OLCF/CADES Federated Scientific Data Management System

System Overview and Design

Dale V. Stansberry

Advanced Data and Workflows Group

Oak Ridge Leadership Computing Facility

Oak Ridge National Laboratories

February, 2019

# Abstract

This document provides a technical description of a Federated Scientific Data Management System currently under development within the Compute and Data Environment for Science (CADES) of the Oak Ridge Leadership Computing Facility (OLCF). This document includes the rationale, features, use cases, architecture, and high-level design of the system. This document supersedes all previous documents located in the associated project repository (located at <https://code.ornl.gov/sdms-dev/SDMS>). Note that the original designation of this system was simply the “Scientific Data Management System”, or “SDMS”. This terminology is retained within this document for internal use only. The public name for this system has not been finalized, but currently “DataFed” (short for “Data Federation”) is the most likely candidate.

# Introduction

The SDMS is a federated, big-data storage, collaboration, and full-life-cycle management system for computational science and/or data analytics within distributed high-performance computing (HPC) and/or cloud-computing environments. The SDMS provides a suite of features and capabilities that ease the typical data access and management burdens associated with complex, data-oriented research activities – allowing researchers to focus on their research activities instead of developing ad hoc data management solutions. When fully utilized, the SDMS has the potential to improve the quality and repeatability of research output through a variety of advanced features including metadata and context capture, publish-subscribe data dissemination, and dependency-driven collaboration tools.

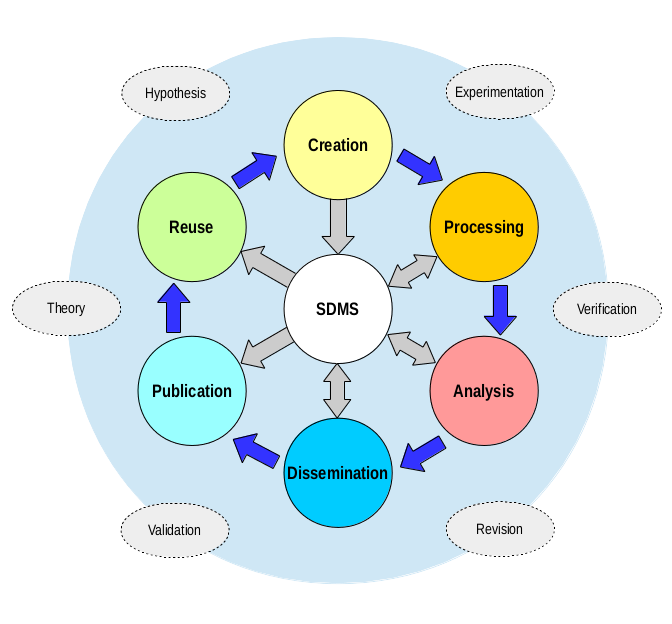
The SDMS overlays and manages the data storage, communication, and security infrastructure present within a network of participating organizations and facilities in order to produce an abstract and homogeneous view of the underlying heterogeneous hardware. The SDMS utilizes a centralized management service that maintains an awareness of, and control over, all of the data stored within this federation – resulting in an integrated, but geographically distributed, data infrastructure with simple and consistent data access semantics. SDMS users are able to access and manage their data from anywhere within the federation - without ambiguity or regard to physical storage location.

The SDMS utilizes an object store paradigm where data is presented as a record (object) within a database with associated unique identifier, raw data, rich metadata, and user-defined relationships. This approach has advantages over the typical hierarchical file system paradigm because it eliminates identity uncertainty (by separating identity from storage location), maintains synchronization between raw data and metadata, and enables data discovery through metadata indexing and search. This approach elevates plain files on an isolated file system into stable and unique data artifacts that can be safely referenced, shared, and discovered with a high degree of certainty.

The SDMS is best suited for mid-tier data access patterns, such as managing important and/or relatively stable data over the duration of a research project. Applications that require low-latency and/or high-bandwidth data I/O should utilize local or high-performance parallel file systems; however, the SDMS can (and should) be used to stage data in and out of these systems once data processing is complete. At the other end of the spectrum, the SDMS can be used for long-term storage and dissemination of reference data; however, the SDMS does not provide all of the features necessary for long-term data curation or data publication - such as disaster recovery, Digital Object Identifiers, or interfaces with external publication indexing services. The extent to which the SDMS will integrate with archival and publication systems is currently under discussion.

## Data Life Cycle

Because the SDMS is a domain-agnostic data management tool, it does not mandate a particular data life cycle model. Rather, the SDMS is intended to support a wide variety of data management scenarios as defined by the users and/or scientific communities that adapt SDMS into their particular data life cycles. The SDMS does, however, provide many features that are important to the creation, access, and reuse aspects of typical data life cycles. Consider the hypothetical data life cycle, shown below:



This data life cycle supports the general scientific method which entails developing hypotheses from observations, designing and conducting experiments, verifying and validating experimental results, and, if successful, developing new (or refining existing) scientific theories. The SDMS provides a common “data backplane” to support the various phases of this process. For example:

* **Data Creation** – The SDMS can be used to capture raw experimental or simulation data, along with metadata describing context (environment, configuration, provenance, parameters, etc.)
* **Data Processing** – The SDMS can be used to stage data into and out of compute environments for data processing/reduction.
* **Data Analysis** – Like the processing phase, the SDMS can be used to stage data for analysis. This phase is likely to produce new data/metadata that would also be stored in the SDMS and associated with the input data (i.e. provenance capture).
* **Data Dissemination** – The SDMS can be used to concisely share data with individuals, teams, or the general public - providing an opportunity for peer review and validation prior to publication. The SDMS provides built-in collaboration features to support geographically distributed research.
* **Data Publication** – While the SDMS is not a data publishing system, it can be used as the source of metadata and raw data needed for external publication. SDMS data records can be updated with resulting DOI numbers and publisher information
* **Data Reuse** – If data is retained in the SDMS after publication, the SDMS can be reused by other researchers to easily locate and access that data for additional studies, such as verification of published results or derivative research.

