LARGE SYNOPTIC SURVEY TELESCOPE

Large Synoptic Survey Telescope (LSST)

Data Access Use Cases

Tim Jenness, Jim Bosch, Michelle Gower, Simon Krughoff, Russell Owen, Pim Schellart, Brian van Klaveren, Dominique Boutigny

LDM-592

Latest Revision: 2018-02-14

Abstract

Use Cases written by the Butler Working Group covering data discovery, data storage, and data retrieval.

Change Record

Version	Date	Description	Owner name
	2017-11-13	Initial version from Working Group [LDM-563] internal spreadsheet.	T. Jenness
1.0	2018-02-14	Approved in RFC-413	T. Jenness

Document source location: `https://github.com/lsst/LDM-592`

Contents

Data Access Use Cases

1 Introduction

The Butler Working Group [LDM-563] was convened in August 2017 to work on requirements and design of the Butler. As part of the process, Use Cases were written to anchor the requirements. In this document we present those Use Cases, grouped by the originating team, with labels that allow the requirements in LDM-556 to reference them.

2 Actors

The use cases make use of a standard set of Actors.

2.1 General

Observer Observatory Operations (e.g., summit, base)

Astronomer End User with data rights to releases

Data Release Board Board responsible for the quality of data releases, including approval of software versions and flagging of data quality.

2.2 Staff Scientist or Staff Developer

Data Facility Scientist Data Facility Scientist

Pipelines Developer Science Operations Scientist (algorithm developer)

Validation Scientist Science Operations Scientist (long term performance)

Validation Scientist Collaborator Collaborator with Science OPS (non-LSST funded)

Developer Non-science LSST-funded developer (e.g., Qserv, Batch Processing Service, LSP)

Batch Production Operator Production Operator for DRP (has additional privileges)

Commissioning Scientist Scientist involved in commissioning (has additional privileges)

2.3 Software Agents

SuperTask Software agent that takes input files, processes them, and supplies output files.

CI System Software agent that runs tests (including integration tests that may involve pipeline processing) on a regular cadence

Batch Control System Software agent that manages SuperTask execution running in the batch production environment

Harvesting Agent Scans the file system when a job completes and stores the resulting files in the Data Backbone.

DMCS Data Management Control System

3 Alert Production

3.1 *AP1*: Prompt processing: process raw images and generate alerts

a. DMCS provides a ”next visit” signal to the prompt processing system running at NCSA.

b. The prompt processing system launches the alert production SuperTask (one instance per CCD) passing it the CCD ID, as well as the VisitInfo.

c. The alert production SuperTask computes the desired criteria for calibration products, position and photometric reference catalogs, and image templates and loads them via the Butler. It then places a load request for the first of the two crosstalk-corrected snap images, and blocks waiting for it to arrive. After the first snap arrives, the task may perform some image processing (e.g. ISR) on that before requesting the second image and blocking again. To get adequate performance this may require the ability for SuperTasks to cooperatively run in parallel, e.g. so one can wait for the second snap while another processes the first snap.

d. The alert generation task (one instance per node) processes the second crosstalk-corrected snap image and combines the two snaps into one exposure. Steps include:

- Perform instrument signature removal on the second snap.
- Detect cosmic rays by comparing the two snaps.
- Combine the two snaps into a single exposure.

- Peform single-frame calibration: determine a PSF, mask additional cosmic rays, if any, refine the WCS based on the position reference catalog and determine a photometric zero point based on the photometric reference catalog.
- Save the calibrated exposure and detected source catalog.
- Subtract the calibrated image from the template to produce a difference image.
- Save the difference image.
- Detect DIA Sources on the difference image.
- Save the DIA Source catalog.

e. The alert production SuperTask gets DIA Objects from the L1 database via the Butler, performing a spatial query and blocking on the result. If this step is slow, it have to be done as a parallel SuperTask that is started while image processing is being done and run in the background. However, it should be done as late as possible in order to give time for the DIA Objects from the previous field to be inserted into the L1 database (though this only matters when two consecutive fields overlap).

f. The alert production SuperTask associates DIA Objects with DIA Sources

g. The alert production SuperTask writes new DIA Sources and new or modified DIA Objects (one per DIA Source) to the L1 database. Note that the modified DIA Objects will be saved as new DIA Objects.

h. The alert generation task issues an alert for each new or modified DIA Object. This will be done via the Butler talking to the alert broker, unless a more direct path is needed for performance.

3.2 *AP1.catchup*: Catch up processing: process images during the day and do not issue alerts

a. This is a minor variant of **AP1**. Some raw images may have to be processed during the day (all will be processed that way when the telescope first goes on sky, later this may be needed if normal prompt processing fails during the night)

The primary differences from **AP1** are:

- The job is managed differently; it will be triggered manually by a Batch Production Operator.
- No alerts will be broadcast.

3.3 *AP1.dev*: A Staff Scientist runs the alert generation task

a. This is the standard variant of any automated task. A user is a Staff Scientist who wants to run all or some of the task, in order to diagnose a problem, try out a new configuration, or try out new code.

b. The user obtains the required input data.

c. The user obtains the original output against which the new output is to be compared.

d. The user launches the appropriate portion of the alert generation task.

e. The alert generation task executes as per AP1, reading input data, saving output data to the specified output repository and (if requested) issuing alerts to a test alert broker.

f. The user compares the new output to the original output.

3.4 *AP2*: A Staff Scientist processes images

a. The user, a Staff Scientist, specifies a visit to be processed.

b. The butler expands the DatasetExpresion to select only those CCDs for which raw image data is available.

c. The task processes each image.

d. The task persists the computed data (`calexp`, etc.)

Note: this use case is intended to highlight the need for the butler to be able to expand a DatasetExpresion. This expansion only relies on the Butler knowing which datasets exist that match a particular incomplete ID. It has nothing to do with searching for data that meets criteria such as PSF size (presumably a different use case).

3.5 *AP3*: A Staff Developer adds a new SuperTask

a. The user, a Staff Developer, writes the new SuperTask and provides a suitable new unique name for task configuration and metadata persistence

b. Because the new butler supports dataset prototypes the developer does NOT have to edit any obs package to add dataset types for this task's configuration and metadata.

