

Table of Contents

Table of Contents	1
Part 1: Integrating STK with Python	2
Problem Statement	2
Solution	2
FAMILIARIZE YOURSELF WITH THE STK PROGRAMMING INTERFACE HELP	3
Try it	9
A final note on the STK Programming Interface Help System:	10
Integrating STK and Python	10
Part 1: Integrating STK and Python (Jupyter Notebooks)	12
Before you start: Integrating STK with Python (Jupyter Notebooks)	12
Part 2: Integration STK with Python (Anaconda)	21
Before you start: Integrating STK with Anaconda	21

Part 1: Integrating STK with Python

Required Product Licenses: STK Professional, STK Integration

You can obtain the necessary license for the training by visiting

<http://licensing.agi.com/stk/evaluation> or calling AGI support.

Problem Statement

You will be analyzing the behavior of a satellite when it has contact with a ground site. The task will be repetitive and you consider methods of automating the process and extracting the data. Knowing that you can integrate STK with other tools you decide to explore the process.

Solution

Analysis in STK can be integrated and automated with code. You decide to run the process with Python. Using the resources on the STK Help and Github you explore how to model a mission with a script.

From this tutorial you will learn how to:

- Connect STK to a programming interface
- Build a scenario through a script
- Extract data from STK

AGI Techs Say: A recorded PowerPoint presentation (https://p.widencdn.net/ysl41m/Part16_Integration_JB), [Jupyter Notebook](#), and Python script (<STK Install Folder>\Data\Resources\stktraining\scripts) accompany this lesson. It is recommended that you follow the presentation while performing the tasks. If you don't want to use the shared files, you can type the commands right in the Python Command Window.

FAMILIARIZE YOURSELF WITH THE STK PROGRAMMING INTERFACE HELP

Before attempting to code anything, take a moment to familiarize yourself with the [STK Programming Interface Help system](#). After going through the Help, we will dive into Integrating STK and Python. We will do this with two examples:

1. Integrating STK with Python using Jupyter Notebooks
2. Integrating STK with Python using Anaconda

The script will be the same, and the intention is to provide two overviews. Historically, integrating STK Desktop and STK Engine was achieved with either win32com or through the comtypes Python module. In STK 12.1, a new STK Python API was developed to provide the following:

- Cross-platform support: Code written using the new API interacting with STK Engine can be used on Windows and on Linux without modifications (assuming the usual cross platform Python guidelines are followed).
- Usability improvements:
 - The new API provides definitions for all enumerations. With win32com, enumerations had to be defined manually based on the corresponding numerical value.
 - Better IDE support through type hints (based on the typing module).

Resources for Integrating STK are available. You can access these in several different ways including via AGI's website and the associated online STK Programming Help page. You can also use the menu bar in STK by selecting Help → Programming Interface Help, or launch the STK Help via the windows Start Menu by selecting Start → All Programs → STK 12 → STK 12 - Programming Interface Help.









You will find a wealth of information within the STK Programming Interface Help system including integration tutorials, code snippets, decision trees, and library references.

The help system tree looks like this:



STK Programming Help

Welcome to STK Programming Interface Help

-  What's New
-  Migration Guide
-  Tutorials, Code Samples, Code Snippets, and Online Resources
-  Select the Right Technology
-  Using Core Libraries
-  Development Environments
-  Library Reference
-  Getting Help

