CDMS DAQ Software User Manual

UCSB HEP CDMS DAQ Group

Ron Ferril
Contents

1 Introduction .................................................. 3
  1.1 Contents of this Document ............................ 4
  1.2 The DAQ Software in Relation to Hardware ........ 4
  1.3 Using this Document .................................. 6

2 CAEN Server, CAEN GUI and Related Software .......... 7
  2.1 The CAEN Server ...................................... 8
  2.2 Starting and Stopping the CAEN Server ............. 8
  2.3 The Log File .......................................... 9
  2.4 The CAEN Configuration File ......................... 10
  2.5 The CAEN Configuration and Recovery Clients .... 11
  2.6 The CAEN GUI ...................................... 12
  2.7 The CAEN Remote GUI ............................... 14
  2.8 Methods of the CAEN Server ......................... 15
  2.9 Deprecated Methods ................................ 16

3 Data Server and Data Server Lite ...................... 17
  3.1 Brief Introduction to the CDMS II Event Format ..... 18
  3.2 Two Servers: the Data Server and the Data Server Lite 18
  3.3 Starting the Data Server ............................ 20
  3.4 Starting the Data Server Lite ....................... 20
  3.5 GUI’s and Other Clients ............................ 21
  3.6 Methods of the Data Server and Data Server Lite .... 23
  3.7 Deprecated Methods ................................ 24

4 Joerger Monitor System (JMS) ......................... 25
  4.1 The JMS Server .................................... 26
  4.2 Starting the JMS Server ................................ 28
  4.3 GUI’s and Other Clients ............................ 29
  4.4 Methods of the JMS Server ........................... 30
  4.5 Deprecated Methods ................................ 31
Chapter 1

Introduction

This chapter introduces the purpose and general content of this document, and gives some introduction to some CDMS DAQ software.
1.1 Contents of this Document

This document explains how to use some CDMS DAQ software I have written. The text assumes the reader has access to the DAQPackage software. This manual is intended for use by users of the software and by programmers. Programmers who wish to write client programs for interfacing with servers included in DAQPackage, or who wish to upgrade or otherwise modify the software, may find this manual useful when combined with some study of the source code. The source code contains comments that explain much more about the internal operations of the software.

Because programmers may modify the source code, this manual may often need revision. Since the manual was written in the LaTeX2e language and then converted from there to other formats (e.g., HTML, DVI, PostScript and PDF), I request that programmers who modify the program also update this manual by editing the LaTeX2e code. If you edit this manual, you should add your name to the first page (which serves as a title page) of this manual and modify the page header to indicate the revision. Notice that the page headers contain indications of the revision date and the author as being me, Ron Ferril. You may want to insert your name there or simply insert “and others” there. The date shown in the header should show the date of the revision. This whole document may be incorporated into other CDMS works. For example, it may be useful to have a manual for the full DAQPackage software rather than just for the software I have written. This version is not even the original version, but is a new edition based on the original. Changes in the software cause the need for revisions of this manual.

Since the software described is not exclusively mine, others may already be changing the source code I have written. Thus, it is possible this manual may become obsolete before this revision is finished. Even as versions of this manual are written, changes in the DAQ software occur. This may be evident in some of the chapters. I would like for the programmers who continue upgrading the software to try to keep this manual current. Also, I ask that people changing the source code continue updating the comments in the source code so future programmers can still follow the plan of the software.

1.2 The DAQ Software in Relation to Hardware

The main purpose of the DAQ software is to obtain data from the CDMS II experiment. This involves the control of hardware and acquiring data from
the hardware. The plan is to allow remote control or remote monitoring of the experiment. Thus, remote clients, including GUI programs, interact with servers that perform actions involving hardware. This involves the use of Java RMI. Some servers also involve CORBA, but the use of CORBA is now restricted to specific uses.

The DAQ software was assembled into a repository or archive called DAQ-Package. Copies of this archive are checked out or distributed throughout CDMS. Although I originally developed several software systems for DAQ, only some of them are planned for use in the experiment. Those systems that remain include the CAEN Server, the Data Server, the Data Server Lite and the Joerger Monitor System (JMS) software. Those being discontinued include the “VME Local Slow Control” (VMELSC) system and various utilities. Some of the systems may be re-named soon (or even before this manual is distributed).

The CAEN Server and related software allow users to control the CAEN high-voltage power supply. This includes configuration and remote monitoring of the power supply. The CAEN Server allows clients to access its methods by use of Java RMI, and involves source code written in Java and C++. The C++ code refers to software libraries from the CAEN manufacturer. These libraries are C libraries. The server involves certain software “bridges” written by Chris Savage to indirectly access special CORBA event channels. The scheme of Java Native Methods is used to interface the Java and C++ parts of the server.

The Joerger Monitor System (JMS) should probably be re-named. This software allows a user to remotely control and monitor the digitizers in the VME crates at the experiment site. The name “Joerger” came about because the original digitizers were only Joergers. The JMS software uses Java RMI for communications between clients and the JMS server. The old CORBA features of the JMS server are no longer needed and may have already been removed.

