



Interface Control Document

Client to Detector Controller Server

XXX.XXX.XXX

4 November 2021



XXX.XXX.XXX

Detector Controller Server Interface

Page 2 of 40

4 November 2021

DOCUMENT APPROVAL

Author Release Note:

Prepared By:

(David Hale)
(Programmer)

Concurrence:

(Add Name)
(Add Title)

(Add Name)
(Add Title)

Approval:

(Add Name)
(Add Title)



TABLE OF CONTENTS

1. INTRODUCTION	8
1.1 BACKGROUND.....	8
1.2 SCOPE.....	8
1.3 DOCUMENT OUTLINE	9
1.4 CHANGE RECORD.....	10
1.5 ABBREVIATIONS, DEFINITIONS AND NOMENCLATURE.....	10
2. WORKED EXAMPLES.....	10
2.1 COMPILATION.....	10
2.2 CONFIGURATION	11
2.3 RUNNING THE SERVER.....	12
2.4 THE SIMPLEST COMMAND-LINE EXPOSURE.....	12
3. INTERFACE SPECIFICATIONS	13
3.1 PROTOCOL.....	13
3.2 CONFIGURATION.....	13
3.2.1 Time Zone.....	14
3.3 PORTS	15
3.3.1 Blocking Port.....	15
3.3.2 Non-Blocking Port.....	15
3.3.3 Asynchronous Message Port.....	15
3.4 CLIENTS.....	15
3.4.1 Scripting languages.....	15
3.4.2 Shell (e.g. csh, bash, etc.).....	16
3.4.3 telnet.....	16
3.5 DAEMONIZE	17
3.6 COMPILATION.....	17
3.6.1 Requirements.....	17
3.6.1.1 For ARC controllers	17
3.6.1.2 For STA/Archon controllers	17
3.6.2 Build Instructions.....	17
4. COMMANDS	18
4.1 COMMAND FORMAT.....	18
4.2 RETURN VALUES	18
4.3 COMMAND SYNTAX.....	19
4.3.1 abort - abort an exposure in progress (ARC-only).....	19
4.3.2 amplifier - set or get which detector amplifier(s) to use	19
4.3.3 autodir - automatically create UTC subdirectory for images.....	19
4.3.4 basename - base image name for FITS image files.....	19



4.3.5	<i>bias</i> – bias voltage.....	20
4.3.6	<i>buffer</i> – PCI/e image buffer mapping (ARC-only).....	20
4.3.7	<i>cds</i> – CDS/Deinterlace configuration key (Archon-only).....	20
4.3.8	<i>close</i> – close the connection to the detector controller.....	20
4.3.9	<i>datacube</i> – data cube state.....	21
4.3.10	<i>echo</i> – test server communication.....	21
4.3.11	<i>exit</i> – exit the server.....	21
4.3.12	<i>expose</i> – initiate an exposure.....	21
4.3.13	<i>exptime</i> – exposure time.....	21
4.3.14	<i>fitsnaming</i> – FITS filename format.....	22
4.3.15	<i>framestatus</i> – read the frame status (Archon-only).....	22
4.3.16	<i>geometry</i> – detector geometry (ARC-only).....	22
4.3.17	<i>getp</i> – get parameter value (Archon-only).....	23
4.3.18	<i>hdrshift</i> – HDR bit-shift value (Archon-only).....	23
4.3.19	<i>heater</i> – control HeaterX module (Archon-only).....	23
4.3.19.1	Set or get the state and target for heater A or B on the specified module.....	23
4.3.19.2	Set or get the PID parameters for heater A or B on the specified module.....	23
4.3.19.3	Set or get the ramp and ramprate for heater A or B on the specified module.....	23
4.3.20	<i>imdir</i> – base image directory for FITS image files.....	24
4.3.21	<i>imnum</i> – image number for FITS image files.....	24
4.3.22	<i>inreg</i> – write to VCPU input register (Archon-only).....	24
4.3.23	<i>interface</i> – return the detector interface type.....	24
4.3.24	<i>key</i> – add FITS keyword to user-defined database.....	25
4.3.25	<i>load</i> – load firmware into detector controller.....	25
4.3.26	<i>loadtiming</i> – load timing script and parameters (Archon-only).....	25
4.3.27	<i>longexposure</i> – long exposure mode to extend exptime unit (Archon-only).....	26
4.3.28	<i>mode</i> – camera mode (Archon-only).....	26
4.3.29	<i>open</i> – open a connection to the detector controller.....	26
4.3.30	<i>sensor</i> – set or get temperature sensor current (Archon-only).....	26
4.3.31	<i>setp</i> – set parameter value (Archon-only).....	27
4.3.32	<i>shutter</i> – set or get the shutter enable state on expose.....	27
4.3.33	<i>systemfile</i> – generate a system file (Archon-only).....	27
4.3.34	<i>useframes</i> – set or get the useframes flag (ARC-only).....	27
4.3.35	<i>writekeys</i> – when to write FITS header keywords.....	27
4.3.36	<i>writep</i> – write parameter value to configuration memory (Archon-only).....	28
4.4	<i>TO-DO</i>	28
4.4.1	<i>vertical binning</i>	28
4.4.2	<i>horizontal binning</i>	28
4.4.3	<i>region of interest</i>	28
4.4.4	<i>erase userkey DB</i>	28



5. ASYNCHRONOUS MESSAGES.....	28
5.1 ASYNCHRONOUS MESSAGE TAGS.....	28
5.1.1 <i>DATAcube:n COMPLETE ERROR</i>	28
5.1.2 <i>EXPOSURE:n (Archon-only)</i>	29
5.1.3 <i>EXPOSURE_d:n (ARC-only)</i>	29
5.1.4 <i>FILE:<fully_qualified_path> COMPLETE ERROR</i>	29
5.1.5 <i>LINECOUNT:n (Archon-only)</i>	29
5.1.6 <i>NOTICE:message</i>	29
5.1.7 <i>PIXELCOUNT:n (ARC-only)</i>	29
5.2 ARCHON NATIVE COMMANDS.....	29
6. FITS HEADERS.....	29
6.1 BUILT-IN KEYWORDS.....	29
6.2 HEADER DATA COLLECTION TIMING.....	30
7. NATIVE CONTROLLER COMMANDS.....	30
7.1 STA/ARCHON.....	31
7.1.1 <i>FRAME</i>	31
7.1.2 <i>STATUS</i>	32
7.1.3 <i>SYSTEM</i>	34
7.2 ARC (ASTROCAM).....	34
7.2.1 <i>POF — Power Off</i>	35
7.2.2 <i>PON — Power On</i>	35
7.2.3 <i>RDM — Read Memory</i>	35
7.2.4 <i>SBN — Set Bias Number</i>	35
7.2.5 <i>SMX — Set Multiplexer</i>	36
7.2.6 <i>TDL — Test Data Link</i>	36
7.2.7 <i>WRM — Write Memory</i>	36
8. FIRMWARE REQUIREMENTS.....	36
8.1 STA/ARCHON.....	36
8.2 ARC (ASTROCAM).....	36
8.2.1 <i>X:STATUS Register</i>	36
9. APPENDIX A – EMULATOR.....	36
9.1 OVERVIEW.....	36
9.2 COMPILATION.....	37
9.3 CONFIGURATION.....	37
9.4 ARCHON MODULE EMULATION.....	38
9.5 RUNNING.....	38
9.6 EMULATION (OR, "WHAT DOES IT EMULATE?").....	39
9.6.1 <i>Archon</i>	39
9.6.1.1 <i>Command List</i>	39



9.6.1.2	Timing.....	39
9.6.1.3	Image Data.....	40
9.6.1.4	Parameters and Configuration Memory.....	40
9.6.2	ARC.....	40
10.	APPENDIX B – OPEN ISSUES	40

DRAFT



TABLE OF FIGURES

Figure 1. Detector Controller Server Interfaces	9
Figure 2. Example server configuration files	12
Figure 3. Example shell script utilizes sendsockcmd to send ASCII strings to a TCP/IP socket.....	16

LIST OF TABLES

Table 1. Server configuration keywords	13
Table 2. Automatically-generated FITS keywords	30
Table 3. ASCII string return values for specified ARC return codes.....	35
Table 4. Emulator configuration keywords	37



1. INTRODUCTION

1.1 BACKGROUND

COO has developed generic detector controller software suitable for operating both STA/Archon and Astronomical Research Camera (a.k.a. "Leach") detector controllers. Regardless of the type of detector (e.g. CCD, IR) a detector controller must perform the same set of functions. These include configuring the controller hardware (load waveforms, set biases, etc.), setting up for an exposure (set geometry, exposure time, etc.), initiating an exposure, reading pixels and saving them in a FITS file. The detector controller software is a server which provides this functionality to a client.

A variety of clients may connect to the server via standard sockets, send ASCII character based commands to access all possible detector functions, and receive back ASCII character replies.

This document describes the interface between a client and the detector controller server so that any client can utilize it in any environment, whether that be in a lab/test/engineering setting or as part of a suite of other software modules of a deployed instrument in a larger observatory environment.

The intended audience for this document includes the following:

- detector engineers who wish to interface to a detector controller directly
- programmers who are integrating a detector controller into an instrument

1.2 SCOPE

The broken black line in Figure 1 illustrates the interface that this document describes, which is the client interface to the detector controller server (blue box in the figure).

