Status Server Requirements

Tom Vermeulen

24 May 2002

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

Contents

1 Introduction **4**

1.1 Purpose 4

1.2 Scope 4

2 General Description **4**

3 Functional Requirements **4**

3.1 Clients must have the ability to create, update, and retrieve objects. 4

3.2 Clients must have the ability to remove objects. 4

3.3 Information within the Status Server must be open and accessible to any connected client within the CFHT network. 5

3.4 Clients must be able to store String, Boolean, Floating Point, and Integer objects. 5

3.5 Clients must be able to place a monitor on objects. 5

3.6 Clients must be able to specify a “deadband” range for monitored floating point and integer objects. . 5

3.7 Clients must be able to specify a minimum age for monitored objects. 5

3.8 Clients must be able to specify the length of time an object can be considered valid. 6

3.9 Objects must be stored in an organized fashion, much like a file system. 6

3.10 The Status Server must have have the facilities to support an archive agent (client). 7

4 Interface Requirements **7**

4.1 Status Server must be accessible via a TCP/IP socket-based networking protocol. 7

4.2 Status Server must be accessible via a C API. 7

4.3 Status Server must be accessible via Shell utilities. 7

5 Performance Requirements **7**

5.1 Individual objects within the Status Server should be updated at a maximum frequency of one hertz. . 8

5.2 The maximum storage size associated with the value of an object within the Status Server should be fixed. 8

5.3 As a goal, the typical transaction latency should be less than 10 milliseconds. 8

6 Security Requirements **8**

6.1 Access must be restricted to the CFHT internal network. 8

7 Disaster Recovery Requirements **8**

7.1 Data must be serialized every 10 minutes for possible recovery in case of a system failure. 8

8 Remote Diagnostic Requirements **9**

8.1 Must be able to telnet into the Status Server. 9

8.2 Must have the ability to initiate a trace on all Status Server activity. 9

9 System Requirements **9**

9.1 Status Server must be designed to run on a UNIX machine within the CFHT network. 9

9.2 Socket implementation must be compatible with the ”Internet Protocol Version 4” (IPv4). 9

10 Document Change Log **9**

A Preliminary Status Server Information **10**

1 Introduction

1.1 Purpose

The purpose of this document is to list a consensus set of requirements for the Status Server which were identified by the user community. Requirements were identified via one-on-one interviews and as a result of the general discussion on December 26th. The intended audience is future users of the Status Server. This release of the document was reviewed by members of the Software Group and incorporates this feedback.

1.2 Scope

Unless otherwise noted, the requirements identified in this document are intended to be implemented in the first release of the Status Server. However, release requirements may dictate the priority and staging of functionality.

2 General Description

The Status Server will serve as an open repository of status and state information easily available to any client within the CFHT network. Clients will be able to view, update, and subscribe to data elements within the Status Server. In some respects, the Status Server could be thought of as a shared memory pool for multiple clients.

Some examples of where the system could be used include:

- A central staging area for the building of FITS files replacing the current template file. This would allow FITS files to be built in a more parallel fashion without the need for client-side synchronization.
- A possible replacement for the state information within passive text file databases (”par” files). Current retrieval or update of information within ”par” files requires a relatively inefficient file scan from NFS mounted files.
- A source for information used in GUI status displays.
- A source for the latest plant monitoring information.

This document describes the requirements identified by the user community for the Status Server. Beyond the critical functional requirements of what the system must provide, the most important requirements are reliability, performance, and flexibility. As more and more clients start to use the Status Server, it is absolutely critical that the system does not fail. In addition, as more clients are added, the number of transactions will be increased. It is important, that the system be optimized to easily handle a large number of transactions. Finally, the system must be flexible, since it must grow and evolve to store increasing amounts of information.

3 Functional Requirements

3.1 Clients must have the ability to create, update, and retrieve objects.

Any client on the CFHT network must have open access to create, update, and retrieve information from the Status Server at any time.

3.2 Clients must have the ability to remove objects.

In some cases, objects within the Status Server may have a short lifetime, so it must be possible to remove an object after it is no longer needed. If the Status Server is used as a staging area for building FITS headers, the FITS headers should probably be removed after the FITS file is created.