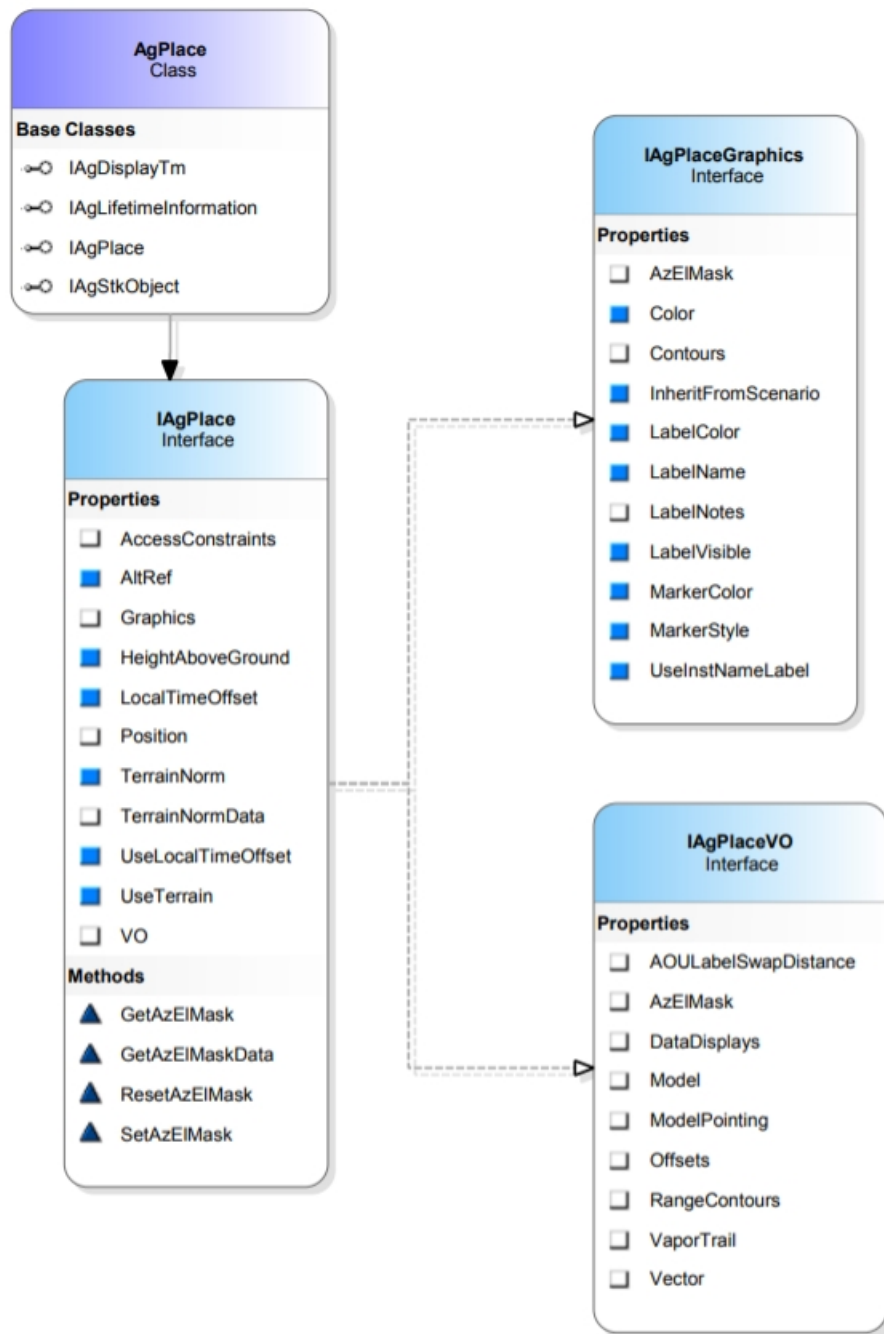
Within the Using Core Libraries directory, you will find additional information on using the STK Object Model and Connect.



STK Programming Help

- 📄 Welcome to STK Programming Interface Help
- 📁 What's New
- 📁 Migration Guide
- 📄 Tutorials, Code Samples, Code Snippets, and Online Resources
- 📁 Select the Right Technology
- 📁 Using Core Libraries**
 - 📁 STK Object Model
 - 📁 Connect
- 📁 Development Environments
- 📁 Library Reference
- 📄 Getting Help

The STK Object Model section listed under Using Core Libraries contains useful code snippets in various programming languages. It also contains information on the various COM libraries that make up the STK Object Model. This information includes useful diagrams that help visualize how the STK Object Model is structured. For example, this small excerpt from the STK Object diagrams which outlines a portion of the basic STK Object Model representation of a Place object:



In the Connect section listed under Using Core Libraries you will find useful Getting Started information outlining the basics of Connect command and response formats. The section also contains useful code snippets that demonstrate the syntax for various Connect commands. Here is an example of a code snippet showing how to create a new missile object:

Create New Missile

[Connect]

```
root.ExecuteCommand('New / */Missile MyMissile');
```

Note: This snippet utilizes STK Object Model to send the Connect command. The raw Connect command is the string being passed to `root.ExecuteCommand()`.

