

LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY  
- LIGO -  
CALIFORNIA INSTITUTE OF TECHNOLOGY  
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

**Document Type** LIGO-T010010-00 - E 02/05/2001

**The Event Monitor API's  
baseline requirements**

James Kent Blackburn

*Distribution of this document:*

LIGO LDAS Group

This is an internal working document  
of the LIGO Project.

**California Institute of Technology**  
**LIGO Project - MS 18-34**  
**Pasadena CA 91125**  
Phone (818) 395-2129  
Fax (818) 304-9834  
E-mail: info@ligo.caltech.edu

**Massachusetts Institute of Technology**  
**LIGO Project - MS 20B-145**  
**Cambridge, MA 01239**  
Phone (617) 253-4824  
Fax (617) 253-7014  
E-mail: info@ligo.mit.edu

WWW: <http://www.ligo.caltech.edu/>

# The Event Monitor API's *baseline requirements*

*James Kent Blackburn*

California Institute of Technology  
LIGO Data Analysis Group  
February 5, 2001

## I. Introduction

### A. General Description:

1. The eventMonitorAPI is responsible for receiving the composite output results (called events) from search codes executed in the wrapperAPI (see LIGO-T990097-E for details on the wrapperAPI and the output data structures) or optionally from the dataConditionAPI in the event that it becomes a significant source of events (see LIGO-T990002-E), and directing the three individual data structures to the appropriate destinations within LDAS using an interpreted command language:
  - a) The interpreted command language to be used is TCL/TK, which provides a command line, scripting and a graphical interface.
  - b) The TCL/TK commands are extended to support low level system interfaces to the algorithms used to “communicate” the data, as well as provide greater computational performance using C++ code that utilizes the standard TCL C code API library in the form of a TCL/TK package.
2. The eventMonitorAPI's TCL/TK script accesses the eventMonitorAPI.rsc file containing necessary information and configuration resources to extend the command set of the TCL/TK language using the eventMonitorAPI package, which exists as a shared object.
3. The eventMonitorAPI will receive its commands and messages from the managerAPI, reporting back to the managerAPI upon completion of each command. This command completion message will include the incoming identification used by the manager to track completion of sequenced commands being handled by the assistant manager levels of the managerAPI.
4. The eventMonitorAPI will receive its event data in the form of the “Internal LDAS Lightweight Data Format” (ILWD). Event data will be received at the eventMonitorAPI's data sockets (described below). Parsed subsets of event data will be sent directly to “users” of the data, e.g., metadataAPI, dataConditionAPI, lightWeightAPI, or user file space (ftp, http areas in LDAS), in the same ILWD format once fully formed. (NOTE: “users” refers to LDAS API's as well as issuers of LDAS user commands in this context.)
5. The eventMonitorAPI store state information found in the event data to file using the binary LDAS “*Internal Lightweight Data Format*” ILWD format with a index determined by the jobID. This file will be place in a location

## The Event Monitor API's baseline requirements

accessible by the dataConditionAPI. This index will be used by future job requests to access the state information stored in the binary ILWD.

### B. The eventMonitorAPI.tcl Script's Requirements:

1. The eventMonitorAPI.tcl script will provide all the functionality inherited by the genericAPI.tcl script (*i.e. help, logging, operator, jobstate & emergency sockets, etc.*).
2. The eventMonitorAPI.tcl script will report to the managerAPI's receive socket upon completion of each command issued by the managerAPI's assistant manager levels. This involves transmission of a message identifying the specific command completed as coded by the managerAPI (*see LIGO-T980115-E for details*).
3. The eventMonitorAPI.tcl script will validate each command received on the operator, jobstate or emergency socket as appropriate for the eventMonitorAPI to evaluate. This includes validation of commands, command options, encryption keys and managerAPI identification indices.
4. In case of an exception occurrence while processing a command, the eventMonitorAPI.tcl layer will report the exception to the ManagerAPI's *emergency socket* along with the necessary command identification issued by the managerAPI for the specific eventMonitorAPI command.  
**Note:** Once reported to the managerAPI, the appropriate *assistant manager* will terminate the high level command and the userAPI that issued this high level command will be notified of the exception.
5. The eventMonitorAPI.tcl script will initialize the optimal number of rows to insert into each individual LDAS database table by reading these values from the eventMonitorAPI.rsc resource file and passing this information into the C++ layer through a TCL extension provided by the eventMonitorAPI.so

