Particle Physics and Astronomy

Research Council

Isaac Newton Group

A Developer's Guide to the NAOMI Observer Software

Version : 1.1

Craige Bevil

Document Identifier : WHT−NAOMI−17

7 August 2002

Isaac Newton Group,
Apartado 321, 38780 S/C La Palma,
Tenerife, Canary Islands

Telephone +34 922 425400
Fax +34 922 425401
Internet cb@ing.iac.es

Table of Contents

Chapter 1. Overview

The purpose of this document is to provide the developer with an overview of the NAOMI top level observing software. This will include a description of the development tree, the tools used in the development of the system, a guide to how to build and install the system and a summary of the architecture of the software.

The actual functionality of the software included in the system is subject to document WHT−NAOMI−15.

Chapter 2. Third Party Development Software

The software provided for the NAOMI observing level software relies heavily on a number of third party software packages. This chapter outlines the third party software that has been used throughout the system.

2.1 TCL/Tk

The majority of the applications associated with the NAOMI observer level software are coded in TCL and Tk. The current version of the TCL/Tk being used by the system is version 8.0.3 and can be located in the */opt/csg* directory tree. The scripts are portable and have been tested upon both Linux and Solaris platforms.

There is a dependancy on version 8.0.3 by the ***incr*** package which is described below. Upgrading TCL/Tk should only be done in conjunction with an upgrade of ***incr***.

2.2 Incr

Incr is a third party package which extends standard TCL/Tk by adding object orientated extensions into the language. The software provided with the NAOMI observer level interface is object orientated based. Much of the software has been generalised and reduced to classes to improve both maintenance and reusability.

The version of ***incr*** currently installed is version 3.0 which is the latest major release of the software. The package is dependant on version 8.0.3 of TCL/Tk. The software is located in the */opt/csg* directory tree.

A later version of ***incr*** is slated for release sometime soon which is compatible with the later releases of TCL/Tk.

2.3 BLT

This is a TCL extension developed by Bell Laboratories in the United States. It provides a number of new widgets which can be used from TCL/Tk and is used most notably by the performance seeing and FOV monitor classes to display the seeing trace and the FOV monitor.

The current version of the software that is installed is version 2.4u and can be located in the */opt/csg* directories.

2.4 DRAMA

The standard message passing protocol adopted by the ING. The packet collection task has a DRAMA interface so that it may be driven by the UltraDAS system in order to acquire packets that it has been configured to collect.

The packet collection task currently uses DRAMA version 1.2b2 of the Dtcl package which is located in the */opt/csg/lib* directory tree.

2.5 EPICS

EPICS is used by the packet collection task in order to acquire packet information from EPICS based embedded subsystems. In the case of the NAOMI observing software, the packet collection task uses EPICS to retrieve header packets from INGRID.

The packet collection task uses version R3.12.2 of EPICS which is located in the directory */opt/INGsoft/epics/R3.12.2*.

2.6 Python

Python is used extensively throughout the embedded NAOMI system provided by Durham and is their scripting language of choice. The API provided by Durham for remote client access to the NAOMI sequencer and the Electra Process Monitor is coded in Python. Subsequently the EPMDataServer is also coded in Python. Python is not used anywhere else within the NAOMI observing level software.

The current version of Python being used is version 1.5.2 and can be located in the */opt/csg* directory tree.

2.7 Man2html

This is a utility which is used in the build process and is used to convert the manual pages that have been written for each of the applications into HTML. The HTML files are subsequently included into the online documentation provided as part of document *WHT−NAOMI−15*.

The application can be found in the **/opt/csg/bin** directories.

