



LIGO Data Analysis System

LIGO PAC 7

Massachusetts Institute of Technology

November 18th - 19th, 1999

*James Kent Blackburn
LIGO Laboratory
California Institute of Technology*

LIGO-G990114-00-E



Introduction

- ❑ System Design Overview
 - ⇒ *if requested*
- ❑ Site Installation
- ❑ Software Construction Review
- ❑ Software Documentation
- ❑ System Testing
- ❑ Next Major Development (Parallel LDAS)
- ❑ Closing Remarks



System Design Milestones

□ System Specification

- ⇒ *LIGO Data Analysis White Paper*: LIGO-M970065-B-E
- ⇒ June, 1997

□ Design Requirements Review

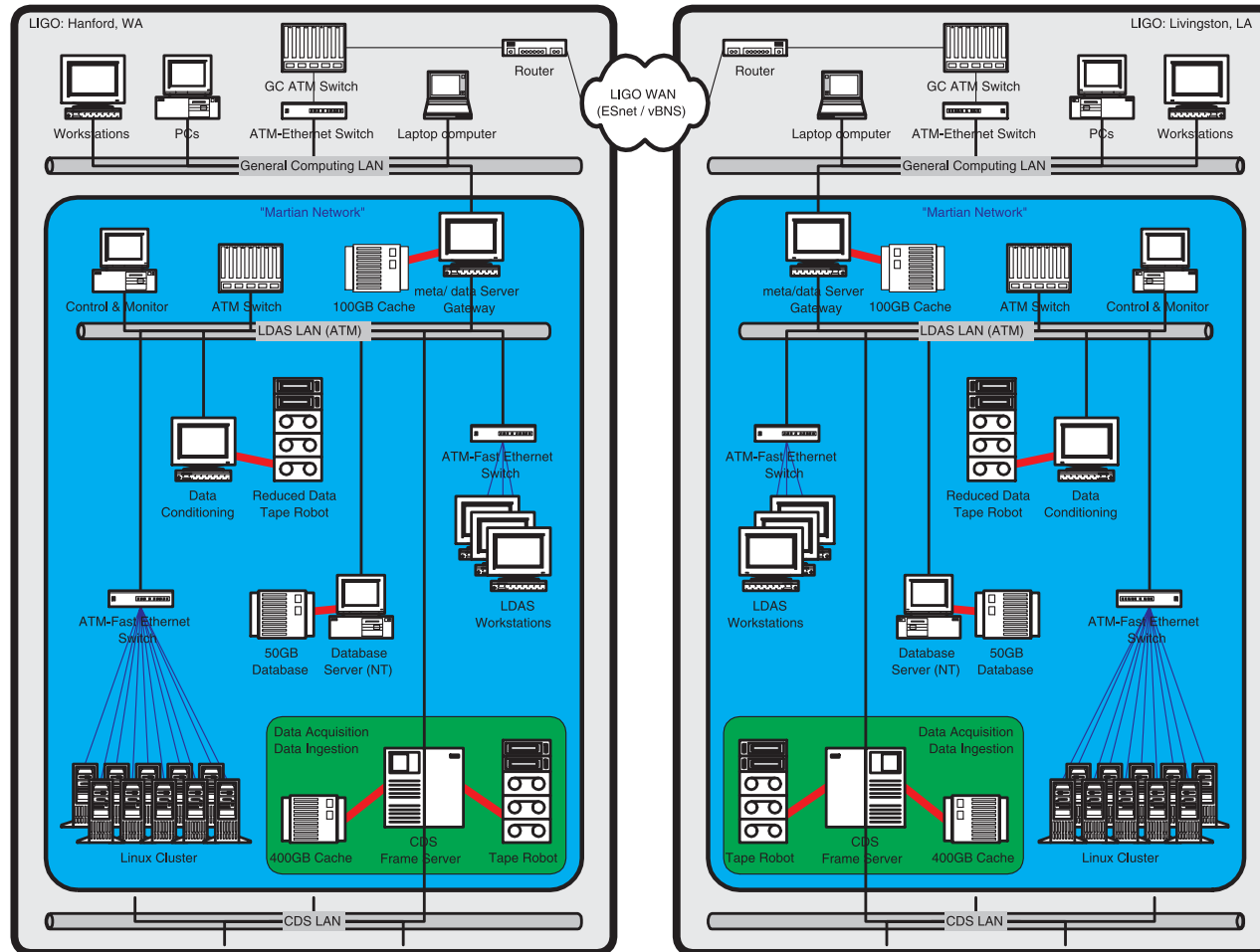
- ⇒ *LDAS Design Requirements*: LIGO-T970159-4-E
- ⇒ *LDAS Conceptual Design*: LIGO-T970160-6-E
- ⇒ December, 1999

□ Preliminary Design Review

- ⇒ *LDAS Preliminary Design*: LIGO-T990001-6-E
- ⇒ March, 1999

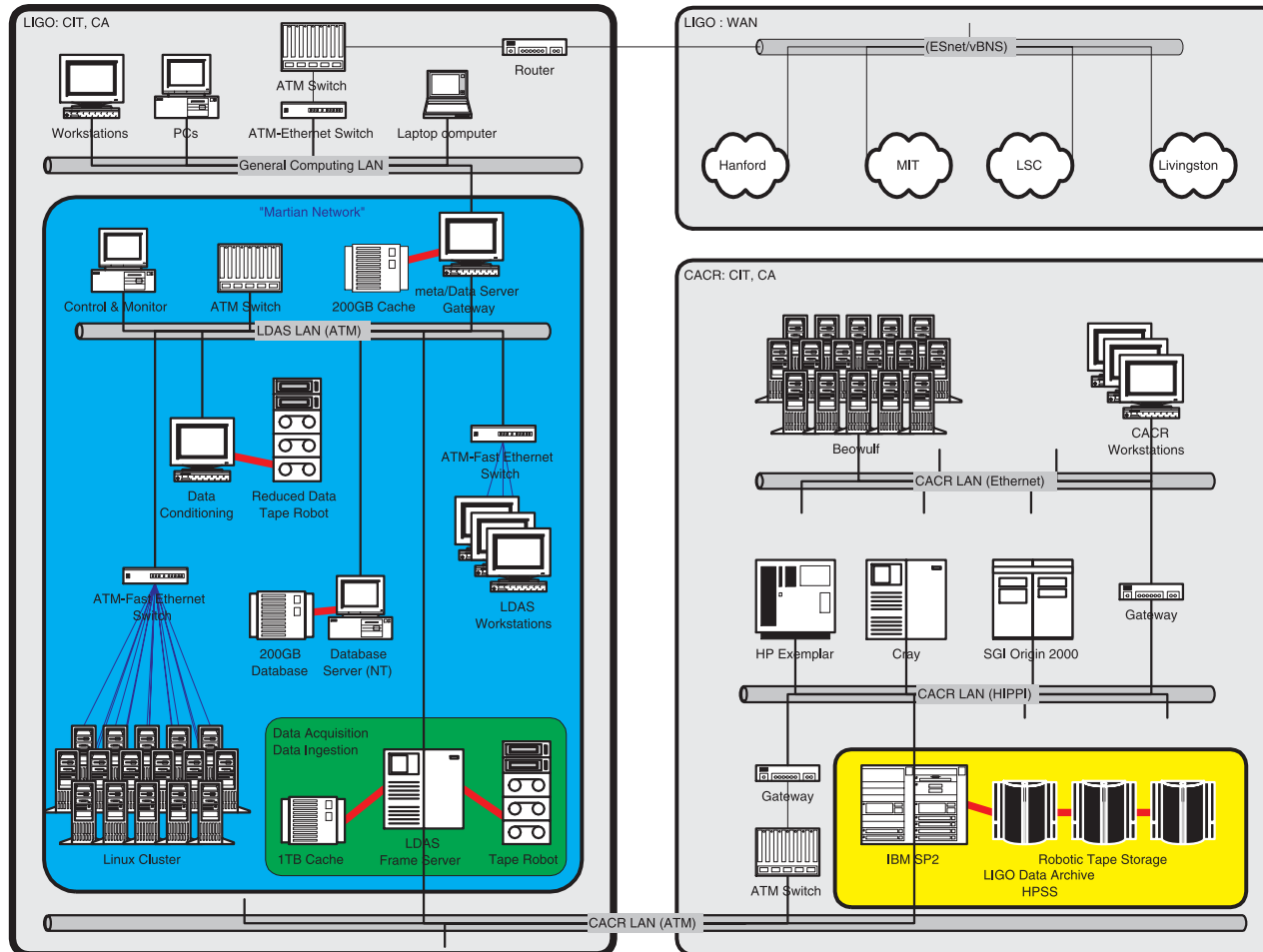


Hardware Architecture ^(site)





Hardware Architecture ^(caltech)





Client - Server - Servlets

❑ Clients: User Interfaces

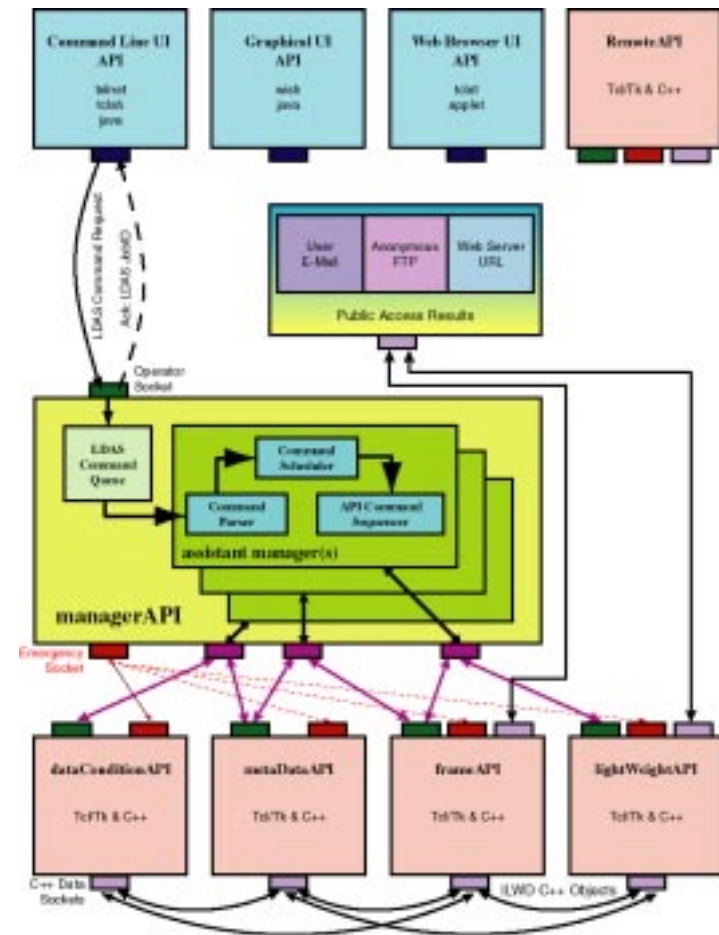
- ⇒ Three types of UIs:
 - ⇒ command line
 - ⇒ graphical
 - ⇒ web browser
- ⇒ Send LDAS Command requests to “Operator Socket” on managerAPI and receive acknowledgement containing “jobID” and other details associated with request.

❑ Server: managerAPI

- ⇒ Uses 3-10 assistant managers to handle each LDAS Command Request independently.

❑ Servlets: LDAS APIs

- ⇒ Internal LDAS APIs & RemoteAPI using LDAS interfaces.
- ⇒ Analysis flow control managed by “meta-scripts” associated with LDAS Commands.
- ⇒ Data communicated internally as ILWD objects
- ⇒ Data shared externally as Frames & LIGO_LW(XML).