3.3 Information within the Status Server must be open and accessible to any connected client within the CFHT network.

Once connected to the Status Server, any client must have the ability to add, update, retrieve, or remove information within the Status Server. This implies that there will not be any required permissions or membership to access control lists in order to view or modify data.

3.4 Clients must be able to store String, Boolean, Floating Point, and Integer objects.

The Status Server must support the ability to store a variety of data types. At a minimum, the following data types must be supported.

- **String** - NULL terminated string with a maximum defined length. The maximum length must be defined to be at least 80 characters to accomodate FITS headers. Strings will consist of a sequence of 8 bit ASCII characters.
- **Boolean** - Data type consisting of two possible values; either true or false.
- **Floating Point** - Double precision floating point number. This number will be internally represented within 64 bits and has a range of roughly +/- 1.79769313486231570E+308.
- **Integer** - Signed integer number. The number will internally be represented within 32 bits and has a range of -2,147,483,648 to 2,147,483,647.

3.5 Clients must be able to place a monitor on objects.

In some cases, a client may be interested to know when the value of a particular object changes, but not interested in constantly polling the Status Server. As a result, it must be possible to place a monitor on objects. By placing a monitor on an object, the client will be informed when the object has changed state and what the new value of the object is.

3.6 Clients must be able to specify a "deadband" range for monitored floating point and integer objects.

In order to reduce the load on both Status Server and client, the client should be able to specify a "deadband" range for both floating point and integer objects. For example, given that the telescope pointing accuracy is around .1 arcseconds, it may make sense to provide a deadband limit surrounding the actual RA and Dec position of the telescope. As a result, monitors would not trigger for cases where a new value is not significantly different from a previous value.

3.7 Clients must be able to specify a minimum age for monitored objects.

The frequency with which a Status Server object is updated may exceed the desired notification frequency for a client monitoring the object. As a result, it must be possible for a client to specify a minimum time period between monitoring notifications. For example, if the datalogger information is stored in the Status Server, it would be updated every 10 seconds. However, a client monitoring a series of datalogger probes may only be interested in receiving an update each hour.

3.8 Clients must be able to specify the length of time an object can be considered valid.

Some of the status information within the Status Server will have a limited useful lifetime beyond which the information should be considered expired or invalid. As an example, the seeing at the end of one evening should not be taken as the current seeing at the start of the next evening.

The valid lifetime of a status object must be the time frame within which clients would be notified of status updates of monitored objects.

3.9 Objects must be stored in an organized fashion, much like a file system.

Status Server objects must be grouped together in a tree-like fashion much like a file system. It would then be possible to traverse and manipulate objects within the Status Server much like traversing a directory tree in a file system. In such a system, it must be possible to access and refer to objects within the Status Server either via a fully qualified path-name combination or a relative path-name combination. This will require that the Status Server or API library keep track of the current directory for each client. An example of such a structure is shown in figure 1.

```
# Status Server database
# ----------------------
#
# Format is /path/and/varname =  # Description in comment#
# Formatting of <sample value> indicates "native" type for this field.
# This is handled internally by the routine that serializes the database.
#
# TRUE/FALSE     - Booleans will have *un-quoted* TRUE or FALSE as the value.
# "string"       - Strings always saved with " as the first char in value field.
# 10.            - Float values will always show a decimal point (even if .0)
# 15             - Numeric values without a decimal indicate integers.
#
# Top-level "directories":
#
# /i/ with a subdirectory for each instrument (often same names as handlers)
# /t/ with subdirectories for each telescope subsystem
# /p/ for plant environment, weather, data-logger variables
# /f/ has subdirectories for each exposure where FITS headers are accumulated
#
/i/                                          # Instruments
/i/megacam/                                  # Megacam agent stuff
/i/megacam/etime        = 10.                # Current exposure time
/i/megacam/etype        = "BIAS"             # Current exposure type
/i/megacam/filter       = 0                  # Current filter position
#
# /i/cfh12k/ are all generated by 12kcom(detcom) and used to be in .,12kcom.par
#
/i/cfh12k/status        = "Idling"           # Camera status for GUI
/i/cfh12k/raster        = "FULL"             # Current raster setting
/i/cfh12k/etime         = 10.                # Current exposure time
/i/cfh12k/etype         = "FLAT"             # Current exposure type
/i/cfh12k/filter        = 0                  # Current filter position
/i/cfh12k/filter[0]     = "R"                # Desc. of filter in slot 0
/i/cfh12k/filter[1]     = "V"                # Desc. of filter in slot 1
/i/cfh12k/filter[2]     = "B"                # Desc. of filter in slot 2
/i/cfh12k/filter[3]     = "I"                # Desc. of filter in slot 3
/i/cfh12k/observer      = "Galileo"          # Current OBSERVER header
/i/cfh12k/object        = "TF dawn"          # Current OBJECT header
/i/cfh12k/comment       = "Twilight flats"   # Current CMMTOBS header
/i/cfh12k/piname        = "Mellier"          # Current PINAME header
/i/cfh12k/runid         = "99IIF142"         # Current RUNID header
```

