

CFHT Data Acquisition



2003 September 19th

This document is available on the Web at: http://software.cfht.hawaii.edu/cfht_data_acquisition/

Abstract

This documents a general plan for data acquisition at the Canada-France-Hawaii Telescope (CFHT). It attempts to cover most relevant software development practices and computing hardware directly involved in the process of controlling a science instrument, reading pixels from a detector (infrared or visible), packaging the pixels as data files, and visualizing the image. The target goal is to accommodate wide field imaging devices like MegaCam and WIRCAM, and provide a general framework for future instruments at CFHT. As of 2003, most of the important infrastructure for status, logging, and command processing is now complete.

Our current computing architecture and goals for the future are the result of careful consideration to available hardware solutions, compatible operating systems and software, and a path of transition from our existing infrastructure (which we came upon by the same methods in the past.) The evaluation, planning, and implementation of these goals are iterative processes, so the things presented here may change as technology improves. In summary, this document presents a general philosophy, the current status, and our best prediction for the near future.

While other software systems at CFHT are at least partly based on the architecture set forth here, there are many which are not described in this document. They include the Telescope Control System, the Elixir project which provides sophisticated data analysis, the Queue Scheduling project which plays the role of an efficient observer during survey mode and Queue Scheduled programs, and the data archive pipeline which packages data for distribution and transfer to other facilities. The information here is limited to what is most relevant to instrument designers and acquisition interface designers. The *users* of these systems will want to read the observing manuals instead. This document is for developers.

Contents

1	Computing Network Diagram	3
2	Computing Network in Detail	4
2.1	Telescope	4
2.1.1	Engineering Interface	4
2.1.2	Remote Auxiliary Control	4
2.1.3	Detector Controller	5
2.2	The 4th Floor	5
2.2.1	Detector Hosts	5
2.2.2	Session Host	5
2.2.3	Display Host	6
2.3	Waimea Headquarters	7
2.3.1	Remote Display Host	7
3	Hardware Platforms	7
4	Operating Systems	8
5	Acquisition Software Philosophy	8
5.1	Goals	9
5.2	Philosophy	9
5.2.1	Astronomy requirements	9
5.2.2	Flexible implementation	10
5.2.3	Low down-time	10
5.2.4	Long term maintainability	10
5.3	Policy	10
6	Session Tools	11
6.1	Window Session Concepts	11
6.1.1	Multi-Head	11
6.1.2	Single Logical Screen	12
6.1.3	Xinerama	12
6.1.4	Screens	13
6.1.5	Xinerama Screens	13
6.1.6	Virtual DeskTop and The Pager	13
6.1.7	Window Placement	14
6.2	X-Server	14
6.2.1	Drivers	14
6.2.2	8-bit PseudoColor	14
6.2.3	24-bit TrueColor	14
6.2.4	24-bit DirectColor	15
6.2.5	Overlays	15
6.3	Window Manager	15
6.3.1	MWM	15
6.3.2	FVWM 1.24r	16
6.3.3	FVWM 2.4	16
7	Infrastructure	17
7.1	Status	17
7.2	Feedback and Logging	17
7.2.1	Message Types	18
7.2.2	Additional Message Types for Agents	19

7.2.3	Additional Message Types for debugging	19
7.3	Director	20
8	User Interface	20
8.1	Command Line Interfaces	20
8.2	Graphical Interfaces	20
8.2.1	Image Display	20
8.2.2	Form-based Interfaces	20
9	Programming Languages	20
9.1	Bourne Shell	20
9.2	C	21
9.3	The Rest	21
10	Development Tools	21
10.1	GNU make and Make.Common	21
10.1.1	Standard Targets Provided by Make.Common	22
10.1.2	Ways to turn on other options from the command line	23
10.1.3	Make Variables	23
10.1.4	Variables Useful in the Makefile	26
10.1.5	Variables That Depend on Local Installation	26
10.2	C Compiler	27
10.3	Symbolic Debugger	27
10.4	Verification Tools	27
10.5	Documentation	27
10.5.1	Web Manuals	27
10.5.2	Printable Manuals	27
10.5.3	Word Processors	27
10.5.4	XFig, L ^A T _E X, L ^A T _E X2HTML, ps2pdf	28
11	Software Organization, Versions, and Conventions	30
11.1	Location of Files	30
11.2	Makefile conventions	31
11.3	Index files	32
11.4	Changes files	33
11.5	RCS Check-in and Check-out	34
11.5.1	Initial Check-in	34
11.5.2	Help, the file is now <i>gone</i> !	35
11.5.3	Check-out	35
11.5.4	Check-in	36
11.5.5	Rlog and RCS keywords	37
11.5.6	RCS and Emacs	38
11.5.7	Summary of Check-in/Check-out	39
11.6	Releases	39
11.7	Archival Back-ups	39
11.7.1	Summary of Archiving/File maintenance commands	41
12	The Observing “Sessions”	42
12.1	History	42
12.2	Template Session	42
12.3	Components of a NEO session	42
12.3.1	Home directory and login (NIS)	43
12.3.2	Obsolete account files	43
12.3.3	Dot Files	44

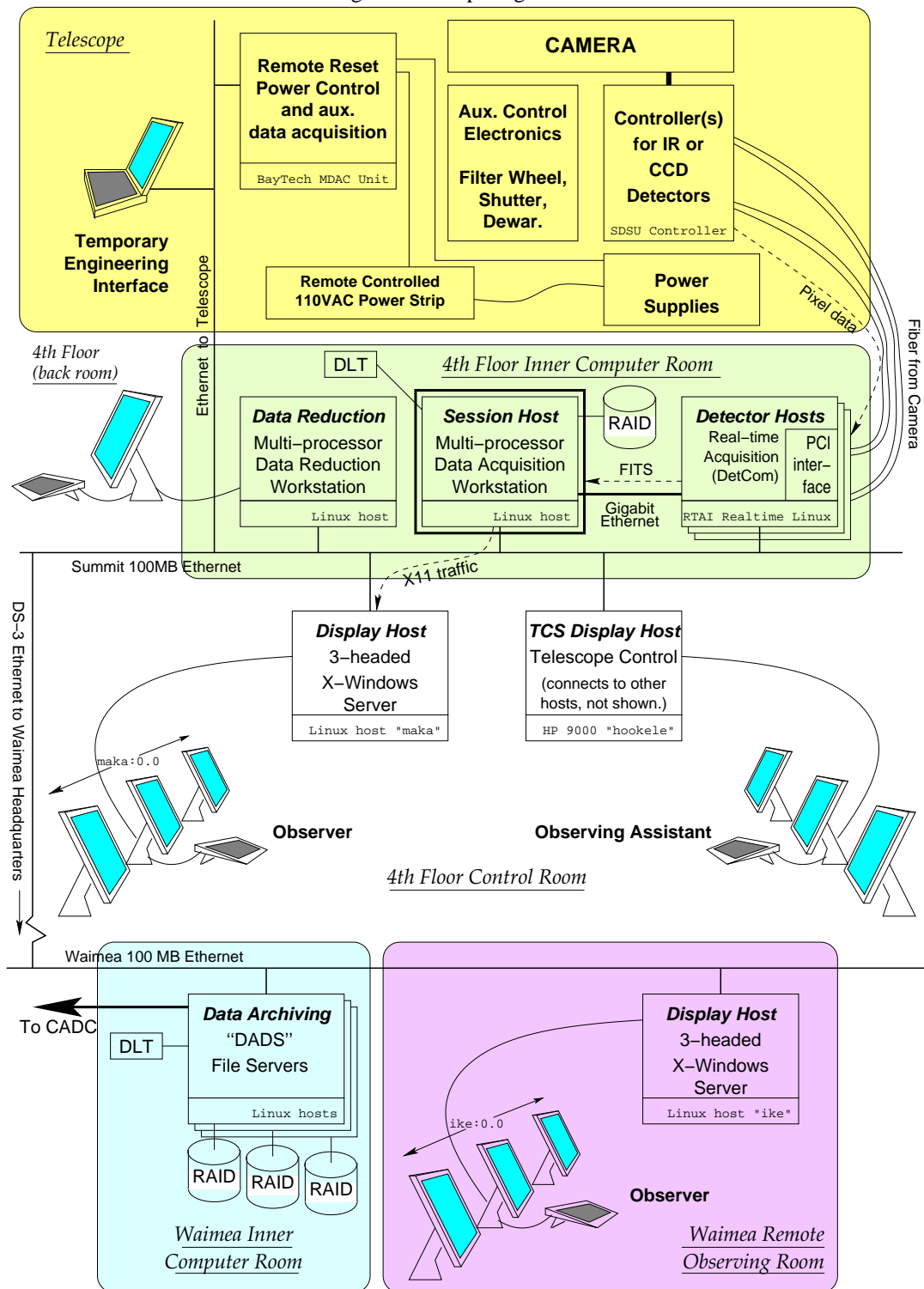
12.3.4 GUI Form(s)	50
12.4 Misc	50
13 Conclusion	51
14 Reference	51
14.1 Environment	51
14.2 File Locations	51
14.3 Manuals	51

List of Figures

1 Computing Network	3
2 Computing Network: Telescope	4
3 Computing Network: 4th Floor	6
4 Computing Network: Waimea	7
5 Window Session: Multi-Head	11
6 Window Session: Single Logical Screen	12
7 Window Session: Xinerama	13
8 Sources of Logging Messages	17

1 Computing Network Diagram

Figure 1: Computing Network



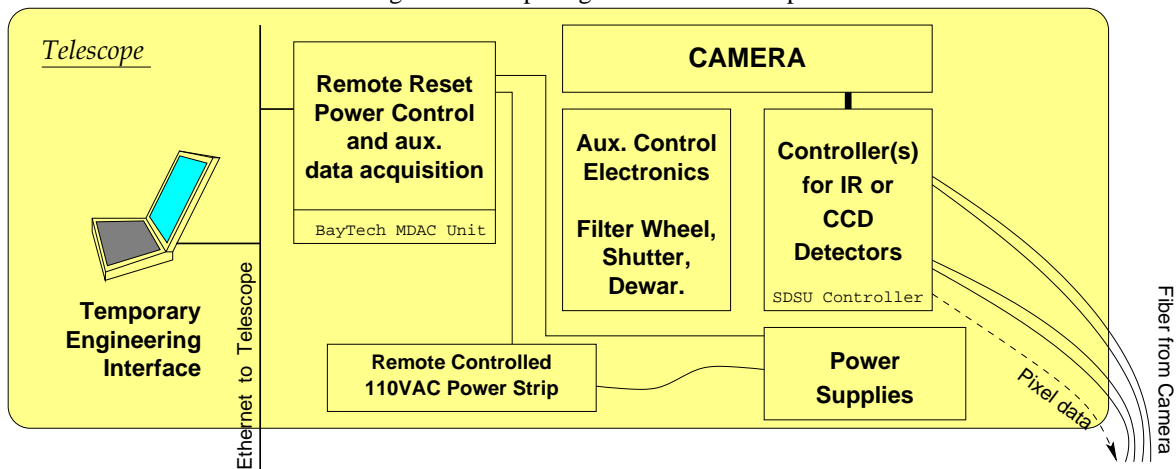
2 Computing Network in Detail

There are three main locations where computing devices are installed. All are interconnected by various speeds of Ethernet at the moment.

2.1 Telescope

For this discussion, anything installed on the telescope itself, whether at Prime, Cassegrain, or even Coude is all considered “Telescope.” [Though this might be a good place for some discussion of differing connection facilities at each?] Only hardware which is required to be near the instrument is installed here. This includes hardware to control power supplies and monitor auxiliary electronics, and the detector controller for the camera itself.

Figure 2: Computing Network: Telescope



2.1.1 Engineering Interface

Locations on the telescope, such as the prime focus, are wired with Ethernet. It is possible to connect a portable computer to have limited access to the instrument. Small “network appliances” (I-Openers) are also currently installed at locations on the telescope to transmit images from sky-monitoring camera cameras.

2.1.2 Remote Auxiliary Control

Ethernet is also used to connect remote diagnostic units like the BayTech “MDAC” which provides a route to Serial connections for other hardware and control of power sources. Units such as this can also be used to acquire basic data (voltages, temperatures, etc.) at points near the instrument. RS232 and other serial connections are commonly used by devices on the telescope, but it is possible to tunnel them through the Ethernet, or through fiber in the case of MegaCam’s controllers. This simplifies error correction and virtually eliminates distance limitations.

2.1.3 Detector Controller

MegaCam uses a detector controller designed at the C.E.A. in France by Jean de Kat, which is based on Analog Devices' "sharc" DSP. This new controller handles the high readout speeds of MegaCam. The guide CCDs for MegaCam use the San Diego State University ("ARC" generation II) controller.

Currently, all our other controllers are also the SDSU II type, by Bob Leach. These consist of a set of boards providing the analog and the digital functions, and are based around the Motorola 56000 Digital Signal Processor (DSP).

A final controller for WIRCAM has not yet been determined, but an SDSU generation III controller is currently being used for characterization and learning about the new Rockwell detectors/multiplexers in the lab in Waimea.

All controllers currently use fiber optic links to send pixel data down to the 4th floor computer room.

2.2 The 4th Floor

Below the telescope, on the 4th floor of the building, is a climate-controlled computer room. It houses two important computing hosts for data acquisition: the **Detector Host** and the **Session Host**.¹ A third host, the **Display Host** is located outside the computer room and close to the screens which it drives, for logistical reasons.

2.2.1 Detector Hosts

The fiber carrying the pixel data from the instrument connects (via a nearby patch-panel) to an interface on a Detector Host's PCI bus. (This is true for both SDSU and MegaCam controllers.) The main function of this host is to reliably read out the detector and provide the data in **FITS format** to the Session Host. Redundant hosts exist mainly for backup, though it is conceivable to use multiple detector hosts at once.