You can and should explore more of the help sections, but one last important section to point out is the Library Reference section.

Under Library Reference, you will find documentation on the STK Object Model, Connect Command Library, Data Providers, and more. The STK Object Model documentation is organized by the individual libraries, so there is a section for STK Objects, a section for STK Util, and so on with each section encapsulating certain aspects of functionality. Within each section items are broken down further by their type. When needed, this can make it quicker to find the exact item you are looking for.

- Library Reference

- Controls

- STK Object Model**

- Abbreviations

- Naming Conventions

- STK Objects Hierarchy

- STK Objects

- Overview

- Objects

- Interfaces

- Methods

- Properties

- Events

- Enumerations

- STK Util

- Overview

- Objects

- Interfaces

- Methods

- Properties

- Enumerations

- STK Astrogator

- STK Graphics Primitives

- STK Vector Geometry Tool

- STK Aviator

- STK Aviator MATLAB

The Connect Command Library, listed under Library Reference, houses the documentation for all of the Connect commands. In this case commands are listed in several different formats, including alphabetically, by object, and by module to assist in locating the exact reference needed.



STK Programming Help

Library Reference

- Controls
- STK Object Model
- Application Object Model
- STK Viewer Application Object Model
- RT3 Object Model
- STK Engine Plugins
- STK UI Plugins
- Data Providers Reference

Connect Command Library

- Command Syntax Elements
- Added and Changed Commands
- Alphabetical Listing
- Deprecated Commands
- Listing by Version
- Application Options
- 2D Graphics window
- 2D Object Graphics
- 3D Graphics window
- 3D Object Graphics
- Object Tools
- Listings by Object
- Listings by Module
- Astrogator Connect
- Component Browser Connect
- Aviator (MissionModeler) Connect

Try it

To find help for the AgStkObjectRoot class:

1. Click Library Reference → STK Object Model → STK Objects
2. Scroll down the alphabetical list until you find the entry entitled "AgStkObjectRoot"
3. Click AgStkObjectRoot to display detailed information and important links, including a link to the IAgStkObjectRoot interface
4. From there, you can access a listing of all the members (methods and properties) of the interface associated with the AgStkObjectRoot object. Similarly, you can find help for any other object or interface, as well as any method, property or enumeration.

A quick way to find help on a given type is to use the Search tab of the Help system. For example, suppose you are interested in the AgECoordinateSystem enum. If you enter "AgECoordinateSystem" in the search field and click List Topics, a list of pages containing that term appears. The entry for the AgECoordinateSystem reference page will appear at the top. Select that entry to display a page defining the enum and listing its members.

A final note on the STK Programming Interface Help System:

In addition to help on programming issues, the help provides information about the real-world context in which the application is to be used. For example, the STK Object Model help page for the SpinAxisConeAngle property tells you that it is writable, is of the Variant data type, and belongs to the IAgSnPtSpinning interface. In addition, it tells you that the property represents "the cone angle used in defining the spin axis; i.e., the angle between the spin axis and the sensor boresight." This latter information is useful in deciding whether and how to use the property in your application, but since the help is mainly intended to provide guidance on programming issues it is best to also reference the STK Help System, where there is generally more information. For example, the help page for "Spinning Sensor Pointing" not only gives more detailed context information but also includes a drawing of spin axis geometry that illustrates the spin axis cone angle quite clearly.