Figure 1: Status Server Hierarchical Name-Value Pair Representation

3.10 The Status Server must have have the facilities to support an archive agent (client).

The Status Server should be designed to hold current information. However, it is likely that some information will need to be stored or archived for later use. It is quite possible that the monitoring capabilities of the Status Server will be sufficient for an archive client to compile a history. If not, the Status Server must be designed in a way so this support is available.

4 Interface Requirements

4.1 Status Server must be accessible via a TCP/IP socket-based networking protocol.

Since the systems containing key status and state information may reside on different hosts and systems within the CFHT network, this information must pass through the network to make it's way to the Status Server. At CFHT, this network consists of a number of hubs, switches, and routers which pass TCP/IP packets.

As a result, the underlying protocol used to transfer Status Server information must be based on TCP/IP.

4.2 Status Server must be accessible via a C API.

C is the most common language used within most client systems and, as a result, a client API to access the Status Server must be available in C. This API must be available for each of the three major UNIX platforms at CFHT; Linux, Solaris, and HP.

It is probable that library support for additional languages will be created following the C API. Tcl/TK and Java have been mentioned as possible additions.

4.3 Status Server must be accessible via Shell utilities.

Shell utilities must be available on Linux, Solaris, and HP to access information within the Status Server.

Utilities must be available to enable the creation, update, retrieval, and removal of objects. Due to the limited state nature of a shell script, it may not be possible to access all the monitoring functionality. However, it may be useful to have the ability to specify an object with an initial value and have the shell return back a new value whenever it changes in the Status Server.

5 Performance Requirements

There are a number of internal and external factors which will affect the performance of the Status Server. The hardware platform, operating system, network bandwidth, packet size, CPU load, and resident memory will all play a part in the throughput, performance, and latency of the Status Server. The target platform and implementation should be designed to maximize performance and throughput and limit latency.

Assuming a well planned design and implementation, it is likely that the overall performance of the Status Server will depend largely on the load placed upon it by each connected client. As a result, it is important to characterize the type of data to be stored in the Status Server and the update frequency of this data.

5.1 Individual objects within the Status Server should be updated at a maximum frequency of one hertz.

Each time an object in the Status Server is updated, there is a chance that the object is being monitored by multiple clients. As a result, a single update can trigger a number of additional actions by the Status Server. In order to maintain sufficient overall performance and throughput, clients must restrict the number of updates to the Status Server and specify an ”age” and ”deadband” range wherever possible. If a client requires a specific piece of information at a more frequent interval, a direct API should be considered between subsystems instead of using the Status Server.

5.2 The maximum storage size associated with the value of an object within the Status Server should be fixed.

The Status Server should be designed to hold a series of small objects. By restricting the maximum size of each element, it is possible to prevent memory and performance bottlenecks as well as help characterize the type of information the Status Server should be designed to store.

A file system should be considered as an alternative for larger pieces of information which must be stored and shared.

5.3 As a goal, the typical transaction latency should be less than 10 milliseconds.

As mentioned earlier, the latency will depend on a number of factors. However, an initial goal should be a latency of less than 10 milliseconds for a round-trip transaction (request-response) over a 100 mbps LAN within the same subnet. Accurate benchmark figures should be available once the Status Server has been implemented.