### C. The eventMonitorAPI.so Package Requirements:

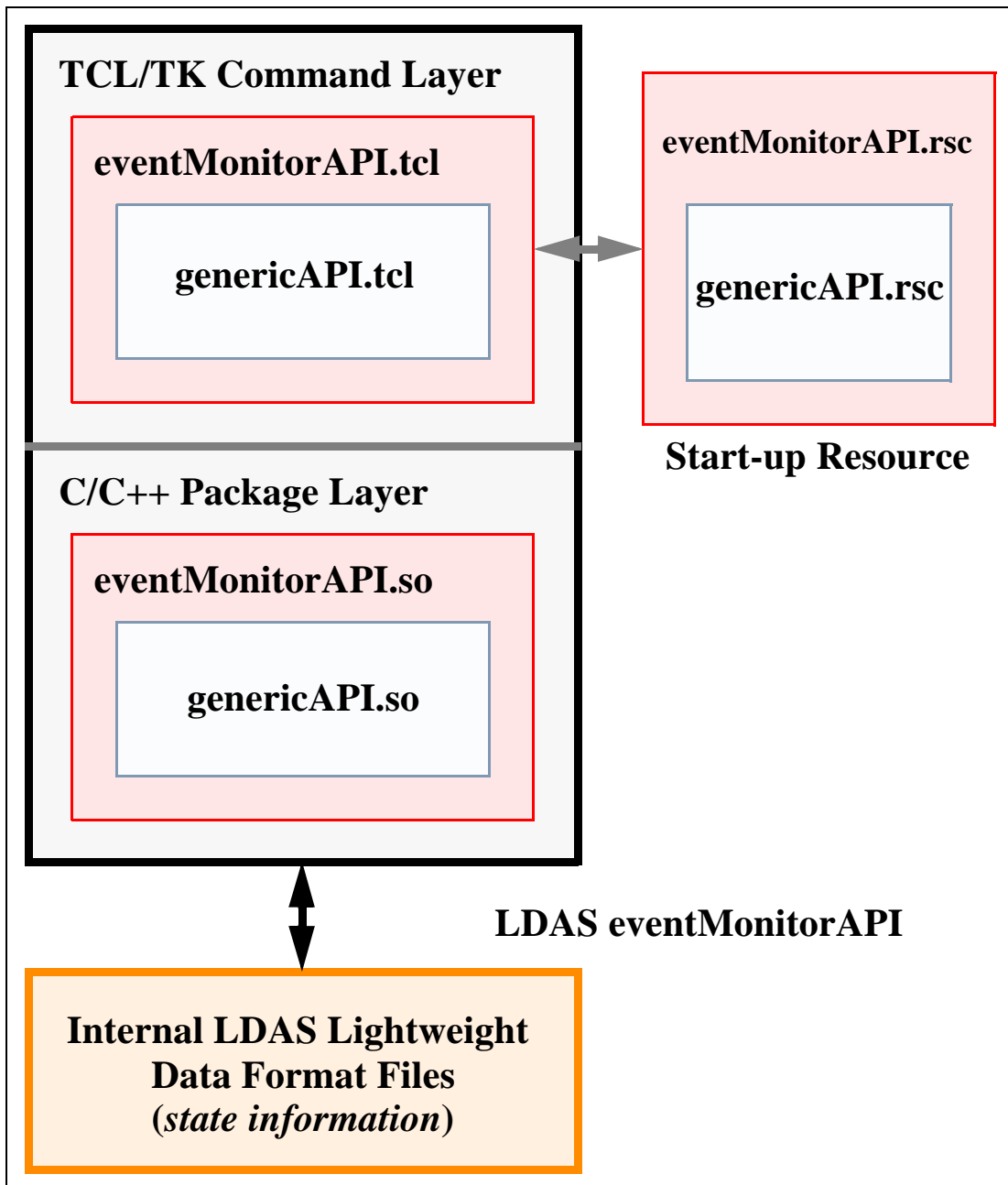
1. The eventMonitorAPI.so package will be developed in C++ using the C language interface to TCL/TK to communicate with the TCL/TK command layer. The wrappers from C++ functionality to TCL/TK command language extensions will be *machine generated* using the SWIG API code writer.
2. The eventMonitorAPI.so package will be completely responsible for interpreting event data in the form of the "Internal LDAS Lightweight Data Format".
3. The eventMonitorAPI.so package will inherit the functionality to communicate ILWD data through the data sockets from the genericAPI.so package.
4. The eventMonitorAPI.so package will support reading and writing of "Internal LDAS Lightweight Data Format files on the local file system.
5. The eventMonitorAPI.so package will be responsible for acquiring the event

## The Event Monitor API's baseline requirements

data results from each wrapperAPI job run on the LDAS cluster. It will accumulate this data until it has recognized the end of the wrapperAPI job through the associated transmission of the process table information (in ILWD format). NOTE: multiple wrapperAPI jobs will be running simultaneously and each will be transmitting snippets of data to the eventMonitorAPI. The eventMonitorAPI.so must manage these and uniquely identify the event data with the job using the jobId attribute found in the ILWD format.

