CFHT Data Acquisition

2016 September 30th

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 (last updated 2016), and our best prediction for the near future.

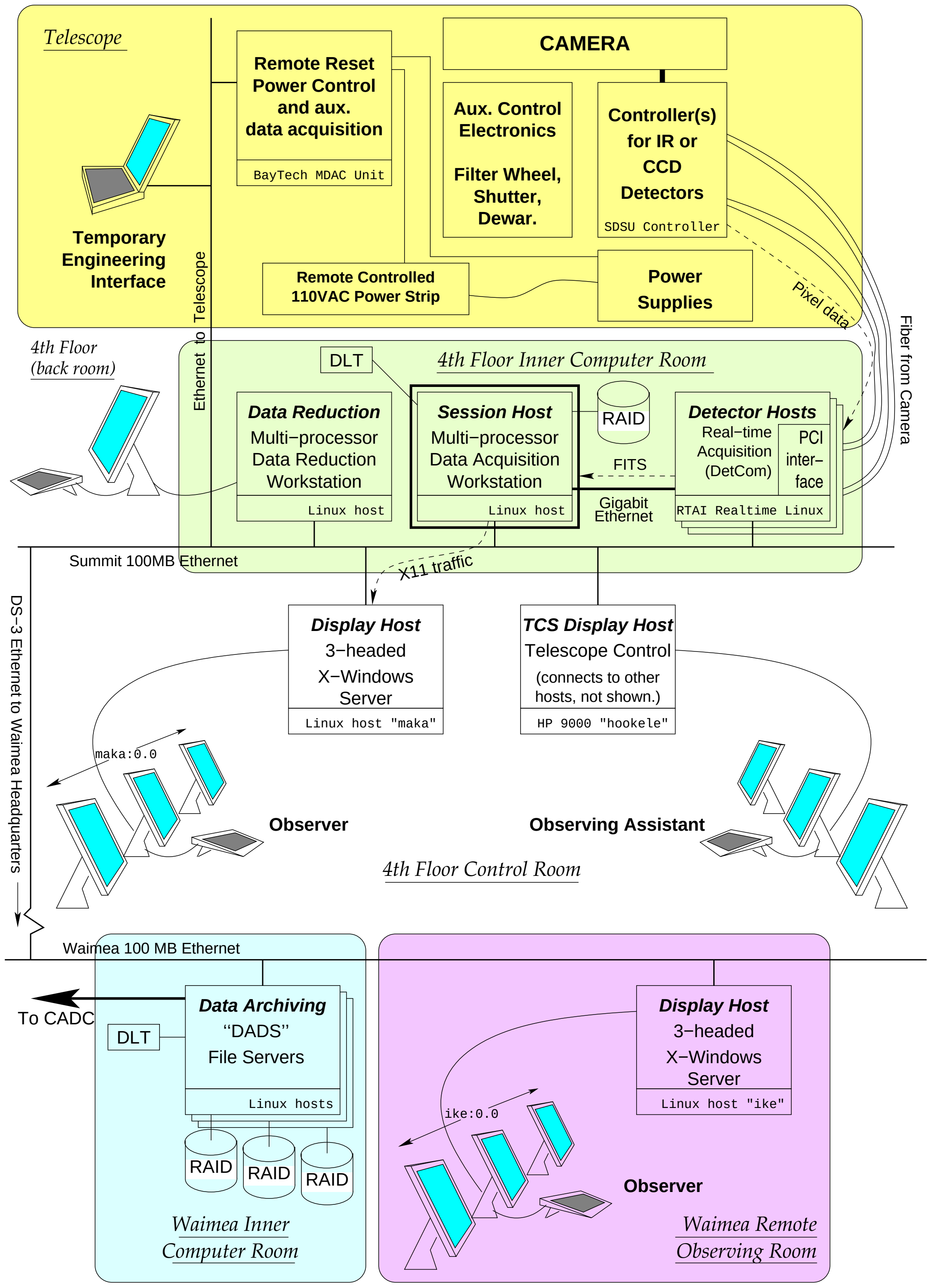
While other software systems at CFHT are at least partly based on the architecture set forth here, there are many which are not described in this document. They include the Telescope Control System, the Elixir project which provides sophisticated data analysis, the Queue Scheduling project which plays 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