Instead the new task uses the standard dataset prototypes for configuration and metadata to persist data.

Note: dataset prototypes also simplify repository configuration and provide flexibility to tasks to save new dataset types (e.g. a one-off QA task could write a new kind of graph without modifying any obs packages).

3.6 *AP4*: Persistence from alert generation pipelines

a. A Pipelines Developer is writing the task that packages up DIAObjects into alerts.

b. They want to publish the alerts to the alert distribution system, but they also want to preserve the difference images and DIASource measurements resulting from those difference images.

4 Architecture

4.1 *ARCH1*: HDF5 Data Products

a. A Developer wants to read in some raw data files in FITS format and do test processing runs writing out FITS, in one case, and HDF5 in the other, in order to compare performance differences. This will include creation of images and catalogs in both formats.

b. From the test runs the Developer reads in the FITS coadd and the HDF5 coadd and compares the Python objects to ensure that the answers were identical.

c. Once the outputs have been validated in HDF5 format, a mini production run is scheduled using HDF5 outputs to validate large scale performance.

4.2 *ARCH2*: Same dataset type, multiple format outputs

a. A Developer wants to confirm that HDF5 and FITS files produced by a task have identical content. Rather than duplicate processing, the task is configured to write each output dataset in HDF5 and in FITS format.

b. Once processing is complete, the Developer runs some code that that reads in both forms of the output for a single dataset type. The resultant Python objects from each input format are compared to confirm that they contain the same information.

4.3 *ARCH3*: Batch processing data with EFD and updated WCS and visit metadata

a. A Science Validation Scientist wants to process all the visits taken in the u filter from an engineering night last month, using a bespoke processing pipeline. The processing will need a number of "high-resolution" telemetry feeds from the EFD and a fully-populated data header for each visit, using updated values, including an updated WCS from the L1 system.

b. They submit their request to the batch system (either directly or through an intermediary) and wait for the results.

c. The workflow system ensures that the correct visit metadata and EFD subset is available to the batch jobs during pre-flight.

d. The jobs complete and the Astronomer accesses the output files and reads them into a notebook for visualization.

4.4 *ARCH4*: Submitting batch jobs via a notebook

a. A Scientist submits a processing request from a notebook to the batch processing system that will result in multiple jobs being executed.

b. They know that each job will have created some output data in a different location but when they are informed that the jobs have been completed they are presented with a single data repository containing all the results.

4.5 *ARCH5*: Provenance in downloaded files

a. An Astronomer downloads a coadd from the Data Access Center portal that comes with full metadata. They want to see how the current version of the coadd has changed compared to the version they are using, which came from an older data release

b. They read the data description document for this release and determine where the provenance is located in the file.

c. They read the two file headers into their Python program and they determine that this coadd did not include one visit that was included in the previous data release.

d. They look up the PVI associated with that visit and download it from the previous data release and compare it with the PVI from the current release.

5 Commissioning

5.1 *COMM1*: Low latency Q/A

a. A Commissioning Scientist wants to analyze commissioning data, including calibration, engineering and astronomical observations shortly after the data being taken to fine tune engineering configuration, solve data acquisition problems, identify software issues. This includes rendezvousing engineering and science data in essentially "real time" with latency of order seconds.

5.2 *COMM2*: High latency Q/A with EFD

a. The Commissioning Scientists will need to do analysis of commissiong data at the commissioning area of the Data Facility with access to the engineering facilities database (EFD) on latencies of days to weeks.

5.3 *COMM3*: Value added repositories

a. Commissioning scientists will create custom repositories containing specific commissioning data.

b. They will then do additional analysis that will add value to the raw data: e.g. source catalogs matched to telemetry, further algorithmic outputs, and also technical reports.

c. These value added repositories then need to be stored in persistence storage at the Data Facility and potentially available to Science Validation Collaborators.

d. Since provenance will be sufficient to reproduce the datasets in the value added repositories, long term persistence is not mandatory and backups need not be performed. The collaborators need a sufficiently long retention time to be able to examine the contents of the value added repositories that reside at the Data Facility.

5.4 *COMM4*: Comparison of the night's data with archived data

a. Commissioning Scientists will want to compare an analysis carried out on the current night's data to data taken at a previous time separated by many months. This means accessing both archived and too-new-to-be-archived data in the same environment.

5.5 *COMM5*: Lightcurve Provenance

a. Commissioning scientists will need to do analysis of lightcurve provenance. That is, given a set of associated source measurements for a particular set of objects, match up each epoch to the observation metadata and any relevant telemetry from the EFD to look for correlations between observational conditions, engineering information, and features in the lightcurves. This could also include tracking the version of the stack used to produce each epoch.

5.6 *COMM6*: Batch processing of commissioning pipelines

a. The Commissioning Scientists will want to build pipelines that are not exactly AP or DRP pipelines, but that will behave similarly in the sense that they will be runnable in the same batch system.

5.7 *COMM7*: Compare processing based on selection in observing or EFD metadata

a. The Commissioning Scientists will want to compare how the system behaves as a function of various quantities. An example is looking at coadds from the same part of the sky in bright sky conditions versus in dark sky conditions.

5.8 *COMM8*: Bulk reprocessing of commissioning pipelines

a. At regular intervals (perhaps coincident with major software releases), the Commissioning Scientists will want to run some set of the commissioning pipelines on all the data taken so far. This will include rerunning all verification analysis scripts on verification data to monitor for regressions.

5.9 *COMM9*: External datasets

a. The Commissioning Scientists will need access to external reference datasets to compare against. For example, compare the width of the stellar locus from PS1 to the width from LSST processing of similar imaging data.

5.10 *COMM10*: Notebook portabiliity

a. The Commissioning Scientists would like to run exactly the same notebook on exactly the same data as in **COMM4**, but this time at the archive center.

5.11 *COMM11*: Simulated images and catalogs

a. Commissioning Scientists will want to compare analyisis on commissioning data to the same analysis on simulated images or catalogs.

