Astronomical Data Analysis Software and Systems XXIII
ASP Conference Series, Vol. 440
K. Summers

Evolution of the LBT Telemetry System

Kellee Summers, Christopher Biddick, Michele D. De La Peña,
Douglas Summers

Large Binocular Telescope Observatory, Tucson, AZ USA

Abstract. The Large Binocular Telescope (LBT) Telescope Control System (TCS) records about 10GB of telemetry data per night. Additionally, the vibration monitoring system records about 9GB of telemetry data per night. Through 2013, we have amassed over 6TB of Hierarchical Data Format (HDF5) files and almost 9TB in a MySQL database of TCS and vibration data. The LBT telemetry system, in its third major revision since 2004, provides the mechanism to capture and store this data.

The telemetry system has evolved from a simple HDF file system with MySQL stream definitions within the TCS, to a separate system using a MySQL database system for the definitions and data, and finally to no database use at all, using HDF5 files.

1. Initial System Design and Implementation

The LBT telemetry system is designed to provide the following: system data for commissioning, failure diagnosis and system tuning; monitoring of key systems for failure mitigation; and access to historical data. The telemetry collection system is required to support sample streams as large as 16kB, at a rate up to 40Hz (Edgin 2012; Edgin & Cushing 2010). Currently, telemetry streams are collected at rates up to 1kHz. From inception, analysis tools were not part of the LBT telemetry system. Third party products were presumed for accessing and analyzing telemetry data.

1.1. HDF Data Files with MySQL Stream Definitions

The first generation telemetry system for the LBT was implemented as a subsystem of the TCS. The data were implemented as sequences of time-tagged values from telescope subsystems to HDF (NCSA 2003) files (De La Peña & Axelrod 2006). As a TCS subsystem, it had a GUI similar to the other TCS GUIs. Telemetry stream types and contents were defined in a MySQL relational database via the GUI. The database also maintained stream status and a catalog of the archived data files. From the GUI the user could start/stop streams from the pre-defined collection, set acquisition parameters, monitor the state of a stream, monitor the state of the telemetry system, and generate graphics via MATLAB's GUI builder, GUIDE.

Only a couple of the TCS subsystems used this system, beginning in February 2004. A few post-processing tools using MATLAB were provided for specific data streams, but nothing generic. This was implemented as a shared library.

Because it was a subsystem of the TCS, the implementation in the TCS subsystems (the telemetry producers) was very minimal. However, this was also the major drawback of the design - only TCS clients could write data, and it was limited to col-

lecting data in the TCS shared memory. This telemetry system was envisioned to be upgraded to allow variables from multiple subsystems to be streamed together, to allow creation of streams on-the-fly, and to have generic graphic display for any stream.

1.2. MySQL Stream Definitions and Data

The goal of the second generation telemetry system was to decouple telemetry stream definitions from the telemetry system (Edgin & Cushing 2010) and telemetry collection from the TCS. The system was implemented over a couple of years beginning in 2007. The design was an API used by the TCS subsystems and the vibration monitoring system, instead of the telemetry system being a subsystem of the TCS.

This system completely re-implemented the previous implementation. The C++/ Java implementation emphasized keeping the data descriptions apart from the telemetry system, so the system did not have to change for new/modified data streams. Telemetry clients built their stream definitions with a series of API calls and registered with the telemetry system for collection.

The implementation used the visitor pattern for telemeter registration to traverse nested hierarchies of unknown depth. The pattern specifies two hierarchies – one for the elements being worked on and one for the visitors. The element hierarchy is the measure classes (bool measure, float measure, etc.) which are derived from metadata. The main visitor in the implementation is the registrar class. The magic of the visitor pattern makes the entire hierarchy registration for a stream happen on a single constructor call to the registrar class.

Telemetry producers are clients - the vibration monitoring system and the TCS subsystems. The API is a shared library, linked into the client's executable.

The new design used a MySQL database for both the telemetry definitions and the data. It is not clear what drove the use of the database for storing the telemetry data. There are no documented requirements for a database. It was likely chosen for ease of accessibility to the recorded data.

In October-2008 all of the TCS subsystems started using this system, recording data to the database. By the beginning of 2013, the database was almost 9TB.

Users accessed the data via a web-based exporter tool written with Google Web Toolkit (GWT) which created comma separated value (CSV) files for import into other data processing packages.

The MySQL database proved too fragile, however. Database corruption was occurring causing discontinuities in the data and downtime when telemetry was not able to be collected. Replication between the mountain and Tucson was particularly problematic, routinely failing. Sometimes it would take days to recover, with telemetry collection down during the recovery.