6. Once the complete event data for a particular job has been acquired the eventMonitorAPI will parse this data into three distinct classes based on the output data result content described in the wrapperAPI documentation:
  - a) **database** - database table(s) information which will be directed to the LDAS metaDataAPI for insertion into the LIGO database.
  - b) **state** - state or static information which will be directed to file in binary ILWD format for reuse in later dataConditionAPI or wrapperAPI tasks.
  - c) **multidimdata** - user defined data which will be directed to the lightWeightAPI or frameAPI for conversion into LIGO\_LW or frame format and posted in the LDAS anonymous ftp or http area based on user request.
7. The eventMonitorAPI.so will provide a TCL extension used to set the values of the optimal number of rows to be inserted into each individual LDAS database table. This values for the individual tables will be provided at initialization by the TCL layer after having established these values from the eventMonitorAPI resource file.
8. The eventMonitorAPI.so will provide the functionality to optimally insert event data into the LDAS database by breaking up large event database results into separate table insertion requests to the database which have no more than 100% of the optimal number of rows per insertion request.
9. The eventMonitorAPI.so will provide fully integrated ILWD instances of the state and multidimdata data objects prior to shipping these off to their respective destinations.

## II. Component Layers of the LDAS eventMonitorAPI



### A. LDAS Distributed eventMonitorAPI:

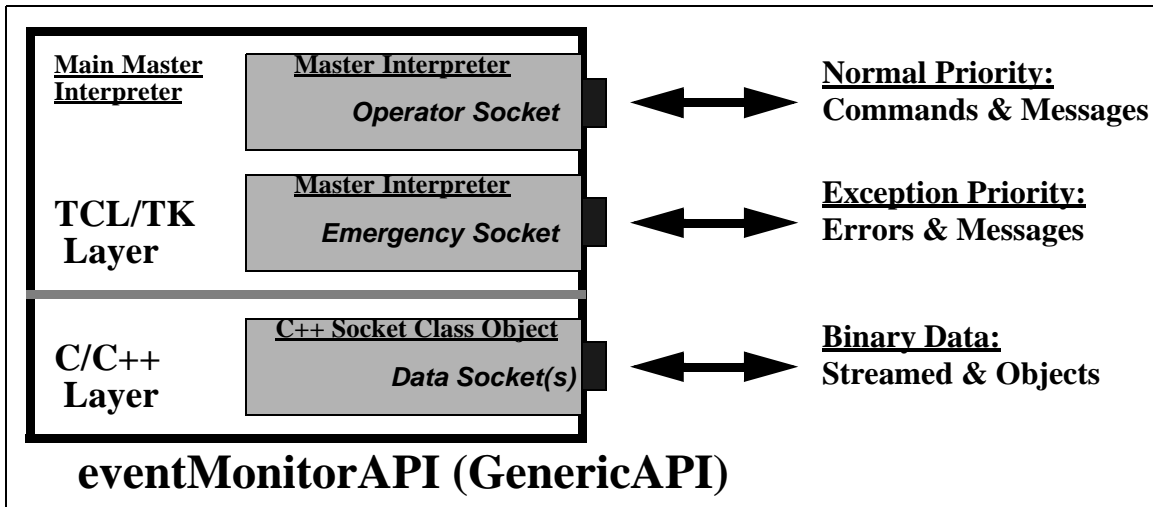
1. The LDAS distributed eventMonitorAPI is made up of two major layers.
  - a) TCL/TK Layer - this layer is the command layer and deals primarily with commands and/or messages and their attributes and/or parameters, as well as communicate with the underlying Package Layer through TCL/

## The Event Monitor API's baseline requirements

TK extensions.