5.12 *COMM12*: On the fly processing of engineering data

a. Commisssioning scientists will need to spawn on-the-fly processing of engineering data (e.g. focus sweeps) that will potentially feed back to the system configuration in a relatively tight loop. This processing may require both one off investigations or batch style processing. In any case, these activities will have to happen on the commissioning clusters.

5.13 *COMM13*: Calibration products pipelines

a. The Commissioning Scientists want to spawn computation of master calibration products on many different cadences: during the night on the order of minutes, daily, monthly, per data release. This will require these pipelines to run both on the commissioining cluster and the archive clusters. There will be a SuperTask that computes these master calibration frames and will be spun up in both contexts.

6 Continuous Integration

6.1 *CI1*: Historical Reductions

a. A Validation Scientist knows from experience that we will never have tests for all the possible regressions. They have worked with the SQuaRE team to set up continuous integration runs on significant (TB) amounts of data on weekly cadences. They want this to be done in the CI system. This provides a set of runs that can be inspected for the source of regressions even if the particular regression has no specific metric registered in SQuaSH. The input data needs to come from a shared source since this will likely be done on cluster resources.

b. The outputs need to be stored in a stable, curated, shared, findable location with provenance so they can be found again.

c. The input data needs to be disaster recoverable, but the outputs (strictly speaking) can be re-generated.

6.2 *CI2*: Export execution metadata to external apps

a. In the CI system the running code will be instrumented to produce measurements of various metrics: both technical, like runtime, and more algorithm related, like number of sources. The CI system packages measurements of coded metrics from each of the potentially parallel executions of the CI pipelines.

b. This information will be exported from the CI system and ingested into a system designed to track and discover regressions and other changes in the behavior of the system.

6.3 *CI3*: CI in cloudy environments

a. The Batch Production Operator wants to run multiple builds and CI runs in multiple VMs within the same cluster.

b. They want to be as efficient as possible so only allocate as many resources as absolutely needed for each run. The CI datasets are of order 1TB, so the Batch Production Operator would like to keep the amount of data replicated to local disk as small as possible. This leads to a desire for the input repositories to reside in a storage format that plays well in the cloud ecosystem.

7 Data Access (DAX)

7.1 *DAX1*: Pre-caching data

a. See **DRP16**

7.2 *DAX2*: Shared local cache of remote data

a. See **DRP17**

7.3 *DAX3*: Load size

a. See **DRP18**

7.4 *DAX5*: Efficient caching of dataset cutouts

a. Astronomers at a meeting are all analyzing the same part of a shared dataset.

b. Hundreds of repeated requests for a subset of the same PVI are sent from these Firefly sessions.

c. The DAX server notices the repeated accesses to the same PVI and uses a data access scheme that minimizes reads.

7.5 *DAX6*: Retrieve raw data from DAC

a. An Astronomer requests a raw data file from the DAC using VO protocols.

b. The Server identifies the relevant file and retrieves it from the Data Backbone. It also queries the visit metadata table and transformed EFD to retrieve the most up to date and correct values for the visit.

c. These values are integrated into the data header and a single file with update metadata is created.

d. The file is returned to the Astronomer.

7.6 *DAX7*: Caching of data supersets or subsets

a. Many users are requesting images, potentially both stitched and postage stamps, from a common region of the sky.

b. In order to reduce reading of network disks and provide low latency performance, DAX services will cache images, potentially dynamically adjusting the size of cached data.

c. Involved in this is the augmentation of headers of the returned images, which is a standard DAX function. The DAX services may need to persist data structures optimized for this type of caching to avoid having to repeat the header augmentation phase.

7.7 *DAX8*: Repository migration and repository versioning

a. A Staff Scientist has a data collection that is too old to be read by the current version of the software.

b. The software they are using reports the version number and suggests they run the migration script.

c. Once the migration script completes the data collection can be read.

d. They have another collection that is from the previous version, but one which is still supported by the current library, and a warning is issued that they may want to migrate to make use of new features.

7.8 *DAX9*: Using the Butler with VO retrieved data originating from butler repositories

a. An Astronomer downloads images from the LSST DAC using Virtual Observatory services.

b. The images may end up in a single directory with no structure. These images might also include composites as separate files.

c. The Astronomer wishes to load these images into their analysis code using the Butler.

8 Data Facility

The bulk of these use cases were sourced from the LSST Data Facility team. Use cases numbered 101 and above were sourced from IN2P3.

8.1 *LDF1*: Submit a processing run to the batch processing service

a. A Production Operator wants to generate coadds for a tract of sky for all good data for a particular filter. They decide on the version of the software to use, the configuration parameters and the specific SuperTasks to execute. The input files that are required for the job to execute must be determined (e.g., using a normal sql query or they query the SuperTasks with the parameters)

   (alternate) During pre-flight, expected output filenames (and paths within compute job) must be determined (e.g., using operator configuration or querying the SuperTasks with the parameters)

   (error) One of the files reported as being an expected output file has a name that is already known to the data backbone. Files in the data backbone should be unique so an exception is raised and the operator must check the configuration that was used by the supertask.

b. The relevant files may be pre-staged to a filesystem closer to the compute site. The job is then submitted to the batch system. The batch system job transfers those files to the processing node and starts the SuperTask, configuring it with the output from the intial query of the SuperTask.

   (error) The SuperTask attempts to read a file that was not included in the initial list created during pre-flight. The SuperTask can only see the local disk and can not retrieve any files from the data backbone during processing. This is a fatal error and the processing job terminates.

   (error) The SuperTask should not be allowed to overwrite an existing output file when running inside the batch processing service. This should have been caught as an error during pre-flight, so if it happens during runtime it is an unexpected error.

c. When the job completes, the expected output files that have been configured to be saved to the job's file system, are either stored in a staging area closer to the compute site and then to the data backbone or directly back to the data backbone. At operator's request unexpected output files may be saved as a junk tarball which is also brough back to the data backbone in the same manner.

d. If an expected file is not found, the job reports an error condition and whatever output files will be brought back to the data backbone (same manner as success).

e. Metadata about the files is stored along with the actual files in the data backbone.