The Data Server and the Data Server Lite monitor data records generated by a DAQ software system called the Event Builder. These generated records contain event data which is the result of events in detectors. Clients and GUI programs can use the Data Server and Data Server Lite to monitor the event data. These servers are written in Java and use Java RMI for their communications with clients. The CORBA features of the Data Server are no longer needed and are planned for removal.

Thus far you have seen the idea of clients remotely accessing servers. The servers operate at the experiment site (Soudan). Because of some security concerns, certain restrictions have been established regarding external access to the servers running at the site. One of these restrictions is that remote
clients running outside the site cannot directly access the servers but must indirectly access them through interactions with special manager or control software. Without such interaction, the clients need to be run at the site in order to access the servers. Thus, the manager software controls access to the servers. This restriction was unknown to me. None of the clients I wrote interacted with this manager software and, thus, I found that they were not allowed to access the servers without being run at the experiment site. However, at least one of my clients (the CAEN Remote GUI) has been modified by Rupak to use the special manager software to remotely access a server by use of the special manager software. This “manager” software is part of the Run Control system and is beyond the scope of this manual.

1.3 Using this Document

The chapters of this document relate to various parts of the DAQPackage software. Features of programs and scripts are explained. These features include the correct starting and shutting down of servers, the use of clients, and software interfaces the servers have for allowing clients to interact with them. The chapters dealing with specific DAQ software systems have sections describing the starting and stopping of servers, and either the use of clients or the software interfaces provided for clients.

In regards to clients, my decisions for what to document (about the clients) in this manual were based partly on who wrote the clients. I did not intend to document software written by other members of the DAQ team. Thus, when I wrote a server and other people intended to write the GUI clients, I found it good to document the software interfaces the clients could use to access the server. If I was to write the GUI client also (as in the original CAEN GUI), I thought I should also document the use of the GUI. Some of the clients I provided are for testing purposes. These were necessary because of questions people had regarding suspected problems with servers. The test clients allowed me to determine whether the problems were in the servers or with a client somebody was using. This is important since clients frequently change.

The reader of this manual should be careful to notice mention of a feature as being either “deprecated” or likely modified soon. The “deprecated” features are not maintained and may already malfunction if used. Eventually, they may be removed from the DAQPackage. Reasons why I would suspect certain features to be “likely modified soon” include discussions I have already heard and the possibilities that others are already working on modifications.
Chapter 2

CAEN Server, CAEN GUI and Related Software

This chapter deals with DAQ software for the CAEN power supply.
2.1 The CAEN Server

The CAEN Server allows remote control and monitoring of the CAEN high-voltage power supply. The CAEN manuals (from CAEN) are useful for using clients with the CAEN Server. This is because the server adjusts settings of the power supply and the functions of these settings are described in CAEN manuals. I do try to briefly describe the functions of all the settings in this document and my descriptions may be enough for most operators of the CDMS DAQ CAEN software.

The CAEN Server provides services for clients, and also broadcasts on a CORBA event channel by using Chris Savage’s (Data) Monitoring Bridge. It also broadcasts errors on an event channel by using Chris Savage’s Error Bridge.

2.2 Starting and Stopping the CAEN Server

The CAEN Server can be started using the script “CAENServer.script” found in the “scripts” directory of DAQPackage. This script allows command-line parameters to be passed to the CAEN Server. These parameters specify which modes the server is to use. Currently, there are three special parameters that can be entered on the command line: log, nolog and test. The “log” parameter indicates the server is to generate and maintain a log file. This is the new default operation of the server. Using “nolog” (instead of “log”) prevents the server from generating or maintaining the log file. The log file is generated in a directory called “logs” and placed in the same directory in which DAQPackage sits. If the “logs” directory does not exist, the server will create it. Thus, entering

CAENServer.script log

will launch the CAEN Server and instruct it to generate a log file in the “logs” directory just outside of DAQPackage. If you are using a type of Bash shell, you may need to enter

./CAENServer.script log

to launch the script. The same applies to the following discussion of using CAENServer.script to start the CAEN Server. The log file is in text format. Thus an ordinary text editor can be used to view its contents. Entering

CAENServer.script nolog
will launch the CAEN Server but will instruct it to operate without involving a log file. Entering

```
CAENServer.script
```

will cause the server to begin with its default behavior which currently is to generate the log file.

The “test” command-line parameter is used to cause the CAEN Server to run in debug mode. In this mode, the server will print messages (mostly on standard error output). The “test” parameter can be used with “log” or the “nolog” parameter. Entering

```
CAENServer.script test
```

will cause the CAEN Server to run in debug mode in which it prints messages on standard output or standard error. Command-line parameters can be combined as in

```
CAENServer.script log test
```

and

```
CAENServer.script nolog test
```

to specify debug mode and which log-file mode to use. The debug mode is for use in determining problems, but should not be used for regular running. The printing of so many messages can cause problems such as slower response to clients.

I recommend that the CAEN Configuration Client (described later) be run after the CAEN Server is started. The client directs the server to configure itself for normal operation. Until the server is fully configured by such a client, it may report incorrect channel names and issue unnecessary warnings.

