P48 Mosaic Camera Interface Control Document and Data Pipeline

David Hale
Caltech Optical Observatories
M/S 105-24, 1200 E California Blvd., Pasadena, CA 91125
dhale@astro.caltech.edu

August 29, 2008

Abstract

This document defines the interface to the camera control computer and the output data path of the P48 Mosaic instrument for the Palomar Transient Factory (PTF) Project.

Contents

1 Introduction **4**
1.1 Scope of Document . . . 4

2 Components Description **5**
2.1 Detector Controllers . . . 5
2.2 Camera Host Computers . . . 5
2.2.1 Hardware . . . 5
2.2.2 Operating System . . . 5
2.2.3 Software . . . 6
2.2.4 Communications . . . 6
2.3 Shutter . . . 6
2.3.1 Hardware . . . 6
2.3.2 Interface . . . 6
2.3.3 Software . . . 6
2.4 Filter Changer . . . 6
2.4.1 Hardware . . . 6
2.4.2 Interface . . . 7
2.4.3 Software . . . 7

3 User Interface Application **7**
3.1 Overview . . . 7
3.2 CAMSERV . . . 8

4 CAMSERV Commands **8**
4.1 FITS Header Keywords . . . 8
4.1.1 Header Template File . . . 9
4.1.2 Keyword Values . . . 9
4.2 Image File Commands . . . 9
4.2.1 Imagefile Root Directory . . . 10
4.2.2 Imagefile Directory . . . 10
4.2.3 Imagefile Prefix . . . 10
4.2.4 Imagefile Basename . . . 10
4.2.5 Imagefile Suffix . . . 11
4.2.6 Imagefile Image Number . . . 11
4.2.7 Imagefile Rootname . . . 11
4.2.8 Write To Disk . . . 11
4.3 Exposure Commands . . . 12
4.3.1 Exposure Time . . . 12
4.3.2 Start an Exposure . . . 12
4.3.3 Stop an Exposure . . . 12
4.3.4 Exposure Progress . . . 12
4.4 Filter Changer Commands . . . 13
4.4.1 Move Filter . . . 13
4.4.2 Filter Position . . . 13
4.4.3 Initialize Filter Changer . . . 13
4.4.4 List Filter Names . . . 13
4.5 Vacuum Gauge Commands . . . 14
4.5.1 Vacuum Gauge Control . . . 14

4.5.2 Read Vacuum Gauge . . . 14

5 Configuration Files **14**
5.1 Filter Changer . . . 14
5.2 TCS Header Info . . . 15

6 FITS Header Template **15**
6.1 Default Header Template File . . . 15
6.2 Header Template Format . . . 15
6.2.1 `KEYNAME` . . . 16
6.2.2 `<expression>` . . . 16
6.2.3 `[comment]` . . . 16
6.2.4 `[timing modifier]` . . . 16
6.3 Header Template Example . . . 16
6.4 Special Keywords . . . 17
6.4.1 `FILEINFO` . . . 17
6.4.2 `UTSHUT` . . . 17
6.5 Internal Variables . . . 18

7 Local Data Pipeline **18**
7.1 Disk Organization . . . 18
7.2 Image File Location . . . 19
7.3 Disk Maintenance . . . 19

8 Host Computer Remote Access **19**
8.1 Host Names . . . 20
8.2 Ports . . . 20

A Example Client Interface **21**

List of Figures

1 P48 Mosaic Block Diagram . . . 4
2 Mosaic Disk Layout . . . 18

List of Tables

1 Introduction

1.1 Scope of Document

This camera interface control document (CICD) will describe the software interface layer to the P48 Mosaic camera, the interface by which the camera will receive commands and output pixel data. This document will be used by a programmer wishing to interface to the P48 Mosaic instrument. The local data pipeline will also be covered by this document.

The camera interface will be restricted to controlling the camera and in general will not provide for intercommunication with other subsystems that exist in the observatory (although some closely tied subsystem intercommunications will be included). Specifically, the camera interface will handle all communications between the user and the detector controllers, the shutter, the filter changer, the detector temperature controller, and the dewar vacuum gauge. See Figure 1.

The components within the scope of the camera interface will be briefly described in Section 2. The user interface application will be described in Section 3, and the actual commands will be described in Section 4.