List of Figures

1 Computing Network Diagram

Figure 1: Computing Network



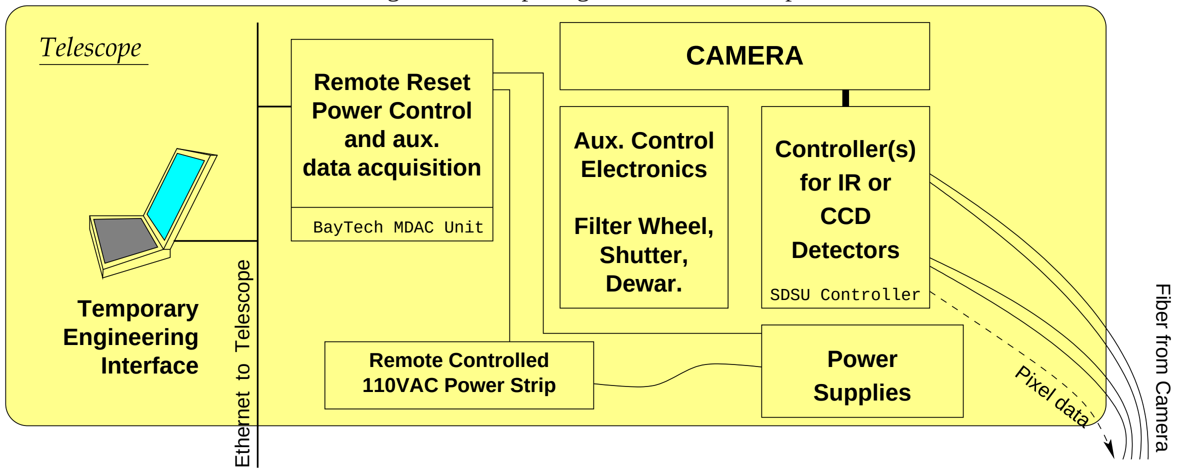
2 Computing Network in Detail

There are three main locations where computing devices are installed. All are interconnected by various speeds of Ethernet (mostly Gigabit, with a few hosts and switch uplinks at 10 GB) as of 2016.

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


Telescope
Temporary Engineering Interface
Ethernet to Telescope
Remote Reset Power Control and aux. data acquisition
BayTech MDAC Unit
CAMERA
Aux. Control Electronics
Filter Wheel, Shutter, Dewar.
Controller(s) for IR or CCD Detectors
SDSU Controller
Remote Controlled 110VAC Power Strip
Power Supplies
Pixel data
Fiber from Camera

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” or “RPC” units. 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 generation II or III, 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).

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.]