### 2.3 The Log File

As already mentioned, the script “CAENServer.script” allows a command-line parameter “log” to instruct the CAEN Server to generate a log file. The generated log file is in text format. An ordinary text editor or text viewer can be used to view its contents. Lines of data start with the time in milliseconds from midnight, January 1st, 1970, universal time (UTC). The values on a line of text are separated by tabs. This allows importing into spreadsheets or into databases.
2.4 The CAEN Configuration File

The CAEN Server can be told to configure itself or the CAEN power supply using a configuration file. Methods for specifying the file are described elsewhere in this chapter. This section deals with the structure of this configuration file. The names given for the CAEN parameters are defined by the manufacturer, CAEN, in the manuals for the power supply.

The configuration file is a text file. Each line of text, other than lines that merely give comments, gives a different parameter for the operation of the CAEN Server or the operation of the CAEN power supply. The first four lines of text are ignored (i.e., skipped) by the CAEN Server because they are reserved for comments. The next three lines give, in order, the following numbers:

1. The number of high-voltage boards to be used in the power supply.

2. The number of high-voltage output channels on each of the high-voltage boards. The product of this number and the number of boards is the number of “channel records” given in this configuration file. (Channel records are described below.)

3. The maximum voltage deviation (in volts) allowed before the server sets warnings regarding the deviations.

The next four lines are ignored because they are for additional comments. After these four lines, the channel records begin. Each channel record consists of ten lines. These lines give, in order, the following parameters:

1. The channel name. This is not the channel name set on the power supply itself, but a name the CAEN Server uses for the channel. When a client requests the channel names, the names set by the configuration file are the ones returned. These channel names are not set until the CAEN Server is told to configure itself or the power supply using this configuration file. The names returned to clients before configuration are strings containing several question marks (“?”).

2. The V0Set value which indicates the voltage (in volts) to be on the channel when that channel is “ramped up.” (The server does not use the V1Set value that CAEN also specifies as an alternative voltage to use.)

3. The I0Set value which indicates the current in microamps the channel is to be able to drive without tripping.
4. The RUp value which indicates the ramp-up rate in volts per second.

5. The RDwn value which indicates the ramp-down rate in volts per second.

6. The Trip value which indicates the suggested time to wait before tripping when the current is too high. This time is given in seconds.

7. The POn value which indicates the desired ramp state when a power supply is restarted. We usually set this to 0 to make sure the channel starts up ramped down and stays ramped down until directed to ramp up. (A 1 would allow the channel to return to what its state was before the restart.)

8. The PDwn value which indicates a particular mode in which the power supply is supposed to ramp when a trip occurs because of the current being too high. We usually use the value 1 for this flag to specify that the voltage is to ramp down rather than just drop quickly when a trip occurs.

9. The Pw value which is either 0 or 1. If 0, the power supply is to ramp down at the rate set by RDwn. If 1, the supply is to ramp up at the rate set by RUp.

10. A blank line that is ignored by the server other than to count it as the last line of the channel record.

Other than the tenth line of the channel record, there are no lines between the channel records. (The view adopted so far is that each channel record consists of ten lines with the tenth line being blank and no lines between channel records. Another view would be that the channel records consist of nine lines with one ignored line between them.) The server ignores lines of text that follow the last channel record.

### 2.5 The CAEN Configuration and Recovery Clients

A configuration client is provided for directing the CAEN Server to configure itself, and a separate recovery client is provided to direct the CAEN Server to configure itself and the CAEN power supply. Both configuration operations use a CAEN configuration file. The clients can also tell the server which file to use as the configuration file. To operate the CAEN Configuration Client from the scripts directory of DAQPackage, you enter
CAENConfigureClient.script [name of configuration file]

where the name of the configuration file includes the path. Thus if the
configuration file is /data/HV.txt then you would enter

    CAENConfigureClient.script /data/HV.txt

or (if you use a type of Bash)

    ./CAENConfigureClient.script /data/HV.txt

to use that configuration file to configure the CAEN Server (but not configure
the power supply). If no configuration file is given, a default file is used.

The CAEN Recovery Client must be used only by people in charge of
setting parameters of the CAEN high-voltage power supply. This recovery
client is used in the same way as the configuration client and performs the
same kind of configuration of the CAEN Server. However, the two clients
differ in that the CAEN Configuration Client does not change any settings
of the power supply but the CAEN Recovery Client does direct the CAEN
Server to set parameters of the power supply. This additional feature—the
setting or changing of settings of the power supply—is the reason the recovery
client should be used only by people in charge of setting parameters.

2.6 The CAEN GUI

The CAEN GUI is sometimes called the “Expert CAEN GUI” to distinguish
it from the CAEN Remote GUI or other GUI applications people write to
access the CAEN Server. This expert GUI allows the user to re-configure
the CAEN Server (using a configuration file), ramp up and down the voltage
for either individual channels or all channels, and monitor the settings for
all channels. The CAEN GUI runs correctly only if the CAEN Server is
running. This is because the GUI uses Rupak’s RMI Naming Service to find
the CAEN Server and won’t continue to run properly if it fails to find the
server on the first try.