8.2 *LDF2*: Calibration Selection

a. A Validation Scientist wants to reprocess some raw data using a variety of calibration files. A configuration file is edited to specify override values for the flatfield files to use for each raw data file. The relevant SuperTask is then given this configuration file and list of input files and returns a complete list of files required for processing. Processing continues as for OPS1.

b. The results from these jobs are consolidated onto a single QA report.

8.3 *LDF3*: Investigations

a. During a batch production it is noticed that some outputs look wrong. A Validation Scientist wants to investigate further. A job execution database is queried to retrieve the job provenance information. The relevant input files, software and configuration files are then downloaded to a local system. The reduction process is configured to persist intermediate files to allow detailed examination of all steps.

b. The software is then executed locally, with all the input files in a human-managed directory structure (more than likely single directory), and the outputs are written locally.

c. Each intermediate is then displayed in turn by the scientist until the particular step is located that generates the bad outputs.

d. The software is then reconfigured to just run the step generating the bad output. The reduction process is run multiple times using the output from the last good step with many different configurations and software versions. When the problem is diagnosed, a report is written that may also include either a fix being committed or a bug report filed.

e. When done with the tests, the Validation Scientist may want to clean up all the test output files.

8.4 *LDF4*: Qserv ingest

a. As data arrives from batch processing, the resultant catalog files are periodically located for a particular region of the sky and are bulk uploaded into a Qserv instance.

8.5 *LDF5*: Alert Processing

a. The *next visit* signal comes from the OCS telling the prompt processing to retrieve the templates and catalog files that will be needed to handle the next visit alert processing.

b. Crosstalk-corrected data are transferred from the summit to the prompt processing system running at NCSA. The data for each CCD is transferred to a specific node along with the templates and catalog files and the job is triggered.

c. The processed visit images are written to disk and the alerts are issued. The PVIs are transferred to the 30 day cache and forwarded on to EPO.

d. The output catalogs are then stored into the L1 database.

8.6 *LDF6*: Processing data from that night at the base

a. As data are being taken at the summit site, the Commissioning Scientist wishes to do bespoke analyses as quickly as possible. They have some Jupyter notebooks running in an instance of the LSP at the Base Facility. They make some minor modifications to the notebook and run it on a set of visits taken within the previous half hour.

b. Based on the results of this analysis they discover an anomaly and decide to plot the calculated seeing quality from their reduction against some environmental parameters read from the EFD.

c. The next morning they want to do a more extensive analysis of the night's data, they log into the LSP at the archive center and run the same notebook, submitting batch jobs for much of the data from the night before. They do not edit this notebook to indicate that the EFD data or raw data maybe coming from a different location.

8.7 *LDF7*: Dataset deletion

a. A developer has a local repository from which they want to permanently delete a dataset. They run a program which deletes the file(s) from disk and updates the local repository

such that the dataset is no longer discoverable.

8.8 *LDF101*: Reprocessing a sky patch with bad observations removed

a. A Staff Scientist has examined a coadd from a mini-DRP run and has determined that one of the observations is bad in some way and should not be included in future processing jobs.

b. The Scientist re-reduces the coadd in the development area to ensure that the coadd looks good without the bad observation. The inputs to the processing run are determined automatically based on the tract specification with the bad observation explicitly flagged to not be included.

c. The coadd looks much better without the observation. The Scientist reports that this observation is suspect, using a standard tool and providing supporting evidence. The Data Release Board examines the report, agrees with the assessment, and sets the appropriate status in the public system indicating that this observation should no longer be included in coadds. This information is known to all processing centres and can be queried by anyone with data rights since this may affect how people treat previously issued L1 data products.

d. When the bad observation is flagged, the same flagging process can be told to flag all data products that have contributions from that observation.

e. A Production Operator scans the list of contaminated products and triggers jobs to reprocess them using the updated list of inputs. Bad observations are automatically dropped from the list of inputs.

8.9 *LDF102*: Retrieve all data for a coadd patch

a. An Astronomer wants to reprocess a coadd from a Data Release with all available data. They first ensure they can reproduce the coadd from the data release by querying the DAC for all the data files that were used in the original data release. They then enable inclusion of bad observations, and retrieve all the raw data that could go into the coadd.

8.10 *LDF103*: Download a subset of data

a. An Astronomer running in a Jupyter notebook in the LSP wants to reprocess a significant amount of data. They have done a query from their notebook to determine all the relevant files and wish to download those files to their home institution.

b. They submit a batch job to store those files in their LSP workspace and when they are notified the files are available they mount the workspace on their local system and retrieve them.

8.11 *LDF104*: Same processing at different sites

a. In order to test that Data Release Production systems are running properly, a Production Operator at NCSA triggers identical processing jobs at multiple sites (e.g., NCSA, France).

b. When all the jobs are completed, the output products from all jobs are in the data backbone. The Production Operator then triggers a QA job to perform a statistical analysis on both sets of output products to determine whether they are scientifically equivalent (within some threshhold). (The output products may either be pre-downloaded from the data backbone or where appropriate can be downloaded from inside the QA job.)

9 Data Release Processing

9.1 *DRP1*: SuperTask Pre-flight

a. SuperTask activator code obtains a graph representation of the datasets and units of data that a Pipeline may use as inputs and/or outputs. Must account for the availability of inputs in the data repository, the relationships between units of data, and user-provided filters on units of data to consider. This includes identifying the appropriate calibration dataset to use with a given raw dataset, and looking up raw data that overlap sky patches or vice versa.

   This is a lower-level use case describing an activity that is a part of (at least) **SCIVAL1** and **SCIVAL2**.

