STK Tutorial Using the Object Model

This tutorial shows how to use the STK Object Model in a custom application to accomplish many of the tasks for which you might otherwise use Connect or the STK GUI. Source code is given in C# and Visual Basic.NET. Familiarity with Microsoft Visual Studio and STK are presumed.

Source Code Location: The source code for this tutorial can be found in the following text file:

```
<STK Install Folder>\Help\LinkedDocuments\ObjectModelTutorial.txt
```

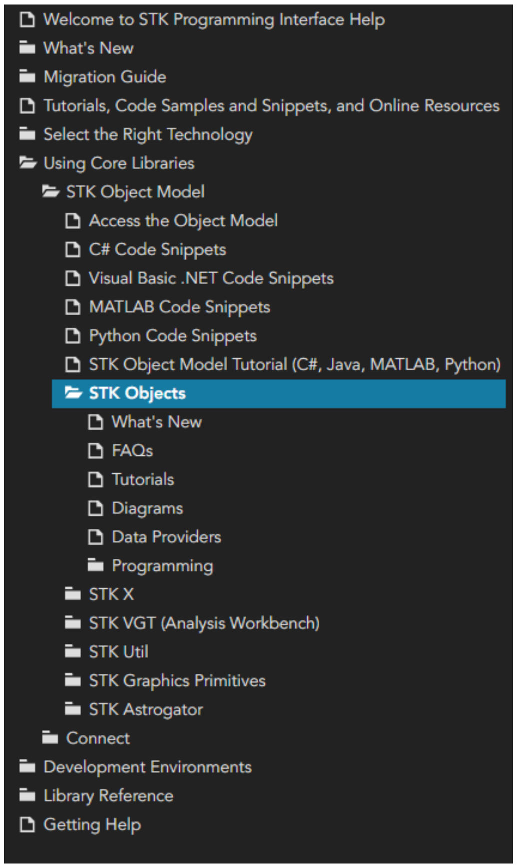
The code is broken into sections corresponding to the sections in this tutorial. Within each section, the C# and VB.NET code are presented in separate subsections.

CONTENTS

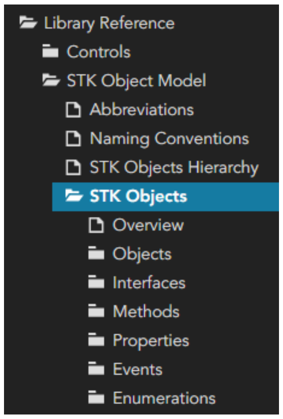
Where to Get Help

Before proceeding with this tutorial, take a moment to familiarize yourself with the parts of the STK Programming Interface Help system that relate to the Object Model and STK X.

In the Contents tab of the Help system, navigate to Using Core Libraries → STK Object Model → STK Objects.



In addition to these introductory sections, the Library Reference for STK Objects is broken down into subsections for the various types in the library.



Try this: To find help for the AgStkObjectRoot class discussed above, click Library Reference→STK Object Model→STK Objects and scroll down the alphabetical list until you find the entry entitled "AgStkObjectRoot". Click AgStkObjectRoot to display a page containing a diagram with several links, including a link to Root (IAgStkObjectRoot Object). From that page you can access a page that lists 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. Click on the Title heading to order the list alphabetically, and the entry for AgECoordinateSystem will appear at the top. Select that entry to display a page defining the enum and listing its members.

> **TIP**: Keep the STK Programming Interface Help system open to the Search tab while you are working through this tutorial. Then you can copy and paste the names of interfaces, enumerations, etc. into the search field to learn more about their meaning and usage.

> **NOTE**: It is not necessary to have STK running to display the Help system. To launch Help, simply open the Windows Start Menu and navigate to All Programs →STK 12 → Help → STK 12 – STK 12 Programming Interface.

In addition to the Object Model (STK Objects and STK Util), you may find it useful to refer to help on STK X controls.

One final note on the STK Programming Interface help: 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 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 it is necessarily somewhat sketchy, since the help is mainly intended to provide guidance on programming issues. However, in the STK Help System, 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.

> **TIP**: For real-world context information on the Object Model types that you use in your application, do not rely solely on the brief definitions in the Object Model help. Refer also to the STK Help System.

Start Visual Studio and Create your Project

You will use two STK X controls for this tutorial. To add these controls and to use the STK Object Model, you must add the following references to your project:

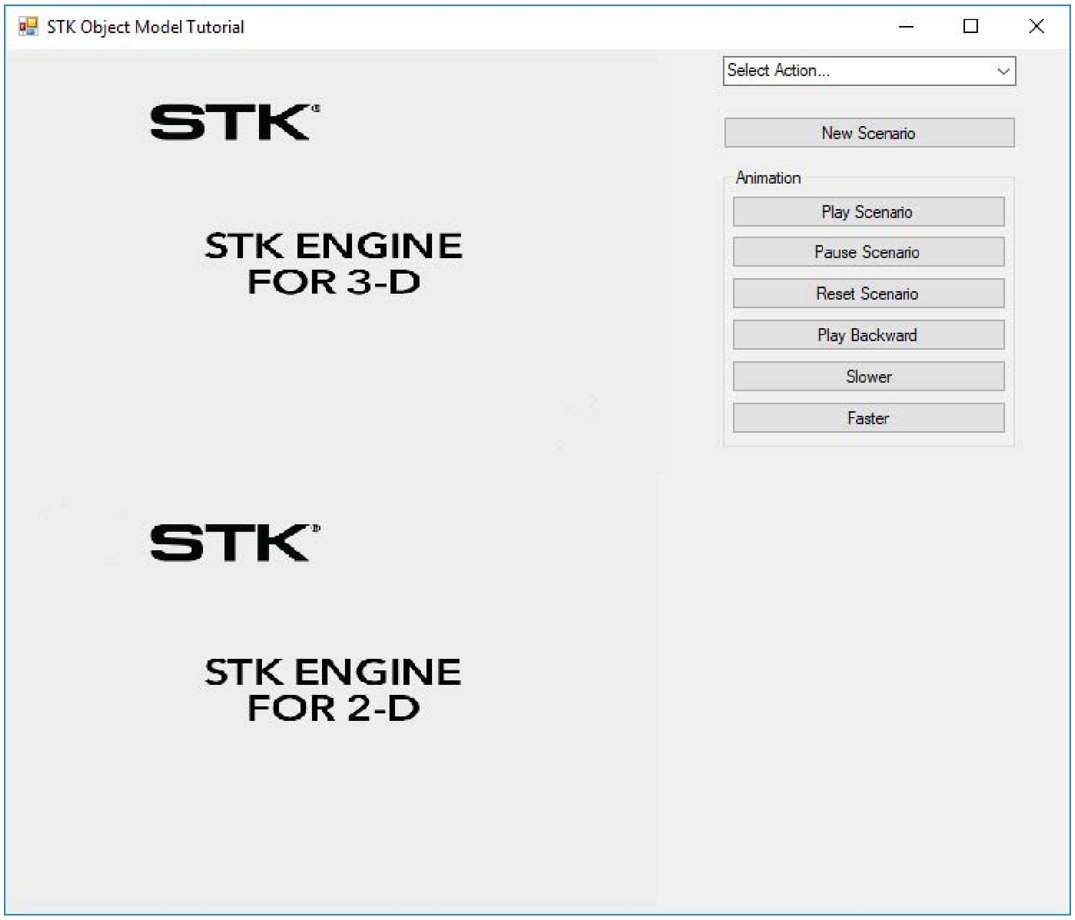
- STK Objects – a COM Library containing types, interfaces, events and classes representing various aspects of the STK application structure.
- STK Util – a library containing objects and enumerations shared by the STK Objects and STK X type libraries.
- STK X - a collection of COM components and ActiveX controls that use AGI's STK Engine embeddable technology.

Here are the steps to set up your project and add the necessary references:

1. Start Visual Studio and click on File and then New Project.
2. Under the Installed templates, navigate to Templates, select either Visual Basic or C#, then in the Windows page choose Windows Forms Application. Configure the rest of your project settings as you see fit. In this tutorial, OMTutorial is used as the project name.
3. Right-click References in the Solution Explorer and select Add References.
4. In the solution explorer under References, select the added reference and make sure the "Embed Interop Types" property is set to false.

Add STK X Controls to your Form

Before you begin writing Object Model code, use the design view in Visual Studio to draw the user interface for your custom application. When you are finished, the GUI you create will look something like this:



The application that you create will allow the user to:

- Create a Scenario and display it in 2D and 3D windows,
- Control the animation of the scenario.

- Select various actions from a combo box: populate the scenario with objects, compute access and impose access constraints.

Here are the steps to create the GUI:

1. Display the automatically-generated `Form1` in design view and stretch the form so that it is approximately square and substantially fills the available workspace (with Visual Studio maximized). Display its properties page and change the Text property from “Form1” to “STK Object Model Tutorial”.
2. If the Toolbox is not already in your workspace, then add it by clicking on View→Toolbox from the menu. Right-click on a blank area in the General section of the toolbox (i.e. neither a control nor the Toolbox title bar), and select Choose Items… from the drop-down menu.
3. On the ".NET Framework Components" tab, click the browse button and browse to the following location: <STK INSTALL FOLDER>/bin/Primary Interop Assemblies
4. Select the AGI.STKX.Controls.Interop.dll and click the Open button, then the OK button.
5. From the `Form1` design view, find AxAgUiAxVOCntrl in the Toolbox, and draw a rectangular shape in the upper left region of your form. Do the same thing with the AxAgUiAx2DCntrl in the lower left region of the form.
6. Place a ComboBox in the upper right corner of the form.
7. Display the properties page for the ComboBox and change its Text property to "Select Action…".
8. Place a button below the ComboBox and change its Text property to "New Scenario".
9. Place a GroupBox below the "New Scenario" button and stretch it so that it will accommodate 6 buttons, arranged vertically. Change its Text property to "Animation".
   If the Embed Interop Types property is set to True, select False from the menu.
10. Arrange 6 buttons vertically in the GroupBox. Starting with the top button and working down, set the Text properties of the buttons to:
    - Play Scenario
    - Pause Scenario
    - Reset Scenario
    - Play Backward
    - Slower
    - Faster

This completes the artistic part of this exercise. Now, on to coding with the Object Model.

Scenario Setup

Add the Core of the STK ObjectModel to your Project

Refer to ObjectModelTutorial.txt for the code segments to be added in this section.

Begin by adding statements at the beginning of the code in Form1.cs (*using* in C#, *Imports* in VB.NET) to gain access to the types defined in the STK Objects and STK Util libraries.

Next, define and initialize an instance of AgStkObjectRoot, the top level object in the Object Model hierarchy within the public partial class, which exposes a set of methods and properties that you will find indispensable in implementing the Object Model in your application. To name just 3 examples, this class includes the following methods:

- NewScenario(): creates a new STK scenario.
- LoadScenario(): loads a scenario file on a specified path.
- ExecuteCommand(): takes a string representing a Connect command as an argument and executes the command.

Finally, add OnFormClosing to close STK Object root properly:

1. Select the Form1 panel.
2. In the Properties panel select the Events button (lightning bolt).
3. Scroll to FormClosing and double click in the space to generate the FormClosing event.

Add Click events for each button:

1. Select the “New Scenario” button.
2. In the Properties panel select the Events button (lightning bolt).
3. Scroll to Click and double click in the space to generate the click event.
4. Repeat steps for all remaining buttons.

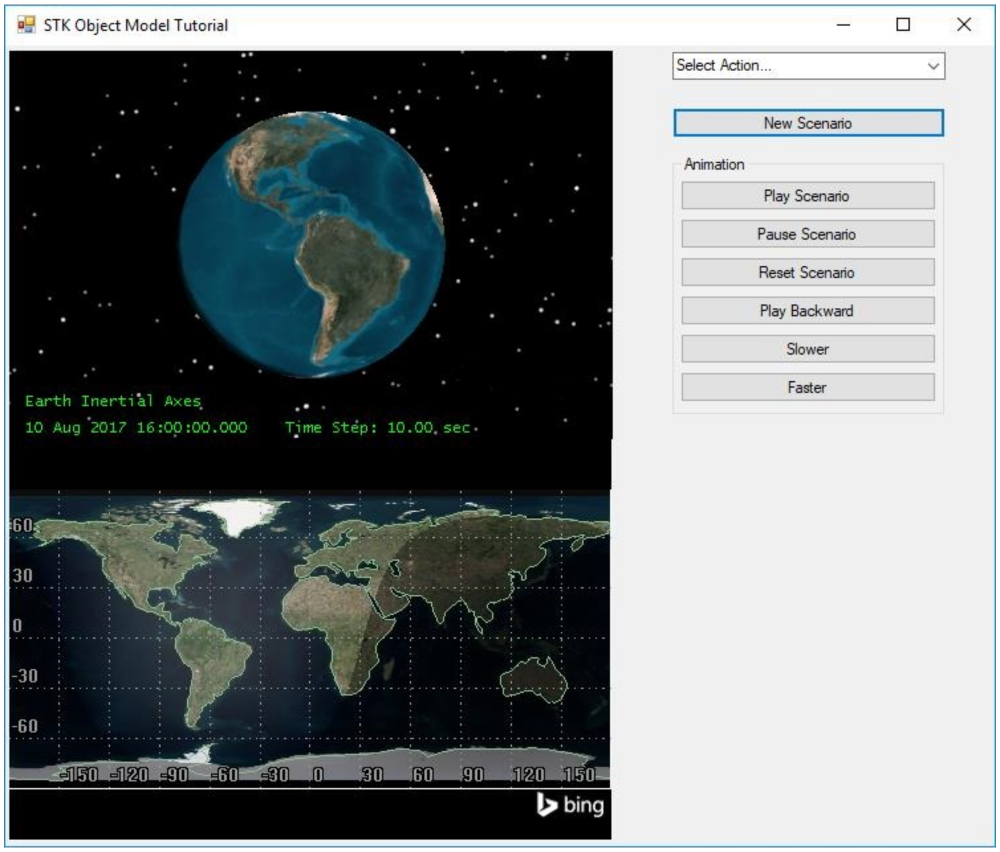
Create a New Scenario

Referring to ObjectModelTutorial.txt, add a function to create a new scenario to the main body of the Form1 class, then add a call to that function to the Click event for the New Scenario button.

