Dataguzzler as an Imaging Application

Stephen D. Holland

June 27, 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

This manual describes the use of Dataguzzler as an imaging application for the EDT PCI-DV/CLINK Camera Link framegrabber. It is presented as an example of a simple, single-purpose Dataguzzler application. For more information on configuring Dataguzzler for more sophisticated applications, see the Dataguzzler manual: *Dataguzzler: A Generalized High Performance Data Acquisition Architecture*.

Chapter 1

Installation

1.1 System requirements

- Linux with 2.6 or newer kernel.
- The EDT PCI-DV/CLINK framegrabber, or a similar API compatible framegrabber.
- A CameraLink or similar camera that can output 8 or 16 bit grayscale.
- The EDTpdv Linux driver, installed in /opt/EDTpdv or /usr/local/EDTpdv.

Dataguzzler should be compiled with the `make` command, and installed with the `make install` command. Watch the "make" output for errors.

In order to enable default use of the Dataguzzler cameralink configuration by default, you must symbolic-link it to dataguzzler.conf:

```
rm -f /usr/local/dataguzzler/conf/dataguzzler.conf
ln -s dataguzzler_camlink.conf /usr/local/dataguzzler/conf/dataguzzler.conf
```

Chapter 2

System Control

The data acquisition system consists of a *dataguzzler server* containing a set of interconnected modules, plus various *clients* that connect to the server and issue commands and transfer data. Commands to the server can be issued from the dataguzzler console or over TCP/IP links. In contrast, the clients are separate programs that themselves connect to the server. To operate dataguzzler, first run the server in a terminal window (xterm):

```
pequod% dataguzzler
dataguzzler>
```

Commands to the server can be issued directly in that terminal window. Clients, on the other hand must be started separately. Open a new terminal window and type `dg_scope`. A oscilloscope display will appear. The oscilloscope display shows the current waveforms and images in dataguzzler memory. Note that it is an observation tool only; no changes or adjustments may be made from the oscilloscope display. Other clients are used to load and save settings and to upload and download waveforms from the server.

2.1 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 4 on page 16.

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. **??**), adjusts the capture frequency form 10 MHz to 1 Mhz, and exits.

The complete list of available commands can be found in chapter 4. 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 4. 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.

2.2 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.

2.3 Utilities

2.3.1 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.

2.3.2 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.

2.3.3 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.

2.3.4 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.

2.3.5 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.

2.3.6 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.

2.3.7 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.

2.3.8 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.

2.3.9 dg_snapshot

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

2.3.10 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`.

2.4 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 3

System Operation

The Dataguzzler-based imager, because of its simplicitly, has only two important parameters. `ICAPT:GEOMETRY` is used to adjust the frame size expected from the capture card (permanent settings may be better placed in dataguzzler_camlink.conf in /usr/local/dataguzzler/conf). `ICAPT:CALCSYNC` can be used to determine whether dataguzzler should allow itself to capture a new frame before processing of the previous frame is complete.

A series of frames can be acquired by defining a math channel with the ACCUM or ACCUMONCE functions. For example, MATH:DEF movie=ACCUMONCE(CAMLINK,100) will acquire a sequence of 100 frames. Various other operations are also available through the MATH module.

Chapter 4

Command Reference

A wealth of commands are provided to communicate with the hardware devices. Each command begins with the name of the corresponding module. These commands can be given either from the console or over a TCP/IP connection. In addition to these commands, `SET?` queries the status of all settings, and `SET` causes all modules to rewrite settings to hardware.

(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

ICAPT:CALCSYNC

Syntax

`ICAPT: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

ICAPT:GEOMETRY

Syntax

`ICAPT: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

MATH:CLEARAVG

Syntax

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

Description

Reset averaging channel <**waveform name**>

Parameters

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

Notes

See also

MATH:CLEARACCUM

Syntax

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

Description

Reset ACCUM channel <**waveform name**>

Parameters

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

MATH:DEF

Syntax

`MATH: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 MATH: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 $20 \log 10(\text{abs}(<\textbf{channel}>))$)

See also

- MATH:UNDEF (pg. 27)

MATH:ENABLE

Syntax

`MATH: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

- MATH:ENABLED (pg. 25)
- MATH:DISABLE (pg. 26)

MATH:ENABLED

Syntax

`MATH: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 MATH:ENABLE <**waveform name**> or MATH:DISABLE <**waveform name**> depending on whether <**waveform name**> is enabled.