9.2 *DRP2*: SuperTask Input

a. A SuperTask retrieves a Python object from a stored dataset, given a handle object provided in pre-flight.

b. Like **DRP2**a, but the dataset is a predefined small part of a larger dataset (such as the PSF of a calibrated exposure). This could be supported by having the larger dataset stored as multiple smaller files.

c. Like **DRP2**a, but the dataset is composed of multiple files, and some files may contain components that should override those in other files. An example of this could be a WCS solution that should then be applied to a raw data file.

d. Like **DRP2**c, but the dataset also contains the results of a query to the EFD (or some alternate representation). This is required for calibration products production.

e. Like **DRP2**a, but the dataset is a flexibly-defined subset of a larger dataset (such as a rectangular subset of an image, or a set of rows in a catalog) that can nevertheless be read more efficiently than the full dataset. This can probably not be supported by splitting the dataset into multiple files.

9.3 *DRP3*: SuperTask Output

a. A SuperTask sends an in-memory Python object to a data repository to be stored as one or more complete files.

This is a lower-level use case describing an activity that is a part of (at least) **SCIVAL1** and **SCIVAL2**.

9.4 *DRP4*: SuperTask Intermediate Elision

a. Like **DRP3** followed by **DRP2**abc, but actual storage is elided for performance for certain datasets that were identified when the Pipeline and/or workflow system was configured.

This is a lower-level use case describing an activity that is a part of (at least) **SCIVAL1** and **SCIVAL2**.

9.5 *DRP5*: SuperTask Input Staging

a. After pre-flight but before at least some SuperTask execution, the Batch Control System transfers all inputs needed by one or more SuperTask executions to the compute resources where the execution will take place.

   This is a lower-level use case describing an activity that is a part of **SCIVAL2** and (possibly, depending on architecture) **SCIVAL1**.

9.6 *DRP6*: SuperTask Output Staging

a. After at least some SuperTask execution, the Batch Control System transfers output datasets from temporary storage on the compute resources where they were produced to permanent storage.

   This is a lower-level use case describing an activity that is a part of **SCIVAL2** and (possibly, depending on architecture) **SCIVAL1**.

9.7 *DRP7*: Analysis Metadata Query

a. A Staff Scientist or Astronomer obtains a list of DataIDs and existing datasets that match an expression within a given output data repository. This includes identifying the appropriate calibration dataset to use with a given raw dataset, and identifying raw data and sky patches from each other based on location.

   This is a lower-level use case describing an activity that is a part of **SCIVAL3** and **SCIVAL4**.

9.8 *DRP8*: Analysis Dataset Input

a. A Staff Scientist or Astronomer obtains an in-memory Python object from a data repository, providing either a DataID or an expression that reduces to a unique DataID.

   This is a lower-level use case describing an activity that is a part of **SCIVAL3** and **SCIVAL4**.

b. Like **DRP8**a, but the dataset is predefined small part of a larger dataset (such as the PSF of a calibrated exposure). This could be supported by having the larger dataset stored as multiple smaller files.

c. Like **DRP8**a, but the dataset is composed of multiple files, and some files may contain components that should override those in other files.

d. Like **DRP8**c, but the dataset also contains the results of a query to the EFD (or some alternate representation). May be avoidable if all relevant queries can be anticipated and run during raw data ingest.

e. Like **DRP8**a, but the dataset is a flexibly-defined subset of a larger dataset (such as a rectangular subset of an image, or a set of rows in a catalog) that can nevertheless be read more efficiently than the full dataset. This can probably not be supported by splitting the dataset into multiple files.

9.9 *DRP10*: LSST Raw Data Ingest

a. Raw data from LSST is ingested into a custom data repository (i.e. not the Data Backbone).

9.10 *DRP11*: External Data Ingest

a. A Staff Scientist or Astronomer adds non-LSST images or catalogs to a data repository, which may include raw images or processed images and catalogs. This information should then be served in a consistent fashion (e.g. get data for same region on the sky for joint processing with Euclid and WFIRST). When the new data are raw images from an instrument for which an LSST `obs` package has been defined, all SCIVAL use cases must be supported on these data.

   This is required for at least some precursor datasets (HSC, DECam, CFHT) for verification and validation prior to first light.

9.11 *DRP12*: Dataset Type Registration

a. SuperTask control code registers a new dataset type with a data repository, allowing instances of that dataset type to be added to the repository in the future.

   This is a low-level use case describing an activity that is a part of **SCIVAL1**d.

9.12 *DRP13*: Skymap Registration

a. A Staff Scientist adds a new tiling of the sky defined by a configuration file to an output data repository, allowing the tracts and patches it defines to be used to identify datasets.

   This is a low-level use case describing an activity that is a part of **SCIVAL1**c.

9.13 *DRP14*: Processing-Output Metadata Update / Ingest

a. SuperTask preflight code (as in **DRP1**) filters images and relates them to sky patches using metadata (such as PSF size and WCS) generated by processing steps that have not been run in the given data repository. While the subsequent processing will start from raw data and will naturally redetermine (and possibly improve upon) those metadata values, estimates of these values from nightly alert processing are both necessary and sufficient for creating an efficient plan for staging data and executing Pipelines that extend to coaddition.

   When values from nightly processing are not available (e.g. in construction, or when testing a new deployment of the operations system), a Staff Scientist sets (or resets) the metadata values from an initial manually-invoked processing run that does not extend to coaddition.

9.14 *DRP15*: Master Calibration Products Registration

a. A Staff Scientist produces a set of master calibration products via a set of processing runs runs (possibly in a different environment) and adds them to a data repository for use in other processing runs. Old master calibration products should remain available, but one or more ”preferred” labels may need to be transferred to the new products. Each set of master calibration products includes data products with (date or visit) validity ranges; selecting the set of calibrations to be used for a processing run is distinct from selecting the data product within that set to use for a particular observation. When the data repository is part of the operations system, this is done by a Batch Production Operator, not arbitrary Staff Scientists.

9.15 *DRP16*: Pre-cache data

a. A Staff Scientist or Astronomer is going to a conference and wants to pre-cache some data from a remote repository, to be able to access through the butler while without network connectivity.

9.16 *DRP17*: Shared cache

a. A Staff Scientist or Astronomer and her colleagues want to access the same data (or overlapping) from a remote repository.

b. The first time someone downloads the data it is stored in a shared local cache.

c. Subsequent retrieval requests from others sharing the on-site proxy can retrieve the same data locally.

