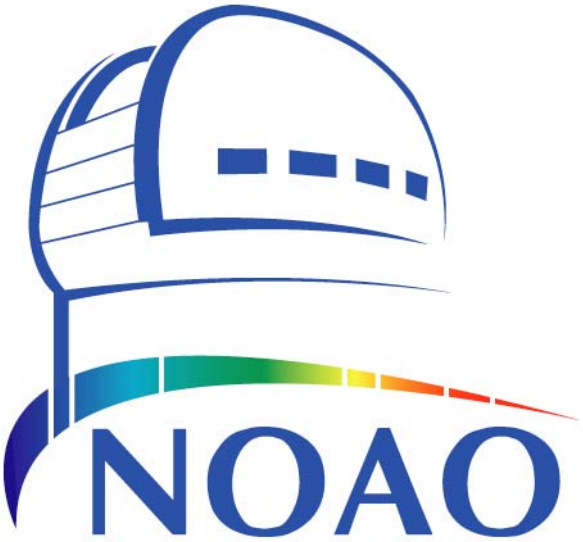
National Optical Astronomy Observatories

GNIRS

GEMINI NEAR INFRA-RED SPECTROGRAPH

SOFTWARE MANUAL

Created by



Peter Ruckle
November 14, 2002

TABLE OF CONTENTS

1 Introduction

1-1 Description

1-1.1 Purpose

This document should help a knowledgeable programmer understand the structure of the GNIRS software system for the purpose of installation, modification, or debugging.

1-1.2 Scope

This document discusses installation and setup of the entire GNIRS software system. It also goes into detail on the structure of the Components Controller and the Instrument Sequencer. The Detector Controller is discussed in order to flag differences between the GNIRS Detector Controller and the equivalent NIRI Detector Controller. The Wavefront Sensor sections also discuss only the modifications from the NIRI installation.

1-2 References

1) gscg.grp.???, *"ICD 13 — Standard Controller*", Bret Goodrich & Andrew Johnson, Gemini 8m Telescopes Project.

2) gscg.kkg.009, "*ICD 1a — The System Command Interface",* Gemini 8m Telescopes Project.

3) gscg.grp.024, "ICD 1b — The Baseline Attribute/Value Interface", Gemini 8m Telescopes Project.

4) cics_smb_017, *ICD 6 — ICS/TCS Direct Control Interface*, Steven Beard, Royal Observatory Edinburgh.

5) cics_smb_014, *"ICD 7a — ICS Subsystem Interfaces*", Steven Beard, Steve Wampler and Chris Mayer, Gemini 8m Telescopes Project.

6) gscg.grp.025, *"ICD 16 — The Parameter Definition Format"*, Steve Wampler, Gemini 8m Telescopes Project.

7) cics_smb_013, "ICD 1.9/3.1 — Science Instrument to Observatory Control System", Steven Beard, Royal Observatory Edinburgh.

8) cics_smb_002, "Core Instrument Control System — Introduction & Specification", Steven Beard, ROE.

9) cics_smb_022, *"CICS Architectural Design Document"*, Steven Beard, Royal Observatory Edinburgh.

10) cics_smb_041, *"CICS Detailed Design Document"*, Steven Beard, Royal Observatory Edinburgh.

11) ICD 1.9.3/1.9.b.5, *"Detector Controller to GNIRS Sequencer"*, Richard Wolff, NOAO.

12) ICD 1.9.b.4/1.9.5.5, "GNIRS Component Controller to GNIRS Sequencer", Richard Wolff, NOAO.

13) gscg.kkg.010, "ICD 2 — Systems Status and Alarm Interfaces", Gemini 8m Telescopes Project.

14) GNAAC.ICD.193_195ab.fm, "ICD 1.9.3[ab] – Detector Controller to NIRI/NIRS ICS Interface", Nick C. Buchholz, NOAO.

15) gscg.grp.???, "ICD 12 – Interlock System", Peregrine McGehee

16) SPE-C-G0037, "Software Design Description", Gemini 8m Telescopes Project

17) ocs.kkg.031, "Sequence Command Specifications", Kim Gillies, Shane Walker & Steven Beard, NOAO/Gemini 8m Telescopes Project.

18) niri_hty_0006, "Near IR Imager (NIRI) and On-Instrument Wavefront Sensor (OIWFS) – Software Installation and Release Notes", Hubert Yamada, Steven Beard, Gemini 8m Telescopes Project.

19) GNAAC.SftwrMntnMan.ncb.020, "Gemini-NOAO-Aladdin Array Controller (GNAAC) Software Maintenance Manual", Nick Buchholz, Peter Ruckle, Gemini 8m Telescopes Project.

20) gnaacInstallationManual.pdf,"Installing the GNAAC Software", Nick Buchholz, Gemini 8m Telescopes Project.

21) dhs_pdr_icd1c/07, "ICD 1c – Baseline DHS Interface", Norman Hill, Severin Gaudet, Dayle Kotturi, Gemini 8m Telescopes Project.

22) dhs_pdr_icd3/26, "ICD 3 – Bulk Data Transfer", Norman Hill, Severin Gaudet, Gemini 8m Telescopes Project.

1-3 Acronyms and Abbreviations

- CAD – Command action directive – The EPICS record that starts actions (processes)
- CAR – Command action response – The EPICS record that marks the current state of a CAD or group of CAD's
- CC – Components controller (located in {BASE}/CC)
- DC – Detector Controller (located in {BASE}/DC)
- DHS – Data Handling System

- EPICS – Experimental Physics Industrial Control System
- GNAAC – Gemini NOAO Aladdin Array Controller
- GNIRS – Gemini Near InfraRed Spectrograph
- IS – Instrument Sequencer (located in {BASE}/IS)
- NIRI – Near InfraRed Imager
- SAD – An EPICS record that stores a status value
- UAE – universal application environment
- WFS – Wavefront Sensor (located in {BASE}/WFS)

1–4 Definitions

- Components Controller - (located in {BASE}/CC) The part of the instrument that controls motor motion and monitors physical status
- Detector Controller – (located in {BASE}/DC) The part of the instrument that takes images
- Instrument Sequencer - (located in {BASE}/WFS) Big Brother in the GNIRS realm.

2 Installation

There are 3 sections of code that must be setup.

- Components Controller / Instrument Sequencer / Detector Controller EPICS/ Wavefront Sensor
- Detector Controller PowerPc (coadder)
- Transputer

The Components Controller and Instrument Sequencer and Wavefront sensor are in a standard Gemini UAE which can be compiled under EPICS 3.12.2 GEM5. The Transputer code is the part of the Detector controller that communicates directly with the array. This section was never incorporated into the UAE so it must be compiled separately. The Detector Controller PowerPc uses EPICS 3.13.4 GEM7, so it must be compiled with a different environment set.

