ObsLocTap: Publishing the Rubin Observing Schedule

William O'Mullane

2026-01-26

1 Introduction

Vera C. Rubin Observatory is required to publish the observing schedule nightly. Following our "VO-first" policy, we choose IVOA ObsLocTAP (Salgado et al., 2021) [1] as the protocol.

This short document discusses some design details.

2 Requirements

LSE-30 (OSS) requires publication of schedule in advance of observing, in OSS-REQ-0378 (Advanced Publishing of Scheduler Sequence):

> **Specification**: The nominal survey schedule shall be published at least **schedSeqAdvance** (24 hours) followed by an updated schedule at least **schedSeqDuration** (2 hours) in advance of each observing visit.

LSE-72 (OCS-DM ICD) requires notification to DM of the "next visit", in OCS-DM-COM-ICD-0031 (Advance Notice of Pointings):

> **Specification**: Advance notice of telescope pointings for science data acquisition shall be made available to Data Management as an OCS event, no later than time **pointingAdvanceNoticeTime** (20 seconds) before the start of the first exposure of a standard visit or the only exposure of an alternate science visit.

LSE-61 (DMSR) flows down OSS-REQ-0378 to DMS-REQ-0353 (Publishing predicted visit schedule, priority 1b):

[1] https://www.ivoa.net/documents/ObsLocTAP/

Draft

Specification: A service shall be provided to publish to the community the next visit location and the predicted visit schedule provided by the OCS. This service shall consist of both a web page for human inspection and a web API to allow automated tools to respond promptly.

Discussion: The next visit and advanced schedule do not need to be published using the same service or protocol.

DMTN-199 (Rubin Observatory Data Security Standards Implementation) documents that the 24-hour requirement, which was added late in the development of the project, derived from a government agency request. The lack of fidelity of such a schedule is acknowledged by the agencies.

3 Design

There are three parts to this:

1. The forecast which comes from the scheduler.
2. The history of which observations were made which we will need to construct.
3. Publishing compliant with the IVOA ObsLocTap spec (Salgado et al., 2021).

3.1 Scheduler forecast

The scheduler puts out an event record with a number of observations in it[2]. Currently the requirement is 2 hours and the system is built around that kind of window. The observing conditions and schedule are felt to be relatively stable on that time frame. But let us assume there will be a record for 24 hours.[3] This shorter record may be produced as frequently as every observation is made with an updated schedule depending on weather etc. Basically when the scheduler has an update a new event is published to SAL.

[2] `https://ts-scheduler.lsst.io/developer-guide/developer-guide.html#operation-modes`

[3] At 24 hours before we would, at the start of the observing night, need to produce the schedule *for the next night*.

These events are available at the USDF in the EFD and the last may be found with this influx query:

```
SELECT * FROM "efd"."autogen"."lsst.sal.Scheduler.logevent_predictedSchedule
          where numberOfTargets > 0 ORDER BY DESC LIMIT 1
```

The current structure of this record is a bunch of columnar arrays representing a time and pointing with the values : `mjd`, `ra`, `dec`, `rotSkyPos`, `nexp`. THe full scheduler schema is at `https://rubin-scheduler.lsst.io/fbs-output-schema.html`. It is understood more values may follow such as field/tile and filter. The potential mapping of these values to the ObsLocTap `Characterization` is given in Table 1. We will however pick this up from the Kafka stream directly.

It seems a separate 24 hour schedule produced each night is preferable to the scheduler team. This makes the forecast easy. There would then also be several shorter `predictedSchedule` events to deal with. The 24 hour schedule is available at USDF and could be picked up independently. The updates could then be handled by looking at two Kafka events namely `lsst.sal.Scheduler.logevent_predictedSchedule` which gives then next 2 hours or so of events and `lsst.sal.Scheduler.logevent_nextVisit` which gives the next pointing (almost certain to happen).

While we are required to give 24 hours notice, any consumers of this 24 hour schedule, as stated in DMTN-199, must acknowledge the lower probability of the observation being carried out compared to the 2 hour updates made during the night.

3.1.1 Handling the schedule

We will need to look at each schedule event record in as it arrives. The best way to do this is to listen tot he Kafka stream which populates the EFD.

