Dataguzzler: A Platform for High Performance Laboratory Data Acquisition

Stephen D. Holland

Version 1.0.0-rc4-devel — September 14, 2007

Dataguzzler was written by Stephen D. Holland at the Iowa State University Center for Nondestructive Evaluation.

See the Dataguzzler web site at http://ahab.cnde.iastate.edu/~sdh4/dg_web/ for more information about Dataguzzler.

This material is based upon work supported by the Air Force Research Laboratory under Contract #FA8650-04-C-5228 at Iowa State University's Center for NDE.

This material is based upon work supported by the Federal Aviation Administration under Contract #DTFA03-98-D-00008, Delivery Order #0037 and performed at Iowa State University's Center for NDE as part of the Engine Titanium Consortium Phase III Thermal Acoustic Studies program.

Thanks to Ricky Reusser for designing the original logo and Kira Scott for drawing the new logo. Thanks to David Holland for suggestions on interfacing with the operating system.

Please take note of the warranty information as stated in the GPL v2.0 license:

Contents

Introduction

We have developed a new data acquisition software architecture with thermosonic nondestructive testing as its first application. The architecture was designed for for optimal flexibility, robustness, and performance given the varied requirements of thermosonic measurement. It consists of a communications protocol, an event driven dispatch loop, and a set of libraries and modules for acquiring, processing, and communicating data.

Chapter 1

Building and Installing Dataguzzler

1.1 Software Requirements

(development components of all libraries are required)

- Linux 2.6 series or higher kernel.
- A LaTeX distribution with PDF support (e.g. tetex) (for building the documentation).
- libcap 1.10 or better (OS component)
- OpenGL libraries (OS component)
- FreeGLUT (OS component or freeglut.sourceforge.net)

1.2 Recommended Software

(development components of all libraries are required)

- FFTW 3.1 or higher, installed with both single- and double-precision support. (OS component or www.fftw.org)
- Octave 2.9.4 or higher (OS component or www.octave.org).
- Python 2.4 or higher (OS component or www.python.org)

1.3 Compatible measurement hardware

(these must be configured in dataguzzler.conf)

- Aglient 33120A and 33220A arbitrary waveform generators, over GPIB, RS-232, or TCP/IP as supported by the instrument.
- Measurement Computing PCI-DAS4020/12 waveform capture card with Warren Jasper's Linux driver from ftp://lx10.tx.ncsu.edu/pub/Linux/drivers
- EDT PCI DV CLINK image capture card (grayscale only) with EDTpdv drivers (www.edt.com)
- HP 34401 Multimeter with a thermistor attached, used as a thermometer over GPIB or RS-232.
- PolyTec OFV-5000 Vibrometer Controller, over RS-232.
- Various GPIB cards supported by the linux-gpib project (linux-gpib.sourceforge.net) through the multiio.so I/O library (included in dataguzzler).
- Many SCPI compliant laboratory instruments with the genericscpi module.
- Tabor WW5061 Arbitrary Waveform Generator.
- Parker/Compumotor ACR-9000 Motion Controller.

1.4 Compile-time configuration

Inspect defs.mk for any parameters that might need adjustment (most parameters are set automatically). The most commonly adjusted parameter is `PREFIX`, the install prefix (which defaults to `/usr/local`. You may also want to adjust `CFLAGS` to disable debugging and turn on optimization or vice-versa.

1.5 Building dataguzzler

Use the 'make' command to compile dataguzzler. After the build is completed, it will respond with the build status, e.g.:

```
Build status
------------
Developer's reference guide doc/ref.pdf: built
dataguzzler server server/dataguzzler: built
FFTW support server/libraries/fftwlink.so: built
```

```
DAS4020 support server/modules/das4020capture.so: built
MultiIO library server/libraries/multiio.so: built
MultiIO library GPIB support: Enabled
EDT Camera Link support server/modules/edtcapture.so: built
dataguzzler library lib/libdataguzzler.a: built
oscilloscope display scope/dg_scope: built
Python support server/libraries/dg_python.so: built
```

Look for any components which should have been built but were not. Scroll back and look for error messages to help troubleshoot.

The following make commands are supported:

- `make all` (default)
- `make install` Install dataguzzler into the PREFIX specified in defs.mk
- `make clean` Remove object, emacs backup, and core dump files.
- `make distclean` Remove object, emacs backup, and core dump files as well as binaries.
- `make depend` Recalculate dependencies

1.6 Installation

Type `make install` to install dataguzzler into the PREFIX specified in defs.mk. This will create a directory PREFIX/dataguzzler-<**version**> containing the dataguzzler binaries. It will also put symbolic links to the dataguzzler programs in PREFIX/bin and a symbolic link PREFIX/dataguzzler to PREFIX/dataguzzler-<**version**>. It is suggested that you copy a dataguzzler configuration file into PREFIX/dataguzzler-<**version**>/dataguzzler.conf. This configuration file probably needs to be selected or edited to use only the hardware and software you have installed. See section 2.2 for more information on configuring Dataguzzler.

1.6.1 Security and permissions

Dataguzzler by default installs as a setuid-root executable. While this technically is a security risk, such risk is minimized by the fact that Dataguzzler immediately drops root permissions on startup (top of main() in main.c). When run as a setuid-root program on Linux, Dataguzzler requests the following special permissions and capabilities from the operating system before discarding its privledges:

- RLIMIT_MEMLOCK set to RLIM_INFINITY (see setrlimit(2)).

- CAP_SYS_NICE to allow the use of POSIX realtime priorities. (see capabilities(7)).
- CAP_SYS_RAWIO to allow direct hardware or direct USB bus access if necessary (see capabilities(7)). Note that this capability is discarded for security reasons if dataguzzler is given an explicit path to a configuration file.

Whether or not dataguzzler is run setuid-root, permissions should be properly set for the device nodes (in /dev) of hardware devices so that they can be used. To make device node permissions persistent, configure udev in /etc/udev.

Chapter 2

System Construction (Software)

2.1 Overall architecture

The primary software component of the data acquisition architecture is the **dataguzzler** program. The dataguzzler program accepts commands from the keyboard or TCP/IP connections and passes these commands on to loadable modules. The modules process commands, return results, and simultaneously interface with the data acquisistion hardware. The modules can make use of loadable libraries as well as other modules. In addition, external programs can communicate with the dataguzzler program by TCP/IP. For example, an oscillosope display, **scope**, is commonly used to display live waveforms. The **dg_grab** and **dg_upload** programs are used to download waveforms from and upload waveforms to the dataguzzler program.

2.2 Configuration

The dataguzzler program is configured at run time according to a configuration file. The configuration file specifies which libraries and modules to load and allows passing configuration parameters to those modules.

2.2.1 Specifying the configuration file

The configuration file name is specified on the Dataguzzler command line. If no configuration file name is specified, the default is "dataguzzler.conf". If the configuration file name contains slashes, it is interpreted as a path to a specific file (note that in this case the CAP_SYS_RAWIO capabilitity is relinquished. See sect. 1.6.1.). Otherwise the configuration file is found by searching the following directories, in order:

1. `/etc`
2. <**prefix**>`/etc` (where <**prefix**> is the installation prefix, e.g. `/usr/local`)
3. <**bindir**>`/..` (where <**bindir**> is the directory containing the dataguzzler binary)
4. <**bindir**> (as above)

2.2.2 Configuration file format

The syntax of the configuration file is best described by example:

```
library "libraries/wfmstore.so" { }
library "libraries/metadata.so" { }

module prototype("modules/module_prototype.so") {
        cardname="/dev/mydevice"
}

module wfm("modules/wfmio.so") { }

module auth("modules/auth.so") {
        AuthCode(127.0.0.1) = "xyzzy"
}
main {
  TCPPort=1649
  maxtimeout=100 ms
}
```

This example configuration file first loads the "wfmstore.so" library with no parameters, then loads the "module_prototype.so" dummy module with the parameter "cardname" set to "/dev/mydevice". It then loads the "wfmio.so" waveform transfer module with no parameters and the "auth.so" remote authentcation module with one authentication code defined. The order of entries in the configuration file determines the order in which the modules and libraries are loaded. See the documentation for specific modules and libraries for specific prerequisites. In general, libraries will not be dependent on modules, so libraries should appear at the top of the configuration file and should be loaded first. Parameters for the dataguzzler kernel should be in a block preceded by the keyword "main". Documentation for the module parameters and dataguzzler kernel parameters is found in the relevant section of the manual. If the dataguzzler binary is setuid root, then dataguzzler will relinquish root privledges after reading the configuration file and initializing libraries and modules.

2.3 Documentation

Obviously architecture-oriented documentation such as this manual is useful for the data acquistions system designer, not for the end-users of the data acquisition system. This manual is typeset in LaTeX and written in such away that the data acquisition system designer can write a manual that extracts relevant information from this manual and presents it in a form suitable for end-users.

Chapter 3

The dataguzzler kernel

3.1 Kernel functionality

The kernel consists of program initialization, including config file processing, the main event loop, command i/o and processing, and various required libraries. The libraries in the kernel are:

- **linklist**: Used internally and by various modules for managing linked lists.
- **library**: Used for loading external libraries.
- **mod**: Used for loading external modules.
- **multipoll**: Improvement over the POSIX poll() system call, used internally.
- **stringstack**: A library for manipulating stacks of strings. Used internally by the configfile parser and by various module command parsers.
- **util**: Miscellaneous convenience routines.
- **rpc**: Remote procedure call library for calling modules from within dataguzzler.

3.2 Configuration parameters

The kernel is configured by specifying parameters within a “main” section in the configuration file:

Parameter	Type	Value
SetQueryPrefix	quoted string	A prefix to be placed in front of the response to all SET queries
SetQueryPostfix	quoted string	A postfix to be placed at the end of the response to all SET queries
TCPPort	integer	The TCP/IP port number for control input
maxtimeout	float, opt. units	The maximum timeout length for the main event loop. Milliseconds are assumed if units are not specified.
rt_priority	integer	POSIX SCHED_RR realtime priority. Default is non-realtime SCHED_OTHER priority. Note that this should be set in a main section at the top of the configuration file so that threads started by the various modules can inherit the realtime priority if they so desire (some do, some don't).
mlockall	boolean	if TRUE, prevent dataguzzler memory from being swapped out to disk. Default FALSE.
initcommand	string	Execute the specified string during initialization, before accepting external or keyboard commands. `initcommand` can be specified multiple times and the strings are concatenated with semicolons.
setqueryfilter	string	specifies a command which will be filtered from the results of SET? queries.

3.3 Kernel commands

(kernel commands begin on next page)

SET

Syntax

```
SET?
```

Description

Query the settings of the dataguzzler server

Parameters

None

Notes

- This command returns most of the internal settings of the data acquisition server, formatted such that they can be saved and resubmitted to restore the internal state.
- This command returns multiple responses to a single query.

See also

3.4 Kernel API and internals

3.4.1 The connection database

Each connection is defined by a struct Conn:

```
struct Conn {
  struct Node Node;
  int ReadSock,WriteSock;
  int CloseReadSock,CloseWriteSock; /* set if we need to close() ReadSock/WriteSock when we close the conne
  struct sockaddr *Addr;

  /* Buffers for data pre-read from ReadSock */
  struct ConnBuf *InStream;

  /* Data waiting to go out to OutSock */
  struct ConnBuf *OutStream;

  /* data already extracted from InStream */
  struct ConnBuf *CurCommand;
  int CurCommandContinues; /* CurCommand ended with a semicolon, so the next command is part of the same se


  /* current result under construction */
  struct ConnBuf *CurResult;


  int Auth; /* non-zero if authenticated */
  int Closing; /* non-zero if closing in progress */
  int LastErrorCode;
  struct pollfd WakeupFd; /* poll() for this fd and event mask and retry processing of CurCommand.
     WakeupFd.revent will be set with the received mask. */
  struct timespec WakeupTime; /* (0,0) indicates no timeout req'd */
  struct List CloseNotifyList;
};
```

This structure defines the data stored for each control connection, including stdin/stdout and any TCP/IP control connections. Pending data is stored in struct ConnBuf:

```
struct ConnBuf {
  char *Data;
  int Size; /* allocated size. 1 extra byte beyond this always allocated for a 0 terminator */
  int Len; /* amount actually used */
```

Once Conn.InStream contains a complete command to be processed, it is transfered to Conn.CurCommand for processing. Then the target module is determined and the appropriate handling function is called. If the function cannot return immediately, it returns -1 (PRS_RETRY) and will be called again on the next pass through the main loop. The handling function writes its result to Conn.CurResult using the ResPrintf() function and when complete Conn.CurResult is flushed to the output socket.

The WakeupFd specifies a file descriptor to poll() and WakeupTime specifies a time to wakeup to ensure that the handling function is called again at the proper time.

The CloseNotifyList contains entries of struct ConnCloseNotify that will be used to notify modules if this connection closes. This is used, for example, to ensure that locked waveforms are automatically unlocked if the connection locking those waveforms dies.

3.4.2 The module database

The kernel maintains a master list of modules each defined by a struct Module:

```
struct Module {
  struct Node Node;
  int initializing;
  int (*ProcessCommand)(struct Module *Mod, struct Conn *Conn,char *CmdBuf,int CmdBufLen);
  char *Name;
  struct timespec WakeupTime; /* this bounds waiting in the main event loop. zero indicates ignore */
  void (*BackgroundJob)(struct Module *Mod); /* if non-NULL, executed every main loop, immediately after se
  struct pollfd WakeupEvent; /* should poll() for this event */
  void (*WakeupJob)(struct Module *Mod,int evmask); /* Executed in response to WakeupEvent */

};
```

It should be mentioned that most modules actually use a longer structure which begins with a struct Module. Therefore the module pointer can be equivalently treated either as a struct Module or as the module-specific structure.

- initializing is a flag that indictates that the module's initialization routine is still being called.
- ProcessCommand() is the routine called when the kernel has determined that the command in Conn.CurCommand should be processed by this module. Return values are defined in mod.h PRS_. . . .
- WakeupTime is used to bound the poll() timeout in the main loop.

- BackgroundJob() is called every pass through the main loop.
- WakeupEvent is a poll() criteron for the main loop, and
- WakeupJob() is called when a WakeupEvent has occured.

3.4.3 Multithreading

The kernel runs a single event-driven thread. Nevertheless, it is expected that modules or libraries will need to spawn off threads in order to complete tasks asynchronously with the main kernel thread. Thread-safety of library routines will be discussed in the documentation of that library; In general, kernel routines are NOT thread safe, however there are some exceptions. The following routines are thread-safe.

- util.c
  - FindEOL();
  - PTimeDiff();
  - my_infnan();
- main.c
  - setnonblocking()
- linklist.c,stringstack.c
  - All routines provided the list/stack involved is somehow locked by the thread.

The best way to notify the kernel that a thread has completed its duties is to write a character to a pipe that the kernel has been told to wait on.

3.4.4 The rpc library

The RPC library provides routines that allow for "remote procedure calls" between modules. Both synchronous and asynchronous calls are supported. The simplest routine is rpc_synchronous():

rpc_synchronous

`int rpc_synchronous(char **res, char *fmt, ...)`
This routine makes an immediate RPC call as specified with the printf() style format string `fmt`. The function returns 1 for success, -1 for error, or 0 if the call did not complete because it would have had to wait (waiting is not permitted in routines called throught rpc_synchronous()). rpc_synchronous returns a pointer to the result string in

`(*res)`. The caller needs to free(*res) when the caller is done with it. Note that there is also a routine: `int rpc_synchronous(char **res, unsigned char *str);` that takes a string argument instead of the printf() style format string.

rpc_asynchronous

```
void rpc_asynchronous(struct Module *Mod,
                      struct Conn *Conn,
                      void *Param,
                      void (*Continuation)(int retval,
                                           unsigned char *res,
                                           struct Module *Mod,
                                           struct Conn *Conn,
                                           void *Param),
                      void (*ConnDestructor)(struct Module *Mod,struct Conn *Conn,void *Param),
                      char *fmt,...)
```

rpc_asynchronous() makes asynchronous RPC calls to modules. The command string is defined by the printf-style "`fmt`" parameter and subsequent optional parameters. When rpc_asynchronous is called, it will attempt to complete the call immediately, and may call `Continuation()` before returning. In general the provided `Continuation()` routine will be called when the asynchronous command is complete. If neither `Conn` nor `ConnDestructor` are NULL and the connection Conn is dropped before the asynchronous call completes, then the routine `ConnDestructor()` will be called as the connection Conn is destroyed (and `Continuation()` will never be called).

Parameters to rpc_asynchronous

- `Conn`: Connection this asynchronous call is associated with (NULL OK).
- `Param`: Arbitrary pointer to be passed to `Continuation()` and `ConnDestructor()`.
- `Continuation`: Function to be called when asynchronous call is complete. Note that the res parameter must be copied and will be free()'d after the continuation function returns.
- `ConnDestructor`: Function to be called if Conn is destroyed before asynchronous call is complete.
- `fmt, ...`: Printf style format string and optional parameters.

Note that there is a parallel call, rpc_asynchronous_str() that does not use the printf-style formatting.

Chapter 4

AnaGram, M4, and Syntax Files

4.1 AnaGram

The kernel needs to be able to parse (interpret) its configuration file. Modules need to be able to parse textual commands issued to them. A parser generator known as AnaGram (formerly sold by Parsifal Software) generates the parsers used by modules to parse commands and used by the kernel to parse its configuration file. This removes the complexity of command and parameter parsing from the human-written code to computer generated code and allows the modules to be written using what is known as "syntax-directed programming". That is, code is associated with "productions". If the production is found in the input, then the corresponding code is executed. For example:

```
(unsigned long)unsigned decinteger
  -> DIGIT:d =(int)(d-'0');
  -> unsigned decinteger:i, DIGIT:d =(i*10)+(int)(d-'0');
```

These three lines of code define the syntactic element "unsigned decinteger" (short for unsigned decimal integer). They state that an unsigned decinteger (which has the C type of unsigned long) can consist of either a DIGIT or an unsigned decinteger followed by a digit. As you can see, the definition is recursive, in that it can contain any number of DIGITs. The productions are indicated by the arrow symbol "->". When the production has been identified in the input, the "reduction procedure" (the C code following the equals sign) is executed. Thus if the characters "12345" are encountered, the first character is processed by the first production and the remaining characters by the second production, and the result is that the numerical value 12345 is determined. It should be warned that it is very easy to define an ambiguous syntax in this way. Readers are referred to the AnaGram manual and the Parsifal Software web site, http://www.parsifalsoft.com for more information.

4.2 M4

While AnaGram is very effective at parsing, in a system such as this where there will be many independent parsers, there will inevitably be a lot of duplication of primitive syntax elements (such as integers, floating point numbers, quoted strings, etc.). To keep the amount of manually modified code under control the GNU M4 macro preprocessor, http://www.gnu.org/software/m4/, is used to process include directives and other macros to minimize the amount of code required in each module.

A variey of file name extensions are used by the M4-preprocessing and AnaGram. To avoid confustion these are enumerated here.

1. ".synm4": Syntax file to be processed by M4. This is the primary specification of the parser, and is usually the correct file to manually modify.
2. ".syn": Syntax file. In general, these files have already been processed by M4 and therefore should not be manually edited. These are the input to AnaGram. Do not edit a .syn file if there is a corresponding .synm4 file.
3. ".synhm4": Syntax file header to be included by M4. These files contain generic syntax used by multiple parsers. They are incorporated by the use of the M4 "include" directive.
4. ".c" and ".h": These may be manually written C code OR AnaGram generate parsers. Do not edit a .c or .h file if there is a corresponding .syn or .synm4 file.

To avoid problems when writing .synm4 and .synhm4 files, the programmer must have an understanding of C, AnaGram, and M4. For example, the C/AnaGram comment is /* This is a comment */ or // This is a comment to the end of the line
The M4 comment is # This is a comment to the end of a line.
To completely comment out a line in this C/AnaGram/M4 hybrid language the string //# should be used to introduce a comment to the end-of-line. This ensures that neigher M4 nor C/AnaGram will process anything in the comment.

Certain commands are treated specially by M4 (unless they are preceded by a # comment initiator). These are specified on the m4 man page. The most common are define, include, and ifdef. Be warned that if any of these appear unintentionally in a .synm4 or .synhm4 file the results will be unexpected (and can be troubleshooted by looking at the generated .syn file). A standard include file "stddef.synhm4" is provided that defines some standard character sets but also changes the default M4 quote characters from ‘’ to something less likely to appear unintentionally, [[]].

4.3 Module support code

The module source code directory include three files designed to be used in every module: module.c, module.h, and module.synhm4. Together these three files provide the baseline code for an AnaGram/M4 module. See module_prototype.synm4 for an example of their use.