9.17 *DRP18*: Load size

a. A Staff Scientist or Astronomer (or the task they are running) wants to be able to access a dataset that is too large to fit in memory as a whole.

b. When requesting the data they specify a smaller region of the image and/or restrict the retrieval to only the parts of the composite that they really need.

9.18 *DRP22*: Transactions

a. A SuperTask or the Batch Control System needs to write a dataset to a repository which is backed by multiple media (e.g., part goes to file, part to a database).

b. The system waits for all write operations to complete.

c. One of the writes fails and an attempt is made to undo the writes that did succeed. The SuperTask is told whether everything was written, or nothing was written, but does not need to understand the case where some of the data was written.

9.19 *DRP23*: User interactions

a. An Astronomer wants to build a query interactively (e.g., using tab-complete or some other predictive help that knows about the dataflow).

9.20 *DRP26*: Compression

a. When running a Pipeline in a certain environment, a developer or operator wants to configure a particular output dataset to be written with compression. This may be implemented by adding a new dataset type and (trivial) SuperTask to write the compressed version (as long as some sort of alias feature allows consuming code to read either the compressed or uncompressed version), or by changing Butler configuration.

9.21 *DRP27*: Iterating over Repository content

a. An Astronomer wants to loop over all Datasets of a particular DatasetType in a repository (e.g. to run jointcal and then make plots for all output).

9.22 *DRP28*: Backwards compatibility

a. A Staff Scientist or Astronomer has existing code (written to use `butler.get()` and `butler.put()` calls) that they wish to use with no (or minimal) changes.

9.23 *DRP29*: Multi-Repository File Removal

a. A shared storage space used for collaborative algorithm development and testing is filling up, and developers respond by removing processing outputs that are not being used anymore. Some of the processing runs to be deleted were based on inputs produced in other processing runs that should not be deleted, and those inputs are hence considered to be part of both the to-be-deleted data repositories and the to-be-kept data repositories. After removing the to-be-deleted data repositories, the files that are no longer used by any repository are identified and deleted, while those in the to-be-kept repositories are not.

9.24 *DRP30*: Forced deletion of datasets

a. One or more data products are deleted by a Production Operator or other authorized actor, without regard for which data repositories or other users may refer to them. In some contexts this may not remove all record of these datasets held outside their primary storage locations, as long as the storage cost of those records is negligible.

9.25 *DRP31*: Compare with previous data releases

a. A new data release is prepared by the Production Operators.

b. The Validation Scientist analyzes a subset of the new data release by directly comparing it with the current Data Release.

c. A statistical analysis is done with the catalogs derived from both data releases to determine if the new data release is suitable for release.

9.26 *DRP32*: Use of rerun collections

a. A Developer runs some processing starting from outputs produced by the Batch Production Operators or another Developer, where those outputs are specified as inputs using a label given as a single string (possibly with path-like namespace components). They specify a new label to use for the output data location. When the processing completes, all of the contents of the inputs (or at least all of the items actually used as inputs) will appear to be present in the output collection.

b. The developer then wants to run additional processing with both inputs and outputs coming from what was previously just the output collection.

10 Science Quality and Reliability Engineering

10.1 *SQR1*: QA Drilldown

a. A Science Validation Collaborator wants to use the Notebooks Aspect of the Science Platform to do some analysis on data that has been processed using the LSST pipelines. They know how to find a difference image that did not turn out well, but do not know if it was the template or science image that caused the problem. They want an easy way to iterate over just the data relevant to the difference image for every dataset type that went into the particular run to try to figure out what data were the problem.

b. Retrieve identified data into local filesystem visible to the notebook.

c. Rebuild intermediates.

d. Iterate through inputs, display them, and look for bad data.

10.2 *SQR1.5*: Improve configs

a. The Science Validation Collaborator finds a particular calibrated exposure that messed up a coadd (from **SQR1**) They then want to rerun the relevant steps that resulted in the erroneous calibrated exposure to see if they can find a configuration that would create a better calibrated exposure: repeatedly run on the local data with various configs.

b. Examine outputs for each configuration

c. Push corrected configuration into the production config.

10.3 *SQR2*: Data discovery based on observation/processing metadata

a. A Science Validation Scientist wants to find all the catalogs and exposures taken in really good seeing and those in moderate seeing in order to compare the ability to estimate the point spread function as it tends toward the undersampled regime.

   Discover data based on e.g. seeing cuts. This includes image data as well as catalogs produced during processing.

b. Stage discovered data to a shared distributed filesystem where they can be examined further.

10.4 *SQR4*: Job/Metric/Measurement persistence

a. A developer instruments an image characterization Task in order to measure a specific set of metrics about the Task.

   The developer wants to publish this set of metrics to SQuaSH, using a descriptive name to simplify retrieval of these results later.

10.5 *SQR5*: Named queries as datasets

a. A Science Validation Scientist would like to include a test on a specific dataset. The test includes finding all point like sources at a certain signal to noise with a bright and faint flux cut. The scientist produces a query that will retrieve these sources for the dataset in question, and they would like to refer to the query by name as opposed re-writing the query every time. They would also like to be able to refer to the same query by the same name in other contexts: i.e., on different, but similar datasets.

10.6 *SQR6*: Named blobs as datasets

a. The Science Validation Scientist is producing a new measurement they would like to track in SQuaSH. An intermediate product of calculating the measurement is a plot of magnitude residual vs. magnitude with some bounds. They would like to persist this so that it is easily retrieveable as a blob later on.

10.7 *SQR7*: Multiple repositories

a. A Commissioning Scientist would like to compare results from reducing the same datasets using two different configurations of the algorithms. They would like to easily access the same data (e.g., calibrated exposures, source catalogs) from within a python interpreter so they can determine how the change in configuration affects the output.

10.8 *SQR9*: DRP Canary Data

a. *Context agnostic mechanism to refer to multiple data releases in the same session.*

   Science Validation Scientists will identify data to use as a "canary" test between releases. They want the size of the data will be manageable, but will provide enough scope to uncover major problems. The CI System will need to access the same data products from multiple data releases in the same environment for comparison.

