NRC·CNRC
From Discovery to Innovation...

Science at work for Canada

HERZBERG EXTENSIBLE ADAPTIVE REAL-TIME CONTROLLER (HEART)

External Interface Definition Document

National Research Council Canada
Herzberg Astronomy and Astrophysics
February 2021

Table of Contents

1. INTRODUCTION 5

1.1 Background 5

1.2 Purpose 5

1.3 Scope 6

1.4 Reference Documents 6

1.5 Change Record 6

1.6 Abbreviations 7

2. SUMMARY INTERFACES 8

3. INTERFACE SPECIFICATIONS 12

3.1 Block Initialization and Configuration 13

3.2 Standard Output Blocks 13

3.2.1 DM Output Custom Handler Interface 13

3.2.2 Tip/Tilt Output Custom Handler Interface 16

3.2.3 LGS FSM Output Custom Handler 17

3.2.4 Telescope Offload Output Custom Handler 20

3.2.5 Externally Streamed Telemetry using Data Socket 20

3.3 Standard Input Blocks 21

3.3.1 Input WFS Custom Handler Interface 21

3.4 Command, Status, Data and Message TCP Sockets 23

3.4.1 Standard Header and Footer 24

3.4.2 Standard Status Reply 26

3.4.3 Downstream AO streams 26

3.5 Checksum Variants 27

3.6 Externally streamed telemetry 27

4. COMMAND STRUCTURE, COMMANDS 28

4.1 Command Structure 28

4.1.1 Command Types 28

4.1.2 Command Properties 31

4.1.3 Standard Commands 31

4.2 Command Status and State 31

4.2.1 Standard States 32

4.2.2 Command State Attribute 32

4.2.3 Standard Command Status 33
4.3 Commands Accepted by the RTC Command Handler and Standard Commands 34
4.3.1 mode 37
4.3.2 calibBackground 39
4.3.3 pipeline 40
4.3.4 loopOpen 41
4.3.5 loopLgsTt 42
4.3.6 loopHigh 43
4.3.7 loopLow 44
4.3.8 offloadTcs 45
4.3.9 subApMaskLgsSet 46
4.3.10 filterTemporalSet 47
4.3.11 loopParamReset 48
4.3.12 paramConfigSave 48
4.3.13 paramConfigSet 49
4.3.14 dmShape 50
4.3.15 ttsSet 51
4.3.16 enableHrtFlags 51
4.3.17 enableSrtFlags 52
4.3.18 changeLoopRate 53
4.3.19 dumpBuffer 54
4.3.20 setTelemRecording 55
4.3.21 calibModePixel 55
4.3.22 calibModeGrad 57
4.3.23 calibModeCmd 58
4.3.24 calibModeWc 59
4.4 Standard Commands 59
5. STATUS, NOTABLE EVENTS AND ALARMS 65
5.1 Published Status 68
5.1.1 Published Status Names and Description 68
5.1.2 Published Status Attributes 72
5.2 Subscribed Status 92
5.3 Notable Events 92
5.3.1 Notable Events Publishing Rate 93

5.3.2 Notable Event Fields 94
5.4 Alarms 95
6. FILE FORMATS OF READ-IN FILES 96
7. ADDITIONAL COMMANDS NOT YET PROMOTED 97
7.1 Commands 98
7.1.1 algoSetLgs 98
7.1.2 algoSetNgs 99
7.1.3 algoSetOiwfsOdgw 100
7.1.4 setLgsTtCntrl 101
7.1.5 lgsfFsmZero 102
7.1.6 loopLgsFocus 102
7.1.7 integratorControl 103
7.1.8 loopLowTier0 104
7.1.9 loopLowTier1 104
7.1.10 loopLowTier2 105
7.1.11 loopLowTier3 106
7.1.12 paramConfigSetSRT 108
7.2 Commands Specific to Architecture 108
7.2.1 loopLgsDither 108
7.2.2 loopTwfs 109
7.2.3 loopNgsDither 110
7.2.4 offloadOiwfsPoa 111
7.2.5 ngsTelDither 112
7.2.6 guideStarHandOff 113
7.2.7 rtsDelete 114
7.2.8 activateSRT 114
7.3 Status 115

1. INTRODUCTION

1.1 BACKGROUND

The Herzberg Extensible Adaptive Real-Time Controller (HEART) External Interface Definition Document is the primary document that describes the external interfaces for all outside sources to interact with a HEART based RTC system. It describes the connection points and their protocols, how issue command and monitor status, and how to receive output from the various components within the HEART system.

To assist with understanding the HEART RTC structure and where the interface points come from, HEART uses the concept of a “block”, which is a reusable software unit from which an RTC implementation is comprised. These block perform various specific deterministic tasks that, when stitched together (in what is referred to as a pipeline), combine together to make an RTC. An RTC can contain multiple pipelines through various points in the process that each provide inputs and output to compose the overall RTC system data flow.

One common component to all HEART based RTCs are the real-time inputs and outputs required to fulfil its role as a RTC. Unfortunately, because of the variability of needs and requirements for RTCs, these inputs and outputs are not always the same. Yet, through the flexibility and configurability of HEART, it provides the necessary functionality to create these inputs and output to match any unique need. To do this, HEART not only provides standardized blocks for many of the common inputs and outputs to RTCs, but also custom handlers to deal any unique interface. For example, an input WFS may need custom code to accept the input from the WFS output format. This is then translated to an expected HEART data structure. For outputs there are standardized blocks for each type of output that will take the data (e.g. DM command vector), which can then be translated and output for specific hardware. These types of inputs and outputs are done at real-time speeds and, as a result, are tightly integrated with the HEART hard real-time (HRT).

By design, the external interfaces assume that integrated systems will connect through an Ethernet connection or GigE. But, from a HEART perspective, the type of interface with the hardware is transparent as the connection will be handled by a custom input handler. This allows for HEART to integrate with components of varying functionality while utilizing the custom blocks parts of the interface blocks capabilities to help with the configuration of the connection.

The primary inputs expected to be send to HEART are in the form of pixels and centroids. The primary outputs are Deformable Mirror (DM) commands, Tip/Tilt (TT) commands, Laser Guide Star (LGS) Fast Steering Mirror (FSM) commands and externally streamed telemetry data.

Additional RTC inputs consist of commands and data from the AO system controller. The RTC includes a command handler which processes the commands and data from the AO system controller and routs it to the necessary parts of the HEART based RTC.

1.2 PURPOSE

This document will discuss how HEART handles the custom real-time inputs and outputs, it also includes a description of the example custom handling functions contained within HEART. The format of commands, data and messages between the RTC and the System Controller are also described here.

This document does not describe how the custom part of the interface is written, rather it describes the custom handler functions and how the inputs and outputs are to be handled.

1.3 SCOPE

This document will only address the custom handlers for inputs and outputs of HEART, also the message format between RTC and AO System Controller.

1.4 REFERENCE DOCUMENTS

RD1 HEART Internal Interface Document

1.5 CHANGE RECORD

Revision	Date	Section	Modifications
DRF01	2020-11-11	All	JD: Initial draft
DRF02	2020-11-16	All	MS: Draft revision.
REL01	2020-11-19	All	Initial Release
REL02	2021-02-15	All	CRD Release

1.6 ABBREVIATIONS

AO – Adaptive Optics

DM – Deformable Mirror

FSM – Fast Steering Mirror

GNAO – Gemini North Adaptive Optics

GVCP – GigE Vision Control Protocol

GVSP – GigE Vision Streaming Protocol

HEART – Herzberg Extensible Adaptive Real-time Controller

HRT – Hard Real-time

ICD – Interface Control Document

LGS – Laster Guide Star

N/A – Not Applicable

NGS – Natural Guide Star

OIWFS – On-Instrument Wavefront Sensor

RTC – Real-time Controller

RTS – Real-time Telemetry Storage

SRT – Soft Real Time system

TBC – This item still needs to be confirmed

TBD – This item still needs to be determined

TT – Tip/Tilt

TTS – Tip/Tilt Stage

WC – Wavefront Corrector

WFS – Wavefront Sensor

2. SUMMARY INTERFACES

The generic HEART RTC context diagram is shown in Figure 1. The RTC Command Handler is the primary entry point into the HEART based RTC. It's receives commands and reports status to the System Controller (or any other connected client). Other outputs and inputs are embedded within the HRT, and the External Telemetry Handler part of the storage system.

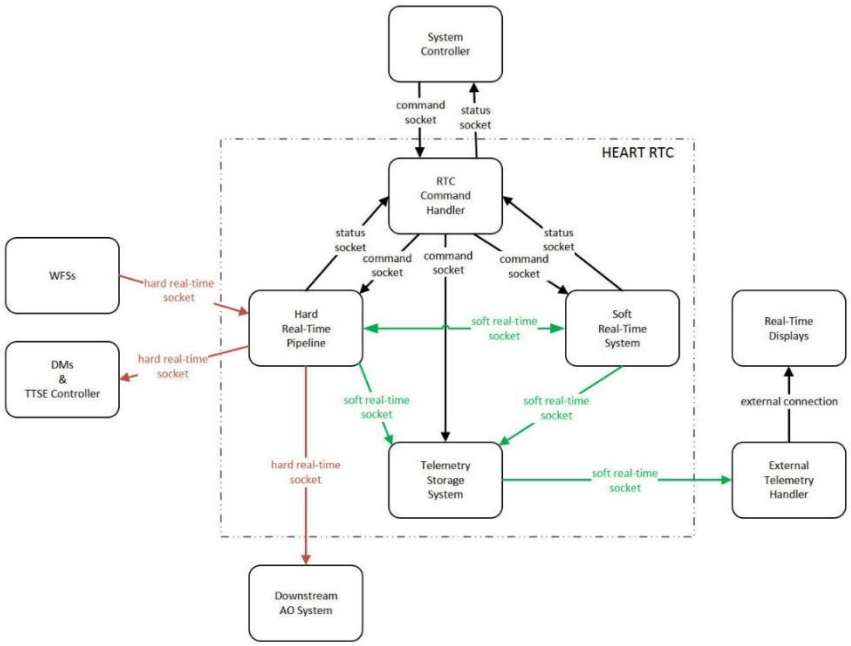

Figure 1 - HEART RTC Context Diagram

The RTC Command Handler accepts and processes commands from an externally connected system via a TCP connection. For example, a System Controller can be used as the primary process responsible for issuing commands and monitoring status, but the RTC Command Handler can accept additions TCP connections to also relay status data. This communication is included in this document.

By default, the real-time data stream interfaces between HEART HRT and external components are expected to be through Ethernet sockets. There are block functions that provide access to configuration information and the ability to open/close standard HEART sockets, a HEART based RTC also provides mechanism for customized interfaces should a device or system not support the HEART format. For example, the GigE Vision Streaming Protocol could be supported for incoming pixel and gradient streams through customized blocks integrated into HEART. Similarly, customization of outputs can be used to support other output device formats.

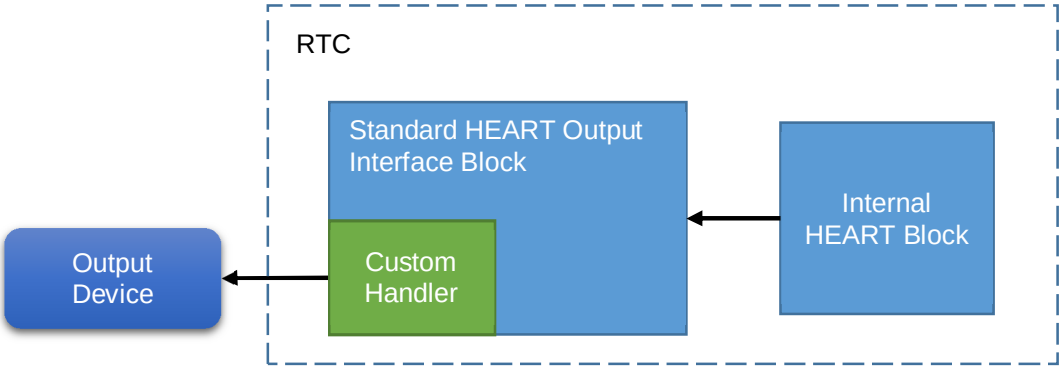

Figure 2 - Standard Output HEART Interface Block

Figure 2 shows the layout for a Standard HEART output interface block. Standard output devices include:

- deformable mirror (DM) controllers,
- tip/tilt (TT) controllers,
- laser fast steering mirror (FSM) controllers,
- telescope offloading, and
- external servers to accept streamed telemetry or modal offload values.

There is a standard HEART output block for each of these types of outputs and each of these blocks will utilize a custom handler to interact with the device. A HEART HRT block that provides output, such as a vector of DM actuator values, would trigger the Standard HEART DM interface block. To support the customized format of the devise, the block would, in turn, call a custom handler to complete the communication. A custom handler (i.e. function) would take the HEART formatted information, translate it, and send it to the DM controller in the required format (Section 3.2).

This document details the specific function parameters to each of the different types of custom handlers. The custom handler is responsible for actually performing the output, which will be specific to the hardware that the custom handler is communicating with. HEART HRT includes an example custom handler, and this document includes details of how that example handler will send data, including details on the datagram to be sent. These are included in HEART so that those interfaces can be tested without knowing the specifics of the real hardware.

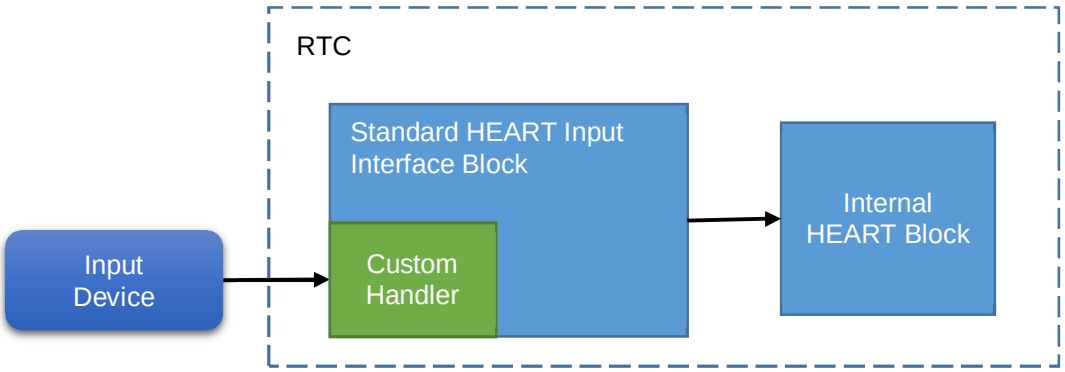

Figure 3 - Standard Input HEART Interface Block

Figure 3 shows the layout for the Standard HEART input interface block. Inputs are generally expected to be pixels and gradients. HEART will contain standard HEART Input interface blocks for the different types of inputs. Much like the outputs, the input blocks will call a custom handler (i.e. function) which will read data, perform a necessary translation, and provide it to the RTC. Section 3.3 details the specific inputs to the custom handler. This custom handler is responsible for performing the reading, which will be specific to the hardware that the custom handler is communicating with. It will then prepare the received data for processing by the HRT.

For example, to handle and process pixels sent from a custom WFS, a HEART Input Interface block that receives the WFS data as an input to the HRT would trigger any downstream block once the pixels start arriving for a frame. This is done through the custom handler (function) that would read the device specific input stream, store it in memory in a standard format and then use HEART functions to inform the RTC that data is available. When reading pixels or gradients for a given WFS exposure, it is important to not wait until all pixels (gradients) are received before triggering the RTC to process it as this would cause a significant delay. To avoid these delays, HEART allows the stream of pixels (gradients) to be processed as it arrives which results in an output stream that can be processed in parallel by a subsequent HEART HRT block. It is the custom handler is responsible for interfacing with the specific hardware and informing internal components when it has arrived and is ready to be processed. HEART HRT includes a sample custom handler, and this document includes details of how that sample handler will receive data, including details on the datagram to be received. This sample handler was developed to be used when simulating inputs and sending it into HEART. This document will detail the interfaces for that custom handler for the different types of input interface blocks.

To support the customization of input and output buffers, HEART has the ability to read configuration files and provide information to the block. For example, custom systems may have specific requirements around socket port. Item's like this can be stored as configuration that is loaded at runtime when starting the customized blocks. Also, many custom handler functions will require a setup function that will be called upon block creation; this setup function is used to perform any custom functionality required upon initialization.