The 10 milliseconds target is based on some benchmarking performed using the single-threaded non-blocking socket server which the QSO Tools use to send commands to director. Typical round-trip latency on an unloaded Pentium 3 500 Mhz server via a 100 mbps LAN is roughly 3 milliseconds. This includes the time to send a command over the network, parse the command, fork a child process, send the command to director via a cli_cmd API call, receive a PASSFAIL response from the child process, and forward the response over the network to the client. In this case, the command used for testing purposes was a say command with an 80 character message.

While the processing the Command Server, used by the QSO Tools, performs is quite different than that of the Status Server, 10 milliseconds should be a reasonable first estimate.

6 Security Requirements

6.1 Access must be restricted to the CFHT internal network.

Access to the Status Server must be restricted to clients within the CFHT network. Using a low numbered port (below 1024) should be sufficient to enable blocking by the router. If access is limited via the port number, it would still be possible for employees to access the Status Server from outside the CFHT network, via ssh port forwarding.

7 Disaster Recovery Requirements

7.1 Data must be serialized every 10 minutes for possible recovery in case of a system failure.

The Status Server must be designed to be as reliable as possible. Reliability must be the number one goal throughout the design, development, and testing phase.

However, down-time is always a possibility either due to a bug in the Status Server, a problem on the machine hosting the Status Server, or for required maintenance or upgrades. As a result, clients must be designed to handle the condition of not being able to connect to the Status Server. In addition, the Status Server must serialize a copy of itself to disk every 10 minutes. The Status Server must then have the ability to be restarted by first loading it's state information from the serialized disk copy. Using this approach, it should be possible to restart the Status Server on another machine in case there is a problem with the machine running the Status Server.

8 Remote Diagnostic Requirements

8.1 Must be able to telnet into the Status Server.

For remote diagnostic purposes, it must be possible to telnet into the Status Server and view information contained within the Server. In order to facilitate this requirement, information must be stored in a human-readable format (not binary data). It must be possible to replicate the same sequence of socket commands used by the API library within a telnet session.

8.2 Must have the ability to initiate a trace on all Status Server activity.

For failure diagnosis and testing purposes, it must be possible to initiate a trace on all Status Server activity.

9 System Requirements

9.1 Status Server must be designed to run on a UNIX machine within the CFHT network.

With UNIX machines being the standard within the CFHT observing environment, the Status Server must be compiled and designed to run on the HP, Solaris, and Linux platforms. In addition, the C API library must be compiled for the HP, Solaris, VxWorks, and Linux platforms as well.

9.2 Socket implementation must be compatible with the "Internet Protocol Version 4" (IPv4).

The networking implementation within the CFHT network is IPv4 and will most likely continue this way for the some time into the future. It's replacement "Internet Protocol Version 6" (IPv6) provides for a larger address space as well as a number of other features. When (and if) the Internet and the CFHT network transitions to the new standard, the Status Server must be modified to support the new standard.

10 Document Change Log

Version	Date	Comments
1.0	March 7, 2002	First release for review.
1.1	March 13, 2002	Revised document based on Software Group review.
1.2	May 24, 2002	Revised document to add VxWorks as a required platform for the C-API, changed the specification of the maximum string length to be at least 80 characters, and removed the ability to trace individual objects or clients.

