Metric Calculation Package Reorgnization

Directory Structure

```
package
 bin                          # Shell scripts
 bin.src                      # Command line python scripts
 docs                         # Non-API documentation
 pipelines                    # YAML pipeline definition files (may have subdirs)
 python                       # All tested python code lives here
    lsst
        faro
            summary           # Tasks that summarize other metric measurements
            base              # Base classes that others can inherit from
            preparation       # Tasks that prepare input for other downstream tasks
            measurement       # Tasks that compute metric measurements
            scripts           # Code invoked by command line python scripts in the bin directory
            utils             # Utility code used by other classes
 tests                        # Unit tests for code under the python directory
    data                      # Data serving unit tests
 ups                          # EUPS directory
```

Class Name Conventions

Connections classes consist of the name of the measurement class they are associated with, plus "Connections" (e.g., `MatchedCatalogTaskConnections` is the connections class for `MatchedCatalogTask`).

Always use the singular form of the dataset type in names. For example, use "Catalog" instead of "Catalogs", "Tract" instead of "Tracts."

Instead of using "MatchedCatalog" in class names, specify *what* has been matched (e.g., `MatchedTract, MatchedPatch` or `MatchedVisit`).

Measurement tasks:

Measurement tasks and their associated config classes have names such as `MetricnameTask/MetricnameTaskConfig` (e.g., `PA1Task/PA1TaskConfig` for metric PA1). The names are CamelCase with the first letter capitalized.

- Classes that can calculate multiple metrics by means of different configs use names like `AMxTask`, where "x" denotes that multiple metrics (AM1, AM2, AM3; i.e., x=1, 2, or 3) can be calculated from the class `AMxTask`. (This is mostly going to apply to KPMs that have defined names such as AM1, ABF1, etc.)

The name of the module containing the class definitions is CamelCase with leading uppercase (e.g.,`TractMeasureTasks.py`). Modules group classes by the dataset type on which metrics will be calculated, and the dataset type should appear in the name. For example, `MatchedVisitMeasureTasks.py` contains measurement tasks that are designed to operate on matched catalogs from visits, while `VisitMeasureTasks.py` contains tasks to calculate metrics per visit.

Catalog generation tasks:

Tasks that create/compile the datasets to be passed to measurement tasks currently have names like `MatchedCatalogTractTask` (and associated Config and Connections classes).

I propose rethinking this naming for 2 reasons:

1. to capture the *action* that is performed by the task,
2. to shorten the names a bit

For example, `MatchedCatalogTractTask` performs the matching of catalogs from coadds at the tract scale. Because "matching" implies that catalogs are being combined, we can get rid of "Catalog" from the name (to address point #2 above). To address point #1, start the name with the action that is being performed (i.e., "match") – perhaps something like `MatchTractTask` ?

Aggregation tasks:

Aggregation tasks have names of the form `DatasetTypeAggregationTask` (e.g., `MatchedCatalogsAggregationTask`).

Because these class names can get very long, I propose the following changes:

1. Get rid of the implied "catalogs",
2. Shorten "Aggregation" to "Agg"

For example, instead of `MatchedCatalogsTractAggregationTask` , use `MatchedTractAggTask` .

Black: summary as currently implemented (21 January 2021)

Blue: proposed

Data Unit	Example Metrics	Inputs (Assembly) Preparation (Prep)	Analysis (Analysis) Measurement (Meas)	Measure	Aggregation (Agg) Summary / (Roll-up)
Generic catalog analysis (used as base classes)	number sources	CatalogAssemblyBaseClass (??)	CatalogAnalysisBaseTask CatalogAnalysisBaseTask		CatalogsAggregationBaseTask CatalogSummaryBaseTask
Matched sources analysis within patch, single-band		**Base:** MatchedCatalogTask(MatchedBaseTask) PatchMatchedPrepTask	**Base:** MatchedCatalogAnalysisTask(CatalogAnalysisBaseTask) PatchMatchedMeasTask		**Base:** MatchedCatalogsAggregationTask(CatalogsAggregationBaseTask) PatchMatchedSummaryTask
Matches sources within patch, multi-band		**Base:** MatchedCatalogMultiTask(MatchedBaseTask) PatchMatchedMultiBandPrepTask	**Base:** MatchedMultiCatalogAnalysisTask (CatalogAnalysisBaseTask) PatchMatchedMultiBandMeasTask		(not implemented??) PatchMatchedMultiBandSummaryTask
Matched sources within tract, single-band		**Base:** MatchedCatalogTractTask (MatchedTractBaseTask) TractMatchedPrepTask	**Base:** MatchedCatalogTractAnalysisTask (CatalogAnalysisBaseTask) TractMatchedMeasTask		**Base:** MatchedCatalogsTractAggregationTask(CatalogsAggregationBaseTask) TractMatchedSummaryTask
Matched sources within tract, multi-band		(not implemented??) TractMatchedMultiPrepTask	(not implemented??) TractMatchedMultiBandMeasTask		(not implemented??) TractMatchedMultiBandSummaryTask
Sources within visit		N/A	**Base:** VisitAnalysisTask (CatalogAnalysisBaseTask) VisitMeasTask		**Base:** VisitAggregationTask VisitSummaryTask
Objects within patch		N/A	**Base:** PatchAnalysisTask(CatalogAnalysisBaseTask) PatchMeasTask		**Base:** PatchAggregationTask(CatalogsAggregationBaseTask) PatchSummaryTask
objects within patch, multi-band		N/A	PatchMultiBandMeasTask		PatchMultiBandSummaryTask
Objects within tract, single-band		N/A	**Base:** TractAnalysisTask (CatalogAnalysisBaseTask) TractMeasTask		(not implemented??) TractSummaryTask
Objects within tract, multi-band	stellar locus width	N/A	**Base:** TractAnalysisMultiFiltTask (TractAnalysisTask) TractMultiBandMeasTask		(not implemented??) TractMultiBandSummaryTask
DIA sources (per-visit?)					
DIA objects (not sure the partitioning, per-patch?)					
Solar System Objects (not sure the partitioning, per-patch?)					
Single-visit image	ghost image control	N/A	VisitImageMeasTask		VisitImageSummaryTask
Coadd image, single-band (per-patch?)	ghost image control	N/A	PatchImageMeasTask		PatchImageSummaryTask
Coadd image, multi-band (per-patch?)		N/A	PatchImageMultiBandMeasTask		PatchImageMultiBandSummaryTask
Template image (subset of coadd image??)					
Injected sources per visit (subset of sources within visit??)	transfer function for individual visits				
Injected sources within patch / tract (subset of objects within patch / tract)	transfer function for coadd				
Injected sources DIA (subset of DIA??)	transfer function for DIA				
Injected sources SSO (subset of SSO??)	transfer function for SSO				