In the example custom handlers provided in HEART, UDP socket communication was used to demonstrate the low latency requirements. When operating within the critical path of the RTC, UDP packet communication is the primary choice of protocol when other means are not available (such as inter-process or cross-system communication). UDP communication should be used when timing constraints are critical, therefore, chosen as a good practice to follow. It is not expected to experience significant data loss/corruption in the controlled environment such as the

RTC system. But, when transmitting data via UDP, checksums can be utilized to validate the data was correctly transmitted as a precautionary measure to indicate networking issues.

3. INTERFACE SPECIFICATIONS

This section details the HEART functions that need to be used in the custom handlers for input and output block. It also describes what is in the example HEART custom handler to provide the outputs and inputs, including the datagrams used for the example. It is expected that these example custom handlers would be replaced with the interface of the real hardware, except when that interface is being simulated. It also details the internal and external command, message, and data formats of the command.

The following C data types are used to specify the datagrams used for HEART example data streams.

Table 3-1- Data type descriptions

C type	Description	Example Uses
uint8_t	8 bit unsigned integer	Used for storing small integer values. Also used for packed 12 bit WFS pixels; two pixels packed per three bytes.
uint16_t	16 bit unsigned integer	Default data type for raw WFS pixels.
uint32_t	32 bit unsigned integer	Used for checksums and sizes.
uint64_t	64 bit unsigned integer	Compact representation of timestamps.
int16_t	16 bit signed integer	Status value.
float	32 bit floating point	Used for the following values: DM actuator displacement, WFS gradients, tip/tilt, calibrated pixels, modal offload values.

All custom handler calls returns a daoErr_t structure. If an error occurs, then errMsg is expected to include a useful, human readable message, including an indication of what is wrong and potentially how to resolve the error.

The structure looks like:

```
typedef struct
{
    const char * caller;                  /* origin of error message */
    daoErr_status_t status;               /* status value */
    char errMsg[DAO_ERR_MSG_BUFFER_SIZE]; /* error message */
} daoErr_t;
```

If no error occurred, then a status value of DAO_ERR_NONE (0) is returned. If an error did occur the value is expected to be negative. Helper macros for populating these error structures will be provided.

Since the RTC has a latency specification, and it is expected that the custom handler functions will not include any artificial delays, or other real-time inefficiencies. In the standard blocks that call the custom handlers, a timestamp will be retrieve before and after the call to the custom handler to verify that the handler is not causing undue delays.

3.1 BLOCK INITIALIZATION AND CONFIGURATION

Each block will have both a configuration and initialization. This provides information that would be required in the Custom Handler Interface. That information is defined in the specific interface ICD. The following are the types of information that may be considered:

- configuration variable to specify the checksum type within the data packets. This would be added to the appropriate ICDs
- network type order indicating if the data streams are going to be converted to network byte order
- data dimensions
- details of the configuration file contents are in [RD1].

3.2 STANDARD OUTPUT BLOCKS

The standard output devices for HEART are Deformable Mirror (DM) controllers, Tip/Tilt (TT) controllers, Laser Guide Star (LGS) Fast Steering Mirrors (FSM), telescope offloading and external computing systems which accept externally streamed telemetry (e.g. for user interfaces or offloading low order modes to the secondary control system). There are standard blocks for each of these devices. Each of them have custom handler interfaces, and are defined in the following sections.

3.2.1 DM Output Custom Handler Interface

The standard HEART DM block will call a custom handler function with the following signature:

daoErr_t dmDriver(int16_t dmTarget, int numAct, const float * actVals);

This device specific custom handler function accepts a DM target specifier (dmTarget), the number of actuators within the DM (numAct), and an array of actuator values (actValues). The custom handler function will convert the data to the appropriate format for the hardware controller and send it to the DM controller. During initialization of the DM block, the custom handler function is installed as the DM output function. This allows the custom handler function to be called via a function pointer when the DM vector computed by HEART is ready to be sent to the DM controller.

Input are:

- dmTarget: can be used by the dmDriver function to internally select the appropriate DM device (if required).
- numAct: number of actuators values in the actVals array
- actVals: One floating point value for each actuator, in microns.

Returns:

- daoErr_t: Defined at the beginning of this section.

It is expected that the DM Controller will reject out of range position demands by holding current position.

3.2.1.1 Example DM Custom Handler

The example DM Custom handler function in HEART takes the actValues and sends one or more UDP datagrams formatted as shown in Table 3-2. It is expected that the DM controller returns a status datagram, formatted as shown in Table 3-3 for each DM command vector (not datagram).

Table 3-2 - DM Vector Datagram Format

Size (bytes)	Data Type	Description
2	uint16_t	DM target identifier.
1	uint8_t	Datagram sequence number within DM vector. The sequence number is reset to zero for the first datagram of each DM vector.
1	uint8_t	Number of datagrams per DM vector. The limit of 255 datagrams per DM vector is not an issue since over 300 actuator values can be sent within a single non-jumbo datagram.
2	uint16_t	Actuator index offset. DM actuator index of first actuator included in datagram. The use of 16 bit actuator indexing allows over 65000 actuators per DM.
2	uint16_t	Number of actuator values included in current datagram.
4	32 bit integer	RTC frame number.
4 * N	32 bit floats	One floating point value for each actuator value sent. Actuator order is dependent upon the DM controller. Units are in microns
4	uint32_t	CRC-32C checksum

The example custom handler will send one or more DM datagrams to each DM controller. The number of datagrams is dependent upon the number of actuators and the Ethernet frame size (i.e. depending on whether DM controller supports jumbo frames). The first six fields constitute the header. The grey row being repeated once for each actuator value sent. Once validated and complete, the DM controller should return a status structure as shown in Table 3-3.

Table 3-3 - DM Vector Status Datagram

Size (bytes)	Data Type	Description
2	uint16_t	DM target identifier
2	uint16_t	Number of actuator values received.

4	uint32_t	RTC frame number.
2	int16_t	Status values: Zero: The DM vector datagram is accepted. Positive values: No datagram was rejected but one is either missing or received late. Negative values: The datagram was rejected.
2	uint16_t	Actuator identifier. If the DM vector was accepted, then this value is 0. If an error is associated with a specific actuator, then the actuator number is specified here.
4	uint32_t	CRC32-C checksum

The following is a list of status values returned from the simulated DM Electronics.

Table 3-4 - DM Status Return Values

Status Value	Description
0	Datagram for the frame was received and applied to DM.
1	The expected datagram was received (and applied) but one additional stale datagram were also received. Discussion: This scenario would occur if a datagram was too late for the previous frame but then arrived before datagrams for the current frame. If all valid datagrams for the current frame are read then the DM shape should be applied despite there being an extra datagram.
2	Timeout error. No datagrams were received within the expected time.
-1	A datagram with invalid checksum was received.
-2	Incorrect target detected. The receiving DM controller does not match the DM target identifier specified in the header.
-3	Invalid header information detected (e.g. invalid number of actuators).
-4	Command rejected: Inter-actuator stroke limit exceeded.
-5	Command rejected: Stroke limit exceeded.
-6	Actuator failure (e.g. electrical disconnection detected).
-7	Electronics failure detected (e.g. the amplifier output does not match command).
-8	Calibration fault: forward and inverse linearization mismatch.
-9	Inter-actuator voltage fault.

3.2.2 Tip/Tilt Output Custom Handler Interface

The standard HEART Tip/Tilt block will call a custom handler function with the following signature. This device specific custom handler function accepts a Tip/tilt target specifier and will send Tip/Tilt commands for the Tip/Tilt stage once per frame. The tipTilt_t data type contains the tip and tilt values as float fields.

daoErr_t ttDriver(const tipTilt_t * ttCommands, tipTilt_t * ttSensors);

The custom handler function will convert the data to the appropriate format for the hardware controller and send it to the tip/tilt controller. During initialization of the Tip/Tilt block, the custom handler function is installed as the tip/tilt output function. This allows the custom handler function to be called via a function pointer when the tip/tilt vector computed by HEART is ready to be sent to the tip/tilt controller.

Input are:

- ttCommands: can be used by the ttDriver function to internally select the appropriate Tip/Tilt device if required.
- ttSensors: Tip and tilt, one floating point value for tip and tilt, in microns.

Returns:

- daoErr_t: Defined at the beginning of this section.

It is expected that the Tip/Tilt Controller will reject out of range position demands by holding current position.

3.2.2.1 Example Tip/Tilt Custom Handler

The example Tip/Tilt Custom handler function in HEART takes the ttSensors and sends the UDP datagrams formatted in Table 3-6. The first field contains a non-negative frame counter. The next two fields are the tip /tilt commands in milli-radians of mechanical tip/tilt relative to their untilted positions.

The simple 32 bit checksum is discussed in Section 3.5. The checksum type used in the tip/tilt datagrams will be specified as part of the tip/tilt configuration.

Table 3-5 – TTSE Command Data Format

Size (bytes)	Data Type	Description
4	*uint32_t*	Frame number
4	float	Tip command (milli-radians rotation about axis #1)
4	float	Tilt command (milli-radians rotation about axis #2)
4	uint32_t	Simple 32 bit checksum

For the example Tip/Tilt custom handler function, when a tip/tilt command is sent with a frame number of zero, it is to be interpreted as a query command. The custom handler should ignore the included tip/tilt values (these will also be zero) and simply return the current tip/tilt sensor values in the status outlined in Table 3-6.

Table 3-6 - TTSE Status Data Format

Size (bytes)	Data Type	Description
4	uint32_t	Frame number for received TTS command.
4	int32_t	Status value (32 bit to align tip/tilt float values).
4	float	Tip sensor value (milli-radians rotation about axis #1)
4	float	Tilt sensor value (milli-radians rotation about axis #2)
4	uint32_t	Simple 32 bit checksum

The status field can contain the status codes listed in Table 3-7.

Table 3-7- TTSE Status Values

Status Value	Description
0	Tip/tilt command datagram was received and no errors were detected.
1	A repeated or stale datagram was received.
-1	An invalid frame number was received.
-2	An invalid tip command was received
-3	An invalid tilt command was received

3.2.3 LGS FSM Output Custom Handler

The RTC will send a command for the FSMs once per AO frame. LGS centering errors are sent to the Laser Fast Steering mirrors (FSMs) (i.e. jitter mirrors). The standard HEART LGS FSM block will call a custom handler function with the following signature. This device specific custom handler function is defined with the following signature:

daoErr_t lgsFsmDriver(int numFSM, int *mask, const float * lgsFsmVals);

The custom handler function will convert the data to the appropriate format for the hardware controller and send it to the DM controller. During initialization of the LGS FSM block, the custom handler function is installed as the LGS FSM output function. This allows the custom handler function to be called via a function pointer when the LGS FSM vector computed by HEART is ready to be sent to the LGS FSM controllers.

The input are:

- numFSM: number of fast steering mirrors (N_{FSM})
- mask: bit mask of enabled FSM
- lgsFsmVals: vector of 2 * N_{FSM} float elements (4 bytes) ordered as follows (Ax, Ay, Bx, By, Cx, Cy,…) and defined in a pre-configured coordinate system. A, B, C, … represent different FSMs. Units of milli-arcseconds on the sky relative to nominal LGS position.

This assumes N_{FSM} fast steering mirrors. If one or more FSM are disabled (as shown by the mask), a value of (0.0, 0.0) will be sent for their command.

Return value:

- daoErr_t: Defined above

It is expected that the Controller will reject out of range position demands by holding the current position.

3.2.3.1 Example LGS FSM Custom Handler

The example LGS FSM Custom handler function in HEART takes the lgsFsmVals and sends the X/Y offset commands in UDP datagrams with the data format list in Table 3-8. The value of N_{FSM} is dependent upon the LGS system but HEART will support at least six LGS FSM.

If more than one FSM command is detected as invalid, then the status value returned will correspond to one of the invalid FSM commands.

The checksum type used in the LGS FSM datagrams will be specified as part of the LGS FSM configuration.

Table 3-8 – Laser FSM (Jitter Mirror) Command Data Format

Size (bytes)	Data Type	Description
4	uint32_t	Frame number
2	uint16_t	N_{FSM} : Number of fast steering mirrors in system.
2	uint16_t	Bit mask of enabled FSM. Bit 0 (least significant bit) corresponds to the first FSM in the list. Tip/tilt values (0,0) are included even for disabled FSM.
2 * N_{FSM}	float	Vector of 2 * N_{FSM} float elements (4 bytes) ordered as follows (Ax, Ay, Bx, By, Cx, Cy,…) and defined in a pre-configured coordinate system. A, B, C, … represent different FSMs. Units of milli-arcseconds on the sky relative to nominal LGS position.
4	uint32_t	Simple 32 bit checksum

This example custom handler assumes N_{FSM} number of fast steering mirrors. If one or more FSM are disabled, a value of (0.0, 0.0) will be sent for their command. The first field contains a non-negative frame counter which is incremented each time the RTC sends the FSM commands. The Vector field contains the x/y centering errors in milli-arcseconds on the sky relative to nominal LGS position. The final field is a simple checksum formed from the datagram contents (excluding the checksum itself). A UDP datagram sent to the FSM with a frame number of zero is to be interpreted as a query command (if that functionality exists). The FSM should ignore the included x/y values (these will be zero also) and simply return the current FSM values in a status datagram outlined below:

Table 3-9 - FSM Status Datagram

Size (bytes)	Data Type	Description
4	uint32_t	Frame number for received FSM command.
4	int32_t	Status value
2	uint16_t	N_{FSM} : Number of fast steering mirrors in system.
2	uint16_t	Bit mask of enabled FSM. Bit 0 (least significant bit) corresponds to the first FSM in the list. Tip/tilt values (0,0) are included even for disabled FSMs.
up to 48	32 bit float	Vector of 2 * N_{FSM} float elements (4 bytes) using the same ordering as Laser FSM Command above. The use of this field is dependent upon the functionality of the LGS FSM.

The status field can contain the status codes in Table 3-10.

Table 3-10- FSM Status Values

Status Value	Description
0	LGS FSM commands were accepted without error.
-1	FSM command Ax is invalid (out of range).
-2	FSM command Ay is invalid (out of range).
-3	FSM command Bx is invalid (out of range).
-4	FSM command By is invalid (out of range).
-5	FSM command Cx is invalid (out of range).
-6	FSM command Cy is invalid (out of range).
-7	FSM command Dx is invalid (out of range).
-8	FSM command Dy is invalid (out of range).
-9	FSM command Ex is invalid (out of range).
-10	FSM command Ey is invalid (out of range).
-11	FSM command Fx is invalid (out of range).
-12	FSM command Fy is invalid (out of range).

If more than one FSM command is detected as invalid, then the status value returned will correspond to one of the invalid FSM commands.

3.2.4 Telescope Offload Output Custom Handler

The RTC will send a command to the telescope at a pre-defined interval rather than every frame. This type of offload is often sent to the Telescope Control System (TCS) and, in the case of GNAO, the TT is sent to the Secondary Control System (SCS). The standard HEART Telescope Offload block will call a custom handler function with the following signature:

daoErr_t telOffloadDriver(int numOffVals, const float * telOffVals);

The custom handler function will convert the data to the appropriate format for the hardware controller and send it to the appropriate telescope offload system. During initialization of the Telescope Offload block, the custom handler function is installed as the Telescope Offload output function. This allows the custom handler function to be called via a function pointer when the offload vector computed by HEART is ready to be sent to the telescope.

The input values are:

- numOffVals: number of modes in the telOffVals array
- telOffVals: telescope offload mode values.

Return value:

- daoErr_t: Defined at the beginning of this section.

It is expected that the telescope controller will reject out of range position demands by holding current position.

3.2.4.1 Example Telescope Control Custom Handler

The example Telescope Control Custom handler function in HEART takes the tip/tilt values stored in telOffVals and sends the commands in UDP datagrams with a data format list in Table 3-11.

Table 3-11 – Telescope Offload Command Data Format

Size (bytes)	Data Type	Description
4	uint32_t	Number of offload modes (two in this example)
4	TBD	Telescope tip/tilt offload

This example custom handler assumes the telescope offload is going to a UDP socket, not a syncrobus.

No response is expected from sending the telescope offloads.

3.2.5 Externally Streamed Telemetry using Data Socket

The RTC will use a separate processing thread to send decimated telemetry data to a single external device (e.g. server). The message socket communication will utilize a TCP connection. Details of the message (MSG) format are found in Section 3.4. The external telemetry server will accept a connection request from the telemetry client on the configured port. Only a single

connection is accepted at a given time. If the TCP connection is lost, the server will accept a new connection.

