Code Overview -- Catalog Simulations

This page summarizes the contents of the different sims packages. It discusses the most useful classes and methods in each and how to import them.

Finding source code

Recall that, to find the source code for any of the packages here, you should look in

```
$LSST_HOME/yourOperatingSystem/packageName/yourVersion/python/lsst/sims/shortPackageName/
```

For example, on my Mac, the source code for sims_photUtils is in

```
lsst_150412/DarwinX86/sims_photUtils/1.0.0-86-g4e41113+4/python/lsst/sims/photUtils/
```

where lsst_150412 is the directory in which I built my stack and 1.0.0-86... is the Git SHA-1 of the version of sims_photUtils I most recently installed.

Sometimes, the package name is broken up further. The source code for sims_catalogs_generation is in

```
lsst_150412/DarwinX86/sims_catalogs_generation/1.0.0-86-g4e41113+4/python/lsst/sims/catalogs/generation/
```

To find the path to the setup version of a package, use

```
eups list -v packageName | grep "setup"
```

Import statements

Because of our directory structure, classes and methods from the LSST sims stack are imported like

```
from lsst.sims.packageName import className
```

so, for example, to import the class PhotometryStars from sims_photUtils, one would use

```
from lsst.sims.photUtils import PhotometryStars
```

In some cases, the source code is further sub-divided in the package directory tree, and an extra level is needed in the import statements. It is also possible that packageName has been broken up into '.' separated words. For example

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

In the discussion below, we present the general format of import statements from each package. We also highlight cases where the source code has been sub-divided inside the directory structure.

Package: sims_utils

Imports from sims_utils take the form

```
from lsst.sims.utils import methodName
```

sims_utils is a package that contains methods that do not depend on any other LSST code. Mostly this means that it contains methods which wrap coordinate transformation routines from PALPY. These are the methods you would use to, for example, convert between equatorial and horizontal coordinates, or find the distance between two points on a sphere. The methods provided by sims_utils are

- calcLmstLast – a method to calculate the local mean sidereal time and the local apparent sidereal time
- galacticFromEquatorial – a method to convert RA, Dec into galactic longitude and latitude

- equatorialFromGalactic – a method to convert galactic longitude and latitude into RA, Dec
- cartesianFromSpherical – convert longitude and latitude into Cartesian coordinates on a unit sphere
- sphericalFromCartesian – convert Cartesian coordinates into longitude and latitude
- rotationMatrixFromVectors – find the matrix to rotate one unit vector into another
- equationOfEquinoxes – see this page.
- calcGmstGast – a method to calculate Greenwich mean and apparent sidereal time
- altAzPaFromRaDec – convert RA, Dec into altitude, azimuth, and parallactic angle
- raDecFromAltAz – convert altitude and azimuth into RA, Dec
- getRotSkyPos – find the angle between north on the sky and 'up' on the camera.
- haversine – find the distance between two points on a sphere
- arcsecFromRadians/radiansFromArcsec – convert radians into arcseconds and vice-versa

sims_utils also defines the class Site which carries data about the LSST site (and can be customized to contain data about any observing site). To access this class use

```
from lsst.sims.utils import Site
```

Package: sims_catalogs_generation

All of the useful code in sims_catalogs_generation is imported like

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

sims_catalogs_generation contains the classes which connect to databases of simulated objects (i.e. this is where the CatalogDBObject class mentioned in the Framework Overview page is defined). sims_catalogs_generation also defines the class ObservationMetaData which stores the meta data describing a simulated telescope pointing. The important classes from sims_catalogs_generation are

- CatalogDBObject – a class which connects and InstanceCatalog to the database (InstanceCatalog will be defined below in our discussion of sims_catalogs_measures).
- DBObject – a class which allows for a general connection to a database file. This is the class you want to use if you want to connect to a database and run your own arbitrary SQL queries.
- fileDBObject – a class which can read in a text file, output it to a database file, and provide a connection to that database which InstanceCatalog can understand. The use of this class is demonstrated in the example iPython notebook

  ```
  sims_catUtils/examples/tutorials/read_in_custom_data.ipynb
  ```

- ObservationMetaData – a class for carrying around data describing a telescope pointing and passing it around in a way that InstanceCatalog can understand.

Package: sims_catalogs_measures

Classes from the package sims_catalogs_measures are imported like

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

The most important class defined in sims_catalogs_measures is the InstanceCatalog class. This is the class which takes the results of the raw database query performed by CatalogDBObject and formats it into a human-readable catalog as specified by the user. In most cases, you will be writing daughter classes that inherit from InstanceCatalog and use its basic infrastructure to create customized catalogs. Examples of this can be found in

```
sims_catUtils/python/lsst/sims/catUtils/exampleCatalogDefinitions/
```

Package: sims_coordUtils

Classes and methods defined by sims_coordUtils are imported like

```
from lsst.sims.coordUtils import className, methodName
```

sims_coordUtils defines methods that apply astrometric affects to the mean RA, Dec values stored in the database of simulated object. It also defines mixin classes that provide getters which allow InstanceCatalog and its daughter classes to calculate columns.

