Framework Overview -- Catalog Simulations

For example applications of the material discussed on this page, see the CatSimTutorial_SimulationsAHM_1503.ipynb iPython notebook in the CatSim directory of the UWSSG LSST-Tutorials repository.

In order to generate a catalog in the catalog simulations framework, the user must write a daughter class that inherits from the base class InstanceCatalog (which is defined in sims_catalogs_measures/python/lsst/sims/catalogs/measures/instance/InstanceCatalog.py). The catalog class will ultimately use a table of data describing (astronomical) objects and a Python ObservationMetaData object describing a specific telescope pointing to generate a catalog of simulated observations. This page begins by discussing how catalog classes are written. We will then discuss how to generate database objects. We will then discuss how to use ObservationMetaData objects to specify a pointing.

In addition to the documentation provided on this page, example scripts walking the user through these features exist in

sims_catUtils/examples/tutorials/

Catalog classes

In order to convert a database into a catalog, the user must write a daughter class that inherits from InstanceCatalog. This class can be very simple. It only needs to specify the names of the columns to be output into the catalog. Consider this example:

```
from lsst.sims.catalogs.measures.instance import InstanceCatalog

class myCatalogClass(InstanceCatalog):
    column_outputs=['id','raJ2000','decJ2000']

#some code reading in or creating a data base object myDatabase
#(we will discuss that more below)

catalog = myCatalogClass(myDatabase)
catalog.write_catalog("catalogFile.txt")
```

Assuming that the columns 'id', 'raJ2000', and 'decJ2000' exist in myDatabase, the above code will write them into the file catalogFile.txt.

Suppose that myDataBase did not contain all of the columns that the user wanted to output. There are two ways that a catalog class can fill in the gaps. The first is to assign a default value to the column in the catalog class. This is illustrated below.

```
from lsst.sims.catalogs.measures.instance import InstanceCatalog

class myCatalogClass(InstanceCatalog):
    column_outputs=['id','raJ2000','decJ2000','nonsenseColumn']
    default_columns=[('nonsenseColumn','word',(str,4))]

#some code reading in or creating a data base object myDatabase
#(we will discuss that more below)

catalog = myCatalogClass(myDatabase)
catalog.write_catalog("catalogFile.txt")
```

As the name suggests, the column 'nonsenseColumn' does not exist in myDatabase. The default_columns statement in the class definition tells myCatalog that, if it cannot find a column named 'nonsenseColumn' in myDatabase, it is to assign a string of length 4 to that column and fill it with the entry 'word'. For another example of default_columns, consider the following class taken from /sims_catUtils/python/lsst/sims/catUtils/exampleCatalogDefinitions /phoSimCatalogExamples.py

```
class PhoSimCatalogPoint(PhosimInputBase, AstrometryStars, PhotometryStars, EBVmixin):
    catalog_type = 'phoSim_catalog_POINT'
    column_outputs = ['prefix', 'uniqueId','raPhoSim','decPhoSim','phoSimMagNorm','sedFilepath',
                      'redshift','shear1','shear2','kappa','raOffset','decOffset',
                      'spatialmodel','galacticExtinctionModel','galacticAv','galacticRv',
                      'internalExtinctionModel']
    default_columns = [('redshift', 0., float),('shear1', 0., float), ('shear2', 0., float),
                       ('kappa', 0., float), ('raOffset', 0., float), ('decOffset', 0., float),
                       ('galacticExtinctionModel', 'CCM', (str,3)),
                       ('internalExtinctionModel', 'none', (str,4))]
    default_formats = {'S':'%s', 'f':'%.9g', 'i':'%i'}
    delimiter = " "
    spatialModel = "point"
    transformations = {'raPhoSim':numpy.degrees, 'decPhoSim':numpy.degrees}
```

Catalog classes, can, of course, be more complicated than this if you want access to more functionality.

The second way that a catalog class can fill in columns missing from the database is to calculate them from columns that do exist in the database. A classic example of this would be if myDataBase contained the file names of SEDs associated with your objects and you wanted to convert them into magnitudes observed through the LSST filters. The catalog class would have to provide a method to read in the SED files, read in the transmission curves associated with the LSST filters, convolve them into a flux, and convert that into a magnitude. Many methods already exist within the catalog simulations framework to provide functionality such as this. They are stored in mixin classes. In Python, a mixin is a class which exists solely so that other classes can inherit from it and use its functionality. To gain access to the functionality provided by the catalog simulations mixins, the user just needs to make sure that her catalog class inherits from the appropriate mixins.