The server will only send data when instructed to by the System Controller. The client reads the header and verifies the package type and format, as defined in Section 3.4. The identifier in the header specifies the type of data. This information is then used to cast the following data into the appropriate structure. No explicit acknowledgment package is sent back in response.

In order to minimize timing jitter on the real-time wavefront correction, the telemetry stream does not share a physical Ethernet device with any of the Ethernet devices used in the hard real-time data streams. If necessary, the RTC command handler and telemetry stream can use separate physical Ethernet devices.

The following types of data can be streamed:

- Pixel intensities, gradients, subaperatures (LGS, OIWFS)
- DM, TTS, and FSM commands
- OIWFS/NGS WFS modes

The structure of the data is shown in Table 3-12 with sizes dependent on the configuration of the system. Data dimension are included in the header.

Table 3-12 - Data Types to be Streamed and structure

Type of data
Pixels intensities
Gradients
Subaperatures
DM commands
TTS commands
FSM Commands
Modes (OIWFS/NGS)

The decimation is configurable.

3.3 STANDARD INPUT BLOCKS

The standard input blocks are used to read either pixels or centroids. There is a standard block for each of these devices. Each of them have custom handler interfaces and are defined in the following sections.

3.3.1 Input WFS Custom Handler Interface

The Input WFS block calls a custom handler function to trigger the downstream block at the start of the arriving data for a frame. The standard HEART Input WFS block will call a custom handler function with the following signature:

```
daoErr_t wfsReader(uint16_t * dest, int width, int height, wfsReadout_t * progress);
```

This device specific custom handler function accepts a WFS target specifier, the dimensions of the readout, and a progress indicator. It reads the device specific input stream, store it in memory in a standard format and then uses HEART functions to inform the RTC when data is available. The custom handler will be repeatedly called until the entire frame is read, repeating until the Input WFS block is told to stop. The block configuration, available to the custom handler during initialization, will include information such as the frame size for each frame.

The custom handler will read the pixels/gradients and put the amount of expected data into the destination pointer location. If the incoming pixel stream does not match the dimensions, then the reading function will read until the total amount is reached for the frame and store the data within the provided buffer. After each data chunk is read and returned, the progress data structure is updated by the custom handler. Once validated, or if errors occur while reading, then the wfsReadout_t structure defined below will be returned.

The input values are:

- dest : pointer to where the data is written
- width: X number of columns read
- height: Y number of rows read
- progress: Structure that indicates progress for the frame

Returns value:

- daoErr_t: Defined above

The wfsReadout_t struct indicates the progress of the reading of the data. It includes:

- Total number of frames read (WFS frame number, increments by 1 every time the WFS starts imaging)
- Number of pixel (gradient) values read for this frame
- Sequence number within WFS frame. The sequence number is reset to zero for the first datagram of each WFS frame.
- Width
- Height
- Timestamp (nanoseconds since epoch) when the read started

3.3.1.1 Standard WFS Format

The WFS standard format is in an array with

- 2D image, represented as a 1D raster array
- the X/Y (0,0) location corresponding to the bottom left hand corner

3.3.1.2 Example WFS Custom Handler

The communication with a Wavefront Sensor in the custom handler can be implemented via either a socket or using the GigE Vision Stream Protocol (GVSP). From the RTC's perspective, all that is required is that the data being received is translated into a the format it can understand. The example WFS Custom handler function in HEART takes pixels/gradients and stores that into memory where they are processed by the downstream blocks.

The example datagram format for incoming wavefront sensor pixels is shown in Table 3-13.

Table 3-13 – Standard WFS Pixels Datagram Format (UDP)

Size (bytes)	Data Type	Description
2	uint16_t	WFS source identifier
2	uint16_t	N_{pix} : Number of pixel (gradient) values included in current datagram.
2	uint16_t	Datagram sequence number within WFS frame. The sequence number is reset to zero for the first datagram of each WFS frame.
2	uint16_t	Number of datagrams per WFS frame.
2	uint16_t	Full image width in pixels (gradients).
2	uint16_t	Full image height in pixels (gradients).
2	uint16_t	Datagram image width in pixels (gradients).
2	uint16_t	Datagram image height in pixels (gradients).
4	uint32_t	Raster scan pixel (gradient) index for first pixel (gradient) within datagram.
4	uint32_t	WFS frame number The frame number will be incremented by 1 every time the WFS starts imaging in response to a frame trigger signal.
8	uint64_t	Timestamp (nanoseconds since epoch).
2 (or 4) * N_{pix}	uint16_t or float	16 bit unsigned pixels or 32 bit floats (e.g. centroids) within datagram are sent in raster scan order. Packed 12 bit pixels are supported which allows storing two pixels per three bytes. These will be unpacked into 16 bit unsigned integers.
4	uint32_t	32 bit checksum

3.4 COMMAND, STATUS, DATA AND MESSAGE TCP SOCKETS

Command sockets are the primary connections dedicated to issuing commands (CMD), receiving command acknowledgments (ACK) and command completion messages (MSG). Status sockets are used to send and receive state and status updates (MSG, same as command completion messages) or data messages (DAT). For these kinds of connection, the primary role is to manage the control communication throughout the RTC system. Command sockets communicate using a

TCP connection and follow the communication protocol outlined below. All internal and external command communication within the RTC will be performed using command sockets. There are one external and three internal command connections detailed below. There are also status connections paralleling the command connections, intended for asynchronous and/or unsolicited messages, such as state or status changes or variable updates.

The System Controller to RTC Command Handler connection socket is the main entry point of communication for the RTC. This is the sole control interface point that allows high level command and control of the RTC from an external system. For the following text, the System Controller is the Command Sender, and the RTC is the Command Receiver.

3.4.1 Standard Header and Footer

Each command, message or data package will start with a fixed length header containing meta-data, as shown in Table 3-14

Table 3-14 - RTC Message Header

Size (bytes)	Data Type	Name	Description
4	uint32_t	**magicNum**	Magic number used to verify start of message/command.
4	uint32_t	**id**	Command (or message) identifier. This specifies the command (or message type) but not the parameters (message content). The interpretation of the payload data is determined by this identifier.
4	uint32_t	**size**	Size of data payload (in bytes).
4	uint32_t	**runId**	Identifier for data instance (e.g. sequence number within data stream).
16	timestamp	**time**	Linux time stamp (e.g. arrival time of last pixel in image).
2	uint16_t	**type**	Type of message (i.e. command(CMD), message(MSG), ack(ACK), data(DAT)).
2	uint16_t	**devNum**	Device number (if applicable; e.g. LGS WFS A).
2	uint16_t	**footer**	Does the message include a footer?
2	uint16_t	**check**	Type of checksum used (if footer exists).
40			Total

If required, the header is followed by a data payload and an optional footer. The format of the optional footer is shown in Table 3-15.

Table 3-15 - RTC Message Footer

Size (bytes)	Data Type	Description
4	uint32_t	Magic number used to verify end of message/command.
4	uint32_t	Command (or message) identifier. This must match the identifier in the header.
4	uint32_t	32 bit checksum (0 if no checksum).

Table 3-16, Table 3-17, and Table 3-18 list the equivalent C structs for the package header and footer.

Table 3-16 RTC Message Magic Number (C struct)

```
typedef union {
    char      str[4];
    uint32_t  num;
} hrtCmd_magic_t;
```

Table 3-17 - RTC Message Header (C struct)

```
typedef struct {
    hrtCmd_magic_t magic;  // "HRT" : 'H', 'R', 'T', 0
                           // 0x48, 0x52, 0x54, 0x00 (network byte order)
                           // 0x00545248 = 5526088 (Intel/AMD 32 bit value)
    int32_t  cmd;      // 32 bit command/message id
    uint32_t  payload; // size of payload in bytes
    int32_t  seqNum;   // identifies instance of data
    struct timespec timestamp;  // time
    int16_t  msgType;  // Command, Ack, Message, Data
    int16_t devNum;  // Specific device of given type (e.g. LGS A).
    uint16_t footer;  // Is a footer included?
    uint16_t chksumType;  // Type of checksum included in the footer?
} hrtCmd_header_t;
```

Table 3-18 RTC - Message Footer (C struct)

```
typedef struct {
    hrtCmd_magic_t magic;  // "hrt" : 'h', 'r', 't', 0
                           // 0x68, 0x72, 0x74, 0x00 (network byte order)
                           // 0x00747268 = 7631464
    int32_t  cmd;      // 32 bit command id, should match header
    uint32_t checksum; // Checksum value (0 if no checksum).
} hrtCmd_footer_t;
```

A package header and any proceeding data are sent in lock step. If the size of proceeding data is set to a non-zero value, then the receiver must read the specified number of bytes and the footer (if applicable), before attempting to read a new package header. If the specified data size is zero, then no additional data bytes are transmitted (no footer is sent if there is no data payload).

If any errors are detected,

- i.e. an invalid magic number was detected in the header or footer, an invalid package type or identifier is detected, or a size mismatch between the specified and expected size of data,

the receiver will discard the message and immediately close the socket. Both sides will then attempt to reestablish communication.

This protocol allows for packages of different types and sizes to be sent over the same socket.

Although the commands sent by the AO system controller are specified as integer values, HEART will contain a lookup table to allow translation between command (message or data structure) names and numeric identifiers.

3.4.2 Standard Status Reply

Status values can be requested, subscribed to or unsubscribed to. When requesting this a reply is sent back as shown in Table 3-19.

Table 3-19 - Standard Status request reply

[comp].[GROUP_]status		
Attribute	Type	Description
requestType	Enum	Type of request can be to be the current value, to subscribe to one or more status items, or unsubscribe to those CURRENT | SUBSCRIBE | UNSUBSCRIBE
args	string	String representation of the status request (may be a single status name, or multiple)
caller	string	identifier of the caller (if available)
runId	int	A run ID associated with the status request.
comp	enum	status of completion: SUCCESS | FAILED | REJECTED
compMsg	string	Completion string explaining why a command is FAILED, or REJECTED. This string will be empty if the command is SUCCESS.

3.4.3 Downstream AO streams

The HRT can provide pixels and DM shapes to downstream AO systems.

The data format package type (DAT) is used to transport complex data structures between servers. The identifier (id) specifies the data type of the following data and the data size specifies

the data type's size. The identifier and corresponding data type size are fixed and shared between the two servers via a common header file.

The DAT message type will be used for downstream AO, with the only different being that it will utilize UDP over TCP, because this is a direction connection.

3.5 CHECKSUM VARIANTS

HEART supports three distinct 32 bit checksums:

- A simple 32 bit XOR (exclusive or) checksum is currently specified for use in RTC when sending tip/tilt values to either the tip/tilt stage electronics or the laser guide star facility fast steering mirrors.
- A 32 bit Fletcher checksum can be used as an alternative if desires as it can quickly calculated with a simple function. It provides better error detection than the XOR function and is a good solution for protecting short datagrams such as those using XOR in RTC.
- The RTC uses a more complex 32 bit CRC32C checksum for larger datagrams and larger volumes of data such as DM actuator values and WFS pixels.

The XOR checksum was chosen since it is very trivial to implement and data errors are expected to be very rare within the short datagrams where its use is specified.

The Fletcher checksum is simple to implement and therefore does not introduce much development burden on the device side of the interface. It provides better error detection than the XOR checksum (https://users.ece.cmu.edu/~koopman/pubs/maxino09_checksums.pdf).

A 16 bit (2 * 8 bit) Fletcher (using 1's complement arithmetic) detects all two bit errors which are separated by less than 2040 bits.

https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/TC-14-49.pdf

https://dl.acm.org/doi/abs/10.1145/53644.53648?download=true

The more powerful CRC32C checksum was selected as an appropriate checksum during the RTC design stages. The CRC32C checksum is widely used, is effective at detecting errors and benefits from hardware acceleration in modern CPUs.

Regardless of which checksum is used, on the transmitting side, the checksum must be computed after converting all multi-byte data elements in the transmission buffer into network byte order. On the receiving side, the checksum must be computed on the received data, prior to conversion to host byte order. For the Fletcher checksum, which performs 16 bit additions, the byte pairs must converted to host order to allow proper 16 bit arithmetic and handling of overflow.

3.6 EXTERNALLY STREAMED TELEMETRY

There are four primary types of telemetry data that will be provided:

- State/Status Telemetry: provide System Controller with status, and other offload values. It is a socket stream over external network. It is non/soft real-time and is provided by the Command Handler (see Section 5 for details, including status provided).

- Stored Telemetry: for engineering purposes, locally written to disk mounted by the System Controller and is non real-time
- External Streamed Telemetry: for external displays or other applications, decimated socket stream over external network and is soft real-time
- Downstream AO Telemetry: for downstream AO correction (e.g. GIRMOS), full-rate socket stream over a dedicated link, and is hard real-time

4. COMMAND STRUCTURE, COMMANDS

The commands accepted by the RTC Command Handler are detailed in Section 4.3 and the details of the structure of the command is detailed in Section 3.4. The status reported by the RTC is detailed in Section 4.2.

4.1 COMMAND STRUCTURE

The following sections defined the different types of commands, command properties, and discusses the standard commands that blocks and pipelines respond to.

4.1.1 Command Types

Most of the commands can be divided into three command types:

- **Simple commands**: Immediate type commands that cover a broad category encompassing activities that complete in a short amount of time (e.g., setting flags, changing modes, performing self-tests, enabling debugging features). The caller can block while awaiting the response as it must occur within 1 second to avoid a timeout.
- **Discrete commands**: Submit type commands that are expected to produce a single discrete change in the state of an assembly, possibly requiring a significant amount of time (e.g., moving a motorized stage between two different positions).
- **Following commands**: Submit type command that initiates continuous "following" of externally generated demand streams. A typical use case is a motorized device following continuous position demands generated by the TCS. A command completion response is generally sent to the caller when the following device is initially on target (in some cases the component developer may include a settling time to ensure that the completion response is not sent until any ringing has subsided).

4.1.1.1 Simple Command

The simple command category is fairly self-explanatory, and can usually be achieved using a blocking **immediate** message as shown in Figure 4.

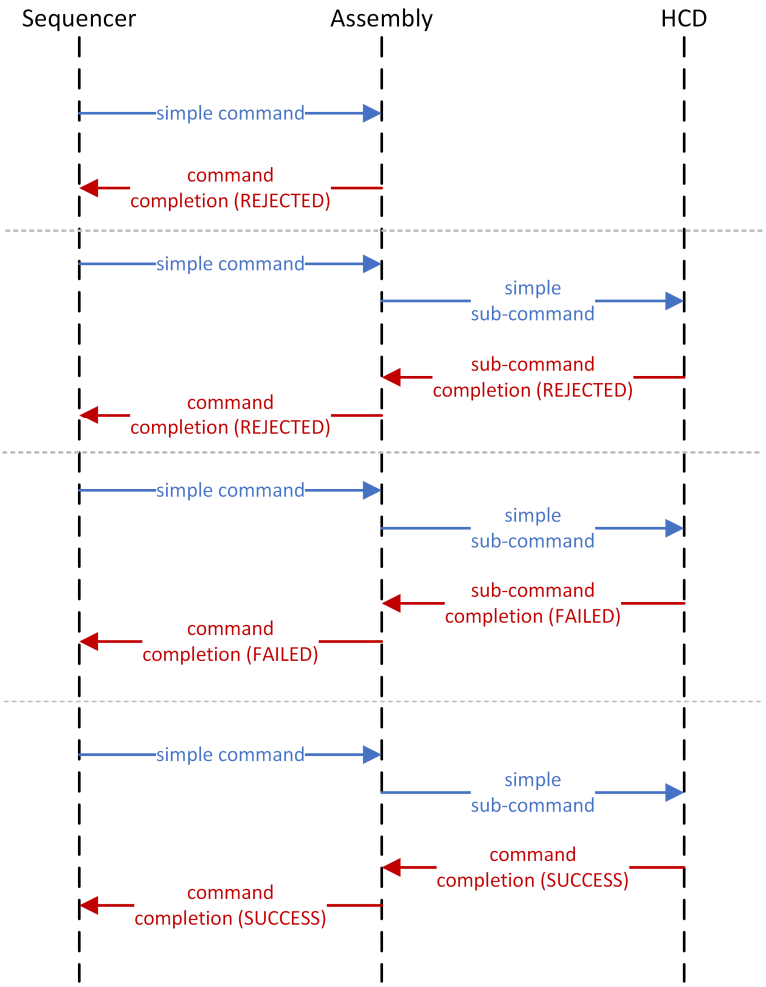

Figure 4 - Simple Command Flow Diagram

Simple commands will block until complete; any subsequent command messages will be queued within the command handler, until the simple command returns.