In “Section: Add the Core of the STK Object Model to your Project”:

1. Add the code from the ObjectModelTutorial.txt file.

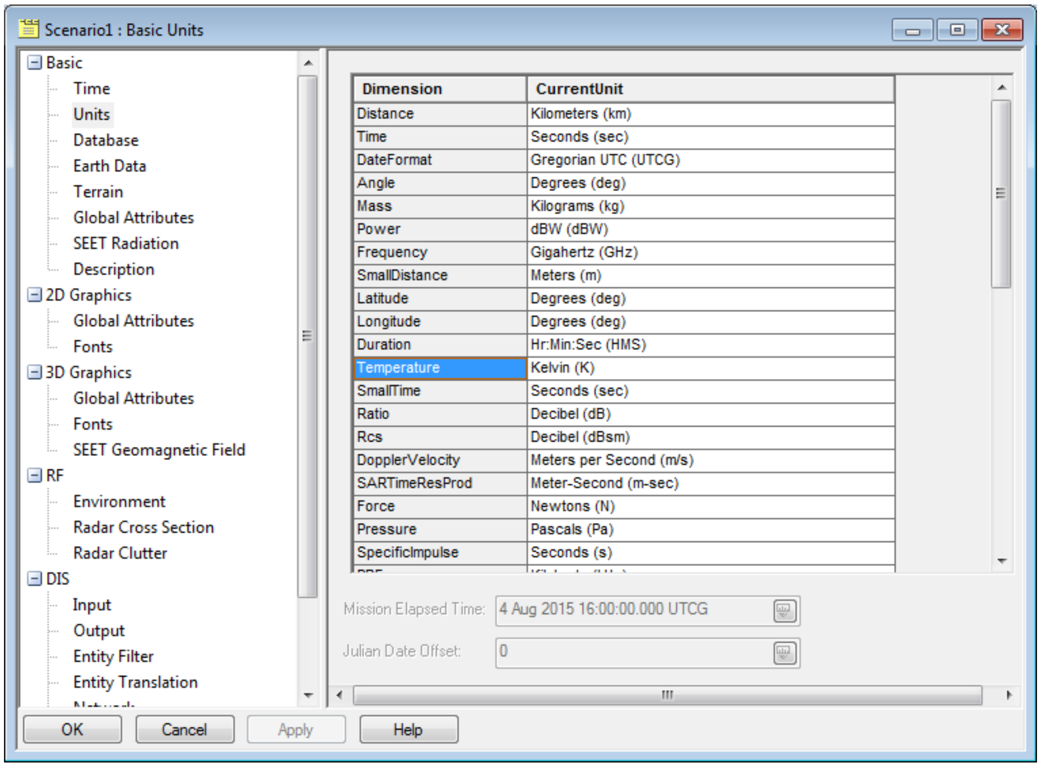
At this point you should be able to build and run your application. Click the New Scenario button to display the 3D Globe and 2D Map:



Set Time Period & Unit Preferences

In STK, the scenario time period and epoch can be set in the Properties Browser. For example, the scenario Start time, Stop time and Epoch can be set in a fairly straightforward way on the Basic Time Period page.
To set the time period and epoch properties in the Object Model, you must first acquire the IAgUnitPrefDimCollection interface (see ObjectModelTutorial.txt for the code to use). Then, using that interface, set the date format to UTCG. Next, acquire an interface to the scenario and use it to set the time period and epoch.

The default units used by the STK GUI are shown on the scenario Basic Units page, which lets you select among available alternative units from a combo box:



Dimension	CurrentUnit
Distance	Kilometers (km)
Time	Seconds (sec)
DateFormat	Gregorian UTC (UTCG)
Angle	Degrees (deg)
Mass	Kilograms (kg)
Power	dBW (dBW)
Frequency	Gigahertz (GHz)
SmallDistance	Meters (m)
Latitude	Degrees (deg)
Longitude	Degrees (deg)
Duration	Hr:Min:Sec (HMS)
Temperature	Kelvin (K)
SmallTime	Seconds (sec)
Ratio	Decibel (dB)
Rcs	Decibel (dBsm)
DopplerVelocity	Meters per Second (m/s)
SARTimeResProd	Meter-Second (m-sec)
Force	Newtons (N)
Pressure	Pascals (Pa)
SpecificImpulse	Seconds (s)

Mission Elapsed Time: 4 Aug 2015 16:00:00.000 UTCG

Julian Date Offset: 0

In the Object Model, you can use the SetCurrentUnit() method to set the any of these units explicitly. Here you will use it to change the preferred unit for Temperature from Kelvin (K) to degrees Celsius (degC).

> **TIP**: If you are in doubt about the available unit abbreviations for a given dimension, launch STK, open the Basic Units page, and display the combo box for the dimension in question.

Add Animation Controls

The needed functionality for adding animation controls to your scenario became available as soon as you initialized the AgStkObjectRoot object. All you need to do as add the appropriate method of that object to the appropriate of each of the animation buttons that you drew on Form1:

1. Add the below code to its respective section in the Form1.cs code.

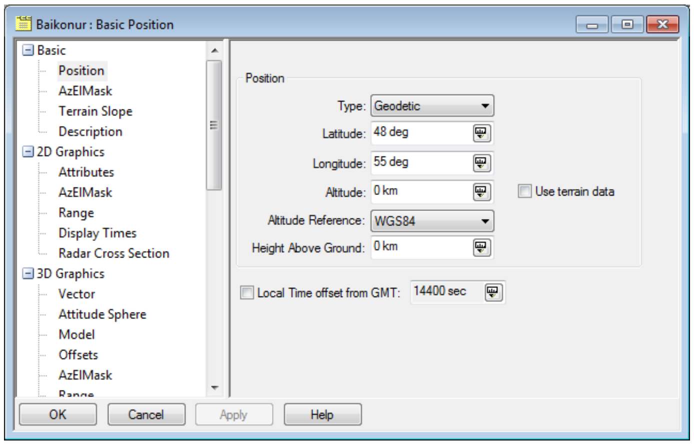
Button text	Code in Click method
Play Scenario	`root.PlayForward()`
Pause Scenario	`root.Pause()`
Reset Scenario	`root.Rewind()`
Play Backward	`root.PlayBackward()`
Slower	`root.Slower()`
Faster	`root.Faster()`

If you are coding in C#, of course, you need to add a semicolon (;) to each of those method statements.

Creating Objects & Setting Their Properties

Create a Facility

There are several ways to specify the position of a facility in STK. You can place a facility anywhere on the Earth (or other central body) via its Basic Position page, which lets you select from among several position types and then specify the coordinates defined by the selected type:



Position
Type: Geodetic
Latitude: 48 deg
Longitude: 55 deg
Altitude: 0 km
☐ Use terrain data
Altitude Reference: WGS84
Height Above Ground: 0 km
☐ Local Time offset from GMT: 14400 sec

In the above example, the facility is defined using the Geodetic position type, make sure the terrain is turned off.

Now refer to the CreateFacilities() method in the accompanying text file. It begins by creating a facility named "Baikonur" and acquiring an interface to the IAgFacility object. position type that you get via a conversion method made available by the IAgPosition interface (through the Position property of the IAgFacility interface). Using the IAgGeodetic interface, set latitude, longitude and altitude values. Then, using the Assign() method of the IAgPosition interface, assign the IAgGeodetic interface back into the facility object.

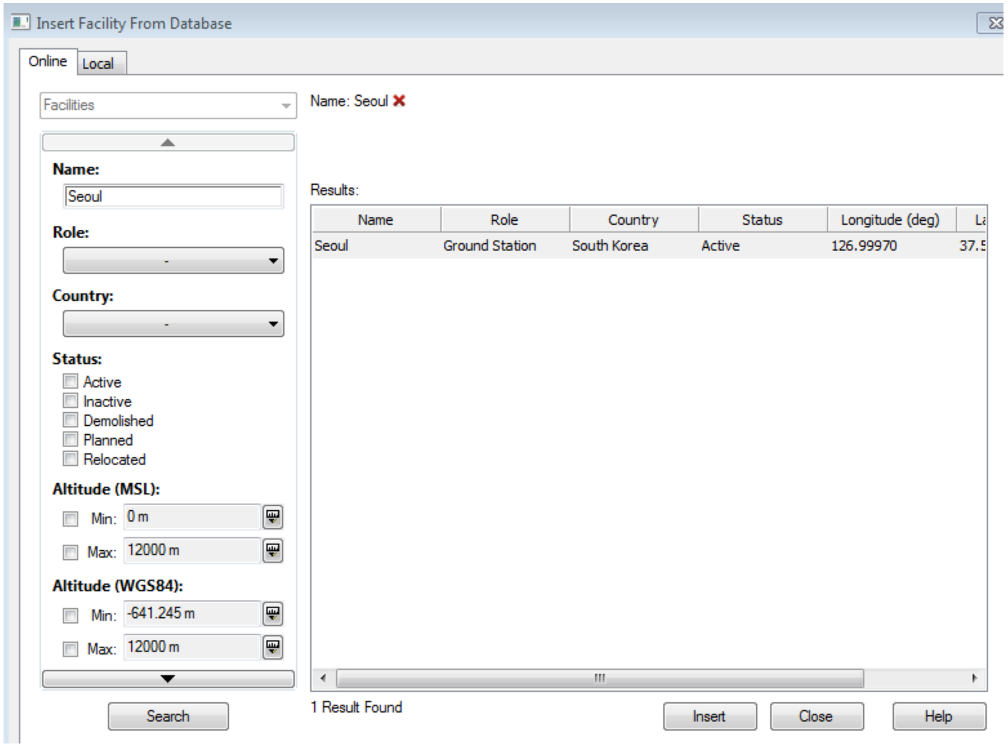
> **NOTE**: If the latter assignment is not done, the facility's position will not update. You will encounter this technicality in other interfaces as well, including any interface that inherits from IAgOrientation.

Additional code lets you add short and long descriptions to the facility object, emulating the functionality of the facility's Basic Description page in STK.

The next block of code in the CreateFacilities() method creates and positions the Perth and Wallops facilities. Since you are setting all the position properties at once, it makes sense to use a one-step method made available by the IAgPosition interface, AssignGeodetic(). In addition to letting us assign latitude, longitude and altitude in one line of code, this method eliminates the steps of acquiring the IAgGeodetic interface and assigning it back into the facility object.

> **TIP**: This coarse-grained approach, which is available for all position types, is ideal where, as here, you are setting all the position properties at once. The finer-grained approach that you used (unnecessarily) for the Baikonur facility is better suited to cases in which only one or a small subset of properties is being tweaked.

Another way to create and position a facility in STK is to use the From Standard Object Database:



Name: Seoul

Name: Seoul

Role: -

Country: -

Status:
- [ ] Active
- [ ] Inactive
- [ ] Demolished
- [ ] Planned
- [ ] Relocated

Altitude (MSL):
- [ ] Min: 0 m
- [ ] Max: 12000 m

Altitude (WGS84):
- [ ] Min: -641.245 m
- [ ] Max: 12000 m

Results:

Name	Role	Country	Status	Longitude (deg)	La
Seoul	Ground Station	South Korea	Active	126.99970	37.5

1 Result Found

The above example illustrates a search of the database using Name (Seoul) as the search criteria.