Integrating STK and Python

AGI Techs Say: The (<https://help.agi.com/stkdevkit/index.html>) has several code snippets to assist you. Visit our FAQ site to learn more about (<http://agiweb.force.com/faqs/articles/Keyword/Getting->

Started-STK-COM-integration-using-Python). Additional Python Resources are on Github (<https://github.com/AnalyticalGraphicsInc/STKCodeExamples/tree/master/StkAutomation/Python>).

AGI Techs Say: This lesson is broken up into two different parts. Each covers the same content, just using different programs (Jupyter vs Anaconda). Using either method or lesson is acceptable as they both cover the same topic. It is simply a personal preference as to which topic you would like to follow. We will begin by covering Jupyter Notebooks.

Part 1: Integrating STK and Python (Jupyter Notebooks)

Before you start: Integrating STK with Python (Jupyter Notebooks)

This lesson was written with Python 3.6, but you can use any version of Python. See this [FAQ](#) for details on compatible versions. . Before starting this lesson verify that you have a working Python environment. In this example, Jupyter Notebooks are used. You may also use your preferred Python environment (i.e. WinPython, Spyder, Anaconda, or others).

For new users interested in using Jupyter, it is first recommended to install Anaconda from the STK Full Install or from (<https://www.anaconda.com/products/individual>) which will install Python, Jupyter, and commonly used packages. Once Anaconda is installed you can open a new Anaconda terminal and run a Jupyter notebook with the command

```
jupyter notebook
```

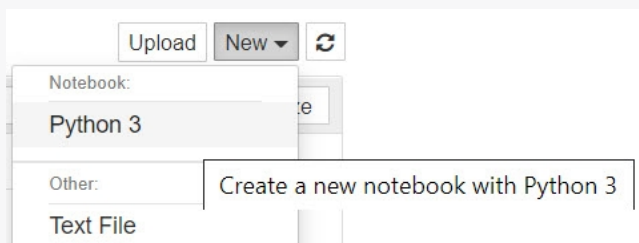
This launches a new tab in the browser. You can now start using the notebook.

TASK: INTEGRATING STK AND PYTHON

There are multiple ways to open Jupyter Notebooks. You may use the script linked above or follow below and open the tool using the Windows Start Menu.

1. Open the Windows start menu.
2. Search for and select the Jupyter Notebook.
3. This will open up a Jupyter Terminal and a new tab in your web browser.
4. Navigate to a folder location where you would like to save your script.

5. In the top right of the web-page select New, then Python 3. This will open a new browser tab. This is where we will write our script.



Note: The most recent version of the notebook file for this lesson is available for [download](#). You can also use the .ipynb file from the STK Install (located at STK Install Folder\Data\Resources\stktraining\scripts). In any situation, it would be useful to have it open as a resource.

6. Save the notebook and give it a unique name.

TASK: CREATE A NEW INSTANCE OF STK FROM JUPYTER NOTEBOOKS

We will be using Jupyter notebooks. We will add cells and enter our script into sections and run them. You may also use the completed script by adding or writing in any missing lines of code. If you are building your script from scratch, use the following commands and paste them into the notebook cell:

1. Set up your workspace.

Note: The new STK Python API is only available with STK 12.1 or later. If not installed, use pip to install it. Example code: `python - m pip install "C:\Program Files\AGI\STK 12\bin\AgPythonAPI\agi.stk12-12.2.0-py3-none-any.whl"` If using an older version of STK then use win32com or comtypes.

```
# Set up your Python workspace
```

```
from agi.stk12.stkdesktop import STKDesktop
```

```
from agi.stk12.stkobjects import *
```

2. Create a new instance of STK.

```
# Create a new instance of STK12. Optional arguments set the  
application visible state and the user-control (whether the  
application remains open after exiting python.
```

```
stk = STKDesktop.StartApplication(visible=True, userControl=True)
```

Note: # NOTE FOR STK WEB: you can take advantage of STK/SDF SSO by changing your script to connect to an active instance instead of creating a new instance of STK:

```
# stk = STKDesktop.AttachToApplication()
```