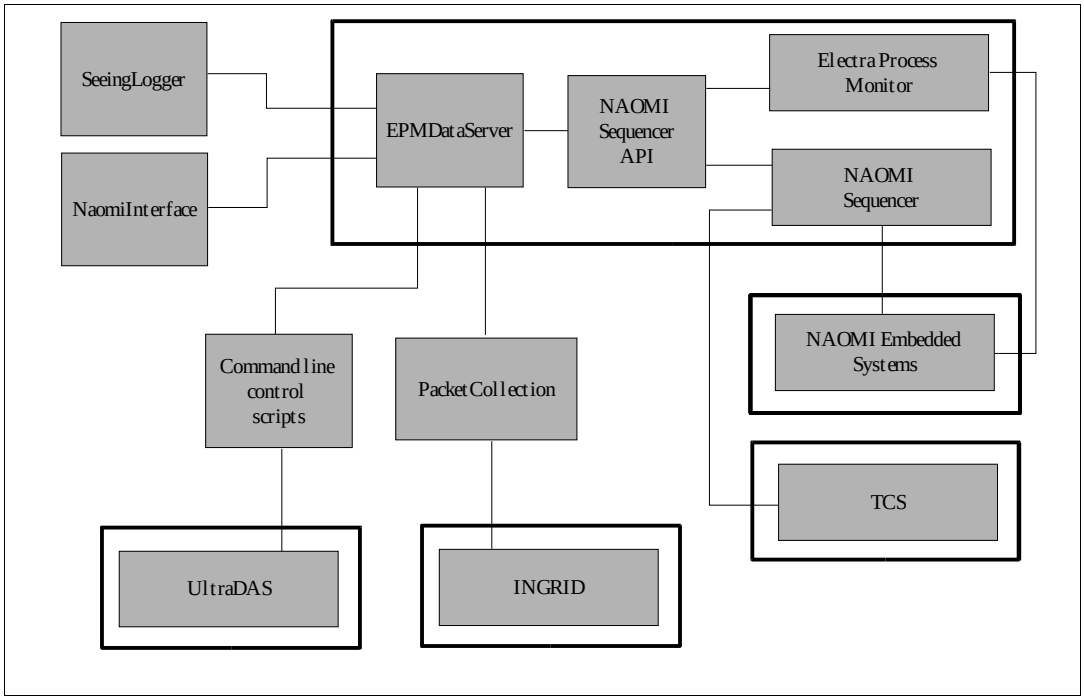
Chapter 3. The System Architecture

This chapter describes in brief the purpose of each of the software components that make up the NAOMI observing system, how they interact and a guide to the internal architecture of each of the components.

3.1 An Overview

The following illustration outlines the architecture of the top level NAOMI software developed at the ING and and data flows between the systems and subsystems.

A rectangle with a thick black line delimits a system boundary. Thin black lines indicate data flows.



3.2 The EPMDataServer

The NAOMI system as provided by Durham and the ATC contains a number of embedded computer systems. The interface to these embedded systems from a external client of NAOMI is via the *NAOMI Sequencer* and the *Electra Process Monitor (EPM).* The *NAOMI Sequencer* application was developed by the Durham team and is accompanied by a client API that applications external to the NAOMI system can use to control the NAOMI system. The *Electra Process Monitor* is used to store *process state variables* relating to the status of the NAOMI system. The NAOMI client API also provides an interface to the EPM so that clients can interrogate the status of the NAOMI system. In it's present state, the API is presented as a *Python* module which entails that any third party application *must* also be coded in Python. It should be noted that at this point in time, there does not exists a DRAMA interface to the NAOMI system.

In order to restrict the proliferation of yet another programming language within the ING, it was decided to develop a single *Python* task that would act as a bridge between third party

client applications and NAOMI using an interface which was *neutral* of the programming language used. The result of these endeavours was the *EPMDataServer*, a Python based script that uses the HTTP protocol to communicate with it's clients. This has the following advantages:

- It's an open standard, HTTP libraries exist for many different programming languages and are normally included as part of the distribution. All of the following programming languages come complete with HTTP libraries; Python, Tcl and Java. Third party libraries also exist for C/C++. Control can even be affected from a web browser.
- In the case of TCL, Java and Python, as the HTTP library comes complete with the programming tool, no special installations are necessary in order to communicate with the EPMDataServer.
- It is very flexible, the HTTP protocol is the language of the web, client applications can be located anywhere on the global network.

3.3 Running the EPMDataServer

The interface to the NAOMI sequencer from the EPMDataServer requires the installation of the Durham supplied GPM communication libraries to be installed upon the machine that runs the *EPMDataServer* in order for the client API to communicate with the NAOMI sequencer. At present only *navis.ing.iac.es* has the relevant libraries installed upon it. Therefore, prior to starting *any* of the NAOMI Observing Software described in this document, the developer ***must*** ensure that an instance of the EPMDataServer is running on *navis.ing.iac.es*.

All EPMDataServer *client* software must be instructed to direct NAOMI related status/control requests to *navis.ing.iac.es*. This may be done by setting the environment variable *NAOMIHOST* to *navis.ing.iac.es* prior to starting any of the NAOMI observing software on *taurus.* This is normally done by the *obssys* command when the observer selects the observing system to use.

See document *WHT−NAOMI−15* for information relating to starting and stopping the EPMDataServer.

3.4 The EPMDataServer HTTP Client Interface

Clients of the EPMDataServer request an action to be performed by the EPMDataServer using the standard HTTP GET and POST protocols. The exact action to be performed is specified by the URL specified by the client. The following subsections explain more clearly the URLs which are supported by the EPMDataServer.

All accesses of the EPMDataServer should be addressed to port *40000* on the machine that is hosting the EPMDataServer.

It should be noted that character case is significant in *all* HTTP transactions with the EPMDataServer.

3.4.1 The *NAOMI/Status* GET Request

The purpose of this URL is to return a HTML document of MIME type **text/html** which auto−refreshes and which can be requested and displayed from a standard web browser. It will contain the current state of the EPMDataServer and Electra process variables from the EPM. This is the easiest way of deducing whether the EPMDataServer is actually running upon a machine. By default the page will update every 15 seconds. An optional suffix may be appended to the URL in order to request that the page is not automatically refreshed. By

suffixing the URL with */on*, the page returned by the HTTP server will continually refresh. By suffixing the URL with */off*, the page returned by the HTTP server will *not* continually refresh.