4.1.1.2 Discrete Command

The main difference between a simple command and discrete command is that the discrete commands can reasonably be expected to send completion messages back to the caller after some period of time as shown in Figure 5. Therefore, discrete commands will be sent using **submit**.

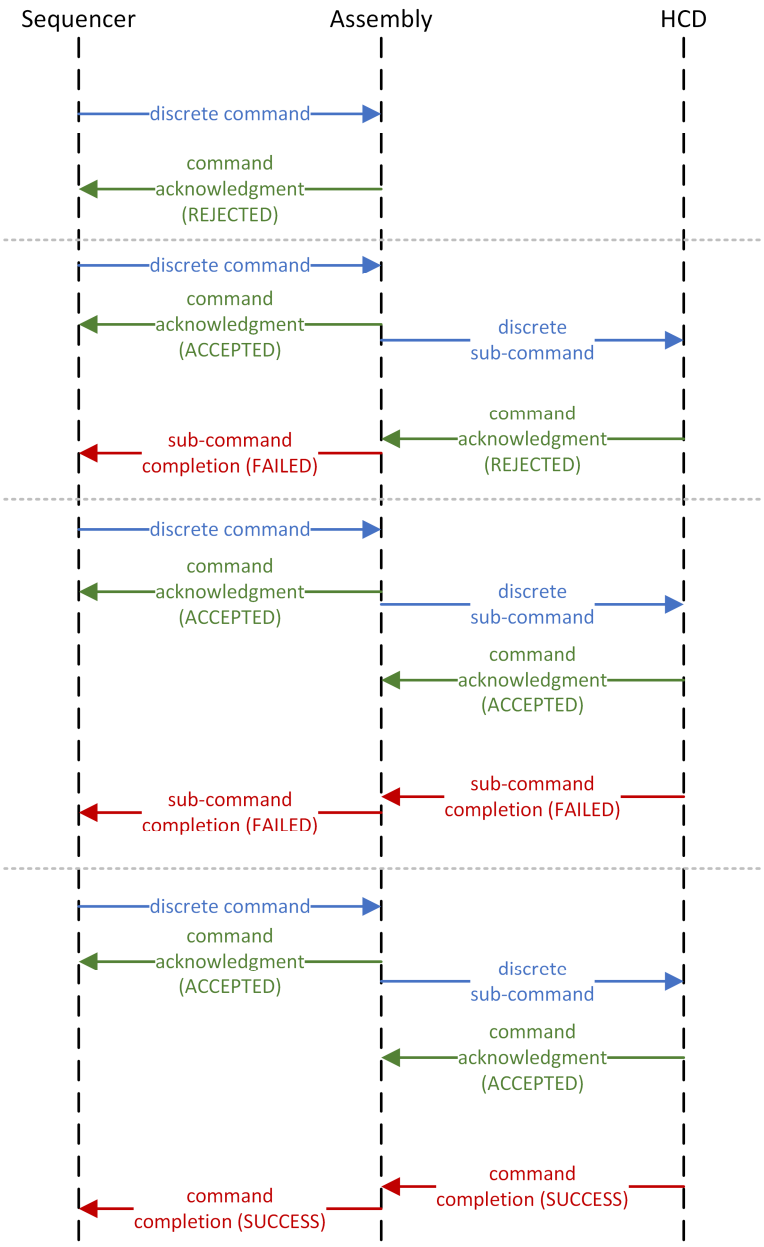

Figure 5 - Discrete Command Flow Diagram

When a discrete command is initially received, it blocks until the acknowledgment response is returned; any subsequent command messages during this period will be queued until the acknowledgment is sent. After the command is accepted, the acknowledgment is sent, and the command is executing, any subsequent command messages will be immediately checked and either accepted or rejected. If the new incoming command is rejected, the first command completes normally. If the new command is accepted, and the first command is pre-emptible, the

first command is cancelled. In the case were the first command is not pre-emptible, the new incoming command will be rejected. If there are new commands queued, the process will repeat.

4.1.2 Command Properties

Some commands will have special properties that will dictate how they behave and how they interact with other commands and the current commands state. The possible properties are:

- pre-emptible – a discrete command that can be interrupted by other commands
- friendly – a simple command that can be executed while another command is active, without preempting it
- scheduled – a discrete command that is acknowledged right away, but whose execution is delayed until a specified time.

4.1.3 Standard Commands

This section lists the standard command sets (that should then be extended on a per-component basis), and discusses the types of activities initiated by commands. A base command set is defined for most blocks and pipes. These are further detailed in Section 4.4.

Table 4-1 - Standard assembly command set

Name	Command Type	Block/Pipe/ Process	Description
config	discrete	Process	Reload configuration files only, set default conditions, etc. This command will be triggered internally, with default parameter as a part of the lifecycle initialization
start	simple	Pipe	A pipe starts all blocks in its pipeline
stop	simple	Pipe	A pipe stops all blocks in its pipeline
pause	simple	Pipe	any loops are closed then they will be opened and not clear any current state
resume	simple	Pipe	any loops that were closed and currently open then they will be closed and not clear any current state
destroy	simple	Pipe	All worker threads in the Blocks are stopped
init	simple	Block/Process	Re-initialize all internal variables to ….. the default ,re-read configuration files
debug	simple	Block/Pipe/Process	Set the logging debug level
shutdown	simple	Block/Pipe/Process	Stop all processes and exit

4.2 COMMAND STATUS AND STATE

4.2.1 Standard States

Components (such as Blocks, Pipes or Processes) will constantly publish information indicating what they are doing, whether they are able to accept commands, and any problems that may have occurred.

The effective state of the command handler cannot easily be represented by a single state value. A more accurate state description can be constructed by defining the state of a component as an N-tuple of sub-states.

This defines the component's state in terms of N-dimensions of state-space (analogous to a state-space model). All components will share some common sub-states (e.g. a command state) but would have additional sub-states to fit the component's needs.

State is reported outside the RTC via status.

4.2.2 Command State Attribute

The *cmd* (or command) state attribute represents the state of the component with respect to command handling as shown in Figure 6 Note that a REJECTED command does not change the *cmd* state.

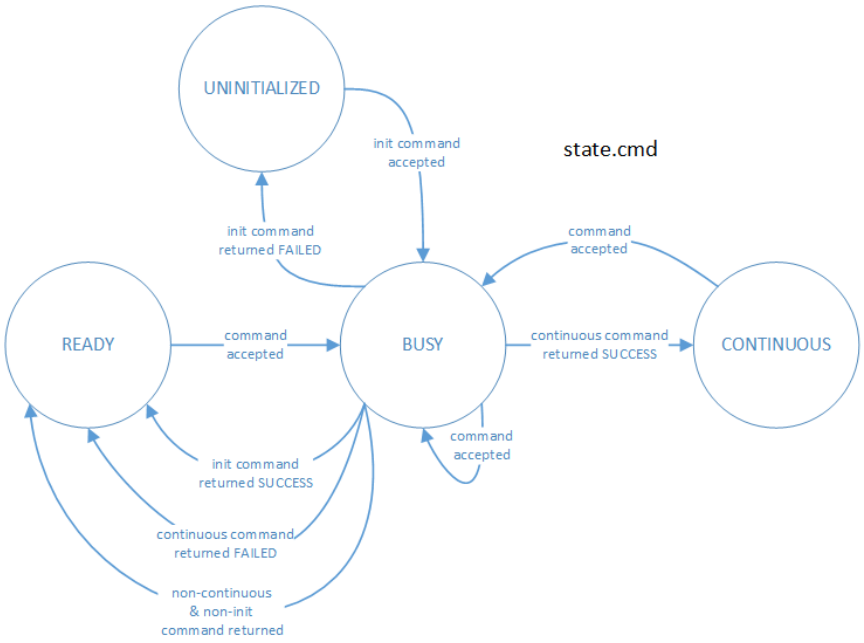

Figure 6 - Command State Attribute Diagram

The four states have the following meanings:

- **UNINITIALIZED**: the component is not properly initialized. Generally the only command that will be accepted in the state is init.
- **READY**: the component is ready to accept commands and is not processing / executing a command, and any background operations are not active.
- **BUSY**: the component is currently processing / executing a command. Generally only some commands will be accepted while in this state.

- **CONTINUOUS**: The component has accepted a continuous-type (loopClose) command and the component is currently still in the continuous mode (i.e. taking continuous action or periodic action such as a follow mode or tracking). This state is somewhere between ready and busy. Generally, only some commands will be accepted while in this state.

4.2.3 Standard Command Status

Command status is not the same thing as state. Its value indicates the response to a command. It can also be reported via status publishing.

4.2.3.1 Recent Command Status

In addition to the status of commands actively being run, another important piece of information for a component to publish is its most recent responses to commands specific command. This is particularly important where a chain of functions is being executed. In general, commands, command acknowledgements, and command completion status are sent between components using peer-to-peer connections. In order to improve debugging capabilities and to provide information for GUIs, eavesdropping by a third party can be facilitated by publishing command state, on a command basis. It will follow the naming convention [comp].[GROUP_]cmdStatus, where GROUP is the command name, and comp is the name of the components.

Table 4-2 - Standard RTC command status

[comp].[GROUP_]cmdStatus		
Attribute	Type	Description
cmd	string	Name of the most recent command processed by the command handler
args	string	String representation of the command arguments.
caller	string	identifier of the caller (if available)
runId	int	A run ID associated with the command.
ack	enum	initial command acknowledgement: ACCEPTED | REJECTED Always set to ACCEPTED for simple commands
ackMsg	string	Acknowledgement string explaining why a command is REJECTED. This string will be empty if the command is ACCEPTED. Always an empty string for **simple** commands.
comp	enum	status of completion: INPROGRESS | SUCCESS | FAILED | INTERRUPTED | REJECTED The REJECTED status only applies to **simple** type commands, since they do not

		send an acknowledgement, yet the command may still be invalid and therefore rejected.
compMsg	string	Completion string explaining why a command is FAILED, INTERRUPTED or REJECTED. This string will be empty if the command is INPROGRESS or SUCCESS.

This command status is published when it changes so that information can be provided for a GUI. When a critical failure occurs the appropriate alarms will be raised. If the critical failure was triggered by a command, the *compMsg* will also report a relevant error message.

When a command is INPROGRESS and a second command attempts to preempt the command but is rejected, either because the new command is invalid or the current command is not pre-emptible, the component will update the ackMsg with the REJECTED and will not affect the comp status of INPROGRESS. This provides a history of event in the event stream while maintaining only one command active at a time.

When a command is INPROGRESS but is successfully preempted, the assembly will first update this event item, reporting that the preempted command was INTERRUPTED, before updating it with the new commands details (e.g., new runId).

When a command is INPROGRESS and a second friendly command is received, the component must briefly update this event item with the friendly command's busy and completion status and then restore the currently active command to INPROGRESS. This provides a history of event in the event stream while maintaining only one command active at a time.

If the 'validate' flag is set for a command, i.e. the assembly does not actually execute the command, it only checks if the command is valid, then [comp].[GROUP_]*cmdStatus.ack and ackMsg are* updated, meaning there is event record of 'validate' commands.

4.3 COMMANDS ACCEPTED BY THE RTC COMMAND HANDLER AND STANDARD COMMANDS

The following sections detail the commands accepted by HEART. As each RTC varies in it's capabilities and functionality, it is expected that there will be custom commands required for any custom RTC. Those would be detailed in an interface control document that is specific for that RTC and would augment the list below. If an RTC does not have a need for a specific standard function, for example, if it didn't have an NGS WFS, then any commands relevant to NGS would be a **NOOP**. Standard commands that most blocks and pipelines respond to are called Standard commands and they are detailed in Section 4.4.

The commands are sent to the Command Handler using TCP sockets described in Section 3.4. In the message header the following information is given below in each of the tables for each of the commands:

Table 4-3 - Command structure

Name in RTC Message Header	Name in the Command Tables below	Description
magicNum	HRT	Magic number used to verify start of message/command.

id	**Command Name**	Command (or message) identifier. This specifies the command (or message type) but not the parameters (message content). The interpretation of the payload data is determined by this identifier.
size	size of the **Parameters(in)**	Size of data payload (in bytes).
runId	Incremented number	Identifier for data instance (e.g. sequence number within data stream). This is returned in the reply
time	Time now	Linux time stamp (e.g. arrival time of last pixel in image).
type	**CMD**	Type of message (i.e. command(CMD), message(MSG), ack(ACK), data(DAT)).
devNum	N/A	Device number (if applicable; e.g. LGS WFS A).
footer	No	Does the message include a footer?
check	N/A	Type of checksum used (if footer exists).
Payload:		
Format for payload is detailed in the “Parameters(in)” section of the tables below. Examples of the payload are: For an integer: “-config mode.lgsEnable=6” For an array of something: “-config mode.lgsEnable={1, 2, 3, 4}” For a string: “-config calibLgsBackground.filename=a/b/c/d.fits”		

The reply is in the format that follows.

Table 4-4 - Reply to a command structure

Name in RTC Message Header	Name in the Command Tables below	Description
magicNum	HRT	Magic number used to verify start of message/command.
id	**Command Name**	Command that was sent

size	size of cmdStatus (Table 4-2)	Size of data payload (in bytes).
runId	Number given when the command was sent	Identifier for data instance (e.g. sequence number within data stream). This is returned in the reply
time	Time now	Linux time stamp (e.g. arrival time of last pixel in image).
type	**ACK**	Type of message (i.e. command(CMD), message(MSG), ack(ACK), data(DAT)).
devNum	N/A	Device number (if applicable; e.g. LGS WFS A).
footer	No	Does the message include a footer?
check	N/A	Type of checksum used (if footer exists).
Payload:		
cmdStatus structure (Table 4-2)		
cmd		Name of the most recent command processed by the command handler
args		String representation of the command arguments.
caller		identifier of the caller (if available)
runId	runId above	A run ID associated with the command.
ack	ACCEPTED	initial command acknowledgement: ACCEPTED | REJECTED
ackMsg	Empty	Acknowledgement string explaining why a command is REJECTED. This string will be empty if the command is ACCEPTED. Always an empty string for **simple** commands.
comp	status	status of completion: INPROGRESS | SUCCESS | FAILED | INTERRUPTED | REJECTED
compMsg	String, if needed	Completion string explaining why a command is FAILED, INTERRUPTED or REJECTED. This string will be empty if the command is INPROGRESS or SUCCESS.

4.3.1 mode

Description	sets the AO mode of the RTC				
Type	simple				
Command Name	**mode**				
Parameters (in)	Field	Description	Type	Units	Required
	lgsEnable	If an enable flag is true then the RTC will use the corresponding pixel stream for high-order correction. For NGS or SL mode, all LGS WFSs should be disabled. The array is ordered LGS WFS [A B C D E F].	array[6] of boolean		yes
	rateHo	high-order loop rate	double ($0.1 \leq x \leq 800$)	Hz	yes
	rateLo	low-order loop rate	double ($0.1 \leq x \leq 800$)	Hz	yes
	rateLot	low-order truth loop rate	double ($0.1 \leq x \leq 400$)	Hz	yes
	rateWc	wavefront corrector loop rate	double ($0.1 \leq x \leq 800$)	Hz	yes
	ngsBin	width of NGS bin in pixels	enum: (1, 2, 4, 8, 16)		no
	ngsMode	NGS mode. The array is ordered NGS [A B C D E F G]	array[7] of enum: (NONE, TT, TTF)		yes
	tier1	Tier 1 detector.	enum: (NGSA, NGSB, NGSC, NGSD, NGSE, NGSF, NGSG)		
	tier2	Tier 2 detectors. The array is ordered Tier 2 [A B]	array[2] of enum: (NGSA, NGSB, NGSC, NGSD, NGSE, NGSF, NGSG)		yes
	tier3	Tier 3 detectors. The array is ordered Tier 3 [A B C D]	array[4] of enum: (NGSA, NGSB, NGSC, NGSD, NGSE, NGSF, NGSG)		yes
	pol	indicates whether to perform pseudo open-loop (POL) feedback	boolean		yes
Parameters (out)					

Status values affected	Precondition: state.cmd = READY Execution: state.cmd = BUSY At Completion state.cmd = READY mode.rateLo = {input rateLo} mode.rateLo = {input rateLo} mode.rateLot = {input rateLot} mode.rateWc = {input rateWc} mode.tier1 = {input tier1} mode.tier2[] = {input tier2} mode.tier3[] = {input tier3} mode.tier3f = {input tier3f} lgsState.enable[] = {input lgsEnable} ngsState.bin = {input ngsBin} ngsState.enable[] = {input ngsMode}

Action and Response

On receiving this command the RTC will deactivate the RTC pipeline, via the pipelineDeactivate command and (re)-configure the AO mode of the RTC. This command may also (re-)established the private communication with the SRT-RPG by creating the mode appropriate connection end-points. Once the AO mode has been configured, the RTC will begin to listen to pixels streams and perform pixel calibration for quick-look GUIs. However, it will not be saved or processed further.