An initial prototype could simply publish this to a web page or JSON file. This may be a useful tool in any case[4].

For ObsLocTap a separate record must be created for each pointing in a database lets call this the ObsLocTAP database in this document. ObsLocTap refers to such an entry as a `Character-`

[4] A start has been made in `https://github.com/lsst-dm/obsloctap`

Column Name	Mapping to Rubin
t_planning	logevent_predictedSchedule.mjd
target_name	target_name
obs_id	dataId['exposure'] or obsid from camera - initially some random number
obs_collection	
s_ra	ra
s_dec	dec
s_fov	3
s_region	we could do this though not sure we should store it
s_resolution	0.2 arcsec
t_min	dimensionRecord.timespan.start
t_max	dimensionRecord.timespan.end
t_exptime	t_max - t_min
t_resolution	15s
em_min	Start in spectral coordinates - filter low edge in meters
em_max	Stop in spectral coordinates - filter high edge in meters
em_res_power	
o_ucd	phot.flux.density?
pol_states	NULL
pol_xel	0
facility_name	Vera C. Rubin Observatory
instrument_name	dataId['instrument']
t_plan_exptime	logevent_predictedSchedule.mjd
category	Fixed
priority	0 = in the target queue, 1 = summit look ahead queue, 2 = 24 hour look ahead prediction
execution_status	One of the following values: Scheduled, Unscheduled, Performed, Aborted
tracking_type	Sidereal

TABLE 1: IVOA Obsplan columns, proposed mapping to Rubin metadata

`ization`. Putting this in a Postgres database would seem feasible and allow a fairly standard TAP implementation to be deployed atop. Postgres is already available at USDF. IVOA ObsLocTap considers a table called `obsplan` and the schema is fixed in the spec (reproduced in Appendix A). The schema shall be defined in the schema repository [5] allowing us to generate the SQL from the description in the repo.

Based on the required schema of ObsLocTap (reproduced in Appendix A our schema and mapping to the fields shown in subsection 3.1 is provided in Table 1.

[5] SDM Schema repo `https://github.com/lsst/sdm_schemas/tree/main/yml`

So let us assume we shall have a Postgres table with the IVOA fields as in Table 1. As each new event record is seen the new pointings not already in the database must be added.

The 24 hour schedule can be input with priority 2 (denoting lower likelihood of observation). We can use `observationStartMJD` and `visitExposureTime` as `t_min,t_max` addign a buffer:

$$\begin{aligned} buffer &= 10m \\ t_{min} &= observationStartMJD - buffer \\ t_{max} &= observationStartMJD + visitExposureTime + buffer \end{aligned} \tag{1}$$

As some of the new pointings will replace existing ones, the existing entries should be marked `unscheduled`. The simple approach we would follow for new `predictedSchedule` events is to `unschedule` any existing `Characterization` in the time span of the new event. Observations in the `predictedSchedule` would be marked with priority 1. Observations in the `nextVisit` would be marked with priority 0.

3.2 History of observations

Next we need to record which observations were made to fill the `ObsDataSet`. There is currently no ID in the scheduler forecast so there is no explicit matching of observation to forecast. However as of January 2025 the Scheduler will start to include `target_id` which should propagate through the messages allow easier association of predicted and actual observations. Otherwise for every observation made we shall have to do either:

1. If there is a record in the ObsLocTAP database for this pointing mark it as done (set `execution_status=Performed`). We may want to duplicate more than the time to this table but the exposure ID allows one to get any info we needed from the butler registry. Initially at least we will stick with the schema in the ObsLocTap spec.
2. It there is no record create a record with all the forecast field blank.[6] If there is a record around this time but not this pointing/filter mark that one `execution_status=Unscheduled`

This table would now overlap the exposure log but it is not obvious we should combine these as they serve slightly different purposes. It is further noted by the scheduler team that very few of the scheduled observations would be made as planned. The

[6] It is not clear the ObsLocTap requires this.