The URL is as follows;

http://<hostname>:40000:/NAOMI/status/[on|off]

The <*hostname*> should be the name of the host that has the EPMDataServer running upon it . The easiest way to appreciate the mechanism employed is to ensure that the EPMDataServer is running on ***navis.ing.iac.es*** and then type the following into the URL locator on any web browser which will result in the status page being retrieved and displayed.

http://navis.ing.iac.es:40000:/NAOMI/status/on

The file that determines which of the EPM process variables are displayed in the status page is located in the directory */opt/software/Electra/lib/keyword.epm* on *navis.ing.iac.es*.

See section 3.4.7 for details of HTTP error responses which may be sent as a result of executing this command.

Note : Case is significant when specifying the URL.

3.4.2 The *NAOMI/status* POST Request

The purpose of the HTTP NAOMI/Status POST request is to retrieve from the EPM data server the values of the EPM variables that have been posted by the HTTP client. The URL is as follows;

http://<hostname>:40000:/NAOMI/status

The EPM variables that are to be returned by the EPMDataServer should be posted to the server in the following manner;

<variable name>,<variable name>,<variable name>, ...

An example follows;

CorrectedSeeingEstimate,UncorrectedSeeingEstimate

Will result in the current values of the EPM variables *CorrectedSeeingEstimate* and *UncorrectedSeeingEstimate* being returned by the EPMDataServer. The data will be returned in the HTTP object returned by the EPMDataServer as type MIME type **text/plain** and in the following format;

<variable name>\t<value>\n

<variable name>\t<value>\n

<variable name>\t<value>\n

....

An example of a document returned when the EPMDataServer is requested to return the values for the *CorrectedSeeingEstimate* and *UncorrectedSeeingEstimate* process variables follows;

CorrectedSeeingEstimate\t0.6\n

UncorrectedSeeingEstimate\t1.0\n

If a process variable is requested from the EPM that does not exist, the EPM will simply omit the process variable from the document returned.

See section 3.4.7 for details of HTTP error responses which may be sent as a result of executing this command.

Note : Case is significant when specifying the URL.

3.4.3 The NAOMI/command POST Request

The *NAOMI/command* POST request will be used by a HTTP client to instruct the EPMDataServer to in turn instruct the NAOMI sequencer to execute a command. The NAOMI sequencer provides a wide number of commands which allow various actions to be performed within the NAOMI subsystem. For example, the NAOMI sequencer provides a command *OpenWFSloop()* which allows the WFS control loop to be opened. An exhaustive list of the commands which are supported by the NAOMI sequencer can be found in the document *WHT−NAOMI−16*.

The details of the URL request to be made to the EPMDataServer from a client in order to initiate an action in the NAOMI sequencer are as follows;

http://<hostname>:40000:/NAOMI/command/<NAOMI sequencer command>

The <*hostname*> should be the name of the machine that has the EPMDataServer running upon it.

The <*NAOMI sequencer command*> should be the name of the command inside of the NAOMI sequencer that should be executed. For example, the <*NAOMI sequencer command*> would be in the case of a client wanting to open the WFS control loop, *OpenWFSloop.* The URL would therefore be as follows if the EPMDataServer was running on *navis.ing.iac.es*;

http://navis.ing.iac.es:40000/NAOMI/command/OpenWFSLoop

In some cases the actual commands to be executed on the sequencer require arguments to be specified. Command arguments for NAOMI sequencer commands can be specified in the POST request sent by the HTTP client in the following manner;

<argument>^\t<argument>^\t<argument>^\t ....

The argument separator token is the character pair **^\t**. The arguments are subsequently passed to the NAOMI sequencer command ***in the order*** that they were specified in the POST request.

The EPMDataServer will return a HTTP object of MIME type **text/plain** in response to a *NAOMI/command* request. The format of this response will be as follows;

CommandStatus\t<Current command status>\n

StatusReturned\t<Status returned>\n

ErrorMessage\t<Error message>\n

ErrorStatus\t<Error status>\n

Tag\t<Tag name>\n

If the command is initiated successfully by the NAOMI sequencer, a *tag* will be returned which will be an index to a variable in the EPM which will be subsequently be used by the client in order to track the progress of the command in the NAOMI sequencer. This *tag* will *not* be created in the event that the command could *not* be started by the NAOMI sequencer.

The semantics associated with the status variables outlined in this response are outlined in greater detail in the document *WHT−NAOMI−16*.

See section 3.4.7 for details of HTTP error responses which may be sent as a result of executing this command.

Note : Case is significant when specifying the URL.

3.4.4 The NAOMI/CommandStatus POST Request

The *NAOMI/CommandStatus* POST request will be used by a client in order to ascertain the progress of a command that it might have previously initiated in the NAOMI sequencer. The URL associated with this request is as follows;

http://<hostname>:40000:/NAOMI/CommandStatus

Where <*hostname*> is the name of the machine that is currently hosting the EPMDataServer.

The client should post as part of the HTTP request, the name of a ***tag*** *(see section 3.4.3 for more information about command tags*) associated with a command currently running in the NAOMI sequencer. The EPMDataServer will use this to index and return the correct command status information in the EPM process monitor.

If the command tag is found in the EPM, the EPMDataServer will return a HTTP object of MIME type **text/plain** describing the current state of the command associated with the tag.