The methods provided by sims_coordUtils are

- refractionCoefficients – a method to calculate the coefficients of atmospheric refraction for a given observing site.
- applyRefraction – a method to apply the coefficients calculated by refractionDoefficients to an object at a given zenith distance.

- applyPrecision – a method to apply precession and nutation to mean RA, Dec provided in the International Celestial Reference System (ICRS).
- applyProperMotion – a method to apply proper motion to mean RA, Dec provided in the ICRS.
- appGeoFromICRS – a method to convert mean RA, Dec provided in the ICRS to an apparent geocentric position by applying precession, nutation, proper motion, parallax, and annual aberration.
- observedFromAppGeo – a method to convert apparent geocentric RA, Dec into observed RA, Dec by applying refraction and diurnal aberration.
- observedFromICRS – a method to convert mean RA, Dec provided in the ICRS to observed position. This is equivalent to calling appGeoFromICRS and observedFromAppGeo in series.
- calculatePupilCoordinates – a method to convert observed RA, Dec into position in the telescope's pupil.
- calculateGnomonicProjection – a method to convert observed RA, Dec into position on the telescope's focal plane assuming a perfect gnomonic projection.
- findChipName – a method that takes an observed RA, Dec (or pupil coordinates) and returns the name of the actual detector on the telescope that sees the object.
- calculatePixelCoordinates – a method that takes an observed RA, Dec (or pupil coordinates) and returns the pixel coordinates in the detector that actually sees the object.
- calculateFocalPlaneCoordinates – a method that takes an observed RA, Dec (or pupil coordinates) and returns the true position in the telescope's focal plane.

The classes provided by sims_coordUtils are (see this page for a list of the actual getters provided by these classes)

- AstrometryBase – a mixin which provides getters for astrometric quantities that are agnostic to whether a source is galactic or extra-galactic.
- AstrometryStars – a mixin which provides getters for astrometric quantities defined for stars (parallax and proper motion are taken into account). Note that this class inherits from AstrometryBase and thus provides all of its functionality as well.
- AstrometryGalaxies – a mixin which provides getters for astrometric quantities defined for galaxies (parallax and proper motion are taken into account). Note that this class inherits from AstrometryBase and thus provides all of its functionality as well.
- CameraCoords – a mixin which provides getters for quantities associated with the camera (chip name, pixel position, and focal plane position).

Package: sims_photUtils

Classes and methods defined in sims_photUtils are imported like

```
from lsst.sims.photUtils import className, methodName
```

sims_photUtils provides methods and classes to calculate the magnitudes of objects in different bandpasses as well as the noise associated with those calculations.

Two very important classes defined in sims_photUtils are Bandpass and Sed. These are the classes which read in and manipulate instrumental throughput curves and spectral energy distributions (SEDs), respectively. Most importantly, the Sed class provides the main methods for calculating the magnitude of a source in a given bandpass.

Methods provided in the Sed class are:

- setSED – assigns an SED from numpy arrays
- readSED_flambda/readSED_fnu – read an SED from a text file.
- setFlatSED – set the SED to have a flat flux density as a function of frequency.
- redshiftSED – a method to apply a redshift to an SED
- setupCCMab – a method to calculate the coefficients of the O'Donnell 1994 dust model
- addCCMDust – a method to apply the dust model calculated by setupCCMab (for historical reasons, the dust model is referred to as CCM, even though the actual model is the O'Donnell model)
- multiplySED – a method to multiply two SEDs together
- calcADU – a method to calculate the number of electronic counts registered for the SED through a given bandpass throughput.
- calcMag – a method to calculate the magnitude of the SED through a given bandpass.
- calcFlux – a method to calculate the flux of the SED through a given bandpass.
- calcFluxNorm/multiplyFluxNorm – methods to normalize the SED to a given magnitude in a given bandpass.
- calcSNR_psf – a method to calculate the signal to noise ratio of an observation of the SED given a bandpass and a sky emission SED.
- calcAstrometricError – a method to calculate the error in astrometric position measured for an object with the given SED given a value of m5.

Other methods provided by sims_photUtils for calculating photometric quantities that are not associated with the Sed class are

- setM5 – a method to renormalize a given sky emission SED so that m5 through a given bandpass has a given value.
- calcM5 – a method to calculate the value of m5 given a sky emission SED and a bandpass.
- calcSNR_gamma – a method to calculate the signal to noise of an SED given an m5 value and bandpasss using the model given by equation 5 of the LSST Overview Paper.

Other classes provided by sims_photUtils are (remember, to see the specific getters provided by the mixins below, go to this page.)