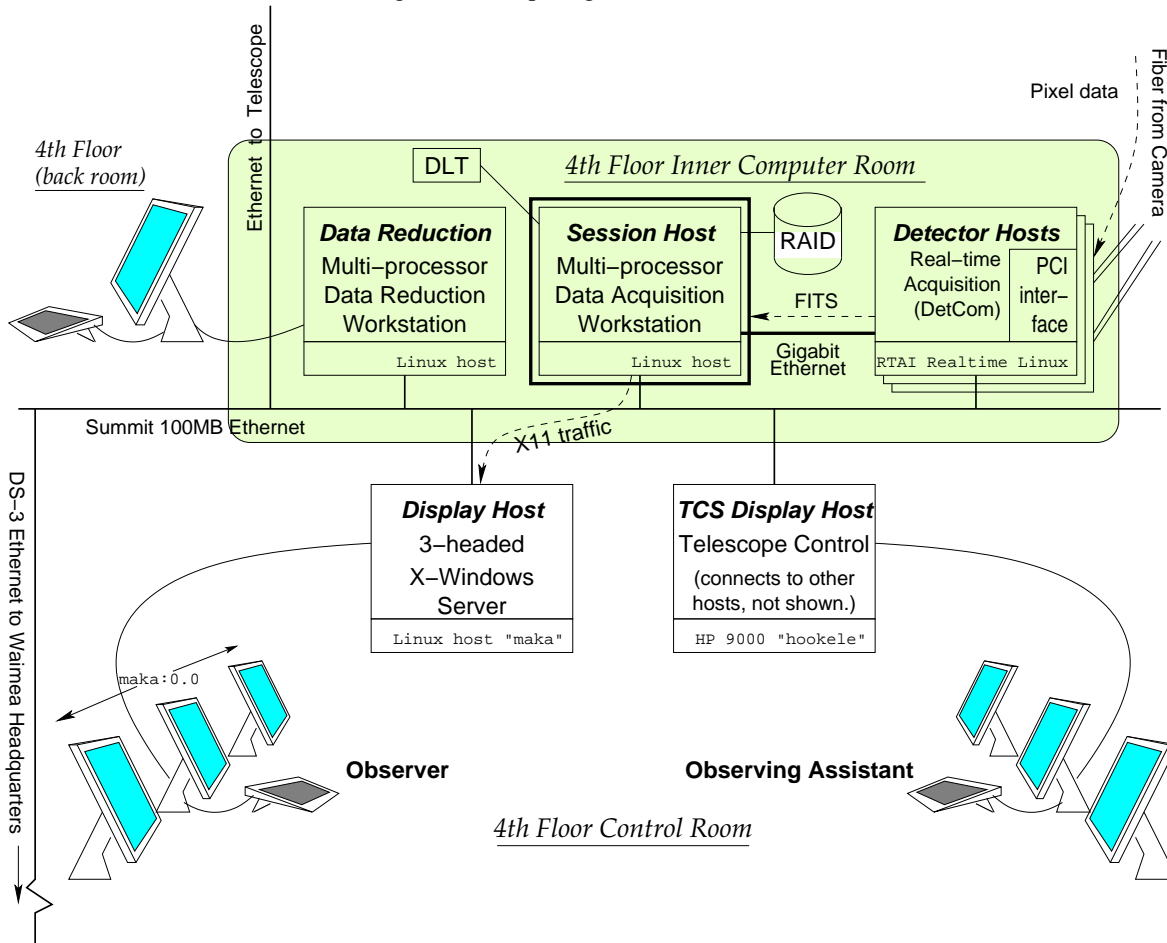
For infrared acquisition, the Detector Host may perform more complex tasks, such as co-adding frames and performing calculations for on-the-ramp integration. (It is preferable to us to do these tasks on a Unix host than on a DSP.) Each host will be configured with sufficient memory and local disk space for these tasks. If possible, the local disk space should be sufficient to also take over the functions of the Session Host (see below) for a full night in an emergency. Thus, if everything save what we have described so far were to fail, it is still possible to do a lot with just the detector controller and the detector host. In a lab setting, this is often the normal mode of operation. It also means our engineering interfaces are not radically different from our observing interfaces. [NOTE: This mode is not practical for the Queue Scheduled observing mode we use with MegaCam. Instead, we make sure that down-stream hardware of the Detector Hosts are also redundant.]

2.2.2 Session Host

The Session Host runs all other acquisition software and user interface components. Images are transferred across either a 1 GBit switch for MegaCam, or 100 MBit switch for other detector systems. An alternative to the Detector and Session host could be one computer, but since the Session Host also houses the **RAID disk** storing most of the data and experiences more unpredictable *bus* loads, there may be complications with that approach. The Gigabit Ethernet solution has already been shown to be one viable option, and is currently being used for MegaCam (and most of the data reduction network at this time.)

¹A "host" refers to a distinct computer box running some kind of operating system.

Figure 3: Computing Network: 4th Floor



As the image is read out from the array by the Detector Host, it is simultaneously copied to the Session Host over the Ethernet Link, and is therefore both on disk, and cached and ready for fast **image display** by the Session Host within seconds of the end of the readout. **Instrument control** and high level **sequencing** may also be handled by the Session Host. Any quick **data evaluation** required by observation and needing fast access to the data may run here as well, but for MegaCam, separate real-time elixir systems exist for this purpose but in Waimea and at the Summit. The Session Host is the computer with which the user has the most interaction.

2.2.3 Display Host

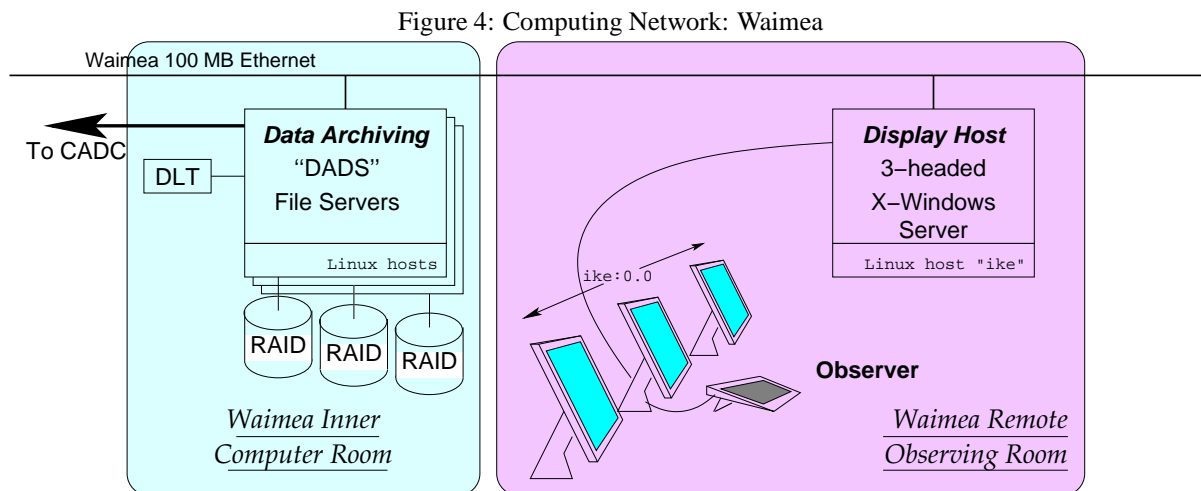
The Display Host for the observer runs a multi-headed X-Server. No other processes run on this machine. Hard drives may or may not be present in this machine. If they are they will only be used for temporary or local swap space. The network speed requirement for this host will never exceed the needs of X11 traffic to update the screen. Three flatpanel screens, currently 1600x1024 pixels each, connect to three digital-output video cards in the Display Host. A single keyboard and mouse controls all three as one virtual screen of 4800x1024 pixels. (More on this in a later section.)

2.3 Waimea Headquarters

A high bandwidth network (DS3) connects the Summit and Waimea computing facilities.

2.3.1 Remote Display Host

In Waimea, an identical display host is located in the Remote Observing Room. Once again, all processes displayed here run far away, on the Session Host at the summit. A method to duplicate the contents of Summit Display Host's screen in Waimea (Virtual Network Computing) has been explored, but the approach of making each control interface capable of having multiple instances is being favored. Such a solution is currently being used for both graphical (X11 or Web based) and command line interfaces. Like the Summit Display Host, the network speed and latency needs of this Remote Display Host are limited to screen updates, and are already met by our DS3 network.



3 Hardware Platforms

Most new computers being purchased by CFHT are Intel-based. In fact, almost all of them were a single model from Dell based on the Pentium III and Intel chipsets. They have proven to be very versatile. They fill the role of Display Host, Session Host, and Detector Host well. This makes the individual machines (and the parts inside) act more as building blocks than hardware purchased for a specific task. While the extremely competitive PC market has some products with the wrong corners cut, properly engineered hardware is not so difficult to find. We opt out of most maintenance contracts and put the total saved toward buying complete redundant spares to cover potential failures. Over several years of purchasing dozens of workstations, none have had system failures (power supplies, CPU fans, or other) and a small percentage of hard drives which were purchased separately have developed problems.

With any approach to purchasing computer hardware, it helps to have an understanding of the factors which can affect critical machines adversely, especially those at the summit:

- Unreliable memory (we use only certified, error-correcting RAM)
- "mini" CPU cooling fans (Dell's workstations use larger fans which live longer)

- Marginal chipsets (We use only Intel chipsets)
- Marginal motherboard designs (Intel's own and Dell's designs have been reliable)

Approval for use at the high elevation of Mauna Kea (4200m) is not always easy to find, so we often are left to determine that one experimentally on our own. We have had consumer-grade PCs (serving functions not related to operations) at the Summit for years and experienced very few problems with them.

IA32 (Pentium) computers, even different models from the same manufacturer, can be leagues apart in reliability and design. The top ones are comparable to workstations from Sun and Hewlett Packard, but lower in cost, especially when considering upgrades. Because "PC" designs also have the momentum and competition of the very large consumer PC market behind them, we're counting on the processors and architectures to continue to be a driving force in the future. This means we make a software system that runs at an acceptable speed today, and PC companies do the work for us to make it run faster in the future.

4 Operating Systems

Hardware is also only as useful as the supporting software which exists for it. The wide range of support for PC hardware under the Linux operating system is what makes the total combination of Intel hardware and Linux an attractive package. We also choose our hardware (network cards, graphics cards, etc.) based on whether properly written Linux drivers exist. With the wide selection of hardware available, we can afford to do this easily, and still end up with very cost effective solutions.²

For data acquisition, **Unix** has been the environment of choice at CFHT for many years. On the Intel machines, the **2.2 Linux kernel** is an extremely stable platform, and interoperates well with our existing Sun and HP machines which are running Solaris and HP-UX. To the end user, there is truly no difference in using a Linux environment today as compared to any other modern Unix. Linux likely will, and largely already has become the standard Unix. (Such statements caused mixed and mostly skeptical reactions a few years ago, but are now being taken more seriously.) Even though Unix itself may be "old", nothing better has come along yet *for the job we do*. Most of our older machines are now being replaced by Intel/Linux. The figures in the previous section illustrate a target which will probably be reached before the end of 2002.

Our Detector Hosts run a special version of Linux called **RTAI Realtime Linux**. This is a hard (deterministic) real-time operating system which was required to handle the strict requirements of reading out MegaCam's controller. RTAI Realtime Linux lacks some of the features of a real-time operating system like OS9 or VxWorks, but it provides exactly what was needed for MegaCam (deterministic response in one place in an interrupt handler.) Moreover, it lacks *none* of the features of regular Linux. From a programming and maintenance standpoint, the single difference is the Detector Hosts boot a different kernel image file.

5 Acquisition Software Philosophy

To be written . . .

²For example, the **Gigabit Ethernet** mentioned earlier in use by MegaCam is currently a card from **Sysconnect** which has had a stable and well supported Linux driver that has been part of the "stock" Linux kernel for years. Using a standard (copper) crossover cable between the Detector Host and the Session Host, we transfer images at a rate of 32 MBytes/second (during a readout) even with overheads of the NFS protocol. FTP transfers to a RAM disk proceed at sustained speeds of 55 MBytes/second. These cards are available for \$500 or less each.

[This section should include discussion of how choice of software affects operations, maintenance, and most importantly how well we reach our science goals.]

Purpose of this section. Describe the general approach of the Canada-France-Hawaii Telescope Corporation's software group. There are two main reasons for having this document. First, many of the goals, philosophies, and policies described below are believed, in general, to be the most appropriate for solving problems in our field. Choices made for a mature, ground based telescope may differ from those for a space program or industry, Second, we believe that having a group which shares as many common philosophies as possible – whatever they may be – makes for the most efficient working situation.

5.1 Goals

1. Meet the astronomy requirements.
2. Flexible implementation.
3. Very low down-time.
4. Long-term maintainability.
5. Easy to understand interfaces.
6. Accessible interfaces (mainly acquisition side - provide easy remote engineering)
7. Open framework (to contributions from non-software staff)
8. At a reasonable cost in both money and employee time.

5.2 Philosophy

(Ways to get to the goals)

Since the company has limited money and resources, we must agree on a balanced approach to meeting as many of the goals as possible. Here is the general philosophy on meeting each of the 8 goals above:

5.2.1 Astronomy requirements

Ideally, we would receive these requirements. In reality, it often falls on us to "extract" the requirements from the astronomer. As part of collecting the requirements, both priority, and a rough idea of how long it will be needed are important. Often, we'd rather implement the must-have list completely first because the overhead in maintaining "quick hacks" can become overwhelming later on.

This leads into why we determine how long something will be used as part of gathering requirements. If a solution doesn't have much potential to be re-used for something else in the future, and may only be used for a matter of weeks for a special purpose, sometimes a "hack" is appropriate.

5.2.2 Flexible implementation

The fact that requirements do change often in our field reflects not indeciveness on the part of scientists, but the nature of science itself. Having strict requirements that are static for years could severely limit discovery. As a result, flexibility in software implementations ranks high.

One way to meet this goal is to make sure that more than one person can always react to user requests for changes on any system. For the acquisition and analysis systems, the time scale for these changes or additions is the most demanding. We strive to meet this goal by (a) spreading knowledge, and (b) designing in a way to effect a change that is simple as possible.

5.2.3 Low down-time

[This applies mostly to the acquisition system. Way to rewrite it to be more encompassing?]

a. No immediate dependencies on anything outside the summit network. b. Redundancy where possible c. Often acceptable to cover (a) OR (b) at any given time. E.g., redundancy can be provided by a Waimea computer, such that when we're on the back-up, we're temporarily in violation of (a). (Due to the fact that we feel double failures are unlikely enough not to be worth trying to cover.)

5.2.4 Long term maintainability

a. Use stable, mature tools and languages. b. Avoid choices which require a specific commercial product. c. Where possible, use things which are widely known by software engineers or anyone else we are likely to be able to hire in the future.

...

Framework should be open, simple to learn.

Re-use existing framework, or upgrade it if it doesn't meet a new need.

5.3 Policy

1. Hire "general purpose" people. All members are expected to have a general base of knowledge that lets them share in a lot of tasks. Specialized person with a very clear corner of responsibility is more likely to be unevenly loaded.
2. GPL -
3. Make smooth transitions.
4. Support multiple architectures.

[The rest of the sections need to go into a fair amount of detail about everything from X-Server "depths" to Status Servers . . . This is just a *rough* start for the section headers right now, and I know it is not well balanced/filled out yet. Please give feedback.]