The command will also configure which detector input streams will be used for low-order correction. These streams are split into tiers as listed below:

- Tier 0 (optional) NGS providing TT(FA) measurements for acquisition purposes
- Tier 1 (required) single detector that provides LO TTF(A) measurements
- Tier 2 (optional) 1 to 2 detectors that provide LO TT or LO TTF(A) measurements
- Tier 3 (optional) 1 to 4 detectors that provide LOT TT measurements
- Tier 3F (optional) single detector that provides LOT focus measurements
- The restrictions for Tier assignments are given below:
    - Tier 0 can only be the NGS
    - Tier 1 must be either the NGS or a TTF OIWFS
    - Tier 2 can be the NGS or any TT/TTF OIWFS or ODGW
    - Tier 3 can be any OIWFS or ODGW
    - Tier 3F can only be a TTF OIWFS
    - a single detector can be assigned to multiple tiers
    - the same detector cannot be assigned to the same tier twice
    - the same detector cannot be assigned to both Tier 1 and Tier 2
- For Tiers 2 and 3, the order in which the detectors are specified defines the order in which the gradients will be assembled for low-order and low-order truth mode reconstruction. If a tier entry is specified as NONE, then that entry is skipped while assembling the low-order gradient vector (changes the size of the gradient vector)

- The wavefront corrector rate must be a multiple of the high-order loop rate and the steady-state low-order loop rate. The steady-state low-order loop rate must be a multiple of the steady-state low-order truth loop rate.
- Any OIWFS that is assigned to Tier 1 or Tier 3F must be set to TTF mode, while any OIWFS that is assigned to Tier 2 or Tier 3 can be either set to TT or TTF mode. All OIWFSs not assigned to a low-order tier must be set to the NONE mode.
- Logic:
    - if {input lgsEnable[*] == false}, then loop.lgsTt[*] = INACTIVE
    - if {input tier2[] != ODGW* and input tier3[] != ODGW*}, then odgwState.enable[ODGW*] = false else odgwState.enable[ODGW*] = true
    - if {input tier2[*] == NONE}, then loop.tier2[*] = INACTIVE
    - if {input tier3[*] == NONE}, then loop.tier3[*] = INACTIVE
    - if {input tier3f == NONE}, then loop.tier3f = INACTIVE

4.3.2 calibBackground

<table>
<tr><td>Description</td><td colspan="5">instructs the RTC to take new normalized WFS sky & instrument background and/or save the background to the file, used for engineering purposes</td></tr>
<tr><td>Type</td><td colspan="5">simple</td></tr>
<tr><td>Command Name</td><td colspan="5">calibBackground</td></tr>
<tr><td rowspan="5">Parameters (in)</td><td>Field</td><td>Description</td><td>Type</td><td>Units</td><td>Required</td></tr>
<tr><td>det</td><td>Which detectors (HO or LO)</td><td>HO | LO</td><td></td><td>yes</td></tr>
<tr><td>avg</td><td>number of frames to average</td><td>integer (0 ≤ x)</td><td></td><td>no. (see notes)</td></tr>
<tr><td>filename</td><td>name of FITS file to save to</td><td>string</td><td></td><td>no. (see notes)</td></tr>
<tr><td>type</td><td>This could be DARK/SKY or FLAT</td><td>enum{ DARK_SKY | FLAT }</td><td></td><td>no. (see notes).</td></tr>
<tr><td>Parameters (out)</td><td colspan="5"></td></tr>
<tr><td>Status values affected</td><td colspan="5">Precondition:
state.cmd = READY
loop.ready = true
Execution:
state.cmd = BUSY
state.lgsBackgroud = true
At Completion:
state.cmd = READY
state.lgsBackgroud = false
if {input filename not specified}, then state.unsavedLgsSky = true
if {input filename is specified}, then state.unsavedLgsSky = false
if {input filename is different from current LGS sky config file}, then state.unsavedConfig = true</td></tr>
</table>

Action and Response

This command applies to all enabled LGS WFSs. If the number of frames to average is specified, then the RTC will average the specified number of frames to measure the current backgrounds or flat. The current detector backgrounds, specified in the configuration file, are subtracted from the measured backgrounds to produce the sky & instrument backgrounds. These sky & instrument backgrounds are then normalized to a 1 second exposure and stored as the current normalized LGS WFS sky & instrument backgrounds in the RTC (at which point the command returns). For a FLAT, the frames are averaged and saved to file. Subsequent call to this command will replace the normalized sky & instrument backgrounds and calls to the init command will reset all backgrounds to the defaults specified in the configuration file.

If the input filename parameter is specified, then the RTC will save the current normalized sky & instrument backgrounds to file using the configuration name given, after generating a new backgrounds if instructed to do so (i.e. number of frames to average is specified). Note that the configuration file will only contain the normalized sky & instrument backgrounds, and should be different from the configuration name used by the init command.

If neither input argument is specified, then this command will return with an error.

If pixels are not received from all enabled LGS WFSs in a timely manner, then this command will return with an error and no backgrounds will be updated or saved.

A call to the stop command will abort this process.

Discussion: It is assumed that the lasers will be detuned before executing this command so the sky & instrument backgrounds will include Rayleigh backscattering.

4.3.3 pipeline

Description	(re-)activate the RTC pipeline by starting pixel processing				
Type	simple				
Command Name	**pipeline**				
Parameters (in)	Field	Description	Type	Units	Required
	enable	indicates whether to enable or disable the pipeline	boolean		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY state.mode = true state.SRT-RPGDate = true Execution: state.cmd = BUSY At Completion: state.cmd = READY loop.ready = true | false calibWc.override = false calibDetector.pixelMode[] = STOP calibDetector.gradMode[] = STOP lgsState.algo = COG_STATIC | NONE | STOP				

	lgsState.tt = KALMAN ngsState.algo = COG_STATIC | NONE odgwState.algo[] = COG_STATIC | NONE ngsState.algo[] = COG_STATIC | NONE ngsState.acqTable[] = false odgwState.acqTable[] = false ngsState.algo[] = NONE

Action and Response

This command (re-)activates or deactivates the RTC pipeline. The activation will start pixel processing, including gradient computation using CoG, and will flattening the DMs (via the dmFlatten command), zero the TTS (via the ttsZero command), zero the LGS FSMs (via the lgsfFsmZero command), and resetting integrators and filters (via the loopParamReset command).

Subsequent re-activate calls to this command will re-activate the pipeline, opening all loops (via the loopOpen command), then reset parameters and flatten/zero controlled elements. This command will also start saving telemetry stream to the RTS and pixel streams to the HOP servers.

Note that published control loop specific telemetry or telemetry sent directly to the SRT-RPG will not be started with this command but will begin once the corresponding loop is closed.

If enable is set to false, then the command deactivates the RTC pipeline, i.e. stopping all closed-loop processing (via the loopOpen command), and stopping pixel processing.

The deactivate command is intended to stop the RTC data processing, i.e. at the end of a night. This command does not stop the RTC pipeline software but instead leaves the software in a stand-by state whereby the RTC continues to respond to commands and the pipeline can easily be reactivated. This command also stops the RTS and HOP servers from saving telemetry and pixel data.

Discussion: It is anticipated that pixel reading will always occur as long as the AO mode is set; it is therefore unaffected by this command. However, any pixels read while in this state are simply calibrated but not stored and will not be processed further.

4.3.4 loopOpen

Description	opens all control loops and offloading in the RTC				
Type	simple				
Command Name	**loopOpen**				
Parameters (in)	Field	Description	Type	Units	Required
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true Execution: state.cmd = BUSY At Completion:				

	state.cmd = READY
	loop.ho = IDLE
	loop.lgsFocus = IDLE
	loop.lgsDither = NONE
	loop.lgsTt[] = IDLE | INACTIVE
	loop.lo = IDLE
	loop.telOffloadTt = IDLE
	loop.telOffloadFocus = IDLE
	loop.telOffloadMag = IDLE
	loop.telOffloadMode = IDLE
	
	if (lgsState.algo == MF_UPDATE), then lgsState.algo = MF_STATIC
	else if (lgsState.algo == MF_COG), then lgsState.algo = COG_STATIC
	if (ngsState.algo == COG_UPDATE), then ngsState.algo = COG_STATIC
	if (ngsState.algo == MF_UPDATE), then ngsState.algo = MF_STATIC
	else if (ngsState.algo == MF_COG), then ngsState.algo = COG_STATIC
	if (odgwState.algo == MF_UPDATE), then odgwState.algo = MF_STATIC
	else if (odgwState.algo == MF_COG), then odgwState.algo = COG_STATIC

Action and Response

This command opens all control loops and offloading in the RTC.

Note that this command does not reset the integrator or filter states; use the pipelineActivate or loopParamReset command to reset those parameters.

4.3.5 loopLgsTt

Description	enables or disables the LGS TT loop that sends commands to control Laser FSMs				
Type	simple				
Command Name	**loopLgsTt**				
Parameters (in)	Field	Description	Type	Units	Required
	loop	indicates desired state of the LGS TT loop	enum: (OPEN, CLOSE, CLOSE_WO_DRIFT)		yes
Parameters (out)	state				
Status values affected	Precondition: state.cmd = READY loop.ready = true lgsState.enable[{at least one}] = true Execution: state.cmd = BUSY At Completion: state.cmd = READY if {input loop} = OPEN, then loop.lgsTt[] = IDLE | INACTIVEloop.lgsDriftTt = false if {input loop} = CLOSE, then loop.lgsTt[] != IDLEloop.lgsDriftTt = true if {input loop} = CLOSE_WO_DRIFT, then loop.lgsTt[] != IDLEloop.lgsDriftTt = false				

Action and Response

This command enables or disables the LGS TT loop that sends commands to the LGSF to control LGSF FSMs.

When the loop is set to OPEN, TT correction commands to the LGSF are not updated and the LGS TT loop is halted. However, the if Kalman filter and integrator outputs are not reset, the FSM will remain at their current position (however the internal Kalman filter states are reset); these outputs can be reset via the loopParamReset command. When the loop is set to CLOSE, the LGS TT loop is active and a TT drift terms are added to measured TT errors. When the loop is set to

CLOSE_WO_DRIFT, the LGS TT loop is active but TT drift terms are ignored.

While the LGS TT error is above the LGS FSM TT threshold (defined in the config file), the FSM TT loop state will be set to ACQUIRE. Once the TT error has dropped below the threshold, the FSM TT loop state will change to LOCK. If the LGS TT loop is not IDLE, an enable request will be a no-op.

The control command will only be applied to the LGSF FSM corresponding to the enabled LGS WFSs (lgsState.enable[]). LGSF FSMs corresponding to disabled LGS WFSs will remain in the IDLE state and will be sent zeros.

Discussion: The acquisition of LGS TT may only take a few frames for the FSMs to move to the correct positions and start guiding.

4.3.6 loopHigh

<table>
<tr><td>Description</td><td colspan="5">enables or disables the high-order loop which applies the high-order correction errors to the control path</td></tr>
<tr><td>Type</td><td colspan="5">Simple</td></tr>
<tr><td>Command Name</td><td colspan="5">loopHigh</td></tr>
<tr><td rowspan="3">Parameters (in)</td><td>Field</td><td>Description</td><td>Type</td><td>Units</td><td>Required</td></tr>
<tr><td>enable</td><td>indicates whether to enable or disable the high-order loop</td><td>boolean</td><td></td><td>yes</td></tr>
<tr><td colspan="5"></td></tr>
<tr><td>Parameters (out)</td><td colspan="5"></td></tr>
<tr><td>Status values affected</td><td colspan="5">Precondition:
state.cmd = READY
loop.ready = true
calibWc.override = false
Execution:
state.cmd = BUSY
At Completion:
state.cmd = READY
if {input enable}, then loop.ho != IDLE
if {input !enable}, then loop.ho = IDLE</td></tr>
</table>

Action and Response

This command enables or disables the high-order loop which applies the high-order correction errors to the control path.

While waiting for the loop to stabilize, the high-order loop state will be set to ACQUIRE. Once the loop has stabilized, the high-order loop state will be set to LOCK. If the high-order loop is not IDLE, an enable request will be a no-op.

Disabling the high-order loop does not reset the correction vector stored in the main integrator; it merely stops the high-order error vector. To reset the main integrator, use the command loopParamReset. Note that disabling the high-order correction does not interfere with the low-order path.

4.3.7 loopLow

<table>
<tr><td>Description</td><td colspan="5">enables or disables low-order loops using the low-order detectors, i.e. the LO TT/TTF detectors</td></tr>
<tr><td>Type</td><td colspan="5">Simple</td></tr>
<tr><td>Command Name</td><td colspan="5">loopLowTier</td></tr>
<tr><td rowspan="3">Parameters (in)</td><td>Field</td><td>Description</td><td>Type</td><td>Units</td><td>Required</td></tr>
<tr><td>enable</td><td>indicates whether to enable or disable the low order loop</td><td>boolean</td><td></td><td>yes</td></tr>
<tr><td>Parameters (out)</td><td colspan="4"></td></tr>
<tr><td>Status values affected</td><td colspan="5">Precondition:
state.cmd = READY
loop.ready = true
calibWc.override = false
Execution:
state.cmd = BUSY
At Completion:
state.cmd = READY
if {input enable}, then if {at least one mode.loOrder != NONE}, then loop.lo != IDLE, loop.tier0 = IDLE, loop.loOrder[] != IDLE
if {input !enable}, then loop.loOrder[] = IDLE</td></tr>
</table>

Action and Response

This command enables or disables the low-order loop which apples low-order correction errors to the control path from the low orderdetectors, i.e. the LO TT/TTF detectors.

Enabling this loop triggers the low order acquisition process. Once the acquisition process is complete the state is set to LOCK.

If enabling the low order loop but the detector is inactive (mode.loworder[] = NONE), then that detector is skipped and the corresponding loOrder state will be INACTIVE. If a loOrder detector is already acquiring or locked, this command is a noop for that detector and will proceed to the next loOrder detector.

If there is a handoff procedure, then that will need to be detailed in the custom command..

4.3.8 offloadTcs

Description	enables or disables offloading to the TCS				
Type	simple				
Command Name	**offloadTcs**				
Parameters (in)	Field	Description	Type	Units	Required
	m1Uncontrolled	indicates whether to enable or disable the primary mirror (M1) uncontrolled modes offload	boolean		no
	telComa	Telescope coma offload	boolean		no
	telM1Figure	Primary Mirror figure offload	boolean		no
	telOffloadTt	indicates whether to enable or disable the TT telescope offloading	boolean		no
	telOffloadFocus	indicates whether to enable or disable the focus telescope offloading	boolean		no
	telOffloadMag	indicates whether to enable or disable the magnification telescope offloading	boolean		no
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true Execution: state.cmd = BUSY At Completion: state.cmd = READY if {input telOffloadTt} then loop.telOffloadTt != IDLEelse loop.telOffloadTt = IDLE if {input telOffloadFocus} then loop.telOffloadFocus != IDLEelse loop.telOffloadFocus = IDLE if {input telOffloadMag} then loop.telOffloadMag != IDLEelse loop.telOffloadMag = IDLE if {input telOffloadMode} then loop.telOffloadMode != IDLEelse loop.telOffloadMode = IDLE				

Action and Response

This command enables or disables offloading to the TCS.

When enabled the RTC, it will publish various modes for the TCS (telOffloadTt, telOffloadFocus, telOffloadMag, and/or telOffloadMode).

Disabling the TCS offloading merely stops any offloading sent to the TCS, and does not impact any other pipeline operation.

At least one input argument must be specified.

4.3.9 subApMaskLgsSet

Description	update LGS WFS sub-aperture mask				
Type	simple				
Command Name	**subApMaskLgsSet**				
Parameters (in)	Field	Description	Type	Units	Required
	wfs	LGS WFS sub-aperture mask to update	enum: (LGSWFSA, LGSWFSB, LGSWFSC, LGSWFSD, LGSWFSE, LGSWFSF, ALL)		yes
	mask	sub-aperture mask	int [total # of sub-apertures]		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

This command updates the LGS WFS sub-aperture mask due to expected low-flux sub-aperture and disable sub-aperture based on Rayleigh backscattering.