When considering the above data life cycle, it is important to understand that the SDMS enables this process to span an arbitrary number of geographically distributed facilities with widely dispersed collaborators – all while retaining the same simple interface that would used by a single researcher within a single facility. This is the key concept behind the data federation aspect of the SDMS.

## Features

The SDMS consists of a federation of data storage repositories with a centralized data management and transport service. The SDMS provides a graphical web application and a command-line client. The web application is well suited for data organization, dissemination, and discovery; whereas, the command-line client is best suited for batch processing and workflow integration. Briefly, the key features of the SDMS include:

* Simple and uniform data access from any member facility
* Federated user identity and credential management
* Data tracking via unique identifiers and aliases
* Integration of raw data with user-defined structured metadata
* Live data provenance
* Data collections and views
* Data projects
* Fine-grained access controls
* Publish-subscribe data events and annotations
* Community-driven data taxonomies
* Tagging and labelling
* Metadata query language
* Secure data transfer and storage
* High-performance, reliable data transfer with caching
* Data concurrency controls (read/write locks)
* Locality-aware direct data access

With the SDMS providing a generalized cross-facility data infrastructure, there are many opportunities for users and/or facilities to build powerful, domain-specific applications on top of the data foundation provided by the SDMS. SDMS integration can be achieved through simple scripting, or by building custom applications using SDMS client libraries.

## Background

The concept of the SDMS originated from direct observation and communication with scientists working with large or complex collections of datasets, and/or working within complex, distributed experimental and compute environments – sometimes involving movement of data across facilities and/or security enclaves. The lack of suitable existing data management and indexing solutions has resulted in a proliferation of ad hoc domain-specific data management/transport solutions developed by individual research teams. These ad hoc solutions are typically incompatible with one another, have fragile implementations, and poorly supported. More importantly, a significant effort is required to develop and maintain these solutions – effort that could have been spent on actual research.

Most existing commercial systems are oriented toward enterprise (business) environments and do not address the complexities of HPC environments. The lack of integration, consistency, and HPC suitability among these technologies places a significant burden on scientific researchers and groups who must adopt, learn, configure, then follow complex and/or tedious manual procedures to utilize a piecemeal data pipeline. The result can be wasted time, confusion, and even errors related to properly identifying, locating, provisioning, and disseminating large scientific datasets.

The SDMS was envisioned as a system of integrated technologies that would provide a simple, consistent, and holistic solution for the common data management needs of scientific researchers and users of HPC systems. Unlike grid computing, the SDMS was not intended to integrate or abstract compute resources - nor does would it require complex and/or time-consuming administration of user accounts and security credentials across member organizations. Ultimately, the SDMS should bridge the gap between existing enterprise-level data management solutions and the complex distributed environments associated with high-performance computing.

# Overview

From a user’s perspective, the SDMS can be thought of as a “black box” into which large datasets and associated metadata can be placed and subsequently retrieved from any location within the SDMS federation – essentially a big-data *database* in the cloud. But the SDMS is really more like a big-data *cloud-application* built on top of a database – offering many additional data services beyond simple storage and retrieval. For example, data can be:

* Securely shared with other users
* Owned and managed collectively
* Located using metadata-based queries (such as experiment, simulation, or analysis parameters)
* Organized into collections and taxonomies as reference data
* Cleaned and processed in preparation for publication

The SDMS provides a Web Portal that can be accessed globally, and a command-line interface (CLI) that can be used from within supported compute or data environments. The Web Portal is primarily intended for data organization, sharing, and discovery; whereas the CLI is mainly used for scripting, batch processing, and workflows. (A client library is also available for integrating SDMS capabilities directly into custom applications.) Data transfers can be initiated and monitored from both the Web Portal and thee CLI.

The SDMS provides an abstract view of data within the federation in that it tracks data using unique identifiers rather than the physical storage location (file system path and file name). Users cannot access managed data without utilizing the SDMS first. Data within the SDMS federation is physically stored in SDMS “data repositories” which are services that expose facility-managed storage systems to the SDMS network. From a user (or process) perspective, a data repository may be local or remote. Local repositories are hosted within the same facility or organization where a user process is running and is accessible via a network mounted file system.

The SDMS support three distinct data access patterns:

1. Download of remote or locally stored data to a local file system. This is the simplest access pattern and involves either transferring data from a remote SDMS data repository, or copying data from a local SDMS data repository into a local file system. Once transferred or copied, the data is a normal (unmanaged) file that can be accessed based on ownership and permissions granted by the file system. Any modifications made to the local data file will not be synchronized with the SDMS managed data, nor will changes made to the remote data be synchronized with the local copy. An explicit SDMS update command must be used to push any local modifications back into the associated SDMS data repository.
2. Direct access to a locally cached copy of remote data. This allows direct access to remote data that has been, or will be, transferred to a local SDMS-managed data cache. Using this option is situational and will avoid data transfer latency if the data is already cached; however, in order to utilize this method, a read or write lock must be acquired on the data being accessed. Only one write lock may be held, but read locks are not mutually exclusive, so many users may read the data concurrently. If modifications are made to the cached data, the remote data will be automatically synchronized from the local cache when the write-lock is released (a remote data transfer, during which new write locks cannot be acquired).
3. Direct access to locally stored data. This is similar to case #2 except that a user process can access the physical storage location of the data so long as the associated data repository is local. Like case #2, a read or write lock must be acquired first; however, unlike case #2, no data transfer is required after modifications are made to the data. Note that managed data will never be cached locally if it the data repository is local.