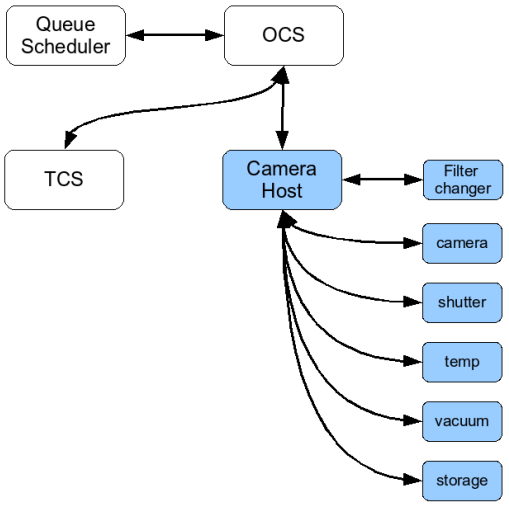

Figure 1: P48 Mosaic Block Diagram. The camera host computer interface (the scope of this document) will be limited to those items shaded in blue.

2 Components Description

This section will describe briefly the various components that make up the instrument.

2.1 Detector Controllers

The P48 Mosaic array of twelve CCDs will be divided into two banks of six CCDs, with each bank handled separately by its own detector controller; controller 0 will be denoted as the "master" and controller 1 will be denoted as the "slave" controller. The controllers are Generation 3 Leach (SDSU) devices and each will be comprised of three 2-channel video boards, three clock boards, and a timing board. A separate computer will be required to communicate with each of the two controllers. Communication between a controller and its respective host computer will be conducted via a fiber optic serial link between the timing board and a corresponding PCI card in the host computer. The controllers will share a common clock so that the exposures and readouts of the two controllers will be synchronized with each other.

2.2 Camera Host Computers

Each controller will require a separate host computer. The two host computers will be hardware-identical except that the "master" host will have an additional 1TB hard disk.

2.2.1 Hardware

The camera host computers will be based on a Supermicro X7DAE+ motherboard with dual quad-core Xeon 2.5GHz CPUs. They will contain 2GB RAM, a 160GB disk for the operating system, software and user accounts, and a 1TB disk for data storage[1]. The computer will be enclosed in a Supermicro 883T-550 3U rack mountable enclosure.

A Generation 3 Leach controller PCI card will be required to communicate with the detector controller. This will be a 32-bit/33MHz PCI card and it will be inserted into the single 32-bit/33MHz PCI slot available on the X7DAE+ motherboard.

2.2.2 Operating System

Each camera host computer will run the 32 bit version of the Linux operating system distributed as Fedora Core 8 with kernel version 2.6.24.5-85.fc8; other kernels are untested and are not expected to be compatible with the Leach PCI driver. Updates to the operating system will not be supported.

The Leach PCI card device driver supplied by Astronomical Research Cameras (ARC) is called astropci2.0 but ARC has supplied different versions of astropci, each with the same 2.0 version number. The revision number .1 has been added to the PCI driver in use for P48 Mosaic by D.Hale/CIT; hence the PCI driver will be known locally as astropci2.0.1.

[1] The "master" host computer will have an additional 1TB disk

2.2.3 Software

Each computer will run its own software to communicate with its connected controller. The PCI driver and the application software required to run the instrument (described in Section 3) will be started automatically when the computer is turned on; user input will not be required to launch the necessary software. A VNC server (display `12`) will be started automatically to allow access for monitoring the camera server operations. Remote access will be described in Section 8.

2.2.4 Communications

The host names of the computers used for communicating with the master and slave controllers will be known locally as `p48m` and `p48s`, respectively. The user will be required to communicate with `p48m` only.

The Linux TCP/IP Sockets API will be utilized for all communication with `p48m`. Section 8.2 will describe the TCP ports available. The commands to be sent over these ports will be described in Section 3.2.

2.3 Shutter

2.3.1 Hardware

The shutter is a commercial unit which will be supplied by Sci-in Tech.

2.3.2 Interface

The Shutter Interface box will accept RS-232 communications and a TTL signal but the RS-232 communications will not be utilized; only the TTL signal input will be connected. The TTL input (BNC port labeled "OPEN" on the Shutter Interface box) is an active LOW input, meaning a TTL low signal here will open the shutter. This will be driven from J8 on the master controller power board which will be brought to the outside panel of the master controller on a BNC output connector. This BNC output will be labeled "SHUTTER".

2.3.3 Software

The shutter will be controlled by the DSP code running in controller 0 (the "master"). When the controller receives an `expose` command, the DSP code running in the master controller will operate the shutter with a period that corresponds to the exposure time.