Chapter 5

System Control

5.1 Manual control

Dataguzzler is command driven. Commands may be issued either on the terminal from which dataguzzler was run or over a TCP/IP connection. A TCP/IP connection can be established with the telnet command, e.g.:

```
linux% telnet localhost 1649
Trying 127.0.0.1...
Connected to marius (127.0.0.1).
Escape character is '^]'.
auth xyzzy
200 00000009 AUTH_OK
wcapt:freq?
200 00000019 WCAPT:FREQ 10 MHz
wcapt:freq 1 MHz
200 00000018 WCAPT:FREQ 1 MHz
quit
Connection closed by foreign host.
linux%
```

Let's examine the above transcript and learn how to issue commands. The text in **boldface** indicates what was typed by the user, while the `typewriter` text was generated by the computer.

Upon connecting to port 1649 (the dataguzzler port), the first command issued was **auth xyzzy**. This authenticates the user to the dataguzzler server and must be done before any commands can be issued (authentication is not necessary when typing commands on the console). Information on the **AUTH** command can be found in chapter 7 on page 55.

Dataguzzler then responds inititally with exactly 17 characters. The first 3 characters are the return code in decimal. 200 indicates success, 500 or higher indicates an error. The return code is followed by a space and 12 more characters which indicate the length of the response (not including the 17 character header), again in decimal. This is followed by a space, then the remainder of the response, with the length as specified. The response ends in a carriage return and linefeed. These characters are included in the length count. In the case of the AUTH command, the specified length was 9 characters. The actual response was "AUTH_OK", 7 characters, plus the carriage return and linefeed, for a total of 9.

The AUTH_OK response indicates that authentication has been successfully completed and that other commands may be issued. The user then queries the waveform capture capture sample frequency (WCAPT:FREQ, pg. 107), adjusts the capture frequency form 10 MHz to 1 Mhz, and exits.

The complete list of available commands can be found in chapter 7. It is important to realize that the oscilloscope display program has no control capabilities whatsoever. Its sole purpose is to display the waveforms in the dataguzzler memory. All acquisition parameter changes must be performed separately.

Normally each command transmitted is terminated by a carriage return, linefeed, or combination thereof. It is possible to transmit a number of commands as a single atomic unit by separating them with semicolons instead. The composite command must still have a linebreak at the end. The reply generated by the server will consist of a single block containing the corresponding responses similarly separated by semicolons and with a carriage return / linefeed pair at the end. In this situation all the specified commands will be executed together as an atomic unit unless waiting is required by one of the commands. Commands that wait are documented as such in chapter 7. If an error occurs while executing a command, the command's reply may be replaced by an error message. The substring "ERROR" should occur in this message before the first space.

5.2 Oscilloscope display

The oscilloscope display can be started with the `dg_scope` command. The oscilloscope display assumes dataguzzler is running on host `localhost` port `1649` with authentication code `xyzzy`. If dataguzzler is running elsewhere, these parameters can be provided on the command line:

`scope` **<hostname> <port> <authcode>**

Additional parameters defined by X and GLUT can also be provided. See http://www.opengl.org/developers/documentation/glut/spec3/node10.html for a complete list

The oscilloscope display provides live viewing of the waveforms in the dataguzzler memory, and real-time manipulation of that view. *The oscilloscope display is a tool only for viewing the dataguzzler waveforms and settings. The oscilloscope display cannot be used to change the dataguzzler settings.*

The oscilloscope display is designed for keyboard or combined mouse and keyboard interaction. The keyboard commands are as follows:

Enter	Disable or enable display selected waveform
Tab	Select next waveform
Cursor left	Increase Secs/Div, Hz/Div, or pixels/pixel (zoom out horizontally)
Cursor right	Decrease Secs/Div, Hz/Div, or pixels/pixel (zoom in horizontally)
Cursor down	Increase Volts/Div. (zoom out vertically) or decrease image contrast.
Cursor up	Decrease Volts/Div (zoom in vertically) or increase image contrast.
Home	Increase t_0 by one division (look later in waveform)
End	Decrease t_0 by one division (look earlier in waveform)
PgUp	Increase display offset of waveform by one vertical division
PgDn	Decrease display offset of waveform by one vertical division
Insert	Increase image brightness
Delete	Decrease image brightness
','	Select previous frame of a multi-frame image.
'.'	Select next frame of a multi-frame image.
'c'	Cycle between colormaps for image displays
'o'	Set the position or offset to the median value of the selected waveform or frame
'z'	Zero the position and offset of the selected waveform or frame

The oscilloscope window consists of the active oscilloscope area, surrounded by informational displays. The left edge of the window lists the names of the waveforms available from the server. The currently selected waveform name is highlighted. Clicking on a waveform name will select that waveform. If an attenuating probe was used and configured in the server that information (e.g. 10x) will be recorded after the waveform name. Next to the waveform name is a clickable box which controls whether that waveform is currently displayed.

The top line of the window provides the acquisition parameters for the currently selected waveform. t_0 is the time of the first sample. Fs is the sample frequency. n is the total number of samples acquired, and rev is the waveform revision. For AVG and AVGONCE waveforms, the status of the averaging is also indicated.

The bottom of the display lists the current display parameters. t_0 is the time corresponding to the vertical line in the center of the waveform display. f_0 is is the frequency corresponding to the vertical line in the center of the waveform display (for frequency domain waveforms). globalrev is the current overall waveform revision count. Position is the vertical offset of the selected waveform.

The waveform display uses several techniques to reduce aliasing artifacts and improve signal comprehension. If there are more than two waveform points corresponding to a particular vertical line of pixels on the display, then a vertical line is drawn between the extreme pixels. For less than two points, the samples themselves are drawn. If there is no waveform point corresponding to a particular point on the display, an interpolated value is drawn in a lighter color. The user should be warned that the interpolation algorithm is approximate only. Proper interpolation can be performed in post-processing using algorithms such as Matlab INTFILT.

A standalone version of the oscilloscope display can be run with the command `dg_scope_sa`. This version does not require dataguzzler to be running, but instead displays one or more waveform file (`.dgz`) or snapshot file (`.dgs`) specified on the command line.

5.3 Utilities

5.3.1 dg_save_settings

`dg_save_settings` is an external program that downloads the current state of dataguzzler and writes it to a file. The general usage is:
`dg_save_settings` <**settings_file.set**>
Additionally -h <**host_name**> and -a <**authentication_code**> parameters may be provided.
dg_save_settings stores the results of the WFM:WFMS? and SET? commands to the output file.

5.3.2 dg_restore_settings

`dg_restore_settings` is an external program that uploads a stored settings file to dataguzzler. The general usage is:
`dg_restore_settings` <**settings_file.set**>
Additionally -h <**host_name**> and -a <**authentication_code**> parameters may be provided.
dg_restore_settings assumes the file consists of two lines of commands. It passes both lines to dataguzzler to be executed.

5.3.3 dg_grab

`dg_grab` is an external program that downloads waveforms from dataguzzler and writes them to dataguzzler format binary files. See Appendix A for more information on the file format. The general usage is:
`dg_grab` <**waveform_name**> <**file_name**> <**waveform_name2**> <**file_name2**> ...
Additionally -h <**host_name**> and -a <**authentication_code**> parameters may be provided.
As many (waveform_name, file_name) pairs as are desired can be specified.

5.3.4 dg_grab_txt

`dg_grab_txt` is an external program that downloads waveforms from dataguzzler and writes them to ASCII text files. The general usage is:
`dg_grab_txt` <**waveform_name**> <**file_name**> <**waveform_name2**> <**file_name2**> ...
Additionally -h <**host_name**> and -a <**authentication_code**> parameters may be provided.
As many (waveform_name, file_name) pairs as are desired can be specified.

5.3.5 dg_cmd

`dg_cmd` issues a single command to dataguzzler and prints the response to stdout. The general usage is:
`dg_cmd` <**command**>
Additionally -h <**host_name**> and -a <**authentication_code**> parameters may be provided.

5.3.6 dg_upload

`dg_upload` uploads a binary dataguzzler format file to dataguzzler. See Appendix A for more information on the file format. The general usage is:
`dg_upload` <**waveform_name**> <**file_name**> <**waveform_name2**> <**file_name2**> ...
Additionally -h <**host_name**> and -a <**authentication_code**> parameters may be provided.
As many (waveform_name, file_name) pairs as are desired can be specified.

5.3.7 dg_upload_txt

`dg_upload_txt` uploads an ASCII text waveform to dataguzzler. The general usage is:
`dg_upload_txt` <**waveform_name**> <**file_name**> <**waveform_name2**> <**file_name2**> ...
Additionally -h <**host_name**> and -a <**authentication_code**> parameters may be provided.
As many (waveform_name, file_name) pairs as are desired can be specified.

5.3.8 dg_upload_tiff

`dg_upload_tiff` uploads a TIFF image to dataguzzler as a grayscale. The general usage is:
`dg_upload_tiff` <**waveform_name**> <**file_name**> <**waveform_name2**> <**file_name2**> ...
Additionally -h <**host_name**> and -a <**authentication_code**> parameters may be provided.
As many (waveform_name, file_name) pairs as are desired can be specified. Note: `dg_upload_tiff` cannot read grayscale images with more than 8 bits per pixel.

5.3.9 dg_snapshot

`dg_snapshot` saves a consistent snapshot of all waveforms (including dynamic waveforms) to the name (`.dgs` file) specified on the command line.

5.3.10 dg_load_snapshot

`dg_load_snapshot` loads the waveforms in the `.dgs` file specified on the command line into dataguzzler. Please note the potential for conflicts with pre-existing channels. Depending on the nature of the pre-existing channel, the conflict may be resolved one way or the other. In general, only nonexistant channels or preexisting channels owned by the WFMIO module can be overwritten by `dg_load_snapshot`.

5.4 API Reference

A library of C utility functions is installed in /usr/local/dataguzzler/lib, with include files in /usr/local/include. A similar library of functions for Matlab or GNU Octave is installed in /usr/local/dataguzzler/matlab. These libraries are used for convenient access to the dataguzzler server and provide routines for remote access and waveform upload/download. Both APIs lack formal documentation at this time. Nevertheless, as the APIs are simple and straightforward it should not be difficult to learn them anyway. All functions are prototyped in the include files, and the source code for the utilities described above make excellent examples for the C library. For the Matlab/Octave routines see the files dgf_testread.m and dgf_testwrite.m for simple examples, and proccalib.m for a more complicated example.

Chapter 6

Library Reference

Introduction

Libraries are optional collections of routines that are loaded if specified in the configuration file. Unlike a module, libraries are permitted static data (global variables). A library can be loaded only once – a second reference will not load another copy of the library. In addition any global symbols (functions, global variables, etc.) defined in the library are accessible to all other libraries and all modules.

Each library defines a function LibInit:

```
void LibInit(char *LibParams,int LibParamsLen);
```

This routine is called once when the library is loaded, and should be used by the library to perform any needed global initializations.

6.1 wfmstore.so

The waveform store, wfmstore.so, is used to maintain a database of waveforms and waveform data. Of particular note is the distinction between waveforms and channels.

6.1.1 Channels

The waveform store maintains a list of channels, each defined by a struct Channel:

```
struct Channel {
  struct Node Node; /* on ChannelList */
  char *ChannelName; /* Channel name (separate alloc) */
  unsigned long long latestrevision; /* increment on every update (including delete) */
  int Deleted; /* if non-zero, this waveform is deleted (but the structure is only
  freed or removed from the ChannelList when it is recreated). If a waveform of the same name
  is ever recreated, then store latestrevision, destroy this channel and create a new one with latestrevisi
  struct List WfmList; /* list of waveforms in this channel */
  struct List NotifyList; /* List of notifications (struct WfmNotify) to make
     if new waveform is created */
  int Volatile; /* does not need to be saved (OBSOLETE -- NOT USED) */
  char *Creator; /* Creator module name */
  void (*Destructor)(struct Channel *Chan); /* call this when channel deleted if not NULL */
  void *ModSpecific; /* data owned  by creator -- alternative to extending the structure */
  /* may have creator-specific data beyond this point */
};
```

Each channel has a name (ChannelName) and a latestrevision. Latestrevision (except for a deleted channel) must always indicate the revision number of an entry in the waveform list. Channels cannot be entirely erased – but the channel will not appear when listed by the user if the Deleted flag is set. Channel.Creator is the name of the module that created (and owns) the channel. Channel.Destructor() will be called when the channel is deleted. Channel.NotifyList is a list of notifications to perform when a new waveform on this channel is created.

Use the CreateChannel() call to create a new (or reuse an old) channel. This will return the new channel or NULL if a channel of the requested name already exists. A call to CreateChannel() MUST be followed immediately by a call to CreateWfm() (see below)

6.1.2 Waveforms

The waveform store consists of each channel's lists of waveforms, each defined by a struct Wfm:

```
struct Wfm {
  struct Node Node; /* on WfmList */
  char *ChannelName; /* separate alloc */
  struct List MetaData;
  int ReadyFlag; /* if zero, this is just a placeholder and no actual
    data or metadata ready to be read in yet */
  unsigned long long wfmrevision;
  dg_real *data; /* pointer to mmap'd POSIX shared memory data */
  size_t mmaplen; /* # of bytes for mmap/munmap() */
  int len; /* total number of elements  (=dimlen[0]*dimlen[1]*...*dimlen[ndim-1]) */
  int ndim; /* number of dimensions */
  int *dimlen; /* length of each dimension */
  void (*Destructor)(struct Wfm *Wfm); /* call this on delete() if not NULL */
  char *shm_name; /* POSIX shared memory file name */
  int refcount; /* reference count for this waveform */
};
```

Essential to the definition of a waveform is whether the waveform is “ready”. A module should create a new waveform when the meaningful event which defines the waveform has occurred by calling CreateWfm(). This is a promise that the waveform data will be available soon. Once the waveform is complete, the creator should call NotifyChannel() with WfmReady==1 to indicate that the waveform is ready. Modules can request notification when specific channels get new waveforms and can use this to trigger updates. This is how the wfmmath.so module works. When a waveform update is defined (WfmReady==0), wfmmath.so processes its dependencies and defines new versions of all the dependent waveforms. As the calculations complete, those waveforms become “ready”. Once they are all ready, that updated list of waveforms would be available from wfmio with WFMIO:LISTREADY?.

The ChannelName specifies the channel this waveform is on. MetaData is a list of MetaDatums containing useful information about the channel (see the metadata section below). ReadyFlag indicates whether the enclosed data is “Final” or in progress. In progress data should not (in general) be read as it is incomplete. wfmrevision indicates the revision of the enclosed data. data points to the storage area for the waveform data. ndim is the number of dimensions (must be at least 1). dimlen points to an array of integers that define the lengths of each dimension. Destructor will be called when the waveform is deleted (because it is no longer in use). shm_name defines the name of the shared memory object (for shm_open) for the waveform data. refcount is a reference count for the waveform. It is automatically set to 1 when created to represent the fact that it is the current revision of its channel. When another revision is created, this revisions reference count will be decremented and it will be automatically deleted unless otherwise locked by a call to WfmReference(). Note that once the ReadyFlag is set, Wfm's are “Final” and may not be changed.

A new waveform is created by a call to CreateWfm(). Each call to CreateWfm() must be matched (not necessarily immediately) with a call to NotifyChannel(Chan,Wfm,1) once the data for the new waveform is ready. Note that between CreateWfm() and NotifyChannel() a waveform listing will show this as the current revision even though it is not available yet. Any attempt to download the waveform or its metadata will result in the command being blocked until the data is available.

6.1.3 Transactions

The StartTransaction() and EndTransaction() calls are used to ensure that if multiple waveforms are to be updated as an atomic event, that the event is indeed treated as atomic. For example the CreateWfm() calls for each channel of a multichannel waveform acquisition should be treated atomically. This is achieved by placing the CreateWfm()

calls between StartTransaction() and EndTransaction(). This results in update notifications being postponed until EndTransaction() is called at which all of the new waveforms have been created. Transactions can be nested, but notifications will only be generated once EndTransaction is called for the last time. Be warned that multiple Update() calls can happen during EndTransaction, but that only the last is guaranteed to have all the latest relevant versions defined.

6.1.4 Metadata

Each waveform is stored along with various metadata. Some of this metadata is standardized. Metadata can be of type string, real, or integer. The standardized metadata elements are:

Name	Type	Value
ProbeAtten	double	Attenuation factor of probe
MinLevel	double	minimum level achievable in this waveform
MaxLevel	double	maximum level achievable in this waveform
CoordN	string	Axis label for dimension N
UnitsN	string	units for dimension N
IniValN	double	Coordinate of first element of dimension N
StepN	double	Element step size of dimension N

6.1.5 Physical storage

The acquired waveforms are stored in POSIX shared memory. On Linux this means the virtual directory /dev/shm. By default, /dev/shm is limited in size to half the physical memory. When acquiring large waveforms or large numbers of images or performing large amounts of averaging with AVG instead of AVGONCE this can fill up, leading to a "Bus error". Solutions include fixing any bug causing waveforms to stay in memory, using AVGONCE instead of AVG, cleaning out cruft in /dev/shm (`rm ''/dev/shm/dg_*''`), or expanding /dev/shm. /dev/shm can be expanded by editing /etc/fstab and changing the options entry from `defaults` to `size=4g` or the size of your choice.

6.2 fftwlink.so

fftwlink serves as a link to the FFTW library. It does four things:

1. Provides centralized configuration management.
2. Link in the FFTW library (version 3).
3. Provides uniform function names regardless of whether USE_SINGLE_PRECISION is set.
4. Provides a mutex to lock when calling fftw functions (except fftw_execute()).

6.2.1 Configuration parameters

Parameter	Type	Value
nthreads	integer	number of threads to use for fftw calculations authentication code (default 2).
fftw_estimate	(none)	flag to enable FFTW_ESTIMATE type planning
fftw_measure	(none)	flag to enable FFTW_MEASURE type planning
fftw_patient	(none)	flag to enable FFTW_PATIENT type planning
fftw_exhaustive	(none)	flag to enable FFTW_EXHAUSTIVE type planning

6.2.2 Types

- fftwlink_plan_t: USE_SINGLE_PRECISION-independent replacement for fftw_plan

6.2.3 Global variables

- int fftwlink_nthreads: User configured number of threads
- pthread_mutex_t fftwlink_mutex: Mutex to lock while calling fftw routines other than fftw_execute.
- fftwlink_planning_flags: User specified flags to OR when creating a plan.

6.2.4 Functions

- fftwlink_destroy_plan(): USE_SINGLE_PRECISION-independent replacement for fftw_destroy_plan()
- fftwlink_plan_r2r_1d(): USE_SINGLE_PRECISION-independent replacement for fftw_plan_r2r_1d()
- fftwlink_execute(): USE_SINGLE_PRECISION-independent replacement for fftw_execute()
- fftwlink_plan_many_r2r(): USE_SINGLE_PRECISION-independent replacement for fftw_plan_many_r2r()

6.2.5 Notes

- You must pthread_mutex_lock() fftwlink_mutex before making fftw calls. You should then pthread_mutex_unlock() fftwlink_mutex before calling fftwlink_execute() on your plan so that other threads can build plans while yours executes.
- If fftw_measure, fftw_patient, or fftw_exhaustive are set, expect to wait a while for fftw to build wisdom. During this time fftwlink_mutex must be locked, so any other thread that tries to use fftw will block.

6.3 library_prototype.so

This is an example library that uses an AnaGram parser to read its configuration file. The editable source file is `library_prototype.synm4`.

6.4 metadata.so

The metadata library extends the metadata-handling capabilities of wfmstore.so. It has two main functions:

1. Providing support routines for reading and writing metadata: `EscapeMetaDatumString` and `ParseEscapedMetaDatumString`
2. Assisting modules that import external metadata when they create a waveform. Such modules create a `struct MetaData` to store a list of preset metadata and a list of queries for dynamic metadata. When the module creates a new waveform, it calls GrabNextMetaData() to snapshot the preset metadata and query the dynamic metadata. As dynamic metadata may not be instantly available, the parameter DoneCallback() will be called once all the dynamic metadata has been collected (but may be called immediately).

An illustrative example of AnaGram code for specifying metadata is in the comments at the start of metadata.c

6.4.1 Prerequisites

- Library rpc.so

6.4.2 Configuration parameters

The following configuration parameters are used not by the metadata library itself, but by modules which subscribe to its services:

Parameter	Type	Value
setstaticmetadatum(identifier)	metadatum value	Specify a metadatum name (identifier) and value to be specified with new waveforms created by this module. The metadatum value can be a quoted string to create a string metadatum, can be an integer (no decimal point) to create an integer metadatum, or can be a number with a decimal point to create a floatin point metadatum.
addquerymetadatum(identifier)	quoted string	Specify that the module should issue the command specified by the quoted string and interpret the result as the value of a metadatum named by the identifier.

6.4.3 Commands

Since metadata.so is a library, not a module, it cannot accept commands directly. However, the following commands are accepted by modules which subscribe to metadata.so's services:

(commands begin on next page)

metadata:SETSTATICMETADATUM

Syntax

`metadata:SETSTATICMETADATUM` <**metadatum name**> <**metadatum value**>

Description

Specify that the module should add a piece of metadatum to each acquisition.