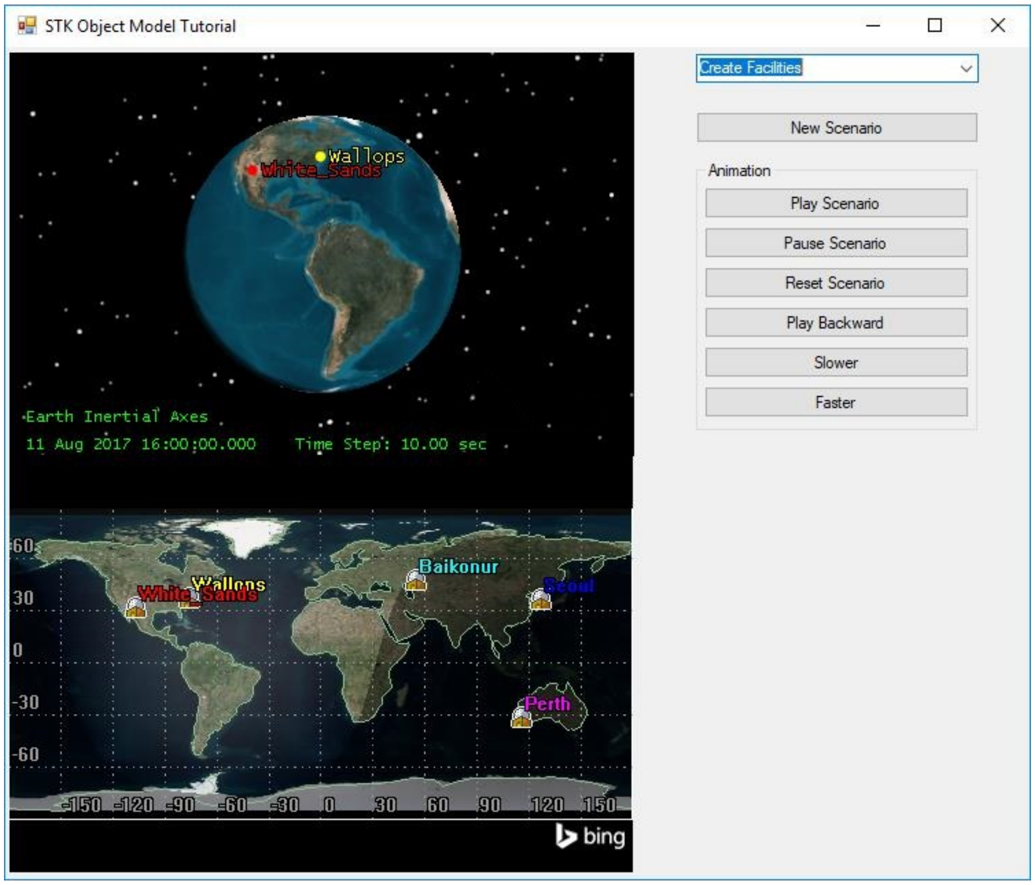
The above process is captured by the Connect command ImportFromDB, which, in turn, can be leveraged by the Object Model via the ExecuteCommand method of the AgStkObjectRoot class. The CreateFacilities() method uses this approach to add two facilities to the scenario: Seoul and WhiteSands. A helper function, GetSTKDataDir(), takes database type as an argument (e.g., Facility, Satellite, City) and executes the GetSTKHomeDir Connect command and returns a string representing the location of the desired database directory.

The variables representing the facilities need to be added to the Form1 class, where they will be available for use by other methods in that class.

In our application, operations affecting the scenario – adding objects, computing access, setting constraints – are triggered by selections from a combo box. Referring again to the text file, you will find code to populate the ComboBox and to call the CreateFacilities() method when the appropriate selection is made.

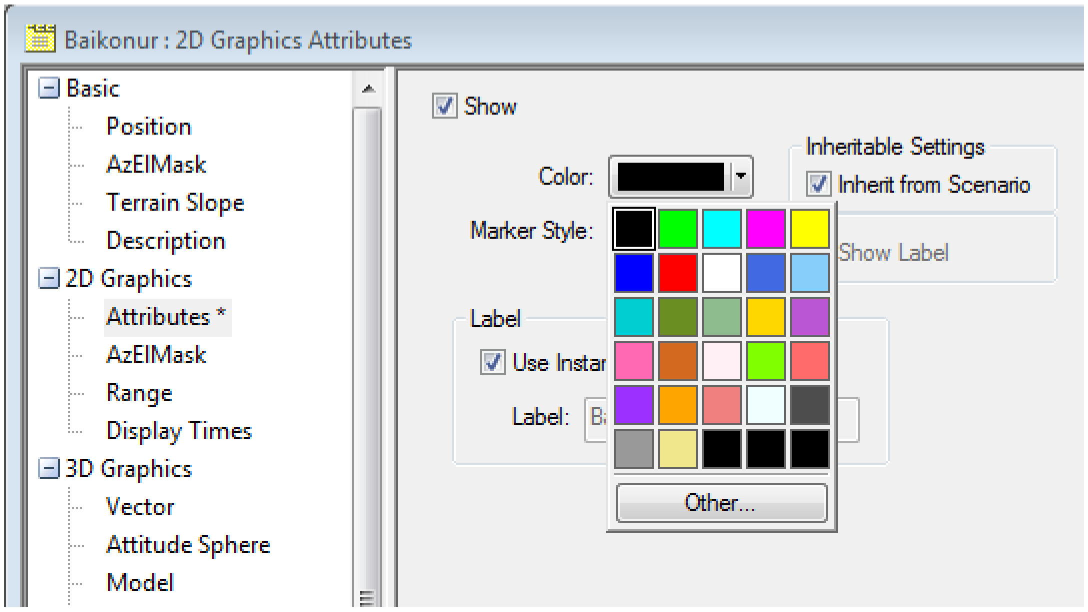
> **NOTE**: The SelectedIndexChanged() method includes calls to the BeginUpdate() and EndUpdate() methods of the AgStkObjectRoot class. The BeginUpdate() method suspends graphic updates, which helps speed up processing of a block of code. The graphic updates are then carried out in a batch when you invoke EndUpdate().

Build and run the application, create a scenario, and select Create Facilities from the combo box:



Set Some Facility Properties

When you create an object in STK, it comes with a host of properties – Basic, 2D Graphics, 3D Graphics, Constraints, etc. – that you can set in the Properties Browser. A simple example is the 2D Graphics Attributes page, available for most objects, including the facility:



The above illustration shows how to change the color in which the facility displays in the 2D and 3D windows.

Similarly, when you create an object using the Object Model and acquire the corresponding interface, you expose a variety of properties for the programmer to configure. As noted above, you declared the variables representing the facilities at the Form1 level, so that they are now available for a new method that changes their display colors. Referring to the accompanying text file, the ChangeFacilitiesColor() method consists of 5 statements – one for each facility – invoking the Color property of IAgFaGraphics interface (via the Graphics property of the IAgFacility interface). Each statement assigns to the Color property a system-defined color.

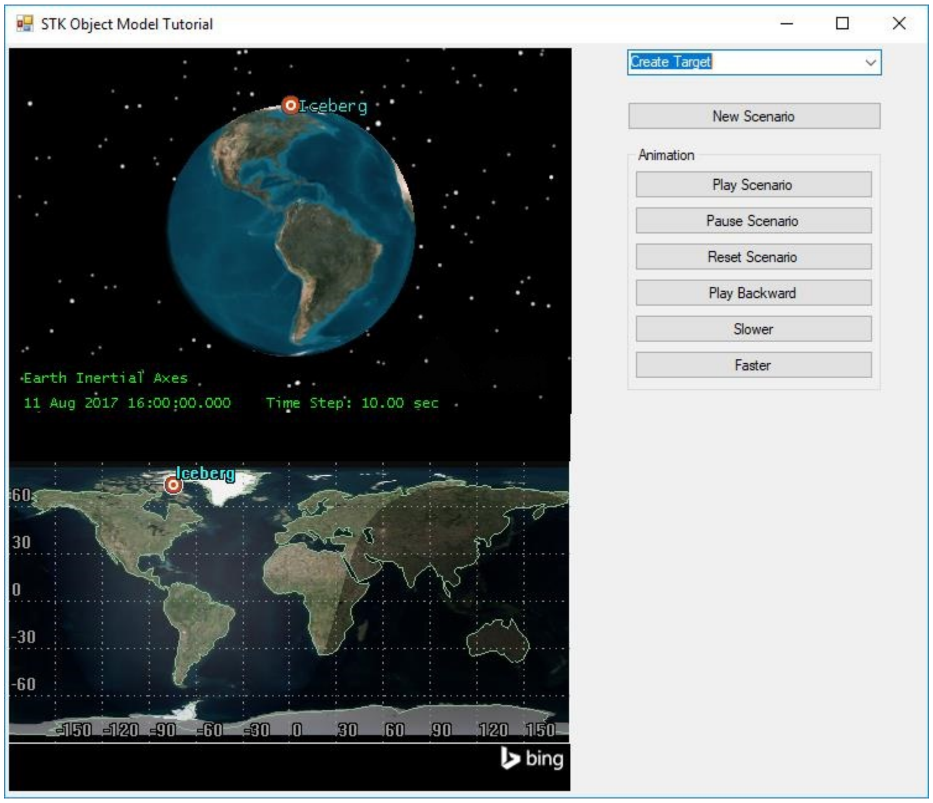
Additional code updates the SelectedIndexChanged() method.

To try out the newly added code, build and run your application, create a scenario, and select Create Facilities and Change Facilities Color from the combo box.

Create a Target

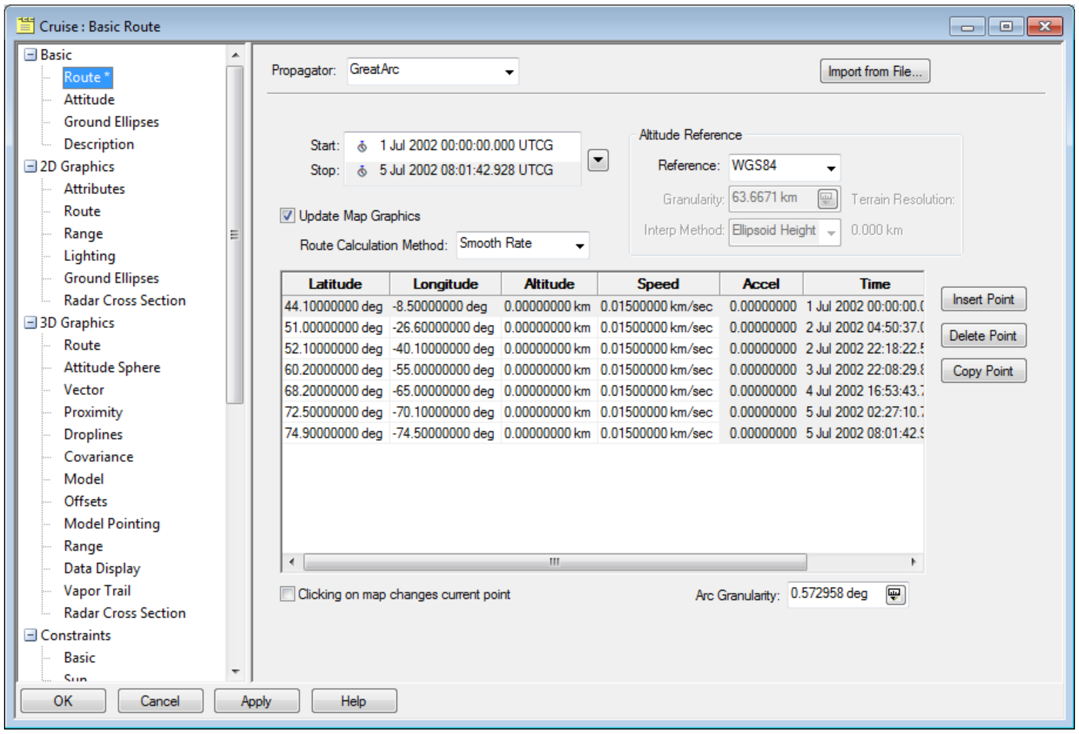
The code for this section should look familiar, since the AssignGeodetic() method used in CreateTarget() is to the same as that which was used to create the Perth and Wallops facilities.