2-1 Components Controller, Instrument Sequencer, Wavefront Sensor and Detector Controller

Obtain the latest version of the GNIRS tar file. (GNIRS1.0.tar.gz)
Uncompress and untar the file into the desired directory. This directory will be the {BASE} directory referred to throughout the rest of the document. Source your local epics.csh file. The current version of the Components Controller, Detector Controller EPICS, Instrument Sequencer and Wavefront sensor code use EPICS version 3.12.2 GEM5

1. edit **{BASE}/epics.csh** and source it.
2. Modify **{BASE}/nirs.env** for your local environment.
3. Run **{BASE}/nirs.env** in the {BASE} directory. This will set the environment variables necessary for setup and compilation.
4. Modify {BASE}/dl/nirsStart for the local environment.
5. Run **gmake** from the root directory. This compiles the code in the {BASE}/IS, {BASE}/CC, {BASE}/DC, {BASE}/WFS, and {BASE} directories. The Instrument Sequencer, Components Controller, Wavefront Sensor, Detector Controller (except PowerPc) and some global files in {BASE} should now be compiled.
6. create tp link in {BASE}/DC/bin

   cd {BASE}/DC/bin

   ln –s {BASE}/DC/tpSource/tp

2-2 Detector Controller-PowerPc installation ({BASE}/DC/ppc)

The Detector Controller has two CPU's on a split backplane. There is an mv167 running under GEM5 that has the Detector Controller database loaded on it. The other CPU is a PowerPc which gets the data and sends it to the DHS. This section

discusses the PowerPc installation. The mv167 installation was discussed in section 2–1 .

1. Modify {**BASE}/DC/ppc/epicsGEM7.cshrc** and **{BASE}/DC/ppc/ppcInstall** for your local environment.
2. Source **{BASE}/DC/ppc/epicsGEM7.cshrc**. This sets up the environment for EPICS 3.13.4 GEM7.
3. Source the **{BASE}/DC/ppc/ppcInstall** script. This runs the GEM7 version of applSetup which sets up the EPICS UAE.
4. Run **gmake** to compile the code.

2-2.1 Transputer code Installation ({BASE}/DC/tpSource)

The process of compiling the Transputer code is described in "Installing the GNAAC Software" (20). Refer specifically to the **Installation Procedure** for any environmental changes that must be made, and next to **Building the GNAAC Software** for instructions on how to build the icon, sequencer and ucode software.

2-3 Wavefront Sensor Installation

The Wavefrons Sensor installation can be performed from the top level directory, or it can be performed locally in the WFS directory. Check 18) niri_hty_0006, "Near IR Imager (NIRI) and On-Instrument Wavefront Sensor (OIWFS) – Software Installation and Release Notes", Hubert Yamada, Steven Beard, Gemini 8m Telescopes Project. to become familiarized with the OIWFS.

1. Change directories to {BASE}
2. run nirs.env
3. Change directories to **{BASE}/WFS**
4. Run gmake - this should compile the whole system
5. make dependency lists in **NIRS_DIR, NIRS_LIBDIR, NIRS_ENGDIR, NIRS_WFSWFSDIR**
6. cd to the src directories in each of the four directories and type "**gmake depends**"

2-4 Set vxWorks startup

The four vxWorks machines need to have their startup parameters set. Have all the information listed below ready when the instrument is started for the first time.

1. vxWorks file
2. Startup file
3. IP address
4. name
5. Host IP address

6. Host name
7. gateway

- Detector Controller-EPICS
  1. vxWorks file - {BASE}/vxWorks/mv162/vxWorks
  2. Startup file – {BASE}/DC/bin/mv162/startup.gnirs.DCEPICS

- Detector Controller-PowerPc
  1. vxWorks file - {BASE}/DC/ppc/dc-data/vxWorks/vxWorks
  2. Startup file - {BASE}/DC/ppc/bin/ppc604/startup.gnirs.DCppc

- Components Controller/Instrument Sequencer
  1. vxWorks file - {BASE}/vxWorks/mv167/vxWorks
  2. Startup file - {BASE}/bin/mv167/startup.gnirs.CC

- Wavefront Sensor
  1. vxWorks file - {BASE}/vxWorks/mv167/vxWorks
  2. Startup file - {BASE}/WFS/WFS/bin/mv167/startup.gnirs

2-5 User Interface Startup

The Engineering screens and console windows are started by typing **nirsStart** in a terminal window. The script is located in **{BASE}/bin/solaris** If this is not in the path environment variable; use the whole path to invoke it. The console windows for the Wavefront Sensor, Components Controller, and Detector Controller are connected to the CPU's through the annex in the Detector Controller thermal enclosure. To connect to one of the thermal enclosure CPU's without using the startup script, type a command such as the following in an xterm or console window.

telnet [annex name] [port number]

port numbers

5006 Detector Controller data handler
5005 Detector Controller EPICS
5003[1] Wavefront Sensor
5004[1] Components Controller

[1] The ports 5003 and 5004 correspond to serial port 1 and serial port 2 on the back of the Detector Controller. The port numbers for the Wavefront sensor and Components controller may change if the serial ports are plugged into the opposite serial port.

3 Overview

3-1 Hardware

The Components controller thermal enclosure houses the hardware and software for running the components controller, instrument sequencer and Wavefront Sensor. There is a split VME backplane, which houses the Components Controller and Instrument Sequencer on one side, and the Wave Front Sensor on the other. The Instrument Sequencer software communicates with and controls all elements of the instrument including:

- Components Controller (CC)
- Wave Front Sensor (WFS)
- Detector Controller (DC).

The documentation for the Wave Front Sensor is in "Near IR Imager (NIRI) and On-Instrument Wavefront Sensor (OIWFS) - Software Installation and Release Notes "(18).

The Detector Controller is in a separate thermal enclosure and communicates with the other components (CC, IS) through its EPICS interface. Its documentation is included in the "Gemini-NOAO-Aladdin Array Controller (GNAAC) Software Maintenance Manual" (19).

3-2 Directory Structure Functional Descriptions

The base directory for GNIRS is a UAE that contains all of the software necessary to run the instrument. The four sections (IS, CC, DC, WFS) are located in subdirectories below the base directory. Each of these is also organized with a standard Gemini UAE.

3-3 Startup ({BASE}/startup)

The startup file which is loaded into the Components Controller CPU is located in **{BASE}/bin/{machine}/startup.IS**. This CPU is also where the Instrument Sequencer runs from. The precompiled version is in **{BASE}/startup/startup.IS.vws**. You must run gmake in the startup directory for changes to be transferred to the bin directory.

3-3.1 Instrument Sequencer ({BASE}/IS)

The Instrument Sequencer is based on the GMOS Instrument Sequencer; therefore, it functions and looks very similar. The main GNIRS APPLY and CAR records that the OCS communicates with are located in the Instrument Sequencer. All 'PRESET' and 'START' directives originate with this apply record, and all CAD responses funnel through the applyC CAR record in this database. Any parameters that need to be set in CAD records are set directly by the OCS or user, or indirectly by the Instrument Sequencer. Another major function of the Instrument Sequencer is to ensure that processes that could affect the observation are not started during an observation. The IS engineering interface contains a general overall control of the entire system. Sometimes it is

necessary to look at the lower level (CC, DC, WFS) engineering screens for more detailed information or more fine control of individual components.

