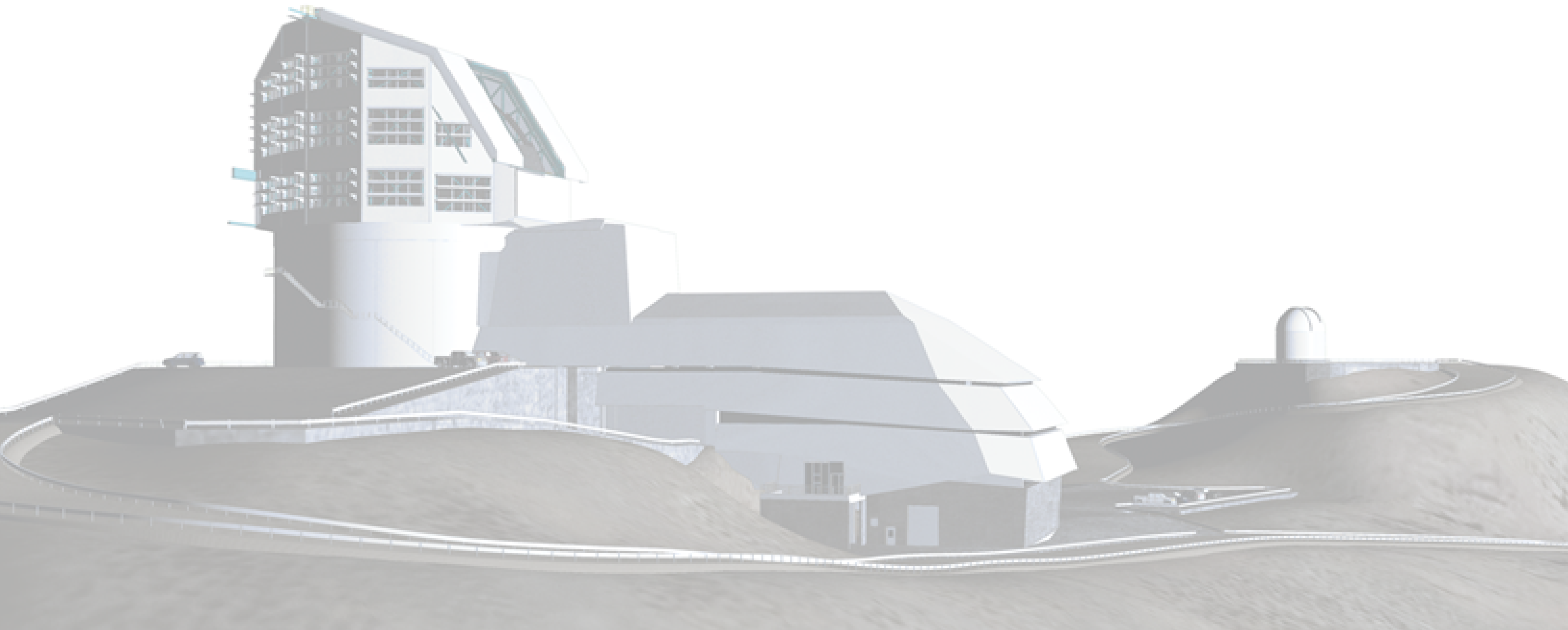
Vera C. Rubin Observatory
Systems Engineering

# Project Documentation Future State Report

Chuck Claver (chair), David Cabrera (co-Chair), Rob McKercher (co-chair), John Andrew, Diane Hascall, Patrick Ingraham, Tony Johnson, Kristen Metzger, Austin Roberts, Jonathan Sick, Matthew Rumore.

SITCOMTN-014

Latest Revision: 2021-08-30



## Abstract

This document is a report on the recommended future state and organization of Rubin Construction Documentation. This will constitute the planned technical documentation package delivered from the Construction Project to Operation as part of the criteria for construction completeness.