To start the CAEN GUI, you can enter

    CAENGUI.script

to operate the GUI in the default mode. Command-line parameters can also
be used to affect sizes of the font or table used in the display. The allowable
numbers of command-line parameters are zero, one and three. You should
not use two or four parameters since doing so would confuse the GUI. To try
a font size of 10 (points), you enter
CAENGUI.script 10

The units of the size are points. A font size of “0” is ignored. This allows other command-line parameters to be entered without adjusting the font size.

In order to set the size of the display frame, you provide the width and height in pixels in the next two command-line parameters. For example, to leave the font size to its default and set a frame size of 1200 width by 800 height, you enter

CAENGUI.script 0 1200 800

The “0” indicates that the font size is to be left alone (at its default value). If either the width parameter or the height parameter are “0” then the GUI will use the default values for the width and height.

The display of the CAEN GUI has three sections:

1. a table of values for all the high-voltage channels of the CAEN power supply,

2. an area for display of text messages, and

3. an array of buttons (on the right side of the display).

Each row of the table of values lists information for a single high-voltage channel of the CAEN power supply. The first column gives the name of the channel. This name is set by the configuration file. Currently, it is not the same name that is set on the power supply. The front panel of the power supply allows a user to set names for the channels. Perhaps future versions of the CAEN Server will set and read even these names, but there has not yet been much reason to do this.

The Vmon and Imon columns show the voltages and currents currently output from the power supply. The V0Set and I0Set columns show the voltages and trip voltages set. The RUp and RDWn columns show the rates set for ramping up and down. The Trip column shows the delay before tripping when the I0Set current is exceeded. The POn column shows a flag that allows (if set to 1) or does not allow (if set to 0) ramping back to a previous state after powering back up after a shutdown. The PDwn column shows a flag that specifies how the power supply is to respond to a “trip” condition (for excess current). If the flag is set (1), then the supply will ramp down. If the flag is unset (0), then the supply will suddenly kill the voltage output.

The Pw column is special. When Pw is set to 1 (ramp up), the power supply ramps up (if not already up) to the voltage set by V0Set. If Pw is
0 (ramp down), the supply ramps down to zero. (There are the V1Set and I1Set CAEN parameters, but the CAEN Server does not use these because CDMS doesn’t use them.) You can change the ramp state of a single channel by right-clicking in the Pw column with the mouse (or other pointing device such as a track ball). Right-clicking in the Pw column and the row for a channel will toggle the Pw state for that channel.

The area for displaying text messages is for status messages. The messages are shown in green if they are neither warnings nor error messages. A “Test Message Monitor” button is provided for seeing the different colors with which messages can be displayed.

The buttons at the right side of the GUI display include the following:

1. A “Start GUI” button for start the GUI interacting with the CAEN Server. After starting this, the GUI automatically and periodically updates its table by calling methods of the CAEN Server.

2. A “Ramp Up All” button to cause all channels (not already ramped up) to ramp up.

3. A “Ramp Down All” button to ramp down all channels.

4. A “Read CAEN” button to update the table display rather than waiting for the GUI to automatically update the display.

5. A “Setup CAEN” button to re-configure the CAEN power supply back to its previous configuration using the same configuration file as before. This assumes that a client has already set up the CAEN Server.

6. The “Test Message Monitor” button already mentioned.

7. An “Exit” button for quitting the GUI.

The “Setup CAEN” button may be unnecessary and be omitted. To re-configure the CAEN Server, the same configuration file as used before is used again. If no configuration file has been used, the re-configuration operation fails.

2.7 The CAEN Remote GUI

The CAEN Remote GUI is a simple GUI that has no controls but does have a table listing some of the values listed by the CAEN GUI. The CAEN Remote GUI is mainly for operation by Rupak’s Run Control GUI. For this reason, it is barely mentioned at all in this manual. There is still a script for
starting the CAEN Remote GUI independently of the Run Control GUI. The Remote GUI can be started by entering “CAENRemoteGUI.script” while in the scripts directory of DAQPackage.

2.8 Methods of the CAEN Server

The CAEN Server provides several methods for clients, and these methods are documented by the combination of the CAENServer.java file and this chapter. All these methods are accessed by clients using Java RMI. The getcaenvalues and setcaenvalues methods are used to access single parameters of the CAEN power supply. The setupcaen method uses the previous configuration file (assuming there was a previous configuration file) used to configure the CAEN Server to re-configure the server and the power supply. The precaensetup method is similar in that it configures the server but it does not affect the power supply. The checkcaen method reads CAEN data into arrays for reading by the readcaen method. The caen_names method or the (equivalent) readNames method returns an array of channel names as already set by the configuration file (or a bunch of question marks for each channel that has not been named yet by the configuration file). The getwarnings method returns indicators of channels that are not ramped up but should be either ramped up or ramping up. These warnings are not necessarily errors unless the channels persist in not becoming ramped up.

The CAEN Server keeps track of voltages and currents on the outputs of the power supplies. The server fills data arrays so that clients can check the recent history of the voltages and currents. Each minute, the server updates the data arrays. When the arrays fill to capacity, the oldest value in each array is dumped when a new value is placed into the array. The order from oldest to newest is maintained in the array (element 0 being the oldest). The readVoltages and readCurrents methods retrieve these brief historical records. This is useful for graphical displays by clients that chart the voltages and currents as functions of time.