Update the SelectedIndexChanged() method, then build and run your application, create a scenario, and select Create Target from the combo box:



Create a Ship

A ship object in STK is one of 3 objects (the others are the aircraft and ground vehicle objects) that use the Great Arc propagator. For a Great Arc vehicle, you define the route by specifying waypoints, and in the STK GUI, this is done on the Basic Route page:



Propagator: GreatArc

Start: 1 Jul 2002 00:00:00.000 UTCG
Stop: 5 Jul 2002 08:01:42.928 UTCG

Altitude Reference
Reference: WGS84
Granularity: 63.6671 km
Terrain Resolution: 0.000 km
Interp Method: Ellipsoid Height

[✓] Update Map Graphics
Route Calculation Method: Smooth Rate

Latitude	Longitude	Altitude	Speed	Accel	Time
44.10000000 deg	-8.50000000 deg	0.00000000 km	0.01500000 km/sec	0.00000000	1 Jul 2002 00:00:00.
51.00000000 deg	-26.60000000 deg	0.00000000 km	0.01500000 km/sec	0.00000000	2 Jul 2002 04:50:37.
52.10000000 deg	-40.10000000 deg	0.00000000 km	0.01500000 km/sec	0.00000000	2 Jul 2002 22:18:22.
60.20000000 deg	-55.00000000 deg	0.00000000 km	0.01500000 km/sec	0.00000000	3 Jul 2002 22:08:29.
68.20000000 deg	-65.00000000 deg	0.00000000 km	0.01500000 km/sec	0.00000000	4 Jul 2002 16:53:43.
72.50000000 deg	-70.10000000 deg	0.00000000 km	0.01500000 km/sec	0.00000000	5 Jul 2002 02:27:10.
74.90000000 deg	-74.50000000 deg	0.00000000 km	0.01500000 km/sec	0.00000000	5 Jul 2002 08:01:42.

[ ] Clicking on map changes current point
Arc Granularity: 0.572958 deg

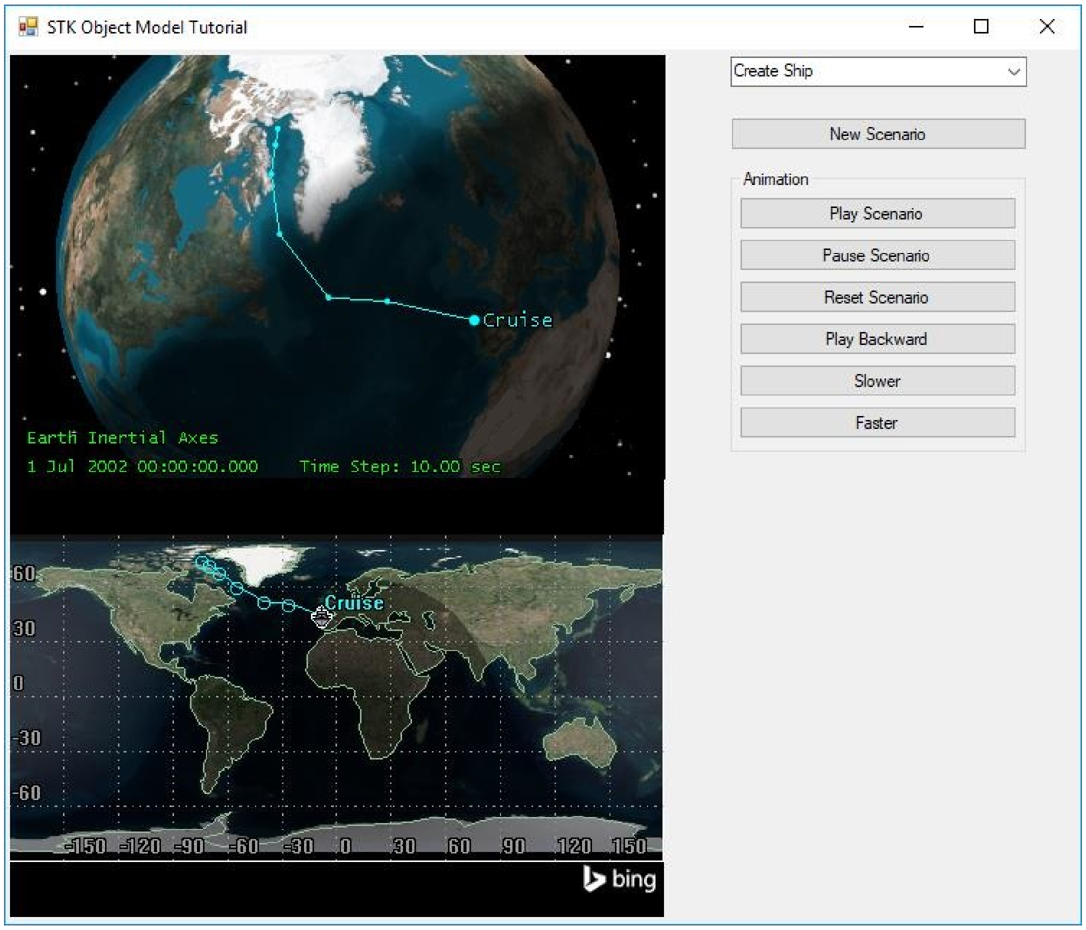
Each row in the table represents a waypoint, with the specified latitude, longitude, altitude, speed, acceleration, time and turn radius. You may notice that two of the columns – those for acceleration and time – are grayed out. Those values are read-only, because the Route Calculation Method is set to Smooth Rate, which derives acceleration and time from speed and position.

The above functionality is captured in the Object Model through the IAgVeWaypointsCollection, IAgVeWaypointsElement and IAgVePropagatorGreatArc interfaces. Here are the highlights:

- A convenience method, AddWaypoint(), lets you add a waypoint to the list, represented by an instance of the IAgVeWaypointsCollection interface. The waypoint - represented by an instance of IAgVeWaypointsElement – has the 5 properties needed for waypoint definition using the Smooth Rate method.
- The CreateShip() method creates a ship object and instantiates the IAgShip interface.
- The IAgVePropagatorGreatArc interface is acquired and used to set a start time, to select the Smooth Rate method (known in the Object Model as eDetermineTimeAccFromVel), and – via its Waypoints property - to collect the waypoints that are added by the convenience method.
- Attitude and 2D Graphics properties are set using the IAgShip interface.
- The Propagate() method of IAgVePropagatorGreatArc is used to propagate the ship's path.

For more details, it is recommended that you consult the Help pages for the interfaces involved in the above process.

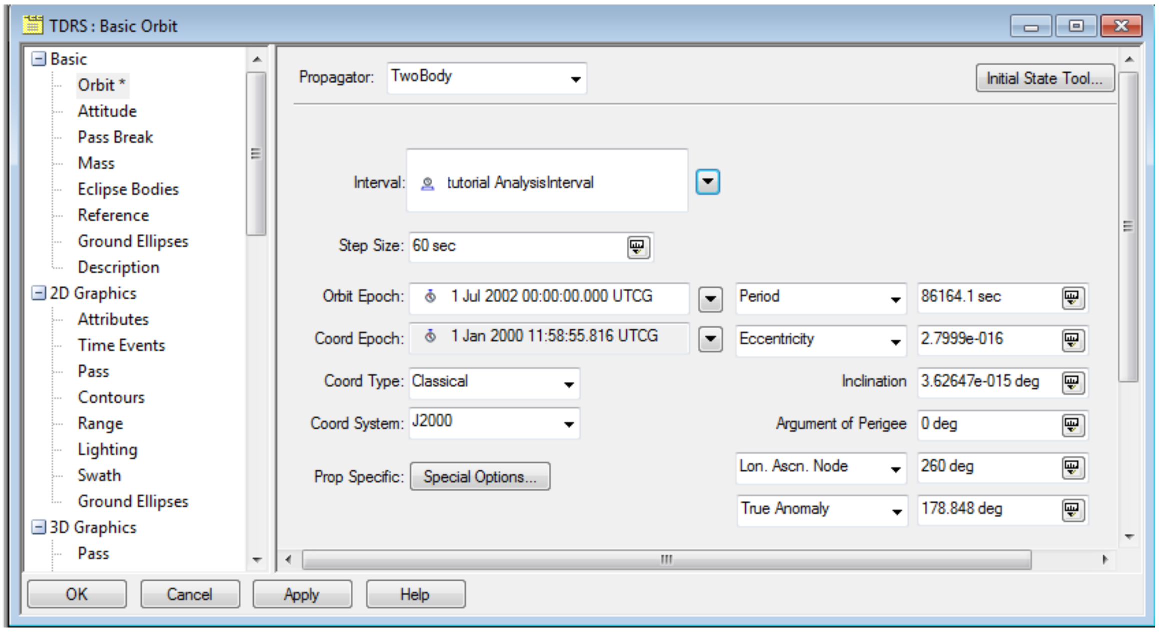
Now update the SelectedIndexChanged() method, then build and run your application, create a scenario, and select Create Ship from the combo box:



Here you can use the mouse to rotate and zoom in on the 3D view, taking advantage of functionality that became available as soon as you dragged the AGI Globe Control onto the form.

Create Some Satellites

The STK GUI generally uses the satellite's Basic Orbit page to define the satellite's initial state, including its time properties, propagator, orbital elements and coordinate system:



Propagator: TwoBody

Interval: tutorial AnalysisInterval

Step Size: 60 sec

Orbit Epoch: 1 Jul 2002 00:00:00.000 UTCG

Coord Epoch: 1 Jan 2000 11:58:55.816 UTCG

Coord Type: Classical

Coord System: J2000

Prop Specific: Special Options...

Period: 86164.1 sec

Eccentricity: 2.7999e-016

Inclination: 3.62647e-015 deg

Argument of Perigee: 0 deg

Lon. Ascn. Node: 260 deg

True Anomaly: 178.848 deg

In the above example, the satellite is propagated using the TwoBody propagator, and the orbit is defined using Classical (Keplerian) elements, with Period and Eccentricity defining the orbit's size and shape; Inclination, Argument of Perigee and Longitude of Ascending Node defining its orientation; and True Anomaly defining the location of the satellite in the orbit.

These choices are reflected in the way in which the TDRS satellite is defined using the Object Model in CreateSatellites(). After creating the satellite, selecting the TwoBody propagator, the Classical coordinate type, and the J2000 coordinate system, and setting some time properties (acquiring the necessary interfaces, of course, at each stage) the CreateSatellites() method turns to the task of configuring the 3 categories of orbital elements: size/shape, orientation and location.

Here is how CreateSatellites() defines size and shape:

- Using the IAgOrbitStateClassical interface, it assigns to the SizeShape property an enum value representing the option of using Period and Eccentricity to define orbit size and shape.
- The SizeShape property of that same interface is used to initialize the IAgClassicalSizeShapePeriod interface.
- Using the latter interface, values are assigned for Eccentricity and Period.

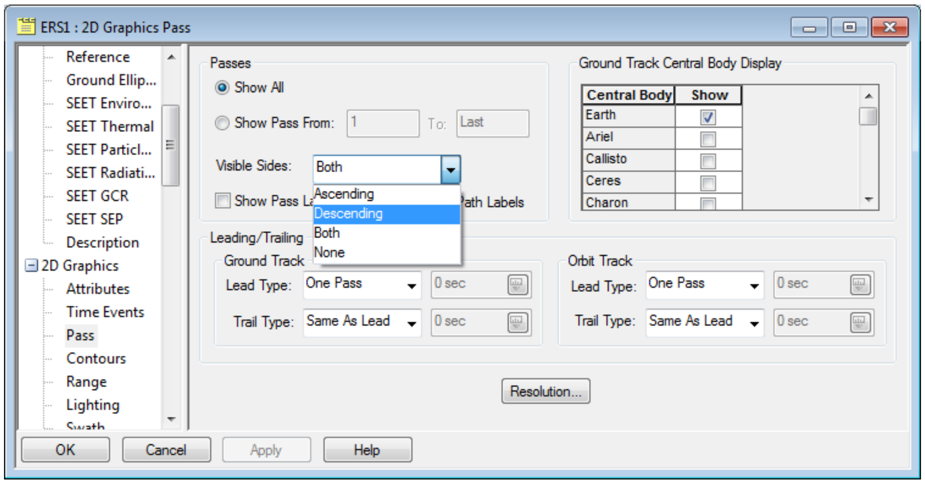
Orientation and location elements are handled similarly.

The orbital elements are then assigned back to the propagator object, and the orbit is propagated.

The TDRS_3 satellite is loaded from the Satellite Database, using code similar to what you saw above for the Facility Database, including the ExecuteCommand() method and the helper function GetSTKDataDir(). An SGP4 propagator is assigned to the satellite and used to set the time period.

The J4 Perturbation propagator is used to propagate the ERS1 and Shuttle satellites. As with the TDRS satellite, the orbits of these satellites are defined using Classical orbital elements, but different choices of elements are used for defining size/shape and orientation.