Parameters

<**metadatum name**>	The name of the metadatum to add.
<**metadatum value**>	The value the metadatum should have.

Notes

- The type of the metadatum is determined syntactically. If <**metadatum value**> is a quoted string, then the metadatum will be of string type. If it is a number with a decimal point, then the metadatum will be of real (floating point) type. If it is a number without a decimal point it will be an integer.

See also

- metadata:DELMETADATUM (pg. 48)

metadata:ADDQUERYMETADATUM

Syntax

`metadata:ADDQUERYMETADATUM` **<metadatum name>** **<query command>**

Description

Specify that the module should add a piece of metadatum to each acquisition by querying another module.

Parameters

<metadatum name>	The name of the metadatum to add.
<query command>	The query command to issue.

Notes

- The type of the metadatum is determined syntactically. If the result of the query command is a quoted string, then the metadatum will be of string type. If it is a number with a decimal point, then the metadatum will be of real (floating point) type. If it is a number without a decimal point it will be an integer.

See also

- metadata:DELMETADATUM (pg. 48)

`metadata:DELMETADATUM`

Syntax

`metadata:DELMETADATUM` <**metadatum name**>

Description

Specify that the module should no longer add the specified piece of metadatum to each acquisition.

Parameters

<**metadatum name**>	The name of the metadatum to no longer add.

Notes

See also

- metadata:SETSTATICMETADATUM (pg. 46)
- metadata:ADDQUERYMETADATUM (pg. 47)

6.5 multiio.so

MultiIO is a library for controlling instruments that use text- or binary-based serial commands. A key advantage is that it presents the same API regardless of the underlying transport mechanism. For example, a benchtop voltmeter can be controlled over serial or GPIB. Code for MultiIO can be transformed from using serial to using gpib simply by changing the "URI" used to open the voltmeter. For example, `mio_open(‘‘serial:///dev/ttyS0:9600:8:n:1’’)` would connect using the serial port `/dev/ttyS0` at 9600 baud. The only change to support gpib (using the linux-gpib.sourceforge.net driver) would be to change that line to `mio_open(‘‘gpib://0:4:0:1:10’’)` if the voltmeter is at gpib primary address 10 (decimal).

For gpib support to work correctly, the file "/etc/gpib.conf" must exist, the driver for your card must be loaded (e.g. `modprobe tnt4882`) and the program `gpib_config` must have been run. Typically these things are done on boot in `/etc/rc.d/rc.local`.

Timeouts can be specified with mio_settimeouts() for the readline() and writetmo() calls.

6.5.1 Functions

MultiIO provides just a handful of routines that together provide a complete API to communicate with laboratory devices or software packages.

- `struct multiio *mio_open(char *uri)` – Open a connection with the specified URI
- `int mio_printf(struct multiio *fh, char *Fmt,...)` – Print a text string to the device. Returns number of characters output or negative to indicate an error or EOF.
- `int mio_scanf(struct multiio *fh, char *Fmt,...)` – Wait for a linefeed-terminated text string from the device, process with scanf. Returns number of conversions performed, otherwise negative in case of error or EOF.
- `char *mio_readline(struct multiio *fh)` – Wait for a linefeed-terminated text string from the device. Returns string to be free()'d, or NULL for error or EOF.
- `int mio_read(struct multiio *fh,void *buf,size_t nbytes)` Read `nbytes` bytes of binary data from the device. Returns length read, 0 for EOF, or negative for error.
- `int mio_write(struct multiio *fh,void *buf,size_t nbytes)` Write `nbytes` bytes of binary data to the device. Returns length written, 0 for EOF, or negative for error.
- `void mio_close(struct multiio *fh)` Close access to the device.

6.5.2 URI's

- `tcp://hostname:portnumber` to open a TCP/IP connection to the specified host and port.

- `fd://readfd:writefd` to use the already-open file descriptors (pipes) readfd and writefd.
- `serial://device:baud:databits:parity:stopbits:flags` to use a serial port (e.g. `serial:///dev/ttyS0:9600:8:n:1`). Flags are delimited by the vertical bar ('—'), and the only flag currently supported is 'nortscts', which disables the use of RTS/CTS handshaking.
- `gpib://board_index:pad:sad:send_eoi:eos` to use gpib with the specified board index, primary address, secondary address, send_eoi flag, and eos character and flags. See the comment at the front of mio_gpib_setup() for more details.

All URI formats in addition accept trailing slash-delimited flags to enable specific modes. The only current such mode is "debug", which will cause all transmitted and received data to be logged to stderr. For example, `tcp://1.2.3.4:56/debug`.

6.6 dg_python.so

6.6.1 Introduction

dg_python.so provides a common interface to python scripting for modules. It has the following functions:

1. Import the libpython.so shared object.
2. Configure the python interpreter for a multithreaded environment
3. Maintain the main thread state (stored in the global variable pythonmainThreadState), with its reference to the main interpreter state.
4. Import a few common modules (currently thread, sys, StringIO)
5. Provide a means to define common Python functions and class definitions through the initialization parameter block of the library
6. Provide a replacement for sys.stdout that stores output strings on a per-thread basis if configured by the thread with sys.stdout.register().

Initialization commands, function definitions, class definitions, etc. can be provided in the configuration block of dg_python.so when the module is initialized. Any variables defined there will be within the scope of the "dg_python" python module (i.e. can be accessed following `import dg_python`).

This library substitutes sys.stdout with a newly defined class dgp_ThreadWriter. Call sys.stdout.register(instance of StringIO.StringIO) to store stdout for your thread in a string. The StringIO instance can be then be obtained from sys.stdout.getwriter(). When done with the thread, please cleanup by calling sys.stdout.unregister().

It also provides a routine called `dg_python.rpc_async()` for making rpc calls to dataguzzler modules. A routine called `dg_python.findmodule()` can be used to report whether a module exists.

6.7 dio8bit.so

dio8bit.so is a library for generically accessing 8 bit digital I/O ports. It alone does not support any hardware, but libraries that do can register with dio8bit.so so that their ports are accessible through the dio8bit.so interface. It is generally acceptible to open a specified port multiple times, and the lower level interfaces handle sharing. Be wary of DDR conflicts, however. Some underlying hardware does not support independent directions for each bit. If this is the case, all code accessing a given port should set the DDR identically.

6.7.1 Functions

- `struct dio8port *dio8open(char *classname,char *devname)` – Open a port with the specified driver class (e.g. das4020dio) and devicename (e.g. /dev/das4020-12/dio0_0B).
- `void dio8setddr(struct dio8port *port, uint8_t readWRITE,uint8_t mask)` – Set the DDR bits specified by mask to the value readWRITE (1 indicates read, 0 indicates write).
- `void dio8close(struct dio8port *port)` – Close the specified port
- `void dio8writeport(struct dio8port *port,uint8_t val)` – Write the specified byte out to the output bits of the port
- `uint8_t dio8readport(struct dio8port *port)` – Read out the current state of the port (input bits) OR'd with the currently specified output bits.
- `struct dio8class *dio8createclass(char *Name,int structlen)` – Allocate a dio8class structure.
- `void dio8addclass(struct dio8class *newclass)` – Add a newly created dio8class structure to the master list

6.8 das4020dio.so

das4020dio.so is a library for accessing the 8 bit digital I/O ports build into the PCI-DAS4020/12 data acquisition card. It registers with dio8bit.so so that its ports are accessible through the dio8bit.so interface with the classname "das4020dio". It is generally acceptible to open a specified port multiple times. Be wary of DDR conflicts, however, because the underlying hardware does not support independent directions for each bit. All code accessing a given port should therefore set the DDR identically.

6.8.1 Prerequisites

- Library dio8bit.so

6.8.2 Functions

- `struct das4020dioport *das4020dioopen(char *devname)` – Open a port with the specified driver class (e.g. das4020dio) and devicename (e.g. /dev/das4020-12/dio0_0B).
- `void das4020diosetddr(struct das4020dioport *port, uint8_t readWRITE,uint8_t mask)` – Set the DDR bits specified by mask to the value readWRITE (1 indicates read, 0 indicates write).
- `void das4020dioclose(struct das4020dioport *port)` – Close the specified port
- `void das4020diowriteport(struct das4020dioport *port,uint8_t val)` – Write the specified byte out to the output bits of the port
- `uint8_t das4020dioreadport(struct das4020dioport *port)` – Read out the current state of the port (input bits) OR'd with the currently specified output bits.

Chapter 7

Module Reference

Introduction

Modules are optional command processors. When specified in the configuration file, a module is loaded, given a specific name, and initialized. When commands using that name are issued, the module is called to process those commands. A single module may be included more than once under different names. Modules are not permitted static data (global variables), but should instead attach any necessary data to the module structure (struct Module). Symbols defined in a module are not accessible in libraries or other modules, but modules may use symbols defined in libraries (provided the library was specified before the module in the configuration file).

7.1 auth.so

The authentication module, auth.so, authenticates TCP/IP connections to the dataguzzler kernel so that they can issue commands. This module is treated specially by the kernel and must have the name “AUTH” to work properly.

7.1.1 Configuration parameters

Parameter	Type	Value
AuthCode(domain name)	quoted string	Enables access from the specified domain name or it subdomains with the specified authentication code.
AuthCode(IP address)	quoted string	Enables access from the specified IP address with th specified authentication code.
AuthCode(IP address/IP netmask)	quoted string	Enables access from the specified IP subnet with th specified authentication code.

7.1.2 Commands

(commands begin on next page)

AUTH

Syntax

`AUTH` <**authcode**>

Description

Authenticates an incoming TCP/IP connection

Parameters

<**authcode**>	Authentication password

Notes

- You must use an authentication code that is valid for the IP address you are connecting from. Authentication is configured using the file `/etc/daq_auth.conf`. If this file does not exist, internal defaults allow connections only on the loopback address (`127.0.0.1`) using `xyzzy` as the authentication code.

See also

7.2 module_prototype.so

module_prototype.so is an example module that doesn't do anything useful.

7.2.1 Configuration parameters

Parameter	Type	Value
cardname	quoted string	Dummy initialization parameter

7.2.2 Commands

(commands begin on next page)

moduleprototype:CHx:PROBEATTEN

Syntax

```
moduleprototype:CHx:PROBEATTEN <attenuation factor>
                CHx:PROBEATTEN?
```

Description

Set or query dummy multichannel parameter

Parameters

<attenuation factor>	Dummy attenuation factor

Notes

- This is just a test. Don't use it in any actual systems

See also

`moduleprototype:FREQ`

Syntax

```
moduleprototype:FREQ <frequency>
                FREQ?
```

Description

Set or query dummy frequency

Parameters

<**frequency**>	Dummy frequency

Notes

- This is just a test. Don't use it in any actual systems

See also

7.3 wfmio.so

wfmio.so allows uploading and downloading of waveform data as well as listing of waveforms in the wfmstore.so library.

7.3.1 Prerequisites

- library wfmstore.so
- library metadata.so

7.3.2 Configuration parameters

Parameter	Type	Value
(none)		

7.3.3 Commands

(commands begin on next page)

wfmio:COPY

Syntax

`wfmio:COPY` <**waveform name**> <**copy name**>

Description

Copy the specified waveform.

Parameters

<**waveform name**>	Name of the original waveform
<**copy name**>	Name for the copy

Notes

- The copy can be deleted with the wfmio:DELETE command.

See also

- wfmio:DELETE (pg. 64)

wfmio:DATA

Syntax

```
wfmio:DATA <waveform name> <revision> <metadata> <data length>
      <waveform data>
      DATA? <waveform name> <revision>
```

Description

Obtain the sample data of a waveform

Parameters

<waveform name>	Name of the waveform of interest
<revision>	Revision of interest
<data length>	Waveform dimensions (see below)
<waveform data>	Binary encoded waveform data
<metadata>	Waveform metadata

Notes

- <**data length**> is the number of dimensions followed by a series of integers, each in square brackets, that define the dimensions of the waveform in samples. For example 3 [65536] [32] [32] specifies that the data is 32x32 waveforms of 65536 points each.
- <**waveform data**> is binary IEEE floating point (either single or double precision according to the result of wfmio:REALSZ) that has been encoded with a binary NOT with (post-inversion) characters 0-32, ';', and '%' replaced with escape sequences. The escape sequence is initiated by the '%' character and consists of '%' followed by the escaped character + 0x80.

See also

- wfmio:REALSZ (pg. 74)
- wfmio:METADATA (pg. 73)

wfmio:DATASHM

Syntax

`wfmio:DATASHM` <**waveform name**> <**revision**> <**metadata**> <**data length**> <**posix shm name**>
`DATASHM?` <**waveform name**> <**revision**>

Description

Obtain the sample data of a waveform through POSIX shared memory

Parameters

<**waveform name**>	Name of the waveform of interest
<**revision**>	Revision of interest
<**metadata**>	Waveform metadata (see wfmio:METADATA).
<**data length**>	Waveform dimensions (see below)
<**posix shm name**>	Specification of the POSIX shared memory name for access to the binary waveform data.

Notes

- Query only. Command syntax indicates format of response.
- <**data length**> is the number of dimensions (an integer) followed by a series of integers, each in square brackets, that define the dimensions of the waveform in samples. For example 3 [65536] [32] [32] specifies that the data is 32x32 waveforms of 65536 points each.
- <**posix shm name**> is the name of a POSIX shared memory area that can be passed to shm_open() to obtain access to the waveform data. The data itself is stored as binary IEEE floating point (either single or double precision according to the result of wfmio:REALSZ)

See also

- wfmio:REALSZ (pg. 74)
- wfmio:DATA (pg. 61)
- wfmio:METADATA (pg. 73)

`wfmio:DELETEALL`

Syntax

```
wfmio:DELETEALL
```

Description

Delete all user-uploaded or user-copied waveforms from the waveform memory.

Parameters

Notes

See also

- wfmio:DELETE (pg. 64)
- wfmio:COPY (pg. 60)
- wfmio:DATA (pg. 61)

wfmio:DELETE

Syntax

`wfmio:DELETE` <**waveform name**>

Description

Delete the specified waveform.

Parameters

<**waveform name**>	Name of the waveform

Notes

- Any revisions of the waveform that are locked will remain in memory, but will not be shown by wfmio:LIST or wfmio:LISTLOCK.
- Only waveforms created by the user can be deleted with the wfmio:DELETE command.

See also

- wfmio:DELETEALL (pg. 63)
- wfmio:COPY (pg. 60)
- wfmio:DATA (pg. 61)

wfmio:GLOBALREADYREV

Syntax

```
wfmio:GLOBALREADYREV <global revision>
      GLOBALREADYREV?
```

Description

Wait for a specific global revision to be ready or query the latest ready global revision.

Parameters

<global revision>	The global waveform revision

Notes

See also

- wfmio:GLOBALREV (pg. 67)
- wfmio:GLOBALREADYREVTIMEOUT (pg. 66)

wfmio:GLOBALREADYREVTIMEOUT

Syntax

`wfmio:GLOBALREADYREVTIMEOUT` <**global revision**> <**timeout**>

Description

Wait for a specific global revision to become ready with a timeout (optional units, default ms)

Parameters

<**global revision**>	The global waveform revision
<**timeout**>	The timeout, in ms unless otherwise specifed

Notes

- The specified global revision is not locked into memory so it may have been discarded in favor of a more recent (ready) waveform by the time you manage to read it.

See also

- wfmio:GLOBALREVTIMEOUT (pg. 68)
- wfmio:GLOBALREADYREV (pg. 65)

wfmio:GLOBALREV

Syntax

```
wfmio:GLOBALREV <global revision>
      GLOBALREV?
```

Description

Wait for a specific global revision or query the current global revision.

Parameters

<global revision>	The global waveform revision

Notes

See also

- wfmio:GLOBALREVTIMEOUT (pg. 68)
- wfmio:GLOBALREADYREV (pg. 65)

`wfmio:GLOBALREVTIMEOUT`

Syntax

`wfmio:GLOBALREVTIMEOUT` <**global revision**> <**timeout**>

Description

Wait for a specific global revision with a timeout (optional units, default ms)

Parameters

<**global revision**>	The global waveform revision
<**timeout**>	The timeout, in ms unless otherwise specifed

Notes

See also

- wfmio:GLOBALREV (pg. 67)
- wfmio:GLOBALREADYREVTIMEOUT (pg. 66)

wfmio:LIST

Syntax

`wfmio:LIST` <**waveform count**> <**global revision**> <**waveform1 name**> <**waveform1 revision**> <**waveform2 name**> <**waveform2 revision**> ...
`LIST?`

Description

List the current revisions of all waveforms

Parameters

<**waveform count**>	Number of waveform descriptions to follow
<**global revision**>	Global revision count corresponding to this waveform revision set.
<**waveform1 name**>	Name of the first waveform
<**waveform1 revision**>	Revision of first waveform

Notes

- Query only. Command syntax indicates format of response.
- Since this routine does not lock the waveforms into memory there is no guarantee that the specified waveforms or revisions will remain in memory

See also

- wfmio:METADATA (pg. 73)
- wfmio:DATA (pg. 61)
- wfmio:LISTLOCK (pg. 71)
- wfmio:LISTREADY (pg. 70)

wfmio:LISTREADY

Syntax

`wfmio:LISTREADY` **<waveform count> <global revision> <waveform1 name> <waveform1 revision> <waveform2 name> <waveform2 revision>** . . .
`LISTREADY?`

Description

List the current "ready" revision state of all waveforms

Parameters

<waveform count>	Number of waveform descriptions to follow
<global revision>	Global revision count corresponding to this waveform revision set.
<waveform1 name>	Name of the first waveform
<waveform1 revision>	Revision of first waveform

Notes

- Query only. Command syntax indicates format of response.
- Since this routine does not lock the waveforms into memory there is no guarantee that the specified waveforms or revisions will remain in memory
- The current "ready" revision state corresponds to a consistent set of waveforms for which all calculations have been completed.

See also

- wfmio:METADATA (pg. 73)
- wfmio:DATA (pg. 61)
- wfmio:LISTREADYLOCK (pg. 72)
- wfmio:LISTREADY (pg. 70)

wfmio:LISTLOCK

Syntax

`wfmio:LISTLOCK` **<waveform1 name> <waveform1 revision> <waveform2 name> <waveform2 revision>** ...
`LISTLOCK?`

Description

List the current revisions of all waveforms and lock those revisions in memory

Parameters

<waveform1 name>	Name of the first waveform
<waveform1 revision>	Revision of first waveform

Notes

- Query only. Command syntax indicates format of response.
- You must call wfmio:UNLOCK on each of the waveform/revision combinations returned, lest they be stuck in memory. A dropped TCP/IP connection automaticall unlocks all waveforms locked by that connection.

See also

- wfmio:METADATA (pg. 73)
- wfmio:DATA (pg. 61)
- wfmio:UNLOCK (pg. 81)
- wfmio:LIST (pg. 69)

wfmio:LISTREADYLOCK

Syntax

`wfmio:LISTREADYLOCK` **<waveform1 name> <waveform1 revision> <waveform2 name> <waveform2 revision>** ...
`LISTREADYLOCK?`

Description

List the current “ready” revision state of all waveforms and lock those revisions in memory

Parameters

<waveform1 name>	Name of the first waveform
<waveform1 revision>	Revision of first waveform

Notes

- Query only. Command syntax indicates format of response.
- The current “ready” revision state corresponds to a consistent set of waveforms for which all calculations have been completed.
- You must call wfmio:UNLOCK on each of the waveform/revision combinations returned, lest they be stuck in memory. A dropped TCP/IP connection automaticall unlocks all waveforms locked by that connection.

See also

- wfmio:METADATA (pg. 73)
- wfmio:DATA (pg. 61)
- wfmio:UNLOCK (pg. 81)
- wfmio:LISTREADY (pg. 70)
- wfmio:LISTLOCK (pg. 71)

`wfmio:METADATA`

Syntax

`wfmio:METADATA?` <**waveform name**> <**revision**>

Description

Obtain the metadata for a waveform

Parameters

<**waveform name**>	Name of the waveform of interest
<**revision**>	Revision of interest

Notes

- The response is of the form `wfmio:METADATA` <**waveform name**> <**revision**> { <**metadatum name**>`:`<**type**>`=`<**value**> <**metadatum name**>`:`<**type**>`=`<**value**> ...} <**data length**>
- <**data length**> is the number of dimensions (an integer) followed by a series of integers, each in square brackets, that define the dimensions of the waveform in samples. For example 3 [65536] [32] [32] specifies that the data is 32x32 waveforms of 65536 points each.
- Valid types are:
  - `integer`: <**value**> is an integer.
  - `string`: <**value**> is a quoted string
  - `real`: <**value**> is a floating point number.