Map (per-dataset?)	coverage map, survey properties	N/A	MapMeasTask (??)		N/A
Database Query (per-dataset?)	random sampling of source or object tables	N/A	QueryMeasTask (??)		N/A
Calibration Products (per-dataset?)	filter bandpass performance				

Anomalies (21 January 2020):

- TractAnalysisMultiFiltTask
- MchCatTractAggTaskConnections
- MatchedCatalogMultiTask, MatchedMultiCatalogAnalysisTask

Intermediate Data Products

Current dataset types:

```
matchedCatalogTract
matchedCatalog
matchedCatalogMulti
```

Questions:

- Are these the intermediate data product names we want? Consistency with class names, etc.

Metric Name Conventions

KPMs should use the short names that are assigned to them in requirements documents (e.g., "PA1" or "GhostAF").

Metrics that are not KPMs should have names that are descriptive of what they are meant to capture. These should be camelCase with leading lower-case. (A made-up example could be something like "colorOutlierFrac.") Note: abbreviated phrases is encouraged for brevity (as with "Frac" in the previous example).

`faro` names each dataset type as "metricvalue_"+`connections.package`+`connections.metric`, where the .package and .metric are defined in the pipeline yaml. The "package" is intended to specify what metrics package from `verify_metrics` contains this metric's definition (e.g., `validate_drp`). To facilitate comparison to these metric definitions, many of the `connections.metric` names contain things like "_design" to specify the level to compare against (i.e., min/design/stretch goals).

Current dataset types:

```
metricvalue_info_nsrcMeasVisit
metricvalue_info_nsrcMeas
metricvalue_validate_drp_AB1_design
metricvalue_validate_drp_PA1
metricvalue_validate_drp_AD2_design
metricvalue_validate_drp_PF1_design_gri
metricvalue_validate_drp_AM3
metricvalue_validate_drp_TE1
metricvalue_validate_drp_TE2
metricvalue_validate_drp_AF2_design
metricvalue_validate_drp_AM2
metricvalue_validate_drp_AD1_design
metricvalue_info_AM1
metricvalue_validate_drp_AM1
metricvalue_validate_drp_PA2_design_gri
metricvalue_validate_drp_AF3_design
metricvalue_validate_drp_AD3_design
metricvalue_validate_drp_AF1_design
metricvalue_Sum_info_nsrcMeas
agg_sum_nsrc_metadata
metricvalue_summary_validate_drp_AB1_design
agg_summary_AB1_design_metadata
metricvalue_Sum_info_nsrcMeasVisit
agg_sum_nsrcVisit_metadata
metricvalue_pipe_analysis_wPerp
agg_sum_nsrc_config
agg_summary_AB1_design_config
agg_sum_nsrcVisit_config
```

Note that for a given dataset type, the measurements in different units of data (e.g., tract, band) are distinguished by their data id. One can find the dimensions of a dataset type as follows:

```
registry.getDatasetType('metricvalue_validate_drp_PA1').dimensions

DimensionGraph({band, instrument, skymap, tract})
```

Questions:

- Do we need the package name in the dataset type name?
- Do we need design/stretch etc. in dataset type name?
- How do we want to distinguish the aggregated results from the "per-unit-data" results?
  - Having a convention for the prefix will be important. For now, we consider "metricvalue_[granular]_" and "metricvalue_summary_". It would be helpful to have distinct prefix for the granular and summary metric values. Not sure what is best name for "granular" metric values.
- How do we want to distinguish the same metric, but different units of data, e.g., patch vs. tract?
- How do we want to indicate instances of the same metric, but with different configurations?