Commented code for the ERS1 satellite implements an option that the STK GUI makes available via the satellite's 2D Graphics Pass page:



Passes: Show All [✓]; Show Pass From: 1 To: Last

Visible Sides: Both

Ground Track — Lead Type: One Pass, 0 sec; Trail Type: Same As Lead, 0 sec

Orbit Track — Lead Type: One Pass, 0 sec; Trail Type: Same As Lead, 0 sec

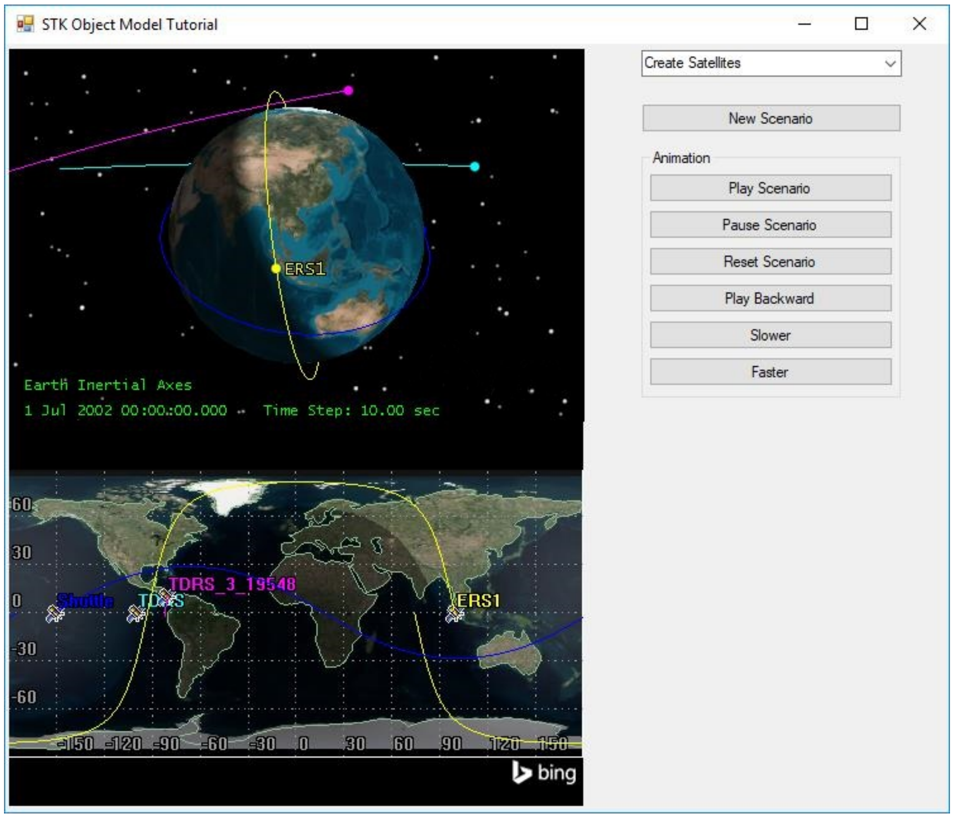
Central Body	Show
Earth	✓
Ariel	
Callisto	
Ceres	
Charon	

Among other things, that page lets you opt to display only the Ascending (south-to-north), or Descending (north-to-south) portion of each orbit pass (or both or neither). The commented code line selects the Descending option.

> **NOTE**: If you uncomment this code line to try it out, be sure to re-comment or remove it, since full path display of the ERS1 satellite is needed for some later activities.

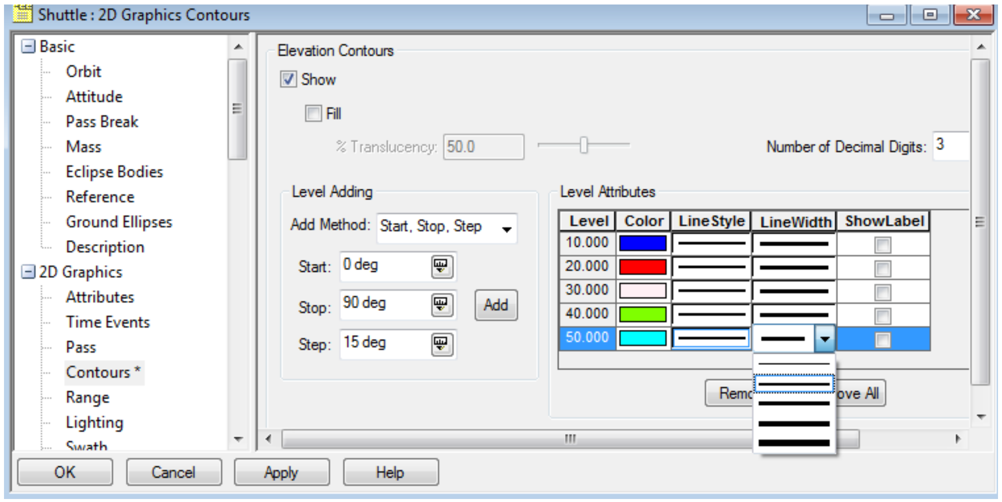
Variable declarations are needed at the Form1 level for the ERS1 and Shuttle satellites, since they are used in other methods of that class.

Update the SelectedIndexChanged() method, build and run your application, create a scenario and select Create Satellites from the combo box:



Another method in this section, ModifyShuttleDisplay(), lets us tweak and enhance the 2D Graphics properties of the Shuttle. After using the IAgVeGfxAttributesOrbit interface to change the satellite's marker and line style, the method draws elevation contours around it.

In the STK GUI, elevation contours are added and configured on the 2D Graphics Contours page:



Elevation Contours

Show [✓]

Fill [ ]

% Translucency: 50.0

Number of Decimal Digits: 3

Level Adding

Add Method: Start, Stop, Step

Start: 0 deg

Stop: 90 deg

Step: 15 deg

Level Attributes

Level	Color	LineStyle	LineWidth	ShowLabel
10.000	🟦			☐
20.000	🟥			☐
30.000	⬜			☐
40.000	🟩			☐
50.000	🟦			☐

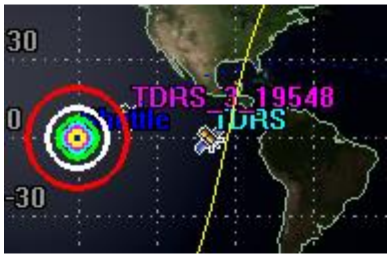
In the above example, you have selected Start, Stop, Step as the Add Method and set the corresponding values to define 6 contour levels spaced 10 degrees apart. Use the Remove

All button to get rid of any existing contour graphics, and then click Add to populate the Level Attributes list with our newly defined contours. After increasing the LineWidth and disabling the ShowLabel option of each contour, select the Show checkbox to display the contours.

Here is how to perform all the above steps in the Object Model:

- Acquire the IAgVeGfxElevContours interface, which exposes general settings for elevation contours.
- Acquire the IAgVeGfxElevationsCollection interface, which makes several methods available for building and managing the set of contour levels.
- Use that interface's RemoveAll() method to get rid of any existing contours.
- Use its AddLevelRange() method to add 6 contours using the Start, Stop, Step method.
- Use its Count property to set up a loop in which you use the IAgVeGfxElevationsElement interface to disable label display and increase line width for each contour level.
- Use the IsVisible property of the IAgVeGfxElevContours interface to display the contours.

Update the SelectedIndexChanged() method, then build and run the application, create a scenario, and select Create Satellites and Modify Shuttle Display from the combo box:



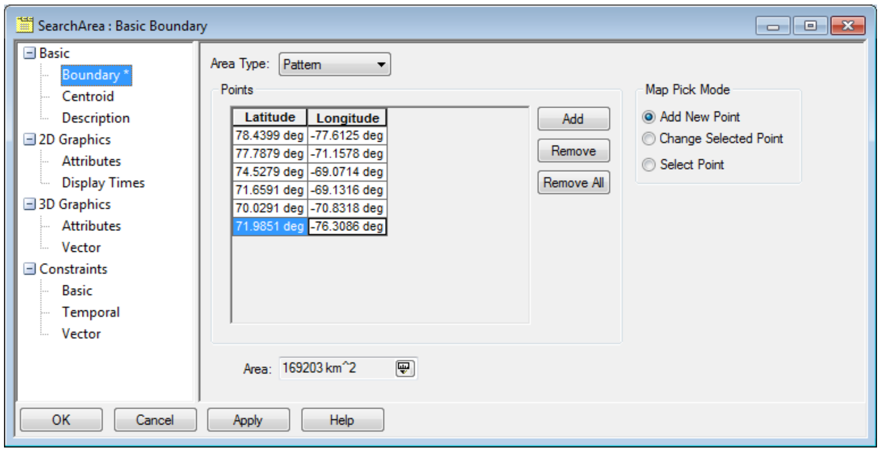
Now animate the scenario and observe the results in your 2D and 3D displays.

Computing & Constraining Access

Compute Access

In this section you will add code to your application to enable it to create an area target and compute line-of-sight access from the ERS1 satellite to the area target.

The boundary of an area target is defined in the STK GUI via the Basic Boundary page:



Area Type: Pattern

Points

Latitude	Longitude
78.4399 deg	-77.6125 deg
77.7879 deg	-71.1578 deg
74.5279 deg	-69.0714 deg
71.6591 deg	-69.1316 deg
70.0291 deg	-70.8318 deg
71.9851 deg	-76.3086 deg

Map Pick Mode: [✓ Add New Point] Change Selected Point Select Point

Area: 169203 km^2

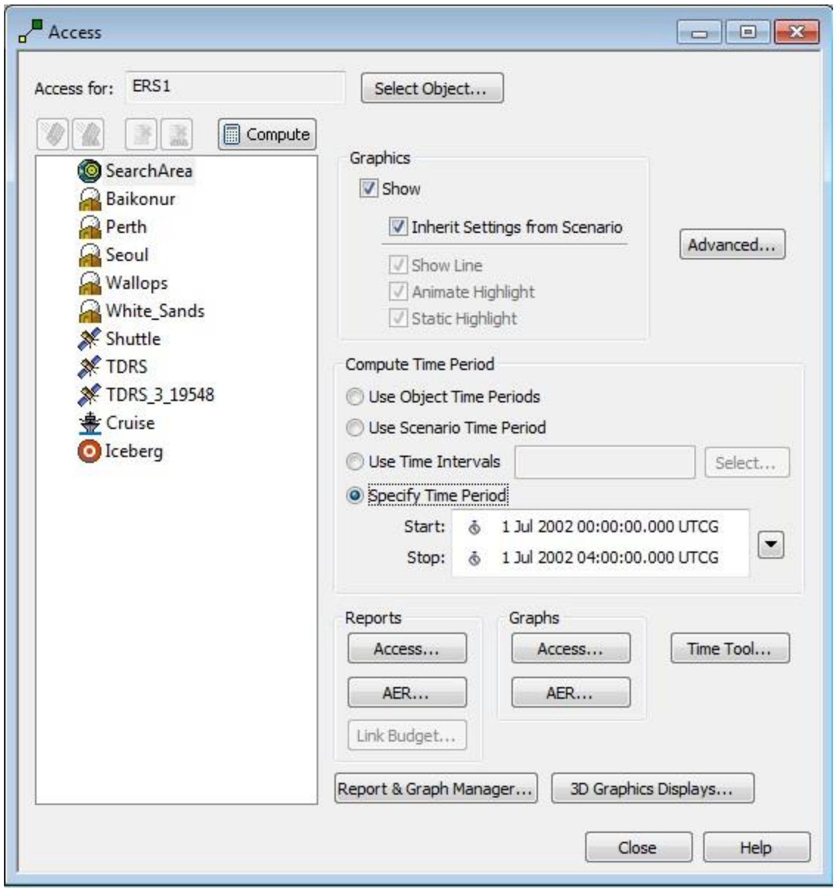
In this example, the Pattern Area Type is selected to define the boundary (the other option is Ellipse). Latitude and longitude values for boundary points are added to a list, and the boundary is created by drawing Great Arc lines between those points.

The definition of an area target using the Object Model largely parallels the GUI approach. After using the IAgATGraphics interface to set some of the 2D Graphics properties of the area target, the CreateAreaTarget() method defines the boundary as follows:

- An enum value representing the Pattern area type is assigned to the area target object.
- The IAgAreaTypePatternCollection interface is acquired and initialized.
- The Add() method of that interface is used to add boundary points.

Additional code uses a course-grained method to position the area target's centroid.

Access is computed in STK by an object-level tool that provides a mechanism for the selection of 'to' objects, as well as various setup and output options:



In the illustration, the Access tool has been launched from the ERS1 satellite – the 'from' object – and the area target has been selected as the 'to' object. Access is computed when someone clicks the Compute button or selects one of the report, graph or display options.