3. Grab a handle on the STK application root.

Recall that the AgStkObjectRoot object is at the apex of the STK Object Model. The associated IAgStkObjectRoot interface will provide the methods and properties to load or create new scenarios and access the Object Model Unit preferences. Through the stk command you have a pointer to the IAgUiApplication interface; however the STK Python API provides a direct handle to the IAgStkObjectRoot via the Root property in STKDesktop or the NewObjectRoot() method in STKEngine.

```
root = stk.Root
```

4. Check that the root object has been built correctly, check the type(). The output will be agi.stk.stkobjects.AgStkObjectRoot.

```
type(root)
```

5. Once the above lines are entered into the cell, click the "Run the cell" button. This will create a new STK window.

TASK: BUILD A NEW SCENARIO

Now that you have launched STK via the Python interface, let's see if we can create a new scenario and set the time period via Python. We will create a new scenario, analysis period and reset the animation time.

1. Click the insert cell below button to add a new cell.
2. Copy the following code to create a new scenario.

```
# 1. Create a new scenario.
```

```
root.NewScenario("Python_STK_Training")
```

```
scenario = root.CurrentScenario
```

3. Copy the following code to set analysis period.

```
# 2. Set the analytical time period.
```

```
scenario.SetTimePeriod('Today', '+24hr')
```

4. Copy the following code to reset the animation time.

```
# 3. Reset the animation time to the newly established start time.
```

```
root.Rewind();
```

5. Click the "Run the cell" button.

TASK: INSERT AND CONFIGURE OBJECTS

With a new scenario created, it's time to populate the scenario with objects. Use the STK Python API and the STK Connect commands, via the ExecuteCommand method, to create a facility and a LEO satellite.

1. Click the insert cell below button to add a new cell.
2. Copy the following code to add a target object to the scenario. Casting the object returned from the New() method allows for better intellisense in your IDE but is optional; the object returned will be AgTarget at runtime even without the case.

```
# 1. Add a target object to the scenario.
```

```
target = AgTarget(scenario.Children.New  
(AgESTKObjectType.eTarget,"GroundTarget"))
```

3. Copy the following code to move the target object to the desired location.

```
#2. Move the Target object to a desired location.
```

```
target.Position.AssignGeodetic(50,-100,0)
```

4. Copy the following code to add a satellite object to the scenario.

```
#3. Add a Satellite object to the scenario. Create a LEO satellite  
using Python and STK Connect commands, via the ExecuteCommand method.
```

```
satellite = AgSatellite(root.CurrentScenario.Children.New  
(AgESTKObjectType.eSatellite,"LeoSat"))
```

```
#Examine the below connect command before running. In it we will be  
using the Set State Classical connect command. Rather than manually  
setting the times we will use the define scenario times. Print them  
to confirm.
```

```
print(scenario.StartTime)
```

```
print(scenario.StopTime)
```


5. Copy the following code to propagate the satellite object for the length of the scenario.

#4. Propagate the Satellite object's orbit.

```
root.ExecuteCommand('SetState */Satellite/LeoSat Classical TwoBody "'  
+ str(scenario.StartTime) + '" "' + str(scenario.StopTime) + '" 60  
ICRF "' + str(scenario.StartTime) + '" 7200000.0 0.0 90 0.0 0.0  
0.0');
```

6. Click the "Run the cell" button.

TASK: COMPUTE ACCESS BETWEEN OBJECTS

You now have a scenario with a Target object and a Satellite object. Determine when the Satellite object can access the Target object.

1. Browse to the STK Programming Interface Help files.
2. Locate and manually enter code into Python to compute an access between two STK Objects using the IAgStkObject interface. The access is between the Satellite object and the Target object.
3. If you cannot locate the code, expand the following
 - a. The location of the required code snippets is STK Programming Interface > Using Core Libraries > STK Object Model > Python Code Snippets. Locate STK Objects > Access. The required snippet is Compute an access between two STK Objects (using IAgStkObject interface).

```
access = satellite.GetAccessToObject(target)
```

```
access.ComputeAccess();
```

4. In the Jupyter Notebook, click the insert cell below button to add a new cell.
5. Enter the above text to the new cell and click the "Run the cell" button.

