Microboone SAM Metadata Proposal

B. Carls, E. Church, H. Greenlee, M. Kirby, Z. Pavlovic, S. Wolbers

April 26, 2013

Contents

1 SAM Overview

Sam is primarily a file catalog of files belonging to a particular experiment. The file catalog is implemented as a physical database somewhere. Users do not (usually) interact with the database directly, but rather client programs send requests to an http(s) server called a samweb server. The samweb server responds to client requests,

which responses may include returning information extracted from the database. The samweb server can also update the sam database.

The samweb server interface supports its own query language (not SQL) for identifying collections of files. Users are allowed to record and name queries permanently in the sam database. Such a memorized query is called a *dataset definition*. A samweb server can be requested to execute a query (whether memorized in the form of a dataset definition or not) and return a list of files. A samweb server can likewise be requested to memorize the list of files obtained as a result of a query, which list is called a *snapshot*. Finally, a snapshot can be used to define a *project*, which consists of a set of files that will be scheduled for delivery to a set of worker jobs.

1.1 SAM Resources

There are several Fermilab redmine sites that contain useful information about sam, and related products. The Fermilab redmine home page is

`https://cdcvs.fnal.gov/redmine`.

A full listing or Fermilab redmine projects and subprojects can be found at

`https://cdcvs.fnal.gov/redmine/projects`.

Specific project redmine pages have urls like

`https://cdcvs.fnal.gov/redmine/projects/`*project-name*,

where *project-name* is the name of a redmine project or subproject. Some redmine projects that are relevant for sam are as follows.

- `sam-main` - Main sam project page.
- `sam-web` - Contains information about metadata and sam query language.
- `sam-web-client` - Command line and python clients.
- `filetransferservice` - File transfer service.
- `ifdhc` - Main page for data handling tools.
- `ifdh-art` - Art data-handling interface (art sam client).

Note that file transfer service and data handling tools are not part of sam per se, but the do interact with sam. They are also outside the scope of this document, except they are listed here for informational purposes.

2 SAM Metadata

Sam associates various kinds of information with files, which information collectively are called sam metadata. The primary purpose of sam metadata is to be used in constructing queries that are part of dataset definitions, and secondarily as a repository for information that may be of interest for other purposes.

2.1 SAM Database

The sam database is designed to have a fixed schema that is the same for every experiment (different experiments will have their own database instances). The sam design does include a provision for adding experiment-specific metadata (see Sec. 2.4), but the underlying database schema doesn't change. The sam database is not a good substitute for having experiment-specific databases with schemas defined for particular experiment-specific purposes (such as trigger database, runs database, Monte Carlo database, etc.).

2.2 SAM Metadata Workshop

A sam metadata workshop was held on Feb. 5, 2013, which was attended by most of the coauthors of this document. A useful introductory talk was presented by Robert Illingworth, which can be found on the `sam-main` redmine documents page.[1]

2.3 Predefined Metadata Fields

A list of predefined metadata fields can be found in Robert Illingworth's metadata workshop talk [1], as well as in the `sam-web` redmine wiki [2]. A list of the most relevant predefined metadata fields is reproduced in Table 1.

In the following sections, we will give further explanations and suggestions regarding some of the fields listed in Table 1.

2.3.1 File Names and File Ids.

Files in the sam database are uniquely identified by either a file id. (an integer) or a file name (a string). In database terms, the file id. is the primary key, and the file name is a unique key. The file name is assigned by the user and can basically be anything. The file id. is generated and used internally by sam. However, users can discover the file ids. of files in sam and use them for queries or sam programming.

The sam file name corresponds to the filename part of a unix-like path (i.e. the part after the final "/"). The sam database does not include any organizational concept like directories. That is, sam metadata is relational rather than hierarchical in nature. People sometimes organize files in a directory tree with duplicate file names, where files are distinguished by being in different directories. This approach does not work with sam, because the filenames themselves are required to be unique.

People sometimes try to invent file naming conventions such that the file name effectively encodes lots of information about how the file was produced (making sometimes for very long filenames). Within reason, there is nothing wrong with this approach (although excessively long file names have been known to cause problems by exceeding path length limits on some systems), file names should not be used as a substitute for having proper metadata. It is poor practice to use the file name in sam queries (if this becomes necessary, that is evidence that the metadata is inadequate).

Table 1: Predefined metadata fields.