2.4 Filter Changer

2.4.1 Hardware

The filter changer assembly is a mechanism which will provide for motorized selection of one of two filters; one filter will always be in the field. Absolute filter positioning will be registered by running the filter frame against a mechanical hard stop. Once the motor runs the filter frame against this stop, the motor carriage will be carried in the opposite direction, tensioning a spring

and actuating a micro-switch. Actuation of this switch will trigger the motor controller firmware to decelerate the motor to a stop.

There will be no safety limit switches; mechanism safety will be accomplished through a "watchdog" type of time-out algorithm running in the hardware controller.

The filter changer will not use encoders; instead, the controller will report a count of the number of steps moved by the micro-stepping motor. Operational performance will be monitored by comparing the number of steps moved against the number of steps required for a correct move. Mechanical problems will be identified by this metric. The user will be responsible for monitoring this metric.

2.4.2 Interface

The filter motor controller will have an RS422 serial port. This serial port will be connected to a device server so that communication with the controller will be made via TCP sockets.

2.4.3 Software

The software will know that there are two filters and it will know these filters as position `1` and `2`. The user will specify by position number (`1` or `2`) the desired filter to be in the field. A configuration file will allow the user to assign filter names and IDs to these position numbers for the purpose of identifying the proper filter in the FITS header. The user will be responsible for maintaining the filter list configuration file (see Section 5.1).

When power is applied to the filter controller it will immediately begin moving the mechanism to locate the positions of the two filters.

When given a `move` command, if the filter position limit switch is not located within 16 seconds, then the controller will take this as a fault condition and will disable the motor. (Nominal filter positioning time will be 15 seconds.) The user will be required to re-initialize the filter before another `move` command will be accepted.

3 User Interface Application

This section describes the actual application software with which the user interfaces to the camera system (identified by those items shaded in blue in Figure 1).

3.1 Overview

The software which runs on each controller host computer will be based on panVIEW, the Pixel Acquisition Node (PAN) version of ArcVIEW. PanVIEW is a command server which is modeled after a Unix kernel; the kernel, or in this case the PAN command server, doesn't know about hardware but only how to communicate with various drivers (or modules). Then, for every required task (e.g. acquisition, FITS writing, etc.) there will be an independent module which knows the specific details of the hardware with which it must communicate. The real functionality comes from the installed modules, while the role of the PAN server will be to parse and pass commands to the modules and return their responses. PanVIEW and its modules are written primarily in LabVIEW which handles most of the command parsing and communications, and

uses C functions for the low-level hardware interfacing and any real-time processing (such as image unscrambling and FITS file writing).

Each computer will run its own instance of panVIEW. The application name for each instance of panVIEW will be `_p48m` and `_p48s` for the master and slave host computers, respectively.

In order to provide a single interface to the user and present the two controllers as though they were one camera, a specialized server (CAMSERV) will run on the master host computer. All instrument I/O will be handled by this single server interface.

3.2 CAMSERV

Because the two controllers that make up the camera are each handled by a separate computer, there will be two instances of panVIEW (one running on each computer). Each instance of panVIEW will not know about the other, nor how many there might be; the job of a panVIEW server will be to communicate with its local controller.

CAMSERV (for CAMera SERVer) will be an application to control multiple panVIEW applications and present them as a unified instrument. Much like panVIEW, CAMSERV will not have any intelligence about the commands it sends or receives, but will know (through a configuration file) how many devices (such as PANs) it needs to talk to and what their addresses are; its job will be to accept and parse commands from the user, send them to the appropriate device, receive the response from the device and report status. CAMSERV will be responsible for merging the FITS files produced by each controller into one mosaic image.

4 CAMSERV Commands

This section describes the commands recognized by CAMSERV. The commands will be shown here in `typewriter font`. Parameters displayed in `[ square brackets ]` will be optional; parameters displayed in `<angle brackets>` will be required; parameters separated by a `|` (pipe symbol) indicate OR. String literals will be displayed *in italics*. The basic structure of a command will be:

```
<device> [all|<app>] [_BLOCK_] <command> [<arguments>]
```

Here, `<device>` is the particular device to receive commands, such as the PAN device, filter device, etc.; `<app>` is the optional application name (or "`all`" – if `<app>` is omitted then it defaults to `all`); `<command>` and `<arguments>` are the command and optional arguments to be sent to `<device>`. `_BLOCK_` is an optional keyword, explained in Section 8.2.