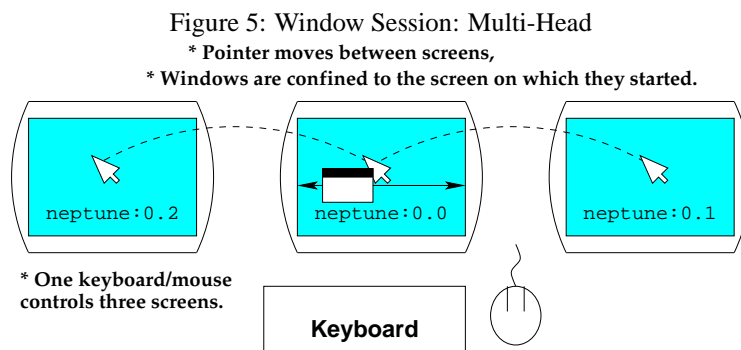
6 Session Tools

6.1 Window Session Concepts

The following sections provide details about our window manager and multi-head X-Server setups. This information is intended for anyone in the future changing things such as window manager scripts, multi-head layouts, or observing session window placement rules (all of which are more complicated when there are three monitors per terminal!) There is a separate document on the Web with more user oriented information for local users who just want to customize their desktop colors or window manager menus. The information here is more low-level, and intended for setting up observing sessions. It is in the interest of anyone developing user interface software for use at CFHT to at least be aware of these basic layers of the window session.

6.1.1 Multi-Head

As the name suggest, Multi-Head involves multiple monitors connected to the same computer, with a single mouse and keyboard. With XFree86, a "ServerLayout" section in the XF86Config file tells the X-Windows system how the monitors are physically located relative to each other.



A basic Multi-Head configuration will come up with one "screen" per monitor. Although the mouse can travel between monitors, windows are confined to the screen on which they were launched (see Single Logical Screen and Xinerama for the New Way.) It is possible to select the screen on which windows are launched by setting the DISPLAY variable or -display option for the application. Assuming windows are launched from the local host:

```
-display :0.0
```

or

```
-display :0
```

selects the default screen, while

```
-display :0.1
```

and

```
-display :0.2
```

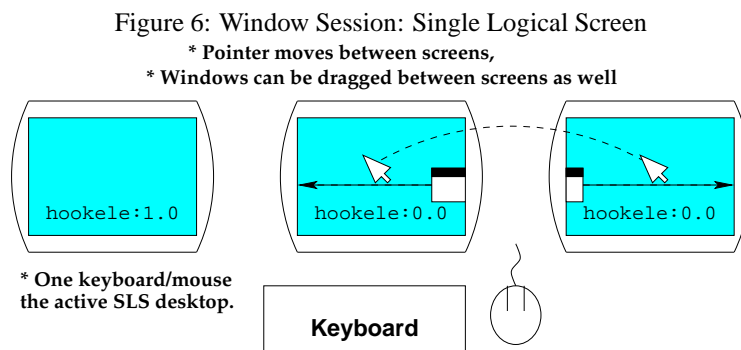
select the extra screens. If they do not exist, such a display setting will cause a failure. Since each screen has a different default value for the DISPLAY variable, no -display option results in a window being launched on the current screen, as expected. (The DISPLAY variable is inherited from the window manager. In the case of FVWM1, there are three window managers running if you have three multi-head screens.)

Figure 5 shows the particular configuration used on our Hewlett Packard 9000 display host “neptune” on the summit.

A small, but handy piece of free software called **x2x** exists to allow separate X-Terminals to be joined into a similar Multi-Head configuration (where applications remain confined to the terminal on which they began.) As in figure 5, only one keyboard/mouse is used to control up to three screens, but the unix DISPLAY names would be something like “xt1:0.0” “xt2:0.0” and “xt3:0.0”.

6.1.2 Single Logical Screen

Another HP on the summit used as the Telescope Control System’s display console, “hookele” has two of its screens merged into one logical screen using HP’s SLS. The main advantage is that windows can be dragged between these two monitors, which are now both seen as one, wide DISPLAY=“hookele:0.0”.



It then becomes the responsibility of the window manager to know where the divisions between the monitors are, or else windows will tend to pop up right on the split, or on completely different screen from the user’s mouse cursor, which can be confusing.

At the time we installed “hookele”, which is the example shown in figure 6 we made our own, custom SLS patch to the FVWM 1.24 window manager to handle this case. The patch only handles two 1280x1024 monitors put together in a 2560x1024 logical screen.

6.1.3 Xinerama

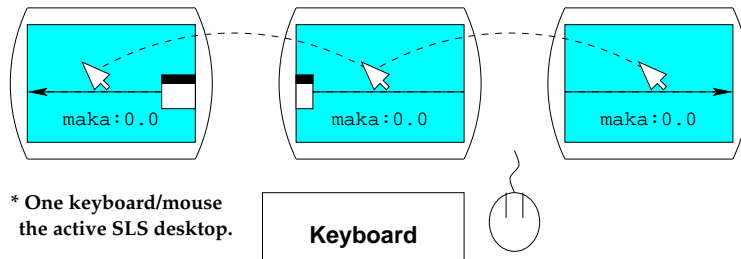
Xinerama and **PanoramiX** are associated with new features of the X-Servers used with Linux. They currently provide some of the same functionality as Hewlett Packard’s SLS. With XFree86, a Xinerama configuration uses the same XF86Config file as a similar Multi-Head config would use, but with a special flag or option to turn on the Xinerama feature.

The Xinerama extension also provides a way for a window manager to query the server about the physical layout

Figure 7: Window Session: Xinerama

* Pointer moves between screens,

* Windows can be dragged between screens as well



* One keyboard/mouse the active SLS desktop.

* Window Manager is "aware" of splits between the monitor so that global screen or xinerama-screen-relative coordinates can be used to place windows, and window manager placement and movement functions can be programmed to avoid putting windows on the split.

of the monitors making up the logical screen. Version 2 of the FVWM window manager is one of the "Xinerama-aware" window managers currently available. Without any special options or hacks, it automatically handles things like placing windows on the same screen as the mouse pointer, or providing "EdgeResistance" to help snap windows to the physical edges of the monitors. This is more important than it may at first seem. Consider the case where the user is working with a graphical tool that opens a confirmation popup. If this popup appears two screens away where the user may not even notice, this is a Bad Thing.

Figure 7 shows the current configuration of the new Linux Display Hosts maka (at the summit) and ike (at Waimea headquarters.)

6.1.4 Screens

6.1.5 Xinerama Screens

6.1.6 Virtual DeskTop and The Pager

"Virtual" window managers FVWM1 and FVWM2 support a concept of a "DeskTop" which is comprised of equal, screen-sized "Pages". In our default configuration, the size of the DeskTop is larger than the screen.

The "Pager" shows the user which page of the desktop is currently visible, and allows them to move the viewport by clicking on a new section of the desktop. The Control+ArrowKeys have also been configured to move by a page at a time in the direction of the arrow pressed.³

³Pages vs. Desks: FVWM in particular supports the concept of multiple DeskTops, all comprised of NxM screen-sized pages each, but we don't enable more than one DeskTop because things can become confusing very quickly that way, unless a DeskTop is 1x1 screen. It would have been perhaps nicer to use "Desks" instead of "Pages" because each Desk can be labelled by the user, similar to the "Workspaces" of CDE. The only problem with this alternative was that the key bindings mentioned above allow the user to pan off-screen, since they don't consult with the FvwmPager module to see how many Desks are actually displayed. Binding the key to moving by a Page rather than a Desk does do boundary checking though, so I chose that.

6.1.7 Window Placement

6.2 X-Server

It is worth mentioning a few features of the “XFree86” X11 display servers which we currently run on our Linux platforms. We originally purchased a commercial X-Server from Xinside Graphics for our first Linux multi-head display, but today aspects of the free X-Server project have surpassed the commercial version, so all of our displays run either XFree86 version 3 or XFree86 version 4.

Probably the commercial X-Server has caught up, and perhaps added other useful features. Our license should still work with new copies of the product if anyone wishes to experiment. For example, recently on a Sony TR2A laptop we found that the Xinside server was the only way to support its full LCD resolution.

6.2.1 Drivers

Because of the wide range of AGP and PCI graphics hardware available for PCs, XFree86 version 4 uses loadable modules for each type of graphics card. To simplify installation of the server in our older, RedHat Linux distribution, which was not based on this model, we have chosen an option which builds all drivers statically into the X-Server binary.

Graphics cards and drivers various features which can be interesting to astronomy applications. The following sections list some of these features and the current state of support for these features. Clearly, both the graphics card and the driver for that card must implement the feature for it to be useful to us.

6.2.2 8-bit PseudoColor

This display mode uses an 8-bit wide (256 entry) color-lookup table. Only 256 colors may be displayed at a time for the whole display. They are chosen from a 24-bit color palette. The color palette could be configured to show 256 grey intensities at a time, or a selected set of colors, or a combination of the two, but the total number of unique colors on the screen can never be greater than 256. Since the color palette must be shared by applications on the same display, there are often problems with applications not getting enough “color cells.” Various models for sharing colors and replacing the colormap with the one of the active application (resulting in “flashing” of the display) exist, but none of these seem worth it today, when memory costs around 10 cents per megabyte.

6.2.3 24-bit TrueColor

All of our new displays and display drivers are capable of using a mode where the intensity of red, green, and blue can be set separately for each pixel to a precision of 8-bits for each primary color. Essentially, any pixel can be any color. This most likely the mode we want to run, but there is a problem.

While PseudoColor mode was mainly invented to save memory, many of the image display tools our users are used to rely on a “trick” which requires manipulation of the current colormap to enhance contrast in different parts of the displayed image. This does not work in TrueColor mode.

6.2.4 24-bit DirectColor

DirectColor mode may provide a solution to this problem, but would require some minor modifications to the image viewer applications. (Some applications such as IDL already support this mode.) DirectColor also uses 24-bits per pixel, where 8-bits represent red, 8-bits green, and 8-bits for blue (just like 24-bit TrueColor), but the final red, green, and blue intensities for the actual pixel are obtained by looking up the value in a lookup table, much like PseudoColor mode.

When these three lookup tables are all initialized to a linear ramp, DirectColor mode is equivalent to TrueColor. When they are initialized to the appropriate gamma curves, they can be used to perform a final gamma correction of the intensity for a particular monitor. For our application, the lookup table can be used exactly as the lookup table in PseudoColor mode by simply setting the same value for Red, Green, and Blue when drawing the image.

We have not determined which of our graphics cards and/or drivers support this mode.

6.2.5 Overlays

Overlay mode is another attractive solution allows an actual 8-bit PseudoColor mode to co-exist with 24-bit TrueColor. Both exist at the same time in two different frame buffers, and the X-Server maintains a mask of which pixels on the screen are currently using which color model. With this option, no modifications to old applications requiring 8-bit PseudoColor would be required, while applications which can make use of more “modern” color models can request TrueColor for their windows at the same time.

Both types of digital video cards which we have used with our 1600SW flatpanel screens support Overlay mode, but only the X-Server for the 3DLabs card implements the feature. The Number Nine X-Server which we use for both three-headed displays does not.

6.3 Window Manager

Window Session software running on the Session Host has been kept purposely light weight. We do not use the CDE, Gnome desktop (which may become the successor to CDE), or KDE. In fact, we run no desktop/session manager at all, only a basic window manager.

FVWM. It has been configured to emulate the Motif Window Manager (MWM) just as the CDE does.

This and a few terminal windows are typically the extent of “standard” desktop software an observer needs to do their work. Features like saving workspace states and high level color management are often even undesirable in our environment.

Almost any display in the building (including PCs running an X-Server such as Exceed) can be used as the Display Host. A “virtual desktop” feature of the window manager can be enabled by the user to gain more desktop space when running on a single monitor.

6.3.1 MWM

The Motif Window Manager is a classic X11 Window manager, and has been the standard for CFHT’s “Pegasus Sessions” in the past. The CDE’s window manager, **dtwm**, has the same look and feel as the Motif Window Manager.

6.3.2 FVWM 1.24r

FVWM is another manager with the ability to emulate the behavior look and feel of Motif Window Manager, making a transition from **mwm** to **fvwm** transparent to users. FVWM uses less resources than MWM and has additional features, some of which are described below. Even at the time we started using FVWM 1, it was no longer supported. This did not turn out to be a burden. We did add some trivial patches to the source code to improve handling of configuration files and behavior of the window manager on our first Single-Logical-Screen X-Server (see below.) It was the simplest and most stable window manager which could run the same way on our Solaris, HP-UX, and Linux platforms.

6.3.3 FVWM 2.4

FVWM version 2.4 introduces proper support for arbitrary Single-Logical-Screen configurations through an X-Server extension. In addition, since FVWM 2.4 is actively supported by its developers, CFHT's does not have to maintain a specially patched version (not that this was a very big problem, but it still helps.)

