

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

Document Type: **LIGO-T980023-00 - Cxx** 3/03/98

**A Reference of Data Server Library for
Data Acquisition System**

H. Ding

Distribution of this draft:

This is an internal working note
of the LIGO Project..

California Institute of Technology
LIGO Project - MS 51-33
Pasadena CA 91125
Phone (626) 395-2129
Fax (626) 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/>

LIGO DRAFT

1 INTRODUCTION

The data server library is designed for the software development for LIGO data acquisition system. It is used in conjunction with the DAQD (Data Acquisition Daemon) data server to give an easy access to the data acquired by the data acquisition system. This document gives a guide of the usage of the library. A list of the function calls will be explained and examples will be provided at the end.

2 CLIENT FUNCTION CALLS

2.1. Connecting to the Data Server

- Build the connection:
int DataConnect(char* serverName, int serverPort, int userPort, void* read_data());
return: integer; 0 if successful.
serverName: string; the IP address where DAQD data server is running.
serverPort: integer; should be 0 unless a special server's TCP port number is needed.
userPort: integer; user's TCP port number; if more than one data server-client type program runs on the same machine at same time, different port numbers should be used.
read_data(): function; to receive requested data; this function is called whenever a data request starts; see examples for the implementation.
- Quit the connection:
void DataQuit();

2.2. Setting Data channels

- Set data channels for data requesting:
int DataChanAdd(const char* chanName, int dataRate);
return: integer; total number of channels for data requesting, or -1 if fails (the requested channel is not available).
chanName: string; the name of the channel to be added, or "all" to add all configured channels.
dataRate: integer; to specify the rate of the data wanted; input 0 if full sample rate needed, or an integer of power of 2 which is not larger than the sample rate of the channel; if chanName is "all" then dataRate will be ignored and full sample rate of data will be sent.
- Delete data channels from data requesting:
int DataChanDelete(const char* chanName);
return: integer; total number of channels for data requesting.
chanName: string; the name of the channel to be deleted, or "all" to delete all the channels.

2.3. Requesting Data

- Request on-line data:
unsigned long DataWriteRealtime();
return: unsigned long; the process ID number.
- Request off-line data:
unsigned long DataWrite(char* startTime, int duration);
return: unsigned long; the process ID number.
startTime: string; the starting time for data requesting; in the format of "yy-mm-dd-hh-mm-ss", e.g., "98-03-02-15-05-30".
duration: integer; to specify the length of the period in seconds for data requesting.

2.4. Requesting Trend Data

- Request on-line trend data:
unsigned long DataWriteTrendRealtime();
return: unsigned long; the process ID number.
- Request off-line trend data:
unsigned long DataWriteTrend(char* startTime, int duration);
return: unsigned long; the process ID number.
startTime: string; the starting time for trend data requesting; in the format of "yy-mm-dd-hh-mm-ss"; e.g., "98-03-02-15-05-30".
if startTime = "0" or "00-00-00-00-00-00", the starting time will be set at the current time minus duration.
duration: integer; to specify the length of the period in seconds for trend data requesting.

2.5. Stop Data Requesting

- Stop data requesting:
void DataWriteStop(unsigned long processID);
processID: unsigned long; the ID number of the process to be stopped.

2.6. Receiving Data

These function calls should be used inside the function `read_data()` (see 2.1. `DataConnect` and the examples for more details).

- Start to receive data:
int DataReadStart(int bufSize);
return: integer; 0 if on-line data received, or the number of blocks of data received.
bufSize: integer; used for allocating some space to hold received data.

- Receiving data:
int DataRead();
return: integer; number of bytes received; <0 if error occurs.
- Stop receiving data and free the space allocated by DataReadStart:
void DataReadStop();

2.7. Looking at Data

- Get data by channel name:
int DataGetCh(const char* chanName, short data[]);
return: integer; rate of the data received.
chanName: string; name of the channel to look at.
data: array of short; data output.
- Get trend data by channel name:
int DataTrendGetCh(const char* chanName, struct DTrend *trendData);
return: integer; 0 if successful.
chanName: string; name of the channel to look at.
trendData: pointer to struct DTrend; trend data output.
struct DTrend { int min; int max; int rms; } (defined in datasrv.h).
- Get the time-stamp:
void DataTimestamp(char* timestamp);
timestamp: string; output; time -stamp of the last received data; in the format of "yy-mm-dd-hh-mm-ss".

2.8. More Information

- Get the names and rates of all the configured channels:
int DataChanList(struct DChList allChan[]);
return: integer; total number of the configured channels.
allChan: array of struct DChList; output for all configured channel names and rates.
struct DChList { char name[MAX_CHANNEL_NAME_LENGTH+1];
int rate; } (defined in datasrv.h).
- Get the sample rate for a given channel:
int DataChanRate(const char* chanName);
return: integer; sample rate of the channel.
chanName: string; name of the channel.

3 EXAMPLES

3.1. example 1

This is an example of using data server library. It connects to the data server, requests on-line data for the channel "IFO_DMRO" at the rate of 16, print the data, and shut down after 30 seconds.

The necessary library files are currently at /home/hding/Cds/Data/Server/Lib/.

Compile example1.c with cc -o example1 example1.c datasrv.o daqc_access.o -lpthread -lsocket -lnsl .

```

/* example1.c */
#include "datasrv.h"
#define LISTENER_PORT 9000 /* Listener listens on this TCP port */
#define DAQD_HOST "131.215.114.15" /* data server machine cdssol5 */
#define BUFSIZE 2000000

char timestring[24];
void* read_data();

void main(int argc, char *argv[])
{
    int i, j;
    char ch[2];
    unsigned long processID = 0;

    if ( DataConnect(DAQD_HOST, 0, LISTENER_PORT, read_data) != 0 ) {
        fprintf ( stderr, "connection failed\n" );
        exit(1);
    }
    DataChanAdd( "IFO_DMRO", 16);
    /* OR: DataChanAdd("all", 0); */
    processID = DataWriteRealtime();
    sleep(30);
    DataWriteStop(processID);
    DataQuit();
    exit(0);
}

/* Thread receives data from DAQD */
void* read_data()
{
    unsigned long blocks;
    short data[16];
    int rate, j;

    if ( !(blocks = DataReadStart(BUFSIZE)) )
        fprintf ( stderr, "receiving data blocks online\n" );
    else

```

```
fprintf(stderr, "receiving %d data blocks!\n", blocks);

while ( 1 ) {
  if ( DataRead() < 0 ) {
    DataReadStop();
    break;
  }

  /* Process received data here */
  DataTimestamp(timestamp);
  printf ("timestamp=%s\n", timestamp );
  rate = DataGetCh("FO_DMRO", data );
  for (j=0; j<rate; j++) {
    printf ("data[%d]=%d ", j, data[j] );
  }
  printf ("\n");
}
fflush (stdout);
return NULL;
}
```

LIGO-DRAFT