The response will be in the following format;

CommandStatus\t<Current command status>\n

StatusReturned\t<Status returned>\n

ErrorMessage\t<Error message>\n

ErrorStatus\t<Error status>\n

Tag\t<Tag name>\n

The ordering of the status variables in the response *cannot* be guaranteed by the EPMDataServer.

The semantics associated with the status variables outlined in the response are outlined in greater detail in the document *WHT−NAOMI−16*.

See section 3.4.7 for details of HTTP error responses which may be sent as a result of executing this command.

3.4.5 The NAOMI/FreeStatus POST Request

The *NAOMI/FreeStatus* HTTP POST request will be used to instruct the EPMDataServer that the client is no longer interested in the command status data associated with the *command tag* specified by the HTTP POST request.

The EPMDataServer will simply return an empty header of MIME type **text/plain** in response to the command tag being successfully deleted from the EPM with a HTTP return code of 200.

See section 3.4.7 for details of HTTP error responses which may be sent as a result of executing this command.

Note : Case is significant when specifying the URL.

3.4.6 The NAOMI/debug/on and NAOMI/debug/off GET Request

The *NAOMI/debug/on* and the *NAOMI/debug/off* commands can be used to instruct the EPM data server to enable or disable the production of debug trace information. Trace information will be written to the *syslog*.

See section 3.4.7 for details of HTTP error responses which may be sent as a result of executing this command.

Note : Case is significant when specifying the URL.

3.4.7 EPMDataServer HTTP Return Codes

By default, if a request is serviced by the EPMDataServer *without* error, a HTTP code of 200 is returned.

The EPMDataServer in the case that an error is encountered whilst it is attempting to service a HTTP request, returns to the HTTP client one of the following HTTP error codes.

HTTP Error Code	***Error Text***
502	Comms error communicating with NAOMI EPM
503	Unidentified NAOMI request received
504	Exception raised when NAOMI sequencer command executed, command failed
505	The NAOMI sequencer does not recognise the command
506	NAOMI sequencer unable to fork and execute the command, seek expert assistance
507	NAOMI sequencer unable to execute command, command already in progress
508	Unknown status returned from the NAOMI sequencer
509	NAOMI Sequencer logic error, no status returned
510	Unable for EPM Data server to talk to NAOMI sequencer
511	Command tag not found in NAOMI sequencer

3.4.7.1 The 502 Error Response

The 502 response is returned when there is an attempt using the *NAOMI/status* POST request to access status information in the EPM but the EPMDataServer could not communicate with the EPM due to a communications problem.

The probable cause for this error is that the EPM is not running or there was a network error.

3.4.7.2 The 503 Error Response

A client has sent a unidentified HTTP request to the server. This may be due to either complete rubbish being set down to the EPMDataServer as part of the URL or there is a discrepancy in the character case of the URL being used to access the server.

3.4.7.3 The 504 error Response

When the NAOMI sequencer attempted to execute the NAOMI sequencer command requested by a HTTP client, the command raised a Python based exception in the sequencer. This may be due to either an internal failure of the remote command being executed by the sequencer or that an incorrect number of arguments have been specified for the command.

This response covers a whole multitude of failures in the execution of a remote command in the NAOMI sequencer.

3.4.7.4 The 505 Error Response

If the command that a HTTP client has requested the EPMDataServer to execute in the NAOMI sequencer does not exist, a 505 HTTP error response will be returned.

3.4.7.5 The 506 Error Response

This error will be returned in the event of a remote HTTP client attempting to start a command in the NAOMI sequencer via the EPMDataServer but the NAOMI sequencer is unable to create a new process in order to execute it.

This is a serious error in the NAOMI sequencer and will need manual intervention to resolve.

3.4.7.6 The 507 Error Response

This response will be returned when a remote HTTP client attempts to start a command in the NAOMI sequencer which is already running.

For example, if the NAOMI sequencer has been requested to point the telescope at a new source and whilst the action is in progress, a second request to move the telescope arrives at the NAOMI sequencer, the 507 error response will be returned to indicate that the sequencer was unable to execute the command as the action was already in progress.

3.4.7.7 The 508 Error Response

The NAOMI sequencer API that is used by the EPMDataServer to communicate with the EPM and the sequencer returns a status value after the execution of some of the API functions. This status response is a *higher* level status response and relates more to the ability of the API to execute the request being made of it rather than the *lower* level error responses which are returned directly from the NAOMI sequencer or the EPM.

The 508 response is returned if the EPMDataServer receives a status response from the API that it does not recognise.

3.4.7.8 The 509 Error Response

If a remote HTTP client requests the EPMDataServer to execute a command in the NAOMI sequencer API that it expects a status from and it gets none, the EPMDataServer returns a 509 response to the client.

3.4.7.9 The 510 Error Response

The 510 error is sent in response to a remote HTTP client attempting to start a command in the NAOMI sequencer but the EPMDataServer was unable to communicate with the NAOMI sequencer.

This will be due to the NAOMI sequencer not running or a network failure.

3.4.7.10 The 511 Error Response

A remote HTTP client has requested the EPMDataServer to return the command status associated with a tag that could not be found in the EPM.