See also

wfmio:REALSZ

Syntax

```
wfmio:REALSZ <bytes per floating point number>
     REALSZ?
```

Description

Returns the number of bytes per floating point number returned by wfmio:DATA or wfmio:DATASHM.

Parameters

<bytes per floating point number>	Number of bytes per floating point number.

Notes

- Query only. Command syntax indicates format of response.
- Should be either 4 (IEEE single precision) or 8 (IEEE double precision).

See also

- wfmio:DATA (pg. 61)
- wfmio:DATASHM (pg. 62)

wfmio:REVISION

Syntax

```
wfmio:REVISION <waveform name> <waveform revision>
          REVISION? <waveform name>
```

Description

List the current revision of the specified waveform

Parameters

<waveform name>	Name of the waveform
<waveform revision>	Revision of the waveform

Notes

- Query only. Command syntax indicates format of response.
- Since this routine does not lock the waveforms into memory there is no guarantee that the specified waveforms or revisions will remain in memory

See also

- wfmio:METADATA (pg. 73)
- wfmio:DATA (pg. 61)
- wfmio:LIST (pg. 69)
- wfmio:REVISIONLOCK (pg. 76)

wfmio:REVISIONLOCK

Syntax

`wfmio:REVISIONLOCK` <**waveform name**> <**waveform revision**>
`REVISIONLOCK?` <**waveform name**>

Description

List the current revision of the specified waveform and lock that revision in memory, or wait for at least a specific revision and lock it in memory

Parameters

<**waveform name**>	Name of the waveform
<**waveform revision**>	Revision of the waveform

Notes

- Query format finds the latest (not necessarily ready) revision of the specified waveform. The command format waits for the specified revision (or a later version) to become available.
- In both cases the name and revision of the locked waveform are returned.
- You must call wfmio:UNLOCK on the waveform/revision combinations returned, lest it be stuck in memory. A dropped TCP/IP connection automaticall unlocks all waveforms locked in memory by that connection.
- This attempts to obtain the specified revision, but may return a more recent version than specified.

See also

- wfmio:METADATA (pg. 73)
- wfmio:DATA (pg. 61)
- wfmio:LIST (pg. 69)
- wfmio:REVISION (pg. 75)

wfmio:REVISIONREADYLOCK

Syntax

`wfmio:REVISIONREADYLOCK` <**waveform name**> <**waveform revision**>

Description

Wait for at least a specific revision of a waveform to become ready, and lock it in memory

Parameters

<**waveform name**>	Name of the waveform
<**waveform revision**>	Revision of the waveform

Notes

- The command format waits for the specified revision (or a later version) to become available and ready.
- In both cases the name and revision of the locked waveform are returned.
- You must call wfmio:UNLOCK on the waveform/revision combinations returned, lest it be stuck in memory. A dropped TCP/IP connection automaticall unlocks all waveforms locked in memory by that connection.
- This attempts to obtain the specified revision, but may return a more recent version than specified.

See also

- wfmio:METADATA (pg. 73)
- wfmio:DATA (pg. 61)
- wfmio:LIST (pg. 69)
- wfmio:REVISION (pg. 75)

wfmio:RPCDATA

Syntax

`wfmio:RPCDATA` <**channel name**>
`RPCDATA` <**host name**>`:`<**pid**>`:0x`<**hexadecimal address**>

Description

Ask wfmio:to allocate a struct Wfm so that data can be provided to a wfmio:-owned waveform. (RPC USE ONLY)

Parameters

<**channel name**>	The channel on which to create a new struct Wfm
<**host name**>	Host name of the wfmio:module
<**pid**>	Process ID wfmio:module
<**hexadecimal address**>	Address of the `struct Wfm`

Notes

- The first syntax specified above is the command syntax. The second syntax is the specification of the response from the server.
- This command will complete immediately.
- Host name and PID are provided for checking purposes. Don't use the returned address unless host name and PID match.
- This call provides an empty struct Wfm. The caller must WfmAlloc() and write data and metadata.
- Once the waveform has been written (or if it couldn't be written because of a hostname mismatch or other error), the caller must call wfmio:RPCDATADONE to issue the WfmNotify() that this waveform is ready.
- To repeat, wfmio:RPCDATADONE MUST be issued shortly after a successful call to wfmio:RPCDATA.
- If wfmio:RPCDATA cannot be used because of a host name or pid mismatch, wfmio:DATA can be used to upload a waveform.

See also

- wfmio:RPCDATADONE (pg. 80)

wfmio:RPCDATADONE

Syntax

```
wfmio:RPCDATADONE <host name>:<pid>:0x<hexadecimal address>
      RPCDATADONE <channel name> <revision>
```

Description

Respond following a wfmio:RPCDATA indicating that the struct Wfm is complete. (RPC USE ONLY)

Parameters

<host name>	Host name of the wfmio:module
<pid>	Process ID wfmio:module
<hexadecimal address>	Address of the `struct Wfm`
<channel name>	The channel on which to create a new struct Wfm
<revision>	revision of the waveform which was just created

Notes

- The first syntax specified above is the command syntax. The second syntax is the specification of the response from the server.
- This command will complete immediately.
- Host name and PID should be those provided by the response from wfmio:RPCDATA.

See also

- wfmio:RPCDATA (pg. 78)

`wfmio:UNLOCK`

Syntax

`wfmio:UNLOCK` **<waveform name>** **<waveform revision>**

Description

Unlock the specified revision of the specified waveform.

Parameters

<waveform name>	Name of the waveform
<waveform revision>	Revision of the waveform

Notes

- The specified waveform and revision must have been previously locked in memory by wfmio:REVISIONLOCK or wfmio:LISTLOCK.

See also

- wfmio:LISTLOCK (pg. 71)
- wfmio:REVISIONLOCK (pg. 76)

wfmio:WFMS

Syntax

```
wfmio:WFMS?
```

Description

Download the full set of user-defined waveforms.

Parameters

Notes

- The response begins with wfmio:DELETEALL; and continues with semicolon-deliniated wfmio:DATA commands containing the data and metadata.
- The output is intended to be fed back in to recreate the waveforms

See also

- wfmio:DATA (pg. 61)

7.4 wfmmath.so

wfmmat.so allows creation of additional channels that are automatically recalculated mathematical functions of pre-exisiting channels.

7.4.1 Prerequisites

- library wfmstore.so
- library fftwlink.so (if any fftw-using functions are included)

7.4.2 Configuration parameters

Parameter	Type	Value
numthreads	unsigned decimal integer	number of worker threads.

7.4.3 Writing mathematical functions

A new mathematical function is built into wfmmath.so by adding the newly created .synhm4 file for the mathematical function to the dependencies of wfmmath.syn in the Makefile in the modules directory, e.g.:

```
wfmmath.syn: wfmmath.synm4 wfmmath_avg.synhm4 wfmmath_my_new_module.synhm4
```

The new module is defined in a synhm4 file and must obey the rules of M4-preprocessed syntax files (e.g. use //# for comments, etc.) Typically the new module begins with the definition of a new production of the token "`math function`", for example

```
(struct MathFcn *)math function
    -> "CORR",'(',identifier,',',identifier,')' =CreateMath(
         "CORR",STK,"01","",
         0,0,0.0,0.0,
         &CalcCorr,&PrintCorr,NULL,NULL,NULL,NULL);
```

The function CreateMath creates a struct MathFcn (defined in wfmmath.synm4) with the specified parameters. The third and fourth parameters are called "dependencyparams" and "stringparams", respectively. These two parameters are strings of digits that indicate which strings to be popped off the string stack count as dependencies or strings. For example, `stringparams="13"` indicates that parameters #1 and #3 (the 2nd and 4th parameters) should be interpreted as strings, not module dependencies.

The last six parameters to CreateMath are functions (or optional functions) which should be defined in an embedded C section inside the .synhm4 file. `CalcFcn(struct ModData *md,struct MathFcn *p)` Defines the function to call withiin a worker thread to perform the calculation. Note that because it is in a thread, the things it can read or write are very much restricted. It may read the data and metadata from waveforms which are ready (ReadyFlag == 0) and have already been locked into memory by a WfmReference(). It may write to the MetaData and data of GenWfm. `PrintFcn(struct ModData *md,struct Conn *c,struct Channel *Chan,struct MathFcn *fcn)` should do a ResPrintf() of the parameters needed to recreate this function. `PrepareFcn(struct ModData *md,struct MathFcn *m)` is called from the main thread context to prepare the MathFcn `m` for calculation. It is called just before `m` is placed on the PendingComputation list. If NULL, no such function is called. `CleanupFcn(struct ModData *md,struct MathFcn *m)` is called from the main thread context after the calculation of MathFcn `m` has completed and immediately after it has been removed from the CompletedComputation list. This function should do any necessary cleanups of data from the computation. `CopyConstructor(struct MathFcn *orig,struct MathFcn *copy)` is called when the use #2 (see below) MathFcn is made from the use #1 MathFcn. This routine can copy any function-specific data if necessary. `Destructor(struct MathFcn *m)` is called then a MathFcn (either use #1 or use #2) is no longer needed.

Be sure to lock the fftwlink_mutex when making fftw calls (except fftwlink_execute).

struct MathFcn

```
struct MathFcn {
  struct Node Node;
  char *FcnName;
  char *Dependencies[MAX_DEPENDENCY_PARAMS]; //# these are the parameters that depend on other waveforms
  unsigned long long DependencyRevisions[MAX_DEPENDENCY_PARAMS]; //# These are the revisions of the depende
  struct Wfm *WfmDependencies[MAX_DEPENDENCY_PARAMS]; //# Only used in use#2 (above), NULL otherwise. WfmRe
  char *StringParams[MAX_STRING_PARAMS];
  int queuedflag; //# Indicate that this calculate has been queued on the PendingComputation List
  int Disabled; //# Use #1 only. If non-zero, do not define new revisions
  long IntegerParam1;
  long IntegerParam2;
  double RealParam1;
  double RealParam2;
  struct List NotifyPtrList; //# Empty unless this is the struct MathFcn pointed to by Chan->ModSpecific
  void (*CalcFcn)(struct ModData *md,struct MathFcn *p); //# function to perform calculation (runs in diffe
  void (*PrintFcn)(struct ModData *md,struct Conn *c,struct Channel *Chan,struct MathFcn *m);
  void (*PrepareFcn)(struct ModData *md,struct MathFcn *m);
  void (*CleanupFcn)(struct ModData *md,struct MathFcn *m);
  void (*CopyConstructor)(struct MathFcn *orig,struct MathFcn *copy); //# Called when the wfm-specific copy
  void (*Destructor)(struct MathFcn *m);
  void *FcnSpecific;
  struct MathFcn *ChanFcn; //# Pointer to the MathFcn of the channel (use #2 only)
  int UseCnt; //# Count of the number of currently pending computations. We can't free up this MathFcn
              //# until the UseCnt goes to zero. (use #1 only)
  struct Wfm *GenWfm; //# Waveform to be written (context #2 only)
};
```

The struct MathFcn is used to define a mathematical operation. It is used in two separate contexts:

1. To define the math function of a struct Channel. In this case Chan-¿ModSpecific points to the MathFcn and the MathFcn is not on a list.
2. To define a pending or completed computation. In this case the MathFcn may be on a list and it is also pointed to by Wfm-¿ModSpecific.

struct ModData

```
struct ModData {
struct Module Mod;
//# Module static data goes here
volatile struct List PendingComputation; //# NOTE: WorkNotifyMutex must be locked to access or modify this
volatile struct List CompletedComputation; //# NOTE: WorkNotifyMutex must be locked to access or modify thi
        struct List ThreadList;
int numthreads;
pthread_cond_t WorkNotify; //# used to notify threads that there is work to be done
pthread_mutex_t WorkNotifyMutex; //# Also locks PendingComputation and CompletedComputation lists
int parentnotifypipe[2]; //# Write a byte to parentnotifypipe[1] to notify main process that a computation
};
```

wfmmath has a private struct ModData structure that is accessible from the code used to define new math functions.

7.4.4 Commands

(commands begin on next page)

`wfmmath:CLEARAVG`

Syntax

`wfmmath:CLEARAVG` <**waveform name**>

Description

Reset averaging channel <**waveform name**>

Parameters

<**waveform name**>	Name of the math waveform to reset.

Notes

See also

`wfmmath:CLEARACCUM`

Syntax

`wfmmath:CLEARACCUM` <**waveform name**>

Description

Reset ACCUM channel <**waveform name**>

Parameters

<**waveform name**>	Name of the wfmmath:ACCUM waveform to reset.

Notes

- The the ACCUM channel will be reset and will be empty until a new version of the waveform it is dependent on appears.

See also

`wfmmath:DEF`

Syntax

`wfmmath:DEF` <**waveform name**>`=`<**function name**>`(`<**parameters . . .** >`)`
    `DEF?` <**waveform name**>

Description

Define a new channel to be a mathematical function of other channel(s), or query the mathmatical function of such a channel

Parameters

<**waveform name**>	Name of the waveform to define of query.
<**function name**>	Name of mathematical function to use
<**parameters . . .** >	Parameters to the mathematical function

Notes

- Allowable functions are:
  - result=AVG(<**channel name**>,<**number of averages**>) or
  - (result,stddev)=AVG(<**channel name**>,<**number of averages**>): Running average (and optional standard deviation) of <**channel name**>.
  - result=AVGONCE(<**channel name**>,<**number of averages**>) or
  - (result,stddev)=AVGONCE(<**channel name**>,<**number of averages**>): Average of (and optional standard deviation) of <**channel name**>.
  - ampl=FFT(<**channel name**>,<**transform dimensions**>) or
  - (ampl,phase)=FFT(<**channel name**>,<**transform dimensions**>): Fourier Transform of <**channel name**>. If <**transform dimensions**> is not specified the transform will be over the first (minor) dimension of the data. <**transform dimensions**> can either be an integer, specifying which dimension to transform over (0 being the first – minor – dimension), or it can be a list of comma separated such integers enclosed within square brackets, in which case the transform will be over all the specified dimensions, e.g. FFT(chan1,[0,2,3]) will cause a Fourier transform over the first, third, and fourth dimension. The highest transformed dimension (fourth dimension in the previous example) will have size (n/2) + 1 where n is the pre-existing length of that dimensions and the result of the division is truncated to the next lower integer, not rounded. All other dimensions will have their pre-existing sizes.
    There can be either one or two result parameters. The first (or only) result parameter is the amplitude of the Fourier transform. The second result parameter is the phase (in radians) of the

Fourier transform. The transform is normalized by multiplying by the product of the step sizes of the transformed dimensions. .

- result=CORR(<**channel 1**>,<**channel 2**>,<**dimensions**>), or
- result=CONV(<**channel 1**>,<**channel 2**>,<**dimensions**>): CORR and CONV perform cross-correlation and convolution respectively of <**channel 1**> with <**channel 2**> over dimensions <**dimensions**>. <**dimensions**> can be an integer, specifying which dimension to correlate/convolve over (0 being the first – minor – dimension), or it can be a list of comma separated such integers enclosed within square brackets, in which case the correlation/convolution will be over all the specified dimensions. If <**dimensions**> is not specified, it will be over the first (minor) dimension.
- result=ACCUM(<**channel name**>,<**number of waveforms**>: Accumulate a series of waveforms into a "waveform" with an extra dimension. For example, if <**channel name**> is two-dimensional 640x512 and <**number of waveforms**> is 6 then the result will be a "cube" of data 640*512*6. Note that all the input waveforms must be the same size or the generated result will be empty. Also note that ACCUM will not include the current revision in its series, but will start accumulating with the next version.
- result=ACCUMONCE(<**channel name**>,<**number of waveforms**>): Like ACCUM but doesn't automatically reset when full. Need to call wfmmath:CLEARACCUM manually.
- result=ADD(<**channel a**>,<**real number b**>) or
- result=ADD(<**channel a**>,<**channel b**>): Compute result=a+b. b may be of lower dimensionality than a, but the dimensions of b must match the first dimensions of a. The result gets a copy of a's metadata.
- result=SUB(<**channel a**>,<**real number b**>) or
- result=SUB(<**channel a**>,<**channel b**>): Compute result=a-b. b may be of lower dimensionality than a, but the dimensions of b must match the first dimensions of a. The result gets a copy of a's metadata.
- result=MUL(<**channel a**>,<**real number b**>) or
- result=MUL(<**channel a**>,<**channel b**>): Compute result=a*b. b may be of lower dimensionality than a, but the dimensions of b must match the first dimensions of a. The result gets a copy of a's metadata.
- result=DIV(<**channel a**>,<**real number b**>) or
- result=DIV(<**channel a**>,<**channel b**>): Compute result=a/b. b may be of lower dimensionality than a, but the dimensions of b must match the first dimensions of a. The result gets a copy of a's metadata.
- result=INT(<**channel**>) or
- result=INT(<**channel**>,<**dimension**>): Integrate <**channel**> over <**dimension**> (default 1). The result has the same dimensionality as <**channel**> and is the running integral over the specified dimension.
- result=DIFF(<**channel**>) or
- result=DIFF(<**channel**>,<**dimension**>): Differentiate <**channel**> over <**dimension**> (default 1). The result has the same dimensionality as <**channel**> and is the derivative over the specified dimension. The resulting waveform will be shifted by exactly 1/2 sample in the forward direction and its last sample will have a value of 0.0.

- result=INTEGRAL(<**channel**>) or
- result=INTEGRAL(<**channel**>,<**dimension**>): Calculate the integral of <**channel**> over the specified dimension (default 1). The result will have one less dimension than <**channel**> and is the integral over all the samples along the specified dimension.
- result=SUBEARLYAVG(<**channel**>,<**threshold**>) or
- result=SUBEARLYAVG(<**channel**>,<**threshold**>,<**dimension**>): Subtract the average of the first elements (up to <**threshold**>) along <**dimension**> of <**channel**> from <**channel**>. <**dimension**> is from 0 (minor dimension) to (ndim-1) (major dimension), with the major dimension used by default.
- result=FILTEREDINT(<**channel**>,<**freq 1**>,<**freq 2**>): Integrate the one-dimensional waveform in <**channel**>, then subtract out its average slope, then high-pass filter it with a frequency-domain raised-cosine that starts at 0.0 at <**freq 1**> and reaches 1.0 at <**freq 2**>.
- result=MAX(<**channel**>): Find the scalar maximum value (over all dimensions) of the specified channel.
- result=CROP(<**channel**>,[axis1min,axis1max],[axis2min,axis2max],...): Crop the specified waveform or image.
- result=DECIMATE(<**channel**>,<**first axis decimate factor**>,<**second axis decimate factor**>, ...): Downsample the specified channel by the specified factors in each axis.
- result=DBABS(<**channel**>): Convert to dB (return $20\log 10$(abs(<**channel**>)))

See also

- wfmmath:UNDEF (pg. 94)

`wfmmath:ENABLE`

Syntax

`wfmmath:ENABLE` <**waveform name**>

Description

Enable math channel <**waveform name**>

Parameters

<**waveform name**>	Name of the math channel to enable.

Notes

- Math channels are enabled by default.
- Enabling one result channel of a function with multiple outputs enables all result channels of that function

See also

- wfmmath:ENABLED (pg. 92)
- wfmmath:DISABLE (pg. 93)

wfmmath:ENABLED

Syntax

`wfmmath:ENABLED?` <**waveform name**>

Description

Determine whether math channel <**waveform name**> is enabled.

Parameters

<**waveform name**>	Name of the math channel to check.

Notes

- Returns response of the form wfmmath:ENABLE <**waveform name**> or wfmmath:DISABLE <**waveform name**> depending on whether <**waveform name**> is enabled.

See also

- wfmmath:ENABLE (pg. 91)
- wfmmath:DISABLE (pg. 93)

`wfmmath:DISABLE`

Syntax

`wfmmath:DISABLE` <**waveform name**>

Description

Disable math channel <**waveform name**>

Parameters

<**waveform name**>	Name of the math channel to disnable.

Notes

- Math channels are enabled by default.
- Disabling one result channel of a function with multiple outputs disables all result channels of that function

See also

- wfmmath:ENABLE (pg. 91)
- wfmmath:ENABLED (pg. 92)

`wfmmath:UNDEF`

Syntax

`wfmmath:UNDEF` <**math channel name**>

Description

Remove the math channel <**math channel name**>

Parameters

<**math channel name**>	Name of the math channel to remove.

Notes

See also

- wfmmath:DEF (pg. 88)
- wfmmath:UNDEFALL (pg. 95)

`wfmmath:UNDEFALL`

Syntax

`wfmmath:UNDEFALL` <**math channel name**>

Description

Delete all math channels

Parameters

Notes

See also

- wfmmath:DEF (pg. 88)
- wfmmath:UNDEF (pg. 94)

`wfmmath:WAITAVG`

Syntax

`wfmmath:WAITAVG` <**waveform name**>

Description

Wait for averaging channel <**waveform name**> to have performed a complete set of averages

Parameters

<**waveform name**>	Name of the math averaging channel to wait for.