Which access pattern should be used depends on a users specific needs and permissions. While accessing locally cached or locally stored data may seem ideal, the read/write bandwidth of this approach will typically be slower than accessing the file from a local high-performance file system. For use cases involving large amounts of file I/O, case #1 may be a better choice. Also, holding a lock on a data record for a long period of time may negatively impact other users needing access to the same data. Also, acquiring read/write locks requires specific permissions that must be granted by the owner or administrator of the associated data.

[Note: in the future, the SDMS may allow locks to be acquired by anyone without special permission, but this will require an advanced concurrency model that will take considerable effort to develop.]

A user process may attempt to utilize cached data opportunistically: if the data is available and not locked, latency is avoided; otherwise, the user process can fall back to remote access (case #1). The SDMS manages local data caches in a least-recently-used manner, so frequently accessed data will be retained longer.

## Globus

The foundational benefit of the SDMS is that data can be accessed from anywhere within the SDMS federation in a *simple* and *uniform* manner – regardless of data storage location, security policies, or network topology. This uniformity is achieved through a combination of using an on-line service called Globus as the user identity and authentication mechanism, as well as for the data transfer mechanism (initiated and controlled by the SDMS). Because of this integration with Globus, SDMS users are required to a Globus account and understand the basics of how Globus data transfer works.

Globus’ data transfer service uses GridFTP to both transfer data and to provide basic file system services (such as navigating and listing directories) for a given storage system. GridFTP is a high-performance, secure, and reliable file transfer service that is able to detect and utilize multiple network pathways to parallelize data transfers to or from remote GridFTP storage locations, called “end-points” in Globus terminology. GridFTP delegates user authentication, and within Globus, this is typically provided by a Globus “My Proxy” server. The primary value-add of Globus is that it supports secure site-specific user authentication and subsequent exchange of user temporary credentials - building a trust relationship between different organizations. Once this trust is established, user may transfer data between authorized end-points without further security interactions.

Since the SDMS uses Globus for remote data transfers, it can only transfer data to or from Globus end-points, and, before transfers can be requested, the SDMS user must “activate” (authorize) those end-points first. This can be done via the Globus Transfer web site (at <https://app.globus.org/file-manager>) or via the SDMS Web Portal. Once activated, an end-point can be used for some period of time determined by the local security policy before requiring re-activation.

SDMS users should familiarize themselves with the Globus online data transfer documentation located at <https://www.globus.org/data-transfer>.

### SDMS Accounts, and Security Credentials

Because the SDMS relies on Globus for user identity and authentication, user must have a Globus account, with a linked Globus ID account, prior to creating an SDMS account. Given an active Globus account, access to the SDMS requires a one-time registration through the SDMS Web Portal at [https://sdms.ornl.gov](https://sdms.ornl.gov/). SDMS users are identified by their Globus ID identity, without the domain suffix.

During the registration process, the SDMS requires users to create a password that is used to authenticate users via the SDMS command-line interface (CLI). The CLI allows users to access most functions of the SDMS interactively; however, the CLI also allows users to configure a given environment for non-interactive SDMS sessions (which is required for scripting, batch jobs, and workflows). This process results in SDMS security credentials being generated and installed in a well-known location within the local environment. These credentials must be kept private, and can be revoked at any time by the user. It is important to understand that these CLI credentials are distinct from the Globus credentials used to log-in and authorize data transfers.

(Note: these credentials may be removed In a future version of the SDMS – if there is a local service that can securely grant access to a users Globus credentials.)

## Data Records

The SDMS defines a “data record” as the basic unit of data including a unique identifier, raw data, system-defined unstructured metadata, and optional user-defined structured metadata. From a user perspective, an SDMS data record is essentially an atomic record within a database that can be created, read, updated, and deleted; however, it is important to understand that raw data is stored separately from the actual database record in an SDMS data repository.

### Identifiers and Aliases

Data records are assigned a stable system identifier that is unique within the scope of the SDMS. These identifiers are large, somewhat random numbers prefixed with “d/”, such as “d/31726937”. Because system identifiers are somewhat difficult for humans to read and remember, the SDMS allows an optional human-readable alias to be defined and use in place of the system identifier. Aliases can contain alphanumeric and limited punctuation characters and must be unique within the scope of the owner of the data.

To ensure uniqueness within the system, the SDMS prefixes user-defined aliases with a value derived from the owners identifier. For data owned by users, the prefix is “u:<user\_id>:”, where <user\_id> is the owning user’s account ID. For data owned by projects, the prefix is “p:<project\_id>:”, where <project\_id> is the project’s system ID. When owners use an alias, the prefix is not required; however, non-owning users must include the prefix in order to disambiguate the alias from their own aliases. The SDMS command-line interface provides a feature to set the active prefix such that users need to specify it each time when working with non-owned data.

Note that the Data Collections follow the same semantics for identifiers and aliases, except that collection identifiers are prefixed with “c/” instead of “d/”.

