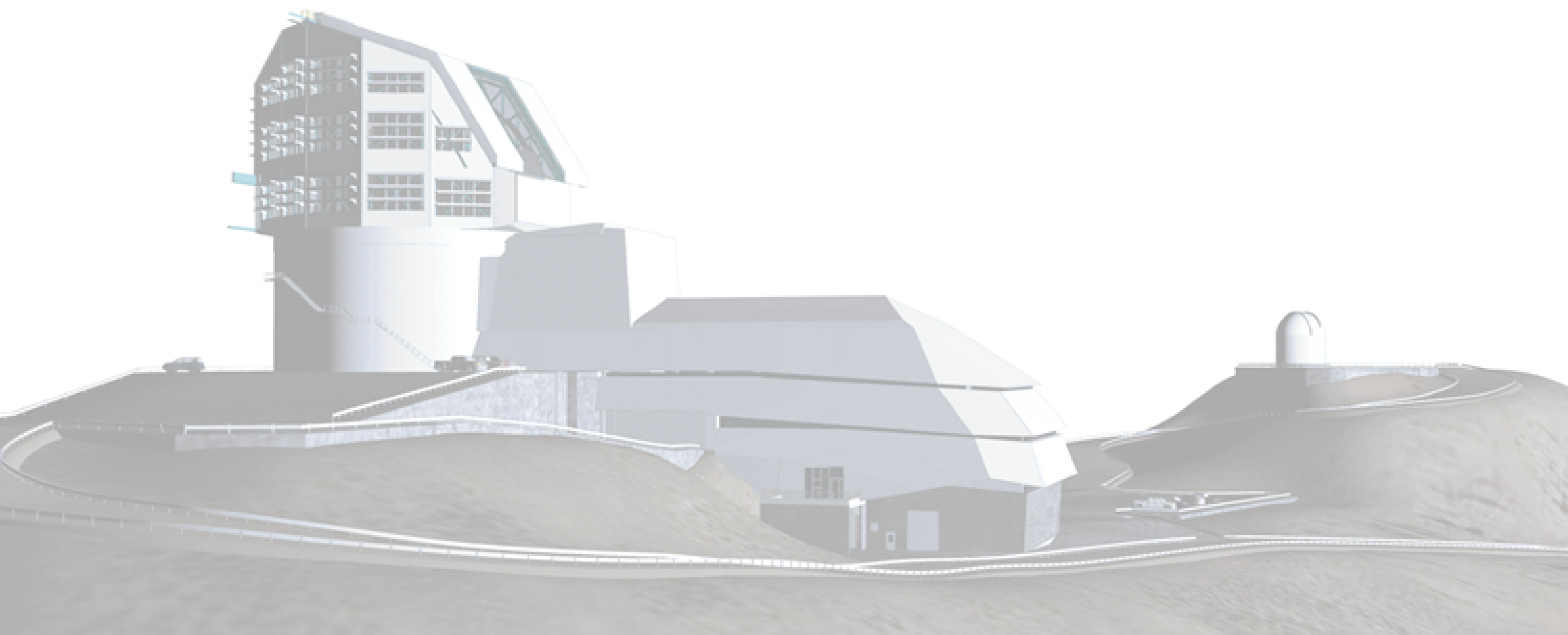
Vera C. Rubin Observatory
Data Management

Schema Management in DM

William O'Mullane, Colin Slater

DMTN-153

Latest Revision: 2020-08-24



Abstract

This note attempts to describe what we mean be schema management, what we would like and how it is currently implemented within DM.

Change Record

Version	Date	Description	Owner name
1	YYYY-MM-DD	Unreleased.	William O'Mullane

Document source location: `https://github.com/lsst-dm/dmtn-153`

Rubin Observatory

Contents

Schema Management in DM

1 Types of Schemas

Abstract schema — DPDD (LSE-163). This describes the scientific content of the tables at a level of detail that the future-user can understand what types of measurements will be produced, without necessarily specifying the exact format of the resulting data. The line between these two levels of detail is inherently blurry; users look to the DPDD to evaluate if project plans are sufficient for their science, but the level of detail required for that evaluation sometimes requires describing implementation choices that may change.

The abstract schema cannot by itself be realized in a concrete table; it lacks the full complement of columns, data type information, and unique column names, and thus requires further elaboration.

"Specification" schemas — LDM-153. This is a schema that *can* be physically realized, and it is designed to fulfill the needs of the DPDD's abstract schema. This schema is necessary both for sizing purposes, and as a "goal" that the pipelines teams can work to as they build and evolve the pipeline output files. The result of the construction project should be for the pipelines to produce data that fully realize this schema.