2. HDF5 Implementation

The third generation telemetry system was created to address robustness issues with the database. When analyzing how to solve the telemetry system problems, we considered several factors which focused us on extricating the database from the existing code:

- The current telemetry system implementation is inordinately complex (over use of nested templates, wrappers, too-many levels of indirection for declarations and definitions) and mostly undocumented.

- We did not have the budget for a more robust database.
- We did not have the budget or manpower to rewrite the system again.
- We wanted maintenance of the system to be simple, including replication.

HDF5 was a natural replacement to the database because it is a simple file interface and there are post-processing interfaces to a wide variety of languages and analysis packages available. Once we decided on HDF5, some of the criteria used to choose the HDF5 Packet Table as the data implementation were:

- It had to support the existing telemetry formats which were "streams" of heterogeneous data.
- We wanted a table format (as opposed to using scaled images to store what was inherently tabular data with each row representing some array).
- The dataset needed to be able to accept streaming data versus pre-prepared tabular information that was just going to be written out to a file.
- Initially, we wanted concurrent access to different datasets in a single HDF5 file. This can be done with a thread-safe version of the HDF5 library. (Although, the system does not currently take advantage of this.)

It was not as easy as just replacing a few low-level classes, but we successfully replaced the database with HDF5 files in the operational observatory environment in just a few months. Additionally, we had to create the logic to build the directory structure based on subsystem names and dates, and make it thread safe. Each TCS subsystem is its own process, spawning separate threads for each stream it records.

The up-to-date releases of the analysis packages read the HDF5 Packet Tables directly. However, for those languages or packages which do not accommodate HDF5 structures directly, we provide a tool that generates CSV files. We modified the HDF5 command-line tool h5dump to generate CSV files specifically for our HDF5 file formats.

Problematic database replication was replaced with simple rsync commands. A web server was created to allow access to the raw files for partner organizations.

3. Future of LBT Telemetry

We have addressed the reliability problems of the LBT telemetry system. The 2013 HDF5 telemetry system has proven very robust in both collecting and replicating the data. Now we need to focus on accessibility and performance. Over the next year, there are several areas to improve.

3.1. Web-based Access

There is an effort underway to better characterize the observatory downtime, instrument overhead, and efficiency. There are at least three known sets of home-grown metrics in use. These metrics are currently captured from various sources, including observing night-log entries. Over the next year, an effort will be made to standardize these metrics and get all the required data into the observatory telemetry system. A standard set of post-processing functions on raw data will be done nightly to provide daily metrics. Web-based access to this metrics data will be required. Additionally, day to day troubleshooting and maintenance planning will be enhanced with engineers having easy access to sets of standard post-processed metrics data.

3.2. Instrument Telemetry

The reasons for facility instruments to provide data to the observatory software infrastructure are varied. Some data is desired for GUI access, some for telemetry, some for alarming and other data for metrics. Currently, there is no consistent interface definition for information that should come from instruments to the software infrastructure, and no instrument data is recorded in the telemetry system.

The telemetry system is built as a standalone C++ library which could be used by any of the instruments to record data. However, rather than use the cumbersome API, it might be easier for instrument teams to just use the HDF5 APIs directly. In the next year, a common interface will be developed and published for instrument teams. The interface will define the required HDF5 format for compatibility with the TCS telemetry, and likely the required directory structure. Specific data items, synchronization, update frequency, etc. will also be defined in the interface, some of which will be instrument-specific. Instrument teams will be free to use the LBT collection library as-is, or write their own implementation of the specified HDF5 telemetry.

3.3. Catalog / Table of Contents

A catalog/table of contents tool would be useful for users. The file structure created by the telemetry system is date and size-based. The system creates a new file if the current file reaches a 100MB size limit, or the date changes. It would be valuable to have some kind of query service to allow users to find what stream has a particular piece of data at a particular time, and where to find the associated files.

3.4. Enhancements

The following are other features that we would like to implement in the telemetry system:

- Historical data from the old database system needs to be extracted and migrated to HDF5 files.
- We should test packing and different HDF file management strategies with the larger streams.
- Real-time access needs improvement - simple, generic graphs to visualize any data by date.
- Finally, this code needs documentation, only the API and exceptions are documented.

References

De La Peña, M. D., & Axelrod, T. 2006, in Astronomical Data Analysis Software and Systems XV, edited by C. Gabriel, C. Arviset, D. Ponz, & E. Solano (San Francisco: ASP), vol. 351 of ASP Conference Series

Edgin, T. 2012, Telemetry Specification, Tech. Rep. CAN 481s345, Large Binocular Telescope Observatory

Edgin, T., & Cushing, N. 2010, in Software and Cyberinfrastructure for Astronomy, 77402, edited by N. M. Radziwill, & A. Bridger (Bellingham: SPIE), vol. 7740 of SPIE Astronomical Telescopes and Instrumentation