The mask input a mask array of length equal to the total number of sub-apertures per LGS WFS if a specific WFS is selected by the wfs input. However if the wfs input specified as ALL, then the input mask array must be the total number of sub-apertures for all LGS WFS order as mask values for LGS WFS [A, B, C, …].

4.3.10 filterTemporalSet

Description	update the HO and/or LO temporal filters				
Type	Simple				
Command Name	**filterTemporalSet**				
Parameters (in)	Field	Description	Type	Units	Required
	hoOrder	Order of the HO temporal filter	int		no (see below)
	hoNumer	HO temporal filter numerator	float [hoOrder+1]		no (see below)
	hoDenom	HO temporal filter denominator	float [hoOrder+1]		no (see below)
	loOrder	Order of the LO temporal filter	int		no (see below)
	loNumModes	The number of LO modes	int		no (see below)
	loNumer	LO temporal filter numerator for each LO mode	float [loOrder+1] [loNumModes]		no (see below)
	loDenom	LO temporal filter denominator for each LO mode	float [loOrder+1] [loNumModes]		no (see below)
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

This command updates the HO and/or LO temporal filters.

If the hoOrder is specified then the hoNumer and hoDenom input must also be specified as a set and vice versa. The same is true for the loOrder, loNumModes, loNumer and loDenom input set. Additionally at least one of the HO filter and LO filter input value sets must be specified.

The HO and LO filter order, and number of LO modes must agree with respective values specified in the configuration file.

4.3.11 loopParamReset

Description	reset one or more loop parameters (calibration)				
Type	simple				
Command Name	**loopParamReset**				
Parameters (in)	Field	Description	Type	Units	Required
	enable	Name or name prefix of the loop parameter to reset. Passing an empty string will reset all loop parameters.	string		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

This command will reset one or more loop parameters. (calibration)

If a parameter name prefix is specified, any parameter with the matching name prefix will be reset. Passing an empty string will reset all loop parameters. The list of all possible loop parameters that will be reset by this command are:

- wc_int - wavefront corrector integrator (= 0)
- wc_tt_lpf - TTS filter states (= 0)
- wc_tel_lpf - Telescope offload high-pass filter states (= 0)
- ho_psd - High-order PSD & WFE (discard gathered statistics)
- ho_turb - turbulence parameters (discard gathered statistics)
- lo_mode_hpf - low-order (Tier 1 & 2) high-pass filter states (= 0)
- lo_mode_lpf - low-order truth (Tier 3 & 3F) low-pass filter states (= 0)
- lo_psd - Low-order PSD & WFE (discard gathered statistics)
- lgs_tt_int - LGSF FSM integrator (= 0)
- lgs_tt_psd - LGS TT PSD (discard gathered statistics)

4.3.12 paramConfigSave

Description	tells the RTC to save the current parameter configuration to specified parameter configuration file.				
Type	simple				
Command Name	**paramConfigSave**				
Parameters (in)	Field	Description	Type	Units	Required
	config	Name of the configuration file to save the current parameters to.	string		yes

Parameters (out)	
Status values affected	Precondition: state.cmd = READY Execution: state.cmd = BUSY At Completion: state.cmd = READY state.unsavedConfig = false

4.3.13 paramConfigSet

Description	tells the RTC to set the value of any configuration parameter, i.e. overriding a parameter that is specified in the parameter configuration file.				
Type	simple				
Command Name	**paramConfigSet**				
Parameters (in)	Field	Description	Type	Units	Required
	param	Configuration parameter identifier	string		no
	value	New value for the specified parameter as a string	string		no
	file	Full path to the file that contains the new data for the specified parameter	string		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = false Execution: state.cmd = BUSY At Completion: state.cmd = READY state.unsavedConfig = true				

Action and Response

This command tells the RTC to set the value of a configuration parameter, i.e. a parameter that is specified in the parameter configuration file. (calibration)

This command is intended for constructing a new parameter configuration file as well as for engineering and debugging purposes. The RTC will update the specified parameter, either by value or loading a file that contains the new value. The file format is dependent on the parameter specified. Note the new parameter is not automatically saved to the configuration service and will be lost if the init command is called. To signal this condition, the unsaved parameter flag is set. To save the current configuration, use the paramConfigSave command.

Exactly one of the value or file arguments must be specified. Consequently the required argument will be set based on the parameter identifier (i.e. smaller parameters must use value while larger parameters must use file).

4.3.14 dmShape

Description	Applies a shape to the selected DM or writes the DM shape to file				
Type	simple				
Command Name	**dmShape**				
Parameters (in)	Field	Description	Type	Units	Required
	dm	DM or DM's to apply the command to	enum: (DM0, DM1, DM2, DM0_and_DM1_and_DM2)		yes
	shape	What shape to assume – options are apply an offset from a file, apply a shape from a file, set to zero volts, set to system flat. Or a request to write a shape to file	enum: {SET_OFFSET_USE_FILE, SET_SHAPE_USE_FILE, SET_ZERO, SET_SYS_FLA, STORE_SHAPE_TO_FILE, T}		yes
	file	Name of file (include path) to read in or write to	string		See notes.
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true loop.ho = IDLE loop.lo = IDLE calibWc.override = false Execution: state.cmd = BUSY At Completion: state.cmd = READY dmShapeSet = shape input value dmShape_File = fileName				

Action and Response

This command applies a shape to the selected DM or writes a shape to a DM. This can apply an offset from a file, apply a shape from a file, result in a flat mirror, set to zero volts, or set to system flat. Or a request to write a DM shape to file.

A SYSTEM input corresponds to sending the system flat, as specified by the SRT-RPG, will be sent to the DM. A MIRROR input corresponds to sending a zero command to the DM, while a VIRGINIZE input means a decaying sinusoid is applied to the DM, result in a flat mirror. A NONE input means the DM will not be moved. This command is rejected if either the high- or low-order loops are closed, or if the DMs are in the override calibration mode.

If both inputs are set to NONE, then this command is no-op and will return a warning.

4.3.15 ttsSet

Description	sends a system zero or mirror zero or actuator command to the TTS				
Type	simple				
Command Name	**ttsZero**				
Parameters (in)	Field	Description	Type	Units	Required
	tts	TTS zero command	enum: (SYSTEM, MIRROR, MANUAL)		yes
	actVal	Actuator values to apply for Manual mode	float[2]		See notes.
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true loop.ho = IDLE loop.lo = IDLE calibWc.override = false Execution: state.cmd = BUSY At Completion: state.cmd = READY ttSet[2] = actuator values sent				

Action and Response

This command sends a system zero, mirror zero, or actuator values to the TTS.

A MIRROR input corresponds to sending a raw zero command to the TTS, meaning the stage is zeroed with respect to physical unit rotation relative to its mount as specified by the configuration file. A SYSTEM input means the system zero position, as specified by the configuration file, will be sent to the TTS. This corresponds to the position of the TTS that nominally aligns the system. In MANUAL mode, it will set the actuator values on the TTS. This command is rejected if either the high- or low-order loops are closed, or if the TTS is in the override calibration mode.

4.3.16 enableHrtFlags

Description	Enables or disables HRT inputs, outputs, or corrections				
Type	simple				
Command Name	**enableHrtFlags**				
Parameters (in)	Field	Description	Type	Units	Required
	enable	indicates whether to enable or disable the internal flag	enum: (start, stop, pause, resume, reset)		yes
	field	Indicates the flag	String		yes

Parameters (out)	
Status values affected	Precondition: state.cmd = READY loop.ready = false Execution: state.cmd = BUSY At Completion: state.cmd = READY

Action and Response

This command sends an enable or disable of the affected internal flags for engineering purposes. Examples are:

- setNgsCentroidingErr Enable/disable NGS centroiding error corrections
- setNgsFieldRotOffset Enable/disable NGS Field Rotation corrections to the System Controller
- setTipTiltOutput Enable/disable TTS output
- setAveTTtoSCS Enable/disable the average tip tilt corrections to the Secondary Control Mirror
- setNgsFieldDist Enable/disable the sending of field distortion modes to the high order loop
- setLgsFocusErrors Enable/disable the sending of LGS Focus errors to the System Controller from the SFS WFS
- setLgsHoWfs Enable/disable using LGS WFS for HO calculations
- setLgsCenteringErr Enable/disable the sending of LGS Centering errors to the Laser Fast Steering Mirrors
- setComaFocus Enable/disable the sending of coma and focus corrections to the System Controller for the Secondary Control System.
- setPrimaryFigure Enable/disable the sending of primary mirror figure corrections to the RTC System Controller
- setLoIntegrators Enable/disable/reset LO Integrator
- setHoIntegrators Enable/disable/reset HO Integrator
- loLoopCtrl LO Pipeline loop control (start, stop, pause, resume, reset)
- hoLoopCtrl HO Pipeline loop control (start, stop, pause, resume, reset)

4.3.17 enableSrtFlags

Description	Enables or disables SRT optimization inputs, outputs, or corrections				
Type	simple				
Command Name	**enableSrtFlags**				
Parameters (in)	Field	Description	Type	Units	Required
	enable	indicates whether to enable or disable the internal flag	enum: (start, stop, pause, resume, reset)		yes

	period	period at which new optimization parameters should be generated.	float	seconds	yes
	field	Indicates the flag	String		yes
	gsMag	Guide star magnitudes	Int		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = false Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

This command sends an enable or disable of the affected SRT internal. Examples are that will enable statistic gathering enable/disable:

- gradient optimization,
- weight optimization,
- loop gains

It will also enable/disable the creating of the SRT-RPG to create a set of control matrices based on the current configuration. After the new batch of statistics are send from the HRT to SRT-RPG, new control matrices are computed by SRT-RPG and send back to the HRT.

:

4.3.18 changeLoopRate

Description	Changes the loop rates on the fly				
Type	simple				
Command Name	**changeLoopRate**				
Parameters (in)	Field	Description	Type	Units	Required
	rateHo	high-order loop rate	double (0.1 ≤ x ≤ 800)	Hz	yes
	rateLo	low-order loop rate	double (0.1 ≤ x ≤ 800)	Hz	yes
	rateLot	low-order truth loop rate	double (0.1 ≤ x ≤ 400)	Hz	no
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true Execution: state.cmd = BUSY At Completion:				

	state.cmd = READY mode.rateLo = {input rateLo} mode.rateLo = {input rateLo} mode.rateLot = {input rateLot}

Action and Response

This command changes the loop rates, assuming the loop is already closed. It also assumes the System Controller will change the WFS rate at the same time as this is called. It also assumes the System Controller will previously adjust any gains required to make sure the loop does not become unstable. It is expected that the loop(s) will take some time to re-settle. During this time the notable event (discussed in Section 5.3) will be muted to avoid spamming alert messages due to the potential change in flux, which could result in a delay in frames (if this is a reduction in the loop rate).

This will inform the SRT to create a new control matrix, and will tell the optimizations and PSD calculations to restart.

4.3.19 dumpBuffer

Description	Dump circular buffer data to file (for engineering purposes)				
Type	simple				
Command Name	**dumpBuffer**				
Parameters (in)	Field	Description	Type	Units	Required
	name	Circular Buffer name	string		yes
	interval	Internal to dump	int	seconds	no
	file	Filename	string		no
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

Write to file (or use a default filename) the buffer selected. If an interval is not provided then dump all of the buffer. The interval is the amount backwards from the current circular buffer.

4.3.20 setTelemRecording

Description	Enable/disable the writing of stored telemetry				
Type	simple				
Command Name	**setTelemRecording**				
Parameters (in)	Field	Description	Type	Units	Required
	enable	indicates whether to enable or disable the writing of stored telemetry to file	boolean		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

Enable/disable the writing of stored telemetry streams to disk. Reading of circular buffers into stored telemetry files. This may not be required if telemetry is always stored.

4.3.21 calibModePixel

Description	Sets which averaged raw pixel data are sent directly to the SRT-RPG for calibration purposes				
Type	simple				
Command Name	**calibModePixel**				
Parameters (in)	Field	Description	Type	Units	Required
	detector	indicates which detector pixel calibration mode to set.	enum: (LGSWFS , ODGW, OIWFS, NGSWFS, ALL)		yes
	mode	raw pixel calibration mode	enum: (STOP, SINGLE, CONTINUOUS, WC_SYNC)		yes
	avg	the number of pixel frames to average before sending to the SRT-RPG. This argument must be specified only when the mode is set to SINGLE, CONTINUOUS or WC_SYNC.	integer (1 ≤ x)	#Pixel frames	no
	delay	the number of pixel frames to delay (skip) before starting to average pixel frames after a WC override command has been applied. This argument must be specified only when the	integer (0 ≤ x)	#Pixel frames	no

		mode is set to WC_SYNC. If an additional WC override command is received and applied before this delay time has elapsed, then the delay counter is reset. If a WC override command is not received after the mode is set to WC_SYNC or the WC override commands are disabled, then no averaged pixels will be returned.			
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true Execution: state.cmd = BUSY At Completion: state.cmd = READY calibDetector.pixelAvg[{input detector}] = {input avg} calibDetector.pixelMode[{input detector}] = {input mode} calibDetector.pixelDelay[{input detector}] = {input delay}				

Action and Response

This command sets which raw pixel data is sent directly to the SRT-RPG for calibration purposes.

In CONTINUOUS mode the RTC will continually stream (averaged) pixel frames to the SRT-RPG. The STOP mode will stop pixel frames from being sent to the SRT-RPG. The SINGLE mode will only send the next (averaged) pixel frame to the SRT-RPG. At this point, the pixel mode is reset to STOP. In the WC_SYNC mode, the average process will be delayed by the specified number of pixel frames after a WC override command has been applied. If an additional WC override command is received and applied before this delay has elapsed, then the delay is reset.

The raw pixel averages value indicates the number of frames that will be averaged before the frame is sent to the SRT-RPG.

4.3.22 calibModeGrad

Description	sets which averaged gradients are sent directly to the SRT-RPG for calibration purposes				
Type	simple				
Command Name	**calibModeGrad**				
Parameters (in)	Field	Description	Type	Units	Required
	detector	indicates which detector gradient calibration mode to set.	enum: (LGSWFS , ODGW, OIWFS, NGSWFS, ALL)		yes
	mode	gradient calibration mode	enum: (STOP, SINGLE, CONTINUOUS, WC_SYNC)		yes
	avg	the number of gradient to average before sending to the SRT-RPG. This argument must be specified only when the mode is set to SINGLE, CONTINUOUS or WC_SYNC.	integer (1 ≤ x)	#Pixel frames	no
	delay	The number of gradients to delay (skip) before starting to average gradient frames after a WC override command has been applied. This argument must be specified only when the mode is set to WC_SYNC. If an additional WC override command is received and applied before this delay time has elapsed, then the delay counter is reset. If a WC override command is not received after the mode is set to WC_SYNC or the WC override commands are disabled, then no averaged gradients will be returned.	integer (0 ≤ x)	#Pixel frames	no
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true Execution: state.cmd = BUSY At Completion: state.cmd = READY calibDetector.gradAvg[{input detector}] = {input avg} calibDetector.gradMode[{input detector}] = {input mode} calibDetector.gradDelay[{input detector}] = {input delay}				

Action and Response

This command sets which averaged gradients are sent directly to the SRT-RPG for calibration purposes.

This command works similarly to calibModePixel command, except instead of operating on raw pixels, this command operates on gradients. This command will average the gradients from the specified detector and then sending that average to the SRT-RPG.

4.3.23 calibModeCmd

Description	sets what averaged DM and TTS commands are sent directly to the SRT-RPG for calibration purposes				
Type	simple				
Command Name	**calibModeCmd**				
Parameters (in)	Field	Description	Type	Units	Required
	wc	indicates which wavefront corrector command calibration mode to set.	enum: (DM0 DM11 TTS ALL)		yes
	mode	command calibration mode	enum: (STOP, SINGLE, CONTINUOUS, WC_SYNC)		yes
	avg	Number of command frames to average before sending to the SRT-RPG. This argument must be specified only when the mode is set to SINGLE, CONTINUOUS or WC_SYNC.	integer (1 ≤ x)		no
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true calibWc.override = false Execution: state.cmd = BUSY At Completion: state.cmd = READY calibWc.avg[{input wc}] = {input avg} calibWc.mode[{input wc}] = {input mode}				

Action and Response

This command sets what averaged DM and TTS commands are sent directly to the SRT-RPG. This command is intended for system flat calibration purposes.