7 Infrastructure

7.1 Status

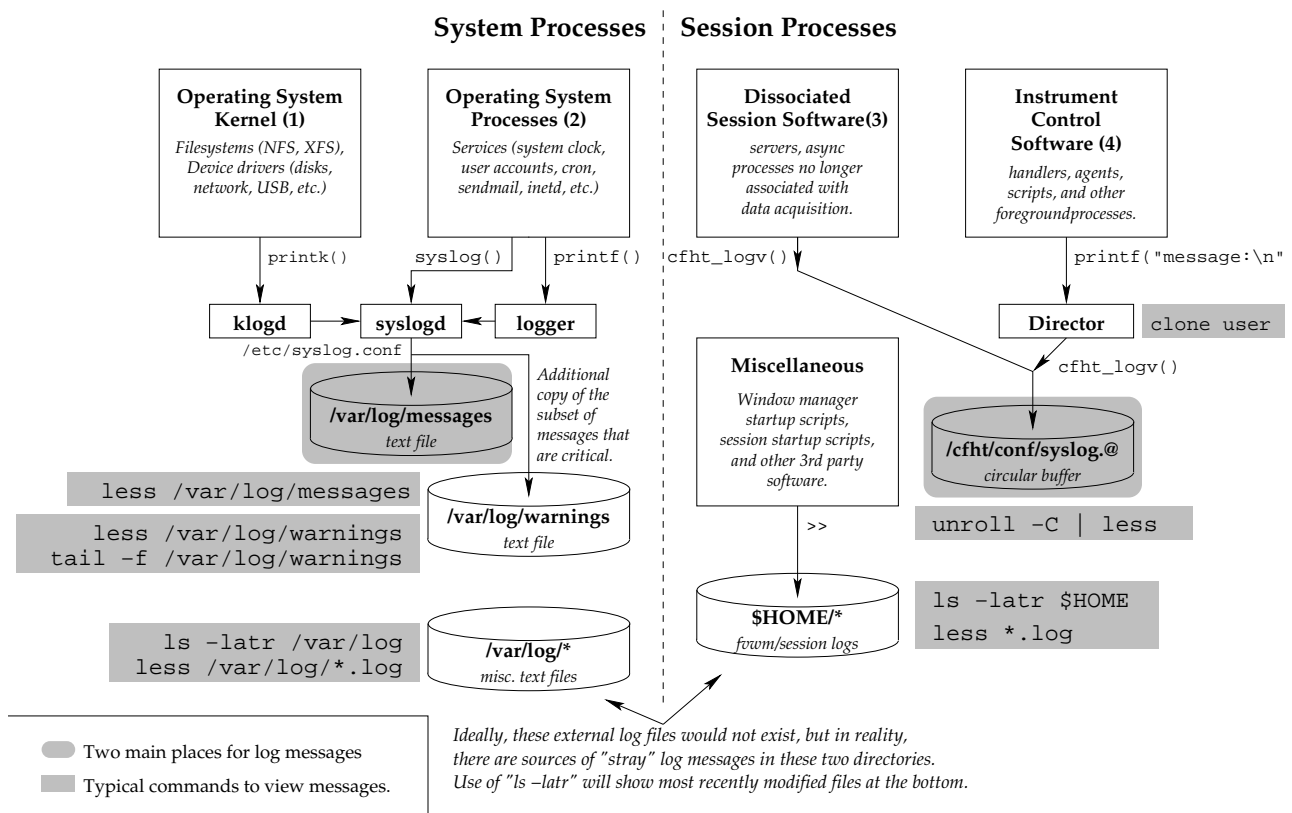
7.2 Feedback and Logging

Programs need to send text feedback messages to the user. Other messages also typically need to be recorded to a log, for later use in debugging, etc. For the developer, it is important to consider the complete set of sources of these messages:

1. Operating system kernel
2. Operating system processes
3. Application software not associated with any session
4. Application software associated with a specific session

Figure 8 illustrates the flow of messages, including the calls programs must make to log, and ways to view the messages.

Figure 8: Sources of Logging Messages



Ideally, items 1 and 2 all use the `syslog()` interface (see “man 3 syslog”). A standard unix process called the `syslogd` captures the messages and routes them to the console, a file, or another machine. Programs which generate messages on `stdout` can be piped into a standard unix tool called `logger`, which redirects each line to the `syslog` interface. The file `/etc/syslog.conf` on the local host determines the behavior of the `syslogd`. At CFHT, we direct everything to the host called “lawelawe”. (Some hosts may still be logging to its predecessor, “central-services.”) On lawelawe, and on any machines which do their own logging, all messages are appended to the file `/var/log/messages`, while a subset including only warnings and errors is appended to `/var/log/warnings`. Once `syslogd` has received a message and appended it to one or both of these files, the original “severity” with which the message was logged is lost. It then becomes difficult to later pick out the important messages from `/var/log/messages`. This is the reason we also log the subset of more important messages to `/var/log/warnings`.

The above being the ideal for system software, the ideal for application software items 3 and 4 is similar. Dissociated application software uses the `cfht_logv()` interface – similar to unix `syslog()` – which relays messages to a process called “roll” and puts them in a circular buffer on the local machine that can be viewed with a program called “unroll.” To make the message format shown by “unroll” similar to `syslog`, a “-C” option has recently been added to this utility (C for “Cooked”.) Messages are displayed in reverse chronological order. To see them in Forward order, use “-F”, and “-h” takes an argument for the number of hours to go back. So, for example, to see the last 5 hours:

```
unroll -CFh 5 | less
```

Other options controlling the behavior of `unroll` can be found in the man page.

Software developers outside of CFHT are typically writing item 4 programs, i.e., the session and instrument control software. The most basic logging option for this class of software is to use a simple `printf()` interface (or, in any language other than C, the appropriate calls to write lines to the program’s standard-output.) The Director shell manages each of these instrument control programs (as Agents) and logs whatever they print on `stdout` and `stderr` to the `cfht_log` facility. Thus, output from instrument control tasks is logged together with output from dissociated tasks on the session host. It is all accessible in one place, using `unroll`. Even when instrument control is distributed among computers other than the session host, the director model allows all messages to be logged centrally, on the session host.

Dissociated programs which do not happen to run on the instrument’s session host can be one exception. An example of such a program is the TCS server. Often, the session host and the TCS server host are not the same. To piece together messages from telescope control, one should use the `unroll` program on the TCS server host as well.

7.2.1 Message Types

In order to have program messages classified appropriately, the use of `printf`’s in your program should follow some guidelines:

- All messages and variable names should be in English.
- Each message should be prefixed by one of the “keys” below.
- Each message must end in a trailing newline (a `\n` is sufficient, though DOS newlines are acceptable too.)
- Word-wrapping happens automatically, so do not embed additional newlines or control characters in the message. (If you need to force a message to continue on the next line, the “key” prefix must be included again, or the message type classification will be lost.
- When including values or other strings in a message, inclose them in opposing single quotes, like this:

error: File 'filename.txt' not found.

alert:

The alert message type is reserved for critical conditions that may require immediate attention from an engineer. This includes things like thermal overloads which could cause damage. Since these conditions could occur with no operator nearby, alert messages trigger a special “alert” program to be launched. The alert program is capable of calling pagers or sending e-mail notification (based on the actual message of the alert.)

Alert messages appear in red, just like error messages, but trigger the alert script which displays a timeout popup (to allow warning e-mails to be prevented.)

error:

Displayed in red (and terminal beeps/de-iconifies). Top level command request has failed. Read warnings above for details.

warning:

Displayed in yellow. Something is not 100% normal and may cause bad results down the line. (But request itself hasn't failed yet.)

status:

Displayed in green. Usually a confirmation that a command succeeded.

7.2.2 Additional Message Types for Agents

progress: ... (..%)

This type of message should be sent by any agent just before it begins a blocking operation (e.g., waiting for hardware to reach a given position.) If it is possible to determine how close to completion the operation is, then a percentage can be added in parentheses, just before the newline in this message. In that case, the program may wish to print a progress message at about once per second to keep the user informed of the progress.

statusbar:

Reserved for use by DetCom in most sessions. The most recent statusbar: line received from any agent is displayed just above the user's prompt in reverse-video or against a blue background.

info_:

The _ above is actually a digit, 1..9. Depending on the digit, this message type replaces one of the 9 optional status bars visible *below* the users prompt against a grey background. Currently, there is no clean mechanism to manage which agent processes should update which lines in the “info panel.”

7.2.3 Additional Message Types for debugging

logonly:

Not usually displayed except in “verbose on” mode. Mostly useful in tracking down problems with the instrument or elsewhere.

debug:

Not usually generated except in “debug on” mode. Useful to track problems with the software itself.

7.3 Director

8 User Interface

We have in the past, and continue to use a variety of tools to build our user interfaces. Some considerations in choosing these tools have been:

- Stability.
- Cost and licensing.
- Remote accessibility (from Waimea, from home).
- Ability to have multiple instances.

8.1 Command Line Interfaces

At the heart of CFHT's Pegasus observing system were network-based clients (handlers) and persistent servers. Similar to the command-line runnable handlers of Pegasus, the New Environment for Observing (NEO) is based around persistent command line interpreters. Like the handlers and servers of Pegasus, these can be developed and tested independently, and graphical user interfaces can be layered on top.

Only one command line interpreter runs on the Detector Host: DetCom. This program communicates with the camera and writes FITS files (via NFS) to the Session Host's disk.

Other command line interpreters to control instrument functions will typically run on the session host, and deposit information into a status server (or temporary file). This information is collected and merged into FITS headers by high level scripts, which also run on the session host and sequence everything.

8.2 Graphical Interfaces

8.2.1 Image Display

ds9 ... Session Host. (Display Host ... slower access to data, size of which is constantly increasing. Bandwidth needed for display is limited by display resolution.)

8.2.2 Form-based Interfaces

9 Programming Languages

9.1 Bourne Shell

Bourne shell for scripts.

9.2 C

C for important libraries.

C is also best suited for the command line interpreters, because command line interpreters implement a very procedural model.

9.3 The Rest

C++ has been used in Director and RPM.

QSO User Tools are implemented in Java.

10 Development Tools

10.1 GNU make and Make.Common

GNU make is freely available and works well on all of our Unix platforms, so we actually *replace* the vendor-provided make to avoid confusion and problems with `Makefile` compatibility. (The vendor-provided make with Linux systems is already the GNU version.) There are Info Pages and Man Pages installed on our Web site.

Make.Common is a Makefile intended to be *included* by all project directory Makefiles. It is designed to make the project Makefiles as simple as possible. To use it, include it *twice*, using the `include` directive, in your regular Makefile:

```
# Example Makefile
# Include Make.Common to set all make variables to defaults:
include ../Make.Common

# ... Project specific variable definitions and targets go here ...

# Include Make.Common again to define the targets for make:
include ../Make.Common
# End of Makefile.
```

(See Make.Common itself for more examples.)

You can only have one **compiled executable or library** *per subdirectory* and the executable or library must be named the same as the directory it is in. You can, however, create a directory structure as deep as is needed to group related programs and libraries together. Just remember to create a symbolic link to `Make.Common` in the level above whenever you make a new directory tree. (See how `/cfht/src/pegasus/cli/` is set up for some examples.)

Shell scripts must all be named `*.sh` and appear in a subdirectory anywhere in your tree called `./scripts/`. There can be multiple `*.sh` in a single `./scripts/` directory. When these are

installed in the `/cfht/bin/` directory, *the .sh is stripped off*. So they are invoked as their basename, i.e., without the `.sh`.

Similarly other **non-compiled files** like configuration files, parameter files, and bitmap files are kept together in a directory called `./confs/` and will get copied to `/cfht/conf/` during a “make install” if needed.

10.1.1 Standard Targets Provided by Make.Common

make momma

This target is only valid in the top-level source directory. It is intended to build and install *all* subdirectories. We recommend typing **make** first (without any explicit target names) to verify that files will be installed in the correct locations. At the top-level, **make** will only print a list of target directories.

make preinstall

This exists to make complete releases build properly. It is used mainly to install header files. For example, if library A and library B both need each other’s header files in order to compile, this is the way to deal with this chicken-and-egg problem. The `make momma` target will take care of calling this at the right time.

make world

If you have a project that is divided into other subdirectories and you run “`make world`” in the project directory, all subdirectories will be built with “`make all install`”. “`make world`” in one of the subdirectories will change to the upper directory and run “`make all install`” from there. So “`make world`” can be used anywhere within a sub-project to build the whole thing.

make all

In a project directory “`make all`” traverses into all the subdirectories, where it builds either the library or the executable for that subdirectory. Executables are not copied from `/cfht/obs/project/*` to `/cfht/bin/`, but some libraries *may* be installed `/cfht/lib/` if another executable requires them.

make install

In a project directory, this runs “`make install`” in all the subdirectories. In a subdirectory, it installs (and builds, if it is out of date) the library or executable in `/cfht/lib/` or `/cfht/bin/` respectively.

make clean

Removes the directory `/cfht/obs/project/`, where object files for the current project are kept.

make depend

The Makefile must not be checked into RCS. Running “`make depend`” will append or replace a list of C source file dependencies (generated by “`gcc -MM`”) to the Makefile.

make execlist

Prints a list of all the programs with their runnable names (shell scripts will have the `.sh` removed) from this directory level down. This is the list of things that gets versioned and installed in `/cfht/bin/` during a “`make install`”.

make titles

Generates emacs-compile-buffer-parsable output of the results of a search of your code for “====” and “%%” markers. Files in each subdirectory are expected to be summarized in a file called “Index”. (See `/cfht/src/pegasus/cli/Index`, for an example.) Copyright headers are automatically inserted into known file types, and DOS newlines (if found) are changed back to unix newlines. The modification time-stamp on the file is *not* altered.

make tar

Makes a tar file snapshot of the current directory tree, excluding RCS subdirectories and emacs backup files.

10.1.2 Ways to turn on other options from the command line**make TARGET=VXSPARC**

Cross compilation! Assuming there's a block already in Make.Common for TARGET=VXSPARC, this will select a different compiler, linker, and obj.CROSSCC/ directory to place the binaries. No modifications needed to your Makefile. Add new cross-configurations to Make.Common.

make /cfht/obs/program/program-pure

Assuming “\$(EXECNAME) \$(EXECNAME)-pure: \$(OBJS) . . .” appears in the Makefile instead of “\$(EXECNAME): \$(OBJS)”, then this will produce a purified copy of the program.

make EXTRA_CFLAGS="-pg”

If you have already used EXTRA_CFLAGS in the Makefile, this will override. The example above causes the program to be compiled with gprof profiling support. A “make clean must be done before the first pass to force all objects to be rebuild with the -pg option.

make EXTRA_CFLAGS="-DDEBUG -DDEBUGDEBUG”

Use this notation to include any arbitrary defines, for a particular invocation of make only. Don't worry about the fact that these extra flags get passed even at link time; gcc will simply ignore them where-ever they do not apply.

make EXTRA_CFLAGS="-g” ?????

Don't do this, and don't try to remove the ‘-g’ from the default options either. GCC can handle doing optimizations while also putting in debugging symbols. The installed versions are always stripped, however, so run the one in the project's ./obj/ directory if debugging symbols are required. It is even possible to analyze cores dumped by one of the stripped versions if you rebuild the exact same executable with symbols again (or if you still have the unstripped version in the ./obj/ directory. Here's an example of how gdb could be invoked on one of these cores:

```
gdb --symbols=/cfht/obs/dumper/dumper --exec=/cfht/bin/dumper --core=.../core
```

10.1.3 Make Variables**VERSION**

Normally this is the date, in the form YYMMDD, but you can override this with something like 1.1 if you prefer.

CONFS

Contains *.def *.par *.bm *.xbm *.rdb *.xrdb by default. Any matches which exist in a subdirectory called ./conf/ are copied to /cfht/conf/ during “make install”. Do not include the ./conf/ in the filenames or filename patterns here.

SCRIPTS

Contains *.sh by default. To limit the list to only certain .sh files within your scripts/ directory, list only the names of those files, but they *must* currently be .sh files.

SUBDIRS

If you define this, only SCRIPTS and CONFS are installed at this directory level, and then each of the directories named in SUBDIRS are traversed. Do not list “scripts” or “conf” in this list. They are handled automatically. Also, all built programs must live at the “leaves” of the directory tree, so there cannot be a program at a directory level which also defines SUBDIRS.

SRCS

Contains *.c and *.cc by default. OBJS (the list of *.o is generated from automatically SRCS. If you override SRCS, OBJS will be adjusted accordingly.

