OpSim queries within the Catalogs Simulation Framework

Access to modern OpSim runs

CatSim catalogs rely on instantiations of the ObservationMetaData class to characterize telescope pointings and observing conditions. Often, users will want to base those conditions on simulated observations produced by OpSim. OpSim runs can be directly turned into ObservationMetaData instantiations using the ObservationMetaDataGenerator class defined in

```
sims_catUtils/python/lsst/sims/catUtils/utils/ObservationMetaDataGenerator.py
```

The ObservationMetaDataGenerator class connects to an OpSim database and then, through the method getObservationMetaData(), allows users to request ObservationMetaData instantiations that fit certain criteria (i.e. RA, Dec, airmass, seeing, etc. within a certain range).

Examples of the use of this class can be found in the CatSimTutorial_SimulationsAHM_1503.ipynb iPython notebook in the UWSST LSST-Tutorials github repository.

The ObservationMetaDataGenerator can be directly accessed using

```
from lsst.sims.catUtils.utils import ObservationMetaDataGenerator

help(ObservationMetaDataGenerator)
```

Note: users wishing to instantiate their own ObservationMetaData objects by hand can find the class defined in

```
sims_catalogs_generation/python/lsst/sims/catalogs/generation/db/ObservationMetaData.py
```

which can be accessed by

```
from lsst.sims.catalogs.generation.db import ObservationMetaData

help(ObservationMetaData)
```

Access to Deprecated OpSim runs

Version 3.61 of the OpSim output is stored in the LSST database and can be queried using the class OpSim3_61DBObject whose source code can be found in /sims_catUtils/python/lsst/sims/catUtils/baseCatalogModels/OpSim3_61DBObject.py. Pointings are stored based on their RA, Dec, MJD, and the OpSim-specific obshistid. ObservationMetaData (suitable for input into InstanceCatalog) can be generated from an obshistid using the class method OpSim3_61DBObject.getObservationMetaData(). We will begin by discussing how to search OpSim for desired pointings (and their obshistid values). We will then show how to convert the pointings into ObservationMetaData instantiations.

Searching OpSim

This example will query and return all of the columns in the OpSim database. It queries the OpSim output looking for a specific airmass value on a specified region of the sky. It then uses the Obshistid column from the output to generate an ObservationMetaData which is in turn used to create a catalog of stars.

```
from lsst.sims.catalogs.generation.db import CatalogDBObject
from lsst.sims.catUtils.baseCatalogModels import OpSim3_61DBObject

obsMD=OpSim3_61DBObject()

#The code below will query the OpSim data base object created above.
#The query will be based on a box in RA, Dec and a specific airmass value
airmassConstraint = "airmass=1.1" #an SQL constraint that the airmass must be equal to
                                  #the passed value



skyBounds = SpatialBounds.getSpatialBounds('box', ra, dec, tol) #bounds on region of sky

query = obsMD.executeConstrainedQuery(skyBounds, constraint=airmassConstraint)

#convert q into observation meta data for use in a catalog
obsMetaData = obsMD.getObservationMetaData(query['Opsim_obshistid'][0],
 radiusDeg, makeCircBounds=True)

#create and output a reference catalog of stars based on our query to opSim
dbobj = CatalogDBObject.from_objid('allstars')

catalog = dbobj.getCatalog('ref_catalog_star', obs_metadata = obsMetaData)
catalog.write_catalog('stars_airmass_test.dat')
```

Below is a list of OpSim column names that will be returned. These names derive from the input interface to phoSim. The explanations come from the table on page 95 of the main phoSim document. Note that, while phoSim requires angles be input in degrees, the OpSim database stores all angles in radians (so outputs from executeConstrainedQuery above will be in radians); the code to output files suitable for phoSim, which we will talk about later, automatically converts to degrees:

- Unrefracted_RA - the RA of the pointing in radians
- Unrefracted_Dec - the Dec of the pointing in radians
- Opsim_obshistid - the obshistid of the pointing
- Opsim_expmjd - the MJD of the pointing
- Opsim_altitude - the altitude of the pointing
- Opsim_azimuth - the azimuth of the pointing
- Opsim_moonra - the RA of the moon in radians
- Opsim_moondec - the Dec of the moon in radians
- Opsim_rotskypos - Angle of sky relative to camera coordinates in radians
- Opsims_rottelpos - Angle of sky relative to telescope in radians
- Opsim_filter - which LSST filter the simulated pointing corresponds to
- Opsim_rawseeing - seeing at zenith at 500nm
- Opsim_sunalt - altitude of the sun in radians
- Opsim_moonalt - altitude of the moon in radians
- Opsim_dist2moon - distance from the pointing to the moon in radians
- Opsim_moonphase - phase of the moon from 0 to 100

Generating ObservationMetaData for an OpSim pointing

Querying OpSim for the desired range in RA, Dec, MJD, airmass, etc. will give the user values of obshistid corresponding to pointings that meet her criteria (assuming that 'Opsim_obshistid' is included in the list of columns to be output by the query). To go from an obshistid to ObservationMetaData suitable for use in a catalog, one must use the method getObservationMetaData which is a part of the OpSim3_61DBObject class. This method is very easy to use. It takes as arguments the obshistid, a radius in degrees, and key word arguments that tell it whether to return the ObservationMetaData for a circular footprint on the sky or a square footprint on the sky. Thus, in the example cod above:

```
myObsMetaData = obsMD.getObservationMetaData(myRows[0][0], 50.0, makeCircBounds=True)
```

will return a circular footprint with a radius of 50 degrees centered on the observation that occupies the first row of myRows. To, instead, create a square footprint that is 100 degrees to a side, just replace makeCircbounds=True with makeBoxBounds=True. The method defaults to makeCircBounds=True if you do not specify.

Return to the main catalog simulations documentation page