# Overview

tbs

# Interface Description

This section provides information needed to incorporate the Orbit and Coverage (O&C) code into a larger system such as TAT-C. It provides the interface to key routines used to access O&C capabilities. Including precise definitions for each argument.

It also provides high level descriptions of the O&C subsystem’s behavior. This is intended to provide a broad outline, the details are provided in the source code itself and in the System Structure section of this document.

## Calling Key Routines

### Propagate

The Propagate function is defined in the Propagator class, and it has the following signature:

virtual Rvector6 Propagate(const AbsoluteDate &toDate);

Argument **&toDate** – this is an AbsoluteDate object; class AbsoluteDate provides the ability to represent dates as either Julian or Gregorian dates. Generally Gregorian dates are used for initialization and Julian dates used for computations. The propagator will propagate the spacecraft’s state to that time.

Return value **Rvector6** – this is a 6 element vector of real numbers representing the spacecraft state. The first three elements of this vector represent the spacecraft’s position in Earth-centered inertial coordinates, the next 3 represent the velocity in the same coordinate frames.

### AccumulateCoverageData

The AccumulateCoverageData function is defined in the CoverageChecker class, and it has two overloaded versions. The one with no arguments is used when propagating and checking for sensor visibility at the same time. The one with time as an argument is used when stepping the event locator multiple times within each orbit propagation step.

virtual IntegerArray AccumulateCoverageData();

virtual IntegerArray AccumulateCoverageData(Real atTime);

details here

## High Level Behavior

This section gives a high level view of how pieces of the general Initialize-Propagate-Postprocess use case work. They are presented as descriptive text and snippets of actual code that show the key concepts of how this subsystem is intended to be used.

### Initialization

In the system test driver, the classes initialize in the following order. Dependencies on predecessor classes are listed for each class.

* LagrangeInterpolator – none
* Earth – none
* AbsoluteDate – none
* OrbitState – none
* Conical Sensor – none
* NadirPointingAttitude – none
* Spacecraft (Attitude, AbsoluteDate, OrbitState ,LaGrangeInterpolator)
* Propagator (Spacecraft)
* TBS for coverage

Note that NadirPointingAttitude is a subclass of Attitude, and that ConicalSensor is a subclass of Sensor.

Note also that is possible to create multiple sensors on one spacecraft, and that CoverageChecker accesses sensor(s) and their field of view via a Spacecraft object containing said sensor(s).

### Propagation & Coverage Without Interpolation

Blah

prop->Propagate(\*date);

while (date->GetJulianDate() < ((Real)startDate + 1.0)) // 5.0))

{

// Compute points in view at time zero!

loopPoints = covChecker->AccumulateCoverageData();

// Propagate

date->Advance(stepSize);

prop->Propagate(\*date);

// Compute lat., lon., and height of s/c w/r/t the ellipsoid

Real jDate = sat1->GetJulianDate();

Rvector6 cartState = sat1->GetCartesianState();

Rvector3 inertialPosVec(cartState(0), cartState(1),cartState(2));

Rvector3 latLonHeight = earth->InertialToBodyFixed(inertialPosVec,

jDate, "Ellipsoid");

}

Blah

### Propagation & Coverage With Interpolation

Blah

prop->Propagate(\*date);

while (date->GetJulianDate() < ((Real) startDate + 1.0)) // 5.0))

{

date->Advance(stepSize);

prop->Propagate(\*date);

propTime = date->GetJulianDate();

// Interpolate when and if needed

if (sat1->TimeToInterpolate(propTime, midRange))

{

while (interpTime < (propTime - midRange))

{

loopPoints = covChecker->

AccumulateCoverageData(interpTime);

interpTime += interpolationStepSize/

GmatTimeConstants::SECS\_PER\_DAY;

}

}

}

Blah

### Computation of Statistics

TBS

# System Structure

## Class Dependencies

The diagram below shows the key dependencies between components. The light shading shows the components that implement the main functions of modeling the spacecraft, propagating the spacecraft state, and identifying when points are within a sensor’s field of viewk. The dark shading indicates the models used by these major functions. Utilities such as vector and matrix arithmetic are not shown on this diagram.

Spacecraft

Coverage

Checker

Sensor

Attitude

Conical

Custom

Rectangular

Orbit

Interpolator

Propagator

POI Data

1,..n

1,..n

1,..n

Point Group

Orbit State

Absolute Date

Note that the class Sensor has three subclasses providing 3 different models of sensor field of views. The conical and rectangular fields of view are self-explanatory, a custom field of view allows the FOV perimeter to be defined by an arbitrary set of points.

The following sections provide the detailed responsibilities for each class.

## Class Responsibilities for Propagation

Xyzzy

|  |  |
| --- | --- |
| Class | Responsibility |
| Propagator |  |
| Spacecraft |  |
| Sensor | The Sensor class defines a field of view, maintains knowledge of its orientation relative to the spacecraft body, and a function which determines if the a point is within the sensor field of view.  There are three subclasses of Sensor. |
| Attitude | O-C uses the subclass NadirPointingAttitude, which orients the spacecraft to the center of the Earth. The main responsibility of this class is to store the rotation from an inertial frame to the nadir pointing reference frame, as computed from spacecraft position and velocity. |
| Orbit State | Orbit State contains the spacecraft position and velocity, which can be set and retrieved as either Keplerian or Cartesian elements. |
| Orbit Interpolator | tbs |
| Absolute Date | This class maintains a representation of date and time. The time can be set or retrieved as either a Gregorian date (year, month, day, hours, minutes and seconds) or a Julian date (days from a standard reference point), and it allows the date and time to be advanced by a number of seconds. This number may be negative to indicate movement backwards in time, however this is not being done for TAT-C. |

Plugh

## Class Responsibilities for Coverage

Xyzzy

|  |  |
| --- | --- |
| Class | Responsibility |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Plugh

# Doxygen Output