Because changes to the "specification" schema potentially have impacts on multiple areas of DM (e.g. storage costs or science impacts), it is change-controlled at the DM-CCB level.

"Concrete" schemas — e.g. HSC reprocessing schemas. These schemas *are* physically realized; they are meant to describe data products that currently exist. These schemas must accurately reflect those data products, regardless of what is specified in the DPDD or LDM-153. They are not subject to change control since there is no project management impact that can be caused by any changes. Inaccuracies may cause different dependent services to break, but this is generally comparable in consequence to any other code breakage.

2 Management of Schemas

The DPDD is a project-level change-controlled document; procedures for modifying it are outside the scope of this document.

The "specification" schema in LDM-153 should be updated whenever necessary to reflect any changes in DM's planned data products at the end of construction. Evolution of the LDM-153 schema is expected during construction as the pipelines evolve and the measurement outputs are better understood.

LDM-153 is change-controlled by the DM-CCB, and its contents are generated from the baselineSchema.yaml file in the `sdm_schemas` repository. Procedurally, `baselineSchema.yaml` should be treated like any other LaTeX input for a change controlled document: changes to it may be merged to master via a normal ticket, but are not "official" until an approved RFC releases a new version of the LDM document.

The `hsc.yaml` schema in `sdm_schemas` is a "concrete" schema that is not subject to change control, but the `ci_hsc` integration tests verify that the outputs from that pipeline execution comply with the schema specified in `hsc.yaml`. This ensures that an up-to-date schema is always available, so that steps like loading an HSC reprocessing run into qserv do not require fixing up all the changes to the schema since the previous ingest.

All other schemas in the `sdm_schemas` repository are "concrete" schemas reflecting specific sets of data products; these may be edited as necessary by a normal ticket workflow.

TODO: we only have one hsc.yaml, should we be creating more reprocessing-specific copies? I.e. one for each RC2 reprocessing and saving them in separate files in sdm_schemas.

3 Schema File Format

The "concrete" schema information needs to be available to a variety of different tools, each with slightly different needs. Because of this, it was not sufficient to adopt a format like SQL `CREATE TABLE` statements that were only suited to one particular use, and difficult to parse for all other uses. Instead, the schemas in the `sdm_schemas` repository are in a yaml format defined by the `Felis` tool. The ease of parsing yaml makes it possible for many different tools to all share the same source of schema information, minimizing intermediate stages.

The current uses of the Felis-defined yaml files are:

1. Qserv ingest — The Felis files are used as inputs to the ingest process.
2. TAP_SCHEMA creation — The Felis tool itself is designed to generate SQL statements that populate the schema database used by the TAP standard.
3. LDM-153 generation — The tables in the document generated from the Yaml files.
4. Pipeline data product verification — Continuous integration tests verify that the pipeline outputs comply with the physical schema in `hsc.yaml`.

Most of these uses depend on the YAML schema files without relying on the Felis tool itself. This is generally a consequence of the YAML format being easy to parse by other tools.

4 Science Pipelines

Data products that are generated by the science pipelines must conform to a physical Felis-defined schema in order to be loaded into databases. The pipelines generate a variety of intermediate catalog products, often in afwTable FITS files, which must be transformed into the user-facing tables by the pipelines team. This transformation is more than naming and units: e.g. fluxes may need to have calibrations applied, or uncertainties in pixel units may be transformed to angular units by using WCS information.

This transformation process is not change controlled; the pipelines team has full control over how the user-facing tables are generated. It is only the definition of the user-facing tables that is change-controlled.

The pipelines code implements the various column transformations via a series of “functors”, each dedicated to a particular type of transformation, and the definition of which functors are applied to which columns is defined by a YAML configuration file. This YAML file has an entirely distinct format from the Felis-defined schemas, and its purpose is distinct. The pipelines code generates data products that comply with a particular Felis schema, but there is no automatic linkage between the two. Simple verification can be performed on the pipelines’ YAML file to ensure that it generates columns with the correct names, but any changes to the output data products require both the Felis YAML and the pipelines' YAML to have corresponding updates applied.

A References

B Acronyms

Acronym	Description
CCB	Change Control Board
DM	Data Management
DMTN	DM Technical Note
DPDD	Data Product Definition Document
FITS	Flexible Image Transport System
HSC	Hyper Suprime-Cam
LDM	LSST Data Management (Document Handle)
LSE	LSST Systems Engineering (Document Handle)
LaTeX	(Leslie) Lamport TeX (document markup language and document preparation system)
RFC	Request For Comment
SQL	Structured Query Language
TAP	Table Access Protocol
WCS	World Coordinate System
YAML	Yet Another Markup Language