   This also implies that the CI system needs some way to refer to releases (and other coherent repositories) by name rather than by path since the location of the repositories is not the same for all contexts.

b. Possibly a duplicate of **SQR1**a – retrieve a coherent set of data given a fully qualified descriptor for a particular data product(s)

10.9 *SQR10*: Efficient notebook execution

a. Given a notebook, identify necessary inputs and outputs.

b. Inspect the existing data products to find products that need to be generated.

c. Transparently execute only the code in cells necessary to produce the missing data products.

   A common use case for astronomers interacting with the LSST Science Platform for exploration will be to do compute and visualization in the same notebook. In fact, it's not uncommon to do multiple types of compute: e.g., rerun processing, aggregate statistics in the same notebook. Being efficient in notebooks by not redoing compute is both a useability and a costing consideration. First, it's not always clear which cells in a notebook are necessary for a particular cell later in the notebook. It's also difficult to tell which cells are expensive just by looking. A common user reaction to this is to simply rerun all cells when they want to update a cell further down the notebook. Second, the

Science Platform resource budget is very tight. Being a factor of two inefficient is a huge hit to the impact that can be made with the Science Platform.

10.10 *SQR12*: Example Datasets

a. A Science Validation Scientist wishes to distribute test datasets with an example in documentation. They would like users to be able to run the example either in the LSP or on their personal compute.

b. The results of the example would be written to a local repository using the Butler.

10.11 *SQR14*: Share datasets via VOSpace

a. An astronomer working with the LSP would like to share an interesting subset discovered in the LSP environment. The astronomer has access to a VOSpace area shared with other astronomers. For efficiency they would like to provide a script that only pulls from the VOSpace when a `get()` is called, but would like the script to have the same interfaces in both the LSP and when dealing with a VOSpace repository.

10.12 *SQR15*: Minimize storage in notebooks

a. The Commissioning Scientist wants to look at parts of several different tracts of coadds. Pulling over a whole tract is obviously inefficient, but even just pulling the four patches that make up any given subsection of a coadd is inefficient. They would like to stream just the subsection of the tract specifically needed.

10.13 *SQR16*: Notebooks run on archived data

a. A Commissioning Scientist wants to do some historical analysis. They would like to use the LSP to start their analysis. This could include even getting data in deep storage on tape.

10.14 *SQR17*: Retrieve Job data based on metric values.

a. On a regular cadence, the continuous integration system runs a script that generates measurements of various properties of a test dataset: e.g. photometric and astromet-

ric repeatability. These measurements with associated data are persisted as a JSON document describing a `Job` object. The JSON object is published to SQuaSH.

b. At some later point, the SQuaRE scientist retrieves all `Jobs` that were run in the last month where the measurement of the photometric repeatablity is greater than 15 mas.

11 Science Validation

11.1 *SCIVAL1*: Developer-initiated processing

a. A Staff Scientist wants to test an alternate configuration for an algorithm that is part of DRP. The first step involves submitting a workflow defined by a single a Pipeline consisting of multiple SuperTasks on a small patch of sky, with the work expected to consume up to a few hundred core-hours of compute effort but only a couple of hours of wall-clock time, producing a few tens of terabytes of output data products. Ideally the batch submission could be performed from the notebook environment described in Direct Ad-Hoc Analysis, but this is not critical.

   The inputs to the processing could be:

   - raw data
   - an official data release
   - a large-scale processing run initiated and managed by an operator (as per Operator-Initiated Processing)
   - another small-scale processing run initiated by the same developer (as per **SCIVAL1**a)
   - another small-scale processing run initiated by a different developer (as per **SCIVAL1**a)
   - a processing run executed automatically on a predefined cadence by the CI System (as per **CI1**)

   The output data products are typically needed for only a few weeks, but should only be deleted by the developer or with the developer's permission. Some mechanism (possibly policy-only) should ensure that data products are not deleted if they are still in use by another developer.

b. Like **SCIVAL1**a, but instead of testing a configuration change, the developer wishes to test a code change, which could be an of the following scenarios:

(a) The new code has already been included in a managed software release.

(b) The new code has been merged to master for a few days, and hence is guaranteed to have been built by the CI system, but has not been included in a managed software release.

(c) The new code has been committed to a branch, but has not yet been merged to master.

c. Like **SCIVAL1**a, but instead of testing a configuration change, the developer wishes to test a new set of calibration products.

(a) The new calibration products have already been made available to the operations production system.

(b) The new calibration products are experimental and have not been made available to the operations production sysetem.

d. Like **SCIVAL1**a, but the Staff Scientist wants to test a totally new algorithm that produces new data product types.

(a) The new data products are defined along an existing combination of units of data (e.g., tract+patch+filter, which is already used by many coadd data products).

(b) The new data products are defined along a new combination of existing units of data (e.g., sensor+filter, which are used individually to define other data products but have not appeared before without visit before now).

11.2 *SCIVAL2*: Operator-initiated processing

a. Like **SCIVAL1**a, but involving a much larger processing run that will consume up to thousands of core weeks and tens of petabytes of storage. The Staff Scientist requesting the run starts by obtaining permission for a large run and defining the storage duration of the outputs according to some TBD policy. A Batch Production Operator then initiates and manage the new processing run, depending on the actual size of the job and the need for human intervention in executing it. In general, this could involve multiple Pipelines and many batch submissions.

If the change being tested is a code change (like **SCIVAL1**b), it can be assumed that the new code has been included in a managed software release, and if it is a new set of calibration products (like **SCIVAL1**c) the calibration products can be assumed to be made available to the operations production system. New data product types (like **SCIVAL1**d)

to support a single processing run need not be supported, as long is there is a separate mechanism for adding new data product types to the production system as e.g., part of a managed software release.