To add stellar photometry to our test catalog, we would modify the code above by adding

```
from lsst.sims.catalogs.measures.instance import InstanceCatalog

#import the stellar photometry mixin
from lsst.sims.photUtils.Photometry import PhotometryStars

#make sure that myCatalogClass inherits from the stellar photometry mixin
class myCatalogClass(InstanceCatalog,PhotometryStar):

    column_outputs=['id','raJ2000','decJ2000','nonsenseColumn',
                    'lsst_u','lsst_g','lsst_r','lsst_i',lsst_z','lsst_y']
    default_columns=[('nonsenseColumn','word',(str,4))]




#some code reading in or creating a data base object myDatabase
#(we will discuss that more below)

catalog = myCatalogClass(myDatabase)
catalog.write_catalog("catalogFile.txt")
```

Inheriting from the mixin PhotometryStars gives myCatalogClass access to the methods which calculate the photometry columns 'lsst_u', 'lsst_g', etc. (provided that the database driving the catalog contains the necessary input data to calculate the photometry, namely: valid SED files for all of the objects in the database).

To see a more real example of this, consider the file sims_catUtils/python/lsst/sims/exampleCatalogDefinitions/obsCatalogExamples.py.

```
"""Instance Catalog"""
import numpy
import eups
from lsst.sims.catalogs.measures.instance import InstanceCatalog
from lsst.sims.coordUtils.Astrometry import AstrometryStars, CameraCoords
from lsst.sims.photUtils.Photometry import PhotometryStars
from lsst.obs.lsstSim.utils import loadCamera

class ObsStarCatalogBase(InstanceCatalog, AstrometryStars, PhotometryStars, CameraCoords):
    comment_char = ''
    camera = loadCamera(eups.productDir('obs_lsstSim'))
    catalog_type = 'obs_star_cat'
    column_outputs = ['id', 'raObserved', 'decObserved', 'lsst_u', 'sigma_lsst_u',
                      'lsst_g', 'sigma_lsst_g', 'lsst_r', 'sigma_lsst_r', 'lsst_i',
                      'sigma_lsst_i', 'lsst_z', 'sigma_lsst_z', 'lsst_y', 'sigma_lsst_y',
                      'chipName', 'xPix', 'yPix']
    default_formats = {'S':'%s', 'f':'%.8f', 'i':'%i'}
```

The class ObsStarCatalogBase inherits from InstanceCatalog as well as the classes AstrometryStars, PhotometryStars, and CameraCoords. The 'id' column already exists in the catalog's database. AstrometryStars is a mixin which gives ObsStarCatalogBase access to methods which calculate astrometric quantities for stars (e.g. raObserved and decObserved). PhotometryStars is a mixin which gives ObsStarCatalogBase access to methods which calculate photometric quantities for stars (lsst_u, lsst_g, lsst_r...). CameraCoords is a mixin which gives ObsStarCatalogBase access to methods which turn astrometric coordinates into coordinates on the physical camera (chipName, xPix, and yPix).

When a catalog class is asked to supply a column value, the InstanceCatalog searches for that value in the following manner (using the method column_by_name defined in InstanceCatalog.py).

- First, the catalog investigates whether or not it has a method to calculate that column value from other data. These methods are referred to as 'getters' because InstanceCatalog looks for them by examining itself to see if it has a method named get_[columnName] (if a getter defines multiple columns, it can have any name that begins with 'get_', but must be marked with the @compound decorator – see the method get_allMags() in /sims_photUtils/python/lsst/sims/photUtils/Photometry.py for an example)

- If that fails, the catalog examines the database to see if the column exists there natively
- If that fails, the catalog checks to see if it has a default_columns statement telling it the value of the column
- If that fails, the catalog will throw an error

List of columns for which getters exist

See this page.

To learn how to write your own getters, see this page.

CatalogDBObject - the data behind the catalog

This is a rough-and-ready user's guide to database objects designed specifically to interface with catalog classes. More general database interfaces do exist in the stack. For a more detailed description of these, see this page.

In the catalog simulations framework, catalog data is supplied by daughter classes of the base class CatalogDBObject, which is defined in /sims_catalogs_generation/python/lsst/sims/catalogs/generation/db/dbConnection.py. Instantiations of CatalogDBObject read and store tables taken from larger SQL database files. For our purposes, tables are subsets of the database devoted to certain types of objects. For example, in the LSST database hosted at the University of Washington, there are tables for galaxies, stars, main sequence stars, RR Lyrae, Cepheid variables, White Dwarfs, and solar system objects (among others).