3-3.2 Components Controller ({BASE}/CC)

The Components Controller runs all the mechanisms in the system, monitors and controls temperatures and monitors pressures.

3-3.2.1 Configuration

The components controller configuration files contain information to set up the motor, temperature and pressure software structures. The configuration files are located in the "{BASE}/CC/pv" directory
The low-level configuration files are listed below.

- gnirsConfig – low-level parameters for motors and sensors
- gnirsMechanisms – position and mechanism names for most of the mechanisms
- gnirsFilters – position and mechanism names for the filter wheels

The only file that should likely change is **gnirsFilters** when new filters are loaded. When this file is changed, the **cadVals.pv**, **fw1.lut**, and **fw2.lut** files should be changed to correspond with the new names. A gmake is required to copy the configuration files to the data directory. The configuration files are loaded when the function vmeInit is run which is a result of the init CAD record being processed. If parameters of any of the other mechanisms need to be changed, the **gnirsMechanisms** and **mechanisms.pv** files need to be modified. Then the same process of gmake and running the init to load the new values into the instrument.

3-3.2.2 Digital Input Output Card (xycom)

The DIO card in the CC VME rack reads and sets voltages. Voltages pertaining to the motor state and cryogenic coolers are connected to this card. Interrupts can be set on the DIO to alert the CPU when certain voltages change. This functionality is used to alert the CPU when the manual/computer switch for the cryo-coolers changes state.

3-3.2.3 Temperature card

The temperature card reads all the temperature diodes in the dewar except the ones used for temperature control of the bench and detectors.

3-3.2.4 Mechanisms

There are three types of mechanisms in the GNIRS dewar rotary, linear, and binary.

3-3.2.4.1 Rotary Mechanism

The rotary mechanism is the simplest of the three. There is a home switch for datuming, but no other feedback. The mechanism will find

the desired position in the direction that requires the least amount of motion. The filter wheels, grating turret, prism turret, focus, and camera are all examples of rotary mechanisms.

3-3.2.4.2 Linear Mechanism

The second type of mechanism is linear. This mechanism can run back and forth between a positive and negative limit. It also includes a home switch somewhere between for datuming. On each side of the travel there are two switches, a soft limit switch, and a hard limit switch. It is considered an error in the software to hit the soft limit, but the state of the datum is unaffected. However, when the hard limit switch is actuated, the datum state is set back to undatumed. This requires the user to datum this mechanism before any further motions. The slit and decker are linear mechanisms.

3-3.2.4.3 Binary Mechanism

The acquisition mirror and environmental cover are examples of binary mechanisms. This is a special case of a linear mechanism with only two defined positions. This mechanism has been implemented using only limit switches on either end. A separate home switch was not implemented. In stead, one of the soft limits function as a home for datuming. The datum is performed by moving the mechanism to one of the limits (defined in the configuration file gnirsMechanisms). The other named position is defined by the opposite soft limit switch.

3-3.2.5 Bench temperature control

The bench temperature control has no software interface.

3–3.3 Detector Controller

The Detector Controller takes the images and sends the data to the GEMINI Data Handling System.

3-3.3.1 Data Coadder (PPC)

The coadder is now controlled by a PowerPc (PPC). Some modifications of the NIRI coadder code were necessary in order to run it with the GNIRS Components Controller.

The major change in the software was adding the functionality of reading the aladdin3 chip to the system. In the previous version of the software there were 2 lines in the middle of the chip that weren't read out and two blank lines were in their place. This behavior was caused by wiring differences between the Aladdin2 and Aladdin3. The new Aladdin 3 version of the microcode now sends the same number of lines with the two blank lines in the middle, but instead of missing 2 lines in the middle, the first and last lines are dropped. This was preferable since very little science can be done with the edge of the chips. Once the image is

transferred into the PowerPc, the two blank lines in the middle are also removed which leaves a 1024 column by 1022 row image in the output. Code had to be added to 3 files to facilitate this change.

- outline.c
- gnCaptBufs.c
- gnTakeData.c

There are blocks of code that reference a detType variable with values of A2, A3_A, or A3_B.

- A2 – Aladdin 2 code
- A3_A – Aladdin 3 code
- A3_B – Aladdin 3 code. This was an experimental version that wasn't used.

The value of detType is taken from an EPICS variable, **{top}detType**. (top is the database prefix "nirs:dc" in this case) The value of the EPICS variable is loaded when the **gnAIII_1024xSU01_4u.cmd** file is parsed by the microcode download when an **Array Setup** is started. The previous NIRI microcodes can be run as long as detType is set to A2 in the corresponding .cmd file.

Another change was necessary to make the software more stable. A semaphore (mutex) was added in any routines that modified global variables related to the state of the capture buffers. All these changes were confined to the gnCaptBufs.c file.

The last major change was to add another semaphore (semDhs) to protect accesses to the DHS. There seemed to be some stability problems when multiple processes were trying to call DHS routines at the same time. These changes affected the file outline.c.

3-3.3.2 EPICS

The GNIRS Detector controller required some additional records and code to function properly and safely with the rest of the GNIRS system.

The first change was related to safety. A new function was added (setArrayActivation) which deactivated the array automatically when temperatures rise to a level where damage to the array is possible. A subroutine record was added to **activChk.sch** which calls a routine to deactivate the array when the temperature it monitors rises above a threshold value defined by MAX_ACTIVATION_TEMP (currently 70 K). There is also a global variable (activationFlag) which when set to one disables this functionality, allowing the array to be activated even when above 70 K. **Activating an array warm can damage the array**. Only experienced personnel should try running the array warm.

Other changes deal with the status of the detector control system. An EPICS record was added to the Detector Controller (masterEnable) which

is set to one after the startup file is completed and the system is ready to accept commands. This prevents the other systems from trying to modify values in the Detector Controller before it is ready. Another record (initDone) was added that is set to one when the initialization is complete. The last modification to the EPICS database was a record (alwaysC) that forces the applyC CAD record to go through a busy-idle transition even if no CAD records are marked. This was added in order to make the detector controller GEMINI compliant.

All of the .adl files in the {BASE}/DC/dl directory had the color palette changed from colors.dl to gmColors.dl so that all the dm screens throughout the system would use the same color palette and wouldn't cause an error when there were no more colors available. In fact all of the .adl files throughout the whole system had this change.

The last change was a bug-fix. It was noticed that after an error was set, and then cleared later on, the doObsSetup process status was never reset. A line was added to this code at the top of the while loop to set the status back to OK.

All of the Detector Controller changes can be updated in the NIRI Detector Controller software so that both systems can be run from the same code base.

3–3.4 Wavefront Sensor

The Wavefront sensor helps the Acquisition & Guiding system in pointing the telescope at a particular star.

