

# Requirements for an ART file “User” Database

Robert Illingworth (SAM), Rob Kutschke (Mu2e), Adam Lyon (g-2, SAM), Andrew Norman (NOvA, SAM), Brian Rebel (NOvA, LArSoft)

We need an interface to a file’s internal SQLite database for adding “user” information about a file (e.g. for SAM metadata and bookkeeping information). We need to be able to add and update the user DB in new, unsaved files and query the user DB in all files. We do not need, and in fact should prohibit, the ability to alter the user DB for files that have been saved previously – just as the data contents of a file open for reading cannot be changed, the user DB is frozen as well.

## Requirement - User Data Structure

We imagine an schema for a user database containing category-key-value triplets (e.g. two-tired key-value data with the category acting as a namespace for keys). The “schema” is as follows:

- Category - string
- Key - string
- Value - union(string, double, int64).

Some examples (all are category, key, value):

- sam, stream, “antinu”
- sam, nEvents, 7000
- sam, parentFiles, “xlf333.dat, zye993.dat, aab121.dat”
- bookkeeping, protonsOnTarget, 24.34e5
- bookkeeping, triggersOr, 0xffe3

This data is at the file level (so this is not meant for event, conditions, or run data more finely grained than a file).

## Requirement - Separate user DB - one for each file, user DB accessible while file is open

We require that the user DBs for each file be separate (that is we need the ability to query and or set values in user DB corresponding to a particular open file). The user DB for a file open for read would be extracted just after the file open (e.g. before any ART run or event processing). The user DB for a file open for write would be created empty upon file open and then written to the file just prior to file close (e.g. after any ART run or event processing).

We assume that there is only one “instance” of the user DB in a file (e.g. we do not “copy” a user DB from one file and paste it into another file -- any transferring of user data from one file to another would occur pragmatically and piecewise using the regular API below).

We require an API for accessing the user DB for a particular file. We require hooks to access this user DB when a file is opened or just before the user DB is written to an output file. We also need to access the user DB for both querying and setting (if file is open for write) during run and event processing.

## Requirement – an API for querying and setting user data

With the API to access the user DB for a particular file, we require the means to query and, where possible, set data within the user DB. We imagine functions of the following form (the following is meant to be pseudocode and not meant to be suggestive of an implementation):

1. Querying a user DB

```
type userDBHandle.getUserData<type>(string categoryLabel,  
                                     string key)
```

```
For example long nEvents = myDataFileUserDB.getUserData<long>("sam",  
                                                             "nEvents");
```

Exceptions should be thrown on error conditions (e.g. category+key combination not present in user DB, wrong type [this behavior should be similar to `parameter.get<>`], user DB inaccessible, etc).

2. For writable user DBs, we need to be able to set values

```
bool userDBHandle.putUserData(string categoryLabel, string key,  
                             <type> value)
```

```
For example, myDataFileUserDB.putUserData("sam", "nEvents", 7000);
```

If the category+key already exists in the user database, then the corresponding value should be overwritten by that passed into the function. If the “put” creates a new DB entry, the function should return “true”. If an old entry is overridden, the function should return “false”. Exceptions should be thrown on errors (e.g. attempting to write to a user DB corresponding to a file open for reading).

3. Extract the list of categories

```
vector<string> userDBHandle.categories();
```

An exception should be thrown if the user DB is inaccessible (e.g. stale user DB handle)

4. Extract the list of keys for a category

```
vector<string> userDBHandle.keys(string category);
```

An exception should be thrown if the user DB is inaccessible (e.g. stale user DB handle)

5. Determine if a category-key exists in the user DB

```
bool userDBHandle.hasKey(string category, string key)
```

An exception should be thrown if the user DB is inaccessible (e.g. stale user DB handle)

The handling of the return vectors in items 3 and 4 (perhaps the user should pass in an empty vector that is filled by the function), is left as an implementation decision.