Notes

- This waits for a *complete* and *ready* averaging channel. It does not lock the completed version in memory.

See also

7.5 das4020capture.so

das4020capture.so provides waveform acquisition from the analog inputs of the Measurement Computing PCI-DAS4020/12 board. It requires Warren Jasper's DAS4020 driver from ftp://lx10.tx.ncsu.edu/pub/Linux/drivers. For this module to work, the driver must be inserted into the kernel (insmod) and permissions on the /dev/das4020-12/ device nodes must be set correctly (see udev rules in the driver README). The driver include file pci-das4020.h must exist in /usr/include or /usr/local/include for this module to compile. Please note that the maximum samplecnt is limited by the size of the driver's buffer. This can be changed by adjusting ADC_BUFF_PHY_SIZE in a2dc.h and recompiling the driver. The das4020 card is ready to accept triggers once the previously captured waveform is READY (CALCSYNC FALSE) or once all dependencies of the previously captured waveform are READY (CALCSYNC TRUE).

The das4020 driver versions prior to 1.18 used different device node names. You will have to edit the config files to use the older device node names if you are using an old driver version.

7.5.1 Prerequisites

- library wfmstore.so
- library metadata.so
- library rpc.so

7.5.2 Configuration parameters

Parameter	Type	Value
cardname	quoted string	DAS4020 analog capture device base name (default is “/dev/das4020-12/ad0_”). The channel number is appended to this base name
numchannels	unsigned integer	Number of channels to use, 1-4 (default is 4).
samplecnt	unsigned integer	Number of samples to record per waveform, default is 100000
range<**i**>	“1V” or “5V”	Voltage range for channel <**i**> (default 1V)
capturefreq	real number	Capture frequency in default units of Hz (default 10 MHz). Note that 20 MHz capture supports only two channels
hwtrigsrc	“ext”,“int”, or “ch1” through “ch4”	hardware trigger source: external connector (“ext”, default), 40 pin header (“int”), or analog triggering on channel 1 through 4
channelprefix	quoted string	prefix of created channel names (default “CH”)
probeatten<**i**>	real number	attenuation factor of the probe on channel <**i**> (default 1.0).
atrigmode	“POS_HIST”, “NEG_HIST”, “POS_SLOPE”, “NEG_SLOPE”, or “WINDOW”	Analog trigger slope and mode (see das4020capture:ATRIGMODE (pg. 102)).
chan<**i**>name	quoted string	Channel name for channel <**i**>, overriding *channelprefix* (above).
atrighigh	Voltage	High voltage level for analog triggering
atriglow	Voltage	Low voltage level for analog triggering
calcsync	boolean (true or false)	If true, don’t allow new acquisitions until computations resulting from the previous acquisition are complete. (default is false)
clksrc	“INTERNAL”, “EXTBNC”, or “ADSTARTTRIG”	clock source for the waveform acquisition: INTERNAL means use the internal 40 MHz crystal. EXTBNC means the bottom BNC connector, and ADSTARTTRIG means use the ADSTARTTRIG pin on the IDC header.
dsfactor<**i**>	unsigned decimal integer	Downsampling factor. If this is greater than 1, it will cause downsampling of channel <**i**>
hwcapturefreq	frequency (Hz)	if clksrc is not `INTERNAL`, hwcapturefreq controls the internal frequency divider. Set hwcapturefreq to the capture frequency you would get with the desired divider setting if your clock was 40 MHz. For example, if you want a /4 divider, you would set hwcapturefreq to 10 MHz. Note that the divider must be at least 2 for two channel or 4 for 4 channels.
fifosize	Fifo size, in 24-bit words	Controls the DAS4020 capture FIFO size, default 32768. Reduce this if the DAS4020 is using too much PCI bandwidth and causing other cards to overflow their FIFO’s. Powers of 2 between 256 and 32768 are acceptible.

Das4020capture also supports the standard metadata initialization configuration parameters `setstaticmetadatum` and `addquerymetadatum`. The metadata provided by these methods is attached to the captured waveforms (all channels). It also supports configuration parameters of the form ch1:setstaticmetadatum to set metadata parameters for a single channel.

7.5.3 Commands

In addition to the commands described on the next pages, das4020capture also supports the standard metadata specification parameters `setstaticmetadatum`, `addquerymetadatum`, and `delmetadatum`. The metadata provided by these methods is attached to the captured waveforms (all channels). It also supports commands of the form das4020capture:ch1:setstaticmetadatum, etc. to set metadata parameters for a single channel.

(commands begin on next page)

das4020capture:ATRIGHIGH

Syntax

```
das4020capture:ATRIGHIGH <trigger voltage>
               ATRIGHIGH? <trigger voltage>
```

Description

Specify or query the low analog trigger voltage.

Parameters

<trigger voltage>	Desired trigger voltage

Notes

- The high analog trigger voltage is used for rising-edge triggering, the hysteresis of falling edge triggering, and window triggering.
- Changing the analog trigger voltage will cancel any acquisition currently in progress on the capture card.
- The actual quantized trigger voltage will be returned
- Changing the gain setting or trigger source may cause the the trigger level to be re-quantized to match the new setting.

See also

- das4020capture:HWTRIGSRC (pg. 109)
- das4020capture:ATRIGLOW (pg. 101)
- das4020capture:ATRIGMODE (pg. 102)

das4020capture:ATRIGLOW

Syntax

```
das4020capture:ATRIGLOW <trigger voltage>
               ATRIGLOW? <trigger voltage>
```

Description

Specify or query the low analog trigger voltage.

Parameters

<trigger voltage>	Desired trigger voltage

Notes

- The low analog trigger voltage is used for falling-edge triggering, the hysteresis of rising edge triggering, and window triggering.
- Changing the analog trigger voltage will cancel any acquisition currently in progress on the capture card.
- The actual quantized trigger voltage will be returned
- Changing the gain setting or trigger source may cause the the trigger level to be re-quantized to match the new setting.

See also

- das4020capture:HWTRIGSRC (pg. 109)
- das4020capture:ATRIGHIGH (pg. 100)
- das4020capture:ATRIGMODE (pg. 102)

das4020capture:ATRIGMODE

Syntax

```
das4020capture:ATRIGMODE <trigger mode>
               ATRIGMODE? <trigger mode>
```

Description

Specify or query the analog trigger slope/mode.

Parameters

<trigger mode>	Desired trigger mode

Notes

Valid trigger modes are:

POS_HIST Trigger on analog voltage rising above ATRIGHIGH voltage with ATRIGLOW hysteresis (i.e. Schmitt trigger)

- **NEG_HIST** Trigger on analog voltage falling below ATRIGLOW voltage with ATRIGHIGH hysteresis (i.e. Schmitt trigger)
- **POS_SLOPE** Trigger on analog voltage rising above ATRIGHIGH voltage.
- **NEG_SLOPE** Trigger on analog voltage falling below ATRIGLOW voltage.
- **WINDOW** Trigger on analog voltage between ATRIGLOW and ATRIGHIGH voltages.

Changing the analog trigger mode will cancel any acquisition currently in progress on the capture card.

See also

- • das4020capture:HWTRIGSRC (pg. 109)
  - das4020capture:ATRIGHIGH (pg. 100)
  - das4020capture:ATRIGLOW (pg. 101)

das4020capture:CALCSYNC

Syntax

```
das4020capture:CALCSYNC <sync_enabled>
               CALCSYNC? <sync_enabled>
```

Description

Set whether new acquisitions should be inhibited until computation from the previous acquisition is complete

Parameters

<sync_enabled>	Whether new acquisitions should be inhibited, `true` or `false`

Notes

- Effective following the next trigger. Computations currently in progress will not be waited for before allowing a trigger.

See also

das4020capture:CLKSRC

Syntax

```
das4020capture:CLKSRC <clock source> Hz
               CLKSRC?
```

Description

Specify or query the clock source for waveform capture A/D

Parameters

<**clock source**>	Desired clock source

Notes

- <**clock source**> may be `INTERNAL`, `EXTBNC`, or `ADSTARTTRIG` to select the internal 40 MHz source, the bottom BNC connector, or the A/D Start Trig pin on the IDC header, respectively.
- Changing the clock source will cancel any acquisition currently in progress on the capture card.
- The card is designed for a maximum clock rate of 40 MHz.
- Internal limitations of the card require divisors of at least 2 (1 or 2 channels) or 4 (4 channels), so depending on the number of channels the maximum sample rate will be 1/2 or 1/4 of the frequency of the external clock.
- Use das4020capture:HWFREQ to set the divisor and das4020capture:FREQ to set the actual divided clock frequency when not in `INTERNAL` mode.

See also

- das4020capture:SAMPLECNT (pg. 113)
- das4020capture:NUMCHANNELS (pg. 110)
- das4020capture:FREQ (pg. 107)
- das4020capture:HWFREQ (pg. 108)

das4020capture:DSFACTOR

Syntax

```
das4020capture:CH<i>:DSFACTOR <factor>
               CH<i>:DSFACTOR?
```

Description

Specify or query the downsampling factor for channel <**i**>

Parameters

<**factor**>	Desired downsampling factor.

Notes

- If the downsampling factor is d, only one of every d samples will be recorded, and the time step will be d times the actual sample period.
- This is useful primarily for waveforms that are slow changing, or if due to unavoidable clocking restraints it is necessary to oversample.

See also

- das4020capture:SAMPLECNT (pg. 113)
- das4020capture:NUMCHANNELS (pg. 110)
- das4020capture:FREQ (pg. 107)
- das4020capture:HWFREQ (pg. 108)

das4020capture:FIFOSIZE

Syntax

```
das4020capture:FIFOSIZE <fifo size>
               FIFOSIZE? <fifo size>
```

Description

Specify or query the amount of the DAS internal FIFO to use.

Parameters

<fifo size>	Desired FIFO size, in 24-bit words

Notes

- The default value of 32768 corresponds to 32768 24 bit words in each of the two (X and Y) FIFOs, for a memory-equivalent of 256 kilobytes.
- This may be set lower to avoid conflicts with other devices that have smaller FIFO's. By reducing the FIFO size, data transmissions are sent in smaller chunks that are less likely to overflow the FIFO of another device that is writing data to memory at the same time.
- Only powers of 2 starting at 256 and going up to 32768 are permitted

See also

- das4020capture:SAMPLECNT (pg. 113)
- das4020capture:NUMCHANNELS (pg. 110)

das4020capture:FREQ

Syntax

```
das4020capture:FREQ <sample frequency> Hz
               FREQ? <sample frequency> Hz
```

Description

Specify or query the number of samples to acquire per second

Parameters

<sample frequency>	Desired sample rate (default Hz)

Notes

- Changing the sample frequency will cancel any acquisition currently in progress on the capture card.
- Only integer fractions of 20 MHz are permitted. When specifying the sample frequency, check the response from the server to determine the actual sample rate.
- The maximum sample rate is 20 MHz (1 or 2 channels) or 10 MHz (4 channels)
- If CLKSRC is not INTERNAL then this must be set to the actual divided clock frequency. In that case the clock devision is determined by das4020capture:HWFREQ.

See also

- das4020capture:SAMPLECNT (pg. 113)
- das4020capture:NUMCHANNELS (pg. 110)
- das4020capture:HWFREQ (pg. 108)

das4020capture:HWFREQ

Syntax

```
das4020capture:HWFREQ <sample frequency> Hz
               HWFREQ? <sample frequency> Hz
```

Description

Specify or query the capture frequency to program the DAS4020 card with

Parameters

<sample frequency>	Desired capture frequency

Notes

- Changing the sample frequency will cancel any acquisition currently in progress on the capture card.
- Only integer fractions of 20 MHz are permitted. When specifying the sample frequency, check the response from the server to determine the actual sample rate.
- The maximum HWFREQ is 20 MHz (1 or 2 channels) or 10 MHz (4 channels)
- If CLKSRC is INTERNAL, this will not be adjustable and will track das4020capture:FREQ.
- If CLKSRC is not INTERNAL then this is used to determine the clock division of the external clock. Program this with the frequency you would use to get the desired division if the external clock were 40 MHz. For example, to get a division of 4, program this to 10 MHz. To get a division of 12, program this to 3.33 MHz.

See also

- das4020capture:SAMPLECNT (pg. 113)
- das4020capture:NUMCHANNELS (pg. 110)
- das4020capture:HWFREQ (pg. 108)

das4020capture:HWTRIGSRC

Syntax

```
das4020capture:HWTRIGSRC <trigger source>
               HWTRIGSRC? <trigger source>
```

Description

Specify or query the trigger source

Parameters

<trigger source>	Desired trigger source: "EXT" for the bottom BNC connector on the card or "INT" for the trigger line on the 40 pin header, or "CH1" through "CH4" for triggering from the analog input channels.

Notes

- Changing the trigger source will cancel any acquisition currently in progress on the capture card.

See also

- das4020capture:ATRIGMODE (pg. 102)

das4020capture:NUMCHANNELS

Syntax

```
das4020capture:NUMCHANNELS <number of channels>
               NUMCHANNELS? <number of channels>
```

Description

Specify or query the number of channels to acquire

Parameters

<number of channels>	Desired number of channels

Notes

- Changing the number of channels will cancel any acquisition currently in progress on the capture card.
- 1, 2, or 4 channels are allowed.
- A sample rate of 20MHz requires 2 or fewer channels

See also

- das4020capture:SAMPLECNT (pg. 113)
- das4020capture:FREQ (pg. 107)

das4020capture:CHi:PROBEATTEN

Syntax

```
das4020capture:CH<i>:PROBEATTEN <attenuation factor>
                CH<i>:PROBEATTEN? <attenuation factor>
```

Description

Specify or query the attenuation factor of the probe attached to channel <**i**>.

Parameters

<**i**>	Channel number: 1-4
<**attenuation factor**>	Desired attenuation factor (default 1.0)

Notes

See also

- das4020capture:CHi:RANGE (pg. 112)

das4020capture:CHi:RANGE

Syntax

```
das4020capture:CH<i>:RANGE <input range>
               CH<i>:RANGE? <input range>
```

Description

Specify or query input gain/attenuation setting for the capture card

Parameters

<i>	Channel number: 1-4
<input range>	Desired input range: 1V or 5V

Notes

- Changing the input range will cancel any acquisition currently in progress on the capture card.
- Current (as of Jan 2006) versions of the driver do not correctly handle the case of different gain settings on different channels. Hopefully this will be fixed eventually. In the mean time, be sure to use the same setting on all four channels.

See also

- das4020capture:CHi:PROBEATTEN (pg. 111)

das4020capture:SAMPLECNT

Syntax

```
das4020capture:SAMPLECNT <number of samples>
               SAMPLECNT? <number of samples>
```

Description

Specify or query the number of samples to acquire per waveform

Parameters

<number of samples>	Number of samples to acquire

Notes

- Changing the number of samples will cancel any acquisition currently in progress on the capture card.
- The maximum permissible SAMPLECNT is determined when the das4020 driver is compiled by the symbol ADC_BUFF_PHY_SIZE specified in a2dc.h. To increase ADC_BUFF_PHY_SIZE you must change that parameter and recompile and reinstall the driver.

See also

- das4020capture:FREQ (pg. 107)

7.6 edtcapture.so

edtcapture.so provides image capture with the EDT PCI-DV CLINK CameraLink framegrabber (http://www.edt.com) and possibly other capture cards supported by the EDT PCI-DV driver.

To use this module, you must have the EDT PCI-DV driver installed in /usr/local/EDTpdv or /opt/EDTpdv. The edtinit shell script (provided with the driver) should be run on system boot, and you need may in addition need to run the initcam program to complete initialization of the driver before using this module (e.g. `/usr/local/EDTpdv/initcam -f /usr/local/EDTpdv/camera_config/sc6000.cfg`). Also be sure that /dev/pdv0 has appropriate permissions.

7.6.1 Prerequisites

- library wfmstore.so
- library metadata.so
- library rpc.so

7.6.2 Configuration parameters

Parameter	Type	Value
devname	quoted string	PDV device name for pdv_open() (default is “pdv”).
unit	unsigned integer	Unit number for pdv_open() (default is 0)
channel	unsigned integer	Channel number for pdv_open() (default is 0)
width	unsigned integer	Width of camera image, in pixels (default is 640)
height	unsigned integer	Height of camera image, in pixels (default is 512)
numbufs	unsigned integer	Size of the ring buffer in frames (default is 10)
channelname	quoted string	Name of the channel to contain the images (default is “EDT”)
calcsync	boolean (true or false)	If true, don’t allow new acquisitions until computations resulting from the previous acquisition are complete. (default is false)
discardtopline	boolean	If true, always ask the EDT library for one extra line, then discard the first line of each image capture

Edtcapture also supports the standard metadata initialization configuration parameters `setstaticmetadatum` and `addquerymetadatum`. The metadata provided by these methods is attached to the captured waveforms.

7.6.3 Commands

In addition to the commands described on the next pages, edtcapture also supports the standard metadata specification parameters `setstaticmetadatum`, `addquerymetadatum`, and `delmetadatum`. The metadata provided by these methods is attached to the captured images.

(commands begin on next page)

edtcapture:CALCSYNC

Syntax

```
edtcapture:CALCSYNC <sync_enabled>
           CALCSYNC? <sync_enabled>
```

Description

Set whether new acquisitions should be inhibited until computation from the previous acquisition is complete

Parameters

<**sync_enabled**>	Whether new acquisitions should be inhibited, `true` or `false`

Notes

- Enabling or disabling CALCSYNC will cancel any acquisition currently in progress on the framegrabber.

See also

edtcapture:GEOMETRY

Syntax

```
edtcapture:GEOMETRY <width>*<height>
          GEOMETRY? <width>*<height>
```

Description

Specify or query the image size expected from the camera.

Parameters

<**width**>	Width of the image, in pixels.
<**height**>	Height of the image, in pixels.

Notes

- Changing the geometry will cancel any acquisition currently in progress on the framegrabber.

See also

7.7 hp34401_thermistor.so

hp34401_thermistor.so transforms a thermistor and HP 34401A benchtop multimeter into a temperature meter. The computer sets the multimeter to resistance mode, transforms the resistance to temperature using a lookup table and linear interpolation, and displays the temperature on the hp34401 display. The temperature can also be read out through the dataguzzler command interface.

7.7.1 Prerequisites

- library multiio.so

7.7.2 Configuration parameters

Parameter	Type	Value
uri	quoted string	multiio URI for HP 34401A. Use URI `''serial:///dev/ttyS0:9600:7:e:2''` for HP34401A serial or `''gpib://0:8:0:0:10''` for HP34401A gpib address 8
rt_priority	unsigned integer	POSIX real time priority of data collection thread (default 0)
inttime	real number	Measurement integration time in power line cycles. Allowed values are .02, .2, 1, 10, or 100. The longer the inttime, the longer the GPIB bus is tied up by the measurement thread waiting for the voltmeter.
timer_period	real number	Desired time between measurments, in seconds.
thermistor_calibration_in_celsius	boolean (true or false)	Set to true if the temperatures in the calibration table are in deg. C. Set to false if the temperatures are in deg. K.
thermistor_calibration	array of real numbers	Thermistor calibration data. Should consist of alternating values of resistance (Ohms) and temperature (deg. K or C), both of which should be monotonic.
timeoutms	integer	communications timeout, in milliseconds. Negative indicates wait forever

7.7.3 Commands

(commands begin on next page)

hp34401thermistor:TEMP

Syntax

```
hp34401thermistor:TEMP? <temperature> C
```

Description

Query the last temperature measured by the termistor.

Parameters

<temperature>	The measured temperature in degrees Celsius.

Notes

See also

- hp34401thermistor:TEMPREV (pg. 120)

hp34401thermistor:TEMPREV

Syntax

hp34401thermistor:TEMPREV? <**temperature**> C <**revision**> <**timestamp_sec**>
<**timestamp_nsec**>

Description

Query the last temperature measured by the termistor.

Parameters

<**temperature**>	The measured temperature in degrees Celsius.
<**revision**>	Revision count of temperature.
<**timestamp_sec**>	Seconds component of the timestamp.
<**timestamp_nsec**>	Nanoseconds component of the timestamp.

Notes

See also

- hp34401thermistor:TEMP (pg. 119)
- hp34401thermistor:WAITTEMPREV (pg. 121)

hp34401thermistor:WAITTEMPREV

Syntax

```
hp34401thermistor:WAITTEMPREV? <revision> WAITTEMPREV <temperature> C
                  <revision> <timestamp_sec> <timestamp_nsec>
```

Description

Wait for the specified revision, then query the last temperature measured by the termistor.

Parameters

<temperature>	The measured temperature in degrees Celsius.
<revision>	Revision count of temperature.
<timestamp_sec>	Seconds component of the timestamp.
<timestamp_nsec>	Nanoseconds component of the timestamp.

Notes

- The first syntax above is the format of the query, the second is the format of the response
- The obtained revision may be later than the requested revision.