4.1 FITS Header Keywords

These commands will manipulate the header template file and should be used with caution. The header template file will be described in Section 6.

4.1.1 Header Template File

device: `pan`
command: `fits get|set hdrfile <filename>`
response: *filename (i.e. the name of the current header file, if* `get`*)*
`DONE` *(if* `set` *okay)*
`WARNING: File does not exists. Set anyway` *(if file does not exist)*

This command will be used to specify a different header template file. Only the filename will be changed; the full path to the file will be fixed as `/home/arcview/fpas/<app>/config/` where `<app>` refers to the application name, `_p48m` or `_p48s` for the master or slave controllers, respectively.

4.1.2 Keyword Values

device: `pan`
command: `fits keyword [args...]`
args: `set|add <name> [type] [value] [//comment]` *(adds* `<name>` *to the template file)*
`get <name>|all` *(return the entry for keyword* `<name>`*, or all entries)*
`delete <name>` *(deletes the entry for keyword* `<name>` *from the template file)*
response: `DONE`

where for `set`, `get`, and `delete` commands, `<name>` will be the name of the keyword; for the `set` command, `[type]` will be the type, which will default to STRING if not specified, `[value]` will be the value of the keyword, and `[//comment]` will be the comment and will be preceded by `//`; for the `get` command, if `all` is specified then the complete list of keywords will be returned, each separated by `\r\n`.

This command will modify the template file.

Example:
`pan fits keyword set MYKEY FLOAT 10.1 // this is my keyword`

will give on the file image headers: *MYKEY = 10.1 / this is my keyword.*

4.2 Image File Commands

This section will use the following terminology; an image file will be composed as:

`<froot><dir><prefix><basename><sufix><number>.fits`

Changing `<froot>` and `<prefix>` will not be permitted; their fixed values will be: `froot=/data_m/`, `/data_s/` and `prefix=m_`, `s_` for the master, slave controllers, respectively.

For example, if the user specifies,

`dir`	images
`basename`	PTF200802043010_1_o_

```
suffix      _NONE_
number      22
```

then the image from the master controller will be saved as:
`/data_m/images/m_PTF200802043010_1_o_0022.fits`,
the image from the slave controller will be saved as:
`/data_s/images/s_PTF200802043010_1_o_0022.fits`,
and the final, merged 12 detector mosaic image will be saved as:
`/data/PTF200802043010_1_o_0022.fits`.

Note that the use of `<dir>` will not be supported in the final image file path. See Section 7 for more information on the local data pipeline. The `.fits` file extension will automatically be appended by the software.

4.2.1 Imagefile Root Directory

device: `pan`
command: `get image.froot`
response: *name*

where *name* will be pre-defined as `/data_m/` and `/data_s/` on the master and slave computers, respectively. This will be a read-only variable.

4.2.2 Imagefile Directory

device: `pan`
command: `get|set image.dir <name>`
response: *name (if* `get`*)*
`DONE` *(if* `set` *okay)*
`ERROR DHE.vi  directory does not exists err -2` (if given a path that doesn't exist)

where *name* will be the name of a subdirectory below `image.froot`. If `_NONE_` is specified for `<name>` then no subdirectory will be used.

4.2.3 Imagefile Prefix

device: `pan`
command: `get image.prefix`
response: *name*

where *name* will be pre-defined as `m_` and `s_` on the master and slave computers, respectively. This will be pre-pended to the image file basename.

4.2.4 Imagefile Basename

device: `pan`
command: `get|set image.basename <name>`

response: *name (if* `get`*)*
`DONE` *(if* `set` *okay)*

where *name* will be the base name of the image file.

4.2.5 Imagefile Suffix

device: `pan`
command: `get|set image.suffix <name>`
response: *name (if* `get`*)*
`DONE` *(if* `set` *okay)*

where *name* will be the suffix to append after the image file basename. If `_NONE_` is specified for `<name>` then no suffix will be appended.

4.2.6 Imagefile Image Number

device: `pan`
command: `get|set image.number <number>`
response: *number (if* `get`*)*
`DONE` *(if* `set` *okay)*

where *number* will be the next image number to append after the image file suffix (or basename if suffix not used). The image number will automatically increment after each exposure if `write_to_disk` is set to `yes` (see command 4.2.8).

4.2.7 Imagefile Rootname

device: `pan`
command: `get|set image.rootname <name>`
response: *name (if* `get`*)*
`DONE` *(if* `set` *okay)*
`ERROR DHE.vi  directory does not exists err -2` (if given a path that doesn't exist)

where the *name* returned will be the combination of `<froot><dir><prefix><basename><suffix>`.
If `set`, then the software will take the name after the last `/` as the `<basename>` and will set it.