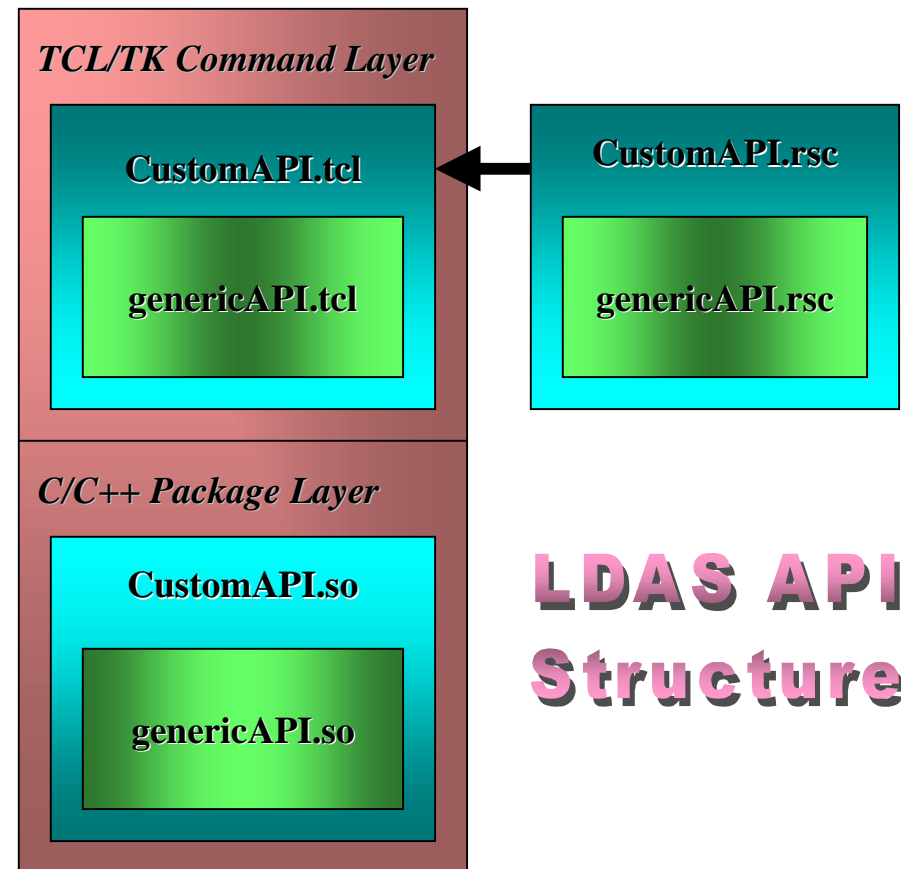




Layered APIs

□ LDAS APIs:

- ⇒ Two Primary Layers:
 - ⇒ Tcl/Tk (glue code)
 - ⇒ C/C++ (Tcl extensions)
 - ⇒ Use Swig to interface
- ⇒ GenericAPI (core) Module:
 - ⇒ Communications
 - ⇒ TCL - C++ using SWIG
 - ⇒ API - API using sockets
 - ⇒ Common Tcl proc functions:
 - ⇒ Logging
 - ⇒ Operator/emergency socket management
 - ⇒ Resource/configuration management
 - ⇒ Common C/C++ extensions:
 - ⇒ Data socket management
 - ⇒ ILWD data objects
 - ⇒ Object serialization
 - ⇒ object save/restore methods
- ⇒ CustomAPI (specialization) Module:
 - ⇒ unique/modular tasks:
 - ⇒ database, frames, ligo_lw, data conditioning, parallel computing, etc.

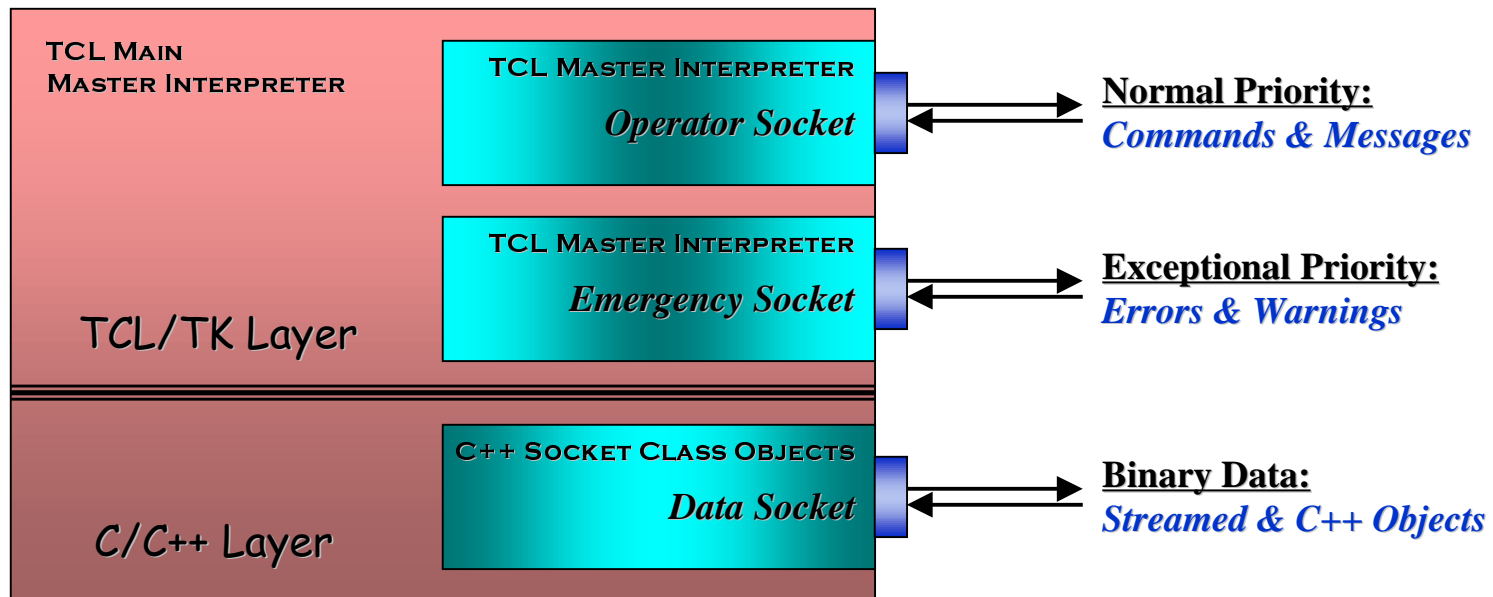




API Socket Communications

□ 3 Types of socket communications in each LDAS API:

- ⇒ Operator Sockets - Normal inter-process *commands & messages* (Tcl Layer)
- ⇒ Emergency Sockets - Error & warning *commands & messages* (Tcl Layer)
- ⇒ Data Sockets - Binary Data in *raw streams* or *C++ objects* (C++ Layer)

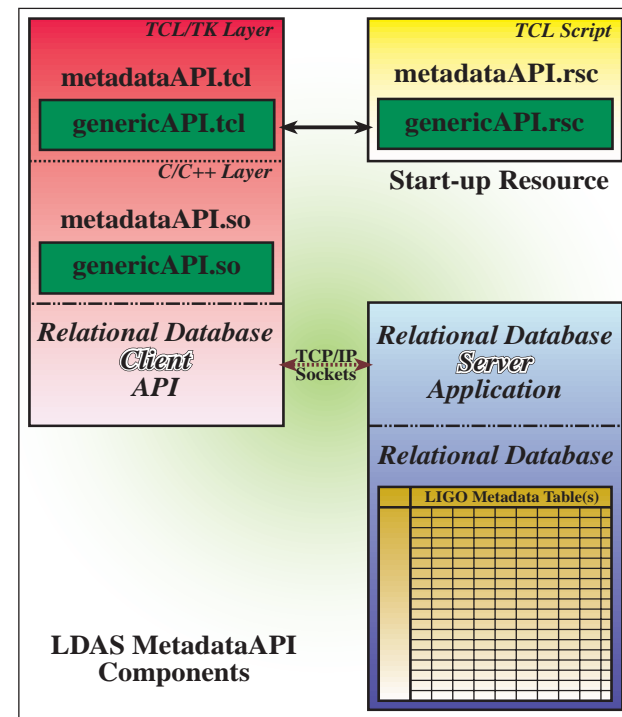
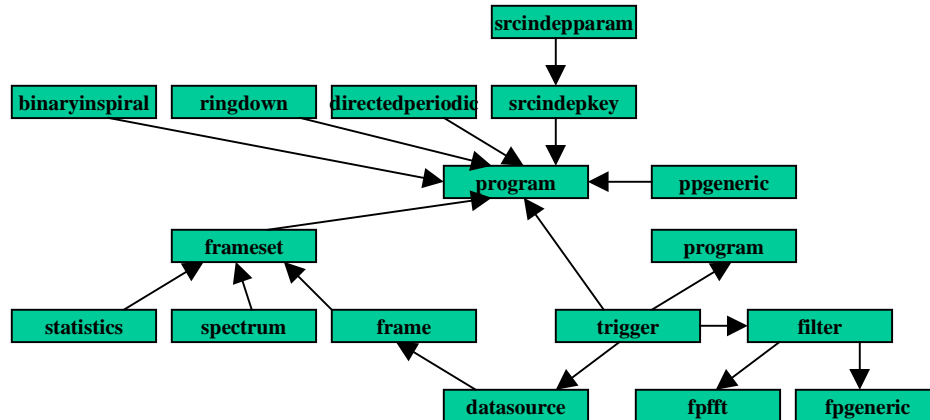




Distributed Database API

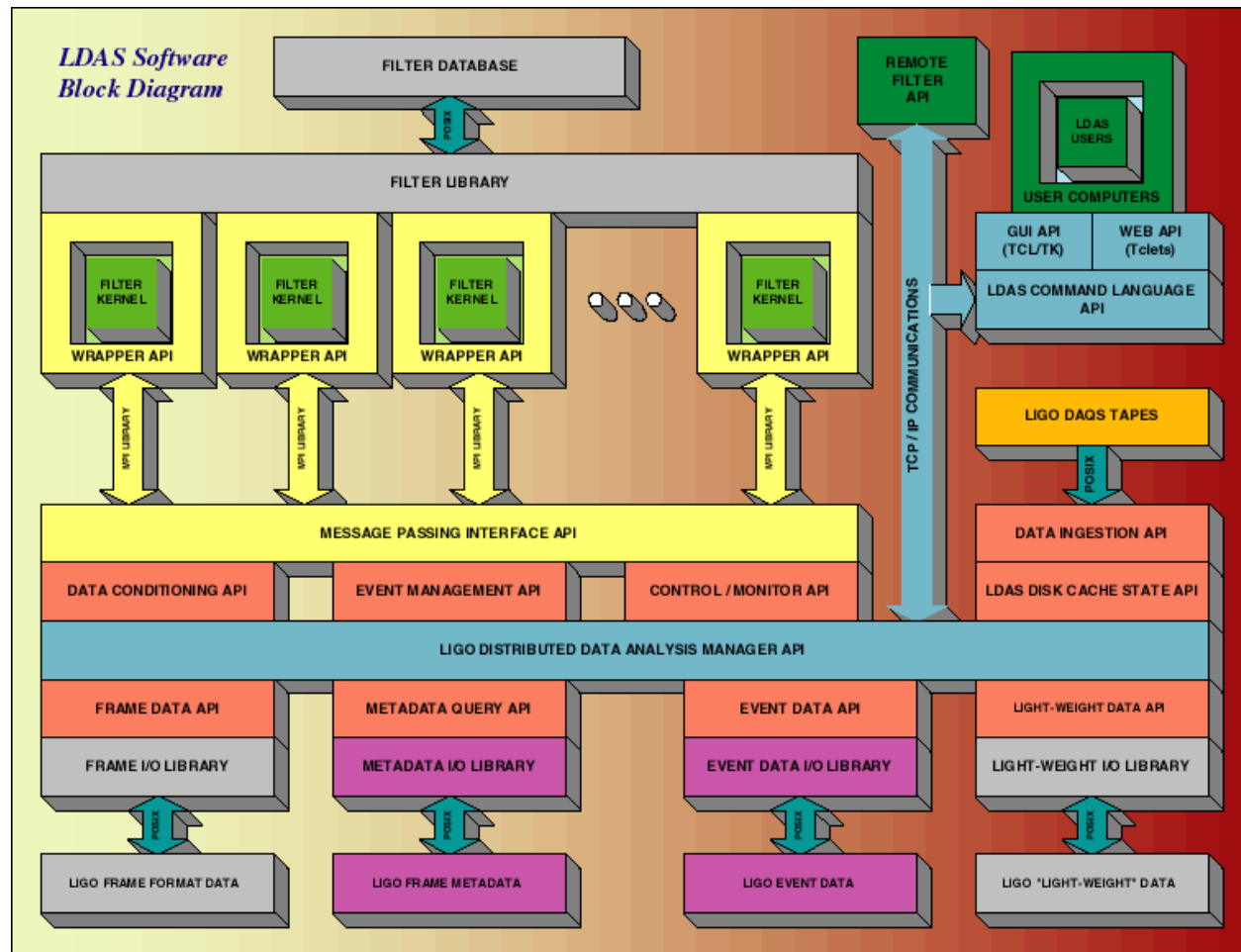
□ metaDataAPI:

- ⇒ Relational Database
 - ⇒ Using ODBC standard for connectivity
 - ⇒ Chose IBM's DB2
 - ⇒ Currently using 5.2, plan upgrade to 6.1
 - ⇒ DB2 server can be on Solaris or Linux now
- ⇒ Table contents
 - ⇒ Centralized around program table
 - ⇒ Frame tables
 - ⇒ Astrophysical event tables
 - ⇒ GDS diagnostic trigger tables
- ⇒ Table relationships