This command works similarly to calibModePixel command, except instead of operating on raw pixels, this command operates on DM and TTS commands. This command will average the command sent to either the DM and TTS and then send that average to the SRT-RPG.

4.3.24 calibModeWc

Description	puts the RTC wavefront controller into a calibration mode where the DMs and TTS can be adjusted directly by the SRT-RPG				
Type	simple				
Command Name	**calibModeWc**				
Parameters (in)	Field	Description	Type	Units	Required
	enable	enable or disable override commands from the SRT-RPG	boolean		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true loop.ho = IDLE loop.lo = IDLE Execution: state.cmd = BUSY At Completion: state.cmd = READY calibWc.override = {input enable} if {input enable == true}, then calibWc.mode[] = STOP				

Action and Response

This command puts the RTC wavefront controller into a calibration mode where the DMs and TTS can be adjusted directly by the SRT-RPG.

This command is intended for calibration purposes. While performing this calibrate override mode, the DMs and TTS are not updated by their normal path in the RTC pipeline, but instead they listen for update streams directly from the SRT-RPG. While not in calibrate override mode, any WC override command from the SRT-RPG is ignored. This command will be rejected if the high or low-order loops are closed.

4.4 STANDARD COMMANDS

This section details the standard commands that are passed directly from the Command Handler to the pipeline and block.

4.4.1.1 config

Description	Reload configuration files				
Type	simple				
Command Name	**config**				
Parameters (in)	Field	Description	Type	Units	Required
	destin	This can be sent to a process, pipeline or block	string		no
	file	This is an optional file to specify	string		no
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ho = IDLE loop.lgsFocus = IDLE loop.lgsDither = NONE loop.lgsTt[] = IDLE | INACTIVE loop.lo = IDLE loop.oiwfsPoa[] = IDLE Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

On receiving this command the RTC will (re)read the default configuration file or, if a filename is supplied (and it exists), then that file will be used instead. If the destin input indicates a specific destination, then only the keywords in the file applicable to the selected block, pipeline or process will be re-read. Otherwise the command handler will update all blocks, pipelines, or processes.

4.4.1.2 start

Description	Starts all blocks in a pipeline				
Type	simple				
Command Name	**start**				
Parameters (in)	Field	Description	Type	Units	Required
	destin	pipeline	string		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ho = IDLE				

	loop.lgsFocus = IDLE loop.lgsDither = NONE loop.lgsTt[] = IDLE | INACTIVE loop.lo = IDLE loop.oiwfsPoa[] = IDLE Execution: state.cmd = BUSY At Completion: state.cmd = READY

Action and Response

On receiving this command the RTC will (re)start all blocks in that pipeline.

4.4.1.3 stop

Description	This command will open all loops, via the loopOpen command, and abort the processing of an active submitted command.				
Type	simple				
Command Name	**stop**				
Parameters (in)	Field	Description	Type	Units	Required
	detain	pipeline	string		no
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

If nothing is currently active, then nothing will be done. If the destin specifies a pipeline, then the pipeline will stop all blocks it contains.

4.4.1.4 pause

Description	If any loops are closed then they will be opened and retain their current state				
Type	simple				
Command Name	**pause**				
Parameters (in)	Field	Description	Type	Units	Required
	destin	This can be sent to a process, pipeline or block	string		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ho = IDLE loop.lgsFocus = IDLE loop.lgsDither = NONE loop.lgsTt[] = IDLE | INACTIVE loop.lo = IDLE loop.oiwfsPoa[] = IDLE Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

On receiving this command the RTC will, any loops that are closed will be opened and any current state, circulr buffer, etc. will not be cleared.

4.4.1.5 resume

Description	If any loops are opened by the pause command then they will be closed using their current state				
Type	simple				
Command Name	**resume**				
Parameters (in)	Field	Description	Type	Units	Required
	destin	This can be sent to a process, pipeline or block	string		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ho = IDLE loop.lgsFocus = IDLE				

	loop.lgsDither = NONE loop.lgsTt[] = IDLE | INACTIVE loop.lo = IDLE loop.oiwfsPoa[] = IDLE Execution: state.cmd = BUSY At Completion: state.cmd = READY

Action and Response

On receiving this command the RTC will, any loops that were opened by the pause command will be closed using any current state, circular buffers, etc.

4.4.1.6 destroy

Description	All worker threads in the specified Blocks are stopped				
Type	simple				
Command Name	**destroy**				
Parameters (in)	Field	Description	Type	Units	Required
	destin	This can be sent to a pipeline or block	string		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ho = IDLE loop.lgsFocus = IDLE loop.lgsDither = NONE loop.lgsTt[] = IDLE | INACTIVE loop.lo = IDLE loop.oiwfsPoa[] = IDLE Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

On receiving this command, the RTC will stop the worker threads in either all blocks in the specified pipeline or the specified block.

4.4.1.7 init

Description	Re-initialize				
Type	simple				
Command Name	**init**				
Parameters (in)	Field	Description	Type	Units	Required
	destin	This can be sent to a process, pipeline or block	string		
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ho = IDLE loop.lgsFocus = IDLE loop.lgsDither = NONE loop.lgsTt[] = IDLE | INACTIVE loop.lo = IDLE loop.oiwfsPoa[] = IDLE Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

On receiving this command, the RTC will (re)initialized the selected block, pipeline or process. This will also re-read the configuration file.

4.4.1.8 debug

Description	Set the debug level on the block/pipe or process				
Type	simple				
Command Name	**config**				
Parameters (in)	Field	Description	Type	Units	Required
	level	Set debugging to this level	enum {}		yes
	destin	This can be sent to a process, pipeline or block	string		
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ho = IDLE loop.lgsFocus = IDLE				

	loop.lgsDither = NONE loop.lgsTt[] = IDLE | INACTIVE loop.lo = IDLE loop.oiwfsPoa[] = IDLE Execution: state.cmd = BUSY At Completion: state.cmd = READY

Action and Response

On receiving this command the RTC will set the debug level for the selected block, pipeline or process.

4.4.1.9 shutdown

Description	Shutdown software				
Type	simple				
Command Name	**shutdown**				
Parameters (in)	Field	Description	Type	Units	Required
	destin	This can be sent to a process	string		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ho = IDLE loop.lgsFocus = IDLE loop.lgsDither = NONE loop.lgsTt[] = IDLE | INACTIVE loop.lo = IDLE loop.oiwfsPoa[] = IDLE Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

On receiving this command the RTC will send a command to shutdown this specified software process.

5. STATUS, NOTABLE EVENTS AND ALARMS

Status can be subscribed to, or published. The request to subscribe uses TCP sockets described in Section 3.4. In the message header the following information is given below in each of the tables for each of the status variables. Once subscribed the data format will be MSG.

Table 5-1 - Status request structure

Name in RTC Message Header	Values	Description
magicNum	HRT	Magic number used to verify start of message/command.
id	**SUBSCRIBE | UNSUBSCRIBE**	Message identifier. This specifies the message type. The interpretation of the payload data is determined by this identifier.
size	size of the payload	Size of data payload (in bytes).
runId	Incremented number	Identifier for data instance (e.g. sequence number within data stream). This is returned in the reply
time	Time now	Linux time stamp (e.g. arrival time of last pixel in image).
type	**MSG**	Type of message (i.e. command(CMD), message(MSG), ack(ACK), data(DAT)).
devNum	N/A	Device number (if applicable; e.g. LGS WFS A).
footer	No	Does the message include a footer?
check	N/A	Type of checksum used (if footer exists).
Payload:		
Format for payload is: If only the current value is requested then: -current <<<put in status name here, listed in Table 5-4 the “Status Name” column>>> To subscribe to the status and get values back: -subscribe << Put in put in a comma separated list of status name here, listed in Table 5-4 the “Status Name” column >>> -unsubscribe ALL | << Put in put in a comma separated list of status name here, listed in Table 5-4 the “Status Name” column >>>		

The reply is in the format that follows.

Table 5-2 - Status request reply structure

Name in RTC Message Header	Name in the Command Tables below	Description
magicNum	HRT	Magic number used to verify start of message/command.
id	**SUBSCRIBE | UNSUBSCRIBE**	Message identifier. This specifies the message type. The interpretation of the payload data is determined by this identifier.
size	size of the payload	Size of data payload (in bytes).
runId	Number given when the request was sent	Identifier for data instance (e.g. sequence number within data stream). This is returned in the reply
time	Time now	Linux time stamp (e.g. arrival time of last pixel in image).
type	**ACK**	Type of message (i.e. command(CMD), message(MSG), ack(ACK), data(DAT)).
devNum	N/A	Device number (if applicable; e.g. LGS WFS A).
footer	No	Does the message include a footer?
check	N/A	Type of checksum used (if footer exists).
Payload:		
If the request was "–current", then the value is sent back in the form: <<< status name here, listed in Table 5-4 the "Status Name" column = then structure listed in Table 5-5>>>		
Status request reply structure (Table 3-19)		
requestType	CURRENT | SUBSCRIBE | UNSUBSCRIBE	Type of the request
args	As was sent	String representation of the status request (may be a single status name, or multiple)
caller		identifier of the caller (if available)
runId	runId above	A run ID associated with the status request.
comp	Status of the request	status of completion: SUCCESS | FAILED | REJECTED
compMsg	Any message if it failed or was rejected	Completion string explaining why a command is FAILED, or REJECTED. This string will be empty if the command is SUCCESS.

After subscription, the status is then sent as a data format package type (DAT) which is used to transport complex data structures between servers. The identifier (id) specifies the data type of the following data and the data size specifies the data type's size. The identifier and corresponding data type size are fixed and shared between the two servers via a common header file.

Table 5-3 - Status value reported

Name in RTC Message Header	Name in the Command Tables below	Description
magicNum	DAT	Magic number used to verify start of message/command.
id	**Status Name**	Status reporting on
size	size of payload	Size of data payload (in bytes).
runId	Number given when the request was sent	Identifier for data instance (e.g. sequence number within data stream). This is returned in the reply
time	Time now	Linux time stamp (e.g. arrival time of last pixel in image).
type	**DAT**	Type of message (i.e. command(CMD), message(MSG), ack(ACK), data(DAT)).
devNum	N/A	Device number (if applicable; e.g. LGS WFS A).
footer	No	Does the message include a footer?
check	N/A	Type of checksum used (if footer exists).
Payload:		
Value, as listed in the Published Status below		

5.1 PUBLISHED STATUS

5.1.1 Published Status Names and Description

The following table lists the status names, description and where the status is available or updated from. In some cases, if a block is not used in the creation of the RTC then that status will not be available.

Table 5-4 – Published Status names, description

Status name	Description	Block / Process / Pipeline
state	RTC overall state	CommandHandler
SRT-RPGConnection	IP addresses and port numbers for direct connections to the SRT-RPG. The IP/port information is represented as a string in the following format XXX.XXX.XXX.XXX:PORT#. If the connection is unavailable the following string will be published 000.000.000.000:000	CommandHandler
turbLgs	LGS SLODAR seeing and turbulence, in LGS mode only	
turbNgs	NGS seeing and turbulence, in NGS or SL mode only	
calibWc	Wavefront Corrector stream calibration parameters. Arrays are ordered [TTS DM0 DM1 DM2, etc].	
calibDetector	Detector pixel & gradient stream calibration parameters. Arrays are ordered [LGSWFS ODGW OIWFS NGS].	
config	configuration information	
mode	AO mode configuration information	CommandHandler
inst	residual instrument field rotation error Discussion: During LGS observation, the RTC can compute the residual field rotation error.	
lgsFocus	LGS drift focus error for the NCC to adjust the position of the LGS trombone mechanism. Discussion: The parameter lgsFocus will be updated at a specified rate. When the rate parameter changes all subsequent updates of lgsFocus will occur at that new rate. A rate of 0 Hz means the RTC will no longer send subsequent offload parameter updates to the NCC. However at some point in future the rate parameter may change again, to a non-zero value, and the offload parameter updates will resume at the new rate.	
lgsPupilCentError	LGS pupil centering error due to telescope and misalignment as measured by the LGS. Arrays are ordered LGS WFS [A B C D E F].	
lgsState	LGS state. Arrays are ordered LGS WFS [A B C D E F].	
lgsfPointOffset	The mispointing offload value for the LGSF laser launch telescope based on Rayleigh backscattering measurements. Arrays are ordered LGS WFS [A B C D E F].	
loop	Control loop states.	Command Handler
m1UncontrolledModes	M1 uncontrolled modes	

	Discussion: The primary mirror (M1) control system may drift due to temperature or humidity variations during observations. This drift will cause shape changes of M1. These are uncontrolled modes which are reported and then acted upon by M1. These are a vector, dependent in size on configuration	
sim	simulated connections flags.	
ngsPupilCentError	NGS pupil centering error due to telescope and misalignment as measured by the NGS. Arrays are ordered NGS WFS [A B C D E F G].	
ngsTtf	low-pass filtered NGS tip/tilt/focus error. . Arrays are ordered NGS WFS [A B C D E F G].	
ngsState	NGS state. Arrays are ordered NGS WFS [A B C D E F G].	
telOffloadTt	tip/tilt telescope offloading Discussion: The RTC computes the tilt modes (defined in the Noll Zernike basis) based on the TTS. The tilt modes are offloaded and then rotated to the telescope mount control system by the TCS. Discussion: The parameter telOffloadTt will be updated at a specified rate. When the rate changes all subsequent updates of telOffloadTt will occur at that new rate. A rate of 0 Hz means the RTC will no longer send subsequent offload parameter updates to the TCS. However at some point in future the rate parameter may change again, to a non-zero value, and the offload parameter updates will resume at the new rate. Different modes (LGS & NGS AO and seeing limited) will have different rates	
telOffloadFocus	focus telescope offloading Discussion: The RTC computes the focus mode (defined in the Noll Zernike basis) based on the DM commands. The focus mode is offloaded and then rotated to the telescope M2 control system by the TCS. Discussion: The parameter telOffloadFocus will be updated at a specified rate. When the rate parameter changes all subsequent updates of telOffloadFocus will occur at that new rate. A rate of 0 Hz means the RTC will no longer send subsequent offload parameter updates to the TCS. However at some point in future the rate parameter may change again, to a non-zero value, and the offload parameter updates will resume at the new rate. Different modes (LGS & NGS AO and seeing limited) will have different rates.	
telOffloadMag	magnification telescope offloading (TBC) Discussion: In LGS WFS mode in which 3 OIWFSs or ODGWs are used, the RTC computes the magnification plate scale mode (created by a combination of telescope M1 curvature and telescope M2 focus) based on the DM0 and DM11 commands. The magnification error is then offloaded to the telescope M1 and M2	

	control system by the TCS. Positive values mean that the image is too large. In NGS of Seeing Limited mode the offload magnification error will nominally be disabled. Discussion: The parameter telOffloadMag will be updated at a specified rate. When the rate parameter changes all subsequent updates of telOffloadMag will occur at that new rate. A rate of 0 Hz means the RTC will no longer send subsequent offload parameter updates to the TCS. However at some point in future the rate parameter may change again, to a non-zero value, and the offload parameter updates will resume at the new rate. Different modes (LGS & NGS AO and seeing limited) will have different rates.	
telOffloadMode	mode telescope offloading The RTC computes the coma modes and additional M1 telescope modes (defined in the Noll Zernike basis) based on the DM commands. Discussion: The coma modes are offloaded and then rotated to the telescope M2 control system by the TCS. Discussion: The parameter telOffloadMode will be updated at a specified rate. When the rate parameter changes all subsequent updates of telOffloadMode will occur at that new rate. A rate of 0 Hz means the RTC will no longer send subsequent offload parameter updates to the TCS. However at some point in future the rate parameter may change again, to a non-zero value, and the offload parameter updates will resume at the new rate. Different modes (LGS & NGS AO and seeing limited) will have different rates.	
pixelCrc	A bad CRC was detected in at least one pixel UDP datagram.	
ngsWfsState.enable[]	Enabled/disabled the reading of each NGS WFS	
ngsWfsCentroid.enable[]	Enabled/disabled inputs for the low order calculations	
ngsCenteringErrState.enable[]	Enabled/disabled the sending of individual NGS centering errors.	
ngsFieldRotOffsetState.enable	Enabled/disabled the sending of NGS field rotation offsets to the System Controller	
ngsTipTiltState.enable	Enabled/disabled the sending of NGS wavefront tip tilt corrections	
ngsAvgTTtoSCSState.enable[]	Enabled/disabled the offload of average tip-tilt corrections to the Secondary Mirror Controller	
ngsFieldDistState.enable[]	Enabled/disabled the sending of field distortion modes to the high order loop	
lgsFocusErrorsState.enable[]	Enabled/disabled the sending of LGS Focus errors to the System Controller from the SFS WFS	

lgsWfsState.enable[]	Enabled/disabled the reading for each LGS WFS	
lgsHoWfsState.enable[]	Enabled/disabled using LGS WFS for HO calculations	
lgsCenteringErrState.enable[]	Enabled/disabled the sending of LGS Centering errors to the Laser Fast Steering Mirrors	
dmShapesState.enable[]	Enabled/disabled the sending of Deformable Mirror shapes to Deformable Mirrors	
comaFocusState.enable[]	Enabled/disabled the sending of coma and focus corrections to the System Controller for the Secondary Control System.	
primaryFigureState.enable	Enabled/disabled the sending of primary mirror figure corrections to the RTC System Controller	
loIntegratorsState[]	LO control state	
hoIntegratorState[]	HO control state	

5.1.2 Published Status Attributes

The following table details the attributes associated with each status item.