4.2.8 Write To Disk

device: `pan`
command: `get|set write_to_disk yes|no`
response: `yes` or `no` *(if* `get`*)*
`DONE` *(if* `set` *okay)*

This command will be used to specify whether an image read by the `expose` command will be written to disk. The default is `yes`.

4.3 Exposure Commands

4.3.1 Exposure Time

device: `pan`
command: `get|set exptime <time> [ms]`
response: *time* `[ms]` *(if* `get`*)*
`DONE` *(if* `set` *okay)*

where `<time>` is a number of msec. The unit `[ms]` may be specified or omitted; if omitted, the default unit is msec. The unit `[ms]` will be returned with the `get` command.

4.3.2 Start an Exposure

device: `pan`
command: `expose`
response: `OK` *(if started OK)*
`ERROR DHE.vi ERROR ERROR There is not enought disk space for image, or there is no write permission err -28` *(if an error)*

Note that the shutter will be operated automatically.

4.3.3 Stop an Exposure

device: `pan`
command: `abort`
response: `DONE`

This command will stop an exposure in progress; it will not stop a readout.

4.3.4 Exposure Progress

device: `pan`
command: `get progress`
response: *all of the following:*
`read =` *n (where n is percentage of readout progress, from 0-100)*
`write = 0` (*not used*)
`exposure =` *n (where n is the number of msec of exposure time completed)*
`imagename = <prefix><basename><sufix><number>.fits` *(see Section 4.2)*
`imagepath = <froot><dir>` *(see Section 4.2)*
`imagenumber =` *n (where n is current image sequence number)*
`state =idle | exposing | reading` *(one of the preceeding)*
`imstatus =` (*not used*)
`imnumber= <number>` *(imagefile image number; see Section 4.2)*
`nimages=` *n (where n is the number of images to take in a sequence)*

This command will return each of the various parameters of the exposure progress, as indicated above.

4.4 Filter Changer Commands

4.4.1 Move Filter

device: `filter`
command: `move 1|2`
response: `DONE` *n (if OK)*
`ERROR: FLT_API.vi ERROR Filter Not Found` *(if error)*

where *n* will be the number of motor steps required to move to the current position.

4.4.2 Filter Position

device: `filter`
command: `get position`
response: `FILTER` *pos*`:` *id* `(`*name*`)`

where *pos* will be the filter position number (`1|2`), *id* will be the filter ID, and *name* will be the filter name. For example, `FILTER 2: 12 (Haoff)`

4.4.3 Initialize Filter Changer

device: `filter`
command: `init`
response: `CAL` *n*

where *n* will be the number of motor steps between the two filters. This command will be used to initialize the location of the two filters. It will be called once automatically when power is applied or when panVIEW is started.

4.4.4 List Filter Names

device: `filter`
command: `list`
response: `FILTER:`
name1 `1 0 0`
name2 `2 0 0`

where *name1* and *name2* will be the names of the two filters as defined in the filter configuration file (see Section 5.1). The first number following the name will correspond to the filter position number (`1|2`). The second and third numbers (`0 0`) are reserved values and will not be utilized. The response will come on three lines as indicated above.

4.5 Vacuum Gauge Commands

4.5.1 Vacuum Gauge Control

device: `vac`
command: `power on|off`
response: `DONE`

This command will turn the vacuum gauge on or off. Turning it on will not guarantee that the gauge warm-up period has been satisfied. The user will be required to allow the required warm-up period before reading a valid value.

4.5.2 Read Vacuum Gauge

device: `vac`
command: `get pressure`
response: *n*

This command will return the pressure *n* read by the vacuum gauge.

5 Configuration Files

This section describes the configuration files which are required to be modified by the user.

5.1 Filter Changer

The filter placed in the field will be located by a position number, `1` or `2`. These position numbers will be matched to the NAMEs and IDs of the installed filters by means of a configuration file. The user will be required to modify this configuration file in order to correctly indicate the filters installed in the filter changer. The information in the configuration file will be that used in the FITS headers.

The filter configuration file will be located on the master host computer, `p48m` under:

`/home/arcview/fpas/_p48m/config/FLT_filters.list`

The format of the filter configuration file will be:

`[FILTER]`
id1`="`*name1* `1 0 0"`
id2`="`*name2* `2 0 0"`

where *id1* and *id2* will refer to the FILTER ID installed in positions 1 and 2, and *name1* and *name2* will refer to the FILTER NAME installed in positions 1 and 2, respectively.