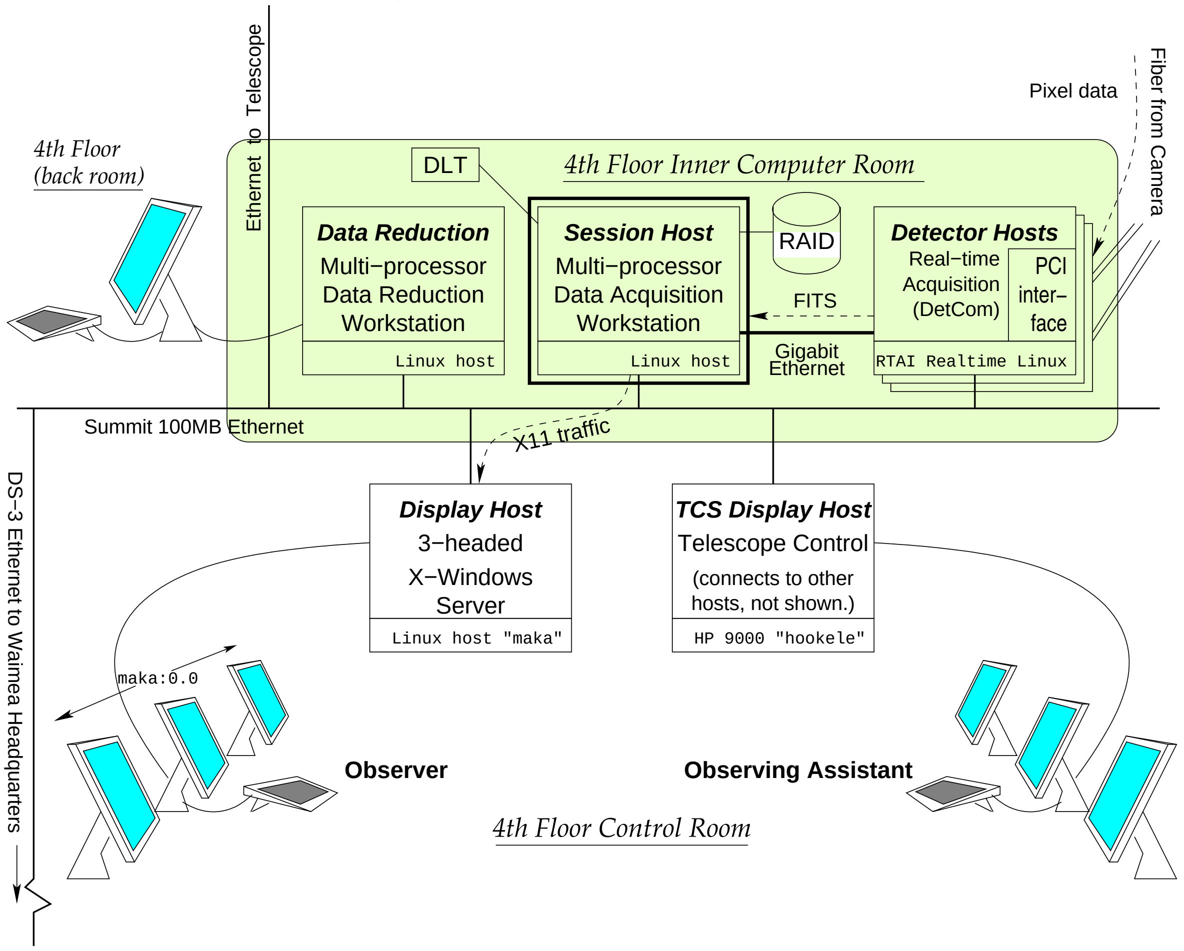
2.2.2 Session Host

The Session Host runs all other acquisition software and user interface components. Images are transferred across either a 1 GBit switch. Since the roll-out of FreeNAS Network Attached Storage devices at CFHT, the Session Host no longer deals with storing the bulk image data. These are written to the NAS RAID volume directly from the Detector Host, over NFS (Network Filesystem protocol.)

Instrument control and high level **sequencing** is 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.

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

Figure 3: Computing Network: 4th Floor



2.2.3 Display Host

The Display Host for the observer runs a multi-headed X-Server. No applications are run locally on the Display Host except a Web browser, should the operator choose to launch one on the acquisition computers. 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.

Through 2015, we used Display Hosts with 3 or 4 monitors. Multiple monitors still exist on the TCS console but we are working to eliminate them. Single, high resolution screens are available today such as the “4K” 32 inch monitors which have been deployed for ike and maka. For most of the first decade of 2000’s, these were 3 separate 1600x1024 screens configured as a single logical screen (SLS, or Xinerama) totalling 4800x1024 pixels. Today, the single monitor has a resolution of 3840x2160.

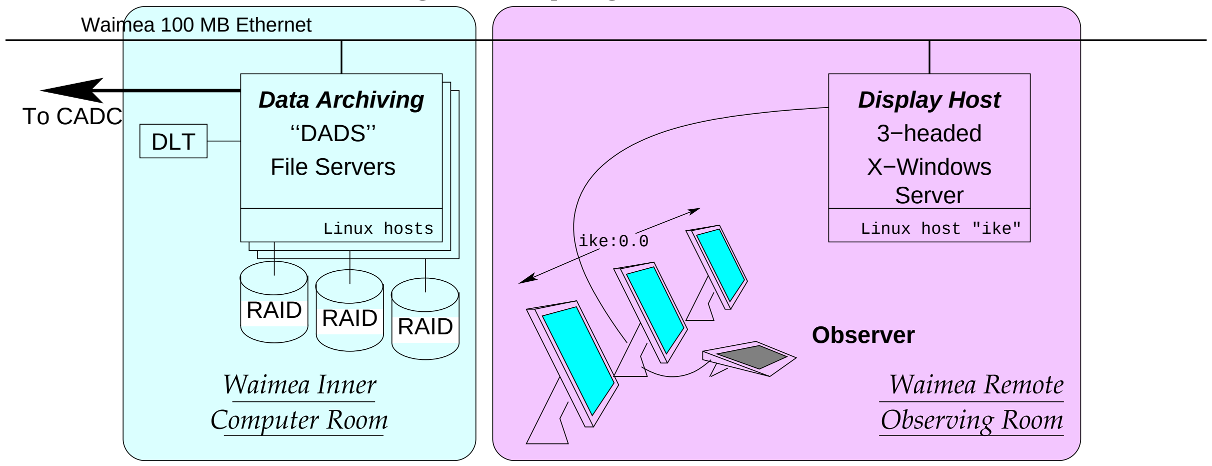
2.3 Waimea Headquarters

A high speed network 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.