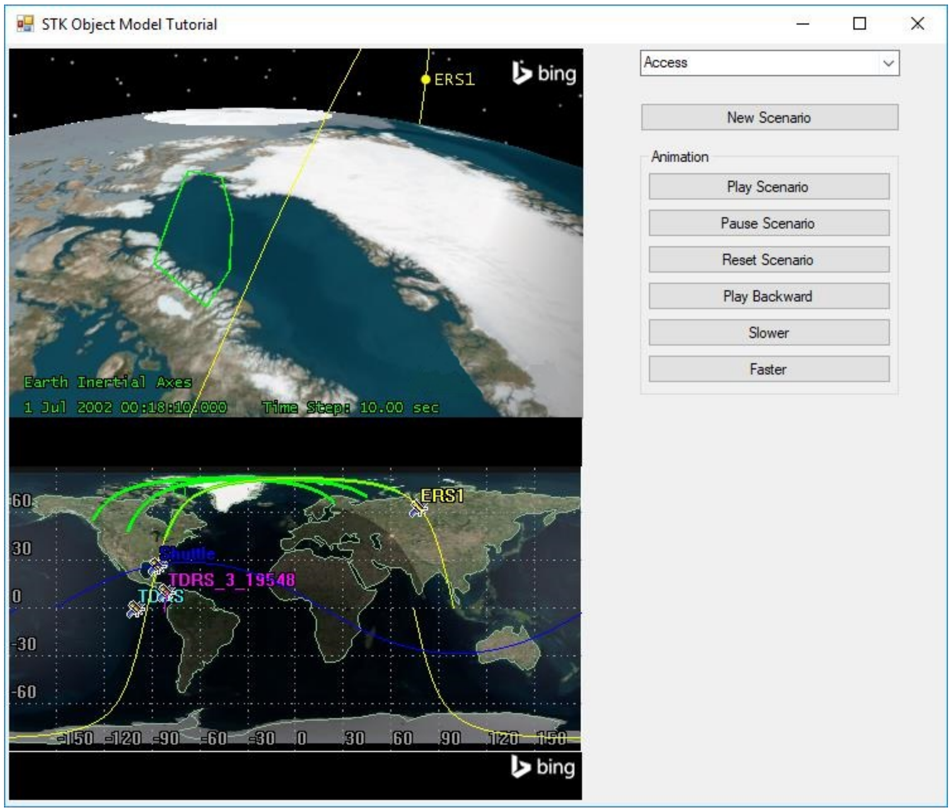
Access computation is carried out in a roughly similar fashion in the Object Model:

- The IAgStkAccess interface is instantiated and initialized with the ERS1 satellite's GetAccessToObject() method, which takes the intended 'to' object – here, the area target – as an argument.
- The AgStkAccess interface's ComputeAccess() method is called.
- The IAgDataPrvInterval interface is acquired and instantiated using the access object's DataProviders property, which returns a list of available data providers for the access.
- The latter interface's Exec() method is used to compute the data for a specified time period and assign it to an instance of the IAgDrResult interface.

The data thus returned is available for further use in the application.

The variables representing the area target and the access object are declared at the Form1 level, so that they are available to other methods in that class.

Update the SelectedIndexChanged() method, build and run the application, create a scenario, and select Create Satellites, Create Area Target and Access (in that order) from the combo box. You will notice a series of thick lines overlaid on the ERS1 ground track, representing intervals of access from ERS1 to the area target. Animate the scenario to observe further graphic effects:



To remove access computations and graphics, the RemoveAccess() method uses the IAgStkAccess interface's method of the same name. To test it, update the SelectedIndexChanged() method, set up the access computation described above, and then select RemoveAccess from the combo box.

Create Some Sensors

You will now get some practice adding and configuring sensors with the Object Model. In STK, sensors are children (sub-objects) of other objects, such as satellites. This relationship is captured in the Object Model by the Children property of an STK object. For comparison, first take a look at the code that creates a scenario-level object, such as a satellite. The code line that creates the ERS1 satellite contains the following sequence:

```
root.CurrentScenario.Children.New(AgESTKObjectType.eSatellite,
"ERS1")
```

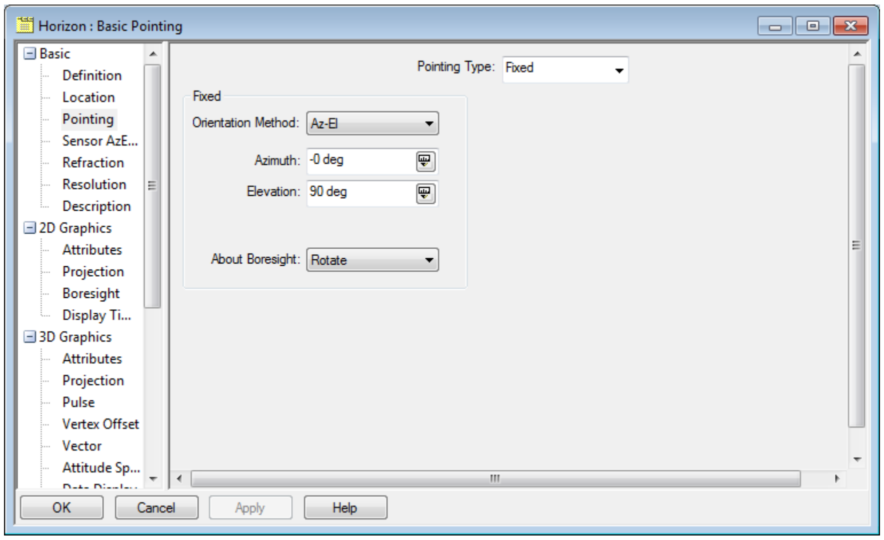
This indicates that the new satellite is being created as a child of the current scenario. Now compare the corresponding sequence in the code line that creates the Horizon sensor:

```
root.CurrentScenario.Children["ERS1"].Children.
New(AgESTKObjectType.eSensor, "Horizon")
```

Here the Children property is used twice: once to indicate that the ERS1 satellite is a child of the scenario, and another time to indicate that the new sensor is a child of ERS1.

Since the Definition properties of the Horizon and Downlink sensors are handled in a fairly straightforward way by the STK GUI and by the Object Model, you can focus here on the Pointing properties.

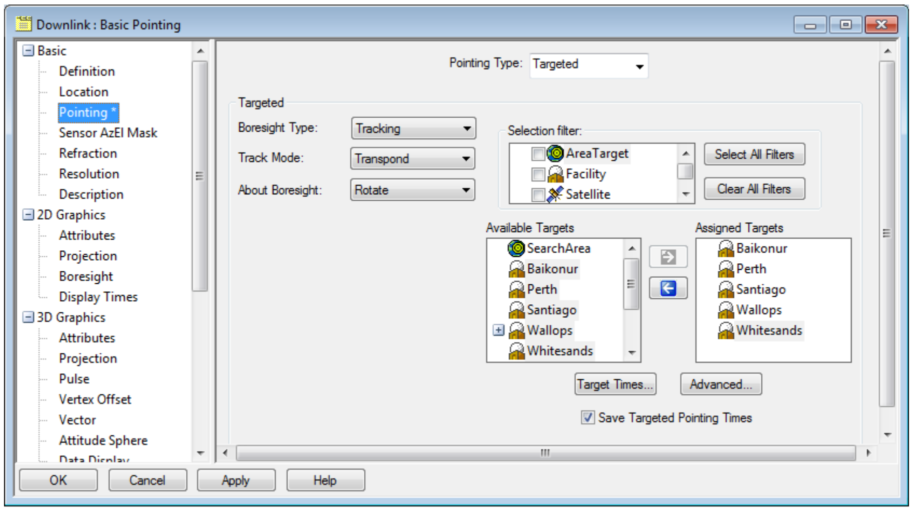
For the Horizon sensor, the Pointing Type is Fixed and the Orientation Method is azimuth-elevation:



Pointing Type: Fixed
Fixed
Orientation Method: Az-El
Azimuth: -0 deg
Elevation: 90 deg
About Boresight: Rotate

The code that implements these choices in CreateSensors(), except for the names of the interfaces, should look familiar by now. The important thing to note is that after you use the IAgOrientationAzEl interface to set the Elevation and AboutBoresight properties, it all must be assigned back into the Orientation property of the interface representing the Fixed pointing type (using the IAgOrientation interface's Assign() method).

The Downlink sensor uses the Targeted Pointing Type, with all the facilities in the scenario assigned as targets:


Downlink : Basic Pointing

Pointing Type: Targeted

Targeted

Boresight Type: Tracking

Track Mode: Transpond

About Boresight: Rotate

Selection filter:

- AreaTarget
- Facility
- Satellite

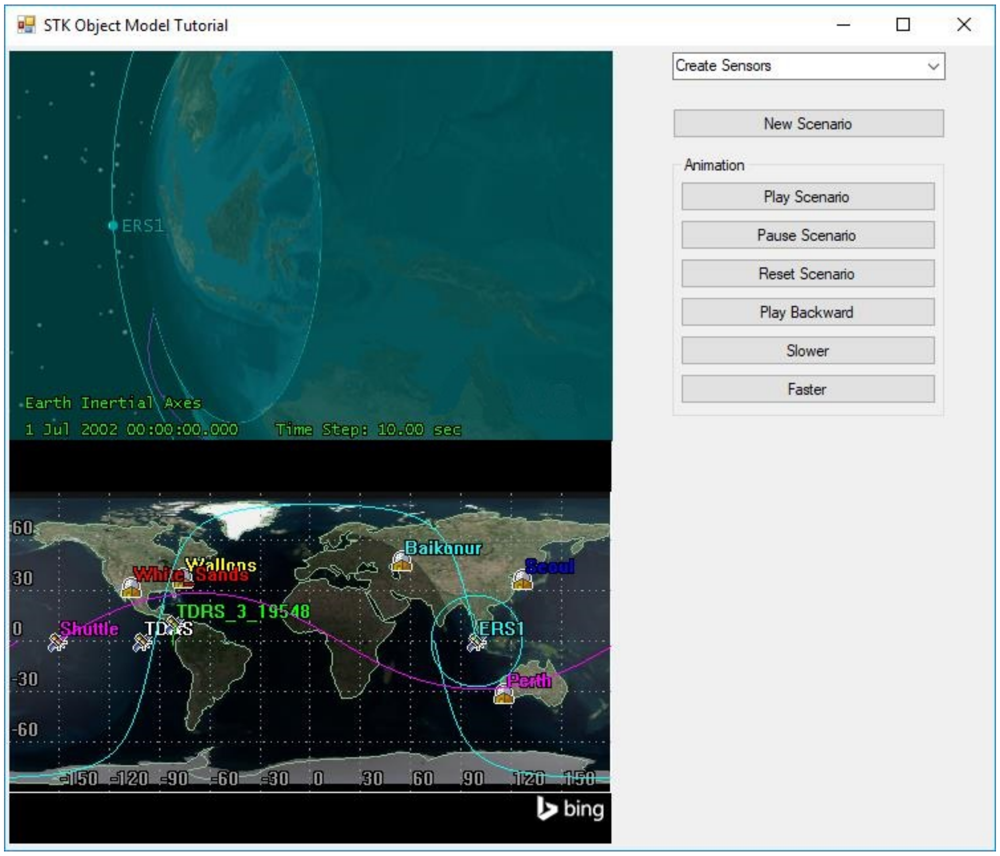
Available Targets	Assigned Targets
SearchArea	Baikonur
Baikonur	Perth
Perth	Santiago
Santiago	Wallops
Wallops	Whitesands
Whitesands	

[✓] Save Targeted Pointing Times

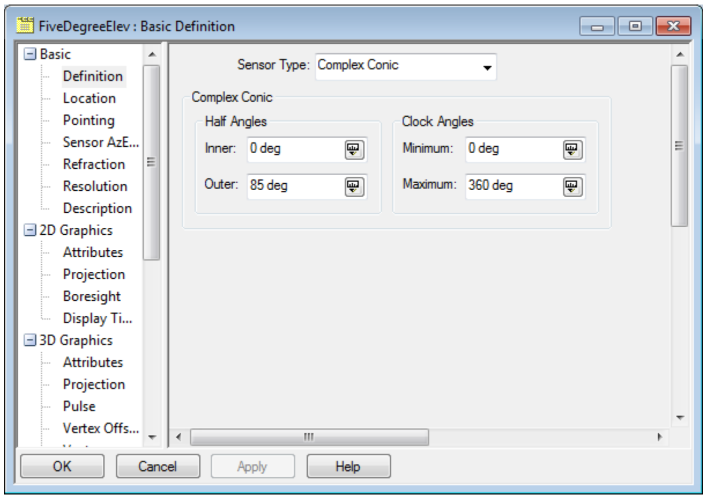
The CreateSensors() method uses the IAgSnTargetCollection interface to build up the collection of targets. As illustrated in the code, a target can be added as a string representing its name, using the Add() method, or it can be added as an object, using the AddObject() method. The line that adds the Wallops facility makes use of the Path property of the IAgStkObject interface, which returns a string.

The variable representing the Downlink sensor needs to be declared at the Form1 level, so that it can be used by other methods in that class.