5.2 TCS Header Info

The user will be required to supply all TCS-based information for the FITS file headers. All TCS information for the FITS headers will be written into a special header template file called `TCS_INFO.tpl`, located on the master host computer `p48m`. The full path to this file will be:

`/home/arcview/fpas/_p48m/config/TCS_INFO.tpl`

This file will follow the formatting rules described in Section 6.2. The master header template file, `P48m_HDR.tpl` will contain the special keyname `FILEINFO` which points to the `TCS_INFO.tpl` file. See Section 6 for a complete discussion on header template file usage, and Section 6.4.1 for a discussion of the special keyname `FILEINFO`.

6 FITS Header Template

The FITS headers generated for each image will be described in a header template file. The header template file will specify the order, names, comments and either an actual value or where to get the values for the specified keywords. The template file will be read and at the moment of generating the image (after an `expose`), it will add the specified keywords to the header, taking the current values of the variables from the specified source.

6.1 Default Header Template File

The default header template files will be:

`/home/arcview/fpas/_p48m/config/P48m_HDR.tpl`
and
`/home/arcview/fpas/_p48s/config/P48s_HDR.tpl`

for the master and slave host computers, respectively. The files will not be identical; the master (`P48m_HDR.tpl`) will have an additional `FILEINFO` entry for reading the TCS info (see Sections 5.2 and 6.4.1).

6.2 Header Template Format

The header template file will be an ASCII-text file, with one line per entry. Each entry will correspond to a new keyword and will take the form of:

`KEYNAME  ='<expresion>' /[comment] [timing_modifier]`

These parameters will be described in the following sub-sections:

6.2.1 KEYNAME

This parameter will be the name of the keyword.

6.2.2 <expression>

This parameter will correspond to the source of the value, or the value itself, and will accept the following values:

`[]`*datatype*`[] <value>`
: This will take `<value>` and add it as an actual value according to the *datatype* specified. If *datatype* is not specified then it will default to STRING. Allowed *datatype*s are: U32 (ULONG), U16 (UINT), U8 (BYTE), I32 (LONG), I16 (INT), I8 (SHORT), FLOAT, DOUBLE, STR (STRING), 2DARR (two-dimensional arrays). Here, U=unsigned, I=signed, FLOAT is 32 bits, DOUBLE is 64 bits. *datatype* can have parenthesis surrounding it as in C; for example, `(FLOAT) 2.5`

`dbs [name]`
: This will get the value from the internal panVIEW database. If `[name]` is specified then the value will come from the database variable `[name]`. If `[name]` is not specified then the database variable will be the same as the `KEYNAME` specified. The keyword *type* will be that of the variable type which is stored in the internal database. See also Section 6.5 for a list of special internal variables which will be allowed in place of `[name]`.

`file <name>`
: When used in conjunction with a special keyword (`FILEINFO`), this will read the specified file "`<name>`" and append it to the headers. See Section 6.4 for a discussion of proper usage of the `FILEINFO` keyword.

6.2.3 [comment]

This parameter will be an optional comment which will appear with the corresponding keyword.

6.2.4 [timing modifier]

By default, header data will be written after an exposure. This parameter will allow the user to override that default and write the keyword with this modifier to the header prior to the data being written. Allowed values for the timing modifier are:

`__BEFORE__`
: This will write the keyword before writing the data.

`__AFTER__`
: This will write the keyword after writing the data.

6.3 Header Template Example

This is an example of a header template file:

```
OBJECT  ='dbs title       ' /object title gets the value of the internal database
                                          variable "title" and place the value there
SOFTVER ='dbs app_ver     ' /software version
HDR_REV ='dbs             ' /header revision gets the value of database variable "hdr_rev"
OBSERVER='dbs             ' /observer
UTSHUT  ='database        ' /UT shutter open this will be the time where the shutter opens
EXPTIME ='dbs             ' /exposure time (secs)
CCDTEMP ='(FLOAT) 112.5   ' /celcious degrees will write 112.5 as float
ANYKEY  ='any comment     ' /if  no datatype  specified, it will be added as string
FILEINFO='file <expresion>' /appended file header see Section 6.4.1
```

6.4 Special Keywords

The following special keywords will be allowed in place of `KEYNAME` in the header template file:

6.4.1 `FILEINFO`

Use of this keyword will specify a file (or files) with extra information which will be added to the headers. Usage:

`FILEINFO ='file <name>'  /[comment]`

The value of `<name>` will correspond to either a specific file or to multiple files by using wildcards. The specified file will follow the same formatting syntax as described for the header template file (Section 6.2). Use of `FILEINFO` will be required for including TCS info.