The grey box at left is any detector controller, e.g. Archon or ARC. The interface between the detector controller server and the detector controller hardware is indicated by the green arrows, the details of which are beyond the scope of this document.

Possible clients are shown by the grey boxes to the right of the interface line. Details of clients suitable for communicating with the detector controller server is beyond the scope of this document, although suggestions and example clients will be given in Section 3.4.

Pixel data written by the detector controller server are shown crossing to the right of the interface because data can be written to any (shared) disk; no special interface is required to access the data, which are stored in standard FITS file format.

Commented [RMS1]: We need to clarify what is meant here by "user". Frankly I would prefer a different term. "Client" perhaps? The thing is that the human user is only the client when the system is operated directly from command line interface, but the more usual application is for there to be "Instrument Control System" acting as intermediary to the human customer, right? You seem to be defining "user" as a person or another piece of software but most readers think of the user as the *person* operating the instrument, normally an astronomer, you are providing quite low level functionality that will not normally be accessed directly. Even the detector engineer in the lab is likely to write a script to run an experiment.

Commented [RMS2]: Ok. I see now that you are acknowledging the distinction. If you lead off with "client", and perhaps even put this paragraph first it will address the problem.

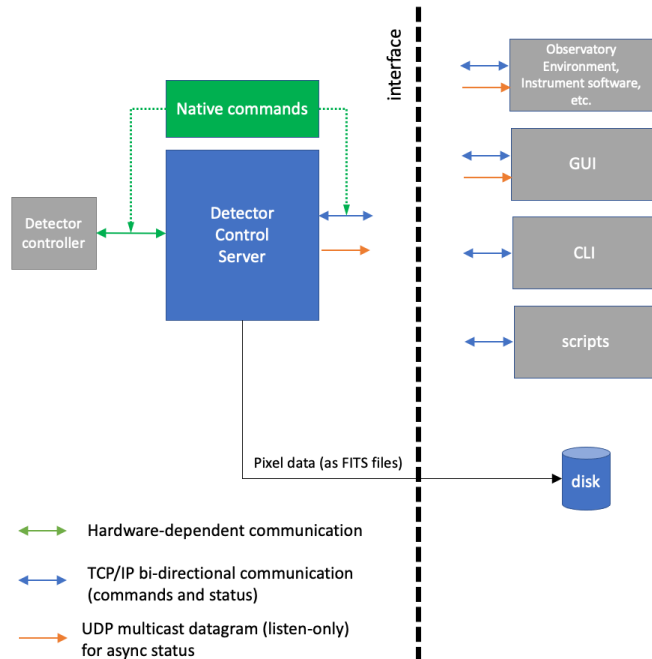


Figure 1. Detector Controller Server Interfaces

1.3

DOCUMENT OUTLINE

Section 1 is this introduction.

Section 2 contains worked examples for using the detector controller server.

The remainder of the document is reference material.

Section 3 describes the interface specifications including protocol, configuration and possible clients.

Section 4 describes the command syntax accepted by the interface.

Section 5 describes the FITS header handling by the server.

Section 7 identifies some commonly used native commands which are passed from the user through the interface directly to the controller, without interaction on the part of the detector controller server software.

Section 8 describes an optional detector controller emulator which can be used for software testing purposes, when a live detector controller is not available.



1.4 CHANGE RECORD

Revision	Date	Section	Modifications
REL01(DRF01)	XXXXX	All	XXXXX

1.5 ABBREVIATIONS, DEFINITIONS AND NOMENCLATURE

ACF	Archon Configuration File
API	Application Programming Interface
ARC	Astronomical Research Cameras, Inc. (a.k.a. "Leach")
BOI	Band of Interest
CIT	California Institute of Technology
CLI	Command Line Interface
COO	Caltech Optical Observatories
DMA	Direct Memory Access
DRP	Data Reduction Pipeline
FITS	Flexible Image Transport System
GUI	Graphical User Interface
ICD	Interface Control Document
ICS	Instrument Control Software
IP	Internet Protocol
OIR	(COO) Optical / Infrared Service Center
PCI/e	Peripheral Component Interconnect/express
ROI	Region of Interest
SNR	Signal to Noise Ratio
STA	Semiconductor Technology Associates, Inc.
TCP	Transmission Control Protocol
TCS	Telescope Control System
UDP	User Datagram Protocol
UTC	Universal Coordinated Time

2. WORKED EXAMPLES

2.1 COMPILATION

Edit the file CMakeLists.txt. The first few lines contain user definitions, as shown below.



```
# -----  
# BEGIN USER DEFINITIONS  
# -----  
# user must define the interface type here  
# valid types are: "AstroCam" | "Archon"  
  
set(INTERFACE_TYPE "Archon")  
  
# If your software is not installed in the indicated directory then change this:  
set(PROJECT_BASE_DIR $ENV{HOME}/Software/camera-interface)  
  
# This prints a large number of debugging messages and may be eliminated in future releases:  
#add_definitions(-DLOGLEVEL_DEBUG)
```

Set (or un-comment) the desired interface type using the syntax:

```
set(INTERFACE_TYPE "<TYPE>").
```

Allowable interface types are (as indicated in the file), **AstroCam** or **Archon**. As shown in the above example, this user has set the interface type to **Archon**.

Next, indicate where your software has been installed, using the syntax:

```
set(PROJECT_BASE_DIR $ENV{HOME}/Software/camera-interface)
```

Above, this user has installed the software in the Software/camera-interface/ sub-directory in their home directory.

Change to the build directory and run cmake and then make, as follows (keystrokes entered by the user are shown in **blue**):

```
[developer@localhost ~]$ cd Software/camera-interface/build  
[developer@localhost build]$ cmake ..  
[developer@localhost build]$ make
```

The executable binary programs are located in ~/Software/camera-interface/bin and for the remaining examples in this document it will be assumed that they are in the user's path.

2.2 CONFIGURATION

Example configuration files are shown in Figure 2 where an Archon-suitable configuration file is shown at left and one for AstroCam is shown at right. The `astrocam.cfg` example shown at right illustrates how to define `DEFAULT_FIRMWARE` as an array for multiple controllers. In this example the system supports four PCI cards.



```
# Example archon.cfg
#
ARCHON_IP=192.168.1.2
ARCHON_PORT=4242
DEFAULT_FIRMWARE=/home/acf/archon.acf
EXPOSE_PARAM=Expose
IMDIR=/data
BASENAME=image
NBPORT=3030      # non-blocking port
BLKPORT=3031     # blocking port
ASYNCPORT=1234   # asynchronous message port
ASYNCGROUP=239.1.1.234
LOGPATH=/home/logs

# Example astrocam.cfg supports 4 controllers
#
DEFAULT_FIRMWARE=(0 /home/dsp/E2V4240/tim.lod)
DEFAULT_FIRMWARE=(1 /home/dsp/E2V4240/tim.lod)
DEFAULT_FIRMWARE=(2 /home/dsp/E2V4240/tim.lod)
DEFAULT_FIRMWARE=(3 /home/dsp/E2V4240/tim.lod)
IMDIR=/Data/E2V4240
BASENAME=image
NBPORT=4000      # non-blocking port
BLKPORT=4001     # blocking port
ASYNCPORT=2345   # asynchronous message port
ASYNCGROUP=239.1.1.234
LOGPATH=/home/E2V4240/logs
```

Figure 2. Example server configuration files

2.3 RUNNING THE SERVER

It is assumed that the detector controller server has been compiled, that the executable is named "cameraserver" and that it is in the user's path, and that it is run in the foreground of its own terminal. Keystrokes entered by the user are shown in **blue**:

```
[developer@localhost ~]$ cameraserver /home/user/camera.cfg
2021-07-09T23:28:29.965163 (Camera::main) this version built Jul  9 2021 16:07:03
2021-07-09T23:28:29.965187 (Camera::main) 13 lines read from /home/user/camera.cfg
2021-07-09T23:28:29.965196 (Camera::Server::configure_server) applied 4 configuration
lines to server
2021-07-09T23:28:29.965283 (Archon::Interface::configure_controller)
applied 7 configuration lines to controller
```

2.4 THE SIMPLEST COMMAND-LINE EXPOSURE

The most simple example is taking a single exposure and saving it to a FITS file. It is assumed that the user is typing in a terminal on the host which is running the detector controller server and that the server is running in another terminal or as a daemon.

The keystrokes entered by the user are shown in **blue**:

```
[developer@localhost ~]$ telnet localhost 3031
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
open
DONE
load
DONE
expose
DONE
```



3. INTERFACE SPECIFICATIONS

3.1 PROTOCOL

The user interface to the detector controller server supports standard TCP/IP sockets. The interface uses simple mnemonic, English-looking ASCII-character commands and return values.

3.2 CONFIGURATION

User-configurable parameters are defined in a configuration file which is read upon start-up of the server. When the server is run as a daemon then the configuration file will also be read when the HUP signal is received. The configuration file is ASCII plain text formatted as:

```
KEY=VALUE # optional comment
```

Configuration keys also support arrays by specifying the configuration line as:

```
KEY=(IDX VALUE)
```

```
KEY=(IDX VALUE)
```

```
etc.
```

This will assign VALUE to index IDX in an array of keys of name KEY. Allowed keywords are shown in Table 1. Any text following the number sign "#" is ignored and may be used to include comments. Some of these keys can only be set in the configuration file and cannot be changed while the server is running, but others are simply defaults which can be readily changed or overridden by a command. Keys which can be changed or overridden via command are indicated in the 3rd column of Table 1 by the command which changes them.