- PhotometricParameters – a class for storing parameters characterizing the photometric performance of a telescope.
- LSSTdefaults – a class for storing default values of observing parameters assumed for LSST.
- PhotometryHardware – a mixin that gives an InstanceCatalog the ability to load data about hardware throughputs for a given telescope (defaults to LSST).
- PhotometryGalaxies – a mixin that gives an InstanceCatalog the ability to calculate the magnitudes of galaxies in LSST filters.
- PhotometryStars – a mixin that gives an InstanceCatalog the ability to calculate the magnitudes of stars in LSST filters.
- EBVbase – a class that provides methods which allow the user to calculate E(B-V) for a given position on the sky using the Schlegel, Finkbeiner, and Davis dust map.
- EBVmixin – a mixin that translates EBVbase into getters.
- CosmologyObject – a class that wraps astropy.cosmology code, allowing the user to calculate cosmological quantities like Omega_m or the Hubble parameter as a function of redshift.
- CosmologyWrapper – a mixin that translates CosmologyObject into a getter for the cosmological distance modulus.

- VariabilityStars/VariabilityGalaxies – mixins which implement the variability model discussed on this page.
- applyIGM – a class with methods for applying intergalactic medium extinction to an Sed.

sims_catalogs_photUtils also provides classes which allow a user to take her own catalog of magnitudes and associate the objects in that catalog with SEDs from the library provided by the stack. These classes are

- selectGalaxySED
- selectStarSED

Examples of their use can be found in

```
sims_photUtils/examples/
```

Finally, the method setupPhotometryCatalog provides a way for a user to 'automatically' associate an InstanceCatalog daughter class with a given ObservationMetaData and CatalogDBObject. In cases where users wish to write many catalogs containing only observations in filters specified by ObservationMetaData instantiations, this method may prove less tedious than writing out InstanceCatalog daughter classes by hand.

Package: sims_catUtils

Classes and methods from sims_catUtils are imported like

```
from lsst.sims.catUtils.subDirectory import className, methodName
```

The available sub-directories will be discussed below.

sims_catUtils is the highest level CatSim package. This is where daughter classes of InstanceCatalog and CatalogDBObject specifically designed to interface with the LSST simulations databases hosted at the University of Washington are defined. It is also where tutorials designed to encompass the functionality of the whole CatSim stack are provided.

Examples and tutorials

The directories

```
sims_catUtils/examples
sims_catUtils/examples/tutorials/
```

contain python scripts and iPython notebooks designed to demonstrate some of the functionality described above. The available examples are

- tutorials/tutorial0[0,1,2,3,4].py – these scripts (and the iPython notebooks of the same name) walk through the most basic functionality of the CatSim stack in what we hope is a logical and progressive order. I highly recommend running through them at least once before you get started.
- tutorials/read_in_custom_data.ipynb – this iPython notebook demonstrates how to read in your own data and interface it with CatSim.
- catalogsExamples.py – examples of code generating different types of catalogs
- configConnection.py – examples illustrating how the LSST stack handles connections to databases
- exampleSDSScatalogs.py – an example of a photometric catalog customized to use SDSS rather than LSST filters. This example defines getters for the SDSS bandpasses. Users wishing to write getters for their own set of bandpasses should consult this example.
- findChipNameExample.py – an example demonstrating how to associate astronomical objects with their positions on a camera's detector.
- generatePhoSimInput.py – an example of how to generate an InstanceCatalog specifically designed to be input to PhoSim.
- makeGalaxyPhotCat.py – an example of a photometric catalog of galaxies.

Sub-directory: baseCatalogModels

Classes from this sub-directory are imported like

```
from lsst.sims.catUtils.baseCatalogModels import className
```

This is the sub-directory in which daughter classes of CatalogDBObject are defined. Each daughter class is designed to interface with a specific table on the University of Washington database. These classes are listed on this page under "DBObject classes which access database tables."

Sub-directory: exampleCatalogDefinitions

Classes from this sub-directory are imported like

```
from lsst.sims.catUtils.exampleCatalogDefinitions import className
```

This is the sub-directory in which daughter classes of InstanceCatalog have been defined. The list of classes defined here is by no means complete, but the should serve as useful examples to the user. They are

- ObsStarCatalogBase – an InstanceCatalog that returns the observed positions, magnitudes, and pixel positions of stars.
- RefCatalogGalaxyBase – an InstanceCatalog that outputs all of the data stored in the database (no calculated quantities) for galaxies.
- GalaxyPhotometry – an InstanceCatalog that adds calculated LSST magnitudes to RefCatalogGalaxyBase.
- RefCatalogStarBase – like RefCatalogGalaxyBase for stars.
- PhoSimCatalogPoint – an InstanceCatalog of point sources designed to be input to PhoSim
- PhoSimCatalogZPoint – an InstanceCatalog of extra-galactic point sources (AGN) designed to be input to PhoSim.
- PhoSimCatalogSersic2D – an InstanceCatalog of extended sources (galaxy bulges and disks) designed to be input to PhoSim

Sub-directory: utils

Methods in this directory are imported like

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

This sub-directory contains the method ObservationMetaDataGenerator which allows the user to read in an OpSim run database and produce realistic ObservationMetaData instantiations from it. Its use is demonstrated in the CatSimTutorial_SimulationAHM_1503.ipynb notebook in the UWSST LSST-Tutorials Git repository.

Sub-directory: galSimInterface

This is where the CatSim-GalSim interface classes are defined. They are discussed in more detail here.

Return to the main catalog simulations documentation page