HDRS

Contains *.h and *.hh by default. If you have internal-only header files, then you can either place those in a subdirectory called ./internal/ and #include them with internal/foo.h, OR explicitly override HDRS in your Makefile.

EXTRA_CFLAGS

This variable is normally empty. Anything in it will be passed as extra options during the compile and link stages. If there are things that do not belong in CCDEFS or CCINCS (below) they can go here.

CCWARN and WERROR

The default for CCWARN is "-Wall -Wstrict-prototypes". The default for WERROR can be defined to -Werror in a separately included top-level Makefile, **Make.Local**. Once a project has compiles cleanly without any warnings, a good way to keep it that way is to make warnings equal errors for that project:

```
CCWARN += $(WERROR)
```

If there is a Make.Local in the toplevel src/ directory which defines "WERROR = -Werror", as we have in our source tree at CFHT, then these projects will fail to build when warnings are introduced.

CCDEFS

This contains defines to be passed to the C source code. It can be added to using "CCDEFS+=..." in the Makefile. Be sure to use += instead of just = because any -Dxxx from Make.Common itself are important to the build process. Exactly one of the following will be defined, if a known system type was detected:

```
-DSUNOS SunOS 4.1.x
-DHPUX HP-UX 9 or 10
-DLINUX Linux 1.x or 2.x
-DCYGWIN32 Cygnus win32 on Windows NT
```

So you can do things like this in the C code:

```
#include <sys/socket.h>

#ifdef SUNOS
/* Old Suns lack prototypes for these... */
int socket(int domain, int type, int protocol);
int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
int listen(int s, int backlog);
#endif
```

. . .

It may also be useful to know the date of the most recently modified source file, the compilation date, and the version number from the Changes file inside the C source. All three of these can be added by putting the following in the Makefile:

```
CCDEFS+=$(VERSIONDEFS)
```

And then the C program would see the following three defines:

```

int
main(int argc, const char* argv[])
{
    printf("Program FOO version %s %.2f %s (compiled %s)\n"
        VERSION, SOURCEDATE, BUILDDATE);
    . . .
}

```

The above example assumes the version number should have two digits displayed after the decimal. Note that **VERSION**, **SOURCEDATE**, and **BUILDDATE** are only available if the project Makefile adds **VERSIONSDEFS** to **CCDEFS**.

In addition, the following **-D**'s may come from **Make.Common** under the following conditions:

```

-DNO_CFHTLOG - If /tmp/pipes/syslog.np was not found at compile-time.
-DUSE_EPICS - If this machine is one on which we want EPICS channel access.
-DHACK_SELECT - If target OS (HP-UX 9) has messed up select() prototype,
                requiring use of SELECT() macro in place of select().
                (In other cases, missing_protos.h defines SELECT to select,
                so always use SELECT() in your code. Only missing_protos.h
                needs to worry about HACK_SELECT.)
-DHACK_XLIBS - Another obscure work-around, only for un-patched HP-UX 9
                systems that have buggy X-libraries in /usr/lib. Only
                "hform" uses this.
-D__USE_FIXED_PROTOTYPES__ - Enables some extra prototypes in the gcc-fixed
                            include files on our SUNOS machines. Not of
                            interest to user code.
-DEXIT_FAILURE=1 - Only on SUNOS, which forgets this in stdlib.h.
-DEXIT_SUCCESS=0 - Only on SUNOS, which forgets this in stdlib.h.

```

On our primary development system, HP-UX 10, only **-DHPUX** usually appears, and possibly **-DUSE_EPICS**.

CCINCS

Use **"CCINCS+=-I..."** to append to the list of directories for include files.

CCLIBS

Use **"CCLIBS+=-L..."** to append to the list of directories to search for libraries.

CCLINK

Set this to a list of **"-llibname"** options for linking. For example, if your program requires symbols from **"libm.a"**, use

```
CCLINK += -lm
```

CCINCSX11, CCLIBSX11, CCLINKX11

These contain extra include paths, library paths, and library names, respectively, to be used with X-applications. Since vendors put these in different places, having these variables defined makes it easy within projects to indicate a program needs X by just saying:

```

CCINCS += $(CCINCSX11)
CCLIBS += $(CCLIBSX11)
CCLINK += $(CCLINKX11)

```

CCLINKNET

Any application that uses unix networking stuff (sockets, hostname lookup, etc.) should add the following to their Makefile:

```
CCLINK += $(CCLINKNET)
```

Because some platforms (namely new Sun Solaris) need to add "-lsocket" and "-lnsl" in the link stage for the proper network routines. Other platforms may need a "-lresolv" here also for host-name lookup. Simply using CCLINKNET takes care of whatever might be needed here.

10.1.4 Variables Useful in the Makefile

HOSTNAME

Contains the name of the host that the project is currently being built on, as returned by the hostname command.

TARGET

This is the equivalent of the "SUNOS", "SOLARIS", "HPUX" symbols defined for C programs, for use within the Makefile. If you wanted certain projects to be built only on a certain architecture, you could use this variable. It contains the name of the operating system, as returned by "uname -s" followed by a dash (-) and the major revision of the OS. For SunOS, this is either a 4 or a 5, and for HP it is either an A (version 9) or a B (version 10). For example, to have certain projects build themselves only on old Suns:

```
SUBDIRS = libdet cdma chmem deti detio lu trafficoff
ifeq ($(TARGET),SunOS-4)
SUBDIRS += detserver rdmem wrmem
endif
```

NO_CFHTLOG

If /tmp/pipes/syslog.np exists on the machine, this variable contains nothing. If /tmp/pipes/syslog.np is missing, this will contain "-DNO_CFHTLOG", and it will be passed to your C programs as well. If you want to do something differently depending on whether cfht_log facilities exist, you could do something like this in the Makefile, if you only need libcfht when cfht_log is used:

```
ifeq ($(NO_CFHTLOG),)
CCLINK += -lcfht
endif
```

And something like this in the C source code:

```
#ifdef NO_CFHTLOG
    syslog(syslog_type, message);
#else
    cfht_log(CFHT_MAIN, cfht_log_type, message);
#endif
```

10.1.5 Variables That Depend on Local Installation

Make.Common assumes that gcc is available. At CFHT, the following variables are set correctly for our installation, but at another site where gcc is not available (at least as a symbolic link) in /usr/local/bin, or in a case where gcc was not compiled to also use the Gnu linker, you may need to tweak some variables in Make.Common itself. (For all the variables mentioned previously, you override them in the individual project Makefiles, but for site installation stuff, you'll have to modify Make.Common).

DIR_GNU

If gcc is in /usr/local/bin or /usr/bin, there is no need to change this. Otherwise, you should set this to the path where 'gcc' can be found.

INSTALL

This must be set to the name of a BSD compatible install program. If you don't have GNU install on your system and you have a compatible install program, you can try putting it here (with full path). Otherwise it is recommended to download "fileutils-X.YY.tar.gz" from ftp.gnu.org or a mirror and install it. Be sure to configure it with ". /configure --program-prefix=g" so it doesn't use names that conflict with the vendor-provided file utilities on your system.

Although I wouldn't recommend it, it is possible to go through Make.Common and search for all \$(INSTALL)'s and replace them with regular "cp" or "mkdir" commands (depending on whether it is installing a file or a directory.) GNU fileutils are not difficult to install, so try that first.

CCLINK

The default setting for CCLINK under Solaris may be problematic if you do not have a gcc that uses the GNU linker. Search for "-Wl,-rpath,". This garbage is needed to ensure that shared libraries will be found even if a user has an incorrect LD_LIBRARY_PATH or if the program needs to run without a complete environment (e.g. from a cgi script.) I think the equivalent of "-rpath" for the Sun linker is "-R", but you can comment out the entire CCLINK setting as long as you have the proper search paths in your LD_LIBRARY_PATH and/or the program is statically linked with any libraries that are in non-standard places.

10.2 C Compiler

10.3 Symbolic Debugger

10.4 Verification Tools

10.5 Documentation

10.5.1 Web Manuals

Each document should have some kind of Web accessible version. A format which can be keyword-searched by standard search engines is also valuable. While many search engines can now scan PDF files, HTML is a much more widespread format, with built-in support in every Web browser client application. So HTML, or something which gets converted to HTML at the Web server end on the fly, is the most desirable format for the Web.

10.5.2 Printable Manuals

Since HTML Web pages do not contain page numbers, and usually also contain graphics that are already rendered at a low resolution (on the order of 100dpi for a typical monitor) they are generally not suitable for printing at 600+ dpi. A link to a hardcopy in either **PostScript** or **PDF** (preferably both) should be provided on the Web page.

10.5.3 Word Processors

MS Word or Frame files are *not* acceptable as distributable, printable copies. Many word processing applications have a save-to-HTML option, and also can create output for a PostScript printer or generate

PDF format.

These give mixed results, but as long as the results are decent for both the HTML and printable versions, any of these are acceptable.

10.5.4 XFig, L^AT_EX, L^AT_EX2HTML, ps2pdf

Writing documentation can be a little like processing astronomy data (though less interesting). We typically come up with a combination of tools and recipes that work well. This section describes the particular recipe used to generate this document. It produces reasonably good results, and has the advantage that everything is in plain text source which can be generated, preprocessed, and re-used in the automated fashions to which programmers and astronomers are accustomed. There is no mandate to use this particular method.

Figures are generated with **XFig**, a simple 2D drawing program. They are saved as **Encapsulated PostScript** with an appropriate scaling factor. Both XFig and L^AT_EX handle a variety of image formats, but a non bitmapped format like EPS gives the nicest results. (GIF, BMP, and JPEG are all bitmapped formats.)

The document itself is written as a L^AT_EX article. The document has this basic structure:

```
\documentclass{article}
\usepackage{fullpage}
\usepackage{times}
\usepackage{html}
\usepackage{epsfig}

\title{DOCUMENT TITLE}
\author{YOUR NAME}
\date{DATE HERE}
\begin{document}
  \maketitle
  \begin{abstract}
    . . . Abstract here . . .
  \end{abstract}
  \tableofcontents
  \listoffigures
  \section{FIRST SECTION}
  . . .
  \section{SECTION SECTION}
  . . .
\end{document}
```

- Without the `fullpage` package, L^AT_EX generates output with fairly large margins, intended for book printing and later trimming.
- The `times` package causes L^AT_EX to use a standard PostScript font, eventually resulting in smaller and higher quality PostScript and PDF.
- The `html` package is part of the L^AT_EX2HTML translator and allows, among other things, hyperlinks in the L^AT_EX source:

```
\htmladdnormallink{text for the link}{http://url/}
```

- The `epsfig` package allows figures from XFig (or any other package which can export EPS) to be included in the following way:

```

\begin{figure}[ht!]
\begin{center}
\caption{CAPTION FOR FIGURE GOES HERE}
\label{optional_label_for_references}
\epsfig{file=YOURFILE.eps}
\end{center}
\end{figure}

```

No matter what, you cannot assume that \LaTeX will place the figure at the exact location where these commands appear. Often, it will create a separate page and put all the figures together. If this is undesirable, you can tune this behavior a little with the following commands:

```

\renewcommand{\topfraction}{0.99}
\renewcommand{\textfraction}{0.1}
\renewcommand{\floatpagefraction}{0.99}

```

The \LaTeX source itself can be edited as raw mark-up in a text editor (similar to editing raw HTML) or a package like LyX can be used for this job. Given the `.eps` Encapsulated PostScript files and the `tex` file with the document, it is then possible to generate all the output formats in one step with a simple Makefile and our friend GNU make:

```

# ----- Makefile for a LaTeX document. -----

DOCUMENT=YOURDOC
FIGURES=FIGURE1.eps FIGURE2.eps FIGURE3.eps
L2HOPTIONS=-antialias -no_antialias_text -split +1 \
-show_section_numbers -long_titles 4

# ----- The rest of the Makefile is generic. -----

all: $(DOCUMENT).ps $(DOCUMENT).pdf $(DOCUMENT)

$(DOCUMENT): $(DOCUMENT).ps .latex2html-init
latex2html $(L2HOPTIONS) $(DOCUMENT)

$(DOCUMENT).pdf: $(DOCUMENT).ps
ps2pdf $< $@

$(DOCUMENT).ps: $(DOCUMENT).tex $(FIGURES)
latex $(DOCUMENT).tex
latex $(DOCUMENT).tex
dvips $(DOCUMENT).dvi -o

```

The Makefile will automatically regenerate all formats (HTML, PS, and PDF) with the single command “make” whenever a figure or the document have been updated. To generate the PDF, it uses `ps2pdf`, and to generate the HTML it uses `\LaTeX 2HTML`. The latter has many features which can be controlled from the \LaTeX source. Check the manual for more information.

11 Software Organization, Versions, and Conventions

Anyone contributing software to the main development tree `/cfht/src/...`, should follow these basic guidelines. You may be able to keep track of what you are doing without following all of this, but in order for everyone in the group to be able to make some basic assumptions about the software in `/cfht/src/`, we all need to comply. If you don't, you could be causing one of your colleagues (and possibly yourself) a huge headache some time in the future. This is an attempt at a practical guide on how to maintain some basic organization in the CFHT source directories. This organization also provides some security for software used in critical functions of observing. Even if your software is not in this category, it has a better chance of being maintained if we follow some consistent methods.

11.1 Location of Files

All CFHT Unix machines now have a `/cfht/` automount point. Subdirectories are as follows:

`/cfht/src/`

This directory is always a mount of `saturn:/usr/local/cfht/dev/`, which contains sub-directories "pegasus" for global session-related software, "medusa" for instrument or sub-system specific software, and "epeus" for third party software used in observing sessions. The Makefile setup will build stuff from `/cfht/src/` even if your true working directory is somewhere in `/usr/local/cfht/dev/`.

`/cfht/man/`

This is always a mount of `saturn:/usr/local/cfht/man/`.

The rest of the directories (`bin`, `lib`, `include`, and `obs`) are different for different machines. Almost all machines at the summit, including the Sparc Engines now have their own, local copies of these directories. Most Solaris 2.6 binaries need to be built on `makani` right now. Other than that, new versions need to be built on every machine that uses them. You can look at `/etc/auto.cfht` on a particular machine to be sure.

`/cfht/bin/`

The file `/cfht/src/local-install.sh` makes a set of symbolic links into `/usr/local/bin` of every machine at CFHT, pointing to a few programs in `/cfht/bin/` that are generally useful. For example, the "clone" command for `director` can be run by any user from any machine (as long as they at least have `/usr/local/bin/` in their `PATH`). `unroll` and `tcshandler` (Oops... this causes it to look for `tcshandler.par`) are a couple of other links created by `local-install.sh` at the moment. `/cfht/bin/` shouldn't need to appear in anyone's `PATH` (if it does, it should be at the end.) Observing accounts have symbolic links in `$HOME/bin` and if you wish to make programs runnable by staff members, add them to the `/cfht/src/local-install.sh` script and re-run it.