Name	Type	Required?	Predefined Values?	Description
file_id	integer	yes	no	File id.
file_name	string	yes	no	File name
file_size	integer	yes	no	File size in bytes
file_type	string	yes	yes	File type
file_format	string	no	yes	File format
data_tier	string	no	yes	Data tier
group	string	no	yes	Group
crc	struct	no		File checksum
crc.crc_value			no	Checksum value
crc.crc_type	integer		yes	Checksum type
application	struct	no		Application
application.family	string		no	Application family
application.name	string		no	Application name
application.version	string		no	Application version
parents	struct array	no	no	List of parent files
parent.file_name	string	no	no	Parent file name
parent.file_id	string	no	no	Parent file id.
event_count	integer	no	no	Number of events
first_event	integer	no	no	First event number
last_event	integer	no	no	Last event number
start_time	date	no	no	File start time
end_time	date	no	no	File end time
data_stream	string	no	yes	Stream
runs	struct array	no		List of runs
run.run_number	integer		no	Run number
run.run_type	string		yes	Run type

Since sam file names are required to be unique, some thought needs to be given in advance to how this uniqueness will be achieved in practice. Any deterministic algorithm for generating file names has a risk of generating duplicates (e.g. if a particular job needs to be rerun, etc.). Therefore, it is good practice to include guaranteed-unique elements in files destined for sam, such as timestamps or job ids.

2.3.2 File Type, File Format, Data Tier

The file format describes the physical format of a file. The file type is used to give a higher level description of what the file is intended to be used for. The data tier represents the stage in a typical processing chain.

Likely values for these fields were discussed by the coauthors in the breakout session of the metadata workshop. Some suggested values are listed in Table 2.

2.3.3 Group

The group field is intended to stand for a group within an experiment, rather than the whole experiment. For example, the group field could be used by a particular physics group to label its particular Monte Carlo files. However, initially it will be sufficient to have one catch-all group for the whole experiment (see Table 2).

2.3.4 File Size and CRC

The file size and crc fields are used by sam and the file transfer service to protect against data corruption. The terms "crc" and "checksum" mean the same thing, and are used interchangably in sam documentation. Although sam allows different crc types, sam and the file transfer service internally use a type of crc called "`adler 32 crc type`," and there is no reason to use any other crc type.

Users can choose not to concern themselves with file sizes and crc's at all, and leave the setting and checking of these fields to sam and the file transfer service, which will automatically set or check (if previously set) size and crc each time a file is transferred. Or, users can gain additional protection from file corruption by calculating these values as early as possible (preferably, on the worker node that first creates the file, before the file is transferred over any network). The sam line mode client helpfully provides a subcommand for calculating the checksum called "`samweb file-checksum`."

2.3.5 Application Information

The program that created the file is identified in sam metadata by a 3-tuple of strings which represent the application (family, name, version). Sam does not require these fields to be predefined in the database, so they can be anything the user chooses.

In the case of art framework programs, the application family will always be "`art`." Likewise, art framework programs will usually use the general purpose larsoft executable "`lar`." In this case, rather than storing the application name as simply "`lar`," which doesn't convey much new information, we will store the application

Table 2: Suggested values for some enumerated metadata fields.

Field	Values
file_type	data mc unknown
file_format	root binaryraw-compressed binaryraw-uncompressed tar text unknown
data_tier	generated simulated detector-simulated raw reconstructed root-tuple root-histogram unknown
group	uboone
crc_type	adler 32 crc type
data_stream	all supernova
run_type	physics special calibration laser cosmic test unknown

name as the process name parameter stored in the fcl job file, or specified on the command line using option "`--process-name`." Of course, there may be special cases (including the daq system) where some executable other than `lar` is used. In these cases, the application name should reflect the name of the actual executable.

In the case of art programs, the combination of the appliation name and the data tier will give a general idea of the purpose and configuration of the program, but not full details. We will definitely want to have more information in the metadata about how art programs were configured via their job fcl files. Our proposal is to add this additional information using experiment-specific metadata (see Sec. 2.4.5).

2.3.6 Parentage Information

The sam metadata contains a field for identifying the parent files (there can be more than one) for each file. At the metadata workshop, the minerva experiment reported that they use this field to record the raw data parents. However, it was certainly the intention of the sam designers that this field should be used to store the immediate parents, and we propose to do that.

2.3.7 Data Stream

As its name implies, this metadata field is intended to identify different output streams that are produced by the same program. The earliest opportunity for streaming is at the daq level, which may generate multiple daq streams based on trigger information. For example, we might choose to write booster and numi beam events to different daq streams. We propose that the predefined `data_stream` be used exclusively for such daq streams, and offline streams be implemented using experiment-specific metadata (see Sec. 2.4.7).

Regarding specific daq streams, see Table 2. We will probably want a catch-all "all" stream. We will definitely need a separate daq stream for supernova data. We may want to define additional daq streams later.