- b) C/C++ Package Layer - this layer is the data engine layer and deals primarily with the binary data and the algorithms and methods needed to manipulate LIGO's data in the *Internal Light-Weight Format* (ILWD)
2. The TCL/TK layer consists of two internal and two external components, designed to optimize code reuse at the level of the command language used in all LDAS API's.
  - a) The eventMonitorAPI.tcl - this TCL/TK script contains specialized TCL/TK procedures and specialized command language extensions which are particular to the eventMonitorAPI in the LDAS architecture.
  - b) The genericAPI.tcl - this TCL/TK script contains the common TCL/TK procedures and command language extensions found in all LDAS API's. the genericAPI.tcl code will be sourced in the eventMonitorAPI.tcl script.
  - c) The eventMonitorAPI.rsc - this TCL/TK script contains the start-up and configuration defaults which are unique to the eventMonitorAPI, including the optimal number of rows to be inserted for each database table.
  - d) The genericAPI.rsc - this TCL/TK script contains the start-up and configuration defaults which are common to each LDAS API. The genericAPI.rsc will be embedded in the eventMonitorAPI.rsc file.
3. The C/C++ package layer consists of two internal components, each developed in C++ and C to take advantage of the higher performance associated with compiled languages which is needed for the types of activities that are being carried out in this layer and loaded as shared objects.
  - a) The eventMonitorAPI.so - this shared object contains the C++ classes and C interface functions needed to extend the command language set of the eventMonitorAPI, allowing it to more efficiently parse and redirect LIGO event data.
  - b) The genericAPI.so - this shared object contains the C++ classes and C interface functions needed to extend the command language set of all API's in LDAS, allowing efficiency and optimal code reuse. It will be linked into the eventMonitorAPI.so shared object directly.

### III. Communications in eventMonitorAPI from GenericAPI



#### A. Socket Based Communications in eventMonitorAPI:

1. The genericAPI will provide the eventMonitorAPI with an internet socket within the TCL/TK Server Layer that is the primary communication port for commands and messages of a normal priority. This port is commonly referred to as the *Operator Socket* to reflect its association with normal operations. Requirements on this socket are that defined by the genericAPI. The genericAPI will also provide the eventMonitorAPI with an internal socket within the TCL/TK Server Layer for exception priority messages. This port is commonly referred to as the *Emergency Socket* to reflect its association with exception handling.
2. The genericAPI will provide the eventMonitorAPI with dynamic TCP/IP sockets within the C/C++ layer that is used to communicate all data (*typically binary data*) in the form of streamed binary data or distributed C++ class objects using the ObjectSpace C++ Component Series Socket Library. This port is commonly referred to as the *Data Socket* to reflect its primary duty in communicating data sets. Requirements on these sockets are defined by the genericAPI.

### IV. Software Development Tools

#### A. TCL/TK:

1. TCL is a string based command language. The language has only a few fundamental constructs and relatively little syntax making it easy to learn. TCL is designed to be the glue that assembles software building blocks into applications. It is an interpreted language, but provides run-time tokenization of commands to achieve near to compiled performance in some cases. TK is an

## The Event Monitor API's baseline requirements

TCL integrated (as of release 8.x) tool-kit for building graphical user interfaces. Using the TCL command interface to TK, it is quick and easy to build powerful user interfaces which are portable between Unix, Windows and Macintosh computers. As of release 8.x of TCL/TK, the language has native support for binary data.

### **B. C and C++:**

1. The C and C++ languages are ANSI standard compiled languages. C has been in use since 1972 and has become one of the most popular and powerful compiled languages in use today. C++ is an object oriented super-set of C which only just became an ANSI/ISO standard in November of 1997. It provided facilities for greater code reuse, software reliability and maintainability than is possible in traditional procedural languages like C and FORTRAN. LIGO's data analysis software development will be dominated by C++ source code.

### **C. MPI:**

1. The parallel software components of the LDAS will use the public domain version of MPI from LAM, release 6.3.2 or greater.
2. The use of MPI code within LDAS will be restricted to the C++ interface bindings and the use of object oriented design technologies whenever possible. The templated analysis filters and associated functions are not required to be developed using C++ and object oriented design techniques. However, they must support bindings to the core C++ slave processes.

### **D. SWIG:**

1. SWIG is a utility to automate the process of building wrappers to C and C++ declarations found in C and C++ source files or a special *interface file* for API's to such languages as TCL, PERL, PYTHON and GUIDE. LDAS will use the TCL interface wrappers to the TCL extension API's.

### **E. Make:**

1. Make is a standard Unix utility for customizing the build process for executables, objects, shared objects, libraries, etc. in an efficient manor which detects the files that have changed and only rebuilds components that depend on the changed files. The Make facility is being extended using AutoConfig, AutoMake and LibTools, all from the public domain.

### **F. CVS:**

1. CVS is the Concurrent Version System. It is based on the public domain (and is public domain itself) software version management utility RSC. CVS is based on the concept of a software source code repository from which multiple software developers can check in and out components of a software from any point in the development history.

## The Event Monitor API's baseline requirements

### G. Documentation:

1. DOC++ is a documentation system for C/C++ and Java. It generates LaTeX or HTML documents, providing for sophisticated online browsing. The documents are extracted directly from the source code files. Documents are hierarchical and structured with formatting and references.
2. TcdDOC is a documentation system for TCL/TK. It generates structured HTML documents directly from the source code, providing for a similar online browsing system to the LDAS help files. Documents include a hyper-text linked table of contents and a hierarchical structured format.