`/cfht/lib/`

At the present time, we only use static libraries, so at runtime, nothing uses `/cfht/lib/`. On `saturn` or in `Waimea`, `/cfht/lib/` generally contains the latest static libraries matching what we are working on in the source tree, so if you build and link on `saturn`, that is what your executable will get. At the summit, we only do "make install"s of libraries when they have been tested, so the stuff in `/cfht/lib/` on `neptune` may differ. This means that if you build executables at the summit, they can behave differently than those built from the *same sources* in `Waimea`. Copying a binary built on `saturn` into an account will not be subject to this, but that could change if we ever start using *sharable* libraries in `/cfht/lib/`. (In that case, your executable would load whatever is in `/cfht/lib/` at runtime, so an executable could change its behavior simply by being moved to the summit.)

/cfht/include/

This directory must always be paired with a matching `/cfht/lib/`. It contains include files that are installed at the same time as the matching library in `/cfht/lib/`. With or without sharable libraries, it will never get used at runtime, though.

/cfht/conf/

This contains installed par-files and other text files that *can* get used by programs at runtime.

/cfht/obs/

This is the last of the architecture-specific directories. It contains intermediate object files (.o) and executables and libraries before they are installed. "make clean" removes entire subdirectories from this tree to force recompilation, but normally the files in `/cfht/obs/` are used by "make" to resolve dependencies and allow it to figure out which components have changed and need to be rebuilt and/or reinstalled.

There is another use for the files in `/cfht/obs/`. When programs are installed in `/cfht/bin/`, they are stripped of debugging symbols. As long as the last step you did was a "make install", there will be a matching copy *with* symbols in `/cfht/obs/progname/progname` which can be used by gdb to debug cores or running versions of the installed copy in `/cfht/bin/`. There is more information on how to do this the Makefile section of this document. (*Note that as soon as you change the source and do another "make" without a "make install" the version in /cfht/obs/ will no longer match, and gdb will warn you of this if it happens.*)

11.2 Makefile conventions

Within "pegasus", "medusa", and "epeus" each project or instrument may contain a subdirectory. When possible, creating a separate subdirectory for each program is recommended because it keeps the Makefiles simple. I currently do not have an example of how to generate more than one *compiled* executable from a single subdirectory. If you make one, please put it here as an example. Most Makefiles include "Make.Common" *twice*. So far I only have three examples:

- A project Makefile which doesn't install anything but just lists the subdirectories that "make world" should traverse. It is not enough to simply give the subdirectories in the order that they should be compiled! You must set up proper dependencies using the form in the example below, or "make -j" for multi-processor machines will not work properly. Here is an example:

```
# Makefile for cli-2.8 project tree
include ../Make.Common
SUBDIRS=libcli director clicmd clicap clidup runon
clicmd.d clicap.d clidup.d runon.d director.d: libcli.d.install
include ../Make.Common
```

The dependency line indicates that libcli must be installed before the others can even be built.

- A Makefile for a C program. Programs should always be at the "leaves" of the directory tree, meaning there should be no more subdirectories for "make" if at this level. Example:

```
# Makefile for directest
include ../Make.Common
$(EXECNAME) $(EXECNAME)-pure: $(OBS) -lcli -lcfht
include ../Make.Common
# Rest of Makefile is auto-generated
```

- A Makefile for a C library. These are even simpler, assuming you just want every *.h file installed and every *.c file compiled:

```

# Makefile for libcli.a
include ../Make.Common
include ../Make.Common
# Rest of Makefile is auto-generated

```

If you need to explicitly list your source files, that is possible too. See the link below.

How does the Makefile know whether it has to generate a library (.a) or an executable? The name of the subdirectory always gives the name of the object being generated, and if it starts with "lib*", the Makefile knows it has to make a library. This means you must create a subdirectory for each library and program, and they must have the same name. More specifics on Makefiles are given in the Make.Common section.

11.3 Index files

Placing some kind of README or file-list at each directory node is generally a good idea. If you use the suggested format below, you will be able to take advantage of the "make titles" function of Make.Common, which automatically inserts a description and copyright comment into the top of several types of source files. Here is the suggested format. The file must be called Index if you want "make titles" to find it:

```

# Description:

package DIRECTOR cli wrapper
version 2.8
organization Canada-France-Hawaii Telescope
email daprog@cfht.hawaii.edu

# Contents:

Makefile Use with GNU make to build director

director.cc main event loop for director program

builtins.cc builtin commands that director handles itself

Curs.hh manages screen output and keyboard input
Curs.cc See Curs.hh

Roll.hh holds of the latest N lines echoed in shared memory
Roll.cc See Roll.hh

Pipes.hh reads line-by-line from pipe or fifo sources
Pipes.cc See Pipes.hh

setserial.h Stty type-stuff used by Pipes.cc to set up serial ports
setserial.c See setserial.h

```

To see how "make titles" inserts this information in C code, see the files in /cfht/src/pegasus/cli/director2/ for an example. "make titles" will also search the files listed in Index for unusually long lines and markers that you can leave within comments. If it finds === (three equal signs) it will be flagged as

a warning (parsable by the emacs compile buffer, so you can click with the middle mouse button in the editor and it will automatically pull up the file with the cursor in the correct place) and %%% (three percent signs) will be flagged as an error. Files that exist but are not identified in Index are also flagged. You can run the command "titler" with no arguments to get some built-in help on this utility.

11.4 Changes files

Placing a file called "Changes" in a subdirectory can serve several purposes. RCS (see below) keeps a change log on a file-by-file basis. If it useful to describe a change in terms of what it did to the over-all project, it might help to maintain a Changes file that tracks this development.

Second, if you make all of the lines in Changes into comments, except for the last line, which sets a version number, then you can include Changes in your Makefiles and use it to assign a "1.0, 1.1, 1.2, 2.0 ..." style versioning to your program, rather than the Pegasus default (the date, in a string *-YYMMDD). In this case, it is usually most meaningful to have a Changes file only for the top-level of a project, and then include it in the Makefiles of all the components, so that a set of utilities all have a consistent version number. Using these manual version numbers gives you a little more flexibility, but you must use it responsibly. For example, once version "2.5" is being used, especially in an observing environment, you had better switch to version "2.6" as soon as you start modifying the code again to avoid confusion, no matter how small the changes might be. On the other hand, with the default dated versioning, you do not have the option to move to a new version number until the next day! Here's an example of the Changes from the cli (director) project:

```
# CLI Version History
#
# 1.0 - First version. Used for uh8k run in 97I.
# 2.0 - Shared memory roll buffer. status: changed to statusbar: message
# -----CHANGE IN SHARED MEMORY SEGMENT SIZE (2.0/2.1 can't clone each other).
# 2.1 - More efficient packing of roll messages (Shm size from 1M->160K)
# 2.2 - Silently handle SIGALRM for client "low priority". Used for aobir 97II.
# 2.3 - Allow spaces in comlist for displaying in help. Used for uh8k 97II.
# -----CHANGE IN SHARED MEMORY SEGMENT SIZE; Versions above/below this
# -----line cannot attach to each other's shared memory segments!
# -----Use of <2.4 should be discontinued anyway, as 2.4 is stable and
# -----fully supports all features of previous 2.x version. Extra entries
# -----have been added to shared memory structures to hopefully avoid the
# -----need for further incompatibilities.
# 2.4 - More tolerant of named pipe problems; blank entries in shm for future
# 2.5 - Better debug info for director and clicmd; re-start write()'s to pipes.
# 2.6 - Minor fixes to curses screen update code; detect rmd() errors.
# Entire environment is now passed to agents on remote hosts.
# Clicmd utility now supports sym-linking to command names.
# Clones can only be activated by entering account password first.
# Clones automatically get infosize of parent if no '-i' option given.
# Infolines that run to end of screen don't erase next line anymore.
# Autoprobe for rxvt turns on color support even if TERM variable wrong.
# Added -t TERM and -C (no title clock) command line options.
# cli_system() no longer interruptable by SIGALRM or other signals.
# cli_sh_cmdstr() and cli_remsh() added to libcli.a
# Now requires Posix "termios.h" terminal i/o routines.
# 2.7 - Removed cli_sh_cmdstr and cli_remsh(). Replaced with external "runon".
```

VERSION=2.7

And it is included in the Makefiles below as follows:

```
include ../Make.Common
include ../Changes
.
.
.
include ../Make.Common
```

So that installed binaries will be versioned as foo-2.7 rather than foo-981209.

11.5 RCS Check-in and Check-out

Once a group of files has reached a stable, usable condition, each file should be "checked in" to RCS. After this has been done, you should NEVER move, use root access to change anything, or manually chmod/chown a file, or otherwise try to circumvent RCS! Learn these simple commands and save all of us a lot of confusion in the long run.

You may find it useful to create the following aliases (the commands "co" and "ci" are already aliased in this way if you use **bash** at CFHT.) The examples below assume you have them. If not, be sure to add the options each time you type the command. Note that you can call the alias something other than co and ci if you don't want to clobber the original command.

ci	"ci -v3 -u"
co	"co -v3 -l"

The -v3 means operate in a mode compatible with RCS version 3. (I'm not sure this is needed by new projects anymore, but it just makes things compatible with the co/ci that are still in /usr/bin/ on our HP-UX 9 machines.) The -u/-l mean that you will always be unlocking a file when you check it in, and locking it when you check it out. Locking just means no one else will be able to edit the file while you're working on it.

11.5.1 Initial Check-in

Lets say you've reached a stage with file foo.x that meets one of the following:

- Things are functioning and you may want to return to this version later if things break.
- Somebody else may need to edit your file soon.
- You fear that someone else may edit the file by accident (if you don't check it in, our default umasks allow anyone else in the group to accidentally edit your files. Once entered into the RCS system, this can no longer happen.)
- The file is at a stage where it is actually being used by engineering staff or observers. (*-i If the file/project has reached this stage, you must have checked it in to RCS by now. The only exception to this will be if you are truly the only maintainer and you wish to make an archival backup of the entire project tree instead. You could do both. But if you only make the archive, give it a version*)

number or date that matches the binaries and leave it in the source tree where others can find it.)

First, make sure an RCS subdirectory exists or RCS will make a mess in the current directory. If it doesn't exist, simply run:

```
% mkdir ./RCS
```

Initial check-in for `foo.x` will then look something like this, assuming you set up the alias properly:

```
% ci foo.x
RCS/foo.x,v <-- foo.x
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> This is my description for file foo.x
>> .[Return]
initial revision: 1.1
done
% _
```

11.5.2 Help, the file is now gone!

Well, it shouldn't have disappeared, but if you didn't alias `ci` to be "`ci -u`" then RCS will remove the file from the current directory. To keep things simple, we never want to use RCS in this way, so if this happens, please fix your aliases and then check the file out again immediately. But don't worry, RCS will never remove the `,v` version of the file that lives in the RCS subdirectory.

11.5.3 Check-out

The next thing you'll probably want to do is check the file out immediately again, to continue editing what will become version 1.2 upon the next check-in:

```
% co foo.x
RCS/foo.x,v --> foo.x
revision 1.1 (locked)
done
% _
```

The "(locked)" is important, and shows that you have exclusive control over editing the file now. In case someone else already grabbed the lock before you, a message like this may show up:

```
% co foo.x
RCS/foo.x,v --> foo.x
co: RCS/foo.x,v: Revision 1.1 is already locked by bonehead.
% _
```

If you see this, try to contact bonehead and get them to check the file back in, using the procedure below. Another thing you might see is this:

```

% co foo.x
RCS/foo.x,v --> foo.x
revision 1.1 (locked)
writable foo.x exists; remove it? [ny](n):
% _

```

Say 'n'! This appears if you already have the file checked out. If you check it out again, you will get back to the old version. The only time you would want to do this is if you want to cancel all of your edits since the last check-in. In this case, I strongly recommend you first run `rcsdiff` on the file to see what changes you are about to lose in reverting to the previous version. (See the `rcsdiff` example below.)

11.5.4 Check-in

If you haven't made any changes, but you want to unlock the file again so someone else can check it out, just run `ci` and it will be smart enough to figure out that there's no need to make a "version 1.2" and it will just return the file to its safe, checked-in state:

```

% ci foo.x
RCS/foo.x,v <-- foo.x
file is unchanged; reverting to previous revision 1.1
done
% _

```

On the other hand, if you've made changes to the file, I strongly recommend running "`rcsdiff -c`" on the file first to double-check that you've only changed the parts of the file you meant to change. `rcsdiff` will show you what's different between the last checked-in version and the version you are about to check in. The `-c` option selects "context-style" output, which is shown below:

```

% rcsdiff -c foo.x | less
=====
RCS file: RCS/foo.x,v
retrieving revision 1.1
diff -c -r1.1 foo.x
*** /tmp/T0a19438      Wed Dec  2 23:29:38 1998
--- foo.x             Wed Dec  2 23:27:55 1998
*****
*** 1 ****
--- 1,2 ----
    This is a dummy file, dummy.
+ Here's a new line I added to the end for version 1.2.
% _

```

New lines are marked with a "+", removed ones with a "-" and changed ones will show up twice, marked with "!" (first the old one, then the new one.) If this all looks good, then check in the file, and summarize the changes shown by `rcsdiff` by typing in a line or two describing what's new.

```

% ci foo.x
RCS/foo.x,v <-- foo.x
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:

```

```
>> Added a line to the file for demo purposes
>> .[Return]
done
% _
```

If you ever receive anything different than what you see in the examples here, PLEASE do not try to force things to work. Get help and clean up the mess immediately, before things get out of hand.

One final note on check-in: To keep things simple, I recommend avoiding *branching* feature of RCS versioning.

11.5.5 Rlog and RCS keywords

You should insert a `Log` token in a comment near the top of your file, so RCS will maintain a log message history below it. This can quickly lead to files with several pages of log messages and a tiny bit of code at the bottom, so (if you are certain that none of the log messages were manually edited) you can trim this log and place a comment at the bottom showing how to get the full log:

```
* $Id: cfp_file.c,v 2.13 1999/08/23 22:09:58 thomas Exp $
* $Log: cfp_file.c,v $
* Revision 2.13 1999/08/23 22:09:58 thomas
* Moved unlink's from DEBUG to DEBUGCPP
*
* Revision 2.12 1999/08/05 01:06:28 thomas
* Fix strcat -> strcpy in one path of interactive copy
*
* . . . RCS Log truncated. Use "rlog" command on this file for complete log.
*/
```

Assuming everyone faithfully put useful messages in the log *at check-in time*, you can always access the log information separately using the `rlog` command:

```
% rlog foo.x | less
RCS file: RCS/foo.x,v
Working file: foo.x
head: 1.2
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 2;      selected revisions: 2
description:
This is my description for file foo.x
-----
revision 1.2
date: 1998/12/02 23:32:00; author: isani; state: Exp; lines: +1 -0
Added a line to the file for demo purposes
-----
revision 1.1
date: 1998/12/02 23:17:40; author: isani; state: Exp;
```

```
Initial revision
```

```
=====
% _
```

Verify this first before trimming a log in the source!