Example:

```
FILEINFO ='file /home/arcview/fpas/_p48m/TCS_INFO.tpl'
```

6.4.2 `UTSHUT`

This keyword will supply the shutter-open time. Usage:

`UTSHUT   ='database   '  /shutter time, value must be database`

This keyword must be set equal to `database`. The value returned will be the moment at which the controller started the exposure. This time will be different from that returned by the TCS due to communication delays. The format of the value returned here will be:

`YYYY-mm-ddTHH:MM:SS.msmsms`

where:

`YYYY`	four digit year
`mm`	two digit month 01-12
`dd`	two digit day 01-31
`T`	standard date-time separator
`HH`	two digit hour 00-23
`MM`	two digit minute 00-59
`SS`	two digit second 00-59
`msmsms`	three digit millisecond 000-999

6.5 Internal Variables

These are some useful internal variables which can be used in place of `<expression>` in Section 6.2.

name	type	comment
`title`	`string`	image title
`observer`	`string`	observer(s) name(s)
`aexptime`	`float`	actual exposure time of the image
`exptime`	`float`	requested exposure time
`detreadtime`	`float`	detector total readout time
`mext`	`bool`	multiple or single extensions (flat) fits file
`hdr_rev`	`string`	GFITS module revision
`comment`	`string`	comment
`UTSHUT`	`database`	time when integration started (shutter opened)

7 Local Data Pipeline

7.1 Disk Organization

The master and slave camera control computers will each have a 1TB disk mounted under `/data_m` and `/data_s`, respectively. The slave disk will be exported via NFS and cross-mounted on the master computer. The master computer will have an additional 1TB data storage disk mounted under `/data`. See Figure 2.

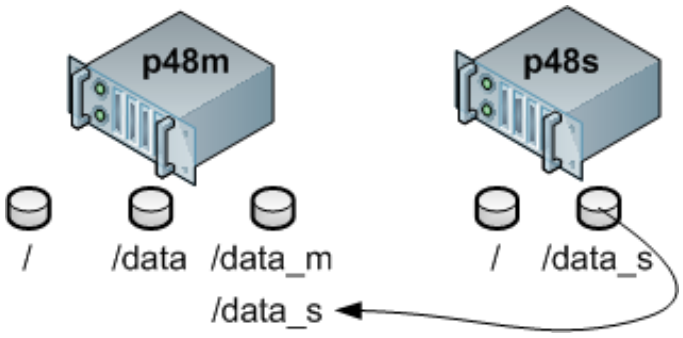

Figure 2: Camera Host Computer Disk Layout. The master host computer will have three disks and the slave host computer will have two disks, as indicated above. The disk containing images from the "slave" controller 1 will be exported via NFS and cross-mounted on p48m. The operating system, software, and user accounts will be on the / (root) disk.

7.2 Image File Location

1×6 mosaic images are stored on the local disk of their respective controller computer with a prefix added in front of the specified name. The prefix is `m_` or `s_` for the master or slave controller, respectively. For example, if the filename given to the camera server is:

```
PTF200802043010_1_o_0022
```

then the master controller it will save the top 6 detectors on its local disk in the file:

```
/data_m/m_PTF200802043010_1_o_0022.fits
```

and the slave controller will save the bottom 6 detectors on its local disk in the file:

```
/data_s/s_PTF200802043010_1_o_0022.fits.
```

The final, merged image file is one multi-extension file which combines the images from the two controllers. The merging will be performed automatically by CAMSERV as soon as it detects that the separate FITS files were created. The final 2×6 mosaic multi-extension FITS file will therefore be named:

```
/data/PTF200802043010_1_o_0022.fits
```

where the `/data` disk is mounted on the master controller computer.

7.3 Disk Maintenance

The camera control interface will not do anything to maintain adequate disk space. The user will be responsible for removing image files as needed, in order to provide space for new images.

8 Host Computer Remote Access

This section will detail the remote access with the host computers. The user will be required to access only the master host computer, which will have the configuration files (Section 5) to be edited by the user.