The config method was previously used to configure the CAEN Server and power supply, but will be used by Run Control to check the configuration of the supply. It sets the (path) name of the configuration file, and then calls the compareConfig method to compare the current setup of the power supply with the specification given in the configuration file. The configCAEN method specifies the configuration file and uses it to configure the power supply.
2.9 Deprecated Methods

In a previous edition of this manual, I listed the startrun, pause, resume, stop and abort methods as being deprecated methods and suggested they may be removed. However, it appears that Run Control will start making use of them again. Thus, they remain but now do not perform significant operations on the CAEN Server.
Chapter 3

Data Server and Data Server Lite

This chapter discusses the operations of the Data Server and the Data Server Lite.
3.1 Brief Introduction to the CDMS II Event Format

The CDMS II Event Format is the format chosen for storing event data in files which are called “event files.” A single event file can contain several event records. Each event record gives event data for one event. (For example, one trigger or several simultaneous triggers in the DAQ system can be taken as one event.) Each event record contains logical records. There are several types of logical record including those for (digitizer) trace data and time-stamp data (called History Buffer data). Details of the Event Format, including information useful for interpreting the event data, are given in CDMS II documents. This chapter tells how the Data Server and Data Server Lite can be used to obtain information from event files, but excludes detailed discussion of how to interpret the event data.

3.2 Two Servers: the Data Server and the Data Server Lite

Both servers—the Data Server and the Data Server Lite—allow remote users to obtain event data. The data can be obtained even while the Event Builder is in operation. The plan of the Data Server differs from that of the Data Server Lite in two main respects:

1. They differ in how event files are selected.

2. They differ in how the event data is obtained and readied for serving to clients.

The Data Server allows clients to request sessions with event files. Each session request specifies an event file. The Data Server allows only one event-file session to be active at a time. Thus, when one session is active, a request for another session is put into a request queue. When a session is ended, the next session request in the queue is processed and its session becomes active.

When a session becomes active, the event file, that was specified by the session request, is briefly opened and scanned for information on event records and the logical records in each event record. This information is used to build a “table of contents” for the event file. The event file is then closed. The session, however, is still active at this point. During the session, clients (including the client that requested the session) can invoke methods of the Data Server to retrieve event data from the event file. These methods briefly
open the event file, use the table of contents to obtain the requested data, and close the event file again. The event data is served back to the client requesting it.

The client that originally requested the event-file session can close the session at any time. Actually, the client is identified by a number that it passed to the Data Server when it requested the session. It is possible to write a client to fake the identity of another, but the client would need the identification number to do this. When the session is closed, the next session request in the queue is processed unless there are no requests in the queue.

The Data Server uses a text file to tell it which paths contain event files ready for reading. This file can be manually written, but probably will usually be written and continually updated by a program or script during Event Builder runs. A typical file may contain entries that look like the following:

```
data/130320_1525
/data/130321_1526
```

Notice that the lines indicate directories and do not include any slashes at the ends of the lines. (For example,

```
data/130320_1525/
```

would be wrong because this entry ends with a slash.) There is one entry per line of text in the text file. The Data Server contains a method that can list the files in the directories specified by the entries in the text file. Clients use this method to determine which event files are ready for processing.

The Data Server Lite also allows retrieval of event data, but only operates on one special event file rather than letting clients choose between files. This special event file contains only one event record. The CDMS DAQ group decided the Event Builder would periodically write such a special event file to provide a sample of event data. Thus, the Data Server Lite is used to provide access to this file.

Since this special file (with only one event record) is much smaller than a typical event file (which usually contain several hundred event records), the Data Server Lite uses a different method of access to the event data. When either the Data Server Lite starts or the special event file is updated (i.e., re-written or appended), the Data Server Lite loads the entire event record into memory. A “table of contents” is also built for this event record so logical records can be fetched quickly.

Clients use the Data Server Lite in ways similar to the ways clients use the original Data Server, but there is no need to request sessions since the file
is already determined; there is no need to select the event file. Also, there is no need to specify which event record to use because there is only one event record in the special file. Thus, clients specify less when accessing the Data Server Lite than clients do when similarly accessing the Data Server.

3.3 Starting the Data Server

The Data Server is started by a script in the scripts directory of DAQPackage. A command-line parameter indicates the text file containing the list of paths that contain event files that are ready for reading. The entire path of the text file should be given. Thus, entering

```
DataServer.script /homes/me/setup/pathnames.txt
```

(if using a type of C shell) or

```
./DataServer.script /homes/me/setup/pathnames.txt
```

(if using a type of Bash) will launch the Data Server and direct it to use the pathnames.txt file found in the /homes/me/setup directory. To run the server in debug mode (which will cause many debug messages to be output to either standard output or standard error), you would enter

```
DataServer.script /homes/me/setup/pathnames.txt test
```

or

```
./DataServer.script /homes/me/setup/pathnames.txt test
```

3.4 Starting the Data Server Lite

The Data Server Lite is started by a script in the scripts directory of DAQPackage. A command-line parameter indicates the special event file that contains the single event record to be used. The entire path of the event file should be given. Thus, entering

```
LiteDataServer.script /data/one_event.dat
```

(if using a type of C shell) or

```
./LiteDataServer.script /data/one_event.dat
```

(if using a type of Bash) will start the Data Server Lite and indicate that the one_event.dat file in the “data” directory is the special event file containing a single event record.
3.5 GUI's and Other Clients

A simple GUI is provided for use with the Data Server and a similar GUI is provided for use with the Data Server Lite. Not all buttons on the first GUI have their functions implemented because there has not yet been much need for these buttons. In addition to the GUI clients, there are several test clients provided. These test clients allow testing the servers and are examples programmers may want to study in order to write new clients.