Above the `Log`, insert `Id` on a comment line. RCS will change this key into a useful summary of information about the file, and will automatically keep it current each time you `co` and `ci` the file. In C code, you should also use the `RCSID()` macro defined in `<cfht/cfht.h>` so that compiled `.o` files contain a string indicating from which version of the C file they were generated. In any case, try to include `Id` in at least some kind of comment near the top of the file so that it is obvious to anyone editing that the file is under RCS. Here's a short C file that shows what `Id`'s look like when you insert them:

```
/*
 * Copyright, blah blah blah blah
 * $Id$
 * $Log$
 * . . .
 */

#include <cfht/cfht.h>

RCSID("$Id");

/* ...code starts here */
```

And here is how they look after RCS has replaced them:

```
/*
 * Copyright, blah blah blah blah
 * $Id: test.c,v 1.1 98/12/03 17:36:24 isani Locked $
 * . . .
 */

#include <cfht/cfht.h>

RCSID("$Id: test.c,v 1.1 98/12/03 17:36:24 isani Locked $");

/* ...code starts here */
```

11.5.6 RCS and Emacs

If you use Emacs or XEmacs, you should be able to use the key-strokes Control-X-v-v to lock and unlock a file instead of running `co` and `ci` from the command line. This way you probably won't get the `-V3` option, but I don't think it is very important except for things like consistent log formatting in older files. `-V3` also affects the formatting of the information in a `$Header$` keyword. Files that have `$Header$` should really have `Id` instead, and there should be no problem changing them to `Id`'s in which case the `-V3` option makes no real difference either way.

11.5.7 Summary of Check-in/Check-out

- Don't use branching version numbers.
- Don't check in files without -u or they'll "disappear".
- Use `Id` inside a comment somewhere if possible. (Please convert old `$Header$`'s to `Id`'s.)
- Use `Log` if appropriate (otherwise: `rlog` will always work)
- The commands you need to know are:
 - `co foo.x` (assuming `co` is aliased)
 - `ci foo.x` (assuming `ci` is aliased)
 - `rlog foo.x`
 - `rcsdiff -c foo.x`
- Never circumvent RCS to "save time"!

11.6 Releases

Pegasus only has ways of making an entire release of all the software in the development tree, and making a binary release of everything used by a specific observing account. These are covered in another document. Releases or backups of source trees can be made on a smaller scale using the more manual methods below.

11.7 Archival Back-ups

Since we currently have no release system that can be applied per project in the Pegasus source trees, and since RCS only works on the level of one file at a time, here are some ways to make backups of an entire source tree manually. Whether you use the advice given here or not, *please be sure that the commands you use preserve ownership, permissions, file time stamps, and file types!* One easy way to do this is to use the GNU version of `cp`, which is installed as "`gcp`" site-wide. When used with the `-a` option ("archive"), it makes an exact backup of the original file. The `-v` option ("verbose") is also recommended. For example, you can use it to make a backup before editing a text file:

```
% ls -latr testfile.par*
... 6239 Jul 21 02:02 testfile.par
% gcp -av testfile.par testfile.par.LAST
% edit testfile.par
% ls -latr testfile.par*
... 6239 Jul 21 02:02 testfile.par.LAST
... 6239 Dec  9 12:31 testfile.par
% _
```

The `-latr` options for `ls` cause it to list all files in long format, sorted with the most recently touched files at the bottom of the listing. You can see the importance of always copying with "`gcp -a`", so that the information from "`ls -latr`" makes sense. You can use timestamps in combination with the "`find`" command as well. Say you were about to start hacking on files, and you want to be able to generate a list of the files you messed with after you're done. "`ls -latr`" will quickly show you files at the current directory level, but if the files are scattered throughout a tree, "`find`" is more appropriate. If you create a marker file *before you start editing*, or use "`touch`" to create it with the appropriate time-stamp after, you can use the `-newer` option to locate all the files that have been modified:

```

% touch MARKER
... Edit a bunch of files from the current directory down ...
% gfind -newer MARKER -print
... Lists all the files you edited (all the files modified after MARKER
    was created ...

```

You might also be able to use a recent .tgz file as the marker. Just remember that gfind will list all files with a modification date later than that of the marker file.

GNU copy can also be used to make copies of entire subdirectories, keeping timestamps, symbolic links, and permissions in tact.

```

% cd /cfht/src/medusa
% gcp -av myproject myproject-snapshot-981111
myproject -> myproject-snapshot-981111
myproject/Makefile -> myproject-snapshot-981111/Makefile
myproject/foo.c -> myproject-snapshot-981111/foo.c
.
.
.
% _

```

However, if your Makefile is set up correctly, there is a better way to make a backup of a project. Let's say your project subdirectory is called "fuzzbuster/", then /cfht/src/medusa/fuzzbuster/ (physically on machine saturn) should *always* be the latest copy of your project files. This should also be the only place where the RCS/ subdirectories appear, otherwise you could end up with two different copies of the same version of the same file. To make a snap-shot of the entire tree, not including the RCS directories, you can use the following target provided by Make.Common (see Makefile conventions to make sure your Makefile properly includes Make.Common for this to work.)

```

% cd /cfht/src/medusa/fuzzbuster/
% make tar
% cd ..
% ls -l fuzzbuster-VERSION.tgz

```

Once you've created a versioned archive, don't make any further changes under that version number! Also, be sure to save any buffers in your editor before making the tar file.

Be careful when extracting the .tgz file, because if you do it in /cfht/src/medusa/, it could overwrite a newer version of itself. One way to avoid this is to temporarily rename the project directory to project-VERSION before running "make tar". You can even leave the directory name with a version number in it (Make.Common will automatically strip off the -VERSION part, if it exists) and make a symlink pointing to it without the version. In any case, before *extracting* a file, you should always look at the contents first, and if necessary, create a subdirectory, cd to it, and untar there. To look at the contents, use the following:

```

% gtar tzvf fuzzbuster-VERSION.tgz | less

```

To check if the tar file matches what is on disk you can use:

```

% gtar dvf fuzzbuster-VERSION.tgz | less

```

To do a full context-style comparison between two versions, extract them both in a scratch space and use `gdiff`:

```
% mkdir TMP
% cd TMP
% gtar xzvf ../fuzzbuster-VERSION-A.tgz
% mv fuzzbuster fuzzbuster-VERSION-A
% gtar xzvf ../fuzzbuster-VERSION-B.tgz
% gdiff -rc fuzzbuster-VERSION-A fuzzbuster | less
```

Or to compare the file with the version on disk:

```
% gdiff -rc fuzzbuster-VERSION-A ../fuzzbuster | less
```

Aside: The output of "`gdiff -rc`" is human-readable, but is also suitable for input to the "`patch`" program for making incremental upgrades.

If a certain version of a program is in use by observing accounts, but source in `/cfht/src/` doesn't match it, because it is under development, it is advisable to keep a `.tgz` file at the same level as the project directory with a clearly marked version. For example, if we are running `rpm-2.0.4` as the Web server at the summit, but the version in the pegasus source tree is `rpm-2.0.5`, then it is useful to be able to quickly access the source for 2.0.4 if there is a problem with the version in use. If some bug is showing up in 2.0.4 that has been fixed in 2.0.5, it will be very confusing looking at the source code for 2.0.5! For this reason, `/cfht/src/pegasus/` should contain at least the `.tgz` file `rpm-2.0.4.tgz` and any other versions that are in use by anything critical. Older `.tgz` files can be archived on tape or CDROM and removed from the pegasus source tree.

11.7.1 Summary of Archiving/File maintenance commands

ls -latr

Lists all files with the most recent ones at the bottom.

gfind -newer *somefile*

Lists files modified after *somefile*.

gcp -av *original backup*

Creates exact backup copies of files or directories.

gdiff -rc *first second*

Generates context diff output of files or directories.

make tar

Creates a compressed tar image of a project, excluding RCS. Use this just before upgrade VERSION number.

gtar czvf FILE.tgz FILES

Creates compressed tar file. (Use "make tar" on source directories.)

gtar dzf FILE.tgz

Lists differences between files on disk and tar file.

gtar tzvf FILE.tgz

Lists everything in the tar file.

tar xzvf FILE.tgz

Extracts the files into the current directory, overwriting anything in the way.

12 The Observing “Sessions”

12.1 History

At CFHT, we refer to the user account for an instrument and all of the software installed within it as the “session”. Sessions which conform to the original design, which includes automatic clean-up, re-build, and release mechanisms are known as “Pegasus sessions.” Practically every session has enough manual edits and tweaks that the only way to restore the setup is from a simple archival backup. Newer sessions like “megacam”, which build on Pegasus software have completely abandoned the auto-build design, are known as “NEO” (New Environment for Observing) sessions. These also lack any “Pegasus Session Manager” (**psm**) tool bar at the top of the screen, but just launch a detector control window and a top-level “RPM” GUI at login time.

If the PSM is missing, “background menus” controlled by the Window Manager (should?) give at least these three options as a partial substitute for the missing PSM:

- Re-start top-level GUI (which itself is kind of similar to most of the PSM buttons, except Web-accessible if RPM is used.)
- Re-start the “director” text window if it dies.
- Log off (and shut down instrument, if prescribed in .fvwm-exitfunction)

12.2 Template Session

Just as the username “**pegasus**” was a non-instrument-specific demo/testing session, the username “**neo**” (with home directory `/h/neo`) serves as a template for what we might want future sessions to look like. At the moment, it is unlikely that existing instrument sessions would work, though, if one just forced all the generic components of `/h/neo` into a specific session. Unfortunately the instrument-specific needs are not as well separated as they should be. It is hoped that **neo** might be useful in the following ways:

1. Testbed
2. Template for starting a new session (wircam, espadons, etc.)
3. To diff against, to more easily highlight what the modifications in other sessions are, versus this “baseline.”

The following subsections describe the generic NEO account, and also how to convert a Pegasus account to use as much of the NEO things (FVWM, Director, Detcom) as possible.

12.3 Components of a NEO session

This is a description of all of the files and programs that make up a CFHT observing “session” such as megacam, cfhtir, and instruments using DetCom as the source for FITS files. It is written from the point of view of setting up a new account from scratch.

12.3.1 Home directory and login (NIS)

The first step to create a data acquisition account (or any other user account, in fact) is to add an entry to the master NIS password database, currently on the machine “niu” in Waimea. **This step is usually performed by the system administrator. Skip to step 2 if an account has already been created.**

A typical observing account entry in `/var/yp/src/passwd` would look like this:

```
neo:xxxxxxxxxxxxx:1234:1000:New Environment for Observing:/h/neo:/apps/gnu/bin/bash
```

Notes:

- For the second field, copy the encrypted password string from another active observing account.
- For the third field, be sure to pick a new, UNUSED number and place the entry in order, SORTED by user ids.
- For the last field, use `/apps/gnu/bin/bash`. “Bash” is the only shell which has a consistently configured version across all platforms at CFHT, and “`/apps/gnu/bin/`” is the path to use, even though some machines also have bash in `/bin/`.

The home directory should be located on a reliable RAID disk at the summit, and the entry in the autofs maps is maintained in the file `/var/yp/src/autofs/SUNautomount/auto.h` also on niu.

```
neo          noeau:/local/h/neo
```

would be the entry matching the passwd example above.

Finally, add the new user account to netgroups “doubt” and “observer” and push the changes out to all machines using the normal procedures.

12.3.2 Obsolete account files

If migrating an old account, all of the following files are obsolete, and at least some of the *must* be removed to avoid problems:

```
xllstart.cfht - done by system wide startup scripts
.xllstart*   - done by .fvwm-xstartup
.xprofile*   - done by .fvwm-xstartup
.xsession    - done by system wide fvwm.xsession
.Xdefaults   - system wide Xdefaults should contain everything necessary.
session      - now handled by system startup scripts, director, etc.
  TODO: Nothing handles make -C /cfht/src/observer config yet.
  TODO: Nothing handles checking that the session is not already in use.
  TODO: Nothing starts vstatus.
.sessionrc   - done by system wide fvwm scripts and .fvwm-initfunction
  TODO: Nothing checks for .disable
```

```

.cshrc      - system wide bash startup scripts take care of it
.login      - system wide bash startup scripts take care of it
.logout     - *If* you have one, see if there's anything in there
             that should be moved to sessionquit.
.mwmrc      - Ignored by fvwm
.twmrc      - Ignored by fvwm
.gnome      - Remove, xdm on Linux doesn't need it
.vue        - Remove, if prompted for window manager on HP, select FVWM.
.vueprofile - ``
.dt         - Remove, if prompted for window manager on Sun, select FVWM.
.dtprofile  - ``
.rhosts     - Remove for security
.ssh        - (directory) Remove for security
.colorsetup - Not used when feedback window is Director.
.,*.header  - Not used by DetCom (will be re-created if still using DetI).
core*       - No comment.
.,aloha.par - Aloha is gone :-()
.,genh.par  - Not needed.

```

Still more files, which are all vestiges of pre-DetCom detector control systems, should be removed to avoid the possibility of scripts obtaining incorrect pixel size or other chip information:

```

.,ccd.par
.,ccd.par
.,chip.par
.,detector.par
.,*.cfg
.,howccd.par
.,whichccd.par
.,template
.,raster.par
.,fraster.par

```

TODO: Make a kill-script which takes care of all this (making backups and verbose commentary along the way.)

12.3.3 Dot Files

Once an empty home directory exists, some dot-files must be set up to configure the account as an observing session. In addition to the files shown here, other dot-files may appear as a result of logging in and using various programs, but those listed here are believed to be the only ones necessary for a functioning session.

```
.fvwm-errors-HOSTNAME
```

This file will be *created*. All output from the rest of the setup scripts ends up here. HOSTNAME is usually the session host, except for very early messages from the login process, when it might be the name of the display host instead. Hint: "ls -latr" will list the most recently touched file(s) at the end of its output.

```
.dt/sessions/lastsession
```

This is only for HPs (and Suns?) to tell their greeter program not to use CDE. The file should contain a single line with:

```
/apps/wm/common/fvwm.xsession
```

which is the name of the script that sets things up and eventually executes fvwm.