3-3.4.1 Startup

There are two versions of the WFS software. The NIRI version assumes that the Components Controller, Instrument sequencer and Wavefront sensor are all on the same CPU. There is an alternate version, written for GNIRS, that has just the Wavefront sensor records. The GNIRS WFS startup file is {BASE}/WFS/WFS/bin/startup.gnirs. Refer to the NIRI/WFS software for more information on installing and running the WFS.

3-3.4.2 Changes

Several records were connected to the wrong prefix. Comparing the database to the dm screens, the database and/or the dm screens were synchronized.

3–4 Software Versions

There are several software packages necessary for compiling and running the GNIRS software. First of all, there are two versions of EPICS, one for the PowerPc, and one for the mv167's. This will probably be the state until the three

mv167's are replaced with PowerPc's. The mv167 is currently run with **EPICS – 3.12.2GEM5**. The PowerPc is run with **EPICS – 3.12.3GEM7.** There are also some general tools which are necessary to compile and run the system. These include:

dm – dm2.4
gcc – 2.95.2
dhs -.19
perl – 5.6
libstdc++ - 2.10
g++ 2.95.2

4 Solaris Environment

4-1 Setup

The UAE is setup by running nirsSetup. Refer to section 2 Installation.

4-2 Compile

Refer to section 2 Installation. Any changes to code, configuration files or scripts will not take effect until a gmake is done which compile and copy the affected files to either the bin or data directory.

4-3 Graphical User Interface

The Gemini system uses dm (version 2.4) to display the engineering screens. The color palettes in all of the .adl files refer to the same file gmColors.dl.

4-3.1 User Interface Startup

Refer to section2–5 for instructions on starting the user interface.

4-4 Reset vxWorks machines

If the vxWorks system locks up and doesn't respond to a control-x at the telnet window, it can be rebooted from a standard solaris command line. There is a program called gnReset which is located in **{BASE}/DC/bin/solaris**. This program opens a socket to the annex in the detector controller and sends a command to the reset module in the Detector Controller or Components Controller. This function has two parameters. The first parameter is the Internet name of the annex. The second parameter tells the program which CPU to reboot. There are 4 possibilities: **dccoadd, dcepics, dcboth wfs, cc.**

ex. gnReset bacon dcboth

5 Interfaces

This section describes the software interfaces between different components of the system.

5–1 Consoles

The console windows are where the vxWorks machines display their output and command line input is accepted. There are four in the system: Components Controller/ Instrument Sequencer, Detector Controller Epics, Detector Controller Coadder, and Wavefront Sensor. Each has its own purpose in the system. If there is an error related to any one of these in EPICS, the console windows may contain further information relating to the error or to the current state. The consoles for each of the vxWorks machines in GNIRS are connected to the remote annex box. Refer to 2–5 **User Interface Startup** for instructions on starting the console windows.

5–2 epicsServer Socket Interface

The Detector Controller-PowerPc uses this socket for reading and writing records to the EPICS database. Since there is no EPICS database loaded on the PowerPc, Channel access could not be used. A client server model was used to bridge this gap. The epicsServer process starts the client process on the PowerPc. This process opens a socket to the server on the DC EPICS machine and then waits for messages on the sendQ. When it receives one, it sends the message across the socket and waits for a response from the EPICS machine over the socket. When it receives the response, it sends it back to the calling process on the responseQ. Access to the queues is protected by a semaphore, so only one process can send a message at a time. If the socket connection is broken, the client task tries to reestablish the connection.

5–3 Epics/OCS Interface

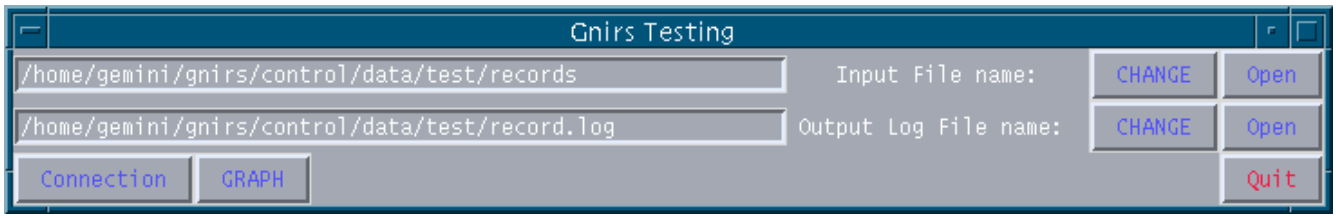
EPICS uses channel access to send values between clients and databases. Much of the communication in the GNIRS system uses EPICS. Check the "Channel Access Reference Manual" for instructions on setting up the EPICS environment in order to use channel access on a network.

5–4 Data Handling System Interface (DHS)

The DHS uses a client server model to send messages and data. There are client libraries available for facilitating access to the DHS. These libraries are described in 21) and 22).

5–5 Temp log

There is a python program that runs on the sun that will log EPICS variables graphically and to a file. The python script is in the {BASE}/testing directory. Change to that directory and type "**python logger.py".** The window below is displayed.



The input file name is the file that lists all the records that should be read in. Change the name in the text area on this screen, or click on the **change** button to open up a file dialog box to pick a file from. Once the name is selected, click on the **open** button to read in the values. The format of the file is listed below.

nirs:sad:cc:benchTempPoint,0,300,
nirs:sad:cc:benchUnderside,0,300,

- The first value on the left is the full epics name of the value to log.
- The second field is the minimum value to show on the graph
- The last value is the largest value to show on the graph.
- The trailing comma is necessary for the parser.

Next, set the filename for the output file. Click the **open** button to open and write header information to the file. The next process is connecting the to the server. There is a server program running on the Components controller named **test**. It is located in {BASE}/testing in the file pTest.c. The compiled version, pTest.o is located in {BASE}/bin/mv167. The **Connection** button on the window will display a dialog box with the Ethernet address and two port numbers. Set the Ethernet address to the Ethernet address of the Components Controller. Changes in the port numbers must be reflected back in logger.py. After things are connected and open you are free to create graphs. Click the **graph** button and select a variable from the list. The graph should display and will be continuously updated while client and server are running.

5-6 Hardware

5-6.1 Coadder

Communication with the coadder is across the VME backplane. The memory address of the coadder registers is stored in oSystem->pDCARegs. The coadder uses interrupts to inform the vxWorks machine of important states. The three states are:

- Coadd done –received frame
- Unscramble done – all frames for this exposure are coadded and unscrambled
- Transfer done – transfer to vxWorks completed

The interrupt routines for these are in **{BASE}/DC/ppc/dc-data/noaoCoadder/intRoutine.c.** The function gnDCAStart initializes the data structures for use with the coadder the functions: setRegisters,

loadVarRegs, and dcaLoadRegister can be used to modify values in the coadder. These are located in the noaoCoadder directory listed above.

5-6.2 Temperature Control (Detector)

The detector temperature is controlled by an NOAO produced card. The card is located in the Detector Controller VME backplane. Commands are sent to it through a serial port which is connected to the Detector Controller-EPICS vxWorks machine. The code for this is located in **{BASE}/DC/tempCard**.