Table 1. Server configuration keywords

KEY	Description	Command override?
ARCHON_IP	IP address for Archon controller (Archon-only) communications	no
ARCHON_PORT	Port number for Archon controller communications (Archon-only)	no
EXPOSE_PARAM	Archon parameter used to trigger an exposure (=1 to trigger exposure) (Archon-only)	no
SHUTENABLE_PARAM	Archon parameter used to enable/disable shutter opening on exposure (=1 to open on expose, =0 to not open on expose) (Archon-only)	no
HDR_SHIFT	Sets the number of bits to right-shift the Archon buffer when in HDR mode, i.e. SAMPLEMODE=1, 32 bits per pixel. (Archon-only)	hdrshift §4.3.18



DEFAULT_FIRMWARE	default firmware to load when load command is issued with no arguments	load §4.3.25
READOUT_TIME	detector readout time in msec	no
IMDIR	default base image directory	imdir §4.3.20
AUTODIR	set = "yes" or "no" to automatically create a UTC subdirectory for images, i.e. <i>IMDIR/YYYYMMDD/filename.fits</i>	autodir §4.3.3
BASENAME	default base image name for FITS files	basename §4.3.3
LOGPATH	fully qualified path to save log files	no
BLKPORT	blocking port number for the server	no
NBPORT	non-blocking port number for the server	no
ASYNCPOR	async message port number	no
ASYNCGROUP	asynchronous broadcast group	no
TM_ZONE	set to local to use local time instead of GMT defaults to GMT if this key is omitted	no
DAEMON	set to yes or no to run as a daemon or not The command line "-d" option will override a "no" here.	no

It is by convention only, and not a requirement that the server configuration file ends in ".cfg". There are no constraints on the configuration filename.

3.2.1 Time Zone

By default, the system will use Universal Coordinated Time (UTC) for all logging and time stamps. Adding the line "TM_ZONE=local" to the configuration file will cause the system to use local time, which may be useful for lab testing. The system will automatically include a FITS header keyword to indicate the time zone.



3.3 PORTS

Communication ports are defined in a configuration file as described in §3.2. The detector controller server uses three ports, a blocking port, a non-blocking port, and an asynchronous message port, defined by `BLKPORT`, `NBPORT`, and `ASYNCPORT`, respectively.

3.3.1 Blocking Port

When a client connects to the blocking port it will remain open as long as the client maintains its connection. This port can be used with Telnet (for example) to easily create a form of a command line interface. Only one command at a time can be received on this port. If the client tries to send a command before the previous command has completed, then it will be ignored.

3.3.2 Non-Blocking Port

The non-blocking port will accept a single command and immediately close the connection. The server spawns a new thread for each non-blocking connection so multiple commands can be received on this port. Since each non-blocking connection is handled by a separate thread of executing, this means that non-blocking port commands are processed in their own thread, as simultaneously as the host CPU allows. Note that the order of processing of commands in these independent threads is not guaranteed so one must take care when sending commands to the non-blocking port. If the order of execution must be guaranteed then the blocking port should be used.

Since each non-blocking connection spawns a separate thread, in order to prevent multiple clients from wasting resources by opening connections to the non-blocking port and not sending a command, all non-blocking port connections will time-out and automatically close after a period of 3 seconds.

3.3.3 Asynchronous Message Port

The asynchronous message port is a connectionless UDP multicast port. The server will multi-cast datagrams to the broadcast group defined in the configuration file by `ASYNCGROUP`. Responses to commands sent on the non-blocking port, lengthy responses to all commands, and other instantaneous status messages (such as exposure time remaining, etc.) will be sent to the async port. Asynchronous status messages are preceded by a message tag to indicate the type or source of the message. These tags are listed in Section 5.

3.4 CLIENTS

Any client which follows the TCP/IP protocol is capable of communicating with the detector controller server. Example clients might include scripting languages (MATLAB, Python, etc.), shell scripts, or even telnet, as described below.

3.4.1 Scripting languages

All common / popular scripting languages such as MATLAB, Python, etc. possess a TCP/IP stack and are capable of communicating with the server. In these instances it would be up to the user to develop an appropriate script for sending commands to, and receiving responses from the detector controller server, in accordance with this document.



3.4.2 Shell (e.g. csh, bash, etc.)

As another example, COO/OIR often uses a simple C-language based program called `sendsockcmd`¹ which accepts command line arguments to send an ASCII string to a specified host:port and then closes the connection. Combining this program with a shell script can make a quick and easy command-line tool for sending commands to the server, which can in turn be listed in a bash script for performing frequently used sequences.

Consider for example, the shell script shown in Figure 3. If the shell script from Figure 3 were saved under the filename "cam" then one could simply type (in blue):

```
[developer@localhost ~]$ cam open
```

which would send the "open" command to the localhost on port 2158. Similarly, entire shell scripts could be built around "cam" to send commands to the detector controller server.

```
#!/bin/csh
setenv camport 2158
setenv camhost localhost
setenv camtimeout 20000
set command = "$argv[1]"
  shift argv
while ($#argv > 0)
  set command = "$command $argv[1]"
  shift argv
end
$HOME/bin/sendsockcmd -h $camhost -p $camport "$command" -t $camtimeout
```

Figure 3. Example shell script utilizes `sendsockcmd` to send ASCII strings to a TCP/IP socket.

3.4.3 telnet

An example client can be as simple as using the `telnet` command which uses the Telnet application protocol. This can be useful as a quick check, or as a way to provide a simple, "no-code" CLI for test and engineering purposes by using telnet to connect to the blocking port. An example telnet session might look like the following (where the user types the strings in blue):

```
[developer@localhost ~]$ telnet localhost 3031
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^'.
open
DONE
load
DONE
buffer 4200 4200
35280000 DONE
geometry 4200 4200
4200 4200 DONE
```

¹ `sendcmd.c`, a 146-line program written by Marco Bonati continues to live on today



```
exptime 2000
2000 DONE
expose
DONE
```

3.5 DAEMONIZE

The user may choose to run the detector controller server as a daemon². This might be done to ensure that the server runs independently of user activity such as logins, logouts, accidentally closing a terminal, etc. To run the server as a daemon, add "-d" to the command line when starting the application or set DAEMON=yes in the server's configuration file.

Commented [HD3]: not yet implemented

3.6 COMPILATION

The type of detector controller to be used must be selected prior to compiling the detector controller server application. This information is also supplied with the software in the README.md file which is included in the git archive.

3.6.1 Requirements

Cmake 3.5 or higher
cfitsio and CCFits libraries (expected in /usr/local/lib)

3.6.1.1 For ARC controllers

C++17
g++ 8.3
ARC API 3.6 and Arc66PCle driver

3.6.1.2 For STA/Archon controllers

C++11
g++ 4.8 or higher

3.6.2 Build Instructions

To select the controller, edit the file CMakeLists.txt in the main camera-interface directory to uncomment one of the following two lines:

```
set(INTERFACE_TYPE "Archon")
#set(INTERFACE_TYPE "AstroCam")
```

by removing the number sign (#) from the desired controller, and inserting a number sign in front of the undesired controller (in the example above, the server will be built for Archon support). Change to the build directory. To start with a clean build, delete the contents of the build directory, including the subdirectory CMakeFiles/, but not the .gitignore file. For example:

```
[developer@localhost camera-interface]$ cd build
[developer@localhost build]$ rm -rf *
```

² A daemon is a background process which is not connected to a parent terminal.



Create the Makefile by running `cmake` (from the build directory) as:

```
[developer@localhost build]$ cmake ..
```

then compile the program

```
[developer@localhost build]$ make
```

The program can then be run with:

```
[developer@localhost build]$ ../bin/cameraserver <file.cfg>
```

4. COMMANDS

4.1. COMMAND FORMAT

In general, commands which require a parameter and value set that parameter to that value. Stating the parameter only, without the value, initiates a query to return the current value, while inclusion of a value implies setting the named parameter to the specified value. Thus, with one exception the "set" and "get" operations are inherent and not sent as separate commands. For example, rather than sending commands like:

"set exptime 100" and "get exptime"

one would *instead* send only:

"exptime 100" and "exptime"

where the former sets the exposure time to 100 and the latter queries the current exposure time.

The exception to this design rule is the `getp` and `setp` commands (Sections 4.3.15 and 4.3.30) used to get and set Archon parameters, respectively. These commands are used to avoid possible confusion between server commands and what might be an allowable Archon parameter name.

Any command received that is not among the syntax in Section 4.3 is assumed to be a native controller command and passed directly to the controller. This allows a user to interact directly with the controller. Every effort has been made to avoid mirroring native and server commands, but in the event that is unavoidable then sending the command in upper case should indicate that it is to be a controller-native command. In general, commands are sent as lower case, even native controller commands.

4.2. RETURN VALUES

Return values are ASCII strings. In general, the response to commands is either `DONE` (on success) or `ERROR` (on error). If the command required or requested a value then on success that value is returned first, before the `DONE`. The command syntax described in Section 4.3 describes in detail what is returned for each command.