TASK: RETRIEVE ACCESS DATA FROM STK

Now that we have computed access between our two objects, we can use the STK data providers to pull data out of our scenario.

1. Click the insert cell below button to add a new cell.
2. Copy the following code to add calls to the access data provider. This will retrieve and view the access data in Python.

```
accessDP = access.DataProviders.Item('Access Data')

results = accessDP.Exec(scenario.StartTime, scenario.StopTime)

accessStartTimes = results.DataSets.GetDataSetByName('Start
Time').GetValues()

accessStopTimes = results.DataSets.GetDataSetByName('Stop
Time').GetValues()

print(accessStartTimes,accessStopTimes)
```

3. Click the "Run the cell" button.

Note: Generating the Start & Stop times in Python can also be done using the following lines of code.

```
accessIntervals = access.ComputedAccessIntervalTimes

dataProviderElements = ['Start Time', 'Stop Time']

for i in range(0,accessIntervals.Count):

    times = accessIntervals.GetInterval(i)
    print(times)
```

More information available in [STK Object Model Tutorial](#).

TASK: RETRIEVE THE SATELLITE ALTITUDE DATA FROM STK

1. Click the insert cell below button to add a new cell.
2. Copy the following code to add a call to the satellite's LLA state data provider, and retrieve and view the altitude of the satellite during an access interval. In the following lines, note how the data providers must be "cast" in order to select the data provider folder, sub-folder, and selection.

```
satelliteDP = satellite.DataProviders.Item('LLA State')

satelliteDP2 = satelliteDP.Group.Item('Fixed')

rptElements = ['Time', 'Lat', 'Lon', 'Alt']

satelliteDPTIMEVar = satelliteDP2.ExecElements
(accessStartTimes,accessStopTimes, 60, rptElements)

satelliteAltitude = satelliteDPTIMEVar.DataSets.GetDataSetByName
('Alt').GetValues()

print(satelliteAltitude)
```

3. Click the "Run the cell" button.

TASK: SAVE YOUR SCENARIO

You have just completed the STK integration with Python tutorial using Jupyter Notebooks. Don't forget to save your work. The same lesson is written in the section below using Spyder. Follow along to get additional practice.

Note: To save your scenario you will want to create a new directory and then tell Python to save all the scenario files there.

1. Copy the following code to create a new folder in your scenario directory.

```
import os
```

```
os.mkdir('Python_STK_Training')
```

2. Open a file explorer window and copy the file path.

3. Copy the following code to save your scenario.

```
directory = os.getcwd() + "\Python_STK_Training"
```

```
root.ExecuteCommand('Save / * \' + directory + '\');
```

4. With your scenario safely saved, you can close out of STK. Workflows like these can be expanded and automated for quickly building and analyzing missions.

Part 2: Integration STK with Python (Anaconda)

Before you start: Integrating STK with Anaconda

This lesson utilizes Python 3.5. Before starting this lesson verify that you have a working Python environment. In this example, Spyder and Anaconda are used. You may also use your preferred python environment (i.e. WinPython and Jupyter or others).

If using a more recent version of Python then you can run the exercise through the sectioned script instead of through a command window.

TASK: INTEGRATING STK WITH PYTHON (ANACONDA)

Launch Spyder using the following steps or use your preferred method.

1. Select the Home tab.
2. Click the Open icon.
3. Browse to the script file located at STK Install Folder\Data\Resources\stktraining\scripts.

Note: The most recent version of the python script for this lesson is available for [download](#). You can also use the .py file from the STK Install (located at STK Install Folder\Data\Resources\stktraining\scripts). In any situation, it would be useful to have it open as a resource.

4. In the Current Folder field, double-click the file named Python_Jupyter_STK_Training.py.

Python is up and running.

TASK: CREATE A NEW INSTANCE OF STK FROM PYTHON

We will use the Python script file to automate building a simple STK scenario from which you will extract data into Python. When connected to STK via Python, while creating your variable, using the Tab key after periods enables IntelliSense which displays all of the options available off of the current interface. Try it.