See also

- hp34401thermistor:TEMP (pg. 119)
- hp34401thermistor:TEMPREV (pg. 120)

7.8 agilent33x20awg.so

agilent33x20awg.so uses an Agilent 33220A (or an older HP/Agilent 33120A) function generator as an arbitrary waveform generator. It places the 33220A in arbitrary waveform burst mode, and uploads a named wavefrom from wfmstore.so as the arbitrary waveform. Note that the named waveform should have no more samples than the amount of storage in the function generator (see agilent33x20awg:NUMPOINTS (pg. 126)). The named waveform should go between voltages of +1V and -1V.

7.8.1 Prerequisites

- library multiio.so
- library wfmstore.so

7.8.2 Configuration parameters

Parameter	Type	Value
uri	quoted string	multiio URI for function generator. Use URI `serial:///dev/ttyS0:9600:8:n:2` for HP33120a serial or ``` ``gpib://0:10:0:0:10'' ``` for 33120A or 33220A gpib address 10 decimal or `tcp://fcngen:5025` for agilent 33220a tcp/ip on host fcngen.
arbchan	channel name	Name of wfmstore.so channel to use as source to upload to waveform generator (default ARB)
ampl	voltage (Volts)	The desired peak-to-peak amplitude (into 50 ohms) (default 100 mV).
offset	voltage (Volts)	Desired voltage offset (into 50 ohms) (default 0 V).
ncyc	integer	desired number of cycles of the arbitrary waveform per trigger
trigsrc	BUS, EXT, or IMM	desired trigger source: BUS (or trigger button on front), EXTernal connector on rear, or IMMedate automatic triggering.
output	ON or OFF	Desired initial output state: ON or OFF
lockout	TRUE or FALSE	should the instrument front panel (excluding the LOCAL button) be locked out?

7.8.3 Commands

(commands begin on next page)

agilent33x20awg:AMPL

Syntax

```
agilent33x20awg:AMPL <voltage amplitude> V
                AMPL? <voltage amplitude> V
```

Description

Specify or query the arbitrary waveform generator peak-to-peak output voltage.

Parameters

<Voltage amplitude>	The desired amplitude (Volts)

Notes

- This assumes the arbitrary waveform (see agilent33x20awg:ARBCHAN) has minimum value -1.0 and maximum value 1.0.

See also

- agilent33x20awg:ARBCHAN (pg. 124)
- agilent33x20awg:OFFSET (pg. 127)

agilent33x20awg:ARBCHAN

Syntax

```
agilent33x20awg:ARBCHAN <channel name>
                ARBCHAN? <channel name>
```

Description

Specify or query the channel used to upload to the arbitrary waveform generator.

Parameters

<channel name >	Name of the wfmstore.so channel.

Notes

See also

agilent33x20awg:NCYC

Syntax

```
agilent33x20awg:NCYC <number of cycles>
                NCYC? <number of cycles>
```

Description

Specify or query the number of arbitrary waveform cycles per trigger.

Parameters

<number of cycles>	The desired number of cycles

Notes

See also

`agilent33x20awg:NUMPOINTS`

Syntax

`agilent33x20awg:NUMPOINTS?` <**number of points**>

Description

Query the function generator memory size.

Parameters

<**number of points**>	Function generator memory size

Notes

- 16000 for HP33120, 65536 for Agilent 33220A.
- This module will refuse to upload waveforms longer than the function generator memory size. Waveforms shorter than the function generator memory size will be zero-padded.

See also

agilent33x20awg:OFFSET

Syntax

```
agilent33x20awg:OFFSET <voltage offset> V
                OFFSET? <voltage offset> V
```

Description

Specify or query the arbitrary waveform generator output voltage offset.

Parameters

<**Voltage offset**>	The desired offset (Volts)

Notes

See also

- agilent33x20awg:AMPL (pg. 123)

agilent33x20awg:OUTPUT

Syntax

```
agilent33x20awg:OUTPUT <output status>
                OUTPUT? <output status>
```

Description

Set or query whether the function generator output is enabled (Agilent 33220A only).

Parameters

<output status>	The output status: ON or OFF

Notes

See also

agilent33x20awg:SAMPLERATE

Syntax

`agilent33x20awg:SAMPLERATE?` <**sample rate**>

Description

Query the function generator hardware sample rate.

Parameters

<**sample rate**>	Sample rate in Hz

Notes

- 40 MHz for HP33120, 50 MHz for Agilent 33220A.
- For best results, the arbitrary waveform should use a sample rate that is a factor of the function generator hardware sample rate.

See also

agilent33x20awg:TRIGSRC

Syntax

```
agilent33x20awg:TRIGSRC <trigger source>
                TRIGSRC? <trigger source>
```

Description

Specify or query the function generator trigger source.

Parameters

<trigger source>	BUS (Button on console), EXTernal trigger, or IMMediate automatic trigger.

Notes

See also

7.9 tabor5061awg.so

tabor5061awg.so controls a Tabor WW5061 as an arbitrary waveform generator. It places the ww5061 in arbitrary waveform burst mode, and uploads a named wavefrom from wfmstore.so as the arbitrary waveform. Note that the named waveform should have no more samples than the amount of storage in the function generator (see tabor5061awg:NUMPOINTS (pg. 137)). The named waveform should go between voltages of +1V and -1V.

Please note that the 5061 firmware (v1.61 as of this writing) is rather buggy. GPIB waveform upload/download does not seem to work properly for unknown reasons, and despite the documentation the instrument does not support waveform upload/download in SCPI over TCP. Instead the instrument switches to a proprietary (and buggy) UDP protocol that doesn't deal with all of the possibilities for dropped packets (hence there's a chance the upload could fail). Also please note to enable the UDP transfers any software or hardware firewall between the computer and the WW5061 must allow UDP packets from the instrument on port 7501. For example (using Linux iptables):

```
iptables -A INPUT -i eth1 -p udp -m udp --dport 7501 -j ACCEPT
```

7.9.1 Prerequisites

- library multiio.so
- library wfmstore.so

7.9.2 Configuration parameters

Parameter	Type	Value
uri	quoted string	multiio URI for function generator. Use URI ``tcp://hostname:23`` tcp/ip on host hostname. GPIB should work also, but currently firmware bugs in the instrument seem to cause it to fail during the waveform upload process.
arbchan	channel name	Name of wfmstore.so channel to use as source to upload to waveform generator (default ARB)
ampl	voltage (Volts)	The desired peak-to-peak amplitude (into 50 ohms) (default 100 mV).
offset	voltage (Volts)	Desired voltage offset (into 50 ohms) (default 0 V).
ncyc	integer	desired number of cycles of the arbitrary waveform per trigger
output	ON or OFF	Desired initial output state: ON or OFF
memsize	integer	Installed memory (max waveform storage) size, in bytes, default 1000000
extclk	boolean	Enable external clock input (SMA connector)
extclkfreq	frequency (Hz)	The initial frequency of the external clock
udphostname	quoted string	the function generator host name (should be the same as in the TCP URI) for UDP waveform upload/download
udplisteninterface	quoted string	IP address or host name of ethernet interface for listening for UDP data from the function generator. If not specified, will listen on all interfaces.

7.9.3 Commands

(commands begin on next page)

tabor5061awg:AMPL

Syntax

```
tabor5061awg:AMPL <voltage amplitude> V
             AMPL? <voltage amplitude> V
```

Description

Specify or query the arbitrary waveform generator peak-to-peak output voltage.

Parameters

<Voltage amplitude>	The desired amplitude (Volts)

Notes

- This assumes the arbitrary waveform (see tabor5061awg:ARBCHAN) has minimum value -1.0 and maximum value 1.0.

See also

- tabor5061awg:ARBCHAN (pg. 134)
- tabor5061awg:OFFSET (pg. 138)

`tabor5061awg:ARBCHAN`

Syntax

```
tabor5061awg:ARBCHAN <channel name>
             ARBCHAN? <channel name>
```

Description

Specify or query the channel used to upload to the arbitrary waveform generator.

Parameters

<**channel name** >	Name of the wfmstore.so channel.

Notes

See also

tabor5061awg:EXTCLKFREQ

Syntax

```
tabor5061awg:EXTCLKFREQ <frequency>
             EXTCLKFREQ DISABLED
             EXTCLKFREQ?
```

Description

Set or query the currently assumed external clock frequency.

Parameters

<frequency>	The currently assumed external clock frequency (default Hz)

Notes

- Changing the external clock frequency triggers a new waveform upload. But you need to update the waveform first. The best sequence is:
  1. Set tabor5061awg:ARBCHAN to a nonexistent/empty waveform.
  2. Adjust the hardware frequency source for the external clock.
  3. Issue tabor5061awg:EXTCLKFREQ with the new clock frequency
  4. Update the correct channel with a waveform with the new clock frequency
  5. Set tabor5061awg:ARBCHAN back to the correct channel
- tabor5061awg:EXTCLKFREQ DISABLED disables use of the external clock.
- When no external clock is in use, the waveform generator generates an internal clock based on the sample period from tabor5061awg:ARBCHAN

See also

`tabor5061awg:NCYC`

Syntax

```
tabor5061awg:NCYC <number of cycles>
             NCYC? <number of cycles>
```

Description

Specify or query the number of arbitrary waveform cycles per trigger.

Parameters

<number of cycles>	The desired number of cycles

Notes

See also

`tabor5061awg:NUMPOINTS`

Syntax

`tabor5061awg:NUMPOINTS?` <**number of points**>

Description

Query the function generator memory size.

Parameters

<**number of points**>	Function generator memory size

Notes

- 524288 or 1048576, depending on model (must be properly set in config file).
- This module will refuse to upload waveforms longer than the function generator memory size.

See also

tabor5061awg:OFFSET

Syntax

```
tabor5061awg:OFFSET <voltage offset> V
             OFFSET? <voltage offset> V
```

Description

Specify or query the arbitrary waveform generator output voltage offset.

Parameters

<Voltage offset>	The desired offset (Volts)

Notes

See also

- tabor5061awg:AMPL (pg. 133)

tabor5061awg:OUTPUT

Syntax

```
tabor5061awg:OUTPUT <output status>
             OUTPUT? <output status>
```

Description

Set or query whether the function generator output is enabled.

Parameters

<output status>	The output status: ON or OFF

Notes

See also

7.10 wfmgen.so

wfmgen.so creates custom-specified burst, sweep, or noise waveforms. If the wfmioname is specified, wfmgen donates ownership of created waveforms/channels to the specified WFMIO-compatible module. Otherwise it maintains ownership itself and reports a list with every SET query, regenerates every waveform with every SET command, and regenerates every waveform if DT is changed.

7.10.1 Prerequisites

- library rpc.so
- library wfmstore.so

7.10.2 Configuration parameters

Parameter	Type	Value
wfmioname	quoted string	name of the wfmio module (gets ownership of the generated waveforms), default WFM (must be within the same process!)
dt	time (sec)	step time for the generated waveforms (default 1 μs)
npoints	unsigned integer	number of points per waveform (default 65536)
timedelay	time (seconds)	time delay before first point. (default 0)
ampl	voltage (Volts)	amplitude of waveforms to generate (half the desired peak-to-peak amplitude) (default 0.5 V)

7.10.3 Commands

(commands begin on next page)

`wfmgen:BURST`

Syntax

`wfmgen:BURST` <**waveform name**> <**f**> <**t0**> <**t1**> <**t2**> <**t3**>

Description

Create a waveform containing a modulated tone burst

Parameters

<**waveform name**>	Name of the waveform to create
<**f**>	Frequency of the burst
<**t0**>**,**<**t1**>**,**<**t2**>**,**<**t3**>	Parameters of the raised-cosine envelope

Notes

- The zero-phase point is t=0
- See wfmgen:WINDOW for information on the raised-cosine envelope parameters.

See also

- wfmgen:WINDOW (pg. 149)

wfmgen:DT

Syntax

```
wfmgen:DT <time_step>
       DT?
```

Description

Set or query the time step used for waveform creation

Parameters

<time_step>	The time step (default seconds)

Notes

- If the wfmgen:module is programmed to store its own waveforms (the wfmioname configuration parameter is not set), then changing DT will cause the module to regenerate all of its own waveforms with the new time step.

See also

- wfmgen:TIMEDELAY (pg. 148)

`wfmgen:FUNC`

Syntax

`wfmgen:FUNC?` <**waveform name**>

Description

Obtain the specification of waveform <**waveform name**>

Parameters

<**waveform name**>	Name of preexisting wfmgen: waveform.

Notes

- The response is in the form wfmgen:<**WFMGENCMD**> <**waveform name**> <**parameters. . .** >

See also

wfmgen:GAUSSIAN

Syntax

wfmgen:GAUSSIAN <**waveform name**> <**t**> <**width**>

Description

Create a waveform containing a Gaussian pulse

Parameters

<**waveform name**>	Name of the waveform to create
<**t0**>	Time of the center of the pulse
<**width**>	Width of the pulse, in units of time.

Notes

- The waveform is truncated to the time range specified in the module configuration
- The function generated is: $exp(-\frac{t-t_0}{2w^2})$.

See also

- wfmgen:SINC (pg. 145)

wfmgen:SINC

Syntax

`wfmgen:SINC` <**waveform name**> <**t**> <**width**>

Description

Create a waveform containing a sinc pulse

Parameters

<**waveform name**>	Name of the waveform to create
<**t0**>	Time of the center of the pulse
<**width**>	Width of the pulse, in units of time.

Notes

- The waveform is truncated to the time range specified in the module configuration
- The function generated is: $\frac{sin(\pi(t-t_0)/w)}{\pi(t-t_0)/w}$.

See also

- wfmgen:GAUSSIAN (pg. 144)

wfmgen:SWEEP

Syntax

`wfmgen:SWEEP` <**waveform name**> <**f1**> <**f2**> <**t0**> <**t1**> <**t2**> <**t3**>

Description

Create a waveform containing a frequency sweep (chirp)

Parameters

<**waveform name**>	Name of the waveform to create
<**f0**>	Starting frequency
<**f1**>	Ending frequency
<**t0**>,<**t1**>,<**t2**>,<**t3**>	Parameters of the raised-cosine envelope

Notes

- The reference phase is -PI/2 at t=t0
- See wfmgen:WINDOW for information on the raised-cosine envelope parameters.
- The waveform is zero for t < t0. It increases to the configured amplitude at t1 following a raised-cosine envelope and stays at the configured amplitude until t2. Then it decreases following a raised cosine envelope to zero at and after t3

See also

- wfmgen:WINDOW (pg. 149)
- wfmgen:SWEEPENVELOPE (pg. 147)

wfmgen:SWEEPENVELOPE

Syntax

`wfmgen:SWEEPENVELOPE` <**waveform name**> <**f1**> <**f2**> <**t0**> <**t1**> <**t2**>
<**t3**> <**envt0**> <**envV0**> <**envt1**> <**envV0**> ...

Description

Create a waveform containing a frequency sweep (chirp), with an extra multiplicative envelope defined by an arbitrary series of (t,V) pairs.

Parameters

<**waveform name**>	Name of the waveform to create
<**f0**>	Starting frequency
<**f1**>	Ending frequency
<**t0**>,<**t1**>,<**t2**>,<**t3**>	Parameters of the raised-cosine envelope
<**envt**<**i**>>, <**envV**<**i**>>	Time and voltage pair #<**i**> of the extra linear interpolation envelope

Notes

- The extra multiplicative envelope is multiplied by the raised cosine envelope defined by <**t0**>, <**t1**>, <**t2**>, and <**t3**>
- The reference phase is -PI/2 at t=t0
- See wfmgen:WINDOW for information on the raised-cosine envelope parameters.
- The waveform is zero for t < t0. It increases to the configured amplitude at t1 following a raised-cosine envelope and stays at the configured amplitude until t2. Then it decreases following a raised cosine envelope to zero at and after t3
- The extra envelope is defined by a series of (time, voltage) pairs following <**t3**>. Voltage at intermediate times will be linearly interpolated. Units are assumed to be seconds and volts by default, respectively.

See also

- wfmgen:WINDOW (pg. 149)
- wfmgen:SWEEP (pg. 146)

wfmgen:TIMEDELAY

Syntax

```
wfmgen:TIMEDELAY <time_delay>
       TIMEDELAY?
```

Description

Set or query the time of the first sample generated in created waveforms

Parameters

<time_delay>	The time delay of the first sample (default seconds)

Notes

- If the wfmgen:module is programmed to store its own waveforms (the wfmioname configuration parameter is not set), then changing TIMEDELAY will cause the module to regenerate all of its own waveforms with the new time delay.

See also

- wfmgen:DT (pg. 142)

wfmgen:WINDOW

Syntax

`wfmgen:WINDOW` <**waveform name**> <**t0**> <**t1**> <**t2**> <**t3**>

Description

Create a waveform containing a split raised-cosine window

Parameters

<**waveform name**>	Name of the waveform to create
<**t0**>,<**t1**>,<**t2**>,<**t3**>	Parameters of the raised-cosine envelope

Notes

- The waveform is zero for t ¡ t0. It increases to the configured amplitude at t1 following a raised-cosine envelope and stays at the configured amplitude until t2. Then it decreases following a raised cosine envelope to zero at and after t3

See also

7.11 polytecvibrometer.so

polytecvibrometer.so supports laser vibrometers based on the PolyTec OFV-5000 vibrometer controller. As currently implemented it operates exclusively using the first velocity decoder module inside the OFV-5000. It has two main functions: (Focus monitoring and control), and (gain adjustment and waveform capture interfacing). It can supply a "Probe Attenuation" and appropriate metadata to a waveform capture driver so as to make the acquired waveforms appear in units of doppler velocity. It also can extract the vibrometer delay and use that to apply a temporal offset the acquired velocity waveforms. If the probe attenuation and static metadata configuration parameters are supplied then this functionality will be enabled. Note that whatever module(s) those commands refer to should be initialized prior to polytecvibrometer.so in the configuration file.

7.11.1 Prerequisites

- library rpc.so
- library multiio.so

7.11.2 Configuration parameters

Parameter	Type	Value
uri	quoted string	URI for communicating with the laser vibrometer, typically serial:///dev/ttyS0:57600:8:n:1:nortscts (default none)
staticmetadatacmd	quoted string	command for setting static metadata of the waveform capture device, e.g. WCAPT:CH4:SETSTATICMETADATUM (default none)
probeattencmd	quoted string	command for setting the probe attenuation of the waveform capture device, e.g. WCAPT:CH4:PROBEATTEN (default none)
probeattenfactor	real number	additional probe attenuation scale factor, to account additional attenuation on the waveform capture input, for example
timeoutms	integer	communications timeout, in ms. (default 3000)
lockout	boolean	whether to lockout the instrument console (default false)
range	real number	initial range setting, in mm/s/V

7.11.3 Commands

(commands begin on next page)

polytecvibrometer:AUTOFOCUS

Syntax

```
polytecvibrometer:AUTOFOCUS
                  AUTOFOCUS? <autofocus status>
```

Description

Initiate an autofocus or query the autofocus status.

Parameters

<**autofocus status**>	“FOCUSING”, “FOCUSED”, or “FAILED”

Notes

- The parameter to the query format illustrates the format of the response to a query.

See also

- polytecvibrometer:WAITAUTOFOCUS (pg. 157)

polytecvibrometer:DELAY

Syntax

```
polytecvibrometer:DELAY?
                  DELAY <delay> us
```

Description

Query the vibrometer's inherent delay for the current sensitivity setting.

Parameters

<**delay**>	Delay induced by the vibrometer, in microseconds

Notes

- The command format given above is provided to illustrate the format of the response to a query.
- The delay is a function of the range setting, polytecvibrometer:RANGE.

See also

- polytecvibrometer:RANGE (pg. 155)

polytecvibrometer:FOCUSPOSITION

Syntax

```
polytecvibrometer:FOCUSPOSITION?
                  FOCUSPOSITION <position>%
```

Description

Query the vibrometer's focus position.

Parameters

<position>	The position response from the vibrometer.

Notes

- The command format given above is provided to illustrate the format of the response to a query.

See also

- polytecvibrometer:AUTOFOCUS (pg. 152)

polytecvibrometer:RANGE

Syntax

```
polytecvibrometer:RANGE <range>
                  RANGE?
```

Description

Set or query the vibrometer sensitivity setting.

Parameters

<range>	One over the sensitivity, in mm/s/V

Notes

- The units of mm/s/V may optionally be supplied
- If the staticmetadatacmd and/or probeattencmd parameters are set, this will trigger adjustment of the probe attenuation, delay, etc. as appropriate
- The allowable range settings depend on the velocity decoder in use. This routine asks the contoller for the list of range settings and picks the closest one to the requested value. The requested value is returned.
- Different ranges may have different inherent delays (see polytecvibrometer:DELAY).
- Different ranges may have different analog bandwidths (see the user manual).