5-6.3 SenTorr Pressure monitor

The SenTorr pressure monitor box is located in the Components Controller thermal enclosure in crate 4. Messages are sent to the SenTorr by a serial port connected to the Components Controller vxWorks machine. The current pressure is read by the **scanStatusTask** process which reads the pressure every 30 seconds. The routines related to the SenTorr pressure monitor are located in **{BASE}/CC/sys/hdwrControl**.

5-6.4 Digital Input Output Board (xycom)

The DIO board is also addressed across the Components Controller VME backplane. The dio_xy240 struct is initially set to the XYCOM_BASE address. Several voltages are connected to the DIO. The most notable of these are:

- Cryogenic head switch state (computer mode, manual mode)(input)
- States of the mechanisms (each mechanism has one of the following)
- Driver enable (output)
- Driver reset (output)
- Driver enabled (input)
- Driver fault (input)

The DIO board has the ability to send interrupts when the state of a voltage changes. The function **xycomSetInterrupt** is used to set an interrupt handler for a particular voltage.

5-6.5 Temperature monitor

There are two temperature monitor boards in the Components Controller. They are set at the addresses **TEMP_P_ADDR** and **TEMP_D_ADDR** for the primary and secondary respectively. The structure **tempMonReg** defines the registers in the cards.

```
struct tempMonReg {
   char pad1;
   unsigned char muxadd;   /* channel number to read*/
   char pad2;
   unsigned char adcstr; /* tells temp card to read temp*/
```

```
        unsigned short adcrd; /* data is stored here from last read*/
        unsigned short extcon;
    } tempMonReg;
```

The function rddewint is an example of how the temperature cards are read.

5-6.6 Oregon Micro Systems motor controller (OMS)

The OMS card also uses VME memory addressing for communicating with the vxWorks machine. The system is set up to use one or more OMS cards, but the addresses must be **GNIRS_MD_ADDR_BASE** for the first card, and each successive card would be the previous card's base address + **GNIRS_MD_ADDR_INCR**. Both of these values are defined in the **{BASE}/CC/include/gnirsCC.h**. The structure **mdRegister,** also defined in the same header file, shows how the OMS registers are laid out. Check the OMS manual for information on programming the card.

6 Components

6-1 Instrument Sequencer

The GNIRS Instrument Sequencer is a modified version of the Instrument Sequencer from GMOS. Changes were made to facilitate connecting to the other subsystems.

6-1.1 Top-level control of instrument.

6-1.1.1 Startup

The startup file is {**BASE}/bin/mv167/startup.IS**. This startup file loads all the files necessary for running the Instrument Sequencer and the Components Controller.

6-1.1.2 Configuration

Several configuration files, located in **{BASE}/IS/pv**, are used to set up EPICS variables. These have .lut and .pv extensions. Others exist in the **{BASE}/CC/pv** directory for the components controller part of the software.

6-1.2 Apply interface to OCS

The OCS uses the GNIRS **apply** and **cad** records in the system to communicate. It will set individual cad record fields throughout the instrument and start them with the **apply** record located in the Instrument sequencer.

6-1.3 Connection to DC, CC, WFS

The instrument sequencer is connected to all of the other subsystems in the instrument through the **apply** and **applyC** records. Channel access links send commands to the apply and cad records in the three subsystems.

6-2 Components Controller

The components controller houses all the electronics to move motors, control and read temperature, and read pressure. Motor control is handled by the CPU, which sends strings to an OMS controller. The OMS controller then sends pulses to a phytron motor driver, which sends impulses directly to the motor. Two NOAO designed vme cards handle temperature reading. A command is sent across the vme back plane to read a particular channel. The correct multiplexers are set, and an a/d chip reads the voltage. The SenTorr box Reads the pressure and reports its values to the CPU through a serial port. EPICS and the OCS can control all of the hardware. But there is a completely functional command line interface, which can be used for debugging and testing.

6–2.1 Low level control

This level is completely command line driven. It is the most basic level at which the whole system can be controlled. A modified startup file can be loaded which doesn't load any of the epics software or databases to help find any hardware or software problems.

6-2.1.1 Startup

The standard startup file is **startup.IS** located in {BASE}/bin/mv167. The editable version is in {BASE}/IS/startup. It contains many EPICS related files and function calls. There are two important files loaded in the low level, ccGlobal.o and hdwrControl.o. All the low level functions for running the system are contained in these two files

6-2.1.2 Hardware Configuration and Initialization

All low level initializations are executed from the function vmeInit. This can be run from the command line, or from the EPICS init CAD routine. This routine initializes and tests all the hardware and loads all the low level configuration files.

There are several functions and global variables that can be used to set the state of the system.

GLOBALS

1. **sendToMotorDebug** - prints motor information to the screen when equal to one
2. **noXYCOM** – if equal to one then there is no communication with DIO board
3. **noTempMon** – if equal to one then there is no communication with temperature monitor board
4. **noSenTorr** – if equal to one then there is no communication with pressure monitor
5. **configDirectory** – the directory where **gnirsConfig, gnirsMechanisms, and gnirsFilters** are located

FUNCTIONS

setSimulation – sets the Components Controller lower level simulation level. If set to non zero, no external hardware will be accessed.

6-2.1.2.1 gnirsConfig

All the mechanism configuration information is contained in this file except the position names. The table below shows the keywords and their parameters.

Table 1 - gnirsConfig headings

Keyword name	Type	Description
Name	string	Name of mechanism
Type	string	type of mechanism {ROTARY, BINARY, LINEAR}
Controller	integer	number of OMS card connected to
Axis	integer	axis on OMS card {0=X, 1=Y, 2=Z, 3=T}
homeLevel	string	home signal is Low or High {"L","H"}
fullTravel	integer	total steps from end to end of travel, or one complete revolution
Backlash	integer	Number of steps to remove backlash
Seek	integer list	maximum acceleration and velocity of mechanism
Probe	integer list	acceleration and velocity when searching for limits or home
probeH	integer list	acceleration and velocity
Backoff	integer list	acceleration and velocity in final positioning
home2	integer list	list of home switch parameters
posLimit	integer list	step position of positive soft limit and distance between soft and hard limits on positive side
negLimit	integer list	step position of negative soft limit and distance between soft and hard limits on negative side
Enable	integer list	location of enable line on DIO and initial value
Reset	integer list	location of reset line on DIO and initial value
Isenabled	integer list	location of isenabled line on DIO and initial value
Fault	integer list	location of fault line on DIO and initial value
initString		

This **gnirsConfig** file also includes linear offset calibration data for the temperature sensors inside the dewar.

6-2.1.2.2 *gnirsMechanisms*

This **gnirsMechanisms** contains information on how to run the motor. This includes what type of mechanism it is, what type of limits there are and where they are in relation to the home, and where hardware connections from the mechanism are connected in the Digital Input Output board (DIO).

[2] The numbers from left to right are: home position, auxiliary home position, DIO port connection, DIO bit connection, auxiliary in use, and the type of home switch {0 for Normal, 1 for positive limit}.