Now update the SelectedIndexChanged() method, then build an run the application, create a scenario, and select Create Facilities, Create Satellites and Create Sensors from the combo box. Animate the scenario and observe the sensor pattern as the Downlink sensor on ERS1 acquires its targets:



The FiveDegElev sensor, attached to the Wallops facility, is of the Complex Conic Sensor Type. The outer cone half-angle is set to 85 deg to limit the sensor's visibility to objects at an elevation of 5 degrees or higher:



Sensor Type: Complex Conic

Complex Conic

Half Angles

Inner: 0 deg

Outer: 85 deg

Clock Angles

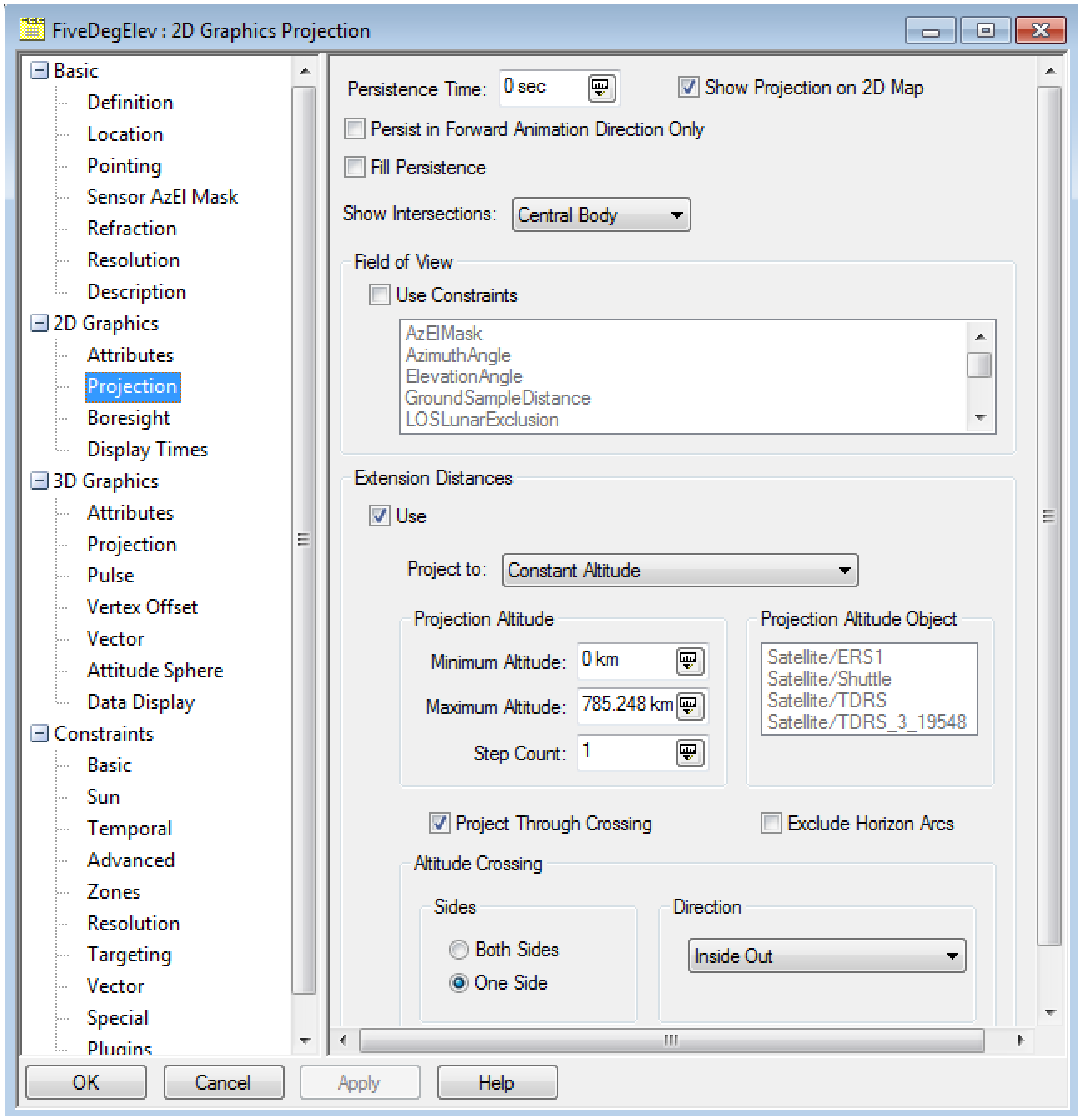
Minimum: 0 deg

Maximum: 360 deg

There is nothing remarkable about the code in LimitSensorVisibility() that implements the above choices. Also, the code that selects a Fixed pointing type and applies the azimuth-elevation orientation method is the same as what you saw for the Horizon sensor in CreateSensors().

NOTE: The effect of setting a 90 deg Elevation here, where the sensor is attached to a facility, is to point the sensor upward, while it had the opposite effect for the Horizon sensor, which is attached to a satellite.

The 2D Graphics Projection properties of the FiveDegElev sensor have been modified to impose a 785.248 km limit on the sensor's projection in the Map window:


FiveDegElev : 2D Graphics Projection

Persistence Time: 0 sec
[✓] Show Projection on 2D Map
[ ] Persist in Forward Animation Direction Only
[ ] Fill Persistence
Show Intersections: Central Body

Field of View
[ ] Use Constraints
AzElMask
AzimuthAngle
ElevationAngle
GroundSampleDistance
LOSLunarExclusion

Extension Distances
[✓] Use
Project to: Constant Altitude

Projection Altitude
Minimum Altitude: 0 km
Maximum Altitude: 785.248 km
Step Count: 1

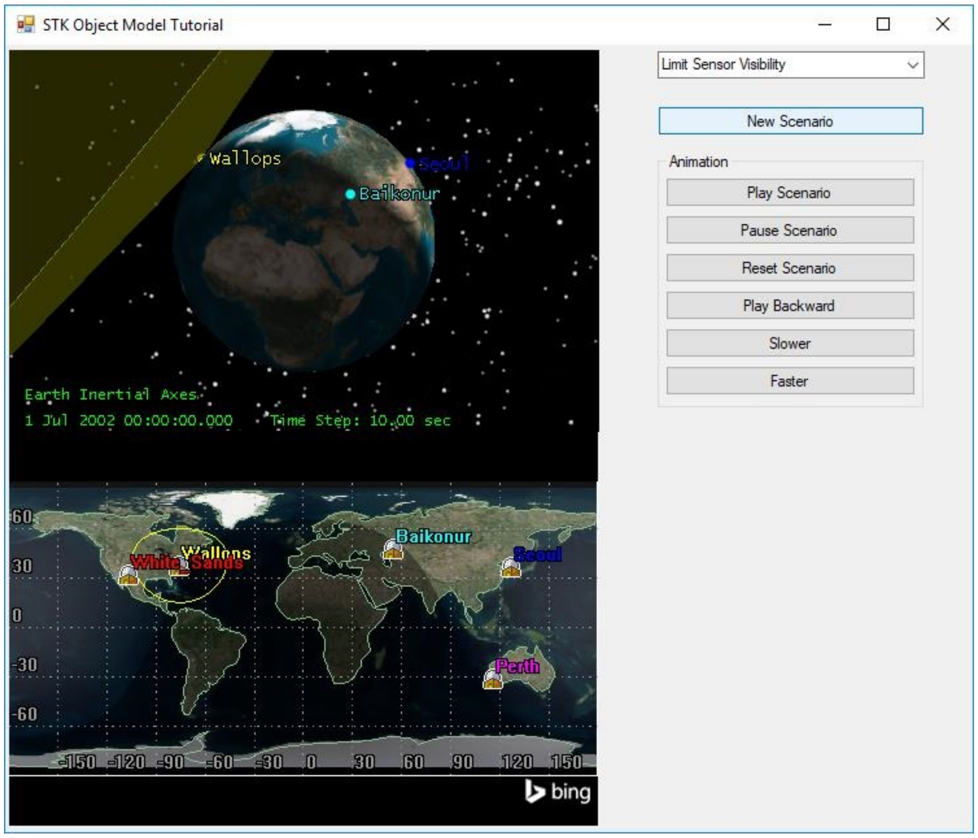
Projection Altitude Object
Satellite/ERS1
Satellite/Shuttle
Satellite/TDRS
Satellite/TDRS_3_19548

[✓] Project Through Crossing
[ ] Exclude Horizon Arcs

Altitude Crossing
Sides: Both Sides [✓ One Side]
Direction: Inside Out

The LimitSensorVisibility method uses the IAgSnProjDisplayDistance interface to implement the above settings, including the selection of Constant Altitude as the 'Project to' option and the settings for minimum and maximum altitude and step count.

Update the SelectedIndexChanged() method, then build and run the application, create a scenario and select Create Facilities and Limit Sensor Visibility from the combo box:

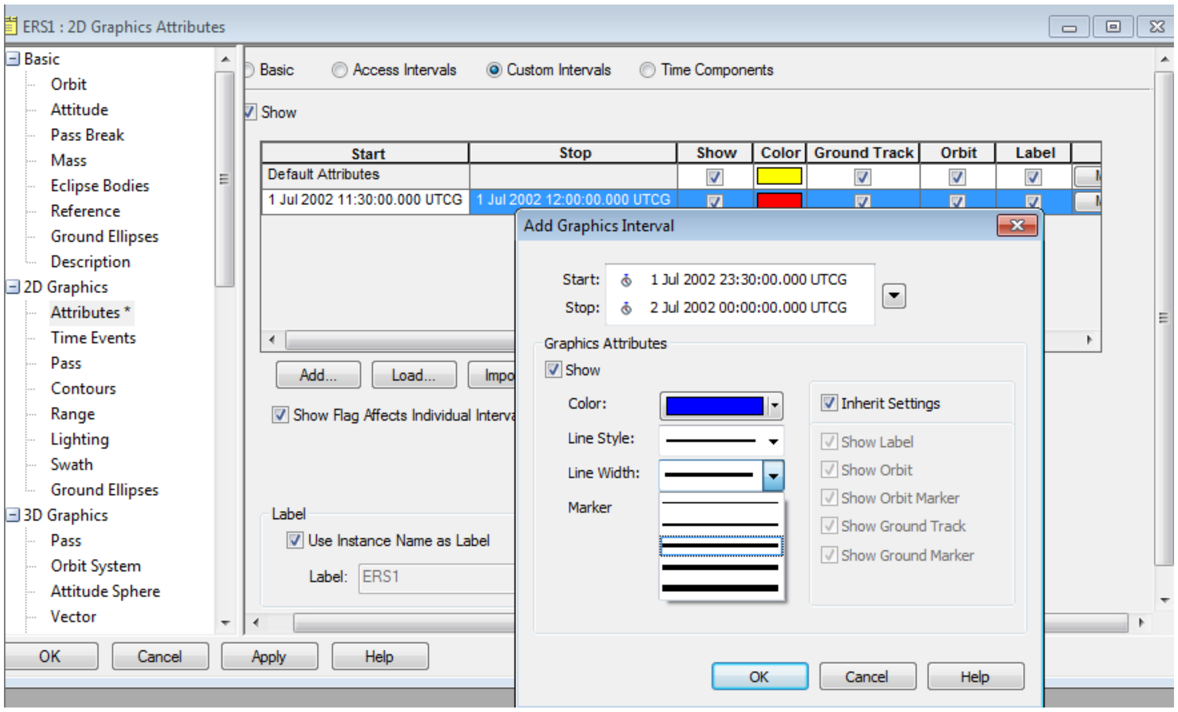


You can further restrict the sensor's field of view by decreasing the outer cone half angle. For example, reduce it to 75 deg to impose a 15 degree elevation requirement.

Control Display Intervals

There are a number of ways in which you can control the times and manner in which an STK object displays in the 2D and 3D windows. Two such options are explored here.