Pipeline Name Conventions

There are three "stages" to calculating metrics with `faro`: generating/compiling the input data, measuring the metric (i.e., "analysis"), and aggregating the measured values. A full pipeline can then chain any number of these steps as needed.

metrics_pipeline_*.yaml: A pipeline consisting of calls to other pipelines (via "imports") that perform the separate stages of metric calculation begins with the phrase `metrics_pipeline`. An example would be `metrics_pipeline_matched.yaml`, which executes `gen_inputs_matched.yaml`, `analysis_matched.yaml`, and `agg_matched.yaml`.

If, as in the above example, the pipeline operates on a specific type of dataset or calculates particular type of metric, this should be made clear in the pipeline name (to the extent possible; in this example, the metrics being calculated are all based on matched visits, so it is named `*_matched.yaml`). *This is done for all types of pipelines so that it is clear which ones may need to be included together in a `metrics_pipeline`.*

gen_inputs_*.yaml: The name of a pipeline that gathers the data to perform measurements on begins with `gen_inputs` (short for "generate" inputs). assembly_*.yaml

analysis_*.yaml: The "analysis" pipelines are the ones that execute the metric measurements (i.e., perform analysis tasks).

agg_*.yaml: "agg" is short for aggregation - these are the pipelines that aggregate values into rolled-up, summary metrics.

Utility function name conventions

Utility function names are all lower case (and snake_case if needed). They should, to the extent possible, explain what the function is designed to return. For example, `phot_repeat.py` is a module containing many photometric repeatability routines. Within this module, the functions have names such as `calc_phot_repeat`.

NOTE: This is not currently true of most of the functions adapted from validate_drp – most of those have camelCase names such as `calcPhotRepeat`.

Notes/Discussion:

What is the purpose of `MatchedCatalogsAnalysis.py` in `metric_pipeline_tasks` ? It doesn't seem to be used.

QUESTIONS:

- The three steps of metric measurement are assembly analysis aggregation. (We're proposing "assembly" instead of "gen_inputs".)
  - A: Proposal is to use: **Preparation, Measurement, Summary**
- Change Multi XBand ("cross-band"; i.e., multiple filters)?
  - A: E.g. AB2. Proposal is to use **MultiBand**
- In dataset type definitions, do we need the metrics package name? [Are all metrics required to have a definition in a .yaml metrics definition file somewhere (as most currently do in `verify_metrics`)?]
  - A: Yes, we would like to retain this convention but **we want to** update the metrics package name to be pointing more towards function than implementation (e.g., associated with a requirements document)
  - e.g validate_drp.yaml srd_performance.yaml. And separate out the non-normative metrics in validate_drp.yaml into (say) dm_metrics.yaml
- In dataset type definitions, do we need design/stretch, etc.? [This typically won't change *how* the metric is measured. Could this instead be packaged in the Measurement object itself?]
  - A: We would like to avoid having the specification details in the dataset type name. Note that there are some requirements that have different thresholds for minimum, design, and stretch, and hence the measurement is done in a different way.
- In dataset type definitions, what conventions do we want for the prefix? For the summary metrics?
  - A:

Progress tracking:

- [x] lsst/faro/base/
- [x] lsst/faro/measurement/

- [x] lsst/faro/preparation/
- [ ] lsst/faro/scripts/
- [x] lsst/faro/summary/
- [ ] lsst/faro/utils/
- [ ] pipelines

Questions/discussion:

- The file `lsst/faro/measurement/GeneralMeasureTasks.py` contains some classes that are Summary tasks rather than Measurement tasks. Propose that we create a new file in the summary directory to hold these general summary tasks, `lsst/faro/measurement/GeneralSummaryTasks.py`.
- In `lsst/faro/measurement/GeneralMeasureTasks.py`, `NumpyAggTask` names the output `Measurement` with `metricvalue_aggname_package_metric`. Do we want this naming behavior? (Is this task used anywhere? If not, maybe it should be? It seems to be a generally useful task.)
- Not sure what to do here: we moved the “Analysis” tasks (e.g., `MatchedCatalogsAnalysis.py`) into `lsst/faro/preparation` . Our proposal was to rename “Analysis” to “Meas”. But we already have `MatchedCatalogMeasureTasks.py` in `lsst/faro/measurement` (it’s the one that contains, e.g., `PA1Task`.) This potential naming ambiguity will be confusing. Should we rename the one that defines the particular KPM measurements (in measurements)? We could call it `MatchedCatalogMetricTasks` instead?
    - I moved all of the "Analysis" tasks to `lsst/faro/measurement`, and renamed them to things like `PatchMeasurement.py`. Files containing individual metrics are now called `PatchMetricMeasureTasks.py` (for example).