# Change Record

| Version | Date | Description | Owner name |
|---|---|---|---|
| 1 | YYYY-MM-DD | Unreleased. | Chuck Claver |

*Document source location:* `https://github.com/lsst-sitcom/sitcomtn-014`

## Contents

**1 Introduction** **1**

**2 Documentation Views** **1**

2.1 Product View . . . . . 2

2.2 Access View . . . . . 2

2.3 Storage View . . . . . 2

**3 Implementation Plan** **2**

3.1 Primary Documentation Sources . . . . . 3

3.2 The Documentation Portal . . . . . 3

3.2.1 Design requirements . . . . . 3

3.2.2 Technical architecture . . . . . 4

3.2.3 Support for multiple views . . . . . 7

3.2.4 Information security . . . . . 8

3.2.5 Effort estimate . . . . . 9

3.3 The Path Forward with DocuShare . . . . . 9

3.4 The Path Forward with PDWM Works . . . . . 9

3.5 Required Resources & Subsystem Responsibilities . . . . . 10

3.6 schedule . . . . . 10

**4 Transition Plan and Workflow** **10**

**5 Risk Assessment** **10**

**A References** **10**

**B Acronyms** **10**

# Project Documentation Future State Report

## 1 Introduction

Put basic description of approach and report structure here.

This report include details on the following topics:

- DocumentationViews
  - ProductView
  - AccessView
  - StorageView
- Implementation Plan
  - PrimarySources
  - DocumentationPortal
  - DocuSharePath
  - PDMWorksPath
  - Resources & Responsibilities
  - Schedule
  - TransitionPlan
- RiskAssessment

## 2 Documentation Views

The section describe three different view of the documentation content:

- The Product View
- The Access View
- The Storage View

### 2.1 Product View

Describe here the properties of the product View and provide diagrams showing the proposed structure for this view.

### 2.2 Access View

Describe here the properties of the Access View and provide diagrams showing the proposed structure for this view.

### 2.3 Storage View

Describe here the properties of the Storage View and provide diagrams showing the proposed structure for this view.

## 3 Implementation Plan

Add here an overview of the proposed implementation plan including the following specific areas of implementation:

- Primary Sources
- Documentation Portal
- DocuShare
- PDM Works
- Resources and Responsibilities
- Schedule
- Transition Workflow

## 3.1 Primary Documentation Sources

Describe here the proposed location of the primary documentation sources. These are a subset of the identified source listed in SITCOMTN-012.

## 3.2 The Documentation Portal

The working group recommends the creation of a *documentation portal* web application as a means of making documentation content discoverable and accessible to Rubin Observatory staff. The documentation portal provides interfaces for both searching (based on content and metadata) and browsing (based on hierarchical categorization) of documentation resources. Documentation does not reside within the portal itself. Rather, the portal's objective is to efficiently link the user to the document where it is stored in any of the observatory's adopted storage platforms (Section 3.1). The portal discussed here is based upon the www.lsst.io website, which provides a search and browsing interface for the Rubin Observatory's public-facing technical documentation. The new portal will be accessible only those with Rubin Observatory staff credentials, and will be purpose-built for observatory and survey operations. This section describes the design principles, technical architecture, security model, and cost estimate of the Rubin Observatory documentation portal.

### 3.2.1 Design requirements

The design requirements of the documentation portal reflect the recommendations made by the Documentation Working Group elsewhere in this report:

1. The role of the documentation portal is to link to documentation resources. The portal itself does not host the content itself, or provide user interfaces for creating and maintaining new versions of documentation content. This requirement reflects the working group's recommendation that documentation content should be hosted on a select set of platforms that are idiomatic for the content and the teams that work with that content (Section 3.1).
2. The documentation portal must provide equal support for content stored in any of the working group's recommended storage platforms.
3. The documentation portal must be capable of supporting several hierarchical browsing

schemes for accessing content based on different organizational views of the documentation (Section 2).

4. The documentation portal should automatically update and sort documentation content, to the greatest extent possible. In other words, curators should only need to maintain documentation in the document's storage platform, without any administrative action through the documentation portal's UI. A consequence of this requirement is that the documentation portal should not persist information about a document that is not available from the document's own storage platform.