### Metadata

Data records include a number of pre-defined, general informational fields that can be viewed and queried by users. Some fields are required, most are optional, and several are generated by the SDMS itself. These fields include:

* **ID** –An automatic field containing the system-generated data record ID.
* **Alias** – An optional field for a human-friendly data record alias.
* **Title** – A required field that is displayed (along with ID and/or alias) whenever data records are listed. The title is intended to be short an concise and is indexed for full-text searching.
* **Description** – An optional field where free-form text may be provided to describe the data record, context, external reference, or any other information deemed useful. This field can be large and is indexed for full-text searching.
* **Keywords** – An optional list of keywords, or key phrases, that is indexed for full-text searching. Most any list delimiter may be used.
* **Topic** – An optional topic that the data record is listed under. Refer to the Topics section of this document for further information.
* **Data Repo** – An automatic field containing the identifier of the data repository that will store the raw data associated with a data record.
* **Data Size** – An automatic numeric field providing the size of the data record’s raw data, in bytes. This field will have a value of 0 prior to uploading raw data.
* **Create Time** – An automatic timestamp field indicating when the data record was created.
* **Update Time** – An automatic timestamp field indicating when the data record was last updated.
* **Data Upload Time** – An automatic timestamp field indicating when raw data was last uploaded.
* **Owner** – An automatic field containing the identifier of the owner of the data record (either a user or project)

In addition to the above system-defined metadata fields, users may also include arbitrary structured metadata with a data record. This structured metadata is defined using the JSON format, and is searchable using the supplied SDMS query language. Structured metadata is extremely valuable for locating and dynamically organizing datasets based on data attributes, parameters, or even calculated results.

JSON is a simple and human readable, yet flexible, structured data representation that can be used to define arbitrarily complex data structures; however, JSON is not suitable for storing binary data such as images or other types of media. The SDMS Web Portal provides a powerful JSON editor to aid users when manually inputting or editing structured metadata. (It can also be uploaded from a JSON file.)

## Data Collections

SDMS data collections are one of several methods available for organizing and controlling access to data. Collections resemble the folders/directories of a file system in that they are hierarchical; however, unlike file systems, data records are not restricted to being placed in a single collection. Collections may be used to set access control rules that are inherited by contained items. Every SDMS user and projects owns a “root” collection that serves as the default collection for new data records and/or collections when an explicit parent collection is not specified.

Like data records, collections have specific attributes, defined below, but do not contain additional domain-specific metadata.

* **ID** –An automatic field containing the system-generated collection ID.
* **Alias** – An optional field for a human-friendly collection alias.
* **Title** – A required field that is displayed (along with ID and/or alias) whenever collections are listed. The title is intended to be short an concise.
* **Description** – An optional field where free-form text may be provided to describe the collection.
* **Create Time** – An automatic timestamp field indicating when the collection was created.
* **Update Time** – An automatic timestamp field indicating when the collection was last updated.
* **Owner** – An automatic field containing the identifier of the owner of the collection (either a user or project)

## Projects

SDMS projects enable hierarchical administration of data that is collectively managed by a team of users. Projects are created by a single user who becomes the project owner, and the owner can then select additional users to be either project managers or project members. Project managers have limited privileges compared to the project owner, but still sufficient to handle day-to-day activities regarding member activities. SDMS projects, rather than individual users, own the data and collections associated with the project; however, project members typically have sufficient privileges to create and maintain data on behalf of the project (member permissions are controlled by project owners and/or managers).

While the SDMS allows individual users to share data and collections, and to grant specific permissions to other users, projects are a more concise mechanism to handle these activities. More importantly, projects may be configured with their own data allocations distinct form the project owner’s allocation(s) - enabling finer-grained local data policy enforcement.

## Data Sharing and Permissions

The SDMS defines a set of permissions that can be applied to both data records and collections in order to control which actions are allowed for specific users, groups of users, or anyone else (default permissions). Permissions apply to both user- and project- owned data and collections, but owners are exempt from permission restrictions and may perform any action at any time. These permissions are defined as follows:

* **View** – Allows a user to see a data record or collection in a listing and to view basic information such as ID, alias, title, etc.
* **Read Metadata** – Allows a user to view a data records associated user-defined metadata.
* **Read Raw Data** – Allows a user to read (download) a data records associated raw data.
* **Write Metadata** – Allows a user to update or replace a data records associated user-defined metadata.
* **Write Raw Data** – Allows a user to write (upload) a data records associated raw data.
* **Administration** – Allows a user to perform admin functions on a data record or collection, such as editing system-defined metadata, setting access controls, or deleting the record or collection.

If there is a need to define the same set of permissions for more than a few users, the SDMS provides a group management feature that allows permissions to be set collectively rather than individual users. Any number of groups may be defined, and any combination of user, group, and default permissions may be assigned to a data record or collection.

SDMS projects have a special pre-defined group called “members” containing the project member users. This group is always associated with the root collection of the project and may be used by the project owner or manager(s) to set default project member permissions.

Permissions set on a specific data record only apply to that record; however, collections have two sets of permissions: one for the collection itself, and another for inherited permissions that apply to items linked to the collection. Inherited permissions are combined, as a union, with any permissions set on linked items, and these combined permissions will be passed through as inherited permission for any nested collections, recursively.

## Data Dissemination and Annotations

The SDMS supports data dissemination through the use of a publish-subscribe mechanism that distributes specific data “events” (such as creation, updates, deprecation, tag changes, annotations, etc.) when they occur to interested users. These events are distributed through the SDMS interfaces and, optionally, via email. These notifications can be used to download updated raw data to local caches, and/or trigger data processing activities. When the SDMS is used to capture provenance information, these data events will propagate to dependent data records allowing downstream users to assess impact and take any appropriate actions.

Annotations are a special data events that enable bi-directional communication between data producers and data consumers, and are used to announce potential errors or other important information. Annotations can be used by data producers to issue warnings or guidance to consumers of the data (including derived data, if linked), or they can be used by data consumers to notify producers of specific potential issues with a given data record (or collection).