See also

- polytecvibrometer:DELAY (pg. 153)

polytecvibrometer:SIGNALLEVEL

Syntax

```
polytecvibrometer:SIGNALLEVEL?
                  SIGNALLEVEL <percent>%
```

Description

Query the vibrometer's optical signal level.

Parameters

<percent>	Percent of maximum signal level (may be logarithmic).

Notes

- The command format given above is provided to illustrate the format of the response to a query.
- The signal level depends dramatically on the quality of the focus.

See also

- polytecvibrometer:AUTOFOCUS (pg. 152)

polytecvibrometer:WAITAUTOFOCUS

Syntax

```
polytecvibrometer:WAITAUTOFOCUS
                  WAITAUTOFOCUS <autofocus status>
```

Description

Wait for an autofocus to complete.

Parameters

<autofocus status>	“FOCUSED”, or “FAILED”

Notes

- The parameter in the second syntax line illustrates the format of the response when focusing is complete.
- An autofocus should have been initated with polytecvibrometer:AUTOFOCUS.

See also

- polytecvibrometer:AUTOFOCUS (pg. 152)

7.12 pyscript.so

pyscript.so provides a mechanism for Python-based scripting. The initialization block consists of two brace-delimited sets of python code. The first such set is for module initialization. It executes in the context of a newly created python module with the same name as the dataguzzler module, and usually contains import statements and variable and function definitions. The special global variable pys_modname is initialized to the name of the script module (capitalized). Please note that the first set of code is not permitted to call dg_python.rpc_async(). It may, however, check for existance of other modules with dg_python.findmodule(). If the initialization code sets the global variable `disabled` to `True`, then module initialization will be cancelled and the module will not be available.

The second set of python code is executed when this module is called. The variable "cmd" is initialized to the command to be parsed. You can use sys.stdout.write() (avoid the print statement) to send a single line of output to the client (never transmit a linefeed!). Global variables can be read directly from the python code, but modifying them requires declaring them with a global statement. A special variable 'retcode' is pre-initialized to 200, and can be used to set an alternative return code for the script. The script will execute atomically within the dataguzzler kernel context, except for calls to `dg_python.rpc_async()` that require waiting, which allow other kernel commands to be processed and execute, until the call is complete.

If the connection dies, rpc_async() will throw an exception and additional attempts to call rpc_async() will fail. The script should end at this point.

7.12.1 Prerequisites

- library rpc.so
- library dg_python.so

7.12.2 Configuration parameters

(The configuration section is a series of two python scripts, each enclosed in braces and indented. The first is executed on module initialization. The second is executed when the module is called. Neither script section may be empty. If no code is to be provided, give the python statement "pass".)

7.12.3 Commands

(none)

7.13 time.so

time.so provides support for waiting specified amounts of time, and may provide further timing functionality in the future. Currently, its primary purpose is to allow python script modules (pyscript.so) to wait for a specified amount of time while allowing the dataguzzler kernel to continue executing.

7.13.1 Prerequisites

(none)

7.13.2 Configuration parameters

(none)

7.13.3 Commands

(commands begin on next page)

time:DELAY

Syntax

```
time:DELAY <time>
     DELAY?
```

Description

Wait for a specified amount of time.

Parameters

<time>	The amount of time to wait, in units of time (default seconds)

Notes

- This command does not occupy the dataguzzler kernel. Commands issued from other connections can execute during the wait.

See also

time:TIMESTAMP

Syntax

```
time:TIMESTAMP?
```

Description

Obtain a timestamp

Parameters

Notes

- The form of the result is a single quoted string, e.g. ”2007-06-27T18:43:59-0500”. This is believed to be ISO-8601 compliant.

See also

7.14 simpletrigger.so

simpletrigger.so provides support for simple trigger generation using a digital output

7.14.1 Prerequisites

- Library dio8bit.so
- Digital I/O driver library.

7.14.2 Configuration parameters

Parameter	Type	Value
portclass	quoted string	Name of the digital I/O driver class to use (e.g. das4020dio)
portdevice	quoted string	Device name to pass to the digital I/O driver
triggermask	unsigned integer	bitmask of bits to use for triggering, usually in hex (e.g. $20)
ddr	unsigned integer	data direction register, usually in hex. Cleared bits indicate write, set bits indicate read.
ddrmask	unsigned integer	mask of ddr bits to set (usually in hex)
mode	INTERNAL or COMPUTER	trigger mode
rate	frequency (Hz)	Initial trigger rate for internal trigger. Also maximum rate for computer-trigger

7.14.3 Commands

(commands begin on next page)

simpletrigger:MODE

Syntax

```
simpletrigger:MODE <trigger mode>
              MODE?
```

Description

Set the mode for the trigger generator.

Parameters

<**trigger mode**>	The desired trigger mode: INTernal or COMPuter

Notes

See also

- simpletrigger:RATE (pg. 164)

simpletrigger:RATE

Syntax

`simpletrigger:RATE` <**trigger frequency**>

Description

Set the frequency of the trigger generator.

Parameters

<**trigger frequency**>	The desired trigger frequency (default units of Hz)

Notes

- This frequency controls the minimum period between triggers (simpletrigger:MODE COMPUTER), or acts as an upper bound on the frequency of internally generated triggers (simpletrigger:MODE INTERNAL)/

See also

- simpletrigger:TRIGGER (pg. 165)
- simpletrigger:MODE (pg. 163)

simpletrigger:TRIGGER

Syntax

```
simpletrigger:TRIGGER
```

Description

Issue a trigger on the configured digital output.

Parameters

Notes

- Triggers will only be generated as fast as specified in simpletrigger:RATE. If you attempt to retrigger too quickly the simpletrigger:TRIGGER command will wait until the trigger period has passed before generating the trigger and returning.

See also

- simpletrigger:RATE (pg. 164)

7.15 isutriggen.so

isutriggens.so provides support for the ISU/CNDE trigger generator circuit board. This board uses an array of LS7366 timer IC's plus discrete logic to generate a master trigger pulse, with hardware-enforced holdoff, plus a delayed function generator trigger and a series of camera frame trigger pulses.

7.15.1 Prerequisites

- library dio8bit.so
- (usually) library das4020dio.so

7.15.2 Configuration parameters

Parameter	Type	Value
trigmindelay	real number	minimum holdoff between master triggers (default seconds)
fcngendelay	real number	delay between master trigger and function generator trigger (default seconds)
camtrigdelay	real number	delay between master trigger and first camera frame trigger (default seconds)
camtrigperiod	real number	delay between camera frame triggers (default seconds)
camtrigframes	integer	number of camera frame triggers per master trigger
portadevice	quoted string	device file for DAS4020dio porta (typically /dev/das4020-12/dio0_0A)
portbdevice	quoted string	device file for DAS4020dio portb (typically /dev/das4020-12/dio0_0B)
portbdevice	quoted string	device file for DAS4020dio portc (typically /dev/das4020-12/dio0_0C)
portclass	quoted string	class library for ports a, b, and c (typically das4020dio)
clockfreq	real number	frequency of master clock, (default Hz)
triggerenabled	boolean	initial trigger enable state (default TRUE)
fcngenlatency	real number	initial assumed function generator latency (default 0 seconds)
cameralatency	real number	initial assumed camera latency (default 0 seconds)

7.15.3 Commands

(commands begin on next page)

isutriggen:CLOCKFREQ

Syntax

```
isutriggen:CLOCKFREQ <frequency>
           CLOCKFREQ?
```

Description

Set or query the frequency of the master clock used by the trigger generator.

Parameters

<frequency>	The master clock frequency (default Hz)

Notes

- The master clock is not *generated* by the trigger generator. It is merely *used* by the trigger generator. This command should be used to ensure that the trigger generator knows the actual frequency of the master clock. If CLOCKFREQ is set incorrectly, the trigger generator will not generate triggers at the correct times.
- The current trigger generator hardware operates properly for master clock frequencies up to 5 MHz
- Adjusting the master clock frequency will requantize the trigger delays and periods to fit the new master clock. Resetting the master clock frequency to the original generally does not restore the original delays/periods.

See also

isutriggen:TRIGGERENABLED

Syntax

```
isutriggen:TRIGGERENABLED <enabled>
           TRIGGERENABLED?
```

Description

Set or query whether the trigger generator will accept new triggers.

Parameters

<enabled>	Boolean (TRUE or FALSE) indicating whether triggering is enabled.

Notes

- TRIGGERENABLED and CAMFREERUN are mutually exclusive
- WAITINHIBITTRIGGER can also be used to temporarily inhibit triggering

See also

- isutriggen:CAMFREERUN (pg. 169)
- isutriggen:WAITINHIBITTRIGGER (pg. 170)

isutriggen:CAMFREERUN

Syntax

```
isutriggen:CAMFREERUN <enabled>
           CAMFREERUN?
```

Description

Set or query whether the trigger generator is in camera freerun mode.

Parameters

<enabled>	Boolean (TRUE or FALSE) indicating whether camera freerun is enabled.

Notes

- TRIGGERENABLED and CAMFREERUN are mutually exclusive

See also

- isutriggen:TRIGGERENABLED (pg. 168)

`isutriggen:WAITINHIBITTRIGGER`

Syntax

```
isutriggen:WAITINHIBITTRIGGER
```

Description

Temporarily lockout triggering

Parameters

Notes

- isutriggen:WAITINHIBITTRIGGER gives the client a lock which can be held by only one client at a time.
- Each call to isutriggen:WAITINHIBITTRIGGER should be followed shortly thereafter by a call to isutriggen:ALLOWTRIGGER to release the lock and reenable triggering.
- isutriggen:WAITINHIBITTRIGGER waits asynchronously for the aforementioned lock to become available. Theoretically, this could take some time.
- To disable triggering for a long time, use isutriggen:TRIGGERENABLED FALSE instead of isutriggen:WAITINHIBITTRIGGER

See also

- isutriggen:ALLOWTRIGGER (pg. 171)
- isutriggen:TRIGGERENABLED (pg. 168)

`isutriggen:ALLOWTRIGGER`

Syntax

```
isutriggen:ALLOWTRIGGER
```

Description

Reenable triggering following isutriggen:WAITINHIBITTRIGGER.

Parameters

Notes

See also

- isutriggen:TRIGGERENABLED (pg. 168)
- isutriggen:WAITINHIBITTRIGGER (pg. 170)

isutriggen:TRIGMINDELAY

Syntax

```
isutriggen:TRIGMINDELAY <minimum holdoff>
           TRIGMINDELAY?
```

Description

Set or query the hardware-enforced minimum holdoff between triggers.

Parameters

<minimum holdoff>	The minimum holdoff (default seconds)

Notes

- The actual value (returned) is rounded up to the nearest multiple of the trigger generator master clock period.
- This time must be longer than the delay to the function generator trigger and the delay to the last camera frame trigger. While this constraint is not currently enforced, violating it may cause the system to malfunction.

See also

isutriggen:FCNGENDELAY

Syntax

```
isutriggen:FCNGENDELAY <function generator delay>
           FCNGENDELAY?
```

Description

Set or query the delay from the master trigger to the function generator trigger.

Parameters

<function generator delay>	The function generator trigger delay (default seconds)

Notes

- The actual delay function generator trigger is reduced by the value of the isutriggen:FCNGENLATENCY.
- The value returned is rounded to the nearest multiple of the trigger generator master clock period and then adjusted by the function generator latency.

See also

- isutriggen:CAMTRIGDELAY (pg. 176)
- isutriggen:CAMTRIGFRAMES (pg. 178)
- isutriggen:CAMTRIGPERIOD (pg. 177)
- isutriggen:FCNGENLATENCY (pg. 174)

isutriggen:FCNGENLATENCY

Syntax

isutriggen:FCNGENLATENCY <**function generator latency**>
FCNGENLATENCY?

Description

Set or query the assumed latency of the function generator trigger.

Parameters

<**function generator latency**>	The function generator latency (default seconds)

Notes

- A positive latency reduces the time between master trigger and function generator trigger
- If the resulting delay is smaller than the hardware can handle (a few isutriggen:CLOCKFREQ periods), the isutriggen:FCNGENDELAY will be silently increased.

See also

- isutriggen:FCNGENDELAY (pg. 173)

isutriggen:CAMERALATENCY

Syntax

```
isutriggen:CAMERALATENCY <camera latency>
           CAMERALATENCY?
```

Description

Set or query the assumed latency of the camera trigger.

Parameters

<camera latency>	The camera latency (default seconds)

Notes

- A positive latency reduces the time between master trigger and first camera trigger.
- The camera frame period is not affected by this parameter.
- If the resulting delay is smaller than the hardware can handle (a few isutriggen:CLOCKFREQ periods), the isutriggen:CAMTRIGDELAY will be silently increased.

See also

- isutriggen:CAMTRIGDELAY (pg. 176)
- isutriggen:CLOCKFREQ (pg. 167)

isutriggen:CAMTRIGDELAY

Syntax

```
isutriggen:CAMTRIGDELAY <camera trigger delay>
           CAMTRIGDELAY?
```

Description

Set or query the delay from the master trigger to the first camera frame trigger.

Parameters

<camera trigger delay>	The delay from the master trigger to the first camera trigger (default seconds).

Notes

- The actual hardware trigger delay is reduced by the specified isutriggen:CAMERALATENCY.
- The actual value (returned) is rounded to the nearest multiple of the trigger generator master clock period before the latency is added.

See also

- isutriggen:CAMERALATENCY (pg. 175)
- isutriggen:CAMTRIGFRAMES (pg. 178)
- isutriggen:CAMTRIGPERIOD (pg. 177)
- isutriggen:FCNGENDELAY (pg. 173)

isutriggen:CAMTRIGPERIOD

Syntax

```
isutriggen:CAMTRIGPERIOD <camera trigger period>
           CAMTRIGPERIOD?
```

Description

Set or query the delay between camera frame triggers.

Parameters

<camera trigger period>	The delay between camera frame triggers (default seconds).

Notes

- The actual value (returned) is rounded to the nearest multiple of the trigger generator master clock period.

See also

- isutriggen:CAMTRIGDELAY (pg. 176)
- isutriggen:CAMTRIGFRAMES (pg. 178)
- isutriggen:FCNGENDELAY (pg. 173)

isutriggen:CAMTRIGFRAMES

Syntax

```
isutriggen:CAMTRIGFRAMES <numtriggers>
           CAMTRIGFRAMES?
```

Description

Set or query the number of camera frame triggers per master trigger.

Parameters

<**numtriggers**>	The number of camera frame triggers.

Notes

See also

- isutriggen:CAMTRIGDELAY (pg. 176)
- isutriggen:CAMTRIGPERIOD (pg. 177)
- isutriggen:FCNGENDELAY (pg. 173)

isutriggen:TRIGGER

Syntax

```
isutriggen:TRIGGER
```

Description

Generate a trigger.

Parameters

Notes

- This will wait for the holdoff timer to expire, if necessary. It is therefore non-atomic.

See also

- isutriggen:TRIGGERENABLED (pg. 168)
- isutriggen:TRIGMINDELAY (pg. 172)

isutriggen:NEXTCAMFRAMETIME

Syntax

```
isutriggen:NEXTCAMFRAMETIME <frametime>
           NEXTCAMFRAMETIME?
```

Description

Query the time of the next frame in the sequence.

Parameters

<frametime>	The time of the next frame

Notes

- Query only; command syntax to illustrate result syntax only.
- isutriggen:NEXTCAMFRAMETIME returns camera frame times in a round-robin fashion.
- Use isutriggen:RESETCAMFRAMETIME to return to the first frame.

See also

- isutriggen:RESETCAMFRAMETIME (pg. 181)

isutriggen:RESETCAMFRAMETIME

Syntax

```
isutriggen:RESETCAMFRAMETIME
```

Description

Reset the frame sequence counter.

Parameters

Notes

- After isutriggen:RESETCAMFRAMETIME, isutriggen:NEXTCAMFRAMETIME returns the time of the first frame after a master trigger.

See also

- isutriggen:NEXTCAMFRAMETIME (pg. 180)

7.16 das4020dac.so

das4020dac.so provides support for the digital to analog converters on the Measurement Computing PCI-DAS4020/12 data acquisition board. The board supports +/- 1V and +/- 10V ranges. In the past we have found that capacitive cable loading makes the output amplifier unstable for the 1V setting. For this reason the default gain setting is 10V. Please note that you MUST set the ndacs configuration parameter and the device0, device1 parameters (usually ndacs=2, device0="/dev/das4020-12/da0_0" and device1="/dev/das4020-12/da0_1"). Please note that the output voltage will be set briefly to 0V during module initialization before being adjusted to the selected level.

7.16.1 Prerequisites

(none)

7.16.2 Configuration parameters

Parameter	Type	Value
ndacs	integer	number of DACs to control (this MUST be set first, and is usually 2)
device<**i**>	quoted string	the device file to use to access DAC <**i**> for 0 <= i < ndacs.
gain<**i**>	1V or 10V	the initial gain setting for DAC <**i**>.
voltage<**i**>	Voltage (volts)	the initial voltage setting for DAC <**i**>.

7.16.3 Commands

(commands begin on next page)

das4020dac:GAIN

Syntax

```
das4020dac:GAIN<channel number> <gain setting>
           GAIN<channel number>?
```

Description

Specify or query the gain setting of DAC channel <**channel number**>.

Parameters

<**channel number**>	The channel number for which to set or query the gain.
<**gain setting**>	The gain setting, 1V or 10V

Notes

- When the gain is changed, the output voltage on that channel will briefly drop to 0V.
- Capacitive loading such as a long cable may cause the DAC output amplifier to become unstable for the 1V gain setting.

See also

- das4020dac:VOLTAGE (pg. 184)

das4020dac:VOLTAGE

Syntax

```
das4020dac:VOLTAGE<channel number> <voltage>
           VOLTAGE<channel number>?
```

Description

Specify or query the output voltage of DAC channel <**channel number**>.

Parameters

<**channel number**>	The channel number for which to set or query the voltage.
<**voltage**>	The voltage to set (default units of Volts)

Notes

See also

- das4020dac:GAIN (pg. 183)

7.17 genericscpi.so

genericscpi.so provides simple support for SCPI compliant laboratory instruments over GPIB, TCP/IP, or RS-232 serial connections. Most modern laboratory instruments conform to the SCPI standard. See http://www.scpiconsortium.org/ for more information and a copy of the specification.

There are several restrictions to this implementation foremost are the inability to save instrument settings (`SET?` is not implemented) and the inability to transfer binary blocks (binary blocks may contain semicolons or newlines, which are used as message terminators by Dataguzzler). A future version may automatically detect binary blocks and escape them as used by wfmio.so, as well as using `*LRN` and `SYST:SET` to implement‘`SET?`.

Please note that queries are done synchronously in the main Dataguzzler thread. That is, if the instrument is slow to respond, all input processing will be held up. Commands are issued asynchronously and the fact that a command has been issued and Dataguzzler has responded does not mean the instrument has completed processing or that no error has occurred. To be sure a command has been completed, issue `*WAI` followed by `*OPC?` and issue a query to determine the actual state of the instrument.

It is expected that system developers may test out a new instrument using genericscpi.so. The source file genericscpi.synm4 may be copied to a new name, and then custom processing can be inserted into the syntax and the ScpiCommand syntax production can be removed to create a fully custom module.

7.17.1 Prerequisites

- library multiio.so

7.17.2 Configuration parameters

Parameter	Type	Value
uri	quoted string	multiio URI for the SCPI instrument. See the multiio.so library documentation (above)
timeout	time (seconds)	communications timeout value

7.17.3 Commands

genericscpi.so directly transcribes incoming commands to the instrument. See the manual of your instrument for details of the commands it accepts.

Suppose an instrument has a command `FREQ:START`, and you have a module SWEEPER implemented with genericscpi.so that connects to this instrument. Then when you type `SWEEPER:FREQ:START 1 MHz`, genericscpi.so will transmit `:FREQ:START 1 MHz` to the instrument and respond with `SWEEPER:FREQ:START 1 MHz`. *For any command given* (i.e. no question mark present), *the response will be exactly what was sent.*.

If a query is given, e.g. `SWEEPER:FREQ:START?` then the query will be sent to the instrument (`:FREQ:START?` in this case). The response will be of the form of the query, truncated at the question mark, followed by a space and whatever response was received from the instrument. Supposing the instrument responded with `1.e6`, the response from Dataguzzler would be `SWEEPER:FREQ:START 1.e6`.

7.18 acr9000.so

acr9000.so provides support for the Parker/Compumotor ACR-9000 motion controller. This module will probably also work with other motion controllers in the Parker/Compumotor AcroLoop series.

In order to work with this module your motion controller must already have its axes configured, using one or more "program levels" of the ACR-9000. One "program level" must be left available for the exclusive use of this module. This defaults to "prog15", but may be set with the "spareprog" configuration parameter.

On module initialization, acr9000.so will automatically probe the motion controller to find configured axes. It will list the axes that it finds to stderr in the form:

```
ACR9000:  Configured axis X on PROG0 AXIS0 PPU 1000
```

acr9000.so can communicate with the motion controller over TCP/IP or serial using tcp:// or serial:// URI's. To connect over TCP/IP to the factory default IP address, use "tcp://192.168.10.40:5002" as the URI.