A Preliminary Status Server Information

The following pages contain information which may be stored in the Status Server. Wherever possible, access rates, datatypes, and sizes have been identified.

```
# Status Server database
# ----------------------
#
# Format is /path/and/varname =  # Description in comment
#
# Formatting of <sample value> indicates "native" type for this field.
# This is handled internally by the routine that serializes the database.
#
# TRUE/FALSE     - Booleans will have *un-quoted* TRUE or FALSE as the value.
# "string"       - Strings always saved with " as the first char in value field.
# 10.            - Float values will always show a decimal point (even if .0)
# 15             - Numeric values without a decimal indicate integers.
#
# Top-level "directories":
#
# /i/ with a subdirectory for each instrument (often same names as handlers)
# /t/ with subdirectories for each telescope subsystem
# /p/ for plant environment, weather, data-logger variables
# /q/ for information associated with queue observing
# /f/ has subdirectories for each exposure where FITS headers are accumulated
#
/i/                                  # Instruments
/i/currentInstrument                 # Current instrument on the telescope
/i/megacam/                          # Megacam agent stuff
/i/megacam/etime       = 10.         # Current exposure time
/i/megacam/etype       = "BIAS"      # Current exposure type
/i/megacam/filter      = 0           # Current filter position
#
# /i/cfh12k/ are all generated by 12kcom(detcom) and used to be in .,12kcom.par
#
/i/cfh12k/status       = "Idling"    # Camera status for GUI
/i/cfh12k/raster       = "FULL"      # Current raster setting
/i/cfh12k/etime        = 10.         # Current exposure time
/i/cfh12k/etype        = "FLAT"      # Current exposure type
/i/cfh12k/filter       = 0           # Current filter position
/i/cfh12k/filter[0]    = "R"         # Desc. of filter in slot 0
/i/cfh12k/filter[1]    = "V"         # Desc. of filter in slot 1
/i/cfh12k/filter[2]    = "B"         # Desc. of filter in slot 2
/i/cfh12k/filter[3]    = "I"         # Desc. of filter in slot 3
/i/cfh12k/observer     = "Galileo"   # Current OBSERVER header
/i/cfh12k/object       = "TF dawn"   # Current OBJECT header
/i/cfh12k/comment      = "Twilight flats"   # Current CMMTOBS header
/i/cfh12k/piname       = "Mellier"   # Current PINAME header
/i/cfh12k/runid        = "99IIF142"  # Current RUNID header
/i/cfh12k/autoclean    = TRUE        # Clean array during idle time?
/i/cfh12k/autosave     = TRUE        # Save during readout?
/i/cfh12k/autovoltage  = FALSE       # Poll DSP voltages?
/i/cfh12k/mef          = FALSE       # Save as one multi-ext FITS?
/i/cfh12k/mainpower    = TRUE        # Is the mainpower on?
/i/cfh12k/detectorhost  = "akua"     # Detector computer hostname
/i/cfh12k/tf12k/period  = "dusk"     # For tf12k script
/i/cfh12k/tf12k/tfiter  = 10         # For tf12k script
/i/cfh12k/focus/etime   = 10.        # For focus script
/i/cfh12k/focus/currentz= 1.691      # For focus script
/i/cfh12k/focus/newz    = 4.4        # For focus script
/i/cfh12k/focus/zcenter = 1.7        # For focus script
/i/cfh12k/focus/detlaz  = 1.         # For focus script
/i/cfh12k/focus/nframes = 2          # For focus script
#
# /i/gecko/ are all generated by geckoh and used to be in .,geckoh.par
#
```

```
/i/gecko/
/i/mos/

/t/                                    # Telescope
/t/dome/
/t/cbcs/                               # Cassegrain Bonnette Control System
/t/pfbcs/                              # Prime Focus Bonnette Control System
/t/guider/
/t/mpc/                                # MegaPrime Controller
/t/mpc/gfsu                            # Guide Focus Sense Unit
/t/mpc/fsa                             # Focus Stage Assembly
/t/mpc/wfss                            # Wave Front Sensor Software
/t/mpc/wfss/seeing = EXPIRED           # Seeing estimate value

/t/tcs/                                # TCS information (40 character strings
                                       # updated approximately 1x per second)
/t/tcs/mode                            # Telescope mode
/t/tcs/position/airmass                # Airmass at current telescope positioning
/t/tcs/position/bonnette               # Bonnette position
/t/tcs/position/dec                    # Dec pointing of the telescope
/t/tcs/position/dome                   # Position of the dome
/t/tcs/position/equinox                # Targeting equinox
/t/tcs/position/instrument             # Instrument positioning (prime, cass, coude, etc.)
/t/tcs/position/instrumentRotation     # Instrument rotation