4.3 COMMAND SYNTAX

This section describes the syntax recognized by the server, listed in alphabetical order.

Where a command can be used for setting or getting a parameter, the parameter value is shown as an optional argument in [square brackets]. Required arguments are shown in <angled brackets>. Either-or arguments are separated by the "vertical bar" | symbol. Commands sent to, and responses returned from the server are shown in `courier` font. Non-literals are shown in *italics*; e.g., "basename [*value*]" means that a value should be supplied, not the word "*value*" but "autodir [yes|no]" means that the literal word "yes" or "no" should be supplied. Commands accepted by only one controller or the other are indicated appropriately as ARC or Archon.

4.3.1 abort – abort an exposure in progress (ARC-only)

usage: abort

returns: DONE on success

ERROR on error

Aborts an exposure currently in progress. This can abort any stage, the exposure delay, readout, etc. Currently only available on ARC (AstroCam).

Commented [HD4]: Steve -- might there be a way to abort an Archon exposure? Would require supporting ACF file.

4.3.2 amplifier – set or get which detector amplifier(s) to use

usage: amplifier [*ampsel*]

returns: *ampsel* DONE on success

ERROR on error

Set or get the amplifier(s) to use. When an argument <ampsel> is included then the amplifier selection will be <ampsel>. When no argument is supplied then the current amplifier selection will be returned.

Commented [HD5]: Not currently implemented

4.3.3 autodir – automatically create UTC subdirectory for images

usage: autodir [yes | no]

returns: yes | no DONE on success

This command controls whether images will be saved into a UTC date subdirectory within the image directory imdir (see §4.3.20). If yes then the subdirectory will automatically be created if it doesn't exist. The format of this directory is YYYYMMDD so that images would be saved under /*imdir*/YYYYMMDD/*basename_suffix.fits*. A default state can also be set in the configuration file with the key AUTODIR=yes | no (see §3.2). If not specified and omitted from the configuration file then the default state is "yes".

When no argument is supplied then the current state is returned.

See also the imdir (4.3.20), basename (4.3.4), and fitsnaming (4.3.14) commands.

4.3.4 basename – base image name for FITS image files

usage: basename [*value*]

returns: *value* DONE on success

ERROR on error

Set or get the base image name for image files. When an argument <value> is included then the image name is set to <value>. When no argument is supplied then the current image name is returned. A default can also be set in the configuration file with the key BASENAME=*value* (see §3.2).



FITS image files are saved to `/imdir/[YYYYMMDD]/basename_suffix.fits`, where the UTC date subdirectory YYYYMMDD is optionally set by `autodir` (see §4.3.3 and Table 1). See also the `imdir` (4.3.20) and `fitsnaming` (4.3.14) commands.

4.3.5 **bias – bias voltage**

usage: `bias <module> <channel> [bias]` (Archon)

usage: `bias <pcinum> <boardid> <dac> <CLK|VID> <adu>` (ARC)

returns: `bias` DONE on success

ERROR on error

Set a bias voltage.

For Archon, specify module number {1:12} and channel {1:30} to read the specified bias. Include a voltage (in floating point volts, E.G. 14.0, -5.5, etc.) to write that voltage to the specified module and channel. The software automatically checks the board type (HVBias, HVXBias, LVBias, LVXBias) and applies limit checks accordingly.

For ARC, specify PCI device `pcinum`={0,1,2,...}, boardid {0:15}, DAC number {0:7}, the string "CLK" or "VID" for the type of bias board (clock or video), and the voltage in A/D units, `adu` {0:4095} which you must scale to the maximum output voltage of the system. E.G. for a 3.3V system, `adu` = 4095×voltage / 3.3.

Commented [HD6]: reading biases for ARC not currently implemented, so all arguments are required (as this is write-only)

4.3.6 **buffer – PCI/e image buffer mapping (ARC-only)**

usage: `buffer [pcinum] [cols rows | size]`

returns: `size` DONE on success

ERROR on error

This command will allocate PCI/e buffer space for performing the DMA transfers, for ARC only. If no argument is given then the size of the currently mapped buffer (in bytes) will be returned.

If a single value `<size>` is given, then a buffer of `<size>` bytes will be allocated and mapped to the PCI/e device.

If two values `<cols>` `<rows>` are given then a suitably sized buffer will be allocated to contain an image of those dimensions.

For systems with more than one installed PCI/e device, the PCI device number `[pcinum]` is optional. If specified then the buffer for that device only will be set (or returned). If not specified then all PCI/e buffers will be set (or returned). For systems with only one PCI/e device, this is always omitted.

4.3.7 **cds – CDS/Deinterlace configuration key (Archon-only)**

usage: `cds <key> [value]`

returns: `value` DONE on success

ERROR on error

4.3.8 **close – close the connection to the detector controller**

usage: `close`

returns: DONE on success

ERROR on error



Closes the connection to the controller. For Archon this closes the TCP sockets; for ARC (AstroCam) this closes the PCI/e driver(s).

4.3.9 **datacube – data cube state**

usage: datacube [true | false]

returns: true | false DONE

When provided with the (optional) state of "true" or "false" (case-insensitive) this will set the datacube state for multiple exposures. When the datacube state is true, multiple exposures will be written as FITS-formatted multi-extension data cubes; when false, separate files will be created for each exposure. See "expose N" in §4.3.12 below.

When no state is supplied then the current state is returned.

The default state is false.

4.3.10 **echo – test server communication**

usage: echo <string>

returns: string DONE

The server will write ("echo") any string received back to the connected socket. This is used for testing communication with the server without requiring any detector controller.

4.3.11 **exit – exit the server**

usage: exit

returns: none

Cleanly closes all connections and exits the server. Note that if the server is run in the foreground, it captures Ctrl-C which will execute this command to cleanly shut down before exiting.

4.3.12 **expose – initiate an exposure**

usage: expose [N]

returns: DONE on success

ERROR on error

Initiate an exposure. The previously set exptime will be used for the exposure time for this image. The command blocks other commands while the exposure is executing and does not return until the image (or optionally-specified [N] images) has been written to disk. If the optional argument [N] is included then it will repeat for [N] exposures and additional exposures will be written to either separate files or in a single data cube, according to the datacube state (see §4.3.9 above).

4.3.13 **exptime – exposure time**

usage: exptime [value]

returns: value <unit> DONE on success

ERROR on error

Set or get the exposure time. When an argument <value> is included then the exposure time is set to <value>. When no argument is supplied then the current exposure time is returned. The units are specified by <unit>. For ARC only msec is possible but for Archon <unit> may be sec or msec, as determined by the state of **longexposure** (§4.3.27). The default unit is msec.



For Archon this requires an ACF which supports an appropriate exposure delay subroutine, and that the exposure delay is controlled by a parameter named "exptime". Since the Archon allows for 20-bit values the largest exposure delay is 0x1FFFFFF or 2,097,151. Represented in milliseconds this equates to a maximum exposure delay of 34.95 minutes. To accommodate longer exposure delays (at the cost of resolution) the command **longexposure** (§4.3.27) will change the Archon exposure delay unit to seconds.

For ARC this requires that the exposure timer is set using the 3-letter command "SET". Since the ARC controller allows for 24-bit values the largest exposure time is 0x1FFFFFF or 33,554,431. Represented in milliseconds this equates to a maximum exposure delay of 9.32 hours.

4.3.14 **fitsnaming – FITS filename format**

usage: **fitsnaming** [*type*]

returns: *type* DONE on success

ERROR on error

Set or get the FITS file naming type. When no parameter is specified the current type is returned. Valid types are "time" and "number".

When <type> is number then the FITS files are saved as:

imdir/[YYYYMMDD]/*basename_imnum.fits*

where imnum is an incremental number which starts at 0000 each time the server is started and increments automatically with each successful exposure, and [YYYYMMDD] is an optional date subdirectory (see "AUTODIR" in Table 1 and §4.3.3).

When type is time then the FITS files are saved as

imdir/[YYYYMMDD]/*basename_YYYYMMDDHHMMSS.fits*

where YYYYMMDDHHMMSS represents the time of the exposure (to the resolution of the current second).

See also the *imdir* (§4.3.20) and *basename* (§4.3.3) commands.

When multiple controllers are in use, *basename* is replaced with *basename_N* where N is the controller number {0, 1, 2, ...} in order to properly correlate the file with the controller.

4.3.15 **framestatus – read the frame status (Archon-only)**

usage: **framestatus**

returns: DONE on success

ERROR on error

This command prints the Archon frame buffer status to the log file in a human-friendly format similar to that which is displayed in the STA archongui. This is the information obtained from the Archon-native "FRAME" command. Since the frame status for all internal buffers is displayed, there are no return values (other than a possible error sending the FRAME command) and this is considered more of a diagnostic, visual command.

4.3.16 **geometry – detector geometry (ARC-only)**

usage: **geometry** [*cols rows*]

returns: *cols rows* DONE on success

ERROR on error

Set or get the detector geometry for ARC (AstroCam) only.

Commented [HD7]: Is this needed? We have the Archon-native FRAME command. In any case, this is NOT YET IMPLEMENTED.



When two arguments are specified then set the cols and rows (respectively) on the detector controller to those specified. When no arguments are given then return the cols and rows.