Table 2 gnirsMechanisms headings

Name	Type	Description
Name	string	name of mechanism
Position	list	{position name (string), step position (integer), focus shift (integer)}
Motor	integer	motor number (should match gnirsConfig
Mechanism		new mechanism marker

6-2.1.2.3 gnirsFilters

This file is like gnirsMechanisms except it applies only to the filter wheels. This file will change when filters are changed. One important note is that the files fw1.lut, fw2.lut and cadVals.pv must correspond to the gnirsFilters file or there will be an error when the affected position is selected.

6-2.1.3 Motor Control

The low-level motor control interface is only from the command line. These functions are used when problems are encountered or when the user wants to bypass some of the standard checking that the software performs. Below is a list of useful functions and how they are used. The parameters are listed in parentheses.

6-2.1.3.1 Motion

position (int motor number, int step position) – This function moves the specified motor to the absolute position given.

checkLimit (int motor number, int direction) - checks to see if mechanism is at specified limit.

motorOn (int motor number) – This function turns the power on for a specified motor. This function is called directly by most of the motion control functions. The only ones that need it to be called prior to evocation are cmd and cmdR.

motorOff (int motor number) - This function turns the power off for a specified motor. This function is called directly by most of the motion control functions. The only ones that need it to be called are *cmd* and *cmdR* after these functions have completed.

creep (int motor number, int switch number, int n) – This function positions the mechanism n steps from the specified switch and moves one step at a time until the switch is actuated. This function uses the same enumeration as *findLimit*.

datum (int motor number) – finds the position of the home switch and sets it to the value specified in gnirsConfig.

findLimit () -- This function moves the specified motor to one of the limits. The limit is specified by:

enum {HOME_SW = 0, POS_LIM, NEG_LIM, HARD_POS_LIM, HARD_NEG_LIM}

abortMotor (int motor number) – Abort motion by a motor.

park (int motor number) – Place a mechanism in the park position.

cmd (int motor number) – This command starts up a shell that sends commands to the OMS cards. The motor number is used to determine which OMS card and which channel on that card to send the commands to. Refer to the OMS user manual for command language assistance. (Note. The user must turn motor on with motorOn prior to use. Escape exits the *cmd* shell. Use motorOff to turn motor off after *cmd* is exited.)

cmdR (int motor number) – This sends a single OMS command to the motor card. . (Note. The user must turn motor on with motorOn prior to use. Use motorOff to turn motor off after *cmdR* returns.)

6-2.1.3.2 Status

printMotorSwitch (int motor number) – Print data in motor structure.

testOMSCards () – Send a message to OMS cards to ensure that they are responding.

6-2.1.4 Temperature read back

The GNIRS dewar has up to 5 7 inputs for temperature monitoring. Below is a list of some of the diagnostic routines that can be run from the command line.

testTempBoards () – Read diagnostic values from temperature boards and check against known values

rddewVotl (char channel number) – read temperature diode and return the voltage.

rddewDegK (char channel number) - read temperature diode and return the temperature in degrees Kelvin.

rddewInt (char channel number) - read temperature diode and return the value in a/d counts.

rddewDegC (char channel number) - read temperature diode and return the value in degrees centigrade.

printTempVolts () – Print all temperature diode voltages.

printTemp (unsigned char channel number) - Print all voltage and temperature information for the specified channel.

rdTemp (unsigned char channel number) – read a channel several times in a row to test stability.

6-2.1.5 Pressure read back

There is a SenTorr box in the components controller thermal enclosure. It is capable of reading two thermocouple gauges and one cold cathode gauge. The SenTorr box is connected to the CC computer through its serial port. The software is in place to read all the sensors, but as of this writing, only the two thermocouples were going to be used. If a cold cathode is added, the function readCCG should be called from scanStatusTask where readSenTorr is called. Below is a list of useful functions for communicating with the SenTorr box.

initSenTorr () – Initializes communication with the SenTorr.
testSenTorr () - Send various commands to the SenTorr and check response.
printPressure () – Reads the SenTorr thermocouples and prints the pressure to the screen.

6-2.1.6 Digital Input Output board

This board is used to read and set voltages. It is used to read the state of the motor drivers. It is also used to control and monitor the state of the cryogenic cooling heads. Below is a list of functions for communicating with the DIO board.

bitHigh (int port number, int bit number) – Set bit connected to port and bit high.

bitLow (int port number, int bit number) – Set bit connected to port and bit low.

xycomReport () – Print all information regarding DIO board.

6-2.1.6.1 Cryogenic cooling

The cryogenic cooling heads can be controlled manually or in software. The components controller thermal enclosure has two switches for operation of the cryogenic coolers. One switch selects either computer or manual. The other switch selects between on and off when the first switch is in manual control. The software shows the current state for the switches, and also has a function that allows the computer to control the state of the heads. These functions can be accessed in cryoCad.dl (in components controller engineering screens) or tempCtrl.dl (accessible from the "Temperature Control" button in the instrument sequencer).

6-2.1.7 System Information

The function *sysInfo* can display any or all of the available status information from the lower level components control. The function has a single parameter, which is parsed to display specific information. The first digit selects the type of information you are looking for, and the rest, if needed will pick a specific element.

Table 3 sysInfo first digit definitions

Digit	Information printed
1	pressure
2	voltage reading from temp sensors
3	motor status
4	dumpMech
5	dumpFilters
6	cryo head state
7	grating information
8	DIO card information

With a first digit of 3 or 4, a motor number can be appended in order to get information for just one motor.

Table 4 Motor numbers and names

Motor number	Motor name
0	Environmental cover
1	Filter wheel 1
2	Filter wheel 2
3	Slit slide
4	Decker slide
5	Acquisition mirror
6	Prism turret
7	Grating turret
8	Camera turret
9	Focus

Table 5 sysinfo Special Cases

Number	Definition
0	Print everything
n99	Print information up to n (n is a digit)

6-2.2 High level control (EPICS)

The OCS will ultimately control the GNIRS system. Until it is finished, the system will be run by using the GNIRS engineering screens.

6-2.2.1 Startup

The startup script loads all of the low level files, and several DHS and EPICS related files. After loading all the files, the EPICS database and DHS client are started. The last major function, *initTasks*, starts all the background processes needed by the CC software.

6-2.2.2 Configuration

The startup file handles the initial configuration. However, changes can be made any time the system is not busy. The pvload function handles most of the EPICS configuration. It loads two files, cadVals.pv and gmSeq.pv. There are a few records that are set directly from the startup file. These are done here mostly because of timing issues. For example, we want the apply for the whole system to be performed only after all other processes are ready.

loading config files (vmeInit) TBD

6-2.2.3 Cad Records

Cad records are used to start all of the commands received from the OCS. If an error occurs in processing the CAD record returns an error message. If an error occurs after the cad processing is over, a CAR record handles the error. The CAD records are designed to check parameters and then start processing the command, or start an external process, when a START directive is received. The external process will change the CAR record back to IDLE or ERROR when it is finished.