Figure 4: Computing Network: Waimea



3 Hardware Platforms

All computers currently being deployed at CFHT are intel architecture, but we have a history which started in Hewlett-Packard (and some Sun Microsystems) workstations. Except for format (rack mountability versus a tower) the same machine can typically 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. Many decades of this policy has proven that system failures (power supplies, hard drives, fans) are low enough that this is a cost effective approach.

To minimize failures that do occur, it helps to have an understanding of the factors which can affect critical machines adversely, especially those at the summit:

- Unreliable memory (we should use only error-correcting RAM from a reliable source)
- "mini" CPU cooling fans (Dell's workstations use larger fans which live longer)
- Marginal chipsets (We use mostly 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.

4 Operating Systems

We use Linux, and preferably 64-bit Linux where possible, for everything to do with data acquisition except for Network Attached Storage units which run FreeBSD, though those are basically “appliances” which only do a single function, that of serving files over NFS protocol.

For data acquisition, specifically, detector hosts, session hosts, and display hosts currently use Sidious Linux (version 12) which has been assembled in-house. None of the software that has been written relies specifically on anything provided by Sidious Linux other than versions of system packages also available on Debian or CentOS or other distributions, but we get to control them tightly. We also change them very infrequently. This is by design, to provide maximum stability to the environment which must run this software for many, many years - often long beyond the point where other distribution’s updates have broken things.

Our MegaCam Detector Hosts run a special version of Linux Kernel 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. It will be phased out this year (2016) with the roll-out of “MegaCam FAST” upgrades. In the past decade-and-a-half, advances in computer hardware and the Linux kernel itself have eliminated the need for a special kernel for this application.

5 Acquisition Software Philosophy

For a good discussion of this topic (and also a good general picture of the current software blocks beyond just “data acquisition”, see our 2015 SPIE paper.

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 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

Multi-Head configurations are being phased out in 2015-2017. Refer to old (2003) versions of this document if you needed information about SLS, Xinerama, and multiple displays.

6.1.2 Virtual DeskTop and The Pager

Although we're eliminating multiple monitors, we're still leaving in place virtual desktops (also known as workspaces, or previously known as a "Pager" in FVWM). The concept is pretty commonly understood now so there's no need to go into detail describing how it works here.

6.2 X-Server

These days (2016) we run everything with the open source Xorg X-Server. Previously, we used servers from the XFree86 project, and also a commercial one from Xinside Graphics, but these days Xorg is pretty much the standard. We have no immediate plans or need to look at Wayland or Mir for our applications at this time.

6.2.1 Drivers

Historically, we've had to support a wide range of graphics hardware but as of 2016 it's pretty much all graphics chips from NVidia (covered by their closed-source driver, though "nouveau" is available for use as well if we need to use it) and intel integrated graphics. There might be a few ATI/Radeon floating around but we can usually support those and other odd graphics card through the VESA standard driver. Remember that our applications do not have an particularly high performance needs. We just need to display a lot of pixels. There is no need for the best 3D or even 2D acceleration for any of the observer's needs.

6.2.2 8-bit PseudoColor, DirectColor, and Overlays

8-bit PseudoColor is definitely a thing of the past by now, but believe it or not we were still hanging on to it 10-15 years ago when the rest of the world had long since moved on because of the way some astronomy-specific image display programs worked. If you don't know what this is, don't worry. It's dead. So are 8-bit Overlays. DirectColor mode may have been interesting to us when computers were slower, but it's also not worth looking at anymore.

6.2.3 24-bit (or 32-bit) TrueColor

All of our new displays and display drivers use 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.

6.3 Window Manager

Window Session software running on the Session Host has been kept purposely light weight. We do not use Gnome session. In fact, we run no desktop/session manager at all, only a basic window manager. The window manager through the mid 90's used to be one called MWM (Motif Window Manager), but then we transitioned to FVWM 1.24r (because it supported virtual desktops, was more lightweight), later FVWM 2.4 (because it supported the multiple monitors which are now mostly gone) and finally XFCE today. XFCE4 is the window manager that Linus Torvalds (was? is?) using. It's very lightweight and does the job for us.

7 Infrastructure

7.1 Status

See Tom for a general overview of the Status Server and related infrastructure.