Software Block Diagram





Site Installation

□ Hanford

- ⇒ June 20th - June 25th, 1999:
 - ⇒ *Initial Hardware Installation*
- ⇒ July 25th - July 30th, 1999:
 - ⇒ *Initial Software Installation*
- ⇒ September 26th - November 1st, 1999:
 - ⇒ *Continued Software & Hardware Installation*

□ Livingston

- ⇒ October 24th - October 29th, 1999:
 - ⇒ *Initial Hardware Installation*



Hanford Hardware Rack

- Hanford: Jun, Jul, Sep, 99
 - ⇒ **dataserver:** *managerAPI / frameAPI*
 - ⇒ Sun Ultra 10 + 18GB HD
 - ⇒ **metaserver:** *metadataAPI*
 - ⇒ Sun Ultra 10 + 18GB HD
 - ⇒ **controlmon:** *internal software mirror*
 - ⇒ Pentium III PC + 18GB HD
 - ⇒ **Fore 1000 ATM switch**
 - ⇒ **Fore 2810 fast ethernet switch**
 - ⇒ **1 sun & 1 linux workstations**
 - ⇒ Sun Ultra10
 - ⇒ Pentium III PC
 - ⇒ **video/keyboard switch**





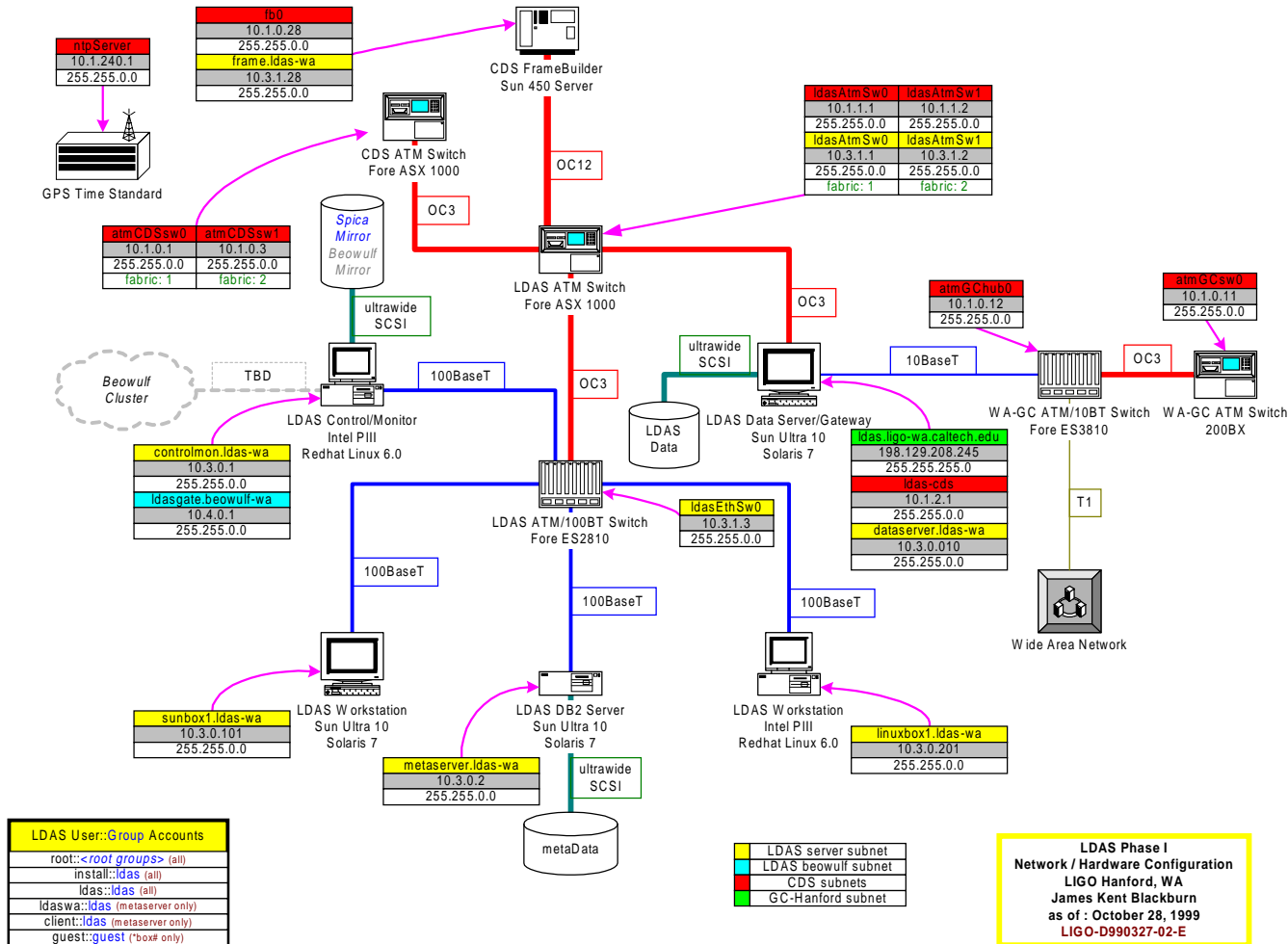
Livingston Rack

- Livingston: Oct, 99
 - ⇒ **dataserver:** *managerAPI / frameAPI*
 - ⇒ Sun Ultra10 + 18GB HD
 - ⇒ **metaserver:** *metadataAPI*
 - ⇒ Sun Ultra10 + 18GB HD
 - ⇒ **controlmon:** *internal software mirror*
 - ⇒ Pentium III PC + 18 GB HD
 - ⇒ **Fore 200BX ATM switch**
 - ⇒ **Fore 2810 fast ethernet switch**
 - ⇒ **1 sun & 1 linux workstations**
 - ⇒ Sun Ultra10
 - ⇒ Pentium III PC
 - ⇒ **video/keyboard switch**



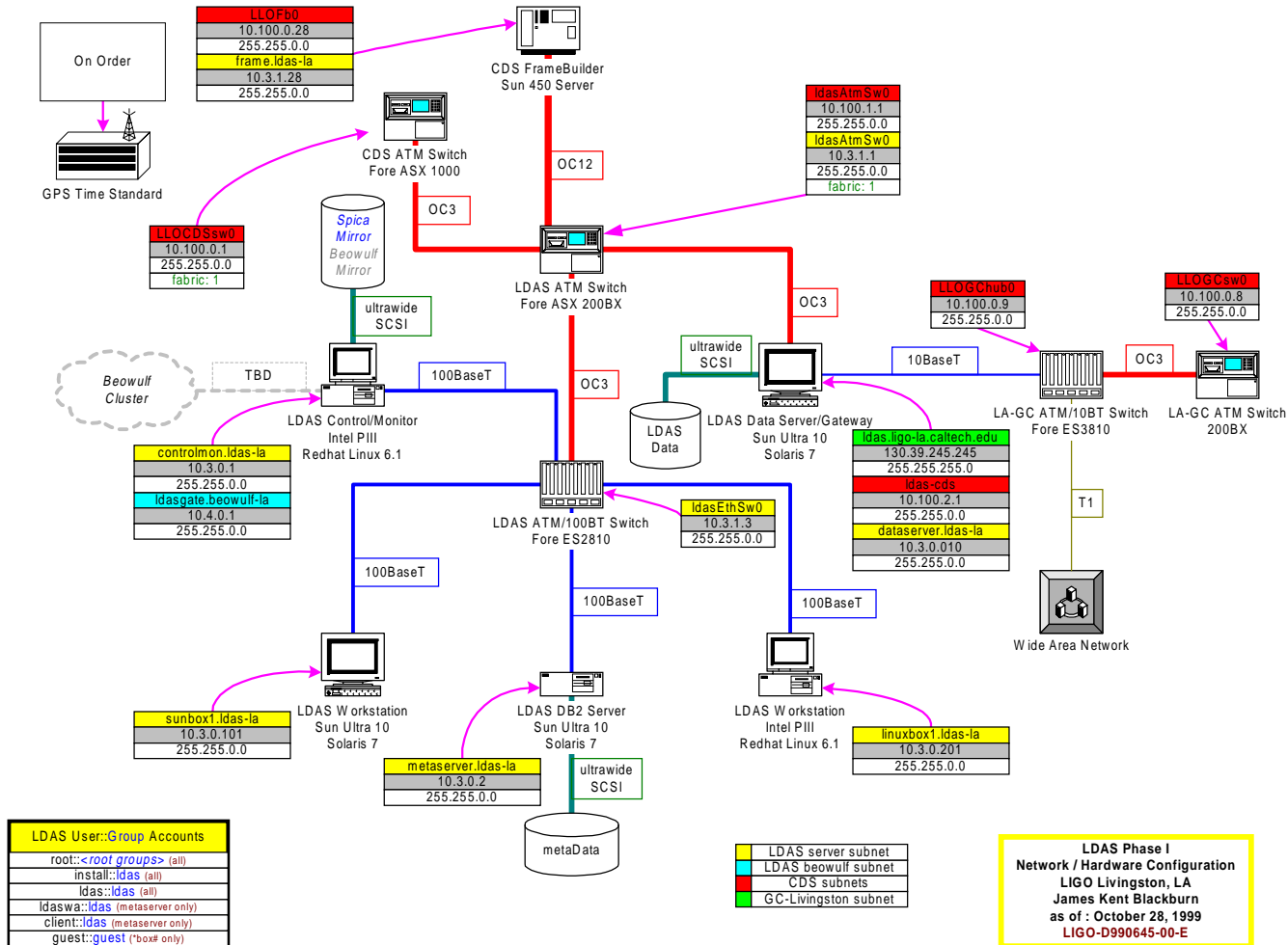


Hanford Network Diagram





Livingston Network Diagram





Site Software Installation

- ❑ Development Server: `spica.ligo.caltech.edu`
 - ⇒ compilers, interpreters, debuggers, editors, etc
 - ⇒ LDAS CVS Repository
 - ⇒ LDAS release installations
 - ⇒ all software managed using “*stow*” utility
 - ⇒ Soon: web based bug/change report system (*bugzilla* being considered)
- ❑ Spica mirrored to sites using “*rsync*” utility.
- ❑ Unix configurations managed with “*cfengine*” utility.
- ❑ Limited unix accounts: {`install`, `ldas`, `ldaswa`, `client`, `guest`, `reboot`, & `root`}
- ❑ Internet gateway provides access to LDAS “private net.”
 - ⇒ LDAS `managerAPI`, `frameAPI`, `lightWeightAPI`
 - ⇒ anonymous ftp
 - ⇒ web service
 - ⇒ ssh services
- ❑ Account logins using “*ssh*” & “*scp*” utilities only!