5. The documentation portal should be secured so that it is only accessible to users with Rubin Observatory staff credentials.

6. The documentation portal should not maintain fine-grained access control for specific documents or categories of documents. For secured documents, the portal relies upon the security mechanisms of the document's own storage platform. The portal should also reduce its metadata storage of confidential documents to ensure that content cannot be inferred from a search, for example.

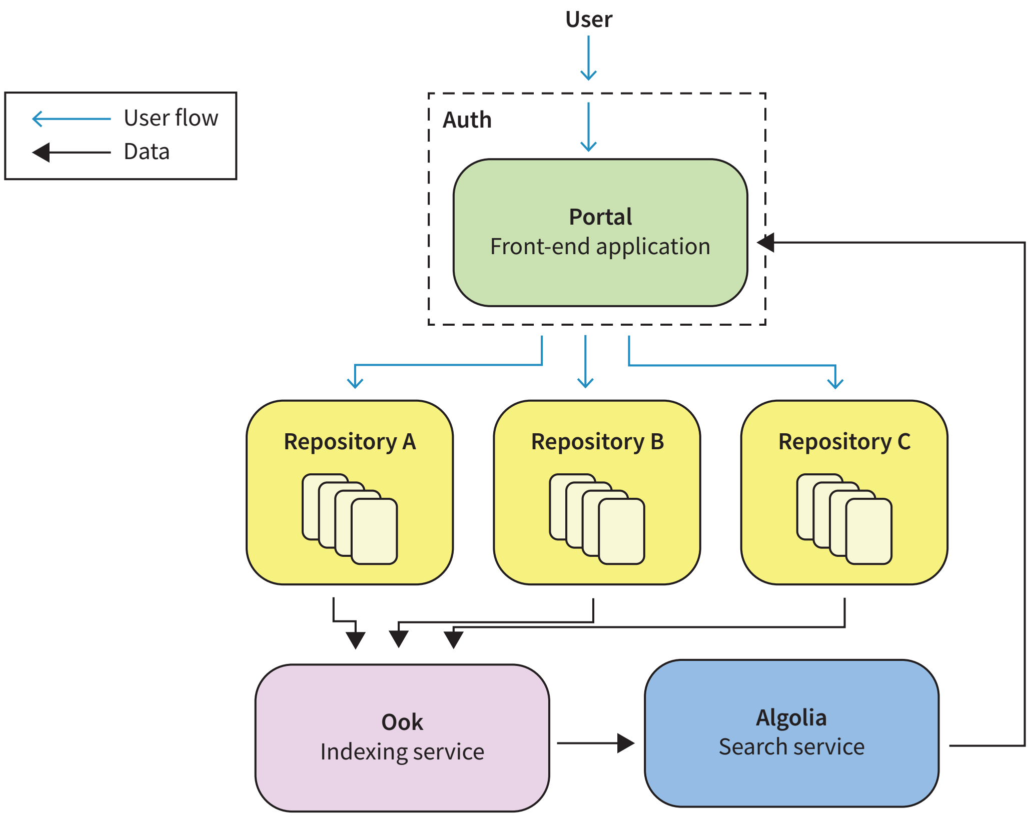
#### 3.2.2 Technical architecture

The architecture described here is based upon that which is already put into production with the www.lsst.io portal for public-facing documentation. Starting from this working archetype relieves a great deal of technical risk and development from the new portal's implementation. Both portals share the use of Algolia as a search backend and Ook as a content indexing service. The documentation portal discussed here will have an independent front-end to support the specific access views recommended by the working group. The documentation portal will also use a separate instance of the Algolia database to eliminate any risks associated with leaking internal documentation to the public-facing www.lsst.io portal.

Figure 1 depicts the components of the documentation portal.