The CAMSERV application will run on the master host computer. Access to the CAMSERV application will be made via TCP/IP sockets, whose ports will be described in Section 8.2.

General access to the master host computer will be allowed via SSH.

A VNC server will run on the master host, utilizing display `12` (upon boot-up) as the display where both the panVIEW and CAMSERV applications will be executed (these applications will be executed automatically upon boot-up).

The disks (cf. Section 7.1) will be exported to allow cross-mounting over NFS, so that the user will be able to update configuration files (Section 5) directly. The user will supply the IP address(es) of any computers which will need to mounts these disks.

8.1 Host Names

The host names for the master and slave control computers will be `p48m` and `p48s`, respectively. The domains and IP addresses are TBD upon delivery to Palomar.

8.2 Ports

The CAMSERV application will listen for TCP connections on two ports on the `p48m` host, as follows:

port	purpose
`2157`	non-blocking
`2158`	single connection, blocking

Commands sent to the single connection blocking port will block subsequent connections until command completion, at which time the response is returned and the connection is closed. This port is useful for queuing commands which must be executed sequentially.

Commands sent to the non-blocking port will be sent immediately to the panVIEW server and return an immediate response. In some cases the response will only be an acknowledgement that the command was received (for example, sending an `expose` command on the non-blocking port will return an immediate `OK`, but the exposure and readout of course will take many seconds to complete. However, this will allow the user to continue to send commands to the server when appropriate. For example, a `filter move n` command could be sent during readout, which would otherwise be blocked on the single-connection blocking port. In this example, the user would be required to check the progress of the exposure (see Section 4.3.4) to ensure that the `state` was `idle` or `reading`. The non-blocking port will allow multiple simultaneous connections.

A command sent to the non-blocking port can be forced to be blocking on a per-command basis by using the `_BLOCK` keyword. This keyword is case-sensitive. For example, if the command:

```
filter _BLOCK_ move 1
```

is sent to the non-blocking port, then it will block the non-blocking port from receiving additional commands until the `filter move` command returns. The `_BLOCK_` keyword will take effect only for the command with which it is issued, and it will not prevent additional connections to the non-blocking port.

A Example Client Interface

This appendix describes an example client interface for communicating with the camera host server. The following example is written in C and should be platform-independent.

```
/*--------------------------------------------------------------------------
| client.c
|--------------------------------------------------------------------------
| This is a simple TCP client which writes a single command to a server,
| reads back any response, and prints it to the screen. No error checking.
|
| by David Hale, CIT
| 2008 Aug 22
|
| compile with:  gcc -ansi -O2 -o client client.c
| (some systems may need also -lnsl and/or -lsocket)
|--------------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>         /* system i/o calls */
#include <strings.h>        /* for bzero */
#include <arpa/inet.h>      /* for htons, inet_addr, other networking svcs */
#include <netinet/in.h>     /* for sockaddr_in struct, other internet defs */
#include <sys/types.h>      /* basic system data types */
#include <sys/socket.h>     /* basic socket defs */
#include <netdb.h>          /* for gethostbyname */

#define BUFSIZE 4096

int main(int argc, char *argv[])
  {
  char               *hostname, reply[BUFSIZE], command[BUFSIZE];
  unsigned short      port;
  unsigned long       inetaddr;
  int                 sockfd;
  struct sockaddr_in  servaddr;
  struct hostent     *hp;

  if (argc < 4)
    {
    printf("\nusage: %s <ipaddress> <port> <command> \n\n", argv[0]);
    exit(-1);
    }
  else                        /* get host:port and command from command line */
    {
    port     = atoi(argv[2]);
    hostname = argv[1];
    sprintf(command,"%s\r\n", argv[3]);   /* add carriage return and newline */
    }
```

```
/* initialize socket address structure and construct internet address */

bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(port);

if ( (inetaddr = inet_addr(hostname)) != -1 )
  memcpy((char *)&servaddr.sin_addr, (char *)&inetaddr, sizeof(inetaddr));
else
  {
  if ( (hp = gethostbyname(hostname)) != NULL)
    memcpy((char *)&servaddr.sin_addr, hp->h_addr, hp->h_length);
  else
    {
    fprintf(stderr, "error constructing server address");
    return(-1);
    }
  }

/* create the socket, connect to server,  write command, read reply */

sockfd = socket(AF_INET, SOCK_STREAM, 0);

connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr));

write(sockfd, command, strlen(command));

read(sockfd, reply, (size_t)BUFSIZE);

printf(reply);

return(0);
}
```