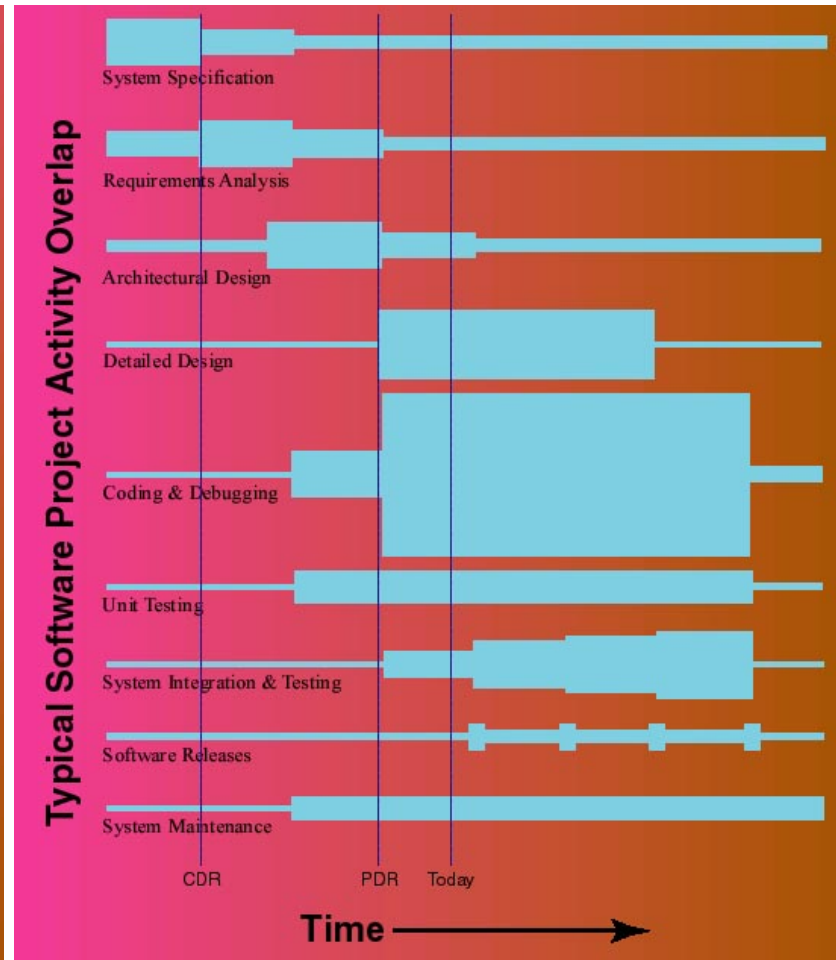
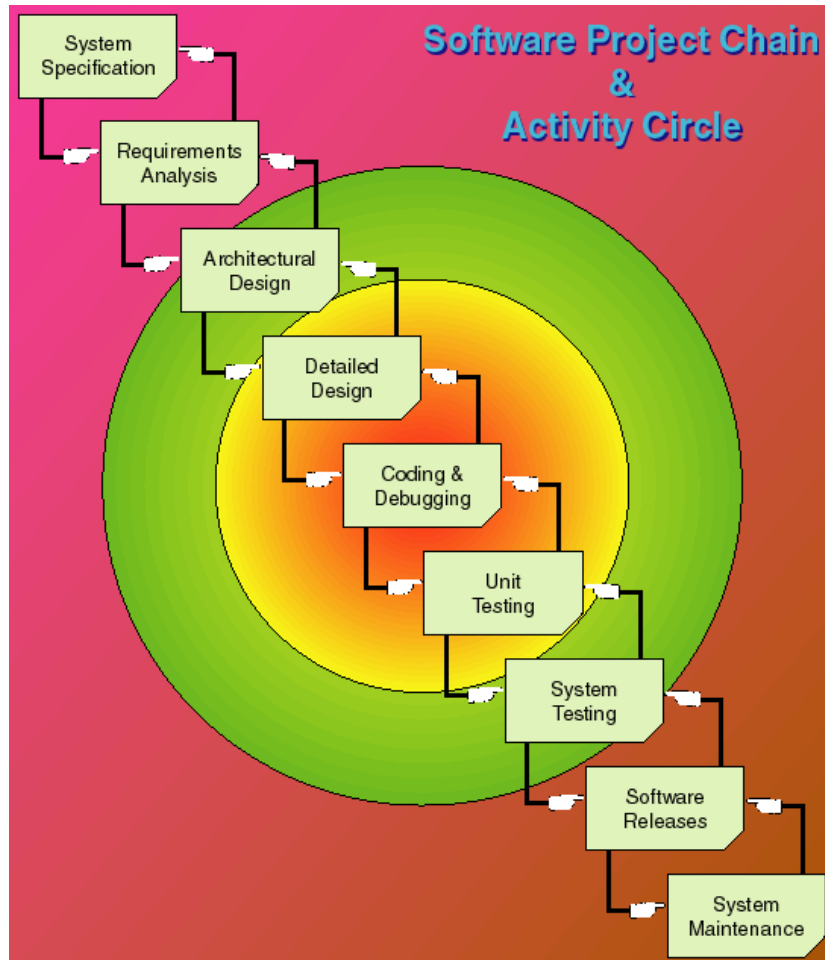


Software Construction Status

- ❑ Software Development Cycle
- ❑ Staffing
- ❑ Progress to Date
- ❑ CVS Repository Details
 - ⇒ C++ Object Oriented Libraries
 - ⇒ C++ API Shared Objects
 - ⇒ Tcl/Tk Scripts