As we saw in our example code above, the CatalogDBObject is passed as the first argument when generating a new catalog object. If the user does not wish to supply her own data and is content to use the data stored in the LSST database, several base classes of CatalogDBObject are provided in /sims_catUtils/python/lsst/sims/catUtils/baseCatalogModels/. CatalogDBObject had a meta-class, CatalogDBObjectMeta, which keeps track of every CatalogDBObject daughter class that has been defined. One way to explore what options exist in CatalogDBObject daughter classes is by examining the contents of this registry:

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

for name in CatalogDBObject.registry:
    print name
```

To actually instantiate a CatalogDBObject using this registry, one needs to call the class method from_objid(), for example

```
myDb = CatalogDBObject.from_objid('rrlystars')
```

will instantiate the class RRLyStarObj, which accesses the table of RRLyrae stars on the UW database.

In order to learn what columns exist in an LSST database table, the user can use the CatalogDBObject.show_db_columns() method. For example

```
from lsst.sims.catalogs.generation.db import CatalogDBObject
from lsst.sims.catUtils.baseCatalogModels import *
myDB = CatalogDBObject.from_objid('cepheidstars')
myDB.show_db_columns()
```

will print the names and datatypes of all of the columns contained within the cepheidstars table onto the screen. In this case 'cepheidstars' is the objid from the CepheidStarObj class in sims_catUtils/python/lsst/sims/catUtils/baseCatalogModels/StarModels.py.

A full list of the pre-defined CatalogDBObject daughter classes can be found here.

Using custom data

Should the user wish to use her own (non-LSST) simulations data, she should consult the example iPython notebook reading_in_custom_data.ipynb in sims_catUtils/examples/tutorials/.

ObservationMetaData - restricting the catalog to a specific pointing

The code immediately above reads a database table into a catalog and outputs every object in that table. If the user is interested in simulating only a patch of the sky at a time, this is not desirable behavior. For this reason, catalog classes accept a keyword argument obs_metadata to limit the objects returned by write_catalog(). The obs_metadata keyword expects an instantiation of the ObservationMetaData class which is defined in /sims_catalogs_generation/python/lsst/sims/catalogs/generation/ObservationMetaData.py. This class carries with it several variables that can be used to define a simulated observation. ObservationMetaData instantiations are used by CatSim to:

- Define the field of view of a simulated observation.
- Store the date of a simulated observation.
- Store the filter or list of filters used by a simulated observation.
- Store the m5 values associated with those filters.
- Store the state of the telescope during a simulated observation.

This information is passed in as keyword arguments when the ObservationMetaData class is instantiated. To see the full list of variables stored in the ObservationMetaData class:

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

help(ObservationMetaData)
```

Here is an example of an ObservationMetaData corresponding to a circular field of view of 1-degree radius centered on RA=200 degrees, Dec=10 degrees, observed through the i filter.

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

myObsMetaData = ObservationMetaData(mjd = 24000,
                   unrefractedRA=200., unrefractedDec=10.,
                   boundType='circle', boundLength=1.,
                   bandpassName = 'i')
```

Here is an example of an ObservationMetaData for a square footprint that is ten degrees on a side (the boundLength is half the length of the side). One can specify a rectangular field of view by making boundLength a list (with the RA half-side-length first).

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

myObsMetaData = ObservationMetaData(mjd=2400,
                  unrefractedRA=15., unrefractedDec=15.,
                  boundType='box', boundLength=5.,
                  bandpassName = 'i')
```

To incorporate ObservationMetaData into a catalog use the following examples:

```
import os
from lsst.sims.catalogs.generation.utils import makeStarTestDB, myTestStars
from lsst.sims.catalogs.measures.instance import InstanceCatalog
from lsst.sims.catalogs.generation.db import ObservationMetaData

class myCatalogClass(InstanceCatalog):
    column_outputs=['id','raJ2000','decJ2000','nonsenseColumn']
    default_columns=[('nonsenseColumn','word',(str,4))]

if os.path.exists('testDatabase.db'):
    os.unlink('testDatabase.db') #delete the database if it exists

makeStarTestDB(size=10000, seedVal=1) #create a stars table of 10000 elements

myDatabase = myTestStars() #read in the table 'stars' into a CatalogDBObject daughter class

#generate a rectangular field of view
myObservationMetaData = ObservationMetaData(mjd=24000, unrefractedRA=150., unrefractedDec=0.,
                                            boundType='box', boundLength=[50., 20.])

catalog = myCatalogClass(myDatabase,obs_metadata=myObservationMetaData)
catalog.write_catalog('catalogFileLimited.txt')
```