6-2.2.3.1 Motor Control

There are four basic functions that the motor can do: datum, park, move to a defined position, and move to step position. Each motor has a cad record for each one of these functions. Datum and park have no parameters and the two move commands have one parameter each. The grating is the one exception; it has 3 parameters. Each motor has one CAR record for all four functions. If the motor is busy, another CAD connected to that mechanism will be prevented from starting.

6-2.2.3.2 Cryo cooler control

The cryo CAD record receives an ON/OFF message from a subroutine record which input from the engineering screen and the hardware itself. The engineering screen has a simple on/off button and the hardware has two switches of importance. The first is a computer/manual switch. This would happen if the CPU was being rebooted, or if debugging was occurring. The computer mode would be useful to test the effect on the image. The second switch is an on/off switch. This would only have effect if the first switch were in manual mode. When the hardware switch is in manual mode, the software has no effect on the state of cryogenic coolers. The hardware switches are located in the Components Controller on the front panel.

6-2.2.3.3 init

The init Cad routine initializes the hardware and software. The hardware and software simulation modes are setup first. Next, vmeInit is run which is described in 6-2.1.2 **Hardware Configuration and Initialization**. When the components controller configuration files are modified, the init CAD routine can be started to load them into the computer.

6-2.2.3.4 debug

The EPICS debugging level can be set from the debug CAD. This allows different amounts of debugging information to be displayed on the screen.

6-2.2.3.5 reboot

This command sends a reboot command to the Components Controller computer.

6-2.2.4 Communication

The communications, between CPU's, are handled through EPICS. The hardware connections are all across the VME back plane except for the connection to the SenTorr, which is a serial connection. The console for the WFS and CC are connected to the annex in the Detector Controller. Connections are described in 2–5 **User Interface Startup**.

6-2.2.5 EPICS Interface

The engineering screens contain all the information necessary to run the GNIRS instrument. The status of the system is kept in the SAD database and also displayed on the engineering screens.

6-2.2.6 Status and information

The SAD records store information about the current state of the system. Many of the SAD records have alarms that will trigger when data goes out of the acceptable range. The ranges are set in the fields of the SAD records themselves and can be changed with xschedit or runtime through the use of a tool like pvload.

6-2.2.6.1 Pressure

The pressure is monitored through the SenTorr electronics. A serial port is connected to the Components Controller CPU. The pressure data is read from the *scanStatusTask* function every 15 seconds. The pressure is loaded into a global array where the SAD records have access to it.

6-2.2.6.2 Motor data

The EPICS SAD database contains information about the current motor state. It has the position name, step position, if it is parked or datumed, the state, and health for all the motors. In the case of the grating, it also shows the current wavelength, order, and tilt associated with the current positon. Refer to the gnirsConfig, gnirsFilters, and gnirsMechanisms configuration files for other parameters related to mechanism operation.

6-2.2.6.3 Temperatures

The temperatures are read with two NOAO supplied temperature cards. These cards reside in the CC side of the VME back plane. The temperature is read by first writing to a memory address that selects the multiplexer address. After a short time, the address that starts an ADC conversion is written to. After another delay, the answer is read back from another memory address. The locations of all these is described by the *tempMonReg* and either **TEMP_P_ADDR** for the

primary card, or **TEMP_D_ADDR** for the secondary (or diagnostic) card. The temperatures are then written to a global array that the SAD database can read values from.

6-2.2.6.4 Health\State

The system health and state are set in the lower level and are propagated up to EPICS. Health is affected by initialization failures, errors in motor motions, or if temperatures or pressures are out of range. The state describes what the instrument is currently doing. There are mechanism level health and state variables as well as global. The overall health takes on the worst case of all the lower level health values.

6–2.3 Common changes

6-2.3.1 Filters

The filter position names can be changed by simply modifying configuration files and reloading them. **gnirsFilters, fw1.lut, fw2.lut and cadVals.pv** store parameters related to the filters. They are located in the **CC/pv** directory. After modifications, perform a gmake in that directory, and then start the init CAD function. The filter position names in all these files must match in order for the system to function properly.

6-2.3.2 Mechanism position names

The process is similar to change the mechanism position names. Start by modifying the **gnirsMechanisms** file. This file is loaded by the lower level code and contains all the names and location of each distinct position of the mechanisms. These names must match the names located in the **mechanisms.pv** file. If they don't, the lower level won't understand where to move for a given position. A start on the **init** CAD will load both files into the instrument.

7 Troubleshooting

7–1 Error Messages

The error messages below are most of the messages that you might see in the console windows or in EPICS records message or error fields.

"*DatmCad: * is busy\n" (*Cad.c/*DatmCad) mechanism is currently busy wait for idle

"*DatmCad: * is not busy\n" (*Cad.c/*datmCad) Mechanism must be busy in order to stop

*qDatmCad: Unrecognized directive" (*Cad.c/*DatmCad) Not a valid EPICS directive(CAD_START, CAD_STOP, CAD_MARK, CAD_PRESET, CAD_CLEAR)

,"* mechanism not datumed\n" (*Cad.c/*ParkCad) Must be datumed before motion

"Bad format in <%s>" (mech.c/setFilters) error reading config file

"Bad index in <%s>(mech.c/setFilters) error reading config file

"Both filters in same wheel <%s, %s>" (mech.c/finishFiltersConfig) filter configuration error

"C%d: no room for command string" (motors.c/sendToMotorDriver)

"C%d ID error -- got <%s>" (motors.c/testOMSCards) Bad response received from OMS card.

"Call goToPos() when not datumed" (motors.c/goToPos)

"Calling BiRead on output port %d, mask %x" (digital.c/xycomBiRead) Reading wrong port

"Calling BoRead on input port %d mask %x" (digital.c/xycomBoRead) Reading wrong port

"Calling BoWrite on input port %d, mask %x, val %x" (digital.c/xycomBoWrite) writing to wrong port

"Calling 'testXYCOM' before init" (digital.c/testXYCOM) initXycom must be run first.

"Cannot find mechanism in %s\n" util.c/readConfig Mechanism name error in config file.

"Cannot open '%s' (util.c/readConfig) There was an error opening a configuration file.

"Cannot set wavelength for mirror" (commands.c/validGratingWavelength, validPreferredWavelength) Mirror doesn't support getting position from wavelength and order

"Can't get controller %d's semaphore" (motors.c/sendToMotorDriver)

"Can't find <%s>" (util.c/doPosition) Position name isn't valid.

"Ch %d err: gnd reads %6.3f" (tandp.c/testTempBoards) Ground voltage is too high on temperature monitor card.

"Ch %d err: 1V reads %6.3f" (tandp.c/testTempBoards) 1 volt reference is out of specifications on temperature monitor card.

"Ch %d err: 100k reads %6.3f" (tandp.c/testTempBoards) 100k ohm reference is out of specifications on temperature monitor card.