This command writes the cols to Y: memory address 0x400001 and the rows to Y: memory address 0x400002.

4.3.17 **getp – get parameter value (Archon-only)**

usage: `getp <parameter>`

returns: *value* DONE on success
ERROR on error

Get a parameter value for Archon only.

When the argument *<parameter>* is a stored parameter name, this command will return the associated value stored in the controller's configuration memory. See also `setp` (§4.3.30).

4.3.18 **hdrshift – HDR bit-shift value (Archon-only)**

usage: `hdrshift [value]`

returns: *value* DONE on success
ERROR on error

Gets [sets] the number of bits to right-shift the Archon buffer when in HDR mode (i.e. `SAMPLEMODE=1`, 32 bits per pixel). The default is 16.

This can also be set in the config file with the parameter:

`HDR_SHIFT=value`

4.3.19 **heater – control HeaterX module (Archon-only)**

There are several forms for the heater command, as described in the following subsections. For each command, ERROR is returned on error. *<module>* refers to the (integer) module number. The heater ID *<A|B>* and requested state *<on|off>* are not case-sensitive. When no optional arguments are provided then both the state and target are returned. When setting a parameter then only the parameter(s) set is(are) returned.

4.3.19.1 Set or get the state and target for heater A or B on the specified module.

usage: `heater <module> < A | B > [<on | off> <target>]`

returns: ON | OFF *target* DONE on success

When state is set to *<on>* then heater A or B on the specified module is enabled and the target temperature is set to *<target>*. When set to off then heater A or B on the specified module is disabled. The target alone may be set, the state alone may be set, or the target may be set at the same time with the ON state (e.g. `heater 1 A ON -205.0`).

4.3.19.2 Set or get the PID parameters for heater A or B on the specified module.

usage: `heater <module> < A | B > PID [<p> <i> <d>]`

returns: *p i d* DONE on success

When setting parameters, all three must be provided in the order indicated, *<p> <i> <d>*. Fractional PIDs are supported with backplane version 1.0.1054 or newer.

4.3.19.3 Set or get the ramp and ramprate for heater A or B on the specified module.

usage: `heater <module> < A | B > RAMP [<on | off> [ramprate]]`



returns: ON | OFF *ramprate* DONE on success

When setting the ramp to <on> then the [*ramprate*] is required and requested ramprate is returned. When setting ramp to <off> then the ramprate is not included, and "0" is returned for the ramprate.

The ramprate alone may be set (e.g. *heater 1 A RAMP 300*), the state alone may be set, or the ramprate may be set at the same time with the ON state (e.g. *heater 1 A RAMP ON 300*).

The RAMP command requires backplane version 1.0.548 or newer.

4.3.20 **imdir – base image directory for FITS image files**

usage: *imdir* [*name*]

returns: *name* DONE on success
ERROR on error

Set or get the base image directory, in which to save files. When an argument <*name*> is included then the base image directory is set to <*name*>. When no argument is supplied then the current image directory is returned.

This sets and returns the *base* directory only. Images can be optionally saved in a date subdirectory of the base image directory, using the current UTC date, i.e., *imdir*/[*YYYYMMDD*]/ either by default (Table 1) or by command (*autodir*, §4.3.3).

4.3.21 **imnum – image number for FITS image files**

usage: *imnum* [*value*]

returns: *value* DONE on success
ERROR on error

Set or get the image number, which is appended to the image base name (e.g. *basename_0001*) when *fitsnaming* is set to "number" (see §4.3.14). When an argument <*value*> is included then the image number is set to <*value*>. When no argument is supplied then the current image number is returned. The image number is automatically incremented after a successful exposure.

4.3.22 **inreg – write to VCPU input register (Archon-only)**

usage: *inreg* <*module*> <*reg*> <*value*>

returns: DONE

This command writes <*value*> to the VCPU input register <*reg*> on module <*module*>. The selected module must support digital IO and contain VCPU support. Register <*reg*> must be in the range {0:15} and <*value*> must be in the range {0:65535}. The Archon command "APPLYDIOM" is sent automatically after successfully writing the configuration key "MODULExxxxMODm/VCPU_INREGi=vvvvv".

4.3.23 **interface – return the detector interface type**

usage: *interface*

returns: *type* DONE

This command returns a string indicating the type of detector controller interface for which the camera interface software has been compiled.



4.3.24 **key – add FITS keyword to user-defined database**

usage: `key < KEYWORD=VALUE//COMMENT | list >`

returns: DONE on success

ERROR on error

This command accesses an internal database to include user-defined FITS keyword in the primage image header. This internal database is held in RAM while the server is running and is not independently saved.

When the argument is of the following form:

`KEYWORD=VALUE//COMMENT`

then a new keyword=`KEYWORD` equal to value=`VALUE` and optional comment=`COMMENT` is inserted (or updated, if keyword already exists) in the internal database. The type (STRING, INT, FLOAT) is automatically detected. The comment is optional and may be omitted along with the slashes (i.e. "key `KEYWORD=VALUE`" is acceptable).

If `VALUE=.` (a period) then that keyword will be deleted from the user database.

If the optional argument "list" is passed then all user-defined keywords will be printed in the logfile.

Commented [HD8]: Future: keyword database list should be output to status port

4.3.25 **load – load firmware into detector controller**

usage: `load [filename]`

returns: DONE on success

ERROR on error

Load firmware onto the controller from the file specified. If filename argument is included then it must be a fully qualified pathname. This would be an ACF file for STA/Archon, or a .lod file for an ARC timing board.

If the optional filename argument is omitted then the default firmware specified in the server's .cfg file will be loaded.

For AstroCam, the API loadControllerFile() is used to upload the .lod file over the fiber to the timing board.

For Archon, the configuration memory is first cleared, then written (WCONFIG) from the specified .ACF file, after which Archon will parse and apply the complete system configuration (APPLYALL). Power to the detector will be off after this operation. The DEFAULT mode is automatically selected (see "mode" §4.3.28).

4.3.26 **loadtiming – load timing script and parameters (Archon-only)**

usage: `loadtiming [filename]`

returns: DONE on success

ERROR on error

Loads the specified ACF file into the Archon configuration memory (WCONFIG), then sends the LOADTIMING command. This parses and compiles the timing script and parameters contained in the configuration memory, and applies them to the system. This resets the timing cores.

If filename argument is included then it must be a fully qualified pathname.

If the optional filename argument is omitted then the default firmware specified in the server's .cfg file will be loaded.

**4.3.27 longexposure – long exposure mode to extend exptime unit (Archon-only)**

usage: longexposure [true | false]
returns: [true|false] DONE on success
ERROR on error

This command applies only to STA/Archon.

When set to true the controller server assumes that the exposure time unit is in seconds. When set to false the exposure time unit is assumed to be in milliseconds.

This command requires compatible Archon firmware (ACF) with a parameter named "longexposure" which will be set =1 when longexposure is true, and =0 when longexposure is false. The ACF must also have appropriate exposure delay subroutines for both cases, to produce second and millisecond delays.

The default start-up condition of the server is longexposure=false.

The comment field of the EXPTIME header keyword is controlled by this value and will be set to "msec" or "sec" (for ARC the exposure time comment will always indicate "msec").

4.3.28 mode – camera mode (Archon-only)

usage: mode [modename]
returns: modename DONE on success
ERROR on error

This command (currently) applies only to STA/Archon. Modes are defined in ACF file as [MODE_modename]. Every ACF file must contain at least one mode called DEFAULT and that mode must define the following three values:

```
[MODE_DEFAULT]  
ARCH:NUM_DETECT=x  
ARCH:HORI_AMPS=y  
ARCH:VERT_AMPS=z
```

where x, y, z are the number of detectors, number of horizontal amplifiers per detector, and vertical amplifiers per detector, respectively.

4.3.29 open – open a connection to the detector controller

usage: open [devlist]
returns: DONE on success
ERROR on error

Opens a connection to the controller by whatever means is supported by the hardware in order to establish a communications channel between the host computer and the controller. This is required before any other operation.

An optional space-delimited device list [devlist] can be supplied to open only the specified devices (currently ARC-only but Archon support for multiple devices is planned).

4.3.30 sensor – set or get temperature sensor current (Archon-only)

usage: sensor <module> < A | B | C > [<current>]
returns: <current> DONE on success
ERROR on error

Commented [HD9]: Should there be AstroCam support for modes? NGPS would be the next customer. May be too early to tell. Maybe mark this as "resolved", leave modes to Archon-only for the time being, and revisit if needed.



This command sets or gets the temperature sensor current for the specified sensor (A,B,C) on the specified module. *<module>* refers to the (integer) module number. *<current>* is specified in nano-amps. This is used only for RTDs.

4.3.31 **setp – set parameter value (Archon-only)**

usage: setp *<name>* *<value>*

returns: *value* DONE on success

ERROR on error

Set a parameter *<name>* to value *<value>* for Archon only. This is the equivalent of Archon native commands FASTPREPPARAM followed by FASTLOADPARAM. See also getp (§4.3.15). Note that this does not re-write configuration memory, so a getp command will not read changes made by setp. See also writep (§4.3.35).

