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

Document Type	**LIGO-T980093-00 - E**	10/8/1998

The Generic API's "base-line 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 51-33
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 Generic API's
"*base-line requirements*"

James Kent Blackburn

California Institute of Technology
LIGO Data Analysis Group
October 8, 1998

I. Introduction

A. General Description:

1. The genericAPI provides the base set of functionality in the form of an interpreted command language that exist in all LIGO Data Analysis System (LDAS) distributed computing API components.
   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 and 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 genericAPI TCL/TK script accesses a genericAPI.rcs file containing needed information and resources to extend the command set of the TCL/TK language using the genericAPI package, which exists in the form of a shared object.
3. The genericAPI will provide setup and configuration for all socket and file based communications used in all other LDAS distributed computing API components.

B. The genericAPI.tcl Script's Requirements:

1. The genericAPI.tcl script will provide the user's "Help" interface in the form of a procedure which can be called in any LDAS distributed API component to provide a web technology based interface to the LDAS help pages.
2. The genericAPI.tcl script will provide procedures for logging of each command or message entered into the interpreter via the TCL/TK layer's sockets as it is received, completed or fails with an exception with accurate and meaningful timestamps, possibly of GPS standard. Each LDAS API uses this functionality local to its own quasi-independent logging file. The ability to read and view these logging files will also be provided by the genericAPI.tcl script.
3. The genericAPI.tcl script will use a resource file named genericAPI.rcs to configure communications via TCL/TK sockets used to communicate commands and messages, and the C language-API sockets used in the packages to send data. These sockets are often referred to as the *operator* and *emer-*

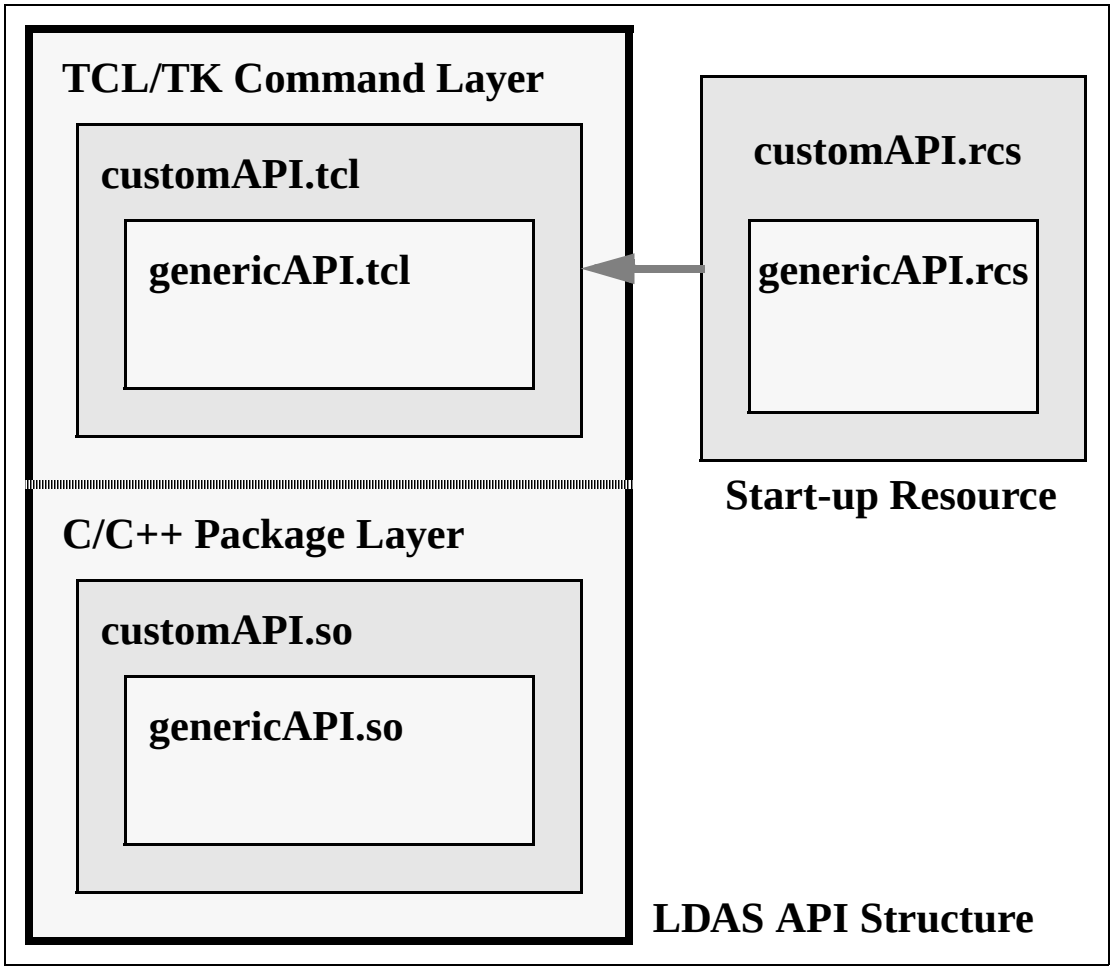
gency sockets in the TCL/TK layer of the API and as the *data* sockets in the C++ layers of the API. This resource file will also manage the loading of generic TCL/TK extensions found in the genericAPI package.