Annotations include a title, type (error, warning, deprecation, or general information), description, author, and status. When an annotation is created by a data consumer, a communication log will be created to capture all exchanges between the annotation author and the owner of the annotated data record. For error and warning annotations, the annotation will eventually be resolved by the data record owner and the annotation will be removed (and a notification sent of the resolution to all consumers).

## Data Taxonomies

Data records may optionally be associated with a data “topic”. Topics are similar to tags in that they are simply a word or label, but, unlike tags, topics are hierarchical. Topic hierarchy is defined by using a fully-qualified dotted topic format where the most general top is specified first, with more specific topics following. For example, given the full topic “science.energy.fusion”, “science” is the most general topics, and “fusion” is the most specific. In this example, “fusion” is not necessarily the most specific topic that is defined over all data records - there could be other data records that have deeper topics, such as “science.energy.fusion.tokamak”.

As topics are defined or changed for data records within the SDMS, a global topic hierarchy is constructed and maintained. Each referenced topic segment becomes a level in the hierarchy, with all data records that reference the same topic sharing the same topic hierarchy. If an existing topic is no longer referenced, that topic is removed form the global topic hierarchy. This activity results in community-driven topic hierarchies, or data taxonomies. The SDMS does not enforce policies or arbitrate conflicts within the resulting taxonomic structure – these matters must be handled outside of the SDMS.

The SDMS global topic hierarchy is available to be browsed or searched by all SDMS users; however, linking data to the topic hierarchy does not ensure that other users can view or access that data – data record owners must still set permissions appropriately (for public read-only access, default or inherited permissions should be set to VIEW, READ\_RAW\_DATA, and READ\_METADATA).

Data Repositories and Allocations

New SDMS users will be able to access the SDMS Web Portal and Command-Line Interface (CLI) in order to search for and download existing (public) data sets; however, they will not be able to create their own data sets until they are granted a storage allocation on one or more SDMS Data Repositories. The SDMS itself does not determine which users are granted allocations; this is determined by individual SDMS data repository administrators. Typically, users with systems accounts at one or more SDMS member facilities will be automatically granted allocations within these facilities (based on local account and/or data policies); however, it also possible for users to explicitly request allocations at specific facilities (as a collaborator of a another user there, for example).

SDMS projects can either utilize an existing allocation belonging to the owner of the project, or they can be granted their own distinct allocation.

## Data Search

The SDMS supports full-text and structured metadata searches for locating data records. Search queries are a combination of search scope, full-text expressions, and metadata filters. Scopes define which segments of the SDMS database will be searched (i.e. personal data, project data, specific collections, etc.). Full-text expressions can be used to find word- or phrase-matches within the title, description, or keywords fields of a data record. Metadata filters are Boolean expressions that can be used to evaluate and compare values of within a data records structured metadata (a variety of mathematical functions are supported). One or more scopes must be specified with a query, but a query may contain only a full-text expression, or only a metadata filter, or both.

### Full-Text Expressions

Full text expressions consist of individual words and/or quoted phrases, separated by spaces, that are searched within the title, description, and/or keywords fields. By default, full text searches are inclusive and will return any record that contains one or more of the specified words and/or phrases within any of the full-text indexed fields (title, description, and keywords). All punctuation is ignored and the order in which words or phrases are specified in the expression does not affect search results.

If more specificity is desired, the expression may include modifiers to define which fields will should be searched and whether individual words must, or must not, be matched, as follows:

* “title:” – evaluates any following words and phrases to title field only
* “desc:” – evaluates any following words and phrases to description field only
* “keyw:” – evaluates any following words and phrases to keywords field only
* “+” – a prefix that requires a word, phrase, or field to match.
* “-” – a prefix that excludes a word, phrase, or field from matching.

If any field specifiers are used (title:, desc:, or keyw:), then any unspecified fields will be ignored. For clarification, several example full text expressions with expected results are shown below:

* *big data* – records that contain either the word “big” or the word “data” (in any order) in the title, description, or keyword fields.
* *“big data”* – records that contain the exact phrase “big data” in the title, description, or keyword fields.
* *+big +data* – records that contain both the words big and data (in any order) in the title, description, or keyword fields.
* *title: big desc: data* – records that contain the word “big” in the title or the word “data” in the description.
* *title: big -data -desc: “big data”* – records that contain the word “big” but not the word “data” in the title and does not contain the phrase “big data” in the description.

### Metadata Filter Expressions

Any of the system- or user-defined metadata fields of a data record may be used in a filter expression. User- defined fields are nested under a special “md” (for “metadata”) field within the data record and can be accessed by prefixing the field name with “md.”, for example: “md.position.x”. This prefix can be omitted so long as the user-defined field name does not conflict with one of the system-define fields. The system-defined metadata field names of a data record are defined as:

* id – (string) Record identifier
* alias – (string) Record alias
* title – (string) Title
* keyw – (string) List of keywords (any separator can be used)
* topic – (string) Topic path (delimited with periods)
* owner – (string) Record owner ID
* size – (numeric) Size of raw data, in bytes
* ct – (numeric) Record creation timestamp (seconds since 1970)
* ut – (numeric) Record update timestamp (seconds since 1970)
* dt – (numeric) Raw data (upload) timestamp (seconds since 1970)
* locked – (Boolean) Record locked flag

*Note that the record description field is not allowed to be used in filter expressions – it can only be used in full-text expressions.*

The operators allowed in filter expressions include:

* = or == (exact equality)
* != (not equals)
* <, <=, >, >=
* ~= (regular expression)
* ? (pattern match, “\_” matches a single character, “%” matches any number of characters)
* in – value is contained in an array field
* && - Boolean AND
* || - Boolean OR
* ! - Boolean NOT (prefix)

Parenthesis may be used to group expressions that are then combined with Boolean operators. A field name used without an operator is implicitly converted to a Boolean value based on automatic conversion rules (i.e. x != 0 is equivalent to true, false otherwise).