3.5 The *seeingLogger* Task

The purpose of this application is to poll the EPM via the EPMDataServer at a specified interval and read out estimates from NAOMI for the corrected and uncorrected seeing. The values are then written to a seeing trace file which by default is stored in **/wht/var/seeingLogs**. More details relating to the structure of this file can be found in the document *WHT−NAOMI−15.*

The seeing trace files can then be subsequently used by other applications. Once such application is the *naomiInterface* program which reads and plots the seeing data written to this file in the seeing performance trace monitor.

The script is developed in *TCL* and *incr* and makes much use of the general purpose OCS support classes that have been identified and created throughout the development of the NAOMI observing software.

The *seeingLogger* will be started automatically when the observing system is started by the observer.

The source code for the *seeingLogger* application can be found in the directory *src/seeingLogger* in the source tree.

3.6 The *packetCollection* Task

This is script developed using *TCL* and *incr* whose primary function is to, when commanded by UltraDAS, retrieve header packets from the system described in it's configuration file and create packet files.

The *packetCollection* task initiates the collections of headers when it receives an ARCHIVE_B or ARCHIVE_E request upon it's DRAMA interface. If it is configured to collect headers from an EPICS based system, it uses the channel access/EPICS TCL extension in order to retrieve the packets from the target system. The *packetCollection task* contains extra logic which enables it to collect headers from NAOMI via the EPMDataServer.

The source for the *packetCollection* can be found in the directory *src/packetCollection* in the source tree.

An instance of the *packetCollection* collection configured for packet collection from both NAOMI and INGRID will be started when the observer starts the observing system. Every instance of the *packetCollection* must register itself upon the DRAMA network and the adopted convention that was agreed is as follows;

<Instrument Name|Camera Name>PKT

For example in the case of the INGRID packet collection task, the name the task is expected to register itself with DRAMA is as follows;

INGRIDPKT

There must also be a corresponding entry in the UltraDAS configuration files which instruct UltraDAS to collect headers from this task after performing an integration.

More details relating to the *packetCollection* task can be found in the document *WHT−NAOMI−15.*

3.7 The *LyotStopAngleUpdate* Application

This application is part of the OSCA CVS project and it's purpose is to automatically update the angle position of the Lyot Stop as the parallatic angle of the telescope changes. The application is started as part of the observing system and then registers itself with the TCS as

a monitor of the telescope position. When the parallatic angle of the telescope changes, the angular position of the Lyot Stop is recalculated and the mechanism is subsequently moved. The automatic update of the mechanism can be enabled and disabled through the use of a DRAMA action which the server exports, the details of which are outlined in the next subsection.

3.7.1 LyotStopAngleUpdate DRAMA Features

The LyotStopAngleUpdate daemon can be controlled externally through it's DRAMA interface. The daemon presents the DRAMA client with a number of the DRAMA actions. The name of the application on the DRAMA messaging system is **LyotStopAngleUpdate**

3.7.1.1 The *STOPUPDATE* Action

This action can be invoked in order to stop the automatic update of the Lyot Stop position depending on the parallatic angle of the TCS. The action takes no arguments.

3.7.1.2 The *STARTUPDATE* Action

This action can be invoked in order to start the automatic update of the Lyot Stop depending on the parallatic angle of the TCS. The action takes no arguments.

3.7.1.3 The *UPDATEANGLEOFFSET* Action

This action can be used to update the engineering constant which is derived by the OSCA engineers and is used as part of the calculation of the Lyot Stop angle. The action expects a single argument expressed as a float. The value will be subsequently stored in the WHTOCS database and as such will be maintained across restarts of the system. When the value is changed, an entry will be made in the syslog to that effect.

3.7.1.4 The *LYOTUPDSTAT* Parameter

The system exports a single SDS structure called **LYOTUPDSTAT** which contains two values called **UpdateStatus** and **OffsetAngle**. The former is the current status of the automatic updating of the Lyot Angle which can be set to either *Update* or *NoUpdate.* The **OffsetAngle** holds the current engineering constant used in the calculation of the Lyot stop angle offset.

3.8 The *naomiInterface* Application

This script is developed using ***TCL***, ***BLT*** and ***incr*** and it is the main GUI through which the observer will interact with the NAOMI system. The application has been heavily abstracted and the actual *naomiInterface* script is very small. It does however draw on the functionality of nearly all of the classes that have been developed to support both the OCS and NAOMI.

The source for the *naomiInterface* application can be found in the directory *src/naomiInterface* in the source tree.

3.9 The command line scripts

There are a number of scripts which may be executed from the command line which can be found in the directory *src/scriptCommands* in the development tree. All of the scripts have been developed in both *TCL* and *incr*. The purpose of the scripts is to allow the observer to

create observation scripts which include the functionality offered by the adaptive optics system, the INGRID instrument and the UltraDAS data acquisition system.

The scripts use many of the classes that have been developed in order to support the OCS and NAOMI in particular. Communication with the NAOMI sequencer is achieved via the EPMDataServer.