The Event Data GUI communicates with the Data Server and can be launched using a script by entering

    EventDataGUI.script

or

    ./EventDataGUI.script

depending on which shell you use. The GUI will not work unless the Data Server is running. The GUI has several tabbed sheets or tabbed panes with the names “Session,” “Detector,” “Veto” and “Others.” The user should start on the “Session” sheet or pane, and click on the “List Files” button. A list of files in the event-file directories will be displayed in the text area under the button. A “Scroll to End” check box is provided. If this box is checked, the text area will scroll to the end automatically each time new text is added. A scroll bar is provided for manual scrolling through the text.

To request an event-file session, the complete path name of any event file seen in the list can be copied (either by hand or using the copy-and-paste method) to the text field (i.e., the blank) for entering the file name. Another text field is to be filled in with a pass-key number. This number is chosen by the user to identify either himself or his client GUI. No client will be permitted to close (end) the requested session without providing this number. Since the user chose the number, this feature prevents other people from closing his session. The pass-key number blank must contain the pass-key number when either the “Request Session” button or the “End Session” button is clicked.

The “Request Session” button requests a session, and the “Get Status” button allows the user to see if his requested session has become active yet. The user may choose to continue pressing the “Get Status” button until he sees the “Session Ready” response in the text area under the button. At the end of a session, the “End Session” button should be clicked to close the session before the “Exit” button is clicked to close the GUI.

The “List Queue” and “List Contents” buttons do not have their functions implemented yet. The “List Queue” button was intended to list the contents
of the session-request queue (omitting the pass-key numbers). The “List Contents” button was intended to list the table of contents for the event file selected. (Future DAQ programmers may want to add the implementations of these buttons.)

The other tabbed sheets (tabbed panes) are self-explanatory. The user should fill in every blank on a sheet before clicking on a button. The blanks are for integers. The functions only operate properly when a requested session is active.

The Lite Event Data GUI is similar to the Event Data GUI, but communicates with the Data Server Lite instead of the Data Server. This Lite GUI can be started by entering

```
LiteEventDataGUI.script
```

or

```
./LiteEventDataGUI.script
```

depending on which shell you use. This GUI works very much like the Event Data GUI, but has no need of requesting sessions since the Data Server Lite always has an active event file: the special event file with only one event record.

Test clients are available for invoking methods of the Data Server and the Data Server Lite. These clients are useful for debugging purposes, and their source code provides examples of properly accessing the server methods. I would suggest these test clients be used by and the source code of these clients be read by people writing clients for accessing the Data Server or Data Server Lite. (People not writing clients may still find these test programs useful for debugging purposes since they allow narrowing the search for a problem down by helping determine whether a problem is with a new client or with the server.)

The scripts for starting these test clients use a command-line parameter to indicate the test to be performed or the server method to be invoked. To run the test client for the Data Server, you enter your request in the following format:

```
DSTestClient.script [integer indicating test]
```

or

```
./DSTestClient.script [integer indicating test]
```
where the “integer indicating test” is a positive integer indicating the test. For example, entering

./DSTestClient.script 1

will perform test number 1. To see what each test does, browse the source code for this client. It is important to know that this test client may still have some tests that use deprecated methods of the Data Server. (See the section on “deprecated methods” near the end of this chapter.) However, for each test that involves deprecated methods, there is another test that uses currently supported methods instead.

The test client for the Data Server Lite is similar. Entering your request in the form

LiteDSTestClient.script [integer indicating test]

or

./LiteDSTestClient.script [integer indicating test]

will invoke a method of the Data Server Lite. For example, entering

./LiteDSTestClient.script 1

will execute test number 1. To see what each test does, browse the source code for this client.

3.6 Methods of the Data Server and Data Server Lite

The Data Server and Data Server Lite provide several methods for clients, and these methods are documented by the comments given in the combination of the DataServer.java file, the DataServerLite.java file and this chapter. All these methods are accessed by clients using Java RMI. Several of the methods of the Data Server Lite are similar to corresponding methods of the Data Server.

It is useful to start this discussion with the methods used to control the event-file sessions of the Data Server. (With the exception of the isSession-Ready method, these methods don’t have equivalents in the Data Server Lite.) The getEventFileList method returns a String array listing the event files that are ready for serving to clients. This method relies on the text
file that indicates the paths the Data Server can expect to contain event files that are ready for processing. The selectEventFile method is used to start an event-file session and the endCurrentSession method ends the current event-file session. (If there are session requests in the Data Server’s request queue when a session is closed, then the next request in the queue is used to start a new session.) The statusCurrentSession, isSessionReady and sessionIDCurrentSession methods can be used by clients to determine if a requested session has become active.