4. The genericAPI.tcl script will provide procedures which allow parameters found in the resource file to be over-ridden.
5. The genericAPI.tcl script will provide a separate interpreter for each socket that is open in the "listen" state to allow thread-like response to messages and commands passed through these sockets.
6. The means for which packages are loaded will be robust and not dependent on system paths or file names. This will be achieved by using the *package facilit*y in TCL/TK.
7. The genericAPI.tcl script will detect and handle exceptions generated by the underlying genericAPI package's C++ layer. This includes reporting, logging and provide clean continuation of the API operation after the exception is reported.
8. The genericAPI.tcl script will provide procedures for receiving LDAS XML tags (LDAS lightweight data) from the underlying package layer and for sending LDAS lightweight data to the underlying package layer.
9. The genericAPI.tcl script will provide a ping procedure which allows the LDAS manager software to detect whether or not each API is alive.

C. The genericAPI.so Package Requirements:

1. The genericAPI.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 between C++ functionality and TCL/TK command language extensions will be *machine generated* using the *SWIG* API code writer.
2. The genericAPI.so package will be responsible for establishing and maintaining socket communications using a C++ class library for sockets developed by *ObjectSpace* and distributed as their *C++ Component Series Class Library.* These sockets will be used to send binary data between LDAS's various distributed API components in the form of streamed sequential bytes and in the form of registered C++ class objects.
3. The genericAPI.so package will translate registered LDAS lightweight data C++ class objects into XML documents for passing up to the TCL/TK layer, as well as translation of XML documents containing registered LDAS lightweight data tags into lightweight class objects in C++.
4. The genericAPI.so package will provide store and restore methods for the local C++ class objects based on the *ObjectSpace Library.*
5. The genericAPI.so package will provide exception handling of the generic C++ methods used to perform the above requirements and provide through the TCL/TK interface, reports of these exceptions to the TCL/TK layer.

II. Nature of the Distributed LDAS API



A. Custom LDAS Distributed API's:

1. Each LDAS distributed API 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/ 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
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 customAPI.tcl - this TCL/TK script contains specialized TCL/TK

procedures and specialized command language extensions which are particular to each custom API 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.

c) The customAPI.rcs - this TCL/TK script contains the start-up and configuration defaults which are unique to each custom API.