The *dither* script deserves a special mention. This script coordinates both actions in NAOMI and UltraDAS in order to achieve it's goal of performing a dithered observation using the adaptive optics system and the INGRID camera. Each point in the dithered observation translates to a small telescope offset being performed which is accomplished by sending commands to the NAOMI sequencer via the EPMDataServer which in turn results in the appropriate DRAMA commands being sent to the TCS in order to perform the slow offset. Once the slow offset has been applied, the script then instructs UltraDAS to perform an integration using the UltraDAS *run* command line script.

The script iterates through this cycle until all of the points in the dithered observation specified have been completed.

More details relating to the various command line scripts that accompany the NAOMI observing software can be found in the document *WHT−NAOMI−15*.

Chapter 4. The Class Packages

Much of the NAOMI observer level software has been abstracted into classes. Groups of related classes have been assembled conveniently into packages. The **package** is a standard device used in TCL to load extensions into the interpreter. Packages are loaded into the interpreter using a construct similar to the following :

```
package require Http
```

The above construct would instruct the TCL interpreter to attempt to find the package **Http** somewhere in the list of directories specified by the **TCLLIBPATH** environment variable.

In the case of the packages associated with the NAOMI observing system, the directory

```
/wht/<system>/lib/classLibrary
```

should be added to the **TCLLIBPATH** so that when one of the applications that comprise the NAOMI observer software requests a package released as part of the NAOMI observer level software, it finds the package associated with the *current* release of the observing system. By default the installation mechanism associated with the NAOMI observer level software installs it's packages into the above directory.

The packages are located underneath the directory *src/classLibrary* in the development tree.

The following sections outline the various packages that are installed as part of the release and include a brief description of the purpose of each of the classes that comprise them.

4.1 The *ICSSupport* Package

This package contains a number of classes which were identified as having a potential functional scope outside of the NAOMI system.

4.1.1 The *ErrorHandling* Class

This provides the consumer of the class with a standard interface for reporting *debug*, *error* and *informational* information relating to the operation of the consumer task. The class includes an interface to the *syslog* so that messages reported by the consumer class will be routed to the *syslog* as well as optionally to *stdout* and if specified, a *text widget* supplied to the class by the consumer. The final option is only open to TCL/Tk based applications rather than pure TCL applications.

All TCL based NAOMI observer level software use this class for the reporting of informational and error messages. The class uses the *OCSgp* package (see section 4.5) for it's interface to the syslog.

4.1.2 The *HTTPInterface* Class

This class forms that backbone of all TCL based NAOMI observer level software which needs to communicate with the EPMDataServer. The class provides methods which allow a consumer to request HTTP objects from a remote server in a simple and effective manner.

4.1.3 The *GeneralPurpose* Class

This class provides a number of static methods therefore there is no need for a consumer of this class to instantiate this class. The class is a place holder for a set of general non−specific methods would could not be logically grouped together.

4.1.4 The *SingleFieldEntry* Class

This class provides the consumer with the means to create a self contained dialogue window which allows the user to enter a data item of a specified data type. Once the user acknowledges the dialogue window, the class then validates the user input and returns to the consumer the value the user entered.

4.2 The *ICSGUISupport* Package

This package will contain a set of general purpose of classes that may be used to used to help with the construction of graphical user interfaces. This package will grow as the ICS develops.

4.2.1 The *DRAMADialControl* Class

This generic class provides a dial control provided by the **vu** widget set. The range of the dial can be specified as well as a mapping between engineering and logical units. Buttons associated with the control can be used to initiate a DRAMA action on a specified DRAMA server. The class also provides the user with the ability to associate a status DRAMA parameter with the control so it can reflect the movement status of the associated mechanism.

The class is a subclass of the *EPICSDialControl* class.

4.2.2 The *EPICSDialControl* Class

This generic class provides a dial control provided by the **vu** widget set. The range of the dial can be specified as well as a mapping between engineering and logical units. Buttons associated with the control can be used to update a EPICS PV. The class also provides the user with the ability to associate a status PV with the control so it can reflect the movement status of the mechanism.

4.2.3 The *EPICSButtonControl* Class

This generic widget control class provides a series of buttons which can be used to offer the user with a finite number of options which can be selected for a particular mechanism. For example it can be used to specify position of a filter wheel. Buttons associated with the control can be used to update a EPICS PV. The class also provides the user with the ability to associate a status PV with the control so it can reflect the movement status of the mechanism.

A number of options exist to configure the widget including the use of the WHTOCS RDBMS.

A mapping may be specified to map engineering units to logical units and vice versa.

4.2.4 The *DRAMAButtonControl* Class

This generic widget control class provides a series of buttons which can be used to offer the user with a finite number of options which can be selected for a particular mechanism. For example it can be used to specify position of a filter wheel. Buttons associated with the control can be used to initiate a DRAMA action. The class also provides the user with the ability to associate a DRAMA parameter with the control so it can reflect the movement status of the mechanism.

A number of options exist to configure the widget including the use of the WHTOCS RDBMS.

A mapping may be specified to map engineering units to logical units and vice versa.

This class is a subclass of the *EPICSButtonControl* class.

4.2.5 The *INGStatusDisplayWidget* Class

This class provides the consumer with a labelled text field dedicated to displaying status. The class is complete with methods for setting the status field and indicating whether the status displayed in the field is good or bad.