`.fvwm-options`

This is a shell script (or any other type of executable) which can echo bash-syntax commands on its stdout. This provides a mechanism to set option variables or override defaults in the FVWM startup sequence. The following is an example of a complete `.fvwm-options` for a session that should run on noeau (regardless of where the user logs in)

```
#!/bin/sh
#
# .fvwm-options - Set some variables for /apps/wm/common/fvwm.xsession
#                 that select options for a Pegasus observing session.
#
SSO_HOST=noeau
SSO_WMNAME=fvwm2
SSO_PEGASUS=1
SSO_COLORS=pegasus
#
# send the variables to the fvwm startup script (calls this script w/ eval)
#
echo "SSO_HOST=$SSO_HOST      ; export SSO_HOST"
echo "SSO_WMNAME=$SSO_WMNAME  ; export SSO_WMNAME"
echo "SSO_PEGASUS=$SSO_PEGASUS ; export SSO_PEGASUS"
echo "SSO_COLORS=$SSO_COLORS   ; export SSO_COLORS"
```

The window manager for observing sessions is FVWM version 2, and the `SSO_PEGASUS=1` setting causes CFHT Pegasus-specific root-menu options to appear. These root-menus are defined in the standard distribution in `/apps/wm/`, but are disabled for normal users.

Another use for the `.fvwm-options` script is to display a pre-login user interface. This could be used to get something like a user preference for default shell in terminal windows. Add this to the end of a session's `.fvwm-options` file use the Tcl/Tk script "sso" to gather preferences from the user:

```
if [ "$HOSTNAME" = "$SSO_HOST" ]; then
#
# If we are already on the real sessionhost, display
# the GUI that lets the user choose a few extra options
# related to the session before the window manager starts:
#
exec /apps/wm/common/sso
fi
```

IMPORTANT: Be sure to run "chmod +x" on the script to make it executable.

TODO: Have `fvwm.xsession` export `SSO_*` so `.fvwm-options` does not have to.

TODO: Is `/apps/wm/common/sso` useful to have in by default?

TODO: Set `SSO_HOST` to a DNS CNAME like `$USER-session.cfht.hawaii.edu`?

TODO: Make `SSO_COLORS=pegasus` the default if `SSO_PEGASUS=1`?

`.fvwm-xstartup`

This script is source'd, so it is not required to be executable. It sets a few environment variables which all CFHT observing sessions may want. These include:

- AOBStatus=On/Off - obtained from aobconfigure.par
- CFHTHOME - What's this for? /usr/local/cfht/dev/medusa/aobonnt/laserdot?
- CFHTSCREENCOUNT=`xscreecount` - C program that counts number of screens
- MainDisplay, StatusDisplay, OtherDisplay, cfhtimageDISPLAY - In a single screen configuration, these are all set to hostname:0. When two screens are present, :0.1 is assigned to StatusDisplay and OtherDisplay. When 3 or 4 screens are present, :0.2 is assigned to OtherDisplay (and StatusDisplay is still :0.1). cfhtimageDISPLAY is the same as OtherDisplay.

The job of this script was previously done by ".xprofile" and ".x11start", but they are no longer used, and should be removed to avoid confusion.

CFH12K and MegaCam did not use this file, but nothing is broken by having the extra environment variables defined. So, again, this is a non-session-specific setup script.

TODO: Incorporate this script into /apps/wm/common/ and have it get executed if SSO_PEGASUS is set. Eliminate it from all home directories.

TODO: raster mess: make note that must copy raster-shift script to .director/bin until solution is found (see /h/moseev/.director/bin/raster) Also copy fraster for now? Latest hack: remove raster+fraster from PSM, and disable all rasters which originated from that interface (see /h/osis/.director/bin/raster). User types raster at the command line (or eventually, from the Expose form) and it sticks. Focus scripts that want to send a raster should be modified to send it in the DetCom style (without the extra "slot" argument that DetI takes) so that the new raster script will not ignore it. A prototype of the new raster.rpm form is in /h/osis/rpm/.

TODO: nex - how to handle this? For now, need nex and neo-par from gecko/.director/bin/

.fvwm-initfunction - Like .fvwm-options, this should also be an executable (usually another shell script) and it is used to launch the session manager tool bar (PSM), director feedback window, and any initial sessionstart processes. The same script can be used by all sessions, since "sessionstart" itself figures out what needs to be started for this specific session, and ".director/startup" determines which agent programs need to be started. See also the descriptions of those two files, below.

This script runs once on each screen.

```
#!/bin/sh
#
# .fvwm-initfunction - Start session user interfaces.
#
# Director is started on StatusDisplay in a color rxvt terminal window.
#
if [ "$DISPLAY" = "$StatusDisplay" ]; then
    rxvt -pixmap none +sb -geometry 80x37-62-1 -e director &
fi
#
# All screens should wait for Director on StatusDisplay to be ready,
# so that logging facilities are completely ready.
#
echo "Waiting for director to be ready for log input"
if ( clicmd @running 60 ); then
    echo "Director is ready"
else
    echo "Director still not ready after 60 seconds!"
    # Abort the session somehow?
fi
#
# The main GUI (PSM and/or an exposure form) are started on MainDisplay.
#
```

```

if [ "$DISPLAY" = "$MainDisplay" ]; then
    sessionstart &
    psm -X
fi

```

TODO: MegaCam wants "director -s"!

TODO: Replace sessionstart, psm -X, hform startup with: sessionstart -gui?

TODO: Replace director invocation with start-cli script. sessionstart -cli?

TODO: Rip out NUMBER_OF_SCREEN and *Display stuff from megacam once .fvwm-xstartup is being used by megacam session.

TODO: Fix handler_status problem with geckoh, and remove hack from .fvwm-initfunction

TODO: Add timestamps to junk in .fvwm-errors-*

TODO: Confirm that there is no reason to (re)start roll during session logins. (OK)

TODO: Figure out best place to ensure \$HOME and \$HOME/bin are in the path.

TODO: Other stuff to be sourced from bashrc?

.fvwm-exitfunction

This is executed whenever anyone logs out of an FVWM session. It must run sessionquit (and sessionquit, like sessionstart, must decide what this specific session needs based on .,config). It also must decide whether or not to "shutdown" director. If the instrument is still on, director is left running so that it can be cloned, even if no X-Sessions are logged in. This also allows the alert notification mechanism (which delivers e-mail if a critical condition occurs with the instrument) to continue to operate even when the session is not "logged in." Only megacam uses this at the moment. All other sessions want to shutdown on logout.

TODO: Implement check for whether or not to do shutdown.

TODO: director-4.27+ is required for shutdown feature/"-s" option to work.

More general information on how .fvwm* files are handled in the CFHT environment are given in /apps/wm/common/fvwmstart.ps

The only thing the window manager startup scripts do, for getting the command line part of the session going, is to start "**director**" without any options. If one wants to have just the command line part of a session, it is usually possible to "su - username" and then just run "**director**" as the session.

It is also possible to run the entire graphical session inside a window (vncviewer) using the script "**session-test**", run from your personal account.

.director/startup

However director is invoked, as long as the -p and -S options were not used, director will run any and all commands it finds in .director/startup when it starts as a master. (If you start more director processes by logging in, if the session allows it, or by su'ing and typing director, you will get a clone window, and the startup script is NOT re-run.)

For now, the contents of the .director/startup file specify

- A detector host.
- The version of detcom to be used for this session.
- The location of DSP code lod files to be used by detcom.

Typical syntax looks like this:

```

@-infosize 9
agent start -R -D -H deticli dethost1:/cfht/bin/detcom-3.47 ~/dsp/lub.lod ~/dsp/lub.lod

```

The '@' and '-' characters do the same thing they do in a Makefile. (Don't echo the command itself, and don't stop executing if the command happens to fail.)

The "infosize 9" maximizes the number of debug/user info lines shown below the prompt area. 9 is probably only needed for engineering, so reduce it if you'd rather not confuse the user with DSP code version status, etc. Generally, since we're doing service observing, some of the low level info is good or at least harmless.

The -R and -D options tell director to (R)estart detcom automatically if it exits (or crashes) and that it is the (D)efault agent to receive any user commands that are not prefixed by a handle.

.,deti.par - Needs to be a link to .,detcom.par until all refs to deti.par (possibly just ccd?) are fixed.

TODO: Remove "-H deticli" when all scripts are updated.

TODO: Remove .,deti.par and use .,detcom.par.

TODO: Standardize a way to select detector host on the fly.

TODO: Add option to director that causes it to ignore .director/sessionhost, and remove comments below about it, and sessionhost.tmp.

.director/sessionhost

This is a "bogus" symbolic link. The contents are the hostname where a clone should go to find director's shared memory. In the way we use it at CFHT, the sessionhost link is removed by the main director process when it exits, so that SSO_HOST in .fvwm-options is the single place to change the session host. If .director/sessionhost exists *without* the presence of .director/sessionhost.tmp, it indicates that the link is not going to be cleaned up automatically by director when it exits. This is not the intended use at CFHT, so if you find "sessionhost" hanging around by itself, remove it manually, or director will be confused if you ever change the account's sessionhost.

\$HOME/bin/go

\$HOME/bin/ccd

\$HOME/bin/ccdh

\$HOME/bin/...

Symbolic links to "clicmd", which is in turn a link to /cfht/bin/clicmd. The actual version of these commands are all in .director/bin/ instead of the HOME/bin/ directory. The placeholders in HOME/bin/ only serve to cause these commands to execute within the context of director, even if they are typed in a shell outside of director.

- The commands will be forced to execute synchronously with other commands in director.
- CFHT log feedback and all output from stdout/stderr will appear in the director window, for the benefit of all "clone" users to see what is going on.

\$HOME/bin/*h

handlers, symbolic links to /cfht/bin/ versions of the instrument handlers needed for this session. All handlers that are called in a -B (BeginFits/pre-exposure) configuration must have been upgraded to use the status server FITS staging area.

The first step in this upgrade is to convert the handler to use libfh. Detailed directions for converting from libff to libfh are found on another page.

The code that makes it possible to dump keywords to the status server need to be incorporated into an fh- ζ ss type library. For now, just cut-and-paste the lines from tcsh or geckoh. Those handlers are also good to look at as examples. NOTE: converted handlers are still 100templates, DetI, and G3v3 detector software.

TODO: Add conversion recipes: FF_OBSTYPE - ζ ENV_OBSTYPE, FF_SEQ_NUM - ζ ENV_SEQNUM, FF_ETEMPLATE - ζ ENV_FFTEMPLATE

TODO: Incorporate fh-_zss routines into a library.

TODO: Other programs needed in \$HOME/bin: clicmd clicap clidup director3-3.33

TODO: Session runid and piname needs to be ignored!

.,config

This file needs a fair amount of hacking if you are starting from an old one that was for DetI, or even more so if the session was pre-DetI.

- xtemp should be removed from autostart
- ccd should be removed from sessionservers (if present)
- Expose button replaced with hform DetCom expose.
- Raster/Fraster - incorporated into new Expose form?
- Detector - remove
- News - remove, it's old.
- Aloha - remove, and use window manager exit instead.

.,Makefile.ccd

Based on handler information found in the session .,config, a set of Makefile rules must be generated that specify which things to run before and after an exposure. By specifying exactly what depends on what, it is possible to allow as many exposure steps to run in parallel as possible. The latest .,Makefile.ccd for each account should be kept in /cfht/src/medusa/detcom/ccd-compat/, but check the session directory for last minute changes. For a new session, one of these must be generated by hand, which is not complex.

.director/bin/ccd

The CCD "momma" compatibility wrapper that causes all Pegasus sessions to call appropriate DetCom commands when an exposure is required. Not for MegaCam and CFH12K. The latest ccd hacks are found in the source tree, in scripts subdirectory of ccd-compat. Again, be sure to check the most recently used observing sessions too.

.director/bin/ccdh

A link to ccd, which replaces both the old ccd momma *and* ccdh.

.director/bin/go

Just a wrapper for "ccd-go", except MegaCam and CFH12K, where it replaces ccd and does the initial exposure sequencing itself. The go script contains this:

```
#!/usr/bin/sh
if (( $# > 0 )) ; then
    iterations=$1
else
    iterations=1
fi
exec ccd -go -iterations ${iterations#0}
```

As always, it must be "chmod +x".

TODO: Move "go" into /cfht/bin/ and consolidate all session items that are just symlinks.

TODO: Remove INSTRUME from DetCom. Add to appropriate instrument handlers.

TODO: Sort out raster mess.

TODO: Why does ccd call HOME/bin/raster?

TODO: Install new version of ccd, now in gecko/.director/bin/

12.3.4 GUI Form(s)

The “Expose” button on the session manager runs “ccd -X” which now brings up hform, pointing at /h/USER/rpm/index.rpm. That file should be a link to ”detcom.rpm”, which in turn can be copied from gecko (until a better place is found for these files.)

TODO: Make installation of RPM forms cleaner.

In order to function, the detcom.rpm form must have access to rasters and current detcom status. Set up the following files like gecko:

```
.,neo.par
rpm/neo.par -> ../.,neo.par
rpm/detcom.par -> ../.,detcom.par
rpm/rasters.par
rpm/rasters.rpm ??
```

TODO: Fix rasters (already mentioned elsewhere too)

TODO: Move this stuff to the status server.

12.4 Misc

At CFHT, our Solaris and HP-UX platforms include CFHT-maintained system-wide directories /apps/wm/ and /apps/gnu/. All of the source code and ”configure” options that can be used to build the software in these directories on another Unix-like platform are referenced in the file /apps/gnu/src/CFHT.README. The main areas of customization are in the Window manager configuration (FVWM 1.24r, and FVWM 2) and some of these customizations are CFHT- or even “Pegasus”-specific.

NOTES:

- statmon is obsolete
- temperature monitor is obsolete
- clock is obsolete

Q+A:

- Standard way to select detector host?
- Standard way to select session host?
- Where to add auto-start GUI windows?
- Where to add auto-start setup scripts?
- Where to add auto-start daemons and agents?

13 Conclusion

14 Reference

14.1 Environment

14.2 File Locations

`/etc/syslog.conf`

Configuration file. Controls where the Linux syslogd writes log messages. Contents set through the “reconfig” utility under Sidious Linux.

`/var/log/messages`

Text file. New messages from all system processes are appended to this by the syslogd.

`/var/log/warnings`

Text file. Subset of the above (only the critical messages.)

`/var/log/*`

Text files. Additional log output which should probably be migrated to syslog some day. from the X-Windows system, firewall scripts, USB plug

14.3 Manuals