11.3 *SCIVAL3*: Direct Ad-Hoc Analysis

a. Following one or more processing runs, a Staff Scientist or Astronomer uses a notebook environment to analyze the results, running a combination of predefined and ad-hoc computation, plotting, and display code. The input data may come from any combination of:

   - official data releases
   - small-scale processing runs (as per **SCIVAL1**a, only for Staff Scientists, not general Astronomers)
   - large-scaling processing runs (as per **SCIVAL2**, only for Staff Scientists, not general Astronomers)
   - processing runs executed automatically on a predefined cadence by the CI System (as per **CI1**, only for Staff Scientists, not general Astronomers)
   - local data repositories in the notebook environment's filesystem or some other per-user space.

   Any combination of these may be compared in a single analysis session.

   The data products analyzed will frequently include catalog data stored in a database. In other cases, the processing run may not have included database ingest originally, but the developer performing the analysis wants to use database-like query functionality, requiring either a way to easily perform ingest at this stage or otherwise aggregate the data and support SQL-like query operations.

   The analysis environment should include an interactive Python prompt in which loaded data products can be introspected, image and optionally tabular data displays can be manipulated, and plots can be created and modified. High-level APIs that link points in scatter plots, overplotted symbols in the image display, and records in the tabular data and interactive Python, allowing them to be selected, highlighted, and filtered jointly in all contexts should be included. These may be utilized both by library code (for predefined metrics and plots) and interactive Python users.

Plots, tabular data, and overplotted images may also be saved or loaded in the same manner as more direct pipeline outputs, allowing code written using the same APIs to be run in a non-interactive context.

11.4 *SCIVAL4*: Multi-scale Spatial Analysis

a. Like **SCIVAL3**, but the Staff Scientist wishes to inspect values of a set of metrics on multiple spatial scales in an image-like display, reflecting different binning in aggregate calculations. To enable efficient inspection of large areas of sky, the metrics to be inspected may need to be precomputed on the coarser grids of interest before interactive analysis begins.

   As with **SCIVAL3**, the inputs to this processing will usually be catalog values stored in a database, but they may be catalog values that have not yet been ingested into a database. The metric values will often be compared to (binned) image data on the same spatial scales, which may also require some preprocessing of image data.

   In addition to the gridded metric values themselves, the outputs may involve intermediate data products that can be used for more in-depth analysis. For example, the summary metric may be the width of a color-color scatter diagram in a certain direction, and the intermediate data product might be the filtered set of data points used to construct the color-color scatter diagram.

   After the necessary preprocessing, the Staff Scientist utilizes the same display tools as in **SCIVAL3**, but with the image display able to zoom out to much larger areas using binned (and stitched) images and overlays of the binned metric values included.

References

[1] **[LDM-556]**, Dubois-Felsmann, G., 2017, *Data Management Middleware Requirements*, LDM-556, URL `https://ls.st/LDM-556`

[2] **[LSE-319]**, Jurić, M., Ciardi, D., Dubois-Felsmann, G., 2017, *LSST Science Platform Vision Document*, LSE-319, URL `https://ls.st/LSE-319`

[3] **[LDM-563]**, O'Mullane, W., Jenness, T., 2017, *Butler Working Group Charge*, LDM-563, URL `https://ls.st/LDM-563`

12 Glossary

Composite Dataset

A Dataset made up of more than one items which are themselves Datasets (e.g. `MaskedImage` + PSF + Metadata = `Exposure`)

DataGraph

A graph structure containing DatasetRefs and DataUnits as vertices and the connections between them as edges.

Dataset

Represents a single discrete entity of data, with associated metadata (e.g. a particular `calexp` for a particular instrument corresponding to a particular visit and sensor).

DatasetExpression

A expression matching one or more Datasets; something like a SQL WHERE against a common schema in which each DataUnit is a table.

DatasetIdentifier (or DataID)

A tuple of DataUnits that can be used to label a Dataset.

DatasetIdentifierType

A tuple of DataUnitTypes that describe the label for a DatasetType.

DatasetRef

The combination of a DatasetIdentifier and the name of DatasetType, which can be used to uniquely identify a Dataset within a DataRepository.

DatasetType

The conceptual type of which Datasets are instances (e.g. `calexp`). A new DatasetType can be defined by combining a StorageClass, a DatasetIdentifierType, and a string name.

DataRepository

A collection of Datasets that has the following three properties:

- Has at most one Dataset per DatasetRef;
- Has a label that humans can parse (i.e. RepositoryRef);
- Given a DatasetRef, provides enough information to construct a filename (or key for an object store, or other URI, or set thereof) that is unique across all DataRepositories in a DataRepositoryGroup.

DataRepositoryGroup
A collection of related DataRepositories.

DataRepositoryRef
A human-parseable name for a DataRepository that uniquely identifies it within a DataRepositoryGroup.

DataUnit
Represents a discrete unit of data (e.g. a particular visit, tract, or filter) that may be composed to form a DatasetIdentifier.

DataUnitType
The conceptual type of which DataUnits are instances (e.g. visit, tract, or filter).

Data Discovery System
A software agent that can be used to identify, filter, and relate Dataset and DataUnits using their metadata.

Data Input System
A software agent that can be used to read a Dataset from storage into a InMemoryDataset in memory.

Data Input/Output System
A software agent that is both an Input System and an Output System.

Data Output System
A software agent that can be used to write a InMemoryDataset in memory to a Dataset in storage.

Data Repository Creation System
System that creates a data repository from components.

Data Transfer System
A software agent that can move Datasets from one storage system to another.

InMemoryDataset
The in-memory manifestation of a Dataset (e.g. an `afw::image::Exposure` instance with the contents of a particular `calexp`).

Scientific Data Format
A file format suitable for data processing and analysis (eg FITS/HDF5 and not JPEG/PNG).

StorageClass

A category of DatasetTypes that utilize the same in-memory classes for their InMemoryDatasets and can be saved to the same file format(s), using the same low-level input and output code.

13 Acronyms

Acronym	Description
DAC	Data Access Center
EFD	Engineering & Facilities Database
EPO	Eduction and Public Outreach
LSP	LSST Science Platform [LSE-319]
OCS	Observatory Control System
PVI	Processed Visit Image