4.3 The *NAOMISupport* Package

This package consists of a number of classes created specifically to support the NAOMI observer level software. The classes contained in this package are detailed in the following sections.

4.3.1 The *NAOMICommand* Class

This class provides the consumer with the facility to execute commands within the NAOMI sequencer. It is used by *all* clients in the observing level software to execute commands in the NAOMI sequencer. The class itself is a consumer of the *HTTPInterface* class (see section 4.1.2) which it uses to communicate with the EPM and the NAOMI sequencer *via* the EPMDataServer.

Commands that are executed in the NAOMI sequencer are assigned a *tag* which is subsequently used for monitoring of the command's progress.

If the command was issued *synchronously*, control will *not* be passed back to the consumer until the command fails, completes or until a specified time−out is reached.

If the command is executed *asynchronously*, control is returned to the consumer as soon as the command is initiated in the NAOMI sequencer. The *NAOMICommand* will then *track* the progress of the command in the *background* and when the command completes, times−out or fails, either an internal *default* command completion callback will be called or a callback that was specified by the consumer in the initial call to the *NAOMICommand* class which resulted in the execution of the command in the NAOMI sequencer. When a command is issued asynchronously and a *command status tag* is returned, the class adds the *tag* to the *command tag pool*. The *tag pool* is used to store the details of *all* of the tags associated with commands that have been issued asynchronously and are still running within the NAOMI sequencer. The class intermittently in the background, for *each* tag present in the pool, queries the EPM in order to establish if the command has completed or not. Once a command is detected as being completed, the *tag* is removed from the tag pool and the specified command completion callback is called.

Details of the commands that may be issued in the NAOMI sequencer though the use of the *NAOMICommand* class are outlined in section 3.4.

4.3.2 The *NAOMIFOVMonitor* Class

This class describes the FOV status page displayed by the *naomiInterface* task (see section 3.8) . It contains all of the methods necessary to build the FOV display page, maintain the FOV status and to command NAOMI to move the WFS probe. The class *completely* encapsulates the functionality of the FOV display and as a consequence, could be easily integrated into other third party applications.

The class is reliant on a number of other classes which are provided as part of the NAOMI observing system such as the *HTTPInterface* and *NAOMICommand* support classes. The packages which contain dependencies are loaded as part of the *package* load sequence.

4.3.3 The *NAOMIGSFile* Class

This class describes the guide star selector page contained within the *naomiInterface* application (see section 3.8). The class provides the means to create the guide star selector page, to load guide stars and science objects from user prepared files and finally to position the telescope and the WFS probe.

The class is reliant on a number of other classes which are provided as part of the NAOMI observing system such as the *HTTPInterface* and *NAOMICommand* classes. The packages which contain the dependencies are loaded as part of the *package* load sequence.

4.3.4 The *NAOMIOSCAControl* Class

This class describes the OSCA control page contained within the *naomiInterface* class. The class provides control over the three mechanisms which constitute OSCA in addition to control over the *LyotStopUpdate* daemon. Using the control page, the user can deploy OSCA, set the Lyot Stop to any valid angular position and select any of the masks in the mask wheel. The LyotStopUpdate daemon can be instructed to not automatically update the Lyot angle and offers the user the ability to set the engineering constant which is used in the calculation of the lyot angle.

4.3.5 The *NAOMIImageGrab* Class

This class was designed to offer the user the ability to acquire guide stars by taking a camera image and allowing the user to position the WFS pickoff probe by simply clicking upon the desired guide star. In addition the class offers some rudimentary image processing features to enhance the image such histogram equalisation as well contrast and brightness adjustment. This class makes use of a third party application called ImageMagick which is used to perform some of the image processing. The class has to perform conversion between various images types in order to accomplish it’s goals. The raw camera image must be converted to FITS before being submitted to ImageMagick and to GIF before it can be displayed by the application.

The class creates a Tk Frame which contains all of the GUI controls for this feature. The calling class should pack this frame into a container frame which in the case of the NAOMI system, is performed by the *naomiInterface* application.

4.3.6 The *NAOMIPerformanceMeter* Class

This class describes the seeing performance meter page contained within the *naomiInterface* application (see section 3.8). The class provides the means to create a seeing trace display, load in seeing trace files which have been created by the *seeingLogger (*see section 3.5) and trace any new seeing data written to the currently monitored trace file upon the display. Furthermore the class provides windowing functionality so that the observer can change both the time window of the data displayed and *zoom* in on parts of the trace which are of particular interest.

The class is reliant on a number of other classes which are provided as part of the NAOMI observing system such as the *HTTPInterface* and *NAOMICommand* classes. The packages which contain the dependencies are loaded as part of the *package* load sequence.

4.3.7 The *NAOMIStatusDisplay* Class

This class describes the status panel display which is located at the foot of the main *naomiInterface* application (see section 3.8). The class provides the means to create a status panel which reflects some of the core NAOMI process status variables of most interest to the astronomer and to maintain the status display.

The class is reliant on a number of other classes which are provided as part of the NAOMI observing system such as the *HTTPInterface* and *NAOMICommand* classes. The packages which contain dependencies are loaded as part of the *package* load sequence.