Example filter expressions:

* (pos.x >= 5 && pos.x <= 10) || (vel.z < 0)
* title ? “%simulation%” && topic = “climate.atlantic.hurricane”
* sin(par.a) > pow(par.b,2.5)

Note: There are currently two known complications with search queries caused by integrated third-party software:

* Floating point numbers with an absolute value less than 1 must include a leading zero.
* Less-than and less-than-or-equal comparisons are non-symmetric for non-numeric data types. This affects records that do not contain a numeric field being evaluated with an expression like “x <= 10”. If “x” is not defined, it treated as a “NULL” value, and “NULL” is defined as being less than all other values – regardless of type. This results in unexpected records being returned as results. The work-around is to include “and x != NULL” for all fields “x” that are being evaluated.

### Search Scope

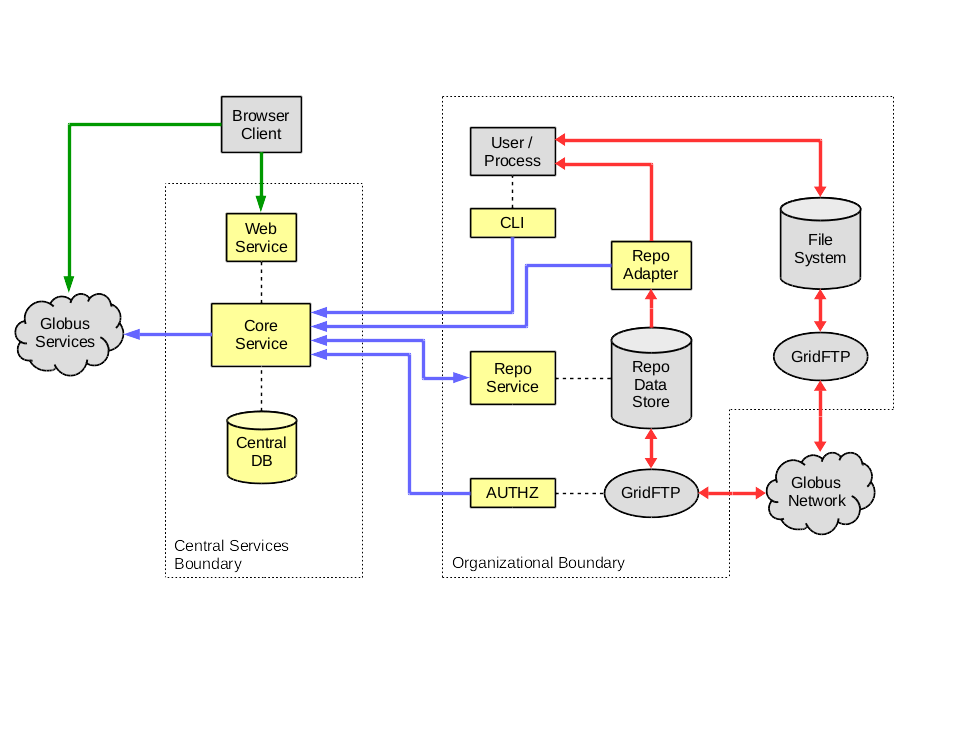
The search scope parameter limits the extent of a search to records that have specific relationships to the user running the query. This is not a typical feature in document search systems because it relies on relational information within the database (in this case implemented as a graph and dependent on a graph schema). The majority of data records in the SDMS will not be accessible by a given random user; therefore, search scope limits query results to records that can be viewed and/or accessed. Search scopes also provide a query performance benefit by avoiding the evaluation of invalid candidate data records.

The SDMS provides a number of built-in search scopes, listed below. Any or all of these scopes may be used in a given search.

1. “My Data” – All data owned by the active user
2. “My Projects” – All data within projects owned by the active user
3. “Other Projects” – All data within projects either managed by the active user, or where the active user is a project member.
4. “Shared Data” – All data that has been directly shared with the active user by other users and projects that the active user would not normally have access to.
5. A custom scope defined by the active user that can include:
   1. Data owned by a specific project
   2. Data owned and shared by a specific user
   3. Data linked to a specific collection (recursive)
   4. Data associated with a specific topic (recursive)

# Architecture

The SDMS consists of a network of geographically distributed data storage systems integrated via a number of custom services and utilities. Globus and GridFTP are enabling technologies providing secure, high performance data transfers, federated user identities, and user credential management. Data storage systems are locally administered file systems or object stores that can be accessed locally and are exported via Globus. Various SDMS software services are used to manage and integrate theses storage systems. The SDMS “Core” service is the control system for the SDMS and is the point at which all data are managed and kept synchronized.



Simplified SDMS Architecture

SDMS storage services are configured at each hosted storage system and provide SDMS managed local caching and fine-grained access controls at the file-system level. Users, scripts, and applications can utilize the SDMS to provision or capture raw data from within a data or compute environment via the client tools which consist of a command-line interface (CLI) and two libraries (C/C++ and python). The client tools are also used to configure local security credentials for non-interactive use cases. The SDMS does not rely on facility managed user credentials, and instead installs system generated credentials derived from the Globus authentication processes.

All network communication between various SDMS software components is binary and is encrypted using modern elliptic-curve cryptographic algorithms, and the web application utilizes TLS (HTTPS) for all service interactions.

