PanView configuration

CTIO 60 inches CHIRON

CHI60S-6.2

La Serena, August 2011

Contents

Introduction

The following document is a reference to the CTIO 60 inches CHIRON panview configuration. It provides a way of understanding the configuration files for panview when handling the echelle detector and controller. **This files should NOT be changed by the regular observer**, but just by the administrator. It is dangerous to change these files unless the person who does knows exactly what he is doing.

Panview is a specific application of a Pixel Acquisition Node (PAN), which has the responsibility of acquiring the pixel data from the detector. Of course "acquiring the pixels" involves being able to handle the detector and controller, send and receive commands from the detector hardware, etc. It may also be in charge of the temperature control of the CCD, etc. ***Figure 1.1*** shows a general diagram of the software, showing were panview fits in the application (connected to PANDEV device)

Panview has been designed so it can handle different detector controllers in a transparent way -basically by keeping the same front end for the command and responses.

In this particular application, panview is in charge of:

a) talking to the detector controller (orange monsoon):
- Command/response handling
- Pixel data acquisition
- header gathering

b) Fits image generation:
- headers and pixels from the controllers
- Receives the header information from the ExposureMeter application and write them down into the headers

Panview is the only one that has the information required to talk to the controller hardware (drivers, libraries, etc)..

Panview has tcp/ip servers to connect to the outside world. In this application is is connected to the PAN device (PANDEV) on the main application, and also to the ExposureMeter application on expmeter60 computer

In the current document we will not describe panview's internals at all, but just the configuration files that are required to handle the chiron detector and controllers.

In chapter 1 we will review the location of the configuration and log files and the main configuration files Since panview is a totally stand-alone application it can start without the main application, and can acquire complete images from the chiron camera. This can be very useful for debugging and engineering purposes (in fact, in the laboratory and characterization work panview is the only one used). In chapter 2 we will review some useful scripts for starting up panview alone -including an engineering GUI-.