/t/tcs/position/ra                     # RA pointing of the telescope

/p/
#
# /p/datalogger/ Data logger information will be updated every 10 seconds
#
/p/datalogger/          # Datalogger probe values
/p/datalogger/probe0    # surface temp, primary mirror east (side)      RTDF85  deg celsius
/p/datalogger/probe1    # surface temp, primary mirror west (side)      RTDF85  deg celsius
/p/datalogger/probe2    # surface temp, primary mirror west (silver)    RTDF85  deg celsius
/p/datalogger/probe3    # surface temp, primary mirror east (silver)    RTDF85  deg celsius
/p/datalogger/probe4    # surface temp, cassion central west            RTDF85  deg celsius
/p/datalogger/probe5    # surface temp, cassion central east            RTDF85  deg celsius
/p/datalogger/probe6    # air temp, top ring west                       RTDF85  deg celsius
/p/datalogger/probe7    # air temp, top ring east                       RTDF85  deg celsius
/p/datalogger/probe8    # surface temp, horseshoe east top              RTDF85  deg celsius
/p/datalogger/probe9    # surface temp, horseshoe east brg pad          RTDF85  deg celsius
/p/datalogger/probe10   # surface temp, horseshoe west brg pad          RTDF85  deg celsius
/p/datalogger/probe11   # surface temp, horseshoe west top              RTDF85  deg celsius
/p/datalogger/probe12   # surface temp, electrical box north            RTDF85  deg celsius
/p/datalogger/probe13   # surface temp, electrical box south            RTDF85  deg celsius
/p/datalogger/probe14   # surface temp, coude’ arm south beam           RTDF85  deg celsius
/p/datalogger/probe15   # surface temp, south beam thrust brg           RTDF85  deg celsius
/p/datalogger/probe16   # air temp, 4th floor crawl space               THM10K  deg celsius
/p/datalogger/probe18   # air temp, north rail 5th floor                RTDF85  deg celsius
/p/datalogger/probe19   # air temp, north rail support beam             RTDF85  deg celsius
/p/datalogger/probe20   # load cell, north                              DCV     DC volts
/p/datalogger/probe21   # load cell, south west                         DCV     DC volts
/p/datalogger/probe22   # dewpoint, outside south mirror cell           DCV     deg celsius
/p/datalogger/probe23   # air temp, top ring east                       THM10K  deg celsius
/p/datalogger/probe24   # surface temp, outside north hand rail         RTDF85  deg celsius
/p/datalogger/probe25   # surface temp, east cell steel                 RTDF85  deg celsius
/p/datalogger/probe26   # surface temp, east mirror under side          RTDF85  deg celsius
/p/datalogger/probe27   # mirror cooling outlet at cell                 RTDF85  deg celsius
/p/datalogger/probe28   # computer room relative humidity               DCV     percent
/p/datalogger/probe29   # computer room temperature                     DCV     deg celsius
/p/datalogger/probe30   # dewpoint, inside south mirror cell            DCV     deg celsius
/p/datalogger/probe31   # barometric pressure, ctrl rm, weathertronics  DCV     millibars
/p/datalogger/probe32   # surface temp, west cell steel                 RTDF85  deg celsius
/p/datalogger/probe33   # air temp, west under mirror                   RTDF85  deg celsius
/p/datalogger/probe34   # relative humidity, weathertron                DCV     DC volts
/p/datalogger/probe35   # air temp, weathertron                         DCV     deg celsius
/p/datalogger/probe36   # air temp, dome top ws side                    DCV     deg celsius
```

```
/p/datalogger/probe37   # air temp,dome top other side                  DCV     deg celsius
/p/datalogger/probe38   # air temp, lower weatherstation side           DCV     deg celsius
/p/datalogger/probe39   # fluid temp, horseshoe top NW pad              RTDF85  deg celsius
/p/datalogger/probe40   # fluid temp, horseshoe bottom NW pad           RTDF85  deg celsius
/p/datalogger/probe41   # fluid temp, horseshoe top NE pad              RTDF85  deg celsius
/p/datalogger/probe42   # fluid temp, horseshoe bottom NE pad           RTDF85  deg celsius
/p/datalogger/probe43   # air temp, 6’ above 5th floor                  RTDF85  deg celsius
/p/datalogger/probe44   # fluid temp, telescope hyd SW pad              RTDF85  deg celsius
/p/datalogger/probe45   # air temp, 2" above 5th floor                  RTDF85  deg celsius
/p/datalogger/probe46   # fluid temp, tele hyd 1st floor supply         RTDF85  deg celsius
/p/datalogger/probe47   # fluid temp, tele hyd 1st floor return         RTDF85  deg celsius
/p/datalogger/probe48   # Inner Coude, Gecko Detector Env. Struct.      RTDF85  deg celsius
/p/datalogger/probe50   # chiller temp control                          DCV     DC volts