3.2.1 Was the observation made ?

There is probably no clear cut answer to this given that there is no ID associated with the forecast. If we assume users of ObsLocTap cared about co-observing then their object could be anywhere in the $9deg^2$ area around the pointing. For a first stab lets say the observation was made if all the following are true :

1. The filter used matched the filter in the forecast. This would seem to fairly important but perhaps it is not - I am unsure of the probability of doing the scheduled observation with the wrong filter. It seems unlikely.
2. The `ra, dec` of the forecast matches (the commanded position should match).
3. The shutter close time of the observation was within 30s of the mjd of the forecast. We are unlikely to be exactly on time so one exposure error seems reasonable.

This is somewhat arbitrary but at least provides a codeable starting point for discussion.

3.2.2 Finding the observations

The butler at the USDF is ingesting the images from the summit within a minute or so. The butler registry can then provide the metadata needed for subsubsection 3.2.1 Again whether this is polling or call back should be discussed. Some assistance from a butler guru will also be needed to work out an efficient query. The prompt processing is trigger at USDF would provide the correct event for doing this for example since it fires on each new image.

3.3 Deployment

It seems the USDF is the correct place for a small process to run which populates the ObsLocTAP database and exposes the obsplan table for the TAP service (subsection 3.4). This should be deployed with Phalanx[7].

[7] `https://phalanx.lsst.io`

3.4 Publishing

Assuming we constructed and populate the obsplan table in the previous sections publishing means having a TAP service for that table. We already use CADC TAP on Qserv and there is a version for Postgres so this step may be fairly straight forward.

Once the TAP service is available then it must be registered. This is describe in section 5 of the spec (Salgado et al., 2021).

4 Implementation

There are two parts to this implementation:

1. The Postgres database and populating it.
2. Serving that up via TAP.

No 2. here is well understood - we have TAP services so we put one in front of the ObsLocTAP database

There are some choices to be made concerning No. 1.

Frossie provided the simple diagram Figure 1 which captures the ideas.

4.1 Create the table

The yaml was added to sdm_schema so Felis may be used to create the schema.

I did not find Felis on pips so I grabbed `git@github.com:lsst/felis.git` and installed it with `pip install .`.

I also cloned `git@github.com:lsst/sdm_schemas.git`.

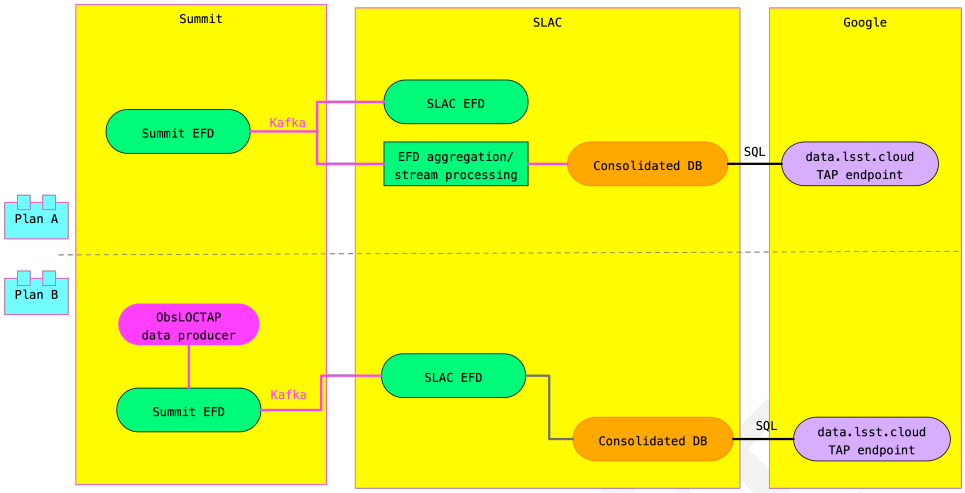

FIGURE 1: Choices on how to populate the Postgres database for ObsLocTap.

