Technical Note **LIGO-T000050-00 - E** 4/14/00

Requirements for LDAS Database User Interface Tools

P. Shawhan

Distribution of this draft:

LDAS Group; LIGO Scientific Collaboration

This is an internal working note
of the LIGO Project

Contents

1 INTRODUCTION

The LIGO Data Analysis System (LDAS) includes a database to store “metadata” of various types, including catalogs of raw data files, astrophysical and environmental event candidates, a record of the state of each interferometer as a function of time, and summary information about data quality and environmental conditions. These types of information, and the relational database tables which have been implemented to store them, have been described in detail in LIGO-T990101-02, “Table Definitions for LDAS Metadata / Event Database”.

The LDAS software design, described in LIGO-T990001-06, “LIGO Data Analysis System Preliminary Design”, includes several components (referred to as “application-programmer interfaces”, or “APIs”) which work together to provide remote access to the database, format translation, job management, and scripting capabilities. These APIs make up the “LDAS side” of the database interface, and their implementation in software is nearly complete. However, the LDAS preliminary design did not cover the “user side” of the database interface, except to outline the communication protocol between a generalized “user API” and the core LDAS system.

Over the past several months, there has been further thought about how users will interact with the database. This has led to the realization that there should be a set of standardized tools on the “user side” of the interface to facilitate insertion and retrieval, rather than leaving each user to figure it out for himself/herself. There are three distinct situations:

1. Various programs run continuously at each LIGO interferometer site to analyze the data in near-real time. These include the LDAS online search for astrophysical event candidates and numerous online diagnostics and data-monitoring programs to detect transients in auxiliary and environmental channels. All of these programs need to be able to insert information into the database while continuing to run.
2. Offline analysis programs may use information from the database, or generate new information to be inserted, or both. Thus, an offline program should be able to submit a database query and parse the results. A given offline job operates on a fixed amount of data, and will therefore end after a finite length of time, so its output can be written to a single file. While an analysis program is being developed, the output file generally should not actually be submitted to the database, but just checked by a human; on the other hand, a finalized analysis program should be able to submit its output file automatically.
3. A general-purpose graphical user interface is needed to allow users to explore the contents of the database. It should help the user construct appropriate database queries, and display the results in a readable manner as rapidly as possible.

The rest of this document describes the general requirements for interacting with the LDAS database, and then discusses some specific requirements for each of the three situations listed above.

2 GENERAL REQUIREMENTS

All data input to or output from the database will be in the form of a “LIGO lightweight” file. This is an XML (eXtensible Mark-up Language) file with a LIGO-specific document type definition, which has been described in LIGO-T990023-01, “LDAS Lightweight Data Format Specification” (but with “XSIL” to be replaced by “LIGO_LW”). In general, a LIGO_LW file can contain several different types of data objects, but all database input and output will be stored in one or more TABLE objects, with reasonably deterministic object naming and file formatting conventions. This simplifies the task of creating tools for database input/output.

Data is inserted into the database by putting the relevant LIGO_LW file in a directory accessible via http or anonymous ftp and then sending a “putMetaData” command to the LDAS managerAPI, over a network socket, specifying the file location as well as the user’s LDAS username and password. The LDAS username is used to determine whether that user is authorized to insert data into the database, since database additions must be approved by the LIGO Scientific Collaboration (through a procedure to be determined). An email message will be sent to the user when the insertion succeeds (or fails); this information is also written to a log file on the LDAS web server.

Data is retrieved from the database by sending a “getMetaData” command to the LDAS managerAPI, specifying a query expressed in SQL (the native language of the commercial database underlying the LDAS system)[1] and the desired return protocol, e.g. creating a file which is retrievable through the LDAS web server. All registered LDAS usernames will be able to read from any database table. An email message will be sent to the user when handling of the query is complete, indicating the location of the results file; this information is also written to a log file on the LDAS web server.

Section 4 contains some examples of socket communication with the LDAS managerAPI in different languages.

The two main goals of user interface tools are to facilitate the correct formatting and/or parsing of LIGO_LW documents and to streamline communication with the LDAS managerAPI. Specific requirements are discussed in the following sections.

3 REQUIREMENTS FOR SPECIFIC CASES

3.1. Online Event Generation

As mentioned in Section 1, online programs process the data as it is collected and must be able to insert information into the database while continuing to run. This information will often be in the form of events (or “triggers”, conventionally used to describe transients detected in channels other than the gravity-wave signal), although periodic summary information (spectra, statistical quantities for key channels, etc.) will also be produced. The database insertion procedure requires a fair amount of sophistication because of a number of issues:

- It is desirable to buffer events and insert many at the same time, because there is a certain overhead associated with any database transaction. On the other hand, there should be a limit to the amount of time an event may sit in a buffer before being inserted.
- Management of LIGO_LW files is simplified if they are written to a directory created for that purpose, rather than a user’s home directory or the current directory when the program was

1. In the future, special-purpose LDAS commands may be created to submit common queries without using SQL.

started.