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

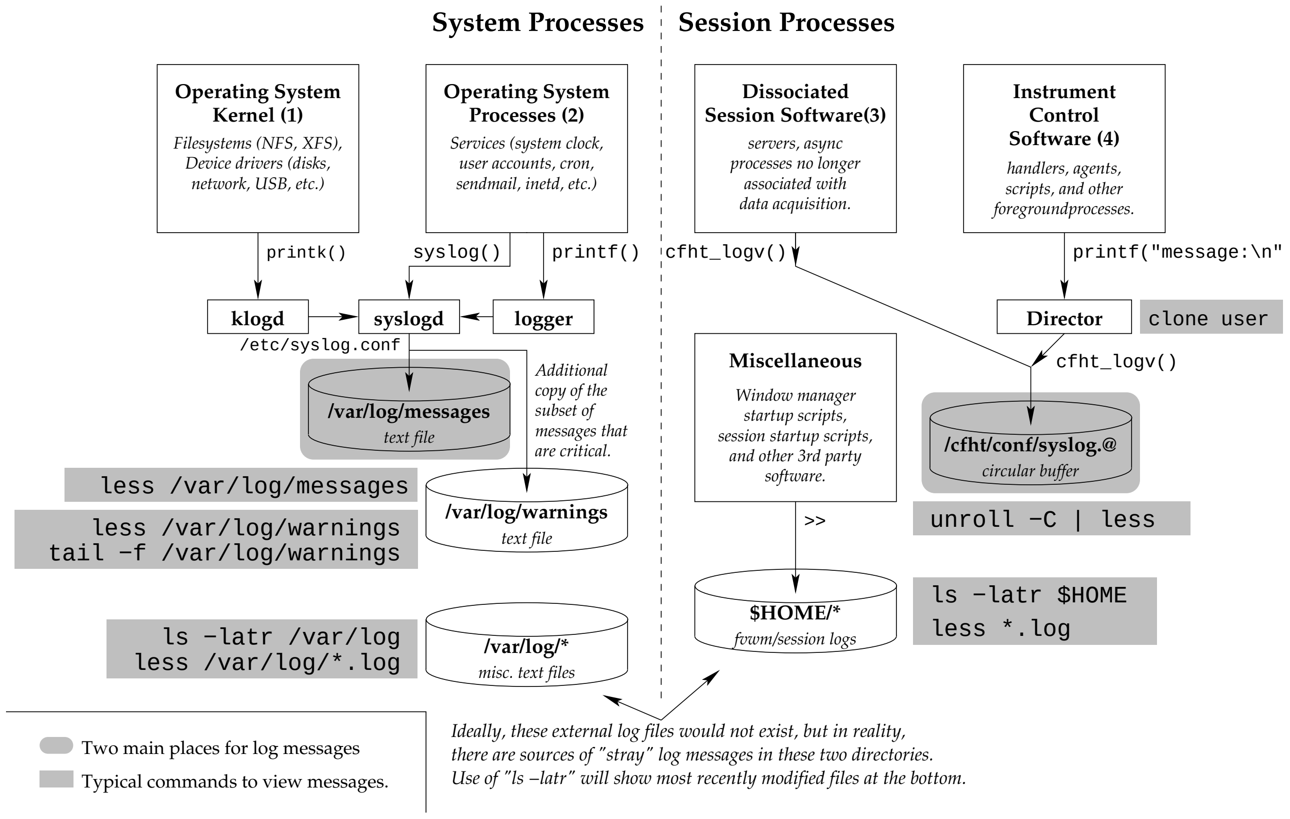
The above being the ideal for system software, the ideal for application software items 3 and 4 is similar. Dissociated application software uses the cfht_logv() interface – similar to unix syslog() – which relays messages to a process called "roll" and puts them in a circular buffer on the local machine 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

Figure 5: Sources of Logging Messages


System Processes
Session Processes
Operating System Kernel (1)
Filesystems (NFS, XFS), Device drivers (disks, network, USB, etc.)
Operating System Processes (2)
Services (system clock, user accounts, cron, sendmail, inetd, etc.)
Dissociated Session Software(3)
servers, async processes no longer associated with data acquisition.
Instrument Control Software (4)
handlers, agents, scripts, and other foregroundprocesses.
printk()
syslog()
printf()
cfht_logv()
printf("message:\n"
klogd
syslogd
logger
Director
/etc/syslog.conf
cfht_logv()
Additional copy of the subset of messages that are critical.
/var/log/messages
text file
Miscellaneous
Window manager startup scripts, session startup scripts, and other 3rd party software.
/cfht/conf/syslog.@
circular buffer
/var/log/warnings
text file
>>
$HOME/*
fvwm/session logs
/var/log/*
misc. text files
Ideally, these external log files would not exist, but in reality, there are sources of "stray" log messages in these two directories. Use of "ls –latr" will show most recently modified files at the bottom.
Two main places for log messages
Typical commands to view messages.

```
clone user
```

```
less /var/log/messages
```

```
less /var/log/warnings
tail -f /var/log/warnings
```

```
unroll -C | less
```

```
ls -latr $HOME
less *.log
```

```
ls -latr /var/log
less /var/log/*.log
```