The Data Server has several methods that have the same names and basic functions as corresponding methods of the Data Server Lite, but a method of one server can differ in return types, return values and input parameters from the method by the same name in the other server. Examples of such methods are the GetData, isSessionReady, getTableOfContents, getAdminRec, GetVetoData, GetDetectorData, GetDetectorDataAndTime, GetDetectorDataAllChannel, GetAllDetectorDataAllChannel, getSUFHistory, getSoudanHistoryVetoTimes, getSoudanHistoryVetoMasks, getSoudanHistoryTrigTimes, getSoudanHistoryTrigMasks, getVetoADC, getTrigger and getRawLogicalRecord methods.

### 3.7 Deprecated Methods

The Data Server contains several methods that are deprecated. These methods are no longer maintained, and some of their functionality has even been disabled. These methods will probably be removed from the DataServer.java eventually. (Some of them may be kept in the DataServerImpl.java file because of their ties to other non-deprecated methods.) The deprecated methods are the getServerMode, getEventFile, getEventNumber, changeServerMode, changeEventFile, getTraceData, getLogicalData, getPeakData, and getTrace methods. These methods were useful when the Data Server had modes in which it could automatically broadcast data over CORBA event channels or automatically load trace data from a file. Such functionality has become less useful for CDMS II and, thus, the methods were no longer maintained.
Chapter 4

Joerger Monitor System (JMS)

This chapter discusses the operations of the Joerger Monitor System (JMS) focusing mainly on the server.
4.1 The JMS Server

The Joerger Monitor System (JMS) Server is a program used to set up and arm digitizers, and retrieve data from those digitizers after they are triggered. The name “Joerger” is applied for a historical reason: originally, the JMS Server was written to control the Joerger brand of digitizers. Thus, this server is perhaps improperly named since it now works with either Joerger or Struck brand digitizers. The JMS is to be used only when the Event Builder is not running. This separation of JMS Server and Event Builder is needed because the two software systems use the same hardware—they both use the digitizers.

When the JMS Server is called upon to set up and arm the digitizers, it uses a configuration file. This configuration file is a text file. Which file is used depends on whether Joerger or Struck brand digitizers are to be used. If Joerger brand, then the file is

```
DAQPackage/configuration/joerger/joerger_address .
```

If Struck brand, then the file to be used is

```
DAQPackage/configuration/struck/struck_address .
```

Both of these files have very similar structure. The first line of either is a two-digit hexadecimal integer telling how many digitizer records follow. Each digitizer record consists of three lines of text. Lines of text that follow the last digitizer record are ignored. Thus, a good place to write comments is at the end of a configuration file. The first line of each digitizer record contains hexadecimal digits that indicate the desired (initial) settings of the digitizer. The second line gives the highest four hexadecimal digits of the VME I/O address of the digitizer. (For Struck digitizers, eight digits could be given.)

The third line differs in purpose between the configuration files for the two brands: for Joerger it gives the name (including path) of the device file to be used for reading data, but for Struck it gives some text that is ignored by the JMS Server except for noting it is of the form /dev/joerger0 or /dev/joerger1 to be like the Joerger digitizer records of the Joerger configuration file. (This inclusion of a line that does nothing was intended to make the writing of the source code easier since the JMS server does not need to be concerned about the brand of digitizer when reading the configuration file.)

The configuration file for the Joerger digitizers gives values for the control registers of the digitizers. Thus, the person writing the file should know the functions of these registers. (This scheme for setting registers has disadvantages compared to the plan of the configuration files used for Struck
digitizers. The Struck brand will probably replace the Joerger brand in the future.) In the first line of a Joerger-digitizer record, each pair of hexadecimal digits gives the value to be written to a control register. The values are for six registers and, thus, there are six pairs of digits. The register values are specified in the following order of registers: CSR1, CSR2, CSR3, GDR high, GDR middle and GDR low. The three GDR registers set the number of samples to take after a trigger before halting the data taking. The three CSR registers set various controls. The Joerger 812 manuals specify these settings. An example of a typical joerger address file is the following:

```
02
233404000800
A300
/dev/joerger3
233404000800
A400
/dev/joerger4
```

Comments can follow the last digitizer record.

The configuration file for the Struck digitizers gives values for various functions of the digitizers rather than mere register values. (I think this is the better method.) The first line of each digitizer record gives settings by a string of hexadecimal digits. The first two digits give a rate index. The digitizer divides its (digitization) clock rate by the value of 2 raised to this index as exponent. Thus, if the clock rate is 80 MHz (as it is for the current model of Struck digitizer we use), an index of 01 would cause digitization to occur at a rate of 40 MHz. Rates of 80, 40, 20, 10, 5 and 2.5 MHz can be used. (The manufacturer warns that rates of 10 MHz and below can cause problems.)

The next two hexadecimal digits in the string indicates the number of data points (digitizer readings) to average together. The number of data points is 2 to the power given by the digits. Thus, a value of 00 causes only one point to be used, but that is the equivalent of saying no averaging takes place. A value of 01 would cause each data point reported by the digitizer to actually be the average of two points (that is, two digitization samples). A value of 02 causes groups of four points to be averaged. A value of 03 causes groups of eight to be averaged. This affects the apparent digitization rate.