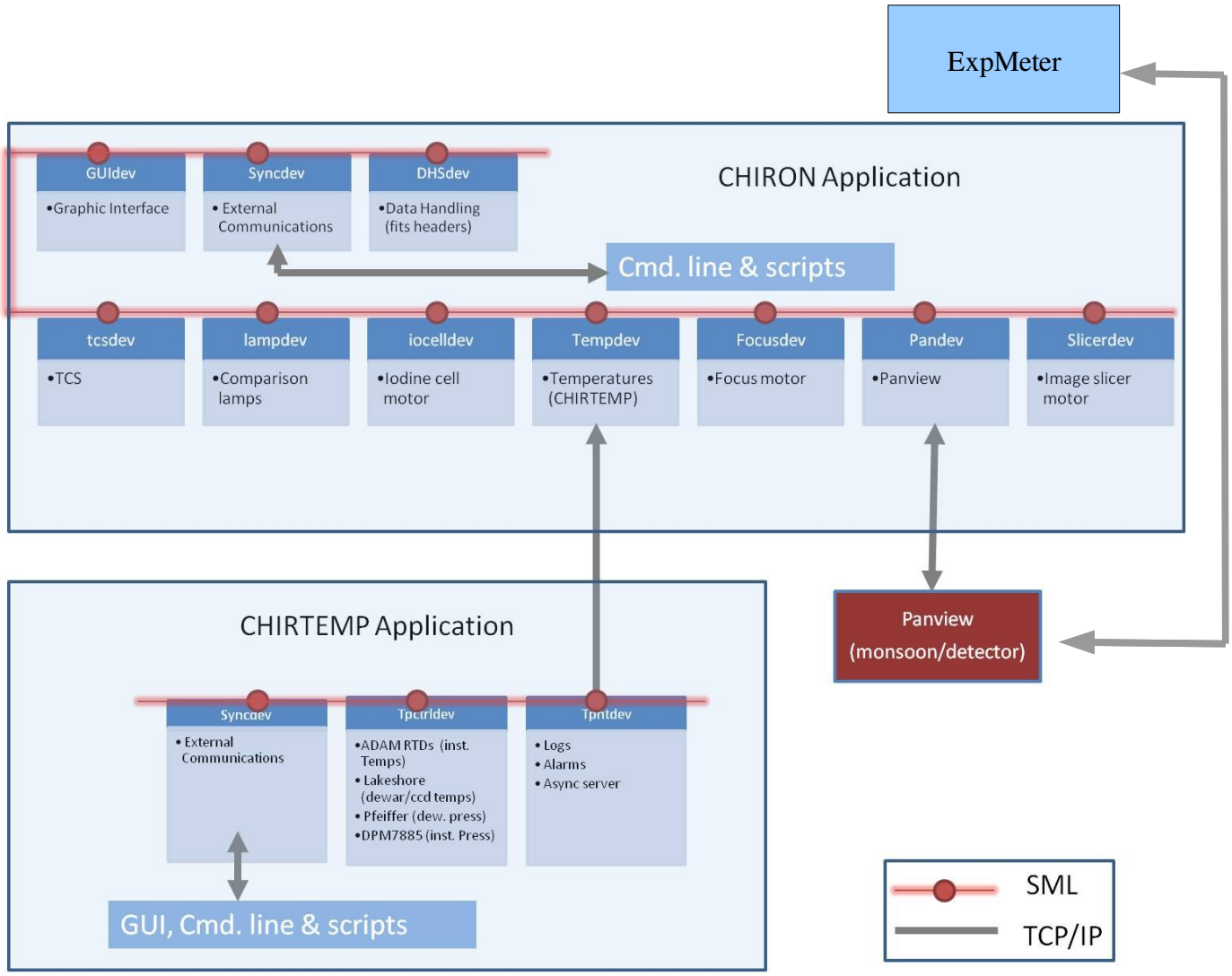

Figure 1.1: panview in the application general diagram

Chapter 1: Panview configuration files

1.1 Locations

Panview configuration files are in **PAN_ROOT/fpas/<fpa_name>/config**

For the current application, **PAN_ROOT** is **/home/observer/panview**, and **fpa_name** is **_chiron**. By convention, all panview fpas (Focal Plane Arrays) start with "_" . So, in the current application
/home/observer/panview/fpas/_chiron/config
are located the most important configuration files. However, there are also some important files located in a "common" area ("common" for all panview fpas, if more than one is handled in the same machine. In this case "_chiron" is the only fpa on the machine). The files we will review here, in the common area, are located in
/home/observer/panview/fpas/common/DET_DATABASE

1.2 _chiron configuration files

Inside _chiron/config are all the internal panview configuration files. From this, the more useful for the user (engineer or who maintains; **the actual observer should not edit these files**). Note that we will indicate only the keywords that could be changed; the remaining should not be touched, except by the experts.

1.2.1 Controller specific files

Inside the configuration directory there is a subdirectory called **DETECTOR**. This is the place where all the controller-specific files are located

The monsoon controller requires three files to run property. The files extensions are:
.csv; this file has all the hardware information and its associated names (boards locations and register naming and characterization, biases name association with location and gain, clocks, etc)
.ucd: this is the sequencer assembler binary, to be uploaded to the master control board at boot time.
.mod: this file is a "macro" that has all the desired "initial" values to the registers (called "attributes"). This file should be called after boot.

These three files will be used/called during the monsoon startup process, that we will describe next.

When the monsoon-module in panview starts, it looks first a file called dhe.conf:

```
[Device]
devno=0                                  //PCI channel number to use to download
type=orange                              // type or monsoon hardware
link=auto                                //type of link -PCI- in use. See notes below
[FPA]
name=CHIRON                        //name of fpa to run. See notes below
modifiers="prescans 50 overscan 50"      //defines initial prescans and overscan
readmode=quad                      //defines initial read mode (amplifiers to be used
[Config]
hdwfile=./chiron_Config.csv              //defines the name of the .csv file to use
transfile=./command_translations.cfg     //translations files. See notes below
libdir=_MODPATH_/MNSN/private/c/lib       //path to driver API libraries
slmon=/opt//sl240/sl240/bin/sl_mon       //path to driver reset program
[Misc]
Commands=dhe_init.mc                     //initialization macro. See notes below
shutdownmacro=chiron_Powerdown.mod       //called when a "shutdown" command is received.
bpp=16                                   //defines bits per pixel in the output images
```