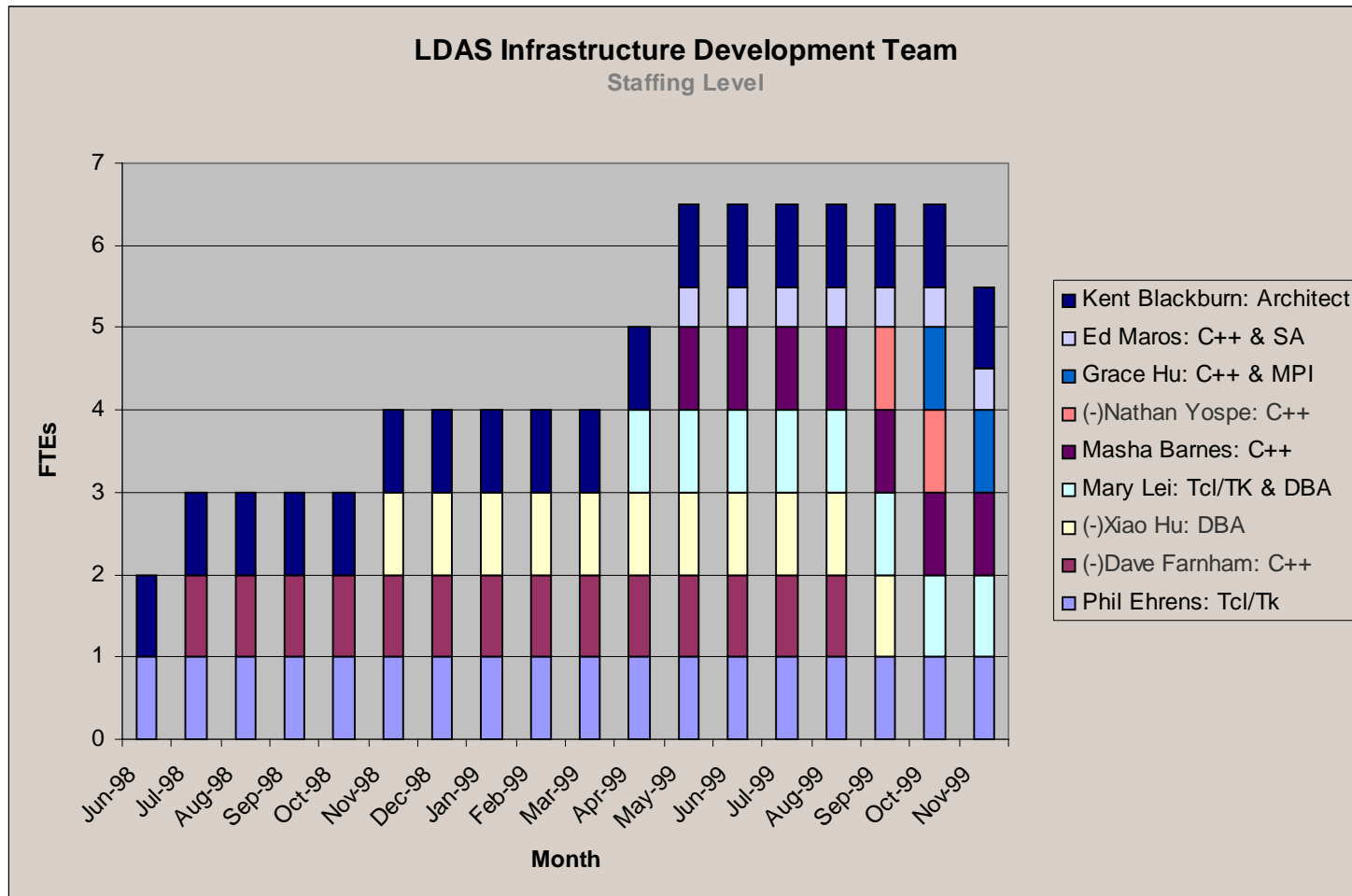


Software Project Activities



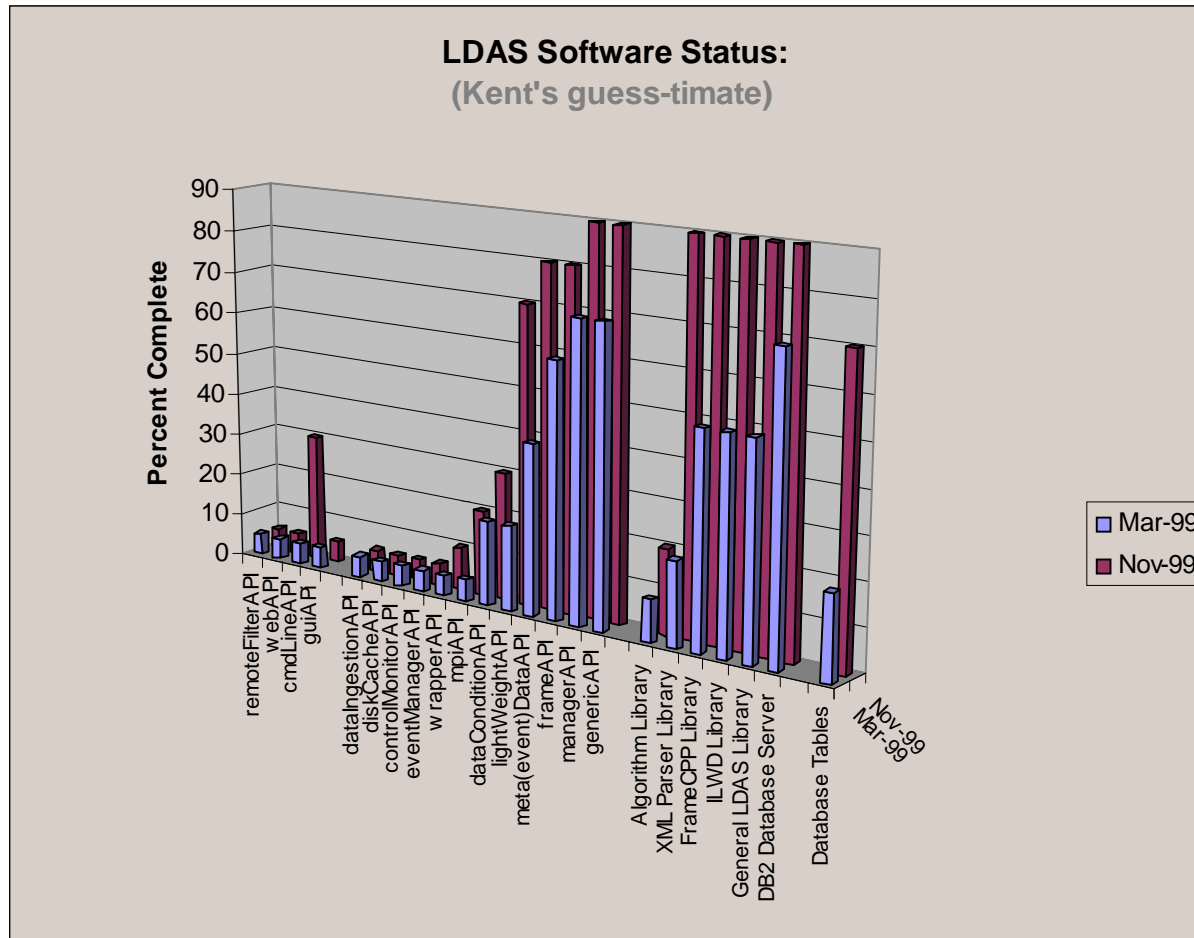


LDAS Software Team



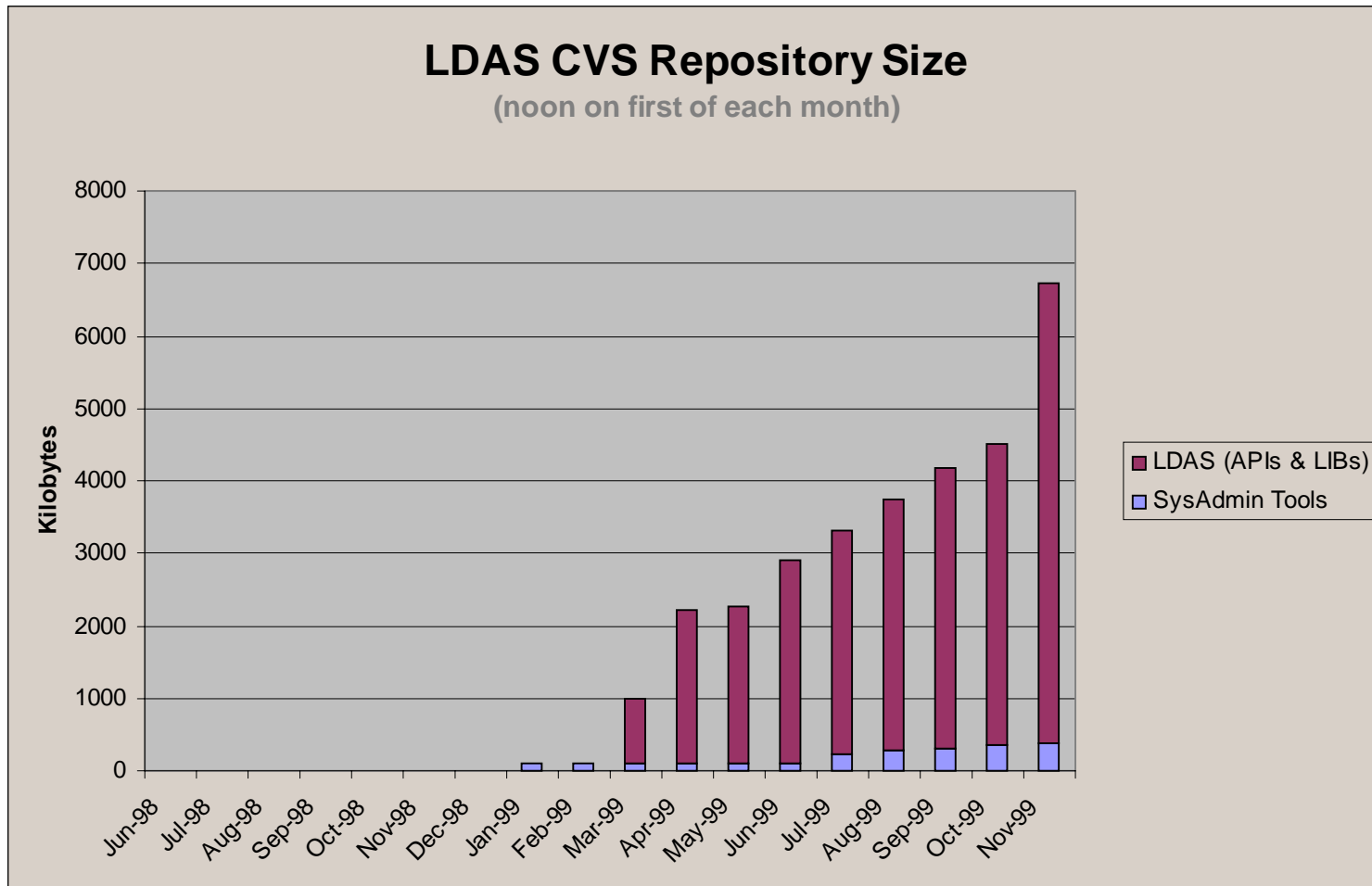


Software Status Estimate



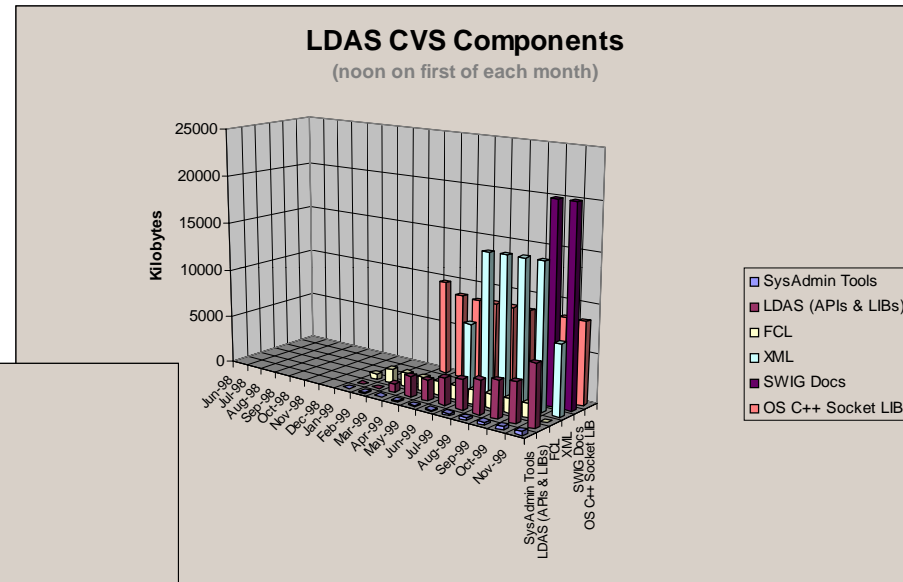
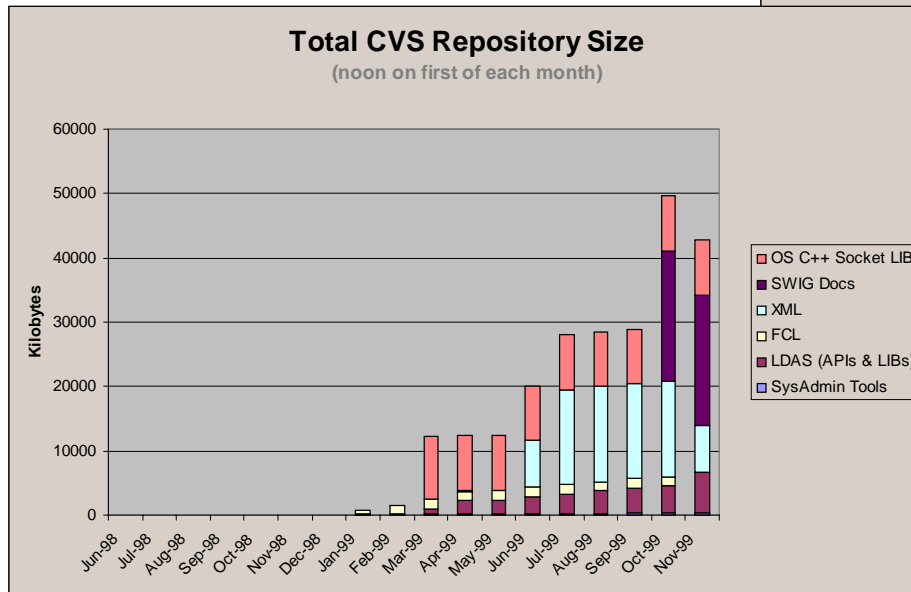