The next two hexadecimal digits enable the stop delay if they are nonzero, but disable the stop delay if zero. I think a good practice is to set this value to 01 if enabling is desired. This is because people think of a 1 as setting an enable bit and 0 as clearing it. The last four hexadecimal digits of the string specify the stop-delay value. This is a post-trigger value. The delay
(or post-trigger digitization) will only happen if the stop delay is enabled. For a post-trigger of 65536, the last four hexadecimal digits would be ffff (or FFFF). This is 65535, but this number doesn't include the data point taken at the trigger time (before the stop delay). This ffff is the maximum the configuration file specifies for a post-trigger. (At the time this manual was being written, CDMS was experimenting with a model of Struck digitizer that allows larger post-trigger values, but the configuration file only allows up to 65536.)

For either configuration file, the second line of each digitizer record gives the address of the digitizer. The content of third line of each record is only important for Joerger digitizers. An example of a typical struck_address file is the following:

```
02
00001ffff
A300
/dev/joeger0
00001ffff
A400
/dev/joeger0
```

Comments can follow the last digitizer record.

Notice the “/dev/joeger0” lines included for compatibility with the joerger_address file.

### 4.2 Starting the JMS Server

The Joerger Monitor System Server (JMS Server) is started by a script. Entering

```
JoergerMonitor01Server.script 3300 Struck
```

or

```
./JoergerMonitor01Server.script 3300 Struck
```

starts the JMS Server and specifies that the Struck brand of PCI-to-VME interface is to be used with a Struck 3300 or 3301 digitizer. The command-line parameters used are the following:

1. “3300” or “Joerger” to indicate the brand of digitizer,
2. “Struck” or “Bit3” to indicate the brand of PCI-to-VME interface,

3. “log” which is an optional argument which, if present on the command line, tells the JMS Server to generate a log file, and

4. “test” which is an optional argument that directs the JMS Server to run in debug mode (generating many lines of text output to either standard output or standard error).

The command-line parameters “3300” and “Bit3” should never be used together because I didn’t implement support for the combination of a “Bit3” PCI-to-VME interface with a Struck digitizer (since such a combination is not expected to be used in CDMS II). As an example of the use of the command-line parameters, entering

./JoergerMonitor01Server.script Joerger Bit3 log test

will cause the JMS Server to expect a Bit3 PCI-to-VME interface with a Joerger digitizer, and direct the server to generate a log file and run in debug mode.

4.3 GUI’s and Other Clients

I did write some GUI programs for the JMS Server, but so frequently was I asked to change methods of the server that the methods my GUI programs used became deprecated. That is, I changed the server so frequently that my graphical clients no longer worked well. The majority of the changes were made to accommodate Rupak’s GUI which now replaces all of my graphical client programs. However, I still wrote several simple test clients. My test clients are useful when the interaction between Rupak’s GUI and the JMS server presents trouble. In that case, my test clients can help determine whether the problem is at the GUI or the server. The source code for the test clients also provides examples of the use of the server’s methods, and should be read by programmers who desire to write clients for use with the JMS.

The test clients are started by scripts with names like

JoergerMonitor01TestClient2.script

and

JoergerMonitor01TestClient3.script
but not the script

   JoergerMonitor01TestClient.script

because that one uses one of my old deprecated methods. Perhaps that one
will eventually be eliminated. (It was probably kept for so long because it
uses an old method that was used by my old GUI clients.) In order to see
what server methods the test clients invoke, you can read the source code for
them.

4.4 Methods of the JMS Server

The JMS Server provides several methods which are documented by the com-
bination of the comments in the JoergerMonitor01.java source-code file and
this chapter. There is some old CORBA source code used in the Java and
C++ portions of the source code, but CORBA features are all deprecated
now. In reading the source code, you may notice that the terms “digitizer
number” and “detector number” are used interchangeably. This is because
people in CDMS have made a one-to-one correspondence between each dig-
itizer and each detector except in the case of the CDMS II Veto System in
which the correspondence is between each detector and each channel. Also,
for historical reasons, several of the methods contain the word “joerger” in
them because the JMS Server was originally specialized to Joerger digitizers.
The old names persist even though the JMS is now used for two digitizer
brands.

The digitizers are configured by the SetupJoerger method, and the con-
figuration can be checked using the GetJoergerSetup method. Reading the
digitizers should be done using the GetData method rather than the jo-
ergerRead or joergerReadAll methods because those later methods are now
used mostly by internal operations of the JMS Server. The joergerGetRate
method returns the digitization rate in kHz, and the joergerGetRateIndex
method returns an index number that indicates the rate. The interpretation
of the index returned by this last method depends on the model of digitizer;
thus, I prefer to use the joergerGetRate method since it returns a rate in
useful units. The pauseServer and resumeServer methods allow a client to
instruct the server to pause its operations or resume them. This feature was
requested by people who didn’t want to shut down the server each time other
software was going to be used to access the digitizers.
4.5 Deprecated Methods

The joegerSetup method is deprecated. It was used by my old GUI clients to configure the server and digitizers, but has been replaced by the SetupJoerger method. It may be eliminated from the server software in the future.