4.3.32 **shutter – set or get the shutter enable state on expose**

usage: shutter [enable | disable]

returns: ENABLED | DISABLED DONE on success

ERROR on error

Set or get the shutter enable state on expose. "1" or "0" also works in place of "enable" or "disable", respectively. When enabled, the shutter is allowed to open on expose; when disabled, the shutter is not allowed to open on expose. When no argument is given then the current enable state is returned. Requires firmware support.

4.3.33 **systemfile – generate a system file (Archon-only)**

usage: systemfile *<filename>*

returns: DONE on success

ERROR on error

Sends the Archon-native SYSTEM command to read the current hardware and generate an ASCII text file containing the system information. Such a file may be read by WDL or used for the emulator described in Section 8.

Commented [HD10]: NOT YET IMPLEMENTED

4.3.34 **useframes – set or get the useframes flag (ARC-only)**

usage: useframes [true | false]

returns: *state* DONE on success

ERROR on error

When an argument [true | false] is passed then set the "useframes" flag to true or false; when no argument is passed then return the current useframes flag. This command is for ARC (AstroCam) only. Not all ARC firmware supports frames (some firmware sends only pixels). You must set useframes to false if your firmware does not support sending of frames.

The state of useframes is true by default on start-up.

4.3.35 **writekeys – when to write FITS header keywords**

usage: writekeys [before | after]

returns: *state* DONE on success

ERROR on error



When an argument [before | after] is passed then set when the server will write user-defined FITS header keywords in relation to the exposure, before or after. When set to "before" then user-defined keywords are written to the header before the exposure; any new keyword settings received after the exposure will be applied to a future exposure. When set to "after" then user-defined keywords are written to the header after the exposure; this allows the user to utilize an exposure delay to collect FITS header information.

When no argument is supplied then return the current state.

The state of writekeys is "before" by default on start-up.

4.3.36 **writep – write parameter value to configuration memory (Archon-only)**

usage: writep <name> <value>

returns: value DONE on success

ERROR on error

Write a parameter <name> = <value> to Archon configuration memory. This is the equivalent of Archon native command WCONFIGxxxxPARAMNAME=VALUE. See also getp (§4.3.15) and setp (§4.3.31).

Commented [HD11]: NOT YET IMPLEMENTED

4.4 TO-DO

I don't want to forget about Steve's list of commands, but I haven't thought much yet about how to include them; the following commands need to be included above:

4.4.1 vertical binning

4.4.2 horizontal binning

4.4.3 region of interest

4.4.4 erase userkey DB

Commented [HD12]: Steve: We need to talk so I understand exactly what has to happen in order to implement these. They need supporting firmware, for sure.

5. ASYNCHRONOUS MESSAGES

Asynchronous messages are broadcast on the asynchronous message port. See Section 3.3.3.

5.1 ASYNCHRONOUS MESSAGE TAGS

Each message is preceded by one of the following tags, as in "TAG:message". This allows a listening client to look for only messages that it might be interested in, by looking for a particular tag and ignoring others.

5.1.1 **DATAcube:n COMPLETE | ERROR**

When datacube is true (cf. §4.3.9) this message is sent after each cube has been written to a FITS extension. The number *n* increments {1, 2, 3, ...} for each extension. If the write was successful then the message terminates with COMPLETE, or ERROR if an error occurred writing the frame.



- 5.1.2 EXPOSURE : *n*** (Archon-only)
Sent during exposure delay to indicate exposure progress, indicates the remaining exposure delay *n*, in the chosen units (cf. Longexposure §4.3.27).
- 5.1.3 EXPOSURE_d : *n*** (ARC-only)
Sent during exposure delay to indicate exposure progress, indicates the remaining exposure delay *n*, in msec for device *d*.
- 5.1.4 FILE:<fully_qualified_path> COMPLETE | ERROR**
This message is sent upon completion of each FITS file written, indicating the fully qualified filename. If the write was successful then the message terminates with COMPLETE, or ERROR if an error occurred writing the file.
Note that in the case of data cubes, this message is in addition to DATACUBE (§5.1.1) which would generate a DATACUBE: message for each cube, plus a FILE: message when all cubes have been written and the file is closed.
- 5.1.5 LINECOUNT : *n*** (Archon-only)
Sent during detector readout to indicate read progress, indicates number of lines read, *n*.
- 5.1.6 NOTICE : *message***
This tag is used for general informational, non-error but important messages. Unlike the activity log, this tag is used sparingly. For example, if a user neglects to set something where a default must be assumed, the system may alert the user via an async message using this tag.
- 5.1.7 PIXELCOUNT : *n*** (ARC-only)
Sent during detector readout to indicate read progress, indicates number of pixels read, *n*.
- 5.2 ARCHON NATIVE COMMANDS**
All Archon native commands will produce an asynchronous status message of the following format:

```
CMD:BEGIN  
CMD:<reply>  
CMD: ...  
CMD:END
```


where *CMD* is the native command and <reply> is the reply from that command, which may be carried onto multiple lines. For any reply which contains multiple responses, each will be returned on a separate line, and the :BEGIN and :END denotes the beginning and ending of the reply. Examples of these are shown in Section 7.1.

6. FITS HEADERS

6.1 BUILT-IN KEYWORDS

The server writes the keywords shown in Table 2.



Table 2. Automatically-generated FITS keywords

KEY	Comment
SIMPLE	file does conform to FITS standard
BITPIX	number of bits per data pixel
NAXIS	number of data axes
NAXIS1	length of data axis1
NAXIS2	length of data axis2
EXTEND	FITS dataset may contain extensions
SERV_VER	server build date
FILENAME	this filename
EXPTIME	exposure time in <unit> (see §4.3.27)
TM_ZONE	time zone, "local" or "UTC" (see §3.2.1)

6.2 HEADER DATA COLLECTION TIMING

Currently, FITS header data collection is performed prior to the exposure, such that the state of header keywords is "locked-in" and saved at the time the exposure starts; changes made to FITS header keywords after the exposure starts are not applied to the current exposure. A future feature will allow changing when the keywords are saved.

7. NATIVE CONTROLLER COMMANDS

Anything received by the server which is not a recognized command listed in Section 4.3 will be interpreted as a native detector controller command. A native command is a command which is "native" or inherent to the particular detector controller and is sent directly to the controller using the controller's specific syntax.

This functionality is illustrated by the green box in Figure 1. Although the figure shows a path completely circumventing the detector controller server, they are certainly received by the server; this is meant to illustrate that these commands have no corresponding "handling" function and are passed directly to the controller by the server and any return value comes directly from the controller. Other than proper formatting, the server does not interact with the command nor the return value.

In particular for the AstroCam controller, when the server returns `DONE` or `ERROR` that indicates the server's success or failure to send the command and receive a return value only. Therefore it is entirely possible for a native command to return an error and the server still report `"DONE"` (as opposed to `"ERROR"`). The client must examine the controller's return value to determine whether the controller returned an error or not.

Since the server does not interact with native commands or their return values, this means that if a native command is sent which changes the image dimensions, for example, then the server would not know about the new dimensions. Therefore, it's possible to produce undesirable behavior and in general, use of native commands is typically limited for use by detector engineers for engineering and testing purposes. However, in some cases (such as



the three query-only native Archon commands outlined in §7.1.1, §0, §7.1.3) native commands might be very useful for reading values directly from the controller which might be needed for a FITS header. Stated another way, query-only native commands are safe to use, while native commands that cause a change in the controller should be used with extreme caution.

For the STA/Archon controller (§7.1), the server will add the required preamble and check only whether or not the return value contained an error. For the ARC (Leach) controller (§7.2), the server will enforce only that the maximum number of arguments is not exceeded, and that the command has exactly 3 letters.

7.1 STA/ARCHON

This section is not meant to be an exhaustive list of Archon commands but only an example, and will list the three native query-only commands that might be of value for most users. Please see the Communications section of the Archon manual for additional details.

Commands to the Archon controller are of the form:

>xxCOMMAND

Commands begin with a greater-than symbol, followed by a two hexadecimal digit reference number, followed by the command itself, and terminated with a newline character ("\n" or ASCII 10, 0x0A). *The detector controller server creates all of this preamble (in blue) for the user; the user need only to send COMMAND to the server.*

Archon's response to this command will be tagged with the supplied hexadecimal reference number xx. The server will verify that a response with the corresponding reference was received and will return that response to the user. Lengthy responses will be sent to the async message port only, in a format suitable for parsing by client applications.

The example commands described here are FRAME, STATUS, SYSTEM.