**3.2.2.1 Algolia** The core function of the portal is to enable access to documentation through browsing and search. To implement this, the portal needs a back-end service that contains metadata about Rubin's documentation holdings and provides interfaces to access and query that metadata from the front-end (website). This search database and interface could be

FIGURE 1: Architecture of the documentation portal. Users find documents on the portal application, which in turn provides links into the original documentation repositories. Data in the portal application is supplied by the Algolia search service, which in turn gets its metadata from the original documents in their repositories via the Ook indexing service.



made for "free" with entirely open-source components such as Elasticsearch and in-house web service. However, a search database and service are sufficiently generic that we cannot add value by making it in-house, and in fact developing, tuning, and operating this service, would costly in terms of labor. For the www.lsst.io public documentation portal, we opted to use Algolia and recommend that we make the same choice for the observatory's documentation portal.

Although www.lsst.io is currently operating on a free open-source license of Algolia, the observatory's internal documentation portal would be an entirely paid license. Algolia prices based on record counts and request rates. In operating www.lsst.io, we found record count to be the limiting factor. At the moment, 1000 records costs $1 per month. 1,000,000 records would cost $850 per month with volume discounts. For reference, the www.lsst.io service currently uses 110,000 records to host all technical notes and change-controlled documents with drafts hosted on lsst.io.

Note that a single document is composed of potentially many records in Algolia. To optimize full-text search, we break a document into smaller records, generally across section boundaries. Our current algorithms generally produce small record, so it is possible to reduce costs by tuning how we segment content into Algolia records. Furthermore, each sorting option requires a separate pre-sort index. Sorting documents by date, and document number, in addition to relevance, consumes three times as many records as only sorting by relevance.

**3.2.2.2 Ook** The Ook service is responsible for continuously indexing content into Algolia. Whereas we chose Algolia to provide a turn-key search database service, for www.lsst.io we chose to build the indexing service in-house to have complete control over how documents are indexed, and what metadata is associated with each document. For example, LaTeX-based documents are indexed based on metadata exported from the Lander PDF landing page generator (also developed in-house), which in turn parses LaTeX syntax in the document source to access metadata such as titles, authors, and so on. The configurability of Ook is beneficial to indexing other types of highly specialized documentation.

The key design principle of Ook is that metadata is extracted from the document as it appears in its repository, rather than requiring direct human interaction with Ook or Algolia to curate the data. This allows Ook to scale well across an organization as large and varied as Rubin because individual teams manage documents as they already do in the repositories they are already familiar with.

Ook is built such that new content types can be added by writing additional Python-based workflows for each content type. Ook itself provides utilities for queuing ingests, converting content and formatting data for Algolia, and working with the Algolia service itself.

Ook indexing operations can be triggered several different ways. For example, the lsst.io service publishes messages to a Kafka cluster whenever documentation is published on that platform; Ook subscribes to those messages and queues indexing workflows. Ook indexing operations can also be scheduled through an HTTP API. Generally, the goal is to trigger indexing operations automatically whenever the source material changes.

The existing Ook indexing workflows work by downloading content from websites and web services (HTTP APIs). If content is not easily accessible, it would be possible to develop an alternative workflow, such as submitting copies of the document directly to Ook for indexing. Some document repositories may offer online access but have hard-to-use APIs (such as DocuShare). These difficulties can be worked around, for example by emulating a web browser to download content and metadata, but at the cost of more fragile indexing workflows.

Ook is currently operated as a Kubernetes application in the Google Cloud. This arrangement is ideal for minimizing operations cost, and providing convenient scaling.

**3.2.2.3 Front-end application** The front-end web application is how users (Rubin Observatory staff) find documentation. The web application does not provide the document itself—instead, the web application provides a search result card that the user can click on to access the document in its original repository. The Algolia services provides all browsing and search functionality; the front-end application provides the user interface over top of Algolia.

In addition to providing a link to the original document, the portal can also provide an immediate view of a document's metadata. Although the front-end application can show a basic view of a document's metadata based on metadata common to all records, the website can be developed to show additional metadata for different types of documents.

For www.lsst.io, we built the site as a React JavaScript application. This allowed us to use and customize the pre-made widgets provided by Algolia for building the user interface.