This allowed the Felis execution, database login and table creation:[8]
Felis create-all –engine-url=”$engine_url” –dry-run sdm_schemas/yml/obsloctap.yaml > obsloctap.sql

psql -h usdf-butler-session.slac.stanford.edu -U rubin lsstdb1

lsstdb1=> create schema obsloctap; lsstdb1=> create schema obsloctap_dev;

SET SEARCH_PATH = obsloctap_dev; \\i obsplan.sql

\\SET SEARCH_PATH = obsloctap; \\i obsplan.sql

Can check the default schema an if it exists with: show SEARCH_PATH ;
dt

4.2 Populating the ObsLocTAP database

Initially at least we should try plan A and do all of this at the USDF. The EFD messages are replicated there so we can pick up the scheduler messages and process them via a stream processor in Kafka (within Sasquatch).

[8] ivoa. was removed from th DDL

4.2.1 24 hour schedule

The 24 hour schedule may be access at USDF by a call to an API in rubin-sim 2.2.3+ sim_archive.fetch_obsloctap_visits(nights=2). This result can be processed and entries with priority 2 put in the obsplan table. It should be robust against a second call so should always try to update or delete overlapping times.

In theory target_id should be in this data i and propagates through the observing chain but it is not yet provided .

The code for this is in schedule24.py.

4.2.2 4 hour schedule

Then we will process lsst.sal.Scheduler.logevent_predictedSchedule [9] to update/create schedule entries.

The contents of this message are not as rich as the 24 hour schedule - but the code created Obsplan objects from them and attempts to do an update in the same manner as the 24 hour schedule.

The code for this is in consumekafka.py

4.2.3 Observation made

Finally we intended to look at lsst.sal.MTHeaderService.logevent_largeFileObjectAvailable which will give us a pointer to the header (url field) and using astro_metadata_translator we can get r obsid, exposure_id, filter and update the ObsLocTAP database appropriately. However all of this is already in ConsDB so we though a query there would be more appropriate.

ConsDB schema is in sql_schemas[10]. Though there is a ConsDB_query client in rubin_nights - the current code accesses directly using the credentials passed to the container. The query to get the observation data looks like this:

[9] see https://usdf-rsp.slac.stanford.edu/kafdrop/topic/lsst.sal.Scheduler.logevent_predictedSchedule/allmessages

[10] https://sdm-schemas.lsst.io/cdb_lsstcam.html

sELECT exposure_id,obs_start_mjd,obs_end_mjd,band,physical_filter,s_ra,s_dec,target_name,science_progra
from cdb_lsstcam.exposure where obs_start_mjd between 60858.98263978243 and 60859.98263978243
and can_see_sky = True order by obs_start_mjd

The numbers and the MJD stamps between which to search.

The code for this is in consdbhelp.py.

4.2.4 Schedule loop

Each of the 3 DB updates run at different cadences. All are called from ScheduleLoop.py - which runs each in its own thread using asyncio runner.

4.3 TAP service

There is already a ticket DM-39729 for the creation of the Felis schema. Another is needed to expose this via tap.

Along the lines of separation of security concerns this would be deployed on the Cloud (US DAC) not in USDF.

Currently the endpoint /schedule will return the next 24 hours from the Obsplan table as a json file. This is deployed only on usdfdev as of Dec 2023 `https://usdf-rsp-dev.slac.stanford.edu/obsloctap/schedule?time=12`

and on production since July 2025 `https://usdf-rsp-dev.slac.stanford.edu/obsloctap/schedule?time=24`

The time parameter is the number of hours look ahead required, a start time may also be provided allowing one to look at historical entries.

As of July 2025 this is connected to the scheduler and provides the next 2 days schedule, it is update 12 hours.

Though not very efficient a simple rendering to HTML is provided at : `https://usdf-rsp.slac.`

stanford.edu/obsloctap/static/viewer.html

4.4 Phalanx

Deployment is with Phalanx see https://phalanx.lsst.io/index.html. Specifically https://phalanx.lsst.io/applications/obsloctap/index.html.

A Obsplan table from ObsLocTap

The obpsplan tabel schema from the ”Observation Locator Table Access Protocol Version 1.0” is provided here for ease of reference.

