Catalog Simulations Documentation

Installation and setup

For instructions on installing the simulations code, go here.

Once you have installed the sims packages, you will need to setup the CatSim packages using this command

```
setup sims_catUtils -t sims
```

Any time you open a new shell and want to run CatSim in it, you need to

```
source loadLSST.bash
setup sims_catUtils -t sims
```

The first will add eups (the LSST package management software) to your PATH. The second will tell eups to add the CatSim code to your PATH.

Note: If you want to run the CatSim-GalSim interface which allows you to seamlessly convert CatSim catalogs into GalSim images, you should replace sims_catUtils with sims_GalSimInterface in the commands above.

Organization of Software

The software in the LSST stack is organized using the package management program EUPS. Broadly speaking, every time you install a new version of the stack using

```
eups distrib install lsst_sims -t sims
```

the packages that you download get placed in their own unique directory tree that looks like

```
$LSST_HOME/yourOperatingSystem/packageName/versionNumber/
```

where LSST_HOME is set by the loadLSST.bash script, yourOperatingSystem will be either `DarwinX86` (Mac) or `Linux64` (Linux), packageName is the name of the LSST software package (e.g. sims_photUtils, daf_base, etc.), and versionNumber will look like either a Git tag or a Git SHA-1. When you tell EUPS to setup a particular version of a package using

```
setup packagName -t versionTag
```

as in

```
setup sims_catUtils -t sims
```

EUPS takes the directory associated with the specified package version (and all of its dependencies) and appends it to your PATH.

To see all of the versions of a package that EUPS knows about, use

```
eups list -v packageName
```

This will print out the path to every version as well as the version tag (e.g. 'sims', 'current', or an LSST-specific build number `bNNNN`) associated with that version. Note that versions of a given package can have multiple versionTags. This means that they are associated with multiple builds of the LSST stack. The version that is actually in your path will also be marked as 'setup'. Thus, to see a list of all of the packages that EUPS has setup, do

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

All of this is to say that if you ever need to inspect the source code of a package you are using, you can find its base directory using 'eups list' as described above. Note that once you are in the base directory, you will still need to do some exploring to find the code you are looking for. For example, the source code for the sims_photUtils package is actually stored in

```
$LSST_HOME/yourOperatingSystem/sims_photUtils/versionNumber/python/lsst
/sims/photUtils/
```

This directory structure was implemented for the sake of unambiguous module importing in the stack's python code.

Tutorials

Note: to run these tutorials, you will need to be able to connect to the University of Washington databases. Instructions for accessing these databases can be found here.

You will find tutorial scripts and iPython notebooks in

```
$LSST_HOME/yourOperatingSystem/sims_catUtils/yourVersion/examples
/tutorials/
```

yourOperatingSystem will be 'DarwinX86' for Mac users and 'Linux64' for Linux users. yourVersion denotes the version of the sims_catUtils package that you installed. This will likely look like a Git SHA-1.

There is also a separate GitHub repository containing example iPython notebooks created for the University of Washington LSST group's internal meetings. These can also be helpful in learning how to use the CatSim stack. The repository can be found here and can be cloned using

```
git clone https://github.com/uwssg/LSST-Tutorials.git
```

Basic CatSim philosophy

The CatSim stack principally exists to create catalogs from simulated universes. The default simulated universe that CatSim accesses lives on a machine at the University of Washington called "fatboy". This simulated universe is really a database that consists of a distribution of galaxies drawn from the Millennium N-body simulation, and Milky Way stars generated with the GalFast software. Even though catalogs are generated by querying fatboy's database, CatSim is designed so that the user should never have to write any raw SQL queries. This is due to the way the catalog-generating classes in CatSim have been written. In broad strokes:

- The user instantiates a daughter of the **`InstanceCatalog`** class which is in charge of actually writing the catalog. This is the class that contains information regarding what astronomical objects and what data regarding those objects should be written to the catalog.
- The user passes the InstanceCatalog an instantiation of a daughter of the **`CatalogDBObject`** class. The `CatalogDBObject` class is the class which actually manipulates the connection to fatboy. Specific `CatalogDBobject` classes are written to connect to specific tables in the fatboy database. Thus, there is one `CatalogDBObject` class for galaxies, a different `CatalogDBObject` class for Solar System objects, a different `CatalogDBObject` for main sequence stars, a different `CatalogDBObject` for white dwarfs, etc. `CatalogDBObject` daughter classes provide one other purpose: naming conventions for data columns vary between fatboy and the `InstanceCatalog` classes (and, indeed, between tables in fatboy). Declination is called `dec` in the galaxy table but `decl` in the star tables. `CatalogDBObject` classes provide simple mappings that smooth over these differences and put all of the database columns into a uniform syntax.
- The user also passes the `InstanceCatalog` an instantiation of the **`ObservationMetaData`** class. This is the class which contains data about the state of the simulated telescope. For the purposes of generating a catalog, the `ObservationMetaData` provides the RA and Dec at which the telescope is pointed as well as the size and shape of its field of view (i.e. it controls the question "which objects in fatboy's database are actually seen by my telescope and thus should be written to my catalog?").

When you ask your `InstanceCatalog` to write itself out, what actually happens is that the `CatalogDBObject` performs a query on fatboy using information from the `ObservationMetaData` to tell it which objects to return and information from the `InstanceCatalog` to tell it what data columns associated with those objects to return.

The Framework Overview page explains how these three structures interact to create a simulated catalog.

The Code Overview page explains the contents of the different source code packages involved in CatSim.

The CatSim-GalSim documentation page refers users to examples explaining how CatSim interfaces with GalSim.

The Database Schema page lists the tables and columns provided in the University of Washington simulation databases.

The Database Contents page explains the physical origins of the tables referenced in the Database Schema page above.

The Database Object page explains the functionality provided by the CatSim database interface classes.

The Provided Getters page list the calculated columns provided by the mixin classes distributed with CatSim.

The How to Write Your Own Getter page explains how to write your own getters to calculate new quantities from those provided by the database.

The OpSim Query page explains how to use CatSim to query OpSim runs.

The Simulated Photometry page explains how to write getters to calculate magnitudes in non-LSST bandpass systems.

The Variability Model page explains how variability models are implemented in CatSim.

The SED page explains the spectral energy density models distributed with CatSim.

The Throughputs page explains the throughput models (telescope response curves, atmospheric extinction, etc.) distributed with CatSim.