4.4 The *PacketCollection* Package

This package contains a single class necessary to perform package collection duties for UltraDAS (see section 3.4.3). It is used primarily by the *packetCollection* task.

4.4.1 The *PacketCollection* Class

This class is loaded by the *packetCollection* task and it's main purpose is to retrieve from either an EPICS or an EPM based system, headers which are required by UltraDAS in order to create it's FITS files.

The class is reliant on a number of different modules being available such as the *HTTPInterface* class and the EPICS channel access TCL extension.

4.5 The OCSgp Package

The package consists of an ANSI C based shared library which is loaded dynamically into the the TCL interpreter at run−time. The library provides the TCL interpreter with a number of new commands which enable it to communicate with the syslog and to execute SLALIB commands.

Chapter 5. Building and Installing the Project

This chapter outlines how the developer can accomplish the building and installation of the project. It also outlines the environment variables that influence the build and installation procedures.

5.1 The Environment

The environment **must** be set up correctly in advance of any development work or installation being undertaken. The following commands need to be executed from the command line so that it will install and build correctly.

```
/opt/csg/bin/bash
cd <top level directory of project>
. setupEnvironment
```

Line 1

Starts the **bash(1)** shell, this is a POSIX compliant shell that I have a personal preference for and must be used when performing builds and installations throughout the project. All shell programming performed throughout the project has been based upon the bash shell and it is not compatible with the **csh**.

Line 2

Changes the current directory to be that of the top level directory of the project.

Line 3

Sources a script which sets up a number of environment variables which are used throughout the project.

The following environment variables will be set up by the *setupEnvironment* script.

5.1.1 The MAKEINCLUDEDIR Variable

The **MAKEINCLUDEDIR** variable holds the name of the directory that contains all of the make include files. Many of the make files included in the project *include* one of the two master makefiles which themselves contain all of the logic necessary to rebuild and install targets specified by the makefile.

5.1.2 The PYTHONPATH Variable

The **PYTHONPATH** variable is used to indicate to the Python interpreter additional directories to look in when searching for modules that may have been imported by a Python script. Modification of this variable is necessary so that the EPMDataServer can pick up the NAOMI Sequencer API stub toolkit which was provided by Durham to assist in the development of the software in the absence of the actual hardware being present at the ING.

After commissioning is complete and NAOMI is on−site, the software should be able to use the *actual* API rather than the stub tool kit. The only software component that this environment variable influences is the *EPMDataServer* software.

5.1.3 The TCLLIBPATH Variable

The **TCLLIBPATH** variable is used to indicate to the TCL interpreter additional directories to look in when searching for packages (see chapter 4) that may have been imported by a TCL script.

By default, the additional directories which are included are the **/opt/csg** directories, the **/wht/lib/itclCA** directories which contain the extra classes that have been provided to facilitate EPICS channel access and the **classLibrary** directory in the development tree which contains all of the development classes which are used by the various components of the system.

5.1.4 The PATH Variable

The **PATH** variable is modified in order that the **/opt/csg/bin** and the **src/scriptCommands** directories are placed at the head of the command search path.

5.1.5 The PREFIX Variable

The standard makefile variable **PREFIX** is set to the root directory of the installation tree which in the case of the WHT observing system is **/wht**. This variable is used by the makefiles when installing the project.

5.1.6 The OBSERVING_SYSTEM Variable

The **OBSERVING_SYSTEM** variable is used by the makefiles in order to instruct them where to install the project. This variable should be set to the name associated with the current observing system that the developer wants to build the system to. For example, this could be **s9−2**.

5.2 Building the Project

A build of the entire project tree can be accomplished by executing the following command in the top level directory of the project;

```
make all
```

This will result in the make program iterating throughout all of the project development sub−directories attempting to build each of the target directories.

Individual sub−directories can also be rebuilt by changing to the directory in question and executing the following command

```
make all
```

from within the directory.

5.3 Installing the Deliverables

In order to install executables and manual pages that form part of the deliverable for the project, the following command needs to be executed from the top level directory of the project. It should be noted that in order to install software into the **/wht** tree, the user must have be **whtsm**.

```
make install
```

This will result in the **make** program iterating throughout all of the project development sub−directories attempting to install targets specified by the makefiles in each of the target directories.

Individual sub−directories can also be installed by changing to the directory in question and executing a

```
make install
```

from within the directory.

It should be noted that there is no implicit *make all* executed on the directories ahead of a *make install* due to the different permissions required for both operations.

The root directory of the installation is defined by the **PREFIX** and **OBSERVING_SYSTEM** environment variables outlined in sections 5.1.5 and 5.1.6. These two variables should be set up as part of the *setupEnvironment* script outlined in section 5.1.

5.4 Installing the documentation

The manual pages provided by installation are installed as part of the *install* target. There is however on−line HTML documentation which can be installed. This documentation can be found in the sub−directory **docs** of the main development tree.

Online documentation can be installed by executing the following command from the Unix command line prompt from the top level directory of the project.

```
make install−docs
```

This must be done as user **docs** as the makefile attempts to install the documentation into the standard CSG online documentation directories.