The front-end application will be accessible only to users who log in. The simplest way to approach this is by putting the application behind a VPN so that the application is completely separate from security concerns. Another approach would be to place the application behind an OAuth proxy to provide a slightly better user experience.

### 3.2.3 Support for multiple views

In Section 2, the documentation working group outlines several views for hierarchically arranging documents trees. These views correspond to navigational structures in the front-end application. Although the front-end application code is generally ”aware” of the different trees, individual documents are placed in the tree on the basis of metadata in their Algolia records, so that the Algolia service can pre-sort and filter documents into the trees. Since Ook supplies metadata to Algolia, and Ook in turn leverages metadata native to the document and the document's repository, the responsibility for curating documents into different views is the responsibility of individuals managing documents in each repository.

### 3.2.4 Information security

Documentation has multiple types of security concerns, such as control over who and how documents are updated, control over who can access documents, and ensuring the long term integrity and preservation of information. Since the documentation portal is not the canonical repository for any documentation, the portal is not involved in controlling document updates and preservation. The portal's key security concern is access control.

The portal is designed to only provide authentication-based access control. Any Rubin staff member with credentials can access any metadata records contained within the documentation portal. Once a user selects a document to view, they are forwarded to that documentation repository and must authenticate with that repository and be subject to its access control rules.

However, the metadata contained in the documentation portal can be potentially rich, even including the full-text content of a document to enable search functionality. It is conceivable that some of this metadata may not be appropriate for observatory-wide access (such as information with export controls, for example). In these cases, the most realistic approach to preserving strict access controls in these situations is to limit what metadata is available. In order of strictness, the following approaches can be used:

1. Omit full-text content of a document from Algolia records.
2. Limit or obfuscate other metadata (such as titles) in the Algolia records.
3. Omit the individual documents altogether from Algolia and instead link to a documentation landing page hosted by the secure document repository itself.

In all cases, controlling how documents are indexed is done by configuring the indexing service, Ook.

### 3.2.5 Effort estimate

The key tasks for implementing this plan are:

1. Building additional content indexing workflows into Ook and updating the metadata schema to accommodate all use cases.
2. Build the documentation portal application.
3. Curate documents so that they are uniformly present in their document repositories and have any metadata that is expected by the Ook indexing workflows.

The last task category can be distributed to teams working in each document repository.

To provide a sense of the scale of the first two tasks, the implementation of the www.lsst.io portal is a useful reference. Building the first version of Ook, which indexes LaTeX documents and Sphinx-based technical notes in lsst.io required 4 weeks of work. Building the front-end website required 6 weeks.

Since that original implementation, we have gained more experience building React web applications and working with Algolia, so we can expect less than 4 weeks of work to build out the internal web application. Although Ook can be expanded as is, the key uncertainly is in the number and complexity of the additional indexing pipelines that need to be built for additional document types and repositories. Generally, though, an internal documentation portal can likely be stood up within 1 to 3 months, with a potential long tail of effort to continue to add support for additional content types.

## 3.3 The Path Forward with DocuShare

Describe the trade study conducted for various was we can move ahead with DOcushare and the proposed server based solution.

## 3.4 The Path Forward with PDWM Works

Describe the maintenance plan of the existing CAD models in PWDM works.

### 3.5 Required Resources & Subsystem Responsibilities

Description of the estimated resources need to cary out this implementation plan and what the roles and responsibilities are for each subsystem in this context.

### 3.6 schedule

Outline the schedule needed for implementation to meet the objective delivering a coherent tachnical document packed at the end of the Project Construction effort.

## 4 Transition Plan and Workflow

Describe the process for transition from the current state to the future state.

## 5 Risk Assessment

Describe the risk exposure if various part of the whole of this implmentation is not conducted.

## A References

## B Acronyms

| Acronym | Description |
|---|---|
| API | Application Programming Interface |
| HTTP | HyperText Transfer Protocol |
| LaTeX | (Leslie) Lamport TeX (document markup language and document preparation system) |
| PDF | Portable Document Format |
| SE | System Engineering |

| | |
|---|---|
| UI | User Interface |
| VPN | virtual private network |