1. Set up your Python workspace.

Note: The new STK Python API is only available with STK 12.1 or later. If not installed, use pip to install it. Example code: `python - m pip install "C:\Program Files\AGI\STK 12\bin\AgPythonAPI\agi.stk12-12.2.0-py3-none-any.whl"` If using an older version of STK then use win32com or comtypes.

```
from agi.stk12.stkdesktop import STKDesktop
```

```
from agi.stk12.stkobjects import *
```

1. Create a new instance of STK. Optional arguments set the application visible state and the user-control (whether the application remains open after exiting Python).

```
# Create a new instance of STK12.
```

```
stk = STKDesktop.StartApplication(visible=True, userControl=True)
```

Note: # NOTE FOR STK WEB: you can take advantage of STK/SDF SSO by changing your script to connect to an active instance instead of creating a new instance of STK:

```
# stk = STKDesktop.AttachToApplication()
```

2. Grab a handle on the STK application root.

Recall that the AgStkObjectRoot object is at the apex of the STK Object Model. The associated IAgStkObjectRoot interface will provide the methods and properties to load or create new scenarios and access the Object Model Unit preferences. Through the stk command you have a pointer to the

IAgUiApplication interface; however the STK Python API provides a direct handle to the IAgStkObjectRoot via the Root property in STKDesktop or the NewObjectRoot() method in STKEngine.

```
root = stk.Root
```

3. Check that the root object has been built correctly, check the type(). The output will be agi.stk.stkobjects.AgStkObjectRoot.

```
type(root)
```

4. Click Enter.
5. In Python, click Run Current Cell or press CTRL + Enter to run the section.

TASK: CREATE A NEW STK SCENARIO FROM INSIDE PYTHON

Now that you have launched STK via the Python interface, let's see if we can create a new scenario and set the time period via Python. We will create a new scenario, analysis period and reset the animation time.

1. In Python, click Run Current Cell or press CTRL + Enter to run the section.
2. Copy the following code to create a new scenario.

```
# 1. Create a new scenario.
```

```
root.NewScenario("Python_STK_Training")
```

```
scenario = root.CurrentScenario
```

3. Copy the following code to set the analysis time period.

```
# 2. Set the analytical time period.
```

```
scenario.SetTimePeriod('Today', '+24hr')
```

4. Copy the following code to reset the animation time.

```
# 3. Reset the animation time to the newly established start time.
```

```
root.Rewind();
```

5. Press CTRL + Enter.

TASK: INSERT AND CONFIGURE OBJECTS

With a new scenario created, it's time to populate the scenario with objects. Take a moment to create a facility and a LEO satellite using Python. Use the STK Python API and the STK Connect commands, via the ExecuteCommand method.

In Python, you can copy and paste all the code from the Editor window (to included comments (#)) from the task, and paste it to the Command Window. Notice how the command to propagate the satellite done via the root.ExecuteCommand() method and connect commands. If you do not have the code, follow the steps below.

1. Copy the following code to add a target object to the scenario. Casting the object returned from the New() method allows for better intellisense in your IDE but is optional; the object returned will be AgTarget at runtime even without the case.

```
#1. Add a target object to the scenario.
```

```
target = AgTarget(scenario.Children.New  
(AgESTKObjectType.eTarget, "GroundTarget"))
```

2. Copy the following code to move the target object to the desired location.

```
#2. Move the Target object to a desired location.
```



```
target.Position.AssignGeodetic(50,-100,0)
```

3. Copy the following code to add a satellite object to the scenario.

```
#3. Add a Satellite object to the scenario. Create a LEO satellite  
using Python and STK Connect commands, via the ExecuteCommand method.
```

```
satellite = AgSatellite(root.CurrentScenario.Children.New  
(AgESTKObjectType.eSatellite,"LeoSat"))
```

```
#Examine the below connect command before running. In it we will be  
using the Set State Classical connect command. Rather than manually  
setting the times we will use the define scenario times. Print them  
to confirm.
```

```
print(scenario.StartTime)
```

```
print(scenario.StopTime)
```