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

See Sidik for a general overview of 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 NAS'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

Data Acquisition software is mostly written in C. There are also a lot of Bourne Shell scripts (intepreted by GNU bash.)

C++ has been used in Director and RPM.

QSO User Tools are implemented in Java.

Astronomers are starting to use a lot of Python. If they intend their Python to eventually be used as real-time analysis during data acquisition, it is requested that they use Python 2.7.9, and limit the modules and extensions they use to the versions present on our current sessionhost, alakai. I.e., they can develop code where ever they like, but test it on a machine like alakai, such as the software development host mohala (which is identical to alakai at the summit.)

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 -Dxxxx 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:

`-D`**SUNOS** SunOS 4.1.x

`-D`**HPUX** HP-UX 9 or 10

`-D`**LINUX** Linux 1.x or 2.x

`-D`**CYGWIN32** 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.
```

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 hostname 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, LaTeX, LaTeX2HTML, 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 LaTeX 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 LaTeX 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, LaTeX generates output with fairly large margins, intended for book printing and later trimming.
- The `times` package causes LaTeX to use a standard PostScript font, eventually resulting in smaller and higher quality PostScript and PDF.
- The `html` package is part of the LaTeX2`HTML` translator and allows, among other things, hyperlinks in the LaTeX 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 LaTeX2`HTML`. 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 `mohala:/local/cfht/src/`, which contains subdirectories `"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/

Some of our programs actually have man pages (documentation.)

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. You can look at `ypcat -k auto.cfht` 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 that 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: $(OBJS) -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 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

Changes files may still serve some purpose, but in general they can probably go away when we start using SVN for everything. You can create a branch in SVN and label it accordingly, or for snapshots along the way that don't need to be branches, SVN also supports tagging releases.

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 `Makefile`s 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

We’ll move /cfht/src/ completely under the management of an SVN repository! The following information will be kept here just a short while longer because it still/also applies to DNS zone files, automount maps, *and other system things which are also on the path to be changed soon.*

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. (–¿ *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

We will use SVN copies (tags, branches) for this once we have migrated everything over from RCS.

11.7.1 Summary of Archiving/File maintenance commands

ls -latr
  Lists all files with the most recent ones at the bottom.

find -newer *somefile*
  Lists files modified after *somefile*.

cp -av *orignal backup*
  Creates exact backup copies of files or directories.

diff -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.

tar czvf FILE.tgz FILES
  Creates compressed tar file. (Use ”make tar” on source directories.)

tar dzf FILE.tgz
  Lists differences between files on disk and tar file.

tar 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 (1980s!) 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. All current (2016) sessions build on Pegasus software but have completely abandoned the auto-build design. These are known as "NEO" (New Environment for Observing) sessions.

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. The "NEO" account has been abandoned for so long, it would be unwise to look at as a good example of how things are set up.

The following subsections describe the generic NEO account, and also how to convert a Pegasus account to use as much of the NEO things (XFCE, 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:/usr/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 `/usr/bin/bash`. "Bash" is the only shell which has a consistently configured version across all platforms at CFHT.

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/LINUXautomount/auto.h` also on nui.

```
neo         alakai:/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 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.

`.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

#
# Set sessionhost here.
#
echo export SSO_HOST=megacam-session
echo export SSO_WMNAME=xfce
#echo export SSO_WMNAME=fvwm2
```

Another (currently unused) 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 /usr/wm/common/sso
fi
```

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

`.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.USERNAME” 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
```

`.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.

More general information on how .fvwm* files are handled in the CFHT environment are given in /usr/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.USERNAME`

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.USERNAME 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.USERNAME 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/ltb.lod ~/dsp/lub.lo
```

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.

`.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.

12.3.3 GUI Form(s)

There are none. Data Aqcuisition is pretty much fully command-driven in Queue Mode now.