Draft

Column Name	Description	Constraint
t_planning	Time in MJD when this observation has been added or modified into the planning log	not null
target_name	Astronomical object observed, if any	
obs_id	Observation ID	not null
obs_collection	Name of the data collection	
s_ra	Central right ascension, ICRS	
s_dec	Central declination, ICRS	
s_fov	Diameter (bounds) of the covered region	
s_region	Sky region covered by the data product (expressed in ICRS frame)	
s_resolution	Spatial resolution of data as FWHM	
t_min	Start time in MJD	not null for execution_status Scheduled or Performed
t_max	Stop time in MJD	not null for execution_status Scheduled or Performed
t_exptime	Total exposure time	not null for execution_status Scheduled or Performed
t_resolution	Temporal resolution FWHM	
em_min	Start in spectral coordinates	
em_max	Stop in spectral coordinates	
em_res_power	Spectral resolving power	
o_ucd	UCD of observable (e.g., phot.flux.density, phot.count, etc.)	
pol_states	List of polarization states or NULL if not applicable	
pol_xel	Number of polarization samples	
facility_name	Name of the facility used for this observation	not null
instrument_name	Name of the instrument used for this observation	
t_plan_exptime	Planned or scheduled exposure time	
category	Observation category. One of the following values: Fixed, Coordinated, Window, Other	not null
priority	Priority level { 0, 1, 2}	not null
execution_status	One of the following values: Planned, Scheduled, Unscheduled, Performed, Aborted	not null
tracking_type	One of the following values: Sidereal, Solar-system-object-tracking, Fixed-az-el-transit	not null

TABLE 2: ObsPlan columns description

VERA C. RUBIN OBSERVATORY

B References

[LSE-30], Claver, C.F., The LSST Systems Engineering Integrated Project Team, 2018, *Observatory System Specifications (OSS)*, Systems Engineering Controlled Document LSE-30, NSF-DOE Vera C. Rubin Observatory, URL https://ls.st/LSE-30

[LSE-61], Dubois-Felsmann, G., Jenness, T., 2019, *Data Management System (DMS) Requirements*, Systems Engineering Controlled Document LSE-61, NSF-DOE Vera C. Rubin Observatory, URL https://lse-61.lsst.io/, doi:10.71929/rubin/2587200

[LSE-72], Dubois-Felsmann, G., Schumacher, G., Selvy, B., 2014, *OCS Command Dictionary for Data Management*, Systems Engineering Controlled Document LSE-72, NSF-DOE Vera C. Rubin Observatory, URL https://ls.st/LSE-72

[DMTN-199], O'Mullane, W., Allbery, R., AlSayyad, Y., et al., 2024, *Rubin Observatory Data Security Standards Implementation*, Data Management Technical Note DMTN-199, NSF-DOE Vera C. Rubin Observatory, URL https://dmtn-199.lsst.io/, doi:10.71929/rubin/2586668

Salgado, J., Ibarra, A., Ehle, M., et al., 2021, Observation Locator Table Access Protocol Version 1.0, IVOA Recommendation 24 July 2021, doi:10.5479/ADS/bib/2021ivoa.spec.0724S, ADS Link

C Acronyms

Acronym	Description
API	Application Programming Interface
CADC	Canadian Astronomy Data Centre
DAC	Data Access Center
DB	DataBase
DESC	Dark Energy Science Collaboration
DM	Data Management
DMS-REQ	Data Management System Requirements prefix
DMSR	DM System Requirements; LSE-61

Draft

DMTN	DM Technical Note
EFD	Engineering and Facility Database
FWHM	Full Width at Half-Maximum
HTML	HyperText Markup Language
ICD	Interface Control Document
ICRS	International Celestial Reference Frame
IVOA	International Virtual Observatory Alliance
JSON	JavaScript Object Notation
LSE	LSST Systems Engineering (Document Handle)
MJD	Modified Julian Date (to be avoided; see also JD)
OCS	Observatory Control System
OSS	Observatory System Specifications; LSE-30
ObsLocTAP	Observation Locator Table Access Protocol (IVOA standard)
SAL	Service Abstraction Layer
SQL	Structured Query Language
TAP	Table Access Protocol (IVOA standard)
UCD	Unified Content Descriptor (IVOA standard)
US	United States
USDF	United States Data Facility
VO	Virtual Observatory

Draft