See also

- MATH:ENABLE (pg. 24)
- MATH:DISABLE (pg. 26)

MATH:DISABLE

Syntax

`MATH: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

- MATH:ENABLE (pg. 24)
- MATH:ENABLED (pg. 25)

MATH:UNDEF

Syntax

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

Description

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

Parameters

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

Notes

See also

- MATH:DEF (pg. 21)
- MATH:UNDEFALL (pg. 28)

MATH:UNDEFALL

Syntax

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

Description

Delete all math channels

Parameters

Notes

See also

- MATH:DEF (pg. 21)
- MATH:UNDEF (pg. 27)

MATH:WAITAVG

Syntax

`MATH: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

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

WFM:COPY

Syntax

`WFM: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 WFM:DELETE command.

See also

- WFM:DELETE (pg. 36)

WFM:DATA

Syntax

```
WFM: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 WFM: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

- WFM:REALSZ (pg. 46)
- WFM:METADATA (pg. 45)

WFM:DATASHM

Syntax

`WFM: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 WFM: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 WFM:REALSZ)

See also

- WFM:REALSZ (pg. 46)
- WFM:DATA (pg. 33)
- WFM:METADATA (pg. 45)

WFM:DELETEALL

Syntax

```
WFM:DELETEALL
```

Description

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

Parameters

Notes

See also

- WFM:DELETE (pg. 36)
- WFM:COPY (pg. 32)
- WFM:DATA (pg. 33)

WFM:DELETE

Syntax

`WFM: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 WFM:LIST or WFM:LISTLOCK.
- Only waveforms created by the user can be deleted with the WFM:DELETE command.

See also

- WFM:DELETEALL (pg. 35)
- WFM:COPY (pg. 32)
- WFM:DATA (pg. 33)

WFM:GLOBALREADYREV

Syntax

```
WFM: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

- WFM:GLOBALREV (pg. 39)
- WFM:GLOBALREADYREVTIMEOUT (pg. 38)

WFM:GLOBALREADYREVTIMEOUT

Syntax

`WFM: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

- WFM:GLOBALREVTIMEOUT (pg. 40)
- WFM:GLOBALREADYREV (pg. 37)

WFM:GLOBALREV

Syntax

```
WFM: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

- WFM:GLOBALREVTIMEOUT (pg. 40)
- WFM:GLOBALREADYREV (pg. 37)

WFM:GLOBALREVTIMEOUT

Syntax

`WFM: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

- WFM:GLOBALREV (pg. 39)
- WFM:GLOBALREADYREVTIMEOUT (pg. 38)

WFM:LIST

Syntax

```
WFM: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

- WFM:METADATA (pg. 45)
- WFM:DATA (pg. 33)
- WFM:LISTLOCK (pg. 43)
- WFM:LISTREADY (pg. 42)

WFM:LISTREADY

Syntax

```
WFM: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

- WFM:METADATA (pg. 45)
- WFM:DATA (pg. 33)
- WFM:LISTREADYLOCK (pg. 44)
- WFM:LISTREADY (pg. 42)

WFM:LISTLOCK

Syntax

```
WFM: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 WFM: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

- WFM:METADATA (pg. 45)
- WFM:DATA (pg. 33)
- WFM:UNLOCK (pg. 50)
- WFM:LIST (pg. 41)

WFM:LISTREADYLOCK

Syntax

```
WFM: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 WFM: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

- WFM:METADATA (pg. 45)
- WFM:DATA (pg. 33)
- WFM:UNLOCK (pg. 50)
- WFM:LISTREADY (pg. 42)
- WFM:LISTLOCK (pg. 43)

WFM:METADATA

Syntax

`WFM: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 `WFM: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

WFM:REALSZ

Syntax

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

Description

Returns the number of bytes per floating point number returned by WFM:DATA or WFM: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

- WFM:DATA (pg. 33)
- WFM:DATASHM (pg. 34)

WFM:REVISION

Syntax

```
WFM: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

- WFM:METADATA (pg. 45)
- WFM:DATA (pg. 33)
- WFM:LIST (pg. 41)
- WFM:REVISIONLOCK (pg. 48)

WFM:REVISIONLOCK

Syntax

```
WFM: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 WFM: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

- WFM:METADATA (pg. 45)
- WFM:DATA (pg. 33)
- WFM:LIST (pg. 41)
- WFM:REVISION (pg. 47)

WFM:REVISIONREADYLOCK

Syntax

`WFM: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 WFM: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

- WFM:METADATA (pg. 45)
- WFM:DATA (pg. 33)
- WFM:LIST (pg. 41)
- WFM:REVISION (pg. 47)

WFM:UNLOCK

Syntax

`WFM: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 WFM:REVISIONLOCK or WFM:LISTLOCK.

See also

- WFM:LISTLOCK (pg. 43)
- WFM:REVISIONLOCK (pg. 48)

WFM:WFMS

Syntax

```
WFM:WFMS?
```

Description

Download the full set of user-defined waveforms.

Parameters

Notes

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

See also

- WFM:DATA (pg. 33)

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