Notes:

- Monsoon controller can be handled using different links (communication channels). This means, using different communications protocol. This defines the PCI card installed, and all its associated drivers/APIs. The key "link" defines what this application is using. The options for the orange monsoon are: "slink", "systran", "slink_d", or "auto". When "auto" it autodetects what PCI card is installed, and based on this choses what to use.
- The name of the fpa to use is passed to panview's internal geometry module, which will use the name as an index to a file that will lead to the whole description of the focal planes. These are the files we will review in the next point (DET_DATABASE files)
- "command translations" is a file that make simple translations between attributes (that must be defined in the .csv file in use) and standard panview commands (basically, to set the exposure time, set this attribute, etc)
- The initialization macro is a macro that is called right after the initialization process has been finished (a macro is, basically, a set of commands)

Once this file has been read we know what the fpa looks like -geometrically speaking- , what is the csv file to use, and what is the initialization macro to call. The system then initializes -open and reset the driver, set the size on the controller, etc-, and finally calls the initialization macro. This macro looks like

```
# setup dacs and register values
```

macro .chiron_FourAmpSetup_500Kpix.mod //here it is called the .mod file
macro dhe_read_quad //this macro sets what amplifiers will be used. See the macro below
//sets the "attribute" to enable the clocks
#set it to single extensions
set image.extensions no //set to sinfle -flat- fits files (no extensions)
#be sure shutter will open automatically
set obs.type object //sets initial image type
#so we know when we start, and we override a possible witting to cols/rows on the mod file
set binning 1 1 //sets initial binning

The ***dhe_read_quad.mc*** macro sets what amplifiers will be read. This macro looks like:

#sets the system to use lower both amplifiers
#load specific sequencer code
macro loadasm chiron_sequencer_500KPix_Binning.ucd // loads the assembler file
 # set arcview read mode
set readmode quad
#set Redirect to channels **//the rest is controller-specific settings**
Redirect[0]=1
Redirect[1]=3
Redirect[2]=7
Redirect[3]=5
 # set xfercount to 4 pixels per transfer
XfrCount=4

1.2.1 Fits header template

Panview generates an image with all the fits headers related to the camera and geometry. The DHS at the application level (DHSDEV) takes the fits image and adds the keywords related to other mechanisms (focus, slicer, temperatures,TCS information, etc).

Panview generates the fits headers based on an image template. The template for chiron is called CHIRON_HDR.tpl and is in the same config directory. It looks like:

OBJECT ='dbs title ' /Name of object observed

OBSERVER='dbs observer ' /observer

PROPID ='dbs propid ' /Proposal ID

OBSID ='dbs expID ' /exposure ID

IMAGETYP='dbs obstype ' /Type of picture (object, dark, comp, etc)

CCDSUM ='dbs ccdsum ' /On chip summation (X, Y)

UTSHUT ='queue ' /UT of shutter open

DATE ='dbs FITSDATE ' /Date FITS file was generated

NAMPSYX ='dbs nampsyx ' /Num amps in y and x (eg. '2 2=quad')

AMPLIST ='dbs amplist ' /Readout order in y,x

GEOMETRY='queue ' /template on GEOM_INFO.tpl

REXPTIME='dbs exptime ' /requested exposure time

EXPTIME ='dbs aexptime ' /Exposure time in secs

DARKTIME='dbs darktime ' /dark current time in secs

DHEINFO ='dbs 2DARR DHEINFO ' /template on DHE_INFO.tpl

PANID ='dbs nodeID ' /PAN identification

COMMENT ='dbs FITSCOMMENT comment ' /comment

"**dbs**" means to get the current value from the internal panview database. The main template calls other templates or internal "tables" (indicated as 2DARR). For example, DHEINFO is a 2D internal table that holds values from the Detector Head Electronics (monsoon), and its template is in the same directory, and called DHE_INFO.tpl

The fits server in panview looks at this template everytime a new fits image will be created, so it is possible to change the headers on the fly modifying this template between images, or -recommended- using on-the-fly commands for modifying the template. For information on these commands see CHI60S-3.X on scripting (look for the panview fits commands).