## SDMS System Components

### Core Service

SDMS core services provides central data record and metadata indexing and storage (without raw data), and utilizes Globus federated identities and security credentials to manage (raw) data transfers to/from storage systems hosted by member facilities. The core service is written in C++ and has a number of interfaces to support the web portal, client access via the CLI, and for other trusted SDMS system components.

### Core Database

The core database stores all SDMS data except for raw data. Currently, ArangoDB is being used as the database technology due to offering required features such as multi-model data storage (relational, document, and graph), clustering capability, and a strong microservice architecture. An SDMS-specific micro service is utilized to isolate other SDMS components from complex database query and graph traversal logic.

### Web Service (Portal)

The web service hosts a web application that provides an interactive graphical user interface that can be used to easily access all SDMS features, including organizing data with custom collection hierarchies, tags, and metadata, managing collectively-owned/generated data via projects, and event-driven data dissemination. The web service also hosts Globus-specific OAuth services necessary for user authentication and registration.

The web service is currently implemented in node.js and communicates with the core service via a private network (without encryption).

### Data Repository Service

An SDMS Data Repository Service manages one or more facility-local data repositories on an underlying file system. This service primarily handles remote core service requests for data management commands such as status checks and deletions, as well as local data cache management. The repo service does not control access to local files – this is implemented by the Data Repo Adapter.

### GridFTP Authorization (AUTHZ) Module

Every SDMS data repository includes a GridFTP server that handles data transfers into and out of an underlying file system. GridFTP supports a number of standard user authorization mechanisms, however, the SDMS uses a custom AUTHZ module integrated with the GridFTP server to provide custom authorization. On GridFTP file read or write requests, this AUTHZ module communicates with the SDMS core service to determine if the associated Globus user has sufficient permissions for the action. Note that the Globus end-points associated with SDMS data repos are configured for Globus authentication only, and no user proxy certificates are used (the SDMS core service maps user Globus IDs to SDMS users during authorization requests).

### Data Repository Adapter

The SDMS data repo adapter is a file system authorization adapter that exports the underlying file system of a local SDMS data repository (via NFS) to local compute and/or analytics environments within the facility. Normally NFS utilizes POSIX user and group IDs and file system permissions to determine what actions a user may take on a file; however, the SDMS authorization system is more advanced than that provided by POSIX and cannot be mapped directly. Instead, the SDMS data repo adapter maps the POSIX user ID associated with a file action to the associated SDMS user account (if any), then uses the SDMS user ID to perform authorization activities. Note that the data repo adapter and the GridFTP AUTHZ module provide the same user-level authorization functions, but for two different interfaces to the data repo file system.

### Command-Line Interface (CLI)

The SDMS CLI provides access to SDMS features that are needed for interactive or scripted data management activities. On running the CLI, users are authenticated either interactively, or automatically if local SDMS credentials have been installed. Local SDMS user credentials may be installed via the CLI after manual authentication. Once credentials are installed, the CLI may be called from non-interactive scripts, batch jobs, and workflows – so long as the users credentials are accessible. Credentials are typically installed in an SDMS directory in a user’s home directory.

The CLI is written in C++ and uses the SDMS C++ client library to communicate with the SDMS core services.

### Client Libraries

The SDMS client libraries provide a high-level programming interface to all SDMS services. Currently a C++ library is available, and a python wrapper library is under development.

## COTS System Components

Globus Services

Globus Network

GridFTP Service

Repository Data Store

Accessible File System(s)

# Use Cases

NOTE: This should be a separate document (probably a spreadsheet) with detailed information for every supported use case (interface, required roles/permissions, state information, inputs, outputs, development status, etc.).

This section defines the individual use cases of the SDMS that will be used for verification of the software implementation. The uses cases are organized in various categories and each indicates which user role and software interface it applies to (default is all roles and all interfaces). The software interfaces include the web portal (Portal) and the command-line interface (CLI), and user roles are defined as:

* **User** – The default role of a user with no special privileges.
* **Owner** – A user who owns a system entity (data record, collection, project, etc.) associated with a given use case.
* **Project Manager** – A user that has been assigned project management privileges for a specific project.
* **Project Member** – A user that has been added to a project as a member.
* **Repository Manager** – A user that can update specific data repository configuration settings and allocations.
* **System Administrator** – A user with all privileges.

Note that in some use cases, user roles are not sufficient to define the required privileges, and in these cases the specific access rules or permissions will be described instead.

* Account Management
  + Register for a new SDMS account (portal)
  + View account settings (portal)
  + Update account settings (portal)
  + Delete account (portal)
  + Install security credentials within a specific environment (CLI)
  + Revoke security credentials for all environment
  + Revoke security credentials for a specific environment
* Data Management (applies to personally owned data)
  + List existing data records by owner (user or project)
  + Create a new data record
  + View, update, and delete an existing data record by ID or alias
  + Upload and download raw data for an existing data record by ID or alias
  + Lock and unlock an existing data record by ID or alias
  + Move an existing data record to a different repository (owner only)
  + Transfer ownership of data record to a different user or project (raw data transferred to allocation of receiving user or project). (owner only)
* Collection Management (owner or admin permission required)
  + List existing collections by owner (user or project)
  + Create a new collection
  + View, update, and delete an existing collection by ID or alias
  + List items (data records and/or collections) linked to an existing collection
  + Add/remove items to/from an existing collection
* Project Management
  + List existing projects by owner, manager, or project member
  + Create a new project
  + Update an existing project (owner only)
  + View an existing project
  + Delete an existing project (owner only)
  + Add/remove members to/from an existing project (owner and managers only)
  + Add/remove managers to/from an existing project (owner only)
* Access Control (applies to data and collections)
  + View access control settings for an existing item (data record or collection)
  + Set default permissions for an existing item
  + Add/remove user-specific permissions for an existing item
  + Add/remove group-specific permissions for an existing item
  + List existing groups associated with owner (user or project)
  + View an existing group
  + Update an existing group
  + Add/remove members to/from existing group
* Shared Data
  + List data records and collections shared by specific users
  + List data records and collections shared by specific projects
* Topic Browsing
  + List data records by topic hierarchy
* Data Search
  + List data records that match a user-specified query
  + Save a user-specified query for later use
  + Load and execute a saved query
* Favorites - TBD
* Tagging - TBD
* Annotations - TBD
* Dissemination - TBD
* Data Repository Management (repo admins only)
  + List administered data repositories
  + View/update administered data repository settings
  + Add, update, and delete user allocations of a data repository
  + View user allocation statistics for a data repository
* System Administration (system administrator role only)
  + Disable a user account
  + Enable a user account
  + Add new data repository record
  + Add/remove administrator users to/from data repository records
  + Delete existing data repository record