- Once written, LIGO_LW files need to be transmitted to LDAS by contacting the managerAPI at the appropriate site. After a file is transmitted, it should probably be archived somewhere or perhaps simply deleted—but in either case, only after the database insertion is known to have succeeded. But an online program should not be expected to wait for the status message from LDAS before continuing to process more data, so a certain degree of asynchronous flow is needed.
- A database downtime should not prevent online programs from generating events; the LIGO_LW files should be kept somewhere, and then inserted when the database is available again.
- Every database entry made by a process is tagged with a unique “process ID” which is assigned when the first information from that process (including its entry in the PROCESS database table) is inserted into the database. Since the LDAS metaDataAPI does not keep a persistent record of the process ID assigned, it must be retrieved from the database and included in all later files submitted by the process. This requires some asynchronous communication.

It is not reasonable to expect the many online programs to all have the sophistication necessary to address the issues listed above. Therefore, all online programs will send event information to an intermediate process which will handle buffering, correct use of process IDs, and all communication with the LDAS managerAPI. For the Data Monitoring Tool (DMT) computers, a program called the “trigger manager” has been written to do this; for the LDAS online (and perhaps offline) event searches, these functions will probably be performed by the eventMonitorAPI.

3.2. Offline Analysis Program Interface

Offline analysis programs need to be able to read from the database—for example, to get a list of locked segments to be analyzed—as well as write to it. Reading normally involves sending a query to the LDAS managerAPI and getting the resulting LIGO_LW file onto a local disk; actually, one can read from any LIGO_LW file containing one or more table objects, e.g. a file written by some other offline analysis program but not ingested into the database. I believe that almost all analyses will just need to loop over the rows in a table, so the LIGO_LW file can be read sequentially rather than building a memory image of the complete document (as is done by true XML parsers). This allows us to read arbitrarily large files without worrying about memory limitations.

The output of an analysis job will normally be a single LIGO_LW file with one or more table objects. It may just be kept on disk for inspection with the graphical user interface and/or input to other analysis programs, or it may be submitted for ingestion into the LDAS database. Because all the information is contained in a single file, ingestion is fairly straightforward and will not require a “trigger manager” or similar process. However, the user will still have to specify which database site to connect to and provide an LDAS username and password with write permission.

“Offline analysis” will be done on many computer systems, not just LDAS computing clusters. Therefore, the sequential input/output interface described above should be included in the LIGO/LSC Algorithm Library (LAL). Thus it will need to be written in C, perhaps with some Tcl helper scripts to simplify communication with LDAS.

3.3. Graphical User Interface

The graphical user interface should display table information in a convenient fashion, whether it is returned by a database query or is from a disk file written by some analysis program. That is, it should be a general-purpose LIGO_LW table viewer. It should provide features to simplify the task of cross-referencing information between database tables known to be related, but should also be able to display unfamiliar tables. Ideally, it should be easy to use even for someone not familiar with the organization of the LDAS database.

The graphical user interface should provide point-and-click methods to construct database queries. Simple queries should be the simplest to construct and execute. Queries will be sent to the LDAS managerAPI, so the user will have to enter his/her LDAS username and password. In the interest of fast turnaround, the program should automatically retrieve the results and display them, circumventing the LDAS email notification method which is better suited for lengthy jobs.

It is highly desirable for the graphical user interface to be usable on any computer system, whether or not other LDAS software is installed. Installation should be reasonably simple, to encourage people to explore the contents of the database.

4 SOCKET COMMUNICATION EXAMPLES

This section contains some code fragments which demonstrate how to communicate with the LDAS managerAPI.

4.1. Tcl

```
;##- Open a socket to the LDAS managerAPI
if { [catch {socket $hostName $port} sockId] } {
    puts "Unable to connect to LDAS manager"
    return ""
}

;##- Send the LDAS command (which was constructed previously)
puts $sockId $ldascmd

;##- Get the response from the managerAPI and print it out
flush $sockId
set jobInfo [read $sockId]
puts $jobInfo

;##- Close the socket connection
close $sockId
```

4.2. perl

```
##- Open a socket to the LDAS managerAPI
use IO::Socket;
$sockId = IO::Socket::INET->new( Proto    => "tcp",
                                 PeerAddr => $hostName,
                                 PeerPort => $port )
    or die "Unable to connect to LDAS manager, aborting";
```

```
##- Send the LDAS command (which was constructed previously)
print $sockId $ldascmd;

##- Get the response from the managerAPI and print it out
@jobInfo = <$sockId>;
print @jobInfo;

##- Close the socket connection
close $sockId;
```