New LDAS Software



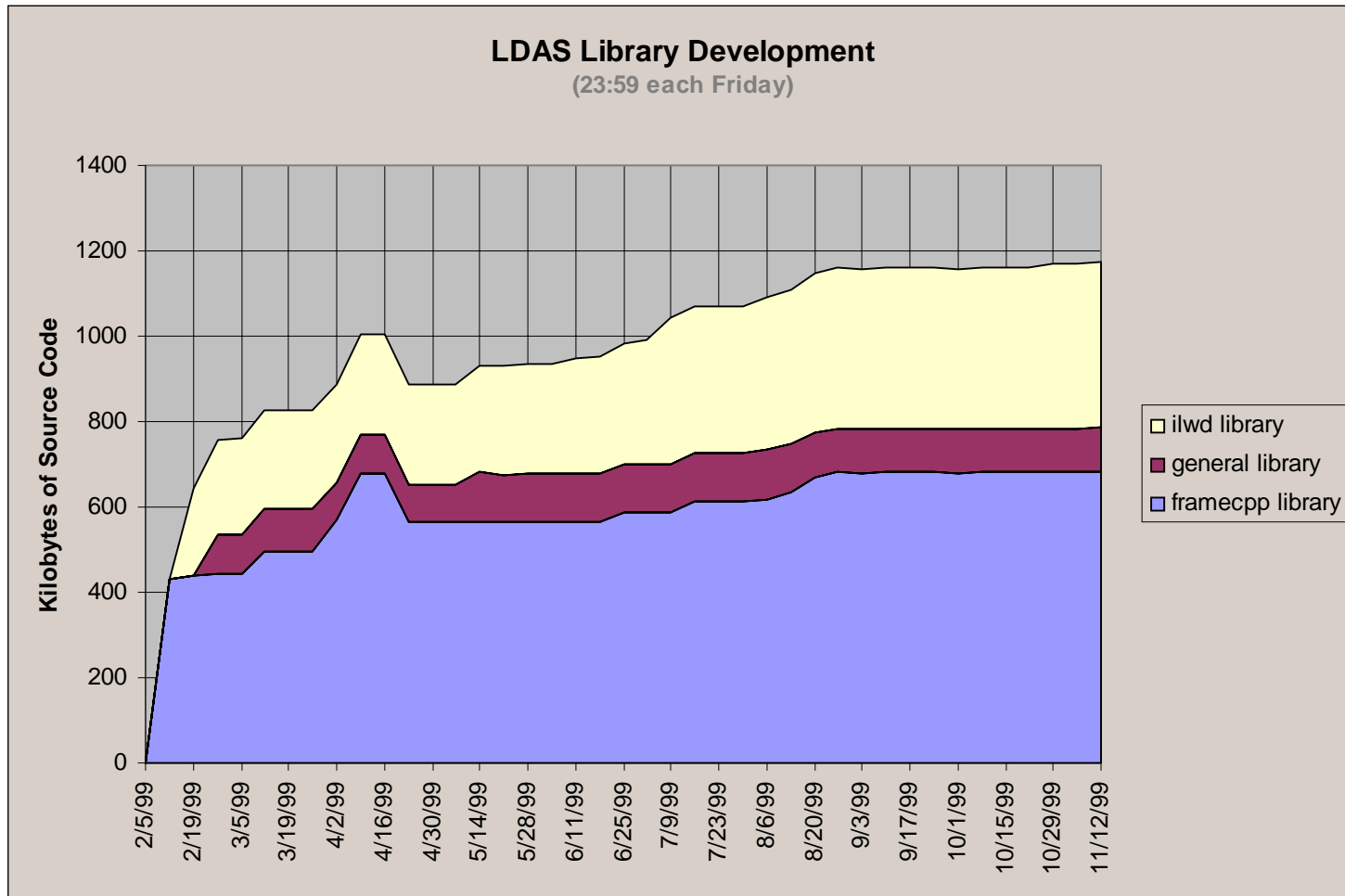


CVS Repository Details



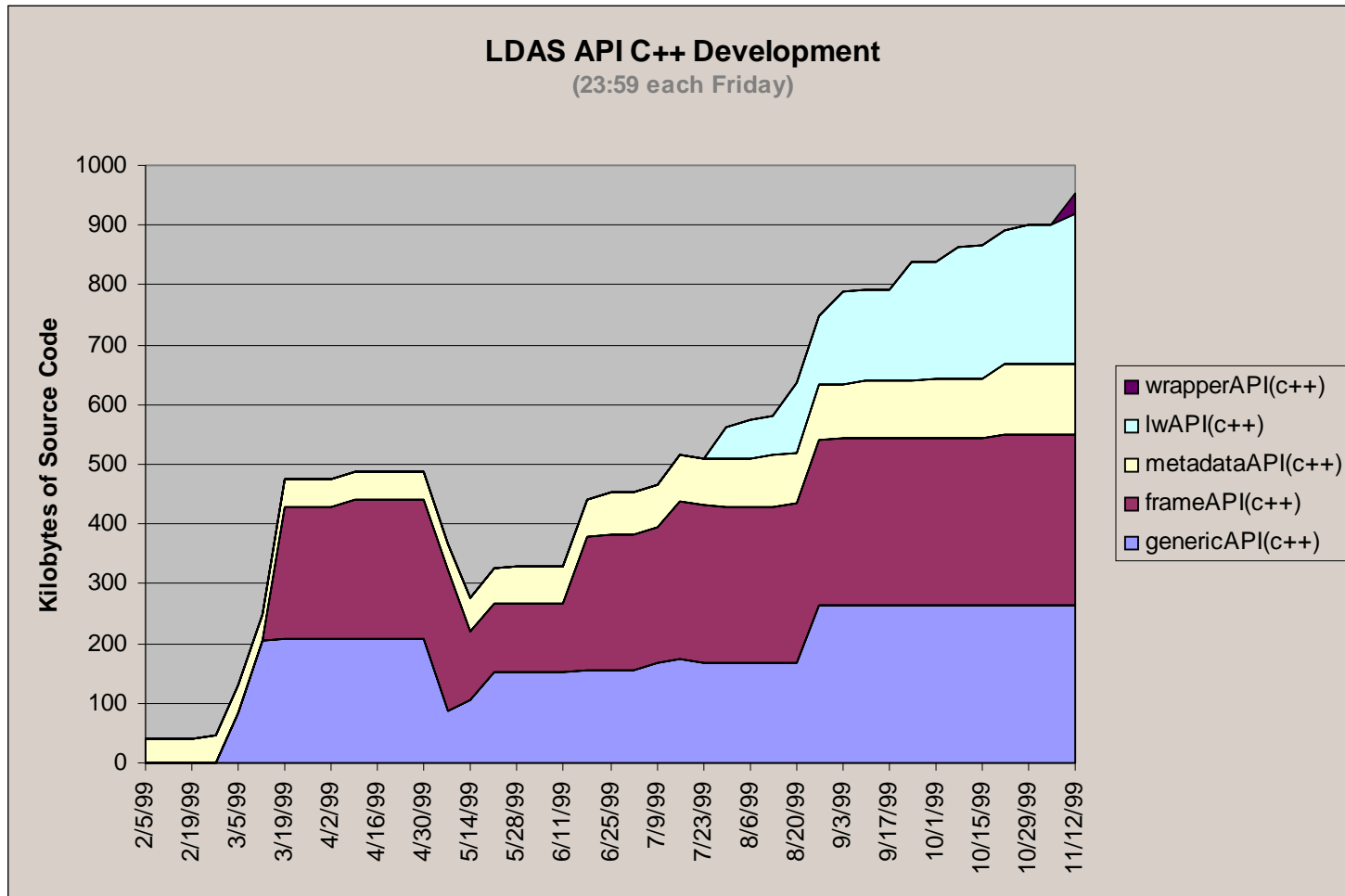


C++ Library Code



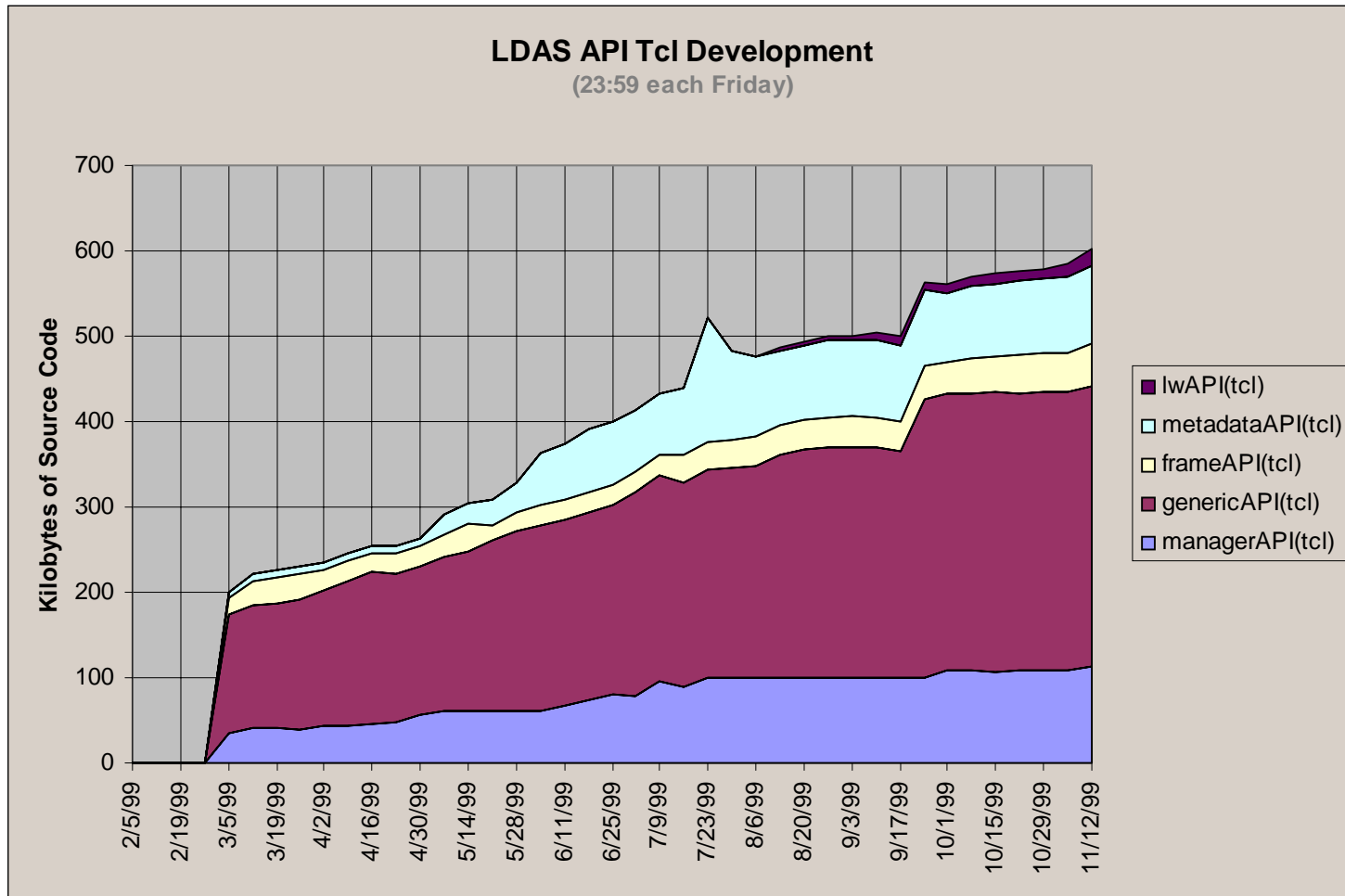


C++ API Shared Object Code





Tcl API Scripting Code





Software Documentation

❑ Web Documentation

- ⇒ **Caltech:** <http://www.ldas.ligo.caltech.edu/ldas/ldas-0.0/doc/index.html>
- ⇒ **Hanford:** <http://www.ldas.ligo-wa.caltech.edu/ldas/ldas-0.0/doc/index.html>
- ⇒ **Livingston:** <http://www.ldas.ligo-la.caltech.edu/ldas/ldas-0.0/doc/index.html>

❑ User Interfaces

❑ Tcl/Tk - *TclDoc*

❑ C++ - *Perceps*

❑ Log Files - *LDAS APIs*



Software Home Page





User Command Model

A screenshot of a Microsoft Internet Explorer browser window. The title bar reads "LDAS User Commands - Microsoft Internet Explorer". The address bar shows the URL "http://opira.ligo.caltech.edu/Adm/Adm-0.0/doc/userAPI.html/index.html". The page content includes the LIGO logo in the top right corner, followed by the heading "The LDAS User Commands". Below this, it states "All user commands have the form:" and provides a Telnet command syntax: `ldasJob (-name { }) -password { } -email { } { } userCmd -opt1 { } ... { }`. It then explains that this is the format of a [Tel](#) command, named `ldasJob`, with two required arguments. A numbered list follows: 1. A Tel list of **user information** consisting of username, password, and e-mail address. All fields must be filled or the command will be rejected. 2. A **user command** in the form of a Tel list, consisting of the name of the user command (`userCmd` in the example) for which there exists a 'meta' macro file, and the required options with their argument lists. Further text explains that an argument must be provided to every option field, some options accept a 'null' argument, and meta macros consist of a prototype declaration and a template of API-specific blocks of Tel code. The page concludes by listing three classes of LDAS User Commands: [getData](#), [putData](#), and [descData](#).



Examples of User Commands

getMetadata

Retrieves meta-data stored in the LIGO database in response to the query given by the user command, returns in several formats either by attaching the resulting file to an e-mail (external email) or by returning an e-mail with a URL pointing to the location of the results (external http or ftp). It is also possible to have the data directly delivered to a local port, which will be the default method for the get to be completed user API.

Usage: `lsc [user "username" password "*****" email "user@physics.edu"] [getMetadata -externalprotocol URL -externaldata_format -query (SQL)]`

Options/Description:

- externalprotocol:** HTTP, FTP, HTTPS, FILE, PORT
- The argument to the **-externalprotocol** option conforms to the usual browser conventions for URLs for determining the location of the results of the user request.
- The **mailto** and **port** options result in an archive format of data to the user.
- The **http**, **ftp**, and **file** options cause a URL pointer to the location of the results to be returned by e-mail.
- The possible formats of the arguments are:
 - `http://hostname/directory/...`
 - `ftp://hostname/directory/...`
 - `mailto:username@domain.com`
 - `file://path/... (NOTE: only use "/")`
 - `port://hostname:portnumber`
- NOTE:** Embedded spaces in the arguments to the **-externalprotocol** option will cause the request to fail.
- externaldata_format:** LINE, LIGO_LV
- The argument to the **-externaldata_format** option is the name of the data type to use in instructing the format of the user response.
- The default arguments **LINE** and **LIGO_LV** are the default for the user request.
- query:** ONLY VALID SQL STATEMENTS
- The argument of **-query** option is any valid sql statement supported by the LIGO database server (MS SQL). If the query cannot run to be executed, a database error message is returned.

Examples of getMetadata command:

- to get LIGO registered user file to get all data from table, however a short format and have results delivered by mail, output

Usage: `lsc [user "lsc" password "*****" email "lsc@physics.edu"] [getMetadata -externalprotocol "mailto:lsc@physics.edu" -externaldata_format -query "Select * from username"]`

getFrameData

Retrieves full or reduced raw frame data in several formats either by attaching the resulting file to an e-mail (external email) or by returning an e-mail with a URL pointing to the location of the results (external http or ftp). It is also possible to have the data directly delivered to a local port, which will be the default method for the get to be completed user API.

Usage: `lsc [user "username" password "*****" email "user@physics.edu"] [getFrameData -externalprotocol URL -externaldata_format -time (START) -endtime (STOP) -delayquery (SQL)]`

Options/Description:

- externalprotocol:** HTTP, FTP, HTTPS, FILE, PORT
- The argument to the **-externalprotocol** option conforms to the usual browser conventions for URLs for determining the location of the results of the user request.
- The **mailto** and **port** options result in an archive format of data to the user.
- The **http**, **ftp**, and **file** options cause a URL pointer to the location of the results to be returned by e-mail.
- The possible formats of the arguments are:
 - `http://hostname/directory/...`
 - `ftp://hostname/directory/...`
 - `mailto:username@domain.com`
 - `file://path/... (NOTE: only use "/")`
 - `port://hostname:portnumber`
- NOTE:** Embedded spaces in the arguments to the **-externalprotocol** option will cause the request to fail.
- externaldata_format:** RAW, LINE, LIGO_LV
- The argument to the **-externaldata_format** option is the name of the data type to use in formatting the results of the user request.
- The possible output formats which the system can produce, which include binary format, **raw**, **line**, and **LIGO_LV** (see URL format).
- time:** | START STOP START-STOP |
- The argument of the **-time** option is a list of times, which may consist of mixed individual times and ranges of times. The times in the list MUST be valid sql times (always 9 digits in length, and greater than 00000000 in any case).
- interval:** | S1 S2 S3 S4 S5 S6 S7 |
- The argument of the **-interval** option is a list of intervals which occur according to the file name string of the time file command with the **interval** option.
- delayquery:** | 1 210-217 800 min_hour=12 |
- The argument of the **-delayquery** option is a list of sql channel names and/or a conditional list (including channel or channel included, which are NOT permitted to point to specific sub channels. An argument of "all" will include all channels in case of the time if the interval is specified as "time", or a full list name of the time if the output format is specified as "line").



Tcl Source Code Docs

LDAS-WW
TclDOC powered!

The frame.tcl Library Module

Modification Date: 99.10.15

Table of Procedures

<ul style="list-style-type: none">frame::ackframe::cancelframe::chanlistframe::closeframe::createframe::createChannelframe::createFileframe::createSocketframe::createStreamframe::destroyframe::destroyChannelframe::destroyFileframe::destroySocketframe::destroyStream	<ul style="list-style-type: none">frame::createChannelframe::createFileframe::createSocketframe::createSocketframe::createStreamframe::destroyframe::destroyChannelframe::destroyFileframe::destroySocketframe::destroyStream
--	--

frame.tcl Version 1.0 Wraps the frameAPI so and the genericAPI so and the genericAPI.tcl for use by the frame API
frame.tcl is a list of the last 'n' records of frame file names with absolute paths to the files.

```
package provide frame 1.0
package require frameAPI
package require eval Stream

set ackval {}
set cache {}
set channels {}
set handlebroken {}
set trigger {}
```

Name: frame::chanlist

Description:
Returns a tcl list of the channel names in the frame.
Will work on a frame file, if given the filename, or a frame_p to data from readFrame

Parameters:

- target - the name of a frame, or a frame data pointer.

