewKeyframeRPC	2
MoveWindowRPC	3

Continued on next page...

... CLI Events continued

EventName	*ArgCount*
MultiCurveAttributes	1
MultiresControlAttributes	1
OnionPeelAttributes	1
OpenCLIClientRPC	1
OpenClientRPC	3
OpenComputeEngineRPC	2
OpenDatabaseRPC	4
OpenGUIClientRPC	1
OpenMDServerRPC	2
OverlayDatabaseRPC	1
ParallelCoordinatesAttributes	1
PersistentParticlesAttributes	1
PickAttributes	1
PlotList	1
PoincareAttributes	1
PrintWindowRPC	0
PrinterAttributes	1
ProcessAttributes	1
ProcessExpressionsRPC	0
ProjectAttributes	1
PromoteOperatorRPC	1
PseudocolorAttributes	1
QueryAttributes	1
QueryOverTimeAttributes	1
QueryRPC	1
RaiseActiveAnnotationObjectsRPC	0
ReOpenDatabaseRPC	2
RecenterViewRPC	0
RedoViewRPC	0
RedrawRPC	0
ReflectAttributes	1
RemoveAllOperatorsRPC	0
RemoveLastOperatorRPC	0
RemoveOperatorRPC	1
RenamePickLabelRPC	1
RenderingAttributes	1
ReplaceDatabaseRPC	2
ReplicateAttributes	1
RequestMetaDataRPC	2
ResampleAttributes	1
ResetAnnotationAttributesRPC	0
ResetAnnotationObjectListRPC	0
ResetInteractorAttributesRPC	0
ResetLightListRPC	0
ResetLineoutColorRPC	0
ResetMaterialAttributesRPC	0
ResetMeshManagementAttributesRPC	0
ResetOperatorOptionsRPC	1
ResetPickAttributesRPC	0

Continued on next page...

... CLI Events continued

EventName	*ArgCount*
ResetPickLetterRPC	0
ResetPlotOptionsRPC	1
ResetQueryOverTimeAttributesRPC	0
ResetViewRPC	0
ResizeWindowRPC	3
RevolveAttributes	1
SaveNamedSelectionRPC	1
SaveViewRPC	0
SaveWindowAttributes	1
SaveWindowRPC	0
ScatterAttributes	1
SendSimulationCommandRPC	4
SetActivePlotsRPC	2
SetActiveTimeSliderRPC	1
SetActiveWindowRPC	1
SetAnimationAttributesRPC	0
SetAnnotationAttributesRPC	0
SetAnnotationObjectOptionsRPC	0
SetAppearanceRPC	0
SetCenterOfRotationRPC	1
SetCreateMeshQualityExpressionsRPC	1
SetCreateTimeDerivativeExpressionsRPC	1
SetCreateVectorMagnitudeExpressionsRPC	1
SetDefaultAnnotationAttributesRPC	0
SetDefaultAnnotationObjectListRPC	0
SetDefaultFileOpenOptionsRPC	0
SetDefaultInteractorAttributesRPC	0
SetDefaultLightListRPC	0
SetDefaultMaterialAttributesRPC	0
SetDefaultMeshManagementAttributesRPC	0
SetDefaultOperatorOptionsRPC	1
SetDefaultPickAttributesRPC	0
SetDefaultPlotOptionsRPC	1
SetDefaultQueryOverTimeAttributesRPC	0
SetGlobalLineoutAttributesRPC	0
SetInteractorAttributesRPC	0
SetKeyframeAttributesRPC	0
SetLightListRPC	0
SetMaterialAttributesRPC	0
SetMeshManagementAttributesRPC	0
SetNamedSelectionAutoApplyRPC	1
SetOperatorOptionsRPC	1
SetPickAttributesRPC	0
SetPlotDatabaseStateRPC	3
SetPlotDescriptionRPC	1
SetPlotFollowsTimeRPC	0
SetPlotFrameRangeRPC	3
SetPlotOptionsRPC	1
SetPlotOrderToFirstRPC	1

Continued on next page...

... CLI Events continued

EventName	*ArgCount*
SetPlotOrderToLastRPC	1
SetPlotSILRestrictionRPC	0
SetQueryFloatFormatRPC	1
SetQueryOverTimeAttributesRPC	0
SetRenderingAttributesRPC	0
SetStateLoggingRPC	0
SetSuppressMessagesRPC	1
SetTimeSliderStateRPC	1
SetToolUpdateModeRPC	1
SetToolbarIconSizeRPC	0
SetTreatAllDBsAsTimeVaryingRPC	1
SetTryHarderCyclesTimesRPC	1
SetView2DRPC	0
SetView3DRPC	0
SetViewAxisArrayRPC	1
SetViewCurveRPC	0
SetViewExtentsTypeRPC	1
SetViewKeyframeRPC	0
SetWindowAreaRPC	1
SetWindowLayoutRPC	1
SetWindowModeRPC	1
ShowAllWindowsRPC	0
ShowToolbarsForAllWindowsRPC	0
ShowToolbarsRPC	0
SliceAttributes	1
SmoothOperatorAttributes	1
SphereSliceAttributes	1
SpreadsheetAttributes	1
StaggerAttributes	1
StreamlineAttributes	1
SubsetAttributes	1
SuppressQueryOutputRPC	1
SurfaceNormalAttributes	1
TensorAttributes	1
ThreeSliceAttributes	1
ThresholdAttributes	1
TimeSliderNextStateRPC	0
TimeSliderPreviousStateRPC	0
ToggleAllowPopupRPC	1
ToggleBoundingBoxModeRPC	0
ToggleCameraViewModeRPC	0
ToggleFullFrameRPC	0
ToggleLockTimeRPC	0
ToggleLockToolsRPC	0
ToggleLockViewModeRPC	0
ToggleMaintainViewModeRPC	0
TogglePerspectiveViewRPC	0
ToggleSpinModeRPC	0
TransformAttributes	1

Continued on next page...

... CLI Events continued

EventName	*ArgCount*
TriangulateRegularPointsAttributes	1
TruecolorAttributes	1
TubeAttributes	1
TurnOffAllLocksRPC	0
UndoViewRPC	0
UpdateColorTableRPC	1
UpdateDBPluginInfoRPC	1
UpdateNamedSelectionRPC	1
VectorAttributes	1
View2DAttributes	1
View3DAttributes	1
ViewCurveAttributes	1
VolumeAttributes	1
WindowInformation	1
WriteConfigFileRPC	0

Chapter 7

Acknowledgments

This document is primarily based on the excellent manual put together by Brad Whitlock of Lawrence Livermore in 2005. Several years afterwards, the content from that manual was converted to serve as online help for the command line interpreter itself. As new routines were added, this online help was updated. In 2010, Jakob van Bethlehem of the University of Groningen wrote a wonderful script to convert the online help to manual form. In 2011, Hank Childs of Lawrence Berkeley merged the descriptions from Brad Whitlock's original manual with the function definitions produced by Jakob's conversion of the online help. The result is this manual.