7.1.1 FRAME

The output from the native FRAME command is written to the async message port in the following format, minus the comments in grey:

```
FRAME:BEGIN
FRAME:TIMER=x           ; Current hexadecimal 64-bit internal timer
FRAME:RBUF=d            ; Current buffer number locked for reading
FRAME:WBUF=d            ; Current buffer number locked for writing
FRAME:BUFnSAMPLE=d      ; Buffer n sample mode, 0: 16 bit, 1: 32 bit
FRAME:BUFnCOMPLETE=d    ; Buffer n complete, 1: buffer ready to read
FRAME:BUFnMODE=d        ; Buffer n mode, 0: top, 1: bottom, 2: split
FRAME:BUFnBASE=d        ; Buffer n base address for fetching
FRAME:BUFnFRAME=d       ; Buffer n frame number
FRAME:BUFnWIDTH=d       ; Buffer n width
FRAME:BUFnHEIGHT=d      ; Buffer n height
FRAME:BUFnPIXELS=d      ; Buffer n pixel progress
FRAME:BUFnLINES=d       ; Buffer n line progress
FRAME:BUFnRAWBLOCKS=d   ; Buffer n raw blocks per line
FRAME:BUFnRAWLINES=d    ; Buffer n raw lines
FRAME:BUFnRAWOFFSET=d   ; Buffer n raw offset
FRAME:BUFnTIMESTAMP=x   ; Buffer n hexadecimal 64-bit time stamp
FRAME:BUFnRETIMESTAMP=x ; Buffer n trigger rising edge time stamp
FRAME:BUFnFETIMESTAMP=x ; Buffer n trigger falling edge time stamp
```



```
FRAME:BUFnREATIMESTAMP=x ; Buffer n trigger A rising edge time stamp
FRAME:BUFnFEATIMESTAMP=x ; Buffer n trigger A falling edge time stamp
FRAME:BUFnREBTIMESTAMP=x ; Buffer n trigger B rising edge time stamp
FRAME:BUFnFEBTIMESTAMP=x ; Buffer n trigger B falling edge time stamp
FRAME:END
```

7.1.2 STATUS

The output from the native STATUS command is written to the async message port in the following format, minus the comments in grey:

```
STATUS:BEGIN
STATUS:VALID=n ; n = 1 if remaining status fields are valid
STATUS:COUNT=n ; Number of times system status has been updated
STATUS:LOG=n ; Number of log entries available
STATUS:POWER=n ; Power status. Possible values:
; 0: Unknown - usually an internal error
; 1: Not Configured - no configuration applied
; 2: Off - power to the CCD is off
; 3: Intermediate - some modules have enabled
; power to the CCD, some have not
; 4: On - Power to the CCD is on
; 5: Standby - System is in standby
STATUS:POWERGOOD=n ; n = 1 when system power supply is good
STATUS:OVERHEAT=n ; n = 1 when system is overheating
STATUS:BACKPLANE_TEMP=f ; Floating point backplane temperature in C
STATUS:P2V5_V=f ; +2.5V system supply voltage in V
STATUS:P2V5_I=f ; +2.5V system supply current in A
STATUS:P5V_V=f ; +5V system supply voltage in V
STATUS:P5V_I=f ; +5V system supply current in A
STATUS:P6V_V=f ; +6V system supply voltage in V
STATUS:P6V_I=f ; +6V system supply current in A
STATUS:N6V_V=f ; -6V system supply voltage in V
STATUS:N6V_I=f ; -6V system supply current in A
STATUS:P17V_V=f ; +17V system supply voltage in V
STATUS:P17V_I=f ; +17V system supply current in A
STATUS:N17V_V=f ; -17V system supply voltage in V
STATUS:N17V_I=f ; -17V system supply current in A
STATUS:P35V_V=f ; +35V system supply voltage in V
STATUS:P35V_I=f ; +35V system supply current in A
STATUS:N35V_V=f ; -35V system supply voltage in V
STATUS:N35V_I=f ; -35V system supply current in A
STATUS:P100V_V=f ; +100V system supply voltage in V
STATUS:P100V_I=f ; +100V system supply current in A
STATUS:N100V_V=f ; -100V system supply voltage in V
STATUS:N100V_I=f ; -100V system supply current in A
STATUS:USER_V=f ; User system supply voltage in V
STATUS:USER_I=f ; User system supply current in A
STATUS:HEATER_V=f ; Heater system supply voltage in V
STATUS:HEATER_I=f ; Heater system supply current in A
STATUS:FANTACH=n ; Fan speed in RPM (Rev F only)
STATUS:MODm/TEMP=f ; Floating point module m temperature in C
```




STATUS:MODm/LVLC_Vn=f ; LV(X)Bias only: Floating point module *m* low
; voltage low current *n* voltage reading in V
; *n* = 1 to 24 maps to LV1 to LV24

STATUS:MODm/LVLC_In=f ; LV(X)Bias only: Floating point module *m* low
; voltage low current *n* current reading in mA
; *n* = 1 to 24 maps to LV1 to LV24

STATUS:MODm/LVHC_Vn=f ; LV(X)Bias only: Floating point module *m* low
; voltage high current *n* voltage reading in V
; *n* = 1 to 6 maps to LV25 to LV30

STATUS:MODm/LVHC_In=f ; LV(X)Bias only: Floating point module *m* low
; voltage high current *n* current reading in mA
; *n* = 1 to 6 maps to LV25 to LV30

STATUS:MODm/HVLC_Vn=f ; HV(X)Bias only: Floating point module *m* high
; voltage low current *n* voltage reading in V
; *n* = 1 to 24 maps to HV1 to HV24

STATUS:MODm/HVLC_In=f ; HV(X)Bias only: Floating point module *m* high
; voltage low current *n* current reading in mA
; *n* = 1 to 24 maps to HV1 to HV24

STATUS:MODm/HVHC_Vn=f ; HV(X)Bias only: Floating point module *m* high
; voltage high current *n* voltage reading in V
; *n* = 1 to 6 maps to HV25 to HV30

STATUS:MODm/HVHC_In=f ; HV(X)Bias only: Floating point module *m* high
; voltage high current *n* current reading in mA
; *n* = 1 to 6 maps to HV25 to HV30

STATUS:MODm/TEMPA=f ; Heater(X) only: Floating point temperature
; sensor A reading in K

STATUS:MODm/TEMPB=f ; Heater(X) only: Floating point temperature
; sensor B reading in K

STATUS:MODm/TEMPC=f ; HeaterX only: Floating point temperature
; sensor C reading in K

STATUS:MODm/HEATERAOUTPUT=f ; Heater only: Floating point heater A
; output in V

STATUS:MODm/HEATERBOUTPUT=f ; Heater only: Floating point heater B
; output in V

STATUS:MODm/HEATERAP=d ; Heater only: Heater A P term contribution
; to PID loop (signed integer)

STATUS:MODm/HEATERAI=d ; Heater only: Heater A I term contribution
; to PID loop (signed integer)

STATUS:MODm/HEATERAD=d ; Heater only: Heater A D term contribution
; to PID loop (signed integer)

STATUS:MODm/HEATERBP=d ; Heater only: Heater B P term contribution
; to PID loop (signed integer)

STATUS:MODm/HEATERBI=d ; Heater only: Heater B I term contribution
; to PID loop (signed integer)

STATUS:MODm/HEATERBD=d ; Heater only: Heater B D term contribution
; to PID loop (signed integer)

STATUS:MODm/DINPUTS=bbbbbbb; LV(X)Bias and Heater(X): reports the status
; of DI01 to DI08 (each is 0=low or 1=high)

STATUS:MODm/MAG_Vn=f ; HS only: Floating point module *m* magnitude *n*
; voltage reading in V

STATUS:MODm/MAG_In=f ; HS only: Floating point module *m* magnitude *n*
; current reading in mA

STATUS:MODm/OFS_Vn=f ; HS only: Floating point module *m* offset *n*
; voltage reading in V



```
STATUS:MODm/OFS_In=f      ; HS only: Floating point module m offset n
                           ; current reading in mA
STATUS:MODm/DINPUTS=bbbb  ; HS and LVDS: reports the status of
                           ; DIO1 to DIO4 (each is 0=low or 1=high)
STATUS:MODm/VCPU_OUTREGn=d ; Modules with DIO: VCPU output register n
                           ; (unsigned 16-bit integer)
STATUS:END
```

7.1.3 SYSTEM

The output from the native SYSTEM command is written to the async message port in the following format, minus the comments in grey:

```
SYSTEM:BEGIN
SYSTEM:BACKPLANE_TYPE=n    ; n = 1 for an X4 backplane, n = 2 for X12
SYSTEM:BACKPLANE_REV=n    ; Backplane PCB revision, 0 = A, 1 = B...
SYSTEM:BACKPLANE_VERSION=n.n.n ; Backplane firmware, major.minor.build
SYSTEM:BACKPLANE_ID=x     ; 16 hexadecimal digit backplane unique ID
SYSTEM:MOD_PRESENT=x      ; Hexadecimal bit field: a 1 in the LSB
                           ; indicates a module is present in slot 1
SYSTEM:MODm_TYPE=n        ; Reports module type for slots 1...n.
                           ; 0: None
                           ; 1: Driver
                           ; 2: AD
                           ; 3: LVBias
                           ; 4: HVBias
                           ; 5: Heater
                           ; 7: HS
                           ; 8: HVXBias
                           ; 9: LVXBias
                           ; 10: LVDS
                           ; 11: HeaterX
                           ; 12: XVBias
                           ; 13: ADF
                           ; 14: ADX
                           ; 15: ADLN
                           ; 16+: Unknown
SYSTEM:MODm_REV=n         ; Module m PCB revision, 0 = A, 1 = B...
SYSTEM:MODm_VERSION=n.n.n ; Module m firmware, major.minor.build
SYSTEM:MODm_ID=x          ; 16 hexadecimal digit module m unique ID
SYSTEM:END
```

7.2 ARC (ASTROCAM)

The ARC detector controller firmware utilizes "3-letter" commands that can have up to four 24-bit arguments. The ARC-22 Timing Board will in most cases reply to these commands with an ASCII 'DON' (0x444F4E) on success or an ASCII 'ERR' (0x455454) on error, although some commands can produce other errors or return values. Table 3 shows return values which are mapped to their ASCII string counterparts so that when the camera server encounters one of these return values, the associated string will instead be returned; all other return values are returned by decimal (base 10) value.