Usage:

```
set chanlist [ frame::chanlist $object ]
```

Comments:
A frame data pointer looks like _XXXXX_Frame_p.
The regular expression must be "false" for some reason.

```
proc frame::chanlist { | target "" } {
    setopt -category -- $target
    {} | JMW a null string
    return {}
}

{ "_[0-9a-z]*_Frame_p" | JMW a frame pointer
    if { [ ! $target ] } {
        set chanlist [ getChannelList $target ]
        set {} |
        return -code error "frame::chanlist: $target"
    }
}

default { JMW a file name, or pathname
    if { [ ! file exists $target ] } {
        set msg "frame::chanlist: file not found \"$target\""
        return -code error $msg
    }
}

if { [ ! $target ] } {
    set target [ frame::fileOfId $target ]
    set chanlist [ getChannelList $target ]
    destructFrame $target
} else {
    return -code error "frame::chanlist: $target"
}
}

return $chanlist
```



C++ Source Code Docs

TCL Extended Frame Functions - Microsoft Internet Explorer

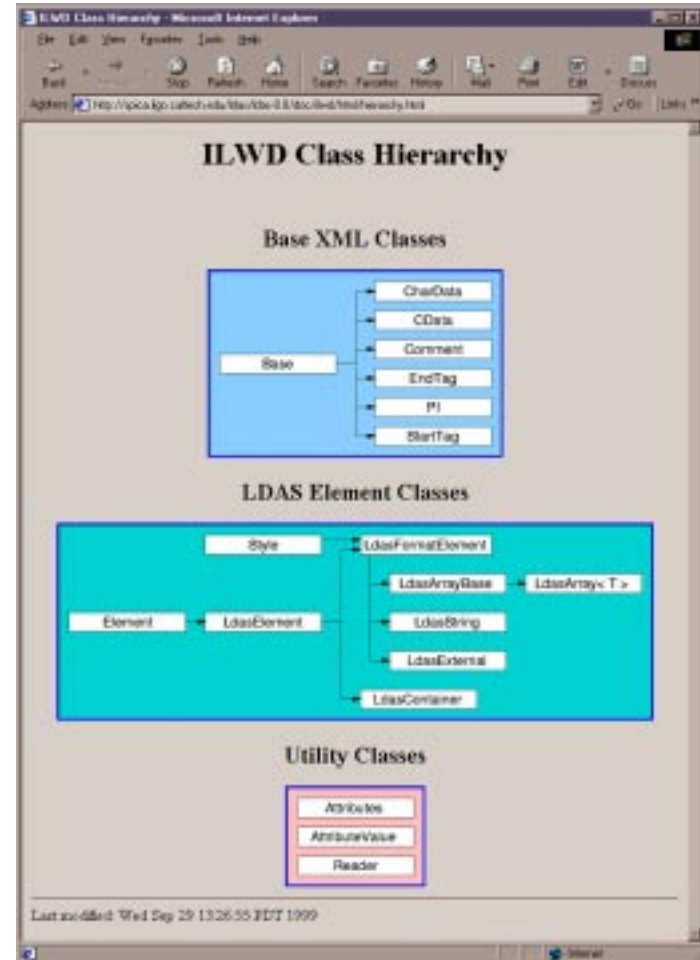
Address: <http://www.ligo.caltech.edu/Mac/00/doc/frame/PL/html/Frame.html>

Frame API

TCL Extended Frame Commands

- [openFrameFile](#) Opens a frame file for reading/writing
- [closeFrameFile](#) Closes a frame file
- [readFrame](#) Reads a frame from a file.
- [writeFrame](#) Writes a frame to a file.
- [destroyFrame](#) Deletes a frame object from memory.
- [getFrameAttribute](#) Returns an attribute of a frame.
- [getFrameData](#) Returns frame data.
- [getChannelList](#) Get ADC Channel List
- [createFrame](#) Creates an empty FrameCPP Frame object as ILWD.
- [createMag](#) Creates a FrameCPP Mag object as ILWD.
- [createHistory](#) Creates a FrameCPP History object as ILWD.
- [createRawData](#) Creates an empty FrameCPP RawData object as ILWD.
- [insertFrameData](#) Insert ILWD Frame Data into an ILWD frame
- [concatData](#) Concatenate Data
- [toFrame](#) Convert an ILWD frame to a FrameCPP frame.
- [sendFrame](#) Send Frame
- [recvFrame](#) Receive Frame
- [sendFrameBinary](#) Send Frame as File
- [recvFrameBinary](#) Receive Frame as File

Done Internet





More C++ Docs

http://www.ligo.caltech.edu/files/Doc/0.8/Doc/Agreement/Agreement.html - Microsoft Internet Explorer

Name: sendElementObject Send an Element as binary

Description:
This command reads an Internal LIGO Light-Weight Data set, called an Element because of its relationship to XML elements, through a Data Socket as a C++ Object. This method of reading Elements is used often because of efficiency.

Usage:
sendElementObject ptSock ptElem

Parameters:

Parameter	Description
ptSock	a pointer to a Data Socket which has previously been opened with the createDataSocket command
ptElem	a pointer to an Element object that has previously been instantiated in the C++ layer

Return value:
none

Exceptions:

Exception	Description
connected_socket	the socket doesn't exist
unconnected_socket	the socket isn't connected
invalid_element	the element doesn't exist

[6.6.3](#)

Name: sendElementObject_t Send element as binary - Threaded

Description:
This command runs the sendElementObject command as a thread.

Usage:
on t1 (sendElementObject_t ptSock ptElem)

Parameters:

Parameter	Description
ptSock	a pointer to a Data Socket which has previously been opened with the createDataSocket command
ptElem	a pointer to an Element object that has previously been instantiated in the C++ layer

Return value:
t1 - A pointer to the thread which was started.

[6.6.3](#)

Description of Comment - Microsoft Internet Explorer

class Comment : public Base

XML Comment.

This class represents an XML comment.

Members

Public

- Comment() const Comment& c) throws (bad_alloc) : Copy constructor
- Comment(const Comment& c) throws (bad_alloc) : Copy constructor
- Comment(Reader& r) throws (bad_alloc, StreamException, FormatException) : Input constructor
- virtual ~Comment() throws () : Destructor
- const Comment& operator=(const Comment& c) throws (bad_alloc) : Assignment operator
- bool operator==(const Comment& c) const throws () : Equal comparison operator
- bool operator!=(const Comment& c) const throws () : Inequality operator
- const string& getComment() const throws () : Returns the comment
- ClassType getClass() const throws ()
- void setComment(const string& c) throws (bad_alloc, FormatException) : Sets the comment
- virtual void write(osstream& stream) const throws (exception) : Writes a comment
- virtual void read(Reader& r) throws (StreamException, FormatException) : Input constructor

Friends

- friend void os_read(os_bstream& Comment&)

Comment(const Comment& c) throws (bad_alloc)

Copy constructor.

Parameters:

Parameter	Description
const Comment& c	The object to copy from



API Log Files

```
The manager API Log File - Microsoft Internet Explorer
http://www.ligo.caltech.edu/~peters/api.log

The manager API Log File
624566653 openLog /usr/petersa/garbage/log/lbk3manager.log.html (files)
624566653 openListenSock port 10001 (operator) opened as mearis as sock?
624566653 openListenSock port 10002 (operator) opened as mearis as sock?
624566653 hplLoop Listening process watchdog started
624566653 hplLoop Listening process watchdog started
624566653 hplLoop PID: manager (624566652) -> frame (624566652): 138029 kb
624566653 hplLoop Listening process leak started
624566653 LeakLogger initial size of manager API: 4032 KB
624566653 LeakLogger total (4032 KB) heap 1418K stack 72K libmanager ()
live () libcc1 544K
624566654 openLog /usr/petersa/garbage/log/lbk3debug0.log.html (files)
624566654 createNamespace Namespace "debug0" created
624566655 debug0:genid opened channel sock12 to mearis port 10000
624566657 debug0:sockhandler Your results are available as files: C1-624566554.D
see: http://ligo.caltech.edu/~petersa/idea_outfiles/debug0/
624566657 debug0:log Stack "debug0:nom" exhausted.
624566657 debug0:close closed channel sock12
624566658 closeLog /usr/petersa/garbage/log/L242debug0.log.html (files) closed
624566658 debug0:delete assistant manager "debug0" destroyed.
624566659 openLog /usr/petersa/garbage/log/lbk3debug1.log.html (files)
624566659 createNamespace Namespace "debug1" created
624566659 debug1:genid opened channel sock12 to mearis port 10000
624566659 debug1:sockhandler Your results are available as files: C1-624566564.D
see: http://ligo.caltech.edu/~petersa/idea_outfiles/debug1/
624566659 debug1:log Stack "debug1:nom" exhausted.
624566659 debug1:close closed channel sock12
624566659 closeLog /usr/petersa/garbage/log/L242debug1.log.html (files) closed
624566659 debug1:delete assistant manager "debug1" destroyed.
624566736 openLog /usr/petersa/garbage/log/lbk3debug2.log.html (files)
624566736 createNamespace Namespace "debug2" created
624566736 debug2:genid opened channel sock12 to mearis port 10000
624566739 debug2:sockhandler Your results are available as files: C1-62456706.R
see: http://ligo.caltech.edu/~petersa/idea_outfiles/debug2/
624566739 debug2:log Stack "debug2:nom" exhausted.
624566739 debug2:close closed channel sock12
624566739 closeLog /usr/petersa/garbage/log/L242debug2.log.html (files) closed
624566739 debug2:delete assistant manager "debug2" destroyed.
624566744 openLog /usr/petersa/garbage/log/l242debug3.log.html (files)
624566744 createNamespace Namespace "debug3" created
624566744 debug3:genid opened channel sock12 to mearis port 10000
624566747 debug3:sockhandler Your results are available as files: C1-62456740.R
see: http://ligo.caltech.edu/~petersa/idea_outfiles/debug3/
624566747 debug3:log Stack "debug3:nom" exhausted.
```

```
The frame API Log File - Microsoft Internet Explorer
http://www.ligo.caltech.edu/~peters/api.log

The frame API Log File
624566634 openLog /usr/petersa/garbage/log/lbk3frame.log.html (files)
624566639 frame:cache cache 10001 files onto frame:cache last frame created
see: 624566611 time now is: 624566610
624566639 openListenSock port 10003 (operator) opened as mearis as sock?
624566639 openListenSock port 10004 (operator) opened as mearis as sock?
624566639 hplLoop Listening process leak started
624566648 LeakLogger initial size of frame API: 20224 KB total (20224 KB) heap
4032K stack 44K libframe 1432K generic 482K lib 1100K libcc1 544K
624566648 hplLoop Listening process hplframe started
624566648 hplLoop Listening process frame_diag started
624566648 frame:diag frame:cache has 10001 elements. element
(1): /home/petersa/frame/Data5/C1-624566639.F element
(10001): /home/petersa/frame/Data5/C1-624566639.F delta_n: 10000
624566652 metaOpen metaOpen debug: "FrameAPI:empty load [ genidAPI ]"
624566655 metaOpen metaOpen debug: "-dataquery all -jobid debug -returnframe
live -userToUserData C1 -time 624566654 -returnToUserData C1-624566654.F
624566655 frame:find frame:find: /home/petersa/frame/Data5/C1-624566654.F
624566655 frame:containerData dumped 280160 LibContainer p as lib?
to /home/petersa/public_html/idea_outfiles/debug3/C1-624566654.R
624566657 LeakLogger LEAKS 5536 KB in 10 seconds (net) total (20560 KB) heap
9744K stack 144K libframe 1432K generic 490K lib 1000K libcc1 544K
624566659 metaOpen metaOpen debug: "-dataquery all -jobid debug -returnframe
live -userToUserData C1 -time 624566594 -returnToUserData C1-624566594.F
624566659 frame:find frame:find: /home/petersa/frame/Data5/C1-624566594.F
624566659 frame:containerData dumped 280160 LibContainer p as lib?
to /home/petersa/public_html/idea_outfiles/debug3/C1-624566594.R
624566657 LeakLogger LEAKS 5536 KB in 80 seconds (net) total (20560 KB) heap
9744K stack 144K libframe 1432K generic 490K lib 1000K libcc1 544K
624566727 metaOpen metaOpen debug: "-dataquery all -jobid debug -returnframe
live -userToUserData C1 -time 62456727 -returnToUserData C1-62456727.F
624566729 frame:find frame:find: /home/petersa/frame/Data5/C1-62456729.F
624566729 frame:containerData dumped 280160 LibContainer p as lib?
to /home/petersa/public_html/idea_outfiles/debug3/C1-62456729.R
624566729 LeakLogger LEAKS 5536 KB in 90 seconds (net) total (20560 KB) heap
9744K stack 144K libframe 1432K generic 490K lib 1000K libcc1 544K
624566744 metaOpen metaOpen debug: "-dataquery all -jobid debug -returnframe
live -userToUserData C1 -time 62456744 -returnToUserData C1-62456744.F
624566744 frame:find frame:find: /home/petersa/frame/Data5/C1-62456744.F
624566744 frame:containerData dumped 280160 LibContainer p as lib?
to /home/petersa/public_html/idea_outfiles/debug3/C1-62456744.R
624566747 LeakLogger LEAKS 5536 KB in 100 seconds (net) total (20560 KB) heap
9744K stack 144K libframe 1432K generic 490K lib 1000K libcc1 544K
624566759 metaOpen metaOpen debug: "-dataquery all -jobid debug -returnframe
live -userToUserData C1 -time 62456759 -returnToUserData C1-62456759.F
624566759 frame:find frame:find: /home/petersa/frame/Data5/C1-62456759.F
```