1. Copy the following code to propagate the satellite object for the length of the scenario.

```
#4. Propagate the Satellite object's orbit.
```

```
root.ExecuteCommand('SetState */Satellite/LeoSat Classical TwoBody "'  
+ str(scenario.StartTime) + '" "' + str(scenario.StopTime) + '" 60  
ICRF "' + str(scenario.StartTime) + '" 7200000.0 0.0 90 0.0 0.0  
0.0');
```

4. Click Enter.

Note: The SetState Classical Connect command and syntax can be found [here](#).

TASK: COMPUTE ACCESS BETWEEN OBJECTS

You now have a scenario with a Target object and a Satellite object. Determine when the Satellite object can access the Target object.

1. Browse to the STK Programming Interface Help files.
2. Locate and manually enter code into Python to compute an access between two STK Objects using the IAgStkObject interface. The access is between the Satellite object and the Target object.
3. If you cannot locate the code, expand the following
 - a. The location of the required code snippets is STK Programming Interface > Using Core Libraries > STK Object Model > Python Code Snippets. Locate STK Objects > Access. The required snippet is Compute an access between two STK Objects (using IAgStkObject interface).

```
access = satellite.GetAccessToObject(target)
```

```
access.ComputeAccess();
```

4. Enter the above text and click Enter.

TASK: RETRIEVE ACCESS DATA FROM STK

Now that we have computed access between our two objects, we can use the STK data providers to pull data out of our scenario.

1. Copy and paste the following code to retrieve and view the access data in Python.

```
accessDP = access.DataProviders.Item('Access Data')
```

```
results = accessDP.Exec(scenario.StartTime, scenario.StopTime)
```

```
accessStartTimes = results.DataSets.GetDataSetByName('Start  
Time').GetValues()
```

```
accessStopTimes = results.DataSets.GetDataSetByName('Stop  
Time').GetValues()
```

```
print(accessStartTimes,accessStopTimes)
```

2. Click Enter.

Note: Generating the Start & Stop times in Python can also be done using the following lines of code.

```
accessIntervals = access.ComputedAccessIntervalTimes
```

```
dataProviderElements = ['Start Time', 'Stop Time']
```

```
for i in range(0,accessIntervals.Count):
```

```
times = accessIntervals.GetInterval(i)
```

```
print(times)
```

More information available in [STK Object Model Tutorial](#).

TASK: RETRIEVE THE SATELLITE ALTITUDE DATA FROM STK

Retrieve and view the altitude of the satellite during an access interval.

1. Copy and paste the following code into Python's Command Window. In the following lines, note how the data providers must be "cast" in order to select the data provider folder, sub-folder, and selection.

```
satelliteDP = satellite.DataProviders.Item('LLA State')
```

```
satelliteDP2 = satelliteDP.Group.Item('Fixed')

rptElements = ['Time', 'Lat', 'Lon', 'Alt']

satelliteDPTIMEVar = satelliteDP2.ExecElements
(accessStartTimes, accessStopTimes, 60, rptElements)

satelliteAltitude = satelliteDPTIMEVar.DataSets.GetDataSetByName
('Alt').GetValues()

print(satelliteAltitude)
```

2. Click Enter.

TASK: **SAVE YOUR SCENARIO**

You have just completed the STK integration with Python tutorial using Jupyter Notebooks. Don't forget to save your work.

Note: To save your scenario you will want to create a new directory and then tell Python to save all the scenario files there.

1. Copy the following code to create a new folder in your scenario directory.

```
import os

os.mkdir('Python_STK_Training')
```

2. Open a file explorer window and copy the file path. The next line will use your custom path.
3. Copy the following code to save your scenario.

```
directory = os.getcwd() + "\Python_STK_Training"
```

```
root.ExecuteCommand('Save / * \' + directory + '\');
```

4. With your scenario safely saved, you can close out of STK. Workflows like these can be expanded and automated for quickly building and analyzing missions.