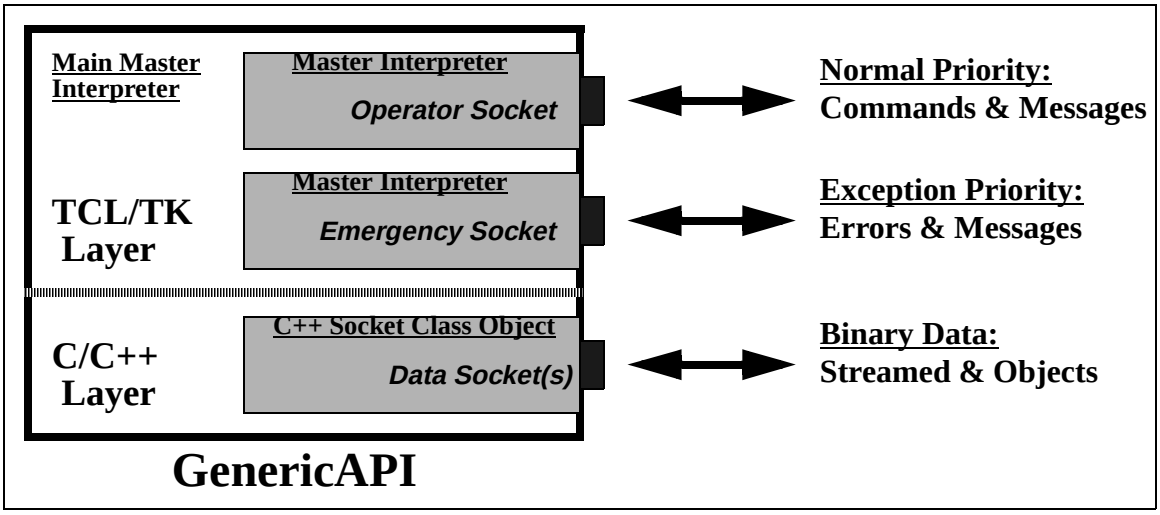
d) The genericAPI.rcs - this TCL/TK script contains the start-up and configuration defaults which are common to each LDAS API.

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 customAPI.so - this shared object contains the C++ classes and C interface functions needed to extend the command language set of each custom API, allowing it to more efficiently perform its duties.

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.

III. Communications Provided by GenericAPI



A. Socket Based Communications in GenericAPI:

1. The genericAPI will provide an internet socket within the TCL/TK 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

a) it run in either a Standard or Safe Master Interpreter depending on the requirements of each specific LDAS API,

b) it have the option of requiring an encrypted key attached to each incoming command for authentication prior to execution of the command as a security safeguard (*depending on the nature of each specific API*),

c) it will provide a queue for incoming commands, allowing for at least 25 commands to be staged in a FIFO during peak levels of communications. If the FIFO ever fills up the manager API will be notified using the *Emergency Socket* discussed below.

2. The genericAPI will provide an internet socket within the TCL/TK layer that is an exception communication port for commands and messages of a highest priority. This port is commonly referred to as the *Emergency Socket* to reflect its association with critical operations. Requirements on this socket are that

   a) it run in either a Safe Master Interpreter supporting only a few exception handing commands for each specific API through the TCL/TK aliasing mechanism,

   b) it have the option of requiring an encrypted key attached to each incoming command for authentication prior to execution of the command as a security safeguard (*depending on the nature of each specific API*),

   c) it will provide a queue for incoming commands, allowing for no more than 5 commands to be staged in a FIFO during peak levels of communications. If the FIFO ever fills up the manager API will be notified.

3. The genericAPI will provide dynamic internet 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 this socket are that

   a) it be capable of managing up to 5 simultaneous socket connections at any given time by using spawned TCL/TK master interpreters (*request for more socket communications will remain in the Operator Socket queue*),

   b) it be capable of streaming raw LIGO Frame data directly,

   c) it be capable of streaming raw LIGO Light-Weight data directly,

   d) it be capable of sending and receiving instances of C++ internal LDAS light-weight data objects (*a simplified subset of the full LIGO Light-Weight data*),

   e) it be capable of storing the states of connected sockets to a file and restore those socket states at the local request of the API.

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 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. 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.

D. 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.If/when LDAS software becomes architecturally dependent, it will be necessary to supplement make with auto-configuration scripts.

E. 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.