2.3.8 Run Information

In the sam metadata, runs are identified by a 2-tuple consisting of a run number (an integer) and a run type (a string). For data, the run number and run type will normally be assigned by the daq system, and carried forward to all descendant files. Some proposed run types are listed in Table 2. Files can have multiple runs.

2.4 Experiment-Specific Metadata Fields

Sam metadata is designed to be extensible by the addition of experiment-specific metadata fields, which are called parameters in sam documentation. These experiment-specific metadata, or parameters, are represented in metadata definitions and queries as a key-value pairs of the form *category.name* = *value*, where *category*, *name* and *value* are whatever the experiment chooses them to be (other than predefined categories and names, such as those listed in Table 1). Only scalar data types

Table 3: Beam parameters.

Parameter	Type	Values
bnb.proton_energy	float	
bnb.target_number	integer	
bnb.horn_number	integer	
bnb.horn_current	float	
bnb.horn_polarity	string	forward/reverse
numi.proton_energy	float	
numi.target_number	integer	
numi.horn1_number	integer	
numi.horn1_current	float	
numi.horn1_polarity	string	forward/reverse
numi.horn2_number	integer	
numi.horn2_current	float	
numi.horn2_polarity	string	forward/reverse

Table 4: Detector parameters.

Parameter	Type	Values
detector.cathode_voltage	float	
detector.plane0_voltage	float	
detector.plane1_voltage	float	
detector.plane2_voltage	float	
detector.pmt	string	on/off

are supported for parameter values (no structs or arrays). Actually, parameter values are represented in the sam database as strings, even if they represent numbers.

2.4.1 Beam Status

Since microboone actually has two neutrino beams, we propose to define two parameter categories called `bnb` and `numi`. Refer to Table 3 for proposed names and values of beam parameters.

2.4.2 Detector Status

Some proposed detector status parameters are listed in Table 4.

Table 5: Trigger parameters.

Parameter	Type
`trigger.name`	string
`trigger.version`	string

Table 6: Online parameters.

Parameter	Type	Value
`online.data_quality`	string	good/bad

2.4.3 Trigger Configuration

There is no possibility to capture all of the details of the trigger configuration in the sam metadata. Therefore, we propose to record the trigger configuration name and version, which can be used to cross reference an external trigger database. Here, the trigger configuration name refers to full suite of triggers that is in effect for any given run, rather than a specific trigger.

We are assuming that the trigger configuration name and version will be unique for any single run. Since sam allows files to be associated with multiple runs, this parameter will only make sense for files that have one run (especially for raw data).

2.4.4 Additional Online Information

This category is intended as a catch-all for everything related to the online or daq configuration, besides beam, detector, and trigger configuration. The source for this information will be the daq system or the shifters. Refer to Table 6.

2.4.5 FCL Configuration

In addition to the application name and version, art programs are fully specified by their fcl configuration. Since the fcl configuration is much too complex to be fully captured by sam metadata, we propose to treat the fcl configuration similarly as the trigger configuration, by specifying the fcl name and version, which name and version can be used to reference an external fcl database (see Table 7). In fact, microboone already has the equivalent of an fcl database, in the form of the ubfcl ups product, which is versioned and backed up in the ubooneoffline svn repository. Using the ubfcl product, the fcl name and version should be fully adequate to determine the exact fcl configuration. We should require that files in sam have their fcl configurations in the ubfcl svn repository.

Table 7: Fcl parameters.

Parameter	Type
`fcl.name`	string
`fcl.version`	string

Table 8: Project parameters.

Parameter	Type
`ub_project.name`	string
`ub_project stage`	string
`ub_project.version`	string

2.4.6 Project Configuration

Sometimes it will be necessary to record not just the configuration of the most recent program that produced the file (the fcl configuration), but the configuration of an entire processing chain. Typical examples of this are MC files, where the generator configuration is relevant at later processing stages. Therefore we propose to include a project parameter (see Table 8). As for similar cases, the project parameter values will be used to cross reference an external project database (for example, the ubxml product in the ubooneoffline repository).

2.4.7 Offline Streaming and Filtering

At some point, we will certainly want to create sparse datasets for various specialized purposes, like particular physics analyses. For many purposes, the combination of application and fcl configuration will be adequate to specify how such filtering was done. To handle, in addition, the case where the same filtering program (or any program) produces multiple output files, we propose to define a filter name parameter (see Table 9).

References

[1] R. Illingworth, `https://cdcvs.fnal.gov/redmine/documents/594`.

Table 9: Filter parameters.

Parameter	Type
`filter.name`	string

[2] https://cdcvs.fnal.gov/redmine/projects/sam-web/wiki/Metadata_format.