"Controller %d has not passed INIT stage" (intfc.c/motorDriverInit) Problem with motor driver

"Controller %d initial status fault 0x%x" (intfc.c/motorDriverInit) Problem with motor driver

"Could not create position node" (mech.c/nodeLookp) Couldn't allocate memory

"Couldn't lock keypad" (tandp.c/readCCG) Sentorr keypad lock failed

"dbGet error = %ld" (epCommon.c/getDbInfo) Error getting epics variable

"dbNameToAddr failed for %s.\n" (epCommon.c/putDbInfo) error writing to epics variable,

"dbPutField failed\n" (epCommon.c/putDbInfo)error writing to epics variable

"error in *Ctrl msgReceive\n" (*Cad.c,*Ctrl) This should never happen. Try reboot.

"Error in readConfig(%s)\n" (util.c/readConfig) There was an error reading a configuration file. Probably a formatting error in the file.

"Failed to convert rhs <%s>" (util.c/parseKeywordValue) An error occurred reading configuration file.

"Failed to send 'WY' to C%d" (motors.c/testOMSCards) Error sending command to OMS card

" 'filter' position unknown" (commands.c/valid)

"Illegal device %d in 'valid()'" (commands.c/valid)

"Illegal grating number in %s" (util.c/readConfig) grating numbers must be between 0 and NUM_GRATINGS

"Illegal motor number in %s" (util.c/readConfig) motor numbers must be between 0 and NUM_MOTORS

"Illegal motor (mech %d) in 'valid()'" (commands.c/valid) motor numbers must be between 0 and NUM_MOTORS

"Illegal temperature sensor in %s" (util.c/readConfig) Temperature sensors are numbered 0 - NUM_TEMPS

"Invalid lambda/order (%f/%d), gdp->a = %f" (commands.c/validGratingWavelength)

"Invalid position for linear mechanism <%s>" (util.c/doPosition) Position should be between positive and negative limits

"Invalid position for rotary mechanism <%s>" (util.c/doPosition) Position should be in the first rotation of travel of a rotary mechanism.

"Invalid semaphore in sendToMotor for C%d" (motors.c/sendToMotorDriver) Reinitialize or reboot

"Missing '>' from SenTorr" (tandp.c/readPort) There was probably something wrong with the message sent to the senTorr.

"msgQSend error in *DatmCad\n" (*Cad.c/*DatmCad) this should never happen try reboot of CC

"No '=' in %s" (util.c/readConfig) Equal sign between parameter and value

"No ']' in section header: %s" (util.c/readConfig) There was an opening bracket but no closing bracket.

"no response from SenTorr" (tandp.c/readPort) Check senTorr power and serial port connection, reinitialize and reboot if necessary.

"No semaphore for SenTorr" (tandp.c/initSenTorr) Error creating semaphore, reboot Components Controller.

"No semaphore for Temperatures" (tandp.c/initTempBoards) Error creating semaphore, reboot Components Controller.

"No semaphore for XYCOM" (digital.c/initXycom) memory error reboot components controller

"Opening SenTorr port failed" (tandp.c/initSenTorr) Couldn't open serial port to sentorr. Check connection.

"pressure values are different" (tandp.c/readCCG) Thermocouples read different

" rddewint address %d out of range\n" (tandp.c/rddewint) temperature channel must be less than NUM_TEMPS.

"redDegC: address %d out of range\n" (tandp.c/rddewDegC) temperature channel must be less than NUM_TEMPS.

"redDegK: address %d out of range\n" (tandp.c/rddewDegK) temperature channel must be less than NUM_TEMPS.

"rdVolt address %d out of range\n" (tandp.c/rddewVolt) temperature channel must be less than NUM_TEMPS.

"readConfigfailed to find keyword in %s" (util.c/parseKeywordValue) An error was encounteted while reading config file.

"Ring buffer for C%d too small" (motors.c/sendToMotorDriver) Can't place motor message on queue.

"Section '%s' not recognized" (util.c/readConfig) valid sections are: [grating], [cryo], [temperature], [filters], [mechanism] and [motor] .

"Section header must be first: file %s" (util.c/readConfig) valis section headers must be first. [grating], [cryo], [temperature], [filters], [mechanism] and [motor] .

"semTake error in rddewint" (tandp.c/rddewint) temperature semaphore error. Reboot Components Controller.

"semTake error in xycomBoWrite" (digital.c/xycomBoWrite)

"SenTorr IG error %s" (tandp.c/readSenTorr) Error returned by sentorr

"SenTorr msg too long <%s>" (tandp.c/sendSenTorr) Shouldn't happen.

"SenTorr read gave error" (tandp.c/readPort) There was probably something wrong in the message sent to the senTorr.

"SenTorr Semaphore failed" (tandp.c/sendSenTorr) Semaphore error for pressure semaphore. Reinitialize.

"SenTorr write failure" (tandp.c/sendSenTorr) shouldn't happpen.

"Setting baud rate for SenTorr PORT failed" (tandp.c/initSentorr) Shouldn't happen.

"string expected in <%s>" (util.c/readConfig) An error occurred while reading mechanism position from configuration file.

"%d system%sfailed hardware test" (intfc.c/testHardware)

"Timeout in sendToMotor for C%d" (motors.c/sendToMotorDriver) Never received response from motor driver. Reinitialize

"Transmit buffer not empty in controller %d" (intfc.c/motorDriverInit) Problem with motor driver

"Unable to create sent ring buffer for md %d." (intfc.c/motorDriverInit) Memory error, reboot components controller

"Unable to create ring buffer for md %d." (intfc.c/motorDriverInit) Memory error, reboot components controller

"Unable to create semDone semaphore for md %d/%d. (intfc.c/motorDriverInit) Memory error, reboot components controller

"Unable to create semResponse for md %d." (intfc.c/motorDriverInit) Memory error, reboot components controller

"Unable to create semOut for md %d." (intfc.c/motorDriverInit) Memory error, reboot components controller

"Unable to enable GNIRS_INTERRUPT." (intfc.c/vmeInit) Memory error, reboot components controller

"Unable to install Motor interrupt handler." (intfc.c/vmeInit) Memory error reboot Components Controller.

"Unable to install XYCOM interrupt handler." (intfc.c/vmeInit) Memory error, reboot components controller

"Unable to spawn scan status task." (intfc.c/vmeInit) Memory error reboot Components Controller.

"Unexpected character 0x%x in controller %d" (intfc.c/motorDriverInit) Problem with motor driver.

"Unknown filter <%s>" (mech.c/finishFiltersConfig) Filter configuration error

"unknown operation in *Ctrl\n" (*Cad.c/*Ctrl) valid operations are STEPS, POS, PARK, DATUM

" %s, %s Unknown position " (commands.c/valid) input position doesn't exist in config files

"\n***\nvmeBoard Init Failed\n***\n" (intfc.c/gnirsInit) One of the vme boards failed initialization.

"wavelength %f outside bounds" (commands.c/validPreferredWavelength)

"XYCOM ID wrong -- got <%s>" (digital.c/textXYCOM) Incorrect return string from xycom.