Table 3. ASCII string return values for specified ARC return codes.

ARC value	return	string returned
0x00455252		ERR
0x00444F4E		DON
0x544F5554		TOUT
0x524F5554		ROUT
0x48455252		HERR
0x00535952		SYR
0x00525354		RST
0x00434E52		CNR

The 3-letter commands that are accepted by the controller are wholly dependent on the firmware that is loaded into the timing board and therefore any combination of 3-letter commands could exist; therefore this section can neither completely nor accurately describe all of the native 3-letter commands for your particular firmware. There are however, some commonly used, almost "standard" commands which are typically flashed into all ARC-22 timing board EEPROMs; these will be described here.

7.2.1 POF — Power Off

usage: POF

Turn off the biases and clocks.

7.2.2 PON — Power On

usage: PON

Turn on the biases and clocks.

7.2.3 RDM — Read Memory

usage: RDM <addr> <val>

Read from memory address <addr>. This is flashed into the on-board EEPROM so it is available even without uploading firmware.

7.2.4 SBN — Set Bias Number

usage: SBN <boardid> <dac#> <BOARD> <adu>

Sets the output DAC number <dac#> on board ID <boardid> to the voltage specified by <adu>.

The ID number of the board <boardid> is {0:15}.

<dac#> is the DAC number {0:7}.

<BOARD> is the string name of the board, "CLK" or "VID" and can be substituted with the hex value of those strings, 0x434C4B or 0x564944, respectively.

<adu> represents the voltage in A/D units, {0:4095} scaled to the max output voltage. E.g. for a 3.3V system, <adu> = 4095 x voltage / 3.3.



7.2.5 **SMX — Set Multiplexer**

usage: SMX <boardid> <mux1> <mux2>

The clock driver board has two multiplexed outputs. This specifies the MUX values to be output to the clock driver board. <boardid> is the board ID of the clock driver {0:15} and <mux1> and <mux2> are the clocks outputs {0:23}.

7.2.6 **TDL — Test Data Link**

usage: TDL <number>

Test fiber optic data link.

Writes a number to the timing board which returns the same number; used to test the operability of the fiber optic data link between the host and the timing board. This is flashed into the on-board EEPROM so it is available even without uploading firmware.

7.2.7 **WRM — Write Memory**

usage: WRM <addr> <val>

Write <val> to address <addr>. Note that the Y:memory starts at address 0x400000. This is flashed into the on-board EEPROM so it is available even without uploading firmware.

8. **FIRMWARE REQUIREMENTS**

The server software has certain expectations when interacting with the detector controller firmware, which are described here. The following is intended as reference for the firmware programmer.

8.1 **STA/ARCHON**

8.2 **ARC (ASTROCAM)**

8.2.1 **X:STATUS Register**

9. **APPENDIX A – EMULATOR**

9.1 **OVERVIEW**

A separate software program exists which allows emulating a detector controller. This allows testing and development of host software such as the COO Detector Controller Server, scripts or other client applications that may use the detector controller server, without the need of having a live detector controller.



The emulator is a stand-alone program apart from the detector controller server, although it was designed to work in conjunction with the detector controller server. It is essentially a "software-controller" -- a piece of software that behaves like a controller behaves. One would connect the emulator just as if connecting to an actual controller.

The emulator performs no logging (just as a detector controller does not) but it will write some messages to standard error, mostly error messages but a few progress messages are shown during potentially long operations such as exposure and readout delays.

There are no changes made to the detector controller server to use the emulator; the same software is used when connecting to the emulator as would be when connecting to a real detector controller. Only the configuration file (§3.2) needs to be changed, in order to instruct the server to connect to the emulated controller instead of the actual controller.

Currently, the emulator only emulates an Archon controller; ARC ("Leach") controller emulation has not yet been implemented.

9.2 COMPILATION

Although the emulator is a stand-alone program it is included with the detector controller server package, and shares some of the same code for convenience. The emulator is automatically built along with the detector controller server package when the "make" command is used (which defaults to "make all"). Alternatively, "make emulator" will compile only the emulator software. See also §3.6.2.

9.3 CONFIGURATION

The emulator uses the same configuration file used by the detector controller server. Two additional configuration keys are required, as shown in Table 4.

Table 4. Emulator configuration keywords

KEY	Description
EMULATOR_PORT	Port number for emulator
EMULATOR_SYSTEM	System file representing Archon modules (Archon-only)

For example,

```
EMULATOR_PORT=8001
```

```
EMULATOR_SYSTEM=/home/user/Software/sandbox/emulator.system
```

The EMULATOR_PORT is a port number that the emulator will listen to. The EMULATOR_SYSTEM file is described in the following section (§9.4).

Note that when running the emulator with the detector controller server, the ARCHON_IP keyword must point to the address of the machine which is running the emulator (or "localhost" if run locally) and the ARCHON_PORT keyword must be the same as the EMULATOR_PORT chosen above. This is because the detector controller server is now going to communicate with emulator instead of the Archon. For all intents and purposes, the emulator is an Archon.



9.4 ARCHON MODULE EMULATION

The Archon controller accepts a number of different types of modules which are addressed by their slot number. To properly emulate the hardware, the emulator must be configured with the specific hardware that is to be emulated. This is done with the file specified by the `EMULATOR_SYSTEM` keyword. Note that there are no requirements on the filename for this file but by convention it is suggested that ".system" be used. The .system file is an ASCII text file formatted as follows:

```
[SYSTEM]
BACKPLANE_REV=x
BACKPLANE_TYPE=x
BACKPLANE_VERSION=x.x.xxx
MODi_REV=x
MODi_TYPE=xx
MODi_VERSION=x.x.xxx
MODj_REV=x
MODj_TYPE=x
MODj_VERSION=x.x.xxxx
:      :      :
:      :      :
```

etc. where (*i, j, ...*) are the slot numbers, and *x* represents the respective revision, type and version of the specified modules.

This file can be generated manually, by hand; or, if a live Archon is available with the desired hardware configuration, the detector controller server command `systemfile` (§4.3.32) can generate the required file.

9.5 RUNNING

It is suggested that the emulator be run in the foreground of its own terminal window because it writes messages to standard error which may be useful to watch. Alternatively, it can be run in the background and standard error can be redirected as best suits the operator. If the program were compiled as,

```
[developer@localhost camera-interface]$ cd build
[developer@localhost build]$ rm -Rf *
[developer@localhost build]$ cmake ..
[developer@localhost build]$ make
```

then the emulator could then be run with:

```
[developer@localhost build]$ ../bin/emulator <file.cfg>
```

where <file.cfg> is the configuration file to be shared with the detector controller server.



9.6 EMULATION (OR, "WHAT DOES IT EMULATE?")

9.6.1 Archon

9.6.1.1 Command List

The following commands provide full, emulated responses:

SYSTEM
STATUS
TIMER
FRAME
FETCH
WCONFIG
RCONFIG
LOADPARAM
FASTLOADPARAM

The following commands are accepted and return immediately as though they were completed successfully:

FETCHLOG
LOCKn
CLEARCONFIG
APPLYALL
POWERON
POWEROFF
LOADTIMING
LOADPARAMS
PREPPARAM
FASTPREPPARAM
RESETTIMING
HOLDTIMING
RELEASETIMING
APPLYMOD
APPLYDIO
APPLYCDS
POLLOFF
POLLON

9.6.1.2 Timing

The requested delay (`exptime`, §4.3.13) is accurately emulated. The COO Detector Controller Server uses the Archon's own internal timer to time the exposure delays (`TIMER`, see Archon manual). Since the emulator emulates the `TIMER` command, it automatically can be used for timing exposure delays.



The emulator doesn't read, analyze, or in any way try to execute the waveforms, but it does try to mimic the frame time when filling its internal buffer with "pixels" just as the Archon would when reading a real device, in order to provide a reasonably accurate timed behavior. The frame-read time is already specified in the configuration file (`READOUT_TIME`) so the emulator uses this for timing. The emulator assumes that `READOUT_TIME` was specified as an upper limit for time-out purposes and produces lines at a rate commensurate with a frame time 90% of the specified `READOUT_TIME`, rounded *down* to the nearest 100 μ sec.

Since the reading of data is performed via TCP/IP (from the emulator to the host software) just as it would from the actual Archon, the timing here should be inherently accurate.

9.6.1.3 Image Data

An image of the dimensions specified by the ACF file is created and can be read by the `FETCH` command. Currently the data is "junk", but a future enhancement would be to fill this buffer with something meaningful.

9.6.1.4 Parameters and Configuration Memory

The emulator uses internal databases to store parameters and configuration keys in the same fashion that Archon does. This means that `RCONFIG` and `WCONFIG` access actual memory and, perhaps of more general interest, the detector controller server commands `getp` (§4.3.17) and `setp` (§4.3.31) also access actual memory. The user can therefore interact with their ACF file and parameters in the same fashion as is done on an actual Archon.

9.6.2 ARC

Not yet implemented.

10. APPENDIX B – OPEN ISSUES

Description	Affected Sections