7.18.1 Prerequisites

- library multiio.so

7.18.2 Configuration parameters

Parameter	Type	Value
uri	quoted string	multiio URI for the ACR-9000. Should be tcp:// or serial://
spareprog	unsigned integer	spare program level for controlling the ACR-9000 (default 15)
axisunits:<**axis**>	units for the axis: in, mil, ft, mm, cm, m, deg, rad, etc.	

7.18.3 Commands

(commands begin on next page)

acr9000:AXIS

Syntax

```
acr9000:<AXIS> <position> <units>
        <AXIS>?
```

Description

Specify or query the position of a motion controller axis.

Parameters

<**AXIS**>	The axis to control or query.
<**position**>	Desired or actual position of the axis
<**units**>	Units of <**position**>

Notes

- If units are not specified, the default units for this axis are assumed
- The query form always returns a number in the default units for this axis.
- The axis cannot move unless it is turned on with acr9000:<**AXIS**>:STATUS.
- Multiple axes may move simultaneously. Multiple moves on a single axis will queue up and execute in sequence. Avoid queueing large numbers of moves.

See also

- acr9000:AXIS:REL (pg. 192)
- acr9000:AXIS:STATUS (pg. 193)

acr9000:AXIS:CANCEL

Syntax

```
acr9000:<AXIS>:CANCEL
        ALL:CANCEL
```

Description

Stop and cancel motion on the specified axis

Parameters

<AXIS>	The axis to stop, or ALL

Notes

- This permanently cancels any current or pending motion on the specified axis.

See also

- acr9000:AXIS (pg. 188)
- acr9000:AXIS:MOVING (pg. 190)
- acr9000:WAIT (pg. 194)

acr9000:AXIS:MOVING

Syntax

`acr9000:<`**AXIS**`>:MOVING?`

Description

Specify or query the position of a motion controller axis.

Parameters

<**AXIS**>	The axis to query.

Notes

- The response will be of the form <**AXIS**>`:MOVING YES` or <**AXIS**>`:MOVING NO`.

See also

- acr9000:AXIS (pg. 188)
- acr9000:AXIS:STATUS (pg. 193)
- acr9000:WAIT (pg. 194)

acr9000:AXIS:NOM

Syntax

`acr9000:<`**AXIS**`>:NOM?`

Description

Query the current nominal position

Parameters

<**AXIS**>	The axis for which to get the position.

Notes

- The nominal position is the intended position according to the trajectory currently being executed. In contrast acr9000:<**AXIS**>? returns the actual position of the axis, which may be slightly away from the nominal position.

See also

- acr9000:AXIS (pg. 188)
- acr9000:AXIS:MOVING (pg. 190)

acr9000:AXIS:REL

Syntax

acr9000:<**AXIS**>:REL <**position**> <**units**>

Description

Initiate a relative position move..

Parameters

<**AXIS**>	The axis to control or query.
<**position**>	Desired relative position of the axis
<**units**>	Units of <**position**>

Notes

- If units are not specified, the default units for this axis are assumed
- The axis cannot move unless it is turned on with acr9000:<**AXIS**>:STATUS.
- The position is relative to the current axis position at the instant this command is processed.

See also

- acr9000:AXIS (pg. 188)
- acr9000:AXIS:MOVING (pg. 190)

acr9000:AXIS:STATUS

Syntax

```
acr9000:<AXIS>:STATUS <axis status>
        <AXIS>:STATUS?
```

Description

Specify or query the status of a motion controller axis.

Parameters

<AXIS>	The axis to control of query.
<axis status>	The axis status, `ON` or `OFF`

Notes

- **<AXIS>** may be `ALL` to control all axes simultaneously.

See also

- acr9000:AXIS (pg. 188)

acr9000:WAIT

Syntax

acr9000:<**AXIS**>:WAIT
ALL:WAIT
WAIT <**axis list**>

Description

Wait for one or more axes to stop.

Parameters

<**AXIS**>	The axis to wait for.
<**axis list**>	A list of axes to wait for, separated with spaces.

Notes

- This waits for all the specified axes to stop (reach their target positions.
- If a WAIT command gets stuck, you can release the wait with the acr9000:<**AXIS**>:CANCEL command on all the axes being waited for.
- If multiple moves are queued up, the WAIT may return at the end of any of the moves. When programming, always maintain a 1:1 mapping between moves and waits.

See also

- acr9000:AXIS (pg. 188)
- acr9000:AXIS:MOVING (pg. 190)

7.19 subproc.so

subproc.so enables nested sub-processes. By using a subprocess, you can reduce the chance that a slow module will clog the main event loop. The disadvantage of subprocesses is that since they have their own copies of libraries, waveforms cannot be readily shared between the subprocess and the main dataguzzler process.

7.19.1 Prerequisites

- library multiio.so

7.19.2 Configuration parameters

The configuration section is transmitted verbatim to the subprocess. In addition, an extra parameter (quoted string) can be added (separated by a comma) within the parentheses of the module specification. This extra parameter is an alternative command to run to start the subprocess. It can be used to, for example, start the subprocess remotely through a script that exec's a remote dataguzzler over `ssh`.

7.19.3 Commands

Commands are passed directly through to the remote dataguzzler. To avoid excess levels of indirection in the command names, you may want to use an empty module name for the module within the subprocess dataguzzler. This way, you can refer to the module within the subprocess just by using the subprocess module name.

Appendix A

Dataguzzler native binary file format

The dataguzzler native file format is that written by the dg_grab download program and that read by the dg_upload program. The dataguzzler native file format consists of a header followed by a series of chunks. The header is the 8 bytes DATAGUZZ (big endian architectures) or ZZUGATAD (little endian architectures). (recall that all Intel-architecture PC's are little endian). The next 16 bytes is the 8 byte header of the first chunk followed by the 8 byte integer length of the first chunk (not including the length of its header). The first chunk data follows its header. The header for the second chunk follows, etc. Note that all chunks are implicitly padded to be multiples of 64 bits (8 bytes) in size. That is, the written size is the actual size, but zeros are added after the chunk to make an even multiple of 8 bytes before the next chunk is written. Note that on little endian architectures the chunk headers specified below are reversed.

Chunk header	contents
DATAGUZZ	wrapper around entire file.
GUZZNWFM	Data describing a named waveform. Should contain a WAVENAME chunk followed by a GUZZWFMD chunk
WAVENAME	String containing the name of the waveform.
GUZZWFMD	Data describing a single waveform. Should contain a METADATA chunk followed by a WFMDIMNS chunk followed by a DATARRYF or DATARRYD chunk
METADATA	a series of METDATUM subchunks.
METDATUM	a METDNAME subchunk containing the metadatum name followed by a METDINTV, METDSTRV, or METDDBLV subchunk
METDINTV	Integer metadatum value, 64 bit signed integer (int64_t)
METDSTRV	String metadatum value. The size of the chunk is the length of the string
METDDBLV	Double precision metadatum value, type double
WFMDIMNS	Waveform dimensions. Starts with two 64 bit integers (uint64_t): The product of the dimensions and the number of the dimensions (ndim). This is followed by ndim uint64_t's containing the lengths of the individual dimensions
DATARRYF	single-precision array data. The first index changes most rapidly
DATARRYD	double-precision array data. The first index changes most rapidly
SNAPSHOT	Snapshot of an experiment (results for a specific set of parameters). This consists of a METADATA chunk with the parameters of the experiment, followed by a series of GUZZNWFM chunks with the data from the snapshot
SNAPSHTS	series of SNAPSHOT chunks
VIBRDATA	SNAPSHOT chunk containing configuration parameters and results from vibration measurement: Two SNAPSHOT chunks followed by a VIBFCETS chunk. See boundary_collect_procedure.pdf for more information.
VIBFCETS	One or more VIBFACET chunks
VIBFACET	Parameters/results of a facet: SNAPSHOT, followed by a series of GUZZNWFM chunks. See boundary_collect_procedure.pdf for more information.

A.1 Standardized file name extensions

Extension	MIME type	contents
dgz	application/x-dataguzzler-waveform	Single unnamed waveform (GUZZWFMD)
dga	application/x-dataguzzler-array	Array of unnamed waveforms (series of GUZZWFMD chunks)
dgs	application/x-dataguzzler-snapshot	Snapshot of named waveforms (SNAPSHOT chunk, typically with the METADATA section empty) (older version was a series GUZZNWFM chunks)
dgd	application/x-dataguzzler-data	Data from a series of experiments (SNAPSHTS chunk)
set	application/x-dataguzzler-settings	This is not a chunked format. It is a raw dump of the output of `WFM:WFMS?` followed by the output of `SET?`
vibr	application/x-dataguzzler-vibration	Data from a series of vibration measurements (VIBRDATA chunk)

A.2 File access API

Dataguzzler file I/O has been implemented in libraries written in C, MATLAB/Octave, and Python+Numpy. The bulk of this API documentation describes the C library. The API is similar in the other languages.

A.2.1 Reading a Dataguzzler file

Open a Dataguzzler file for reading with the `dgf_open()` function:

```
struct dgf_file *infile;

infile=dgf_open(char *filename);
```

`dgf_open()` returns an opaque file handle or NULL if the file could not be opened. After opening the file, the first step is reading the first chunk header. This is done with `dgf_checknextchunk()` or `dgf_nextchunk()` depending on whether or not you know for certain the type of the first chunk.

```
struct dgf_Chunk *Chunk;

Chunk=dgf_nextchunk(struct dgf_file *infile);
Chunk=dgf_checknextchunk(struct dgf_file *infile, char *chunkname);
```

dgf_nextchunk() opens the next available chunk, or returns NULL if there are no chunks left.
dgf_checknextchunk() opens the next available chunk if chunkname matches its eight character name. Otherwise it skips the chunk and returns NULL. If there are no chunks left it also returns NULL.

The dgf_Chunk structure contains several useful members: Name, ChunkLen, and ChunkPos

```
struct dgf_Chunk { /* on ChunkStack */
  struct dgl_Node Node;
  char Name[9]; /* 8 characters + 0 terminator so we can use strcmp() et al. */
  int64_t ChunkStart;
  int64_t ChunkLen; /* used only in read routines */
  int64_t ChunkPos; /* relative to ChunkStart */
};
```

- Name is a null-terminated copy of the 8-character chunk name string.
- ChunkLen is the length of the chunk contents, not including any headers and/or padding.
- ChunkPos is the number of bytes that have been read from the chunk.

The chunk can contain either binary data or nested chunks. dgf_readdata() can be used to read binary data or dgf_nextchunk() can be used to open nested chunks. When done reading the contents of the chunk, you must call dgf_chunkdone() to close the chunk. Once the last chunk is closed, call dgf_close() to close the file.

```
void dgf_readdata(struct dgf_file *infile,void *Buf,int64_t nbytes);
void dgf_chunkdone(struct dgf_file *infile,struct dgf_Chunk *Chunk); /* 2nd parameter may be NULL */
void dgf_close(struct dgf_file *infile);
```

dgf_readdata() reads the specified number of bytes from the current chunk into the specified buffer.
dgf_chunkdone() closes the most recent open chunk so that following calls to dgf_<check>nextchunk() open the following chunk, not nested chunks. dgf_close() closes the file after the last chunk is done.

Special-purpose processing routines are included for a few chunk types. These routines should be called immediately after dgf_<check>nextchunk() and include calls to dgf_chunkdone() to close the chunk.

```
struct dg_wfminfo *dgf_procGUZZWFMD(struct dgf_file *file,char *WfmName);
struct dg_wfminfo *dgf_procGUZZNWFM(struct dgf_file *file);
void dgf_procMETADATA(struct dgf_file *file,struct dgl_List *MetaData);
```

dgf_procGUZZWFMD() processes an unnamed waveform GUZZWFMD chunk. A name (WfmName) or NULL may be provided as a parameter. The routine returns a struct dg_wfminfo * containing the waveform data. Similarly, dgf_procGUZZNWFM() processes a named waveform GUZZNWFM chunk, also returning a struct dg_wfminfo *.
dgf_procMETADATA() reads a METADATA chunk, adding the metadata elements to the pre-existing and pre-initialized MetaData list.

A.2.2 Writing a Dataguzzler file

Open a Dataguzzler file for writing with the `dgf_creat()` function:

```
struct dgf_file *outfile;

outfile=dgf_creat(char *Name);
```

`dgf_creat()` returns an opaque file handle or NULL if the file could not be opened. After opening the file, chunks and nested chunks may be written. Create a chunk with `dgf_startchunk()`

```
void dgf_startchunk(struct dgf_file *outfile,char *chunkname);
```

The `chunkname` parameter gives the eight-character null-terminated chunk ID. The chunk can contain nested chunks (create with `dgf_startchunk()`) or binary data (write with `dgf_writedata()`). When done writing the chunk, end it with `dgf_endchunk()`.

```
void dgf_writedata(struct dgf_file *outfile,void *buf,int64_t nbytes);
void dgf_endchunk(struct dgf_file *outfile);
```

When done with the last chunk, close the file with `dgf_close()`.

```
void dgf_close(struct dgf_file *infile);
```

Special-purpose writing routines are included for a few chunk types. These routines start the chunk, write the contents, and end the chunk.

```
void dgf_writenamedwfm(struct dgf_file *file,struct dg_wfminfo *wfm);
void dgf_writewfm(struct dgf_file *file,struct dg_wfminfo *wfm);
void dgf_writemetadata(struct dgf_file *file,struct dgl_List *MetaData);
```

These routines write `GUZZNWFM`, `GUZZWFMD`, and `METADATA` chunks respectively with the provided waveform/metadata.

A.2.3 MATLAB/Octave file access library

The MATLAB API is essentially similar to the C API. You will need to place the `.m` files somewhere in MATLAB's path. This can be done with the MATLAB `path()` function or with the `MATLABPATH` environment variable. The primary difference between the MATLAB and C APIs is that since MATLAB parameters are passed exclusively by value rather than by reference, the filehandle structure is processed by each routine and a new filehandle is returned by each routine as an extra result of the function. See `dgf_testread.m` for an example.

A.2.4 Python file access library

The Python API is essentially similar to the C API. The Numpy (numerical python) library is required.

You will need to `import dg_file`. Then you can access the routines such as `infile=dg_file.dgf_open(filename)`.

A.2.5

Appendix B

IR Camera Calibration

B.1 Blackbody calibration

The blackbody calibration maps the value from the IR camera A/D converter to blackbody temperature T_b. The blackbody calibration is performed by pointing the camera at a (presumed perfect) blackbody with a thermistor/thermocouple attached to measure the temperature. The blackbody is then slowly heated and IR images are recorded as a function of temperature. The response of the camera is modeled by a polynomial that approximates temperature as a function A/D converter value. The order of this polynomial must be selected in advance and the coefficients (one set of coefficients per pixel) are calculated with a least-squares fit to the measured A/D converter value vs. temperature curve.

One the calibration is loaded, the camera gives images of blackbody temperature T_b instead of images of A/D converter value.

B.2 Graybody correction

The total radiant emittance of a graybody is given by the Stefan-Boltzmann law:

$$\Phi = \epsilon\sigma T^4 \tag{B.1}$$

where Φ is the energy flux in J/(m^{2}s), ϵ is the graybody coefficient, σ is the Stefan-Boltzmann constant, 5.670×10^{-8}J/($^\circ$K^{4}m^{2}s), and T is the temperature $^\circ K$.

The camera is calibrated against a blackbody. The calibration maps the value from the camera A/D converter to blackbody temperature T_b. Therefore the energy flux Φ that corresponds to a camera reading of T_b is σT_b^4.

Suppose the camera is pointed instead at a graybody, with coefficient ϵ. The measured flux Φ comes from two places (ignoring the effect of the atmosphere):

1. Emittance of the graybody
2. Infrared light reflected by the graybody

$$\Phi = \epsilon\sigma T^4 + (1-\epsilon)\sigma T_a^4 \tag{B.2}$$

where T is the actual temperature of the graybody and T_a is the ambient temperature of the surroundings (as reflected by the graybody). The camera, calibrated against a blackbody, gives T_b and we can calculate Φ from T_b: $\Phi = \sigma T_b^4$. Therefore, solving for T, our estimate of the graybody temperature T is:

$$T = \sqrt[4]{\frac{1}{\epsilon}T_b^4 - \frac{1-\epsilon}{\epsilon}T_a^4} \tag{B.3}$$

Therefore to perform the graybody correction, two values are needed: The graybody coefficient ϵ, and the ambient temperature T_a.

Because of the variation in the emissivity and temperature of the surroundings and the potential for specular reflections, the absolute temperature calculated from Eq. B.3 will not be very accurate unless the graybody coefficient is close to 1.0. Nevertheless, as long as ϵ is known accurately, Eq. B.3 should give temperature changes quite accurately. Assuming there are no movements that change what is reflected by the object under test, the second term under the fourth root in Eq. B.3 will be constant. Therefore small changes in T map directly to changes in T_b, independent of T_a:

$$\frac{dT}{dT_b} = \frac{1}{\sqrt[\frac{3}{4}]{\frac{1}{\epsilon}T_b^4 - \frac{1-\epsilon}{\epsilon}T_a^4}}\left(\frac{1}{\epsilon}\right)T_b^3 = \frac{1}{\epsilon}\frac{T_b^3}{T^3} \tag{B.4}$$

In order to perform this correction, we need estimates of both the graybody coefficient ϵ and the ambient temperature T_a, each of which may be functions of position in the image. To estimate ϵ we attach a thermistor (or thermocouple) and a power resistor to the specimen (all sides of the power resistor not in contact with the specimen should be insulated). The specimen is assumed to be a good thermal conductor and therefore of uniform temperature. We record the camera image and temperature. Then we apply current to the power resistor and allow the part to heat up a few degrees. Then we record the new camera image and temperature (from the thermistor/thermocouple). From the two images at the two temperatures, we can solve Eq. B.3 for ϵ and T_a (which should be close to the ambient air temperature):

$$\epsilon = \frac{T_{b1}^4 - T_{b2}^4}{T_1^4 - T_2^4} \tag{B.5}$$

and

$$T_a = \sqrt[4]{\frac{T_{b1}^4 - \epsilon T_1^4}{1-\epsilon}} \tag{B.6}$$

This correction can be done independently for each pixel because the emissivity can vary spatially and the effective ambient temperature is a function of whatever is reflected into each pixel.

B.3 Bad Pixel Correction

Bad pixels are identified as those with very low or very high A/D values, or those with particularly large standard deviations. The bad pixels are replaced (after calibration and correction) with an average of their 4 nearest neighbors. To accomodate clusters of bad pixels, this replacement is applied iteratively (with bad pixel replacements used in the new calculations as soon as they have been set – even in the same iteration) until no pixel changes by more than .003% (approx. .01 deg. K at room temperature).

B.4 IR Camera Calibration and Bad Pixel Correction in Dataguzzler

A utility program `dg_calibrate_camera` is used to record calibration images and temperatures. Calibration images are stored in a `.dga` (Dataguzzler array) file as a series of unnamed frames (`GUZZWFMD` chunks) with the temperature specified as the third coordinate (after the X and Y pixel coordinates). An Octave script, `proccalib.m`, calculates the best-fit coefficients and transforms the `.dga` file into a `.dgs` file that can be loaded into dataguzzler with dg_load_snapshot and used as the second parameter of the IRCALIB math function. This same script identifies bad pixels and the IRCALIB math function also corrects those bad pixels. A second utility program `dg_correct_graybody` can be run while the power resistor mentioned in section B.2 is heating the specimen and this program will upload its measured T_a and ϵ images into Dataguzzler. It is suggested that laboratory personnel leave the laboratory while `dg_correct_graybody` is running so that reflections of moving people do not contaminate the graybody calculations.

Appendix C

License

```
The Dataguzzler main program and modules are licensed in parallel
under the GNU General Public License version 2 (or any later version)
and the GNU General Public License version 2 (or any later version)
with the following exception: Third party data acquisition libraries
or hardware drivers may be treated as if they were a major component
of the operating system; therefore the special exception in GPLV2,
section 3 applies to them. Please note that this exception applies
only to dataguzzler code and libraries, and not any 3rd party code
that might be used or linked-to by dataguzzler.


The Dataguzzler libraries are licensed in parallel under the GNU
Lesser General Public License version 2.1 (or any later version) and
the GNU Lesser General Public License version 2.1 (or any later
version) with the following exception: Third party data acquisition
libraries or hardware drivers may be treated as if they were a major
component of the operating system; therefore the special exception in
GPLV2, section 3 applies to them. Please note that this exception
applies only to dataguzzler code and libraries, and not any 3rd party
code that might be used or linked-to by dataguzzler.

----------------------------------------

    GNU GENERAL PUBLIC LICENSE
       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.
```

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if