Variability models in the Catalogs Simulation Framework

The code driving our variability models is a part of the sims_phoUtils package. Specifically, it is saved in

```
sims_photUtils/python/lsst/sims/photUtils/Variability.py
```

This code defines two mixins, VariabilityStars and VariabilityGalaxies. These mixins provide getters for the columns 'delta_lsst_u', 'delta_lsst_g', etc. and, in the case of VariabilityGalaxies, 'delta_uAgn', 'delta_gAgn', etc. When the photometry getters (see this page) calculate the magnitudes 'lsst_u' or 'uAgn', they check to see if the InstanceCatalog contains getters for 'delta_lsst_u' or 'delta_uAgn'. If it does, these values are calculated and added to the baseline magnitudes before output. Thus, in order to include variability in a catalog, users only have to make sure that the InstanceCatalog daughter class inherits from either VariabilityStars or VariabilityGalaxies. Similarly, if users wish to calculate magnitudes in a different, non-LSST system (e.g. calculate columns such as 'myCustomMagnitude'), they just need to make sure that getter methods are defined for columns named 'delta_myCustomMagnitude' (or whatever the baseline columns are named). Assuming that the getters for 'myCustomMagnitude' are written according to the formalism specified on the Simulated Photometry page, CatSim will know what to do.

Below, we discuss how the provided variability models behind 'delta_lsst_u', 'delta_lsst_g', etc. are actually encoded.

How does Variability calculate the modified magnitudes?

There are multiple models of variability stored in the catalog simulations framework (one for each unique physical source of variability). Currently, all of the provided variability models are light curve models: example light curves (magnitude as a function of time) are stored for each of the LSST bands. When a variability model is called, it is passed in parameters initializing the light curve and an MJD (mean Julian date) for the observation. The variability model returns the change in magnitude relative to the mean magnitude for that model at that MJD. Every variable object stored in the LSST database has an attribute 'varParamStr' which is a string specifying what specific variability model corresponds to that object and what parameters need to be given to correctly initialize the model. This string is read in as an ordinary database column and passed to the method applyVariability which belongs to the class Variability, from which VariabilityStars and VariabilityGalaxies inherit. Below, for example, is the getter for 'delta_lsst_u', 'delta_lsst_g', etc. in the mixin VariabilityStars

```
    @compound('delta_lsst_u', 'delta_lsst_g', 'delta_lsst_r',
             'delta_lsst_i', 'delta_lsst_z', 'delta_lsst_y')
    def get_stellar_variability(self):
        """
        Getter for the change in magnitudes due to stellar
        variability.  The PhotometryStars mixin is clever enough
        to automatically add this to the baseline magnitude.
        """
        varParams = self.column_by_name('varParamStr')

        output = numpy.empty((6,len(varParams)))

        for ii, vv in enumerate(varParams):
            if vv != numpy.unicode_("None") and \
               self.obs_metadata is not None and \
               self.obs_metadata.mjd is not None:

                deltaMag = self.applyVariability(vv)
                output[0][ii] = deltaMag['u']
                output[1][ii] = deltaMag['g']
                output[2][ii] = deltaMag['r']
                output[3][ii] = deltaMag['i']
                output[4][ii] = deltaMag['z']
                output[5][ii] = deltaMag['y']

            else:
                output[0][ii] = 0.0
                output[1][ii] = 0.0
                output[2][ii] = 0.0
                output[3][ii] = 0.0
                output[4][ii] = 0.0
                output[5][ii] = 0.0

        return output
```

Variability.applyVariability interprets the varParamStr and calls the correct variability model, returning a dict of delta_magnitudes.

How are the variability models encoded?

In addition to applyVariability, the Variability class contains method for each of the supported variability light curve models (e.g. 'applyMflare' or 'applyRRly'). These methods are stored in a registry dict Variability._methodRegistry. The keys of this dict are included in varParamStr. When applyVariability is called, it extracts the appropriate key from varParamStr and calls the corresponding method. Here is the source code for applyVariability

```
    def applyVariability(self, varParams):
        """
        varParams will be the varParamStr column from the data base
        This method uses json to convert that into a machine-readable object
        it uses the varMethodName to select the correct variability method from the

        dict self._methodRegistry

        it uses then feeds the pars array to that method, under the assumption
        that the parameters needed by the method can be found therein

        @param [in] varParams is a string object (readable by json) that tells
        us which variability model to use

        @param [out] output is a dict of magnitude offsets keyed to the filter name
        e.g. output['u'] is the magnitude offset in the u band
        """

        if self.variabilityInitialized == False:
            self.initializeVariability(doCache=True)

        varCmd = json.loads(varParams)
        method = varCmd['varMethodName']
        params = varCmd['pars']
        expmjd=self.obs_metadata.mjd
        output = self._methodRegistry[method](self, params,expmjd)
        return output
```

The registry dict Variability._methodRegistry is constructed using decorators. If one examines the source code in Variability.py, one sees that the class Variability is declared as:

```
@register_class
class Variability(PhotometryBase):
```

The @register_class is a decorator defined in sims_catalogs_measures/python/lsst/sims/catalogs/measures/instance/decorators.py. When the class Variability is declared (not when it is instantiated by the user, but when it is declared by Python in preparation for a user instantiation), this decorator looks at the class and, if it does not have an attribute _methodRegistry, it creates an empty dictionary _methodRegistry. The decorator then walks through the methods defined in Variability (or whatever class @register_class has been applied to) and adds any method with an attribute _registryKey to the dictionary _methodRegistry. This is how the catalog simulator stores its physical variability models.

The actual variability models are at the end of Variability.py and they are declared something like

```
    @register_method('applyAgn')
    def applyAgn(self, params, expmjd_in):
```

@register_method is another decorator (also defined in decorators.py). This decorator modifies any method to which it is applied by adding a member attribute _registryKey and setting it equal to the argument of @register_method. Thus, when @register_class walks through the methods of Variability, it will log all of the methods marked by @register_method and store them in _methodRegistry. Put another way, after the class has been declared, a call to

```
_methodRegistry['applyAgn'](arguments)
```

will call the method applyAgn defined above.

Return to main catalog simulations documentation page.