The ExposureMeter application send commands to panview -through a tcp/ip connection- to add/update the header template information, so to include the exposure meter calculation values. This commands are of the type

f*its add EMTIMOPN dbs em_timopn // shutter open time from exposure meter __IFNDEF__ 0000-00-00T00:00:00.000 __AFTER__*

sent at the very beginning, meant to add the new key specified (*EMTIMOPN)* to take the value from the internal database variable indicated *(em_timopn)*, the value specified if the variable is not found/defined (*0000-00-00T00:00:00.000),*and take the value after the readout completes. Then a command like this

dbs set queue em_timopn 2009-08-21T11:27:944.121 STR

every time an image is taken, to update the value of the custom variable "*em_timopn"*

1.3 DETECTOR_DATABASE files

This files are in fpas/common/DET_DATABASE. In general here is the definition of the focal planes, including detectors, position, read modes, etc. Specific documentation on these files can be found on panview's documentation (GEOM module) so we will not go into deep explanations here. We will just mention the files which are relevant for the _echelle focal plane.

1.3.1 FPA

This directory has the main fpa file, called ***CHIRON.fpa.*** We won't go into the description of this file in detail, but the most important entry here is the one that specifies the detector in use:

[DET_1]
ID="E2V-4k_3"
POSITION=(1,1)

This indicates that this FPA is using detector E2V-4k_3 on position (1,1) (respect to the lower-left coordinate of the FPA). IN this case, this is the only detector, so it must go into position (1,1).

1.3.2 DETECTORS

This directory stores all the defined detectors. Note that a "detector" here indicates an specific, physical detector device, with a specific serial number -not a generic type-. According to the FPA file entry, the file to look for it ***E2V_4k_3.det***. Inside this file there will be information specific to this detector:

Type="E2V-4k"
SerialNumber=3
Grade=Science
DetID=E2V-4k
Gain11=3.2
...

The engineer can add any relevant information here. In this case we have specified gains for both amplifiers, read noises, etc. All or any of this information is available for being added into the image headers. The "Type" entry is very important, because it says what "kind" of detector it is, meaning what is the generic detector type.
In this file there can also be specified what readmodes are available to this specific detector.

[READMODES]

quad="quad_chiron.rdm"

This tells that for this detector this readmode is available.

1.3.3 TYPE

This directory has definitions of generic detector types. The specific detector will inherit all the characteristics of its "type". Based on the file Site424_6.det, the file to look here is ***Site424.typ.*** If any key is repeated, then the .det file will override the .typ file

[INFO]
Name="E2V-4k"
Manufacturer="E2V"
Format="4096x4112"
xamps=2
yamps=2
Read_Noise="< 3 e-"

This here should be all "generic" detector type information, as given by the manufacturer. One important entry here -that cannot be missing- is **Format,** because it tells the size of the data area of the detector, in cols X rows. "Readmodes" are usually also specified here, as the standard readmodes for this detector. However, if any readmode is specified in the .det file (as in our case), this generic readmodes are not considered (as in our case, the actual detector has the lower amplifiers unusable, so any readmode that involve those amplifiers is not available)

1.3.4 READMODES

This directory has the definitions of the readmodes. Each readmode has a unique file called <readmode>.rdm. In our case, we should look for the file called quad_chiron.rdm as stated in the E2V-4k_3.det file,

[A_1] **//amplifier 1**
coords=(1,1) **//coords in detector, referred to lower-left corner. See notes below**
format=(cols/2Xrows/2) **amplifier size. "cols" and "rows" will be replaced by the detector format**
type=LL **//amplifier type: Upper Left**
rotated=0 **//not rotated (0 degrees)**
fliped=none **//not flipped**
ampid=11 **//amplifier ID**

[A_2]
coords=(cols/2+1,1)
format=(cols/2Xrows/2)
type=LR
rotated=0
fliped=none
ampid=12

[A_3]
coords=(cols/2+1,rows/2+1)
format=(cols/2Xrows/2)
type=UR
rotated=0
fliped=none
ampid=22

[A_4]
coords=(1,rows/2+1)
format=(cols/2Xrows/2)
type=UL
rotated=0
fliped=none
ampid=21

Notes:

- The amplifier "type" indicates in which direction it is readout. Available types are: Lower Left (LL), Lower Right (LR), Upper Left (UL) and Upper Right (UR). ***Figure 1.2*** shows the reading directions for UR and UL. The long line shows the parallel clock and the short one with the arrow, the serial direction. Note that a UR is a flipped UL, etc.
- The "coords" of the amplifier indicates not where the amplifier is located inside its area, but where the amplifier data area starts, referred to lower-left corner

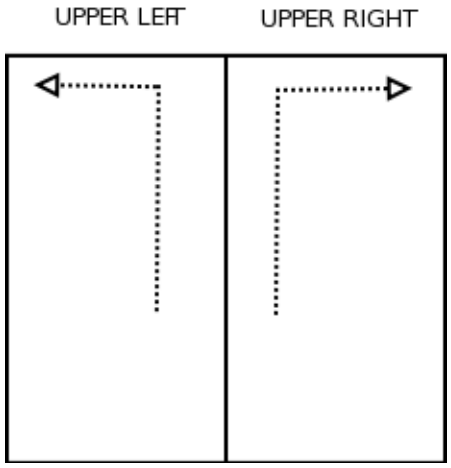

Figure 1.2: Upper Left (UL) and Upper Right (UR) amplifiers

Chapter 2: binaries, scripts and macros

Here we will review briefly the generic scripts and binaries that panview is using in this application. Some scripts are generic to panview, and some were built specifically for this application. The macros described (see ***2.3***) are macros specifically created for this application.

2.1 panview startup/shutdown scripts

Panview scripts are located in PAN_ROOT/panview/bin. In this application, ***/home/observer/panview/bin***

Inside this directory, the most important ones are:

start_panapp:

Starts panview. Its usage is : "start_panapp <fpa_name>". This causes panview to start using the specified fpa configuration files. For this application, panview is started as **start_panapp _chiron**

shutdown_panapp:

Shuts down panview. Its usage is : "shutdown_panapp <fpa_name>". This causes panview to shutdown For this application, panview is shutdown as **shutdown_panapp _chiron**

start_pangui

This scripts starts an engineering GUI for panview. Its usage is "start_pangui <fpa_name>". This starts an engineering gui connected to the specified panview. This gui can be run in parallel to any other panview connection (panview accepts multiple clients). To start the echelle panview engineering gui: **start_pangui _chiron.**

2.2 panview auto-generated script

Every time panview starts (every time "start_panapp <fpaname>") it generates a script that is able to talk to the started application. This script is called "pan<fpaname>". For this application, then, the auto-generated script is called **pan_chiron.** It also creates a soft link called "pan", so the user can simply use the word "**pan**" to talk to panview. This also allows to create scripts that talk directly to panview.
Note that this panview-scripts will bypass the main application. It is possible now to send any panview command to panview through any terminal. For example

> pan expose // this is the same as "pan_chiron expose"

would cause panview to take an image.
For panview commands, please refer to the panview documentation. For a basic sample of panview commands, also see ***CHI60S-3.X*** on scripting

2.3 Macros

A macro is file that has a list of commands separated by newline (basically, one command per line). Panview can read this file and execute all the commands in them, returning a single response as if it were a single command. This allows to build more specialized actions and execute them in an easier way. Using macros in combination with scripts can be very useful. The macro's can take arguments also. The macro interpreter can also execute a system script, as is actually being used in this application.

For this application there are some specific macros and scripts available. These macros can be executes by command line (and scripting) and are also available to the main GUI, usually through drop-down menus. The macros are located in the default macro's directory defined in the main panview config file (PAN_ROOT/fpas/_chiron/config/PAN.cfg). In the current application this directory is PAN_ROOT/fpas/_chiron/config/DETECTOR.

The extension of a macro should be ".mc". If no extension, it assumes some system script. To "manually" execute any of this macros/scripts, requires a simple command like

"pan appmacro <name>". (example: "pan appmacro set_binning 2 2")

2.3.1 speed

To change the readout speed there are two csh scripts

speed_fast/ speed_slow: changes the readout speed to fast or slow mode. It sets the pixel time, changes the gain (e-/adu), and moves the offset of the electronic to match the output. It also sets the gain, read noise and speed mode variables in the internal database -for header purposes-. Note that the GUI reads in this directory for the macros named "speed_XXX.",and presents XXX available in the speed drop-down menu, so any new speed could be made available in the GUI by simply creating a new speed macro.