/p/datalogger/probe51   # surface temp, 5th floor track                 RTDF85  deg celsius
/p/datalogger/probe53   # air temp, 5th floor 2" by electronics box     THM10K  deg celsius
/p/datalogger/probe54   # air temp, mirror cell west                    THM10K  deg celsius
/p/datalogger/probe55   # axial load cell north                         DCV     kilograms
/p/datalogger/probe56   # axial load cell south west                    DCV     kilograms
/p/datalogger/probe57   # axial load cell south east                    DCV     kilograms
/p/datalogger/probe58   # mirror cooling inlet at unit                  RTDF85  deg celsius
/p/datalogger/probe59   # surface temp, concrete floor north pier       RTDF85  deg celsius
/p/datalogger/probe60   # surface temp, concrete floor south pier       RTDF85  deg celsius
/p/datalogger/probe61   # surface temp, concrete floor above control rm. RTDF85  deg celsius
/p/datalogger/probe62   # fluid temp, 1st floor glycol supply           RTDF85  deg celsius
/p/datalogger/probe63   # air temp, pm, south mid. air line             RTDF85  deg celsius
/p/datalogger/probe64   # air temp, spigot north, near cass. bonn.      RTDF85  deg celsius
/p/datalogger/probe65   # air temp, spigot north, near M3               RTDF85  deg celsius
/p/datalogger/probe66   # surface temp, pm, north,under side,near spigot RTDF85 deg celsius
/p/datalogger/probe67   # air temp, pm, north mid. air line             RTDF85  deg celsius
/p/datalogger/probe68   # fluid temp, tele hyd resevoir                 RTDF85  deg celsius
/p/datalogger/probe70   # surface temp, pm, south,under side,near spigot RTDF85 deg celsius
/p/datalogger/probe72   # fluid temp, tele hyd chill outlet             RTDF85  deg celsius
/p/datalogger/probe73   # air temp, observing room                      RTDF85  deg celsius
/p/datalogger/probe74   # air temp, rear-observing room                 RTDF85  deg celsius
/p/datalogger/probe75   # air temp, computer room #3                    RTDF85  deg celsius
/p/datalogger/probe76   # air temp, computer room                       RTDF85  deg celsius
/p/datalogger/probe78   # load cell east                                DCV     DC volts
/p/datalogger/probe79   # used as status indicator                      ---     ---
/p/datalogger/probe80   # F8 LVDT N  (x10)                              DCV     position
/p/datalogger/probe81   # F8 LVDT SE (x10)                              DCV     position
/p/datalogger/probe82   # F8 LVDT SW (x10)                              DCV     position
/p/datalogger/probe83   # F8 FLOW    (x100)                             DCV     remote vac. flow
/p/datalogger/probe84   # weather tower wind speed                      DCV     knots
/p/datalogger/probe85   # weather tower wind dir                        DCV     deg
/p/datalogger/probe86   # weather tower temp                            DCV     deg C
/p/datalogger/probe87   # weather tower RH                              DCV     percent
/p/datalogger/probe88   # Inner Coude Ceiling                           RTDF85  deg celsius
/p/datalogger/probe89   # Inner Coude Air, 1.8 m Above Floor            RTDF85  deg celsius
/p/datalogger/probe90   # Inner Coude Air, 0.75 m Above Floor           RTDF85  deg celsius
/p/datalogger/probe91   # Inner Coude Floor                             RTDF85  deg celsius
/p/datalogger/probe92   # Dewpoint of telescope air prior to cos. reg.  DCV     deg celsius
/p/datalogger/probe93   # Temperature of telescope air prior to cos. reg. DCV   deg celsius
/p/datalogger/probe95   # average wind speed            (84)            ---     ---
/p/datalogger/probe96   # average wind direction        (85)            ---     ---
/p/datalogger/probe97   # average temperature           (86)            ---     ---
/p/datalogger/probe98   # average relative humidity     (87)            ---     ---
/p/datalogger/probe99   # current applied to floor chiller control valve DCI    milliamps

/p/sky/                 # Sky characteristics
/p/sky/currentSeeing    # Quantitative meas. of current seeing          1x/min  double
/p/sky/skyBackground    # Quantitative meas. of the sky background      1x/min  double
/p/sky/photometry       # Quantitative meas. Of sky photometry          1x/min  double

/q/                     # Queue Observing parameters
/q/qrunid               # Run ID associated with a Queue run            1x/day  String  8 chars
/q/observer             # Service Observer for the night                1x/day  String  30 chars
/q/coordinator          # Queue Coordinator for the night               1x/day  String  30 chars
