Database Object classes in the Catalogs Simulation Framework

This is a detailed description of the database interfaces available in the stack. For a more basic, ready-to-use description, see this page.

As discussed on the main CatSim documentation page, CatSim generates catalogs by querying databases of simulated objects. However, CatSim does not expect users to connect to or write queries on those databases by hand. Instead, CatSim provides classes which manage the database connection for the user. These classes are defined in the package sims_catalogs_generation. Their source code can be found in

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

There are three classes defined in this file:

- DBObject – This class will create a connection to a database file. It leaves the querying of that database to the user. This is the class to use if you want to write your own SQL queries by hand. This is not the class to use if you are expecting to interface your database with the InstanceCatalog class to produce a CatSim-generated catalog.
- CatalogDBObject – This class provides all the functionality of DBObject while adding the ability to automatically generate SQL queries based on just a list of column names. This is the class used by InstanceCatalog to create catalogs from databases. CatalogDBObject also provides functionality that allows the user to map columns from an arbitrary database file into a format that InstanceCatalog expect.
- fileDBObject – This class provides the ability to read in a text file and then map it to a CatalogDBObject and either connect that to an InstanceCatalog or output it as a database file for future use.

Users wishing to see an example of how to use CatalogDBObject and fileDBObject to connect a database to an InstanceCatalog should consult the iPython notebook

```
sims_catUtils/examples/tutorials/reading_in_custom_data.ipynb
```

Below we discuss the functionality provided by these three classes.

DBObject

The DBObject class is the most basic way of interfacing with a database. It is designed to connect to a database and allow the user to execute arbitrary queries (written in SQL) on any table or tables in the database.

To connect to a database, you must pass the following kwargs to the DBObject constructor

- database – the name of the database file to which you are connecting (e.g. 'myDatabaseFile.db')
- driver – the dialect of the database to which you are connecting (e.g. 'sqlite' or 'mssql')
- host – (optional) the host machine containing the database (if it is remote)
- port – (optional) the port to connect to on the host machine

You will note that, if you are connecting to a remote database, there is no place for you to enter your username and password. This is because CatSim handles remote database access using the DbAuth class from the LSST Data Management software package daf_persistence. To use this to connect to your own remote database, you must add an entry to your $HOME/.lsst/db-auth.paf file. For an example of what this should look like, consult step (3) of the instructions for connecting to the UW-hosted databases found here.

Once you have connected to your database file, DBObject allows you to call the following methods:

- get_table_names – returns a list of the tables contained in the database
- get_column_names – returns either a dict of lists of column names with the dict keyed to table names, or just a list of the column names in a specified table.
- execute_arbitrary – execute an arbitrary SQL query provided by the user. Results will be returned as a numpy recarray.
- get_arbitrary_chunk_iterator – execute an arbitrary SQL query provided by the user. Results will be returned as an iterator over chunks of rows (i.e. if you specify chunk_size=1000, this will return the rows to you 1000 at a time).

CatalogDBObject

CatalogDBObject is a daughter class of DBObject (meaning that CatalogDBObjects have access to all of the functionality of DBObjects). It is designed specifically to connect databases with InstanceCatalogs (a class defined in /python/lsst/sims/catalogs/measures/instance/InstanceCatalog.py in sims_catalogs_measures). In addition to the functionality of DBObject, CatalogDBObject contains functionality that allows it to take a list of pre-defined columns and return an SQL query on those columns. This is the functionality defined in the CatalogDBObjects section of the framework overview page.

CatalogDBObject contains several member variables that are used to perform InstanceCatalog-based queries:

- epoch is a double that defines the epoch against which astrometric quantities are calculated (i.e. the epoch of the equinox being used)
- objid is a string uniquely identifying each daughter class of CatalogDBObject. The metaclass of CatalogDBObject creates a registry of these daughter classes so that, at run time, the user can create an instantiation of a specific CatalogDBObject daughter class using

```
CatalogDBObject.from_objid('myDaughterClass')
```

assuming that myDaughterClass's objid='myDaughterClass'. The framework overview page contains some concrete examples of this functionality in action.

- tableid is a string indicating what table within the database this particular CatalogDBObject will query.
- idColKey is a string indicating the name of the column in the database table that uniquely identifies each astronomical object
- columns is a list of columns transformations made by this CatalogDBObject to convert raw database columns into outputs the InstanceCatalog expects (converting 'ra' to 'raJ2000', etc.). For an example of how this is used, see the iPython notebook

```
sims_catUtils/examples/tutorials/reading_in_custom_data.ipynb
```

- dbDefaultValues is a dict containing default values to assign to columns in the event that they are 'None' in the database (Note: the defaulted columns must exist in the database; this default applies only if the database-assigned value is None)
- raColName is a string indicating the name of the column in the database containing RA
- decColNam is a string indicating the name of the column in the database containing dec

Note: if you are defining your own CatalogDBObject, you must specify an idColKey, or an error will be thrown.

CatalogDBObject methods are as follows (recall that CatalogDBObject also has access to all of the methods defined for DBObject)

- query_columns – This method accepts a list of column names and automatically construct an SQL query to pull those columns (subject to contraints imposed by an ObservationMetaData instantiation) from the database. This is the principal innovation going from DBObject to CatalogDBObject.
- from_objid – This is a method which allows you to instantiate a daughter class of CatalogDBobject by referring to its objid. See the framework overview page (specifically the CatalogDBObject subsection) for an example of this code in-use.
- getCatalog – This is the equivalent of from_objid for catalog classes. If you know the catalog_type member variable of a specific catalog class, you can instantiate that catalog class from a CatalogDBObject using myCatalogDBObject.getCatalog('myCatalogType'). See the 'Pre-defined catalog classes' section of the framework overview page for an example of how to use this code.

fileDBObject

The fileDBObject class is a daughter of CatalogDBObject whose constructor is designed to read in a text file and convert it into a database file. To see how this is done, consult the iPython notebook

```
sims_catUtils/examples/tutorials/reading_in_custom_data.ipynb
```

Return to the Catalog Simulations Framework Overview page.

Return to the main catalog simulations documentation page.