for a rectangular field of view and:

```
import os
from lsst.sims.catalogs.generation.utils import makeStarTestDB, myTestStars
from lsst.sims.catalogs.measures.instance import InstanceCatalog
from lsst.sims.catalogs.generation.db import ObservationMetaData

class myCatalogClass(InstanceCatalog):
    column_outputs=['id','raJ2000','decJ2000','nonsenseColumn']
    default_columns=[('nonsenseColumn','word',(str,4))]

if os.path.exists('testDatabase.db'):
    os.unlink('testDatabase.db') #delete the database if it exists

makeStarTestDB(size=10000, seedVal=1) #create a stars table of 10000 elements

myDatabase = myTestStars() #read in the table 'stars' into a CatalogDBObject daughter class

#generate a circular field of view
myObservationMetaData = ObservationMetaData(mjd=24000, unrefractedRA=100., unrefractedDec=5.,
                                            boundType='circle', boundLength=20.)

catalog = myCatalogClass(myDatabase,obs_metadata=myObservationMetaData)
catalog.write_catalog('catalogFileLimitedCirc.txt')
```

for a circular field of view.

Pre-defined catalog classes

Just like CatalogDBObject, InstanceCatalog is associated with a meta-class the stores a registry of all defined InstanceCatalog classes. If the user would rather avoid the trouble of writing her own catalog classes, several different catalog classes exist in the directory /sims_catUtils/python/lsst/sims/catutils /exampleCatalogDefinitions/. As with the registry of CatalogDBObject daughter classes, these classes can be inspected using

```
from lsst.sims.catalogs.measures.instance import InstanceCatalog
from lsst.sims.catUtils.exampleCatalogDefinitions import *

for name in InstanceCatalog.registry:
    print name
```

In this case, the InstanceCatalog classes in the registry can be instantiated using the CatalogDBObject method getCatalog, i.e.

```
from lsst.sims.catalogs.generation.db import CatalogDBObject
from lsst.sims.catalogs.measures.instance import InstanceCatalog
from lsst.sims.catUtils.baseCatalogModels import *
from lsst.sims.catUtils.exampleCatalogDefinitions import *

db = CatalogDBObject.from_objid('rrlystars')
db.getCatalog('ref_catalog_star')
```

will create an instantiation of the InstanceCatalog daughter class RefCatalogStarBase linked to an instantiation of the CatalogDBObject daughter class RRLyStarObj.

The available catalog classes are:

In the file refCatalogExamples.py:

- RefCatalogGalaxyBase - (instantiated as 'ref_catalog_galaxy') a catalog for galaxies that simply outputs the contents of the database table
- GalaxyPhotometry - (instantiated as 'galaxy_photometry_cat') a catalog for galaxies that uses the PhotometryGalaxies mixin to calculate photometry of each galaxy component over LSST bandpasses
- RefCatalogStarBase - (instantiated as 'ref_catalog_star') a catalog for stars that outputs the contents of the database table

In the file phosimCatalogExamples.py:

- PhoSimCatalogPoint - (instantiated as 'phoSim_catalog_POINT') generates a phoSim input file for a collection of stars
- PhoSimCatalogZPoint - (instantiated as 'phoSim_catalog_ZPOINT') generates a phoSim input file for a collection of AGN
- PhoSimCatalogSersic2D - (instantiated as 'phoSim_catalog_SERSIC2D') generates a phoSim input file for a collection of galaxy disks and bulges

In the file obsCatalogExamples.py:

- ObsStarCatalogBase - (instantiated as 'obs_str_cat') generates a catalog of stars, calculating both photometry in LSST bands and position on the LSST camera

To use these catalogs, use code similar to this (taken from /sims_catUtils/examples/catalogsExamples.py):

```
import math
from lsst.sims.catalogs.generation.db import CatalogDBObject, ObservationMetaData
#The following is to get the object ids in the registry
import lsst.sims.catUtils.baseCatalogModels as bcm
from lsst.sims.catUtils.exampleCatalogDefinitions import RefCatalogGalaxyBase
from lsst.sims.catalogs.generation.db import makeObsParamsAzAltTel, makeObsParamsRaDecTel

obs_metadata = ObservationMetaData(unrefractedRA=0., unrefractedDec=0., boundType='circle', boundLength=0.01)
dbobj = CatalogDBObject.from_objid('galaxyBase')

#this line is where you use the 'instantiated by....' keys from above
#in this case, we are creating an instantiation of RefGalaxyCatalogBase
#
t = dbobj.getCatalog('ref_catalog_galaxy', obs_metadata=obs_metadata)

filename = 'test_reference.dat'
t.write_catalog(filename, chunk_size=10)
```

Examples of how to create various pre-defined catalogs from the database can be found in /sims_catUtils/examples/catalogsExamples.py.

Return to the main catalog simulations documentation page