```

```
/q/activeQueue          # Current Queue being executed                  8x/day  String  20 chars
/q/currentOG            # Current Observing Group being executed        6x/hr   String  8 chars
/q/currentOB            # Current Observing Block being executed        12x/hr  String  8 chars
/q/currentOBiter        # Current Observing Block iteration             12x/hr  String  3 chars
/q/currentTarget        # Current target                                12x/hr  String  40 chars
/q/currentIC            # Current Instrument Configuration              18x/hr  String  8 chars
/q/currentOGid          # Database ID for the current OG being executed6x/hr   String  20 chars
/q/currentOBid          # Database ID for the current OB being executed12x/hr  String  20 chars
/q/currentICid          # Database ID for the current IC being executed18x/hr  String  20 chars

/f/                     # FITS header cache area
/f/131536o/             # FITS header cache for 131536o.fits
/f/131536o/000202.0="DATE-OBS= '2000-04-26'         / UTC Start of observation"
/f/131536o/000203.0="UTC-OBS = '12:17:43.71'        / New name is UTIME"
/f/131536o/000203.1="UTIME   = '12:17:43.71'        / UT start of observation"
/f/131536o/000204.0="MJD-OBS =        51660.5123114 / New name is MJDATE"
/f/131536o/000204.1="MJDATE  =        51660.5123114 / Modified Julian Date"


MegaCam Information (compiled by Will Rambold)

MegaPrime data sources for data server     Note:  Maximum update rate for any object is 1Hz


Camera                                  When              Via detcom
    Produces
        All the usual detcom stuff      Start of exposure
             "                          End of exposure


Cryogenics Controller                   When              Via status agent
    Produces
        Array temperature               exceeds deadband
        Cold Capacity temperature       exceeds deadband
        Component temperatures          exceeds deadband
        Helium compressor status        on change
        Heater current                  exceeds deadband
        Vacuum reading                  exceeds deadband
        Warning alarms                  on change
        Fault alarms                    on change


Shutter Controller                      When              Via status agent
    Produces
        Shutter position                on change
        Transit times                   When shutter closes
        Exposure time                   When shutter closes
        Flat field source status        on change
        Warning alarms                  on change
        Fault alarms                    on change


Filter Controller                       When             Via status agent
    Produces
        Filter ID list                  on change
        ID of filter in beam            on change
        Component positions             on change
        Warning alarms                  on change
        Fault alarms                    on change


MegaPrime Controller                    When              Via TCS
    Produces
        Probe positions                 on change
        Stabilizing system status       on change
        ISU position                    exceeds deadband
        Guiding errors                  exceeds deadband
```

```
        Guiding state                   on change
        Guide source statistics         exceeds deadband
        Focus position                  exceeds deadband
        Autofocus state                 on change
        Power supply voltages           exceeds deadband
        Fault alarms                    on change


Environment Controller                  When             Via Allen-Bradley
    Produces
        Temperature inside cover        exceeds deadband
        Temperature outside cover       exceeds deadband
        Temperature inside CUTE         exceeds deadband
        Temerature outside CUTE         exceeds deadband
        Component temperatures          exceeds deadband
        Glycol temperature at cover     exceeds deadband
        Glycol flow When at cover       exceeds deadband
        Glycol flow When at CUTE        exceeds deadband
        Flow control position for pulse tube  exceeds deadband
        Flow control position for cover exceeds deadband
        Flow control position for CUTE  exceeds deadband
        Fault alarms                    on change

    Consumes
        Inside dome temperature         on change
        Inside dome humidity            on change


Other Stuff                             When             Via telnet
    Produces
        AC outlet state                 on change
```