One option is to set up custom intervals and alter the display of an object during those intervals. For example, on the 2D Graphics Attributes page for the ERS1 satellite, you can select the Custom Intervals option and alter the display of the satellite and its ground track during selected time intervals:



Basic ( ) Access Intervals [● Custom Intervals] ( ) Time Components

☑ Show

Start	Stop	Show	Color	Ground Track	Orbit	Label
Default Attributes		✓	🟨	✓	✓	✓
1 Jul 2002 11:30:00.000 UTCG	1 Jul 2002 12:00:00.000 UTCG	✓	🟥	✓	✓	✓

☑ Show Flag Affects Individual Interva

Label

☑ Use Instance Name as Label

Label: ERS1

Add Graphics Interval

Start: 1 Jul 2002 23:30:00.000 UTCG

Stop: 2 Jul 2002 00:00:00.000 UTCG

Graphics Attributes

☑ Show

Color: 🟦

Line Style:

Line Width:

Marker

☑ Inherit Settings

☑ Show Label

☑ Show Orbit

☑ Show Orbit Marker

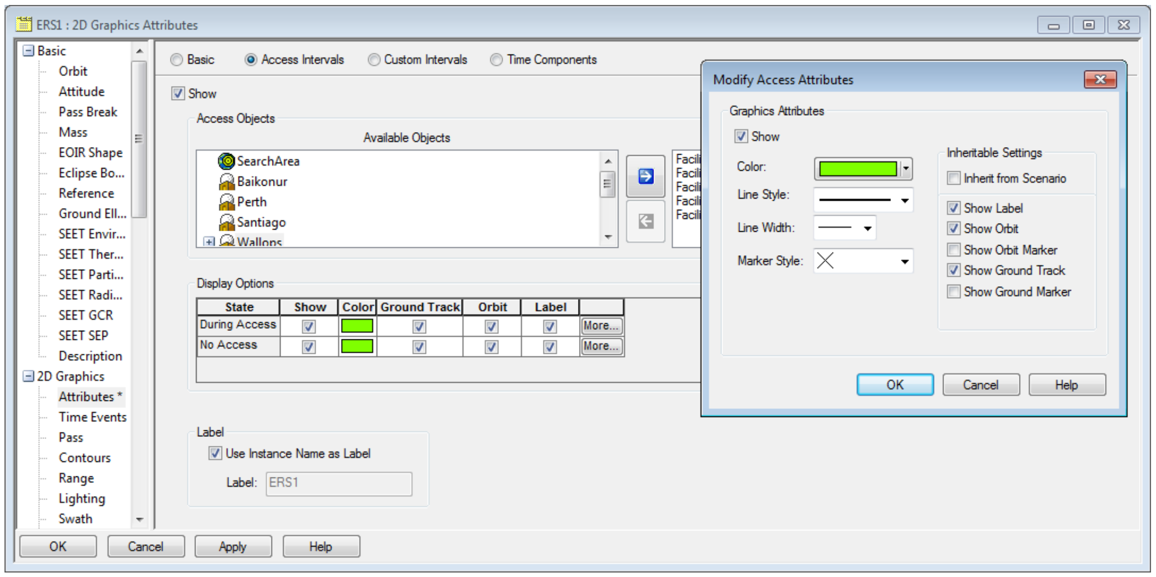
☑ Show Ground Track

☑ Show Ground Marker

To implement the above process in CustomDisplayIntervals(), after selecting the custom intervals approach:

- You acquire the IAgVeGfxAttributesCustom and IAgVeGfxInterval interfaces.
- The Intervals property of the former interface makes available the IAgVeGfxIntervalsCollection interface, whose Add() method is used in specifying the time period of each custom interval and in initializing the IAgVeGfxInterval interface.
- The GfxAttributes property of the latter interface makes available the IAgVeGfxAttributesBasic interface, which exposes such properties as Color and Line, among others. The Line property returns an interface that lets us set the width of the ground track.

Another option for controlling the display of objects in the 2D and 3D windows is to limit their display to defined access periods. For example, you can limit the display of the orbit and ground markers for the ERS1 satellite to periods when it has access to one or more facilities:


ERS1 : 2D Graphics Attributes

- Basic
  - Orbit
  - Attitude
  - Pass Break
  - Mass
  - EOIR Shape
  - Eclipse Bo...
  - Reference
  - Ground Ell...
  - SEET Envir...
  - SEET Ther...
  - SEET Parti...
  - SEET Radi...
  - SEET GCR
  - SEET SEP
  - Description
- 2D Graphics
  - Attributes *
  - Time Events
  - Pass
  - Contours
  - Range
  - Lighting
  - Swath

Basic [✓ Access Intervals] Custom Intervals Time Components

☑ Show

Access Objects

Available Objects: SearchArea, Baikonur, Perth, Santiago, Wallops

Display Options

State	Show	Color	Ground Track	Orbit	Label	
During Access	☑	🟩	☑	☑	☑	More...
No Access	☑	🟩	☑	☑	☑	More...

Label

☑ Use Instance Name as Label

Label: ERS1

Modify Access Attributes

Graphics Attributes

☑ Show

Color: 🟩

Line Style:

Line Width:

Marker Style:

Inheritable Settings

☐ Inherit from Scenario

☑ Show Label

☑ Show Orbit

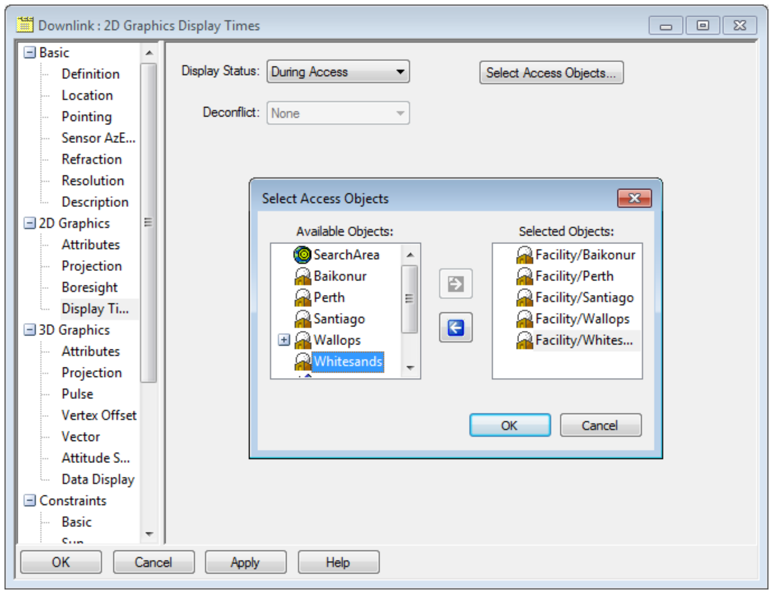
☐ Show Orbit Marker

☑ Show Ground Track

☐ Show Ground Marker

The settings in the Modify Access Attributes dialog disable display of the orbit and ground markers during periods of no access.

Similarly, you can use the 2D Graphics Display Times page for the Downlink sensor to confine its display to periods of facility access:


Downlink : 2D Graphics Display Times

- Basic
  - Definition
  - Location
  - Pointing
  - Sensor AzE...
  - Refraction
  - Resolution
  - Description
- 2D Graphics
  - Attributes
  - Projection
  - Boresight
  - Display Ti...
- 3D Graphics
  - Attributes
  - Projection
  - Pulse
  - Vertex Offset
  - Vector
  - Attitude S...
  - Data Display
- Constraints
  - Basic

Display Status: During Access

Deconflict: None

Select Access Objects

Available Objects: SearchArea, Baikonur, Perth, Santiago, Wallops, Whitesands

Selected Objects: Facility/Baikonur, Facility/Perth, Facility/Santiago, Facility/Wallops, Facility/Whites...

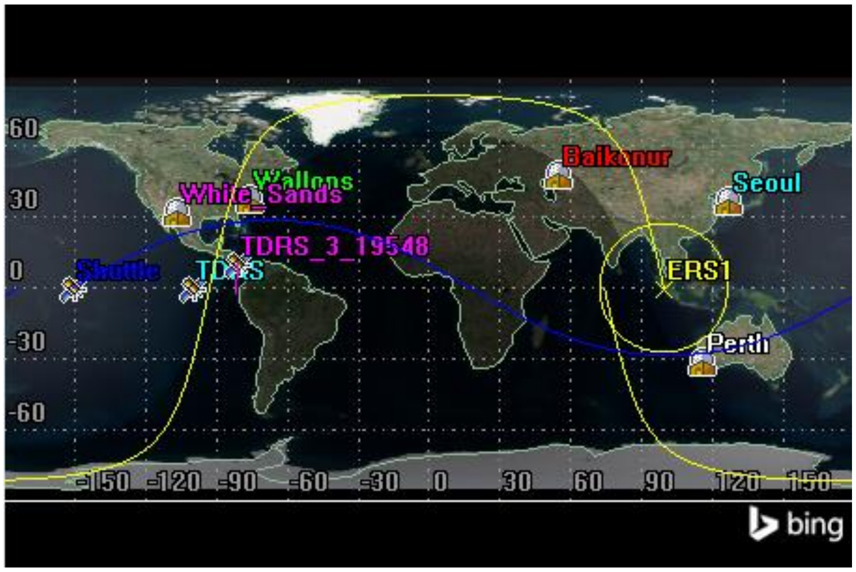
Both of these processes are implemented in the AccessDisplayIntervals() method, each in ways that closely track the STK GUI approach.

For the satellite, you begin as you did in CustomDisplayIntervals(), by selecting the 2D graphics attributes type, except that here you select access intervals instead of custom intervals. Then you acquire the IAgVeGfxAttributesAccess interface and use its AccessObjects property to build up a collection of access objects (facilities). Use the NoAccess property of that interface to initialize the IAgVeGfxAttributesOrbit interface, which is then used to disable the display of certain graphics attributes of the satellite, including orbit and ground track display, during periods of no access.

For the sensor, you acquire the IAgDisplayTm interface and use its SetDisplayStatusType() method to opt for display during periods of access. The DisplayTimesData property of this interface is used to initialize the IAgDuringAccess interface, which, in turn, is used to initialize the IAgObjectLinkCollection interface. That interface provides the machinery to build up the collection of access objects.

The EnlargeTimePeriod() method extends the time period of the ERS1 satellite, giving us 24 hours of data to work with.

Update the SelectedIndexChanged() method, build and run your application, and try out each of these display controls in turn. To see the effect of CustomDisplayIntervals(), create a scenario and select Create Satellites, Enlarge Time Period and Custom Display Intervals from the combo box. In the 2D window, the width and color of the ground track for ERS1 are altered during the half-hour intervals preceding noon and midnight:

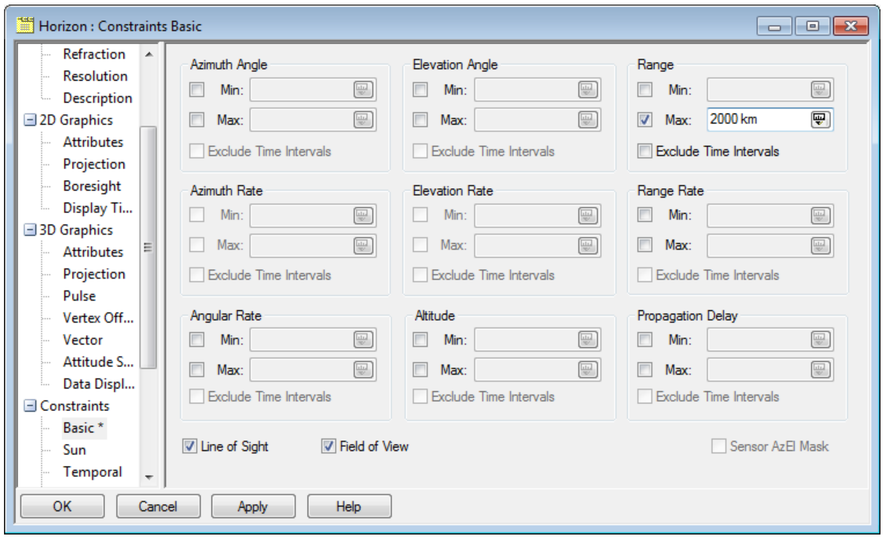


To see the effect of AccessDisplayIntervals(), create a new scenario, and then select Create Facilities, Create Satellites, Create Sensors and Access Display Intervals. Animate the scenario and observe how the ERS1 satellite and its sensors disappear from view in the 2D and 3D windows except when there is access to a facility.

Apply Access Constraints

STK and the Object Model supply a rich inventory of constraints that let you model the effect of environmental factors, equipment limitations, various kinds of obscuration, etc. on access between objects. STK handles access constraints as properties of objects participating in the access.

For example, you can use the Horizon sensor's Constraints Basic page to constrain access from that sensor to occur only when the range to the 'to' object does not exceed 2000 km:



Azimuth Angle: Min: ☐ Max: ☐ Exclude Time Intervals ☐
Elevation Angle: Min: ☐ Max: ☐ Exclude Time Intervals ☐
Range: Min: ☐ Max: ✓ 2000 km Exclude Time Intervals ☐
Azimuth Rate: Min: ☐ Max: ☐ Exclude Time Intervals ☐
Elevation Rate: Min: ☐ Max: ☐ Exclude Time Intervals ☐
Range Rate: Min: ☐ Max: ☐ Exclude Time Intervals ☐
Angular Rate: Min: ☐ Max: ☐ Exclude Time Intervals ☐
Altitude: Min: ☐ Max: ☐ Exclude Time Intervals ☐
Propagation Delay: Min: ☐ Max: ☐ Exclude Time Intervals ☐
✓ Line of Sight ✓ Field of View ☐ Sensor AzEl Mask

The AccessConstraintsRange() method uses properties exposed by the IAgAccessCnstrMinMax interface to enable a maximum range constraint and set its value. The interface is initialized by calling the AddConstraint method of the IAgAccessConstraintCollection interface (made available via the sensor's AccessConstraints property), which returns information on the selected constraint type.

The AccessConstraintsSunElevationAngle() method works similarly.

To help test these constraints, the FacilityAccess() method computes access from the Horizon sensor to Baikonur.

Update the SelectedIndexChanged() method, then build and run your application, and create a scenario. To test the range constraint, select Create Facilities, Create Satellites, Create Sensors, Facility Access and Range Constraint from the combo box. To test the sun elevation angle constraint, create a new scenario and select Create Facilities, Create Satellites, Create Sensors, Enlarge Time Period, Facility Access and SunElevationAngle Constraint from the combo box.

Commented code in both constraint methods invite you to experiment with other values for the respective constraint.