# High-Level Design

Database Schema

* User
  + Name – full name (from Globus ID)
  + Password – SDMS password
  + Email – current e-mail (from Globus ID)
  + Globus IDs – linked by ident edge to uuid table
  + System IDs – linked by ident edges to accn table
  + Allocations – link(s) to alloc table with allocation size and usage
* Data Record
  + ID – system defined
  + Alias – optional user defined. Prefixed with u:<user\_id>: for users or p:<proj\_id>: for projects.
  + Owner – link to user table
  + Title – text
  + Description – text
  + Keywords – text (any delimiter accepted)
  + Topic – optional link to “topics” table
  + Locked – flag indicating read/write lock (true if locked)
  + Public – flag indicating public access enabled (TBD)
  + Data Size – size of data in bytes (0 if data not uploaded)
  + Data Repo – link to data repository if data uploaded
  + Create Time – timestamp of record creation
  + Update Time – timestamp of record update
  + Data Time – timestamp of raw data upload
  + Metadata – structured user-defined metadata in JSON format
* Collection
  + ID – system defined
  + Alias – optional user defined. Prefixed with u:<user\_id>: for users or p:<proj\_id>: for projects.
  + Owner – link to user table
  + Title – text
  + Description – text
  + Create Time – timestamp of collection creation
  + Update Time – timestamp of collection update
* Project
  + ID – user defined (prefixed with “p/”)
  + Owner – link to user table
  + Title – text
  + Description – text
  + Admins – link(s) to user table for project manager roles
  + Create Time – timestamp of project creation
  + Update Time – timestamp of project update
  + Allocations – link(s) to alloc table with allocation size and usage
  + Sub-Allocation – ID of owner allocation with sub-allocation size and usage. Sub-allocation and normal allocations are mutually exclusive.
* Group
  + ID – composite of user ID and group ID
  + Group ID - user defined
  + Owner – link to user table
  + Title – text
  + Description – text
  + Members – link(s) to user table
* Data Repository
  + ID – user defined
  + Title – text
  + Description – text
  + Domain – text (alphanumeric only)
  + Capacity – size of storage in bytes
  + Export path – NFS mount path for domain access
  + Admins – link(s) to user table
  + Allocations – links to user table with allocation size and usage
* Transfer Record
  + Mode – upload or download
  + Status – initial, active, success, error
  + Data ID – ID of data being transferred
  + User ID – ID of user that initiated transfer
  + Start Time – timestamp of transfer init
  + Update Time – timestamp of last status update
  + Repo Path – Globus path to data in repository
  + Local Path – Globus path to data other transfer point
  + Task ID – Globus transfer task ID

# System Concurrency

User operations can be categorized as either discrete or compound. Discrete operations are intrinsically atomic and, by design, must be able to complete within a relatively short duration (milliseconds to a few seconds). Compound operations intrinsically lack atomicity because they involve multiple system components and/or multiple discrete steps that a common synchronization and recovery mechanism (i.e. no transactions). The core service defines compound operations as “tasks” and includes a mechanism (the Task Manager) to ensure that tasks are durable, consistent, and effectively atomic.

* Raw Data Upload/Download – Involves initiating a Globus transfer and verification of access permission when transfer starts.
  + Record deletion – invalidates operation
  + Parent collection deletion – may invalidate ACLs on record(s)
  + Raw data deletion – invalidates raw data transfer
  + Changes to ACLs – may cause transfer to fail
  + Change of allocation or ownership – invalidates raw data transfer and may cause transfer to fail due to ACL changes
* Raw Data Deletion – Triggered by system when a record with raw data is updated to “externally published” and raw data needs to be deleted from associated allocation.
  + Record deletion – invalidates operation
  + Change of allocation or ownership – invalidates operation (location of data changed)
* Record Allocation Change - Involves an owner or system administrator changing record allocation to a different repository. Raw data must be transferred to a new repository and allocation. Typically initiated due to running out of space on a repository.
  + Record deletion – invalidates operation
  + Raw data deletion – invalidates raw data transfer
  + Changes to owner allocation(s) – May invalidate transfer target
* Record Ownership Change – Involves an owner, creator, or system administrator changing record ownership to a different user and allocation. Raw data must be transferred to a new repository and allocation. Typically owner would be a facility project account and creator would initiate change to move data from the facility to a home institute repository.
  + Record deletion
  + Parent collection deletion
  + Raw data deletion
  + Changes to ACLs
  + Change of allocation
* Record Deletion – Involves deletion of raw data on data repository
* Project/User Deletion - Involves deletion of raw data on all associated data repositories.
* Allocation Creation/Deletion – Involves creation/deletion of user/project directories on associated data repository.

Priorities