2.3.2 binning

To change the binning factor there is one csh script that takes two arguments

set_binning: sets the binning factor in the electronics's firmware, and also in panview itself. This macro takes two arguments: <xbin> <ybin>, for example: set_binning 2 2

The macro is indeed a csh script (if panview recognizes that the "macro" is really a script, it executes it as

a system call), and it was generated as such because based on the binning factor the geometry headers should be adjusted to reflect the real binned data, so based on the binning factor requested the geometry keywords are adjusted. The adjustments are done in an empiric way, because it is not always clear if a border pixel should be used or not in advance. This script, as well as the next one (set_roi) invokes another csh script to make the actual header adjustment based on speed, binning and roi. This script is called “adjust_headers” and it is located on $HOME/bin (it must be on the path so the scripts can find it)

2.3.3 Region Of Interest (ROI)

This macro is really a csh script, because it needs to perform more complex actions. It executes the csh script “set_roi” that takes as arguments <xstart> <ystart> <xlen> ylen>. The scripts sets the ROI parameter in the controller (through panview engineering commands) and in panview itself.

2.4 Controller Driver

Panview handles the specific controller using specific drivers/libraries that will depend, on the lowest level, on the controller hardware itself. In the case of the monsoon controller, it uses two types of communication protocols, which means, from the hosts' point of view, different PCI cards, and different drivers and libraries.

2.4.1 Systran

The systran interface is standard orange monsoon communication mechanism. In order to work, there must be an specific systran driver loaded into the kernel. In this case, the driver is located at

/opt/sl240/driver.

To load the driver, there is a script called “load_systran” in /opt/sl240 or in /opt directly. This scripts loads the driver into the kernel. The script should be called at machine's boot time (either by being called from the file /etc/rc.local, or through the standard linux init.d mechanism).

The driver name is ***fxsl.*** To verify that the driver has been property loaded, there are two things to check:

a) To see if the driver is loaded into the kernel:

> /sbin/lsmod | egrep fxs s

> fxsl_module 25072 0

b) To see if the proper entries in the devices linux structure has been created:

> ls -l /dev/fxsla*

should return a list of entries. There should appear at least:

> crw-rw-rw- 1 root observer 247, 0 Dec 22 02:27 /dev/fxsla0

2.4.2 Slink

The slink interface is the protocol adopted by DECam (FermiLabs) so it is also available to any monsoon controller through panview. This can be used as an alternative communication protocol to systran. Note that the link type (systran or slink) is 100% transparent to all the software (if the panview's fpa config file says “link=auto” in the main dhe configuration file).

If slink is in use, the driver should be at

/opt/slink/

and the script ***load_filar*** or ***load_fdrivers*** to load the driver should be inside that directory or in /opt directly. This script should be called at boot machine's boot time from /etc/rc.local or through the standard init.d mechanism.

The driver name is ***filar.*** To verify that the driver has been property loaded, there are two things to check:
a) To see if the driver is loaded into the kernel:

> /sbin/lsmod | egrep filar

> filar   104024 1

b) To see if the proper entries in the devices linux structure has been created:

> ls -l /dev/filar*

should return something like:

> crw-rw-rw- 1 root system 247, 0 Dec 8 10:04 /dev/filar

References

- panview documentation

Glossary

FPA:

Focal Plane Array: The arrangements of detectors on the camera. It includes the geometric and the electronic information of those detectors. The simplest case of an FPA is a single-detector FPA (as the echelle fpa). More complex cases are mosaics of tents (or even hundreds) of detectors

PAN:

Pixel Acquisition Node. This is a software “node” that handles pixels from some specific FPA. The responsibility of the PAN finishes usually when the data has been transmitted or written to disk as a fits image with both pixels and all the detector (or camera) relevant information. Simple FPAs usually have also a single PAN (as the echelle case), but mosaics usually can have several PANs (example: Dark Energy Camera (DECAM) has 6 PANs, The 48" Palomar mosaic has 2 PANs.

Panview is a particular implementation of the generic concept of PAN