System Testing

□ ILWD Class Object Socket Communication

- ⇒ LDAS Internal Light Weight Data C++ Object Transmission Test
- ⇒ Uses two LDAS generic APIs
- ⇒ Time to serialize C++ objects, send out socket, receive on other end, instantiate ILWD object, re-serialize object, send back to source, and re-instantiate ILWD object between two LDAS APIs

□ Database Record Insertions Rates

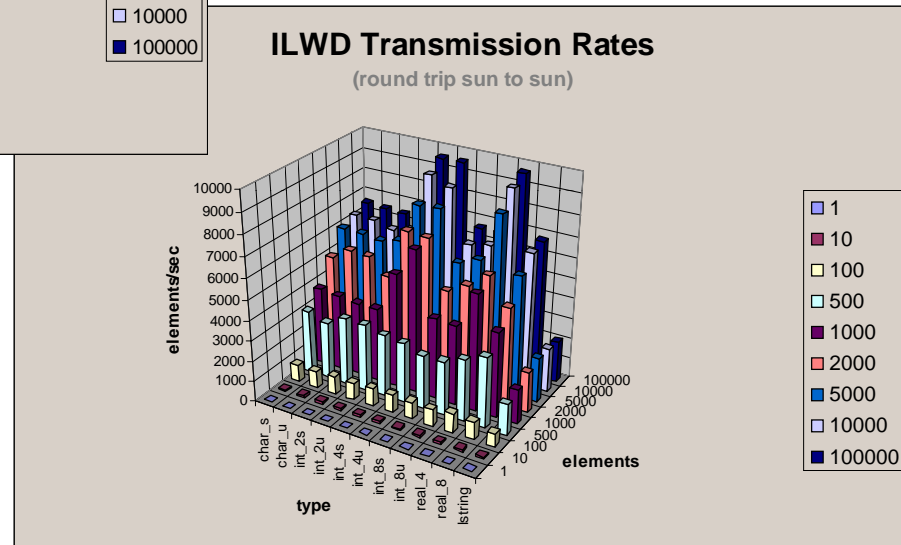
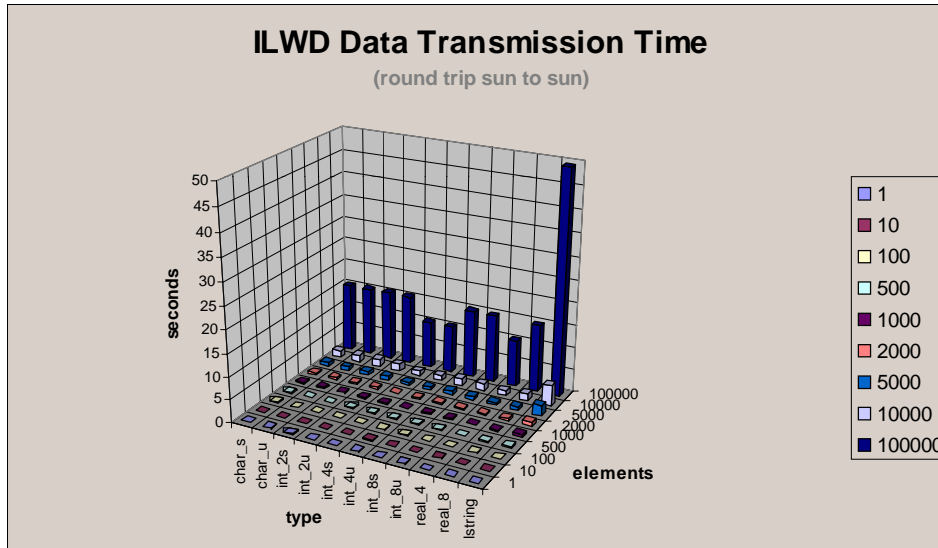
- ⇒ LIGO Metadata Insertion Rates into DB2 tables
- ⇒ Uses managerAPI, metaDataAPI, OpenLink ODBC level 2.5 driver, and DB2 server
- ⇒ Times SQL inserts on "N" rows of each LIGO Table

□ Frame to ILWD Translation Rates

- ⇒ LIGO Frame Files to ILWD translation rates for various channel cuts
- ⇒ Uses frameAPI and frames from Hanford's framebuilder

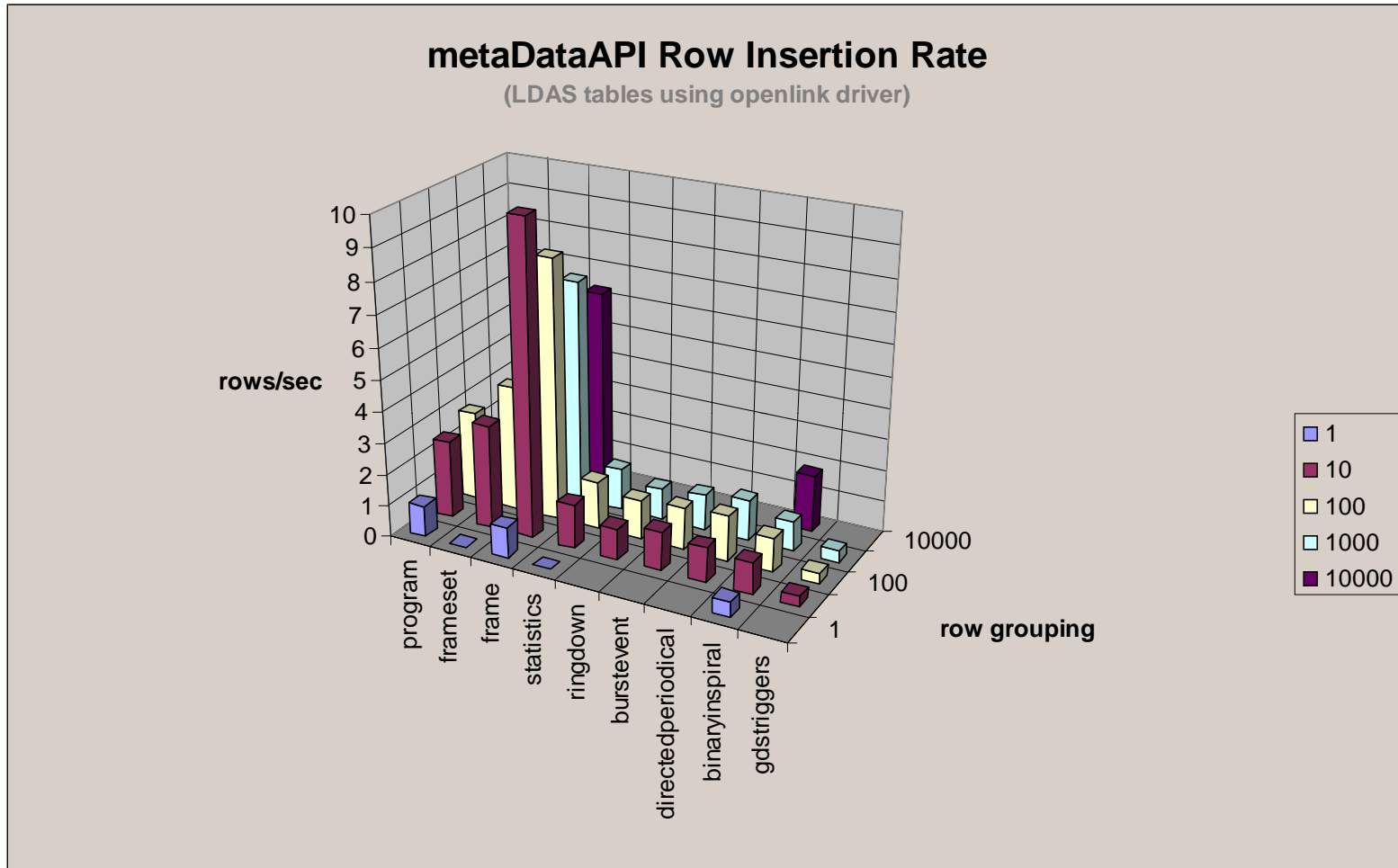


Sun Data Socket Tests



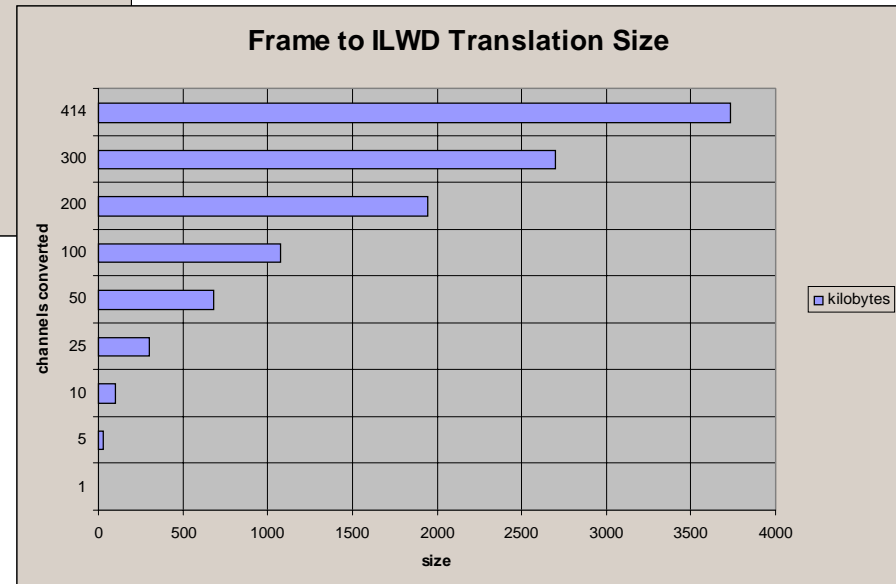
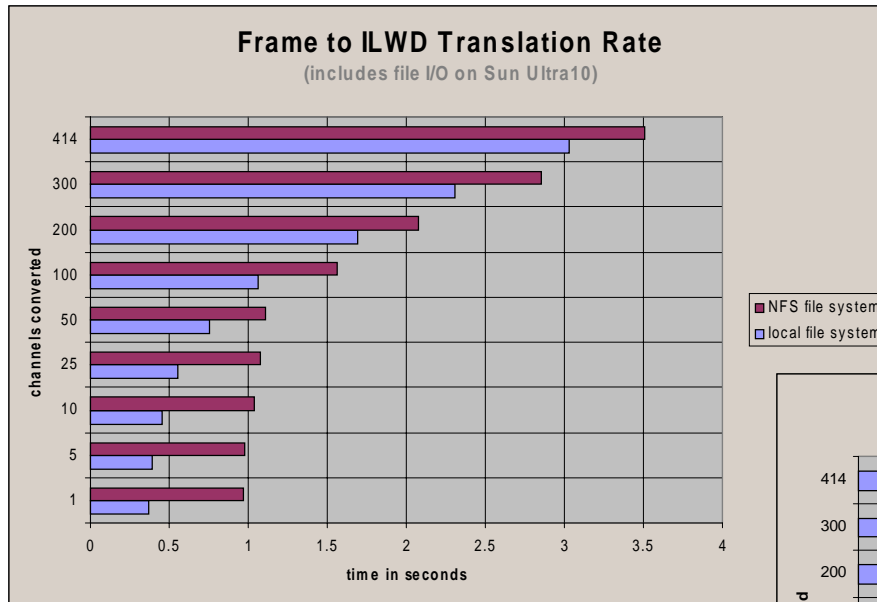


Metadata Ingestion Tests





Frame to ILWD Tests





Next Major Development

Parallel Computing Components

□ mpiAPI:

- ⇒ detailed requirement published
- ⇒ to be written in TCL (no C++)
- ⇒ starts up mpirun script
- ⇒ maintains parallel queues
- ⇒ manages load balancing
- ⇒ communicates messages with the wrapperAPI

□ wrapperAPI:

- ⇒ developing detailed requirements
- ⇒ is started by mpirun scripts
- ⇒ to be written in C++ (no TCL)
- ⇒ using MPICH 1.1.2
- ⇒ implementing lidas communicator class for load balancing
- ⇒ implementing generic algorithm class wrapper
- ⇒ master communicates with mpiAPI
- ⇒ slaves carry out parallel computing



Closing Remarks

- ❑ Even though measurable progress has been demonstrated, LDAS implementation has been very slow in starting up, requiring significant guidance over a young programming staff to implement the nearly complete LDAS software infrastructure.
- ❑ The modular design of the LDAS software will most likely facilitate more rapid code development once junior staff come up the learning curve and more complete software development documentation becomes available on the software developers' web pages.