Table 5-5 – Published Status Attributes

Status name	Attributes			
	Name	Description	Type	Units
state				
	mode	flag indicating if the RTC mode has been configured	enum: (UNINITIALIZED, READY, BUSY)	
	shutdown	flag indicating if the RTC Assembly has be properly shutdown	boolean	
	pol	indicates whether pseudo open-loop (POL) feedback is enabled. When POL is disabled then the uncontrolled mode cleanup path is enabled. Nominally POL will be used in LGS mode and disabled for NGS mode.	boolean	
	SRT-RPGDate	flag indicating if the RTC has all the required data from the SRT-RPG to start the pipeline	boolean	
	lgsBackground	flag indicating if the LGS background task is currently active	boolean	

	unsavedConfig	flag indicating if any configuration parameters have been changed, via the configParamSet or calibLgsBackground, but has not yet been saved	boolean	
	unsavedLgsSky	flag indicating if the current LGS sky & instrument backgrounds have been changed, via the calibLgsBackground command, but has not yet been saved	boolean	
	errMsg	RTC error description, empty string if there are no errors.	string	
SRT-RPGConnection				
	wcc	IP/port for WCC server endpoint.	string	
	lgsHopA	IP/port for LGS HOP A server endpoint.	string	
	lgsHopB	IP/port for LGS HOP B server endpoint.	string	
	lgsHopC	IP/port for LGS HOP C server endpoint.	string	
	lgsHopD	IP/port for LGS HOP D server endpoint.	string	
	lgsHopE	IP/port for LGS HOP E server endpoint.	string	
	lgsHopF	IP/port for LGS HOP F server endpoint.	string	
	ngsHopA	IP/port for NGS HOP A server endpoint.	string	
	ngsHopB	IP/port for NGS HOP B server endpoint.	string	
	ngsHopC	IP/port for NGS HOP C server endpoint.	string	
	ngsHopD	IP/port for NGS HOP D server endpoint.	string	
turbLgs				
	numLayer	number of layers	Integer (1 ≤ x ≤ 12)	
	numLayerBin	number of binned layers	Integer (1 ≤ x ≤ 12)	
	numWfsPair	number of LGS WFS pairs	Integer (1 ≤ x ≤ 15)	
	wfsPairs	LGS WFS pairs Only the first 'numWfsPair' columns in the matrix will be valid, the remaining columns will be set to zero.	array[2,15] of enum: (LGSWFSA LGSWFSB LGSWFSC LGSWFSD LGSWFSE LGSWFSF)	

	profile	C_n^2 turbulence profile (p) Only the first 'numLayer' row in the matrix will be valid, all remaining rows will be set to zero. Rows are ordered from the ground layer up.	array[12] of float	$m^{-5/3}$
	profileBin	Binned C_n^2 turbulence profile (p') Only the first 'numLayer' row in the matrix will be valid, all remaining rows will be set to zero. Rows are ordered from the ground layer up.	array[12] of float	$m^{-5/3}$
	r0	fried parameter (r_0)	float	m
	theta0	isoplanatic angle (θ_0)	float	mas
	theta2	generalized isoplanatic angle (θ_2)	float	mas
	L0	turbulence outer scale (L_0)	float	m
	tau0	coherence time (τ_0)	float	seconds
turbNgs				
	groundLayer	ground layer turbulence (p)	float	$m^{-5/3}$
	r0	fried parameter (r_0)	float	m
	l0	turbulence outer scale (L_0)	float	m
	tau0	coherence time (τ_0)	float	seconds
calibWc				
	avg	number of commands to average before sending the average to the SRT-RPG.	array[3] of integer $(1 \leq x)$	
	mode	flag indicating if RTC is sending averages commands to the SRT-RPG.	array[3] of enum: (STOP, SINGLE, CONTINUOUS)	
	override	flag indicating if RTC is accepting WC override commands from the SRT-RPG, for calibration	boolean	
calibDetector				
	gradAvg	Number of gradient frames to average before sending the average to the SRT-RPG	array[4] of integer $(1 \leq x)$	
	gradMode	Flag indicating if the RTC is sending averages gradients to the SRT-RPG	array[4] of enum: (STOP, SINGLE, CONTINUOUS, WC_SYNC)	

	gradDelay	The number of gradient frames that is delayed (skipped) while in WC_SYNC mode, see pixelDelay for more details.	array[4] of integer (0 ≤ x)	
	pixelAvg	number of pixel frames to average before sending the average to the SRT-RPG.	array[4] of integer (1 ≤ x)	
	pixelMode	flag indicating if RTC is sending averages pixels to the SRT-RPG.	array[4] of enum: (STOP, SINGLE, CONTINUOUS, WC_SYNC)	
	pixelDelay	The number of pixel frames that is delayed (skipped) before the starting to average pixel frames, after a WC override command has been applied, while in WC_SYNC mode	array[4] of integer (0 ≤ x)	
config				
	name	current configuration file name	string	
	version	current configuration file version	string	
mode				
	rateHo	high-order loop rate. In LGS mode this is equal to the LGS WFS frame rate. In NGS or SL mode this is equal to the NGS frame rate	float	Hz
	rateLo	low-order loop rate. This is equal to the low order (Tier 1 and 2) steady-state frame rate.	float	Hz
	rateLot	low-order truth loop rate. This is equal to the Tier 3 and 3F steady-state frame rate.	float	Hz
	rateWc	wavefront correction. In LGS mode this should be at least equal to the LGS WFS frame rate (i.e. the high-order rate). In NGS mode this should be at least equal to the NGS frame rate (i.e. the high-order rate). In SL mode his should be at least to the Tier 1 and 2 steady-state frame rate (i.e. the low-order rate).	float	Hz
inst				
	rot	residual error field rotation [δθ] in the XY plane. A positive rotation is defined as counterclockwise from X axis to Y axis.	double (-360 ≤ x ≤ 360)	degree
	rate	The rate at which the instrument rotation error is updated.	float (0 ≤ x ≤ 20)	Hz
lgsPupilCentError				

	x	shift of the LGS pupil in the x-direction due to misalignments	array[6] of double (0 ≤ x ≤ 0.5)	fraction of pupil diameter
	y	shift of the LGS pupil in the y-direction due to misalignments	array[6] of double (0 ≤ x ≤ 0.5)	fraction of pupil diameter
	rate	The rate at which this telescope pupil shift misalignment parameter is updated. This rate will be equal to minimum of the LGS frame rate and 20 Hz.	array[6] of float (0 ≤ x ≤ 20)	Hz
	time	timestamp	array[6] of double	TAI / PTP
lgsFocus				
	error	net LGS drift focus error (z_b)	double	µm of RMS wavefront error
	wfs	LGS drift focus error per WFS. The arrays is ordered LGS WFS [A B C D E F]. If the LGS WFS is disabled (lgsState.enable = false) then the corresponding focus error term will be set to zero and ignored while producing the net focus error.	array[6] of double	µm of RMS wavefront error
	rate	the rate at which this drift focus error is updated	float (0 ≤ x ≤ 20)	Hz
lgsState				
	enable	flag indicating is LGS WFS has been configured as enabled	array[6] of boolean	
	fluxHigh	number of high-flux subaps	array[6] of integer (0 ≤ x)	
	fluxLow	number of low-flux subaps	array[6] of integer (0 ≤ x)	
	algo	current LGS gradient algorithm	enum: (COG_STATIC, MF_COG, MF_UPDATE, MF_STATIC, CUSTOM, NONE)	

	ttFilter	current tip/tilt control filter for the LGS TT loop.	enum: (INTEGRATOR, KALMAN)	
	focusFilter	state of the LGS gradient focus mode filter	enum: (ALLPASS, NOPASS, LPF)	
lgsfPointOffset				
	x	x mispointing error	array[6] of float	arcsec on the sky of x mispointing
	y	y mispointing error	array[6] of float	arcsec on the sky of y mispointing
loop				
	ho	State of the high-order (HO) wavefront correction loop. • IDLE - HO loop is not active • LOST - at least one active LGS WFS frame is lost, therefore no high-order correction • ACQUIRE - HO loop is active but has not settled (TBC) • LOCK - HO loop has settled	enum: (IDLE, LOST, ACQUIRE, LOCK)	
	lgsFocus	State of the LGS Trombone Focus offloading loop. • IDLE - RTC is not offloading focus • LOST - a LGS spot can not be detect on at least one active LGS WFS, therefore no offloading • ACQUIRE - RTC is offloading focus, but focus error is above the configured focus threshold (from config file) • LOCK - RTC is offloading focus and focus error is below the configured focus threshold (from config file)	enum: (IDLE, LOST, ACQUIRE, LOCK)	
	lgsFocusFilter	State of the LGS Focus filter applied to the high-order gradients.	enum: (ALL, NO, LOW)	

		• ALL • NO • LOW		
	lgsDither	flag indicating the state of LGS WFS dithering in the RTC. • NONE - the RTC will not dither the DM (LGS purposes) nor the LGSF FSMs. • CP - the RTC is currently applying the CP DM dither signal. This dither signal can be enabled independently of the high and low-order loops (see loop.ho & loop.lo). • NCP - the RTC is currently applying a LGSF FSM dither signal, which can be enabled independently of the LGS TT control loop (see loop.lgsTt)	enum: (CP, NCP, NONE)	
	lgsTt	State of the LGS TT control loop that drive the LGSF FSM, this does not include the application of dither signal which can be enabled independently (see published item lgsDither). The array is ordered LGSF FSM [A B C D E F]. • IDLE - RTC is not sending FSM commands • LOST - FSM command is lost, previous FSM command will be use • ACQUIRE - RTC is sending FSM commands, but TT error is above the configured TT threshold (from config file) • LOCK - RTC is sending FSM commands and TT error is below the configured TT threshold (from config file) • INACTIVE - the LGS WFS is disabled, therefore a zero command is sent for this LGSF FSM.	array[6] of enum: (IDLE, LOST, ACQUIRE, LOCK, INACTIVE)	
	lgsDriftTt	Flag indicates whether to TT drift terms are added to the LGS TT control loop	boolean	
	lo	Combined state of all low-order tiers, i.e. all active low-order and low-order truth detectors, (see loop.tier*). The possible overall states are: • IDLE - all LO tiers are IDLE or INACTIVE • LOST - at least one LO tier is LOST	enum: (IDLE, LOST, ACQUIRE, DITHER, LOCK, LOCK_PARTIAL)	

		• ACQUIRE - at least one LO tier is ACQUIRE, but no LO tier is LOST • DITHER - at least one LO tier is DITHER, but no LO tier is LOST or ACQUIRE • LOCK_PARTIAL - at least one LO tier is LOCK, all other tiers states are LOCK, IDLE or INACTIVE • LOCK - all tier states are LOCK or INACTIVE. The overall state is set following these rules of president (in order): • LO state is LOST if any tier state is LOST • LO state is ACQUIRE if any tier state is ACQUIRE • LO state is DITHER if any tier state is DITHER • LO state is LOCK if all tier states are LOCK or INACTIVE • LO state is LOCK_PARTIAL if any tier state is LOCK • else LO state is IDLE		
	loFilter	current temporal filter type used for each LO mode. The array is ordered LO mode [Tip, Tilt, Focus, Plate Scale X, Plate Scale Y, Plate Scale 45]	array[6] of enum: (NONE, TYPE1_ACQ, TYPE1, KALMAN)	
	oiwfsPoa	state of the offload to the OIWFS POAs. The array is ordered OIWFS [1 2 3]. • IDLE - RTC not offloading to OIWFS POA • LOST - the OIWFS frame is lost, therefore offloading value is set to zero. • ACTIVE - offloading to OIWFS POA	array[3] of enum: (IDLE, LOST, ACTIVE)	arcsec
	ngsDither	flag indicating the state of NGS dithering in the RTC. • NONE - then the RTC will not dither the DM (for NGS purposes) and will ignore the NGS FSM position stream. • CP - the RTC assumes the NGS FSM is being dither and will listen to the NGS FSM position stream from the NCC for dither signal estimation. • NCP - the RTC is currently applying the CP DM dither signal. This dither signal can be	enum: (CP, NCP, NONE)	

		enabled independently of the high and low-order loops.		
	ngsSsm	state of the offload to the visible NGS (i.e. NGS) SSM. • IDLE - RTC not offloading to SSM • LOST - NGS frame is lost, therefore no offloading value is set to zero. • ACTIVE- offloading to SSM	enum: (IDLE, LOST, ACTIVE)	
	twfs	State of the LGS TWFS reference vector update process. • IDLE - RTC not update TWFS reference vector • LOST - NGS frame is lost, therefore no TWFS reference vector updates • DITHER - telescope is dithering, therefore no TWFS reference vector updates • LOCK - guide star lock and TWFS reference vector is being updated	enum: (IDLE, LOST, DITHER, LOCK)	
	ready	A flag indicating if the RTC pipeline is ready to close loops. If TRUE then pixel processing has started and the DMs and TTS are being driven to current shape in the main integrator	boolean	
	tier0	State of a NGS Tier 0 wavefront correction loop • IDLE - tier is inactive • LOST - tier frame is lost and not contributing to low-order correction • DITHER - telescope is dithering, tier may be either (re)acquiring or locked on guide star • LOCK - tier is locked on guide star	enum: (IDLE, LOST, DITHER, LOCK)	
	tier1	State of a LO Tier 1 wavefront correction loop • IDLE - tier is inactive • LOST - tier frame is lost and not contributing to low-order correction • ACQUIRE - tier is (re)acquiring • DITHER - telescope is dithering, tier may be either (re)acquiring or locked on guide star • LOCK - tier is locked on guide star	enum: (IDLE, LOST, ACQUIRE, DITHER, LOCK)	

	tier2	State of a LO Tier 2 wavefront correction loop. The array is ordered Tier 2 [A B]. • IDLE - tier is not active • LOST - tier frame is lost and not contributing to low-order correction • ACQUIRE - tier is (re)acquiring • DITHER - telescope is dithering, tier may be either (re)acquiring or locked on guide star • LOCK - tier is locked on guide star • INACTIVE - no detector was specified for this tier, therefore it does no contribute to the low-order loop	array[2] of enum: (IDLE, LOST, ACQUIRE, DITHER, LOCK, INACTIVE)	
	tier3	State of a LOT Tier 3 wavefront correction loop. The array is ordered Tier 3 [A B C D]. • IDLE - tier is inactive • LOST - tier frame is lost and not contributing to low-order correction • ACQUIRE - tier is (re)acquiring • DITHER - telescope is dithering, tier may be either (re)acquiring or locked on guide star • LOCK - tier is locked on guide star • INACTIVE - no detector was specified for this tier, therefore it does no contribute to the low-order loop	array[4] of enum: (IDLE, LOST, ACQUIRE, DITHER, LOCK, INACTIVE)	
	tier3f	State of a LOT Tier 3 wavefront correction loop • IDLE - tier is inactive • LOST - tier frame is lost and not contributing to low-order correction • ACQUIRE - tier is (re)acquiring • DITHER - telescope is dithering, tier may be either (re)acquiring or locked on guide star • LOCK - tier is locked on guide star • INACTIVE - no detector was specified for this tier, therefore it does no contribute to the low-order loop	enum: (IDLE, LOST, ACQUIRE, DITHER, LOCK, INACTIVE)	
	telOffloadTt	state of the TT telescope offload publishing for the TCS • IDLE - The RTC is not publishing telOffloadTt	enum: (IDLE, LOST, LOCK)	

		• LOST - The RTC is publishing telOffloadTt, but due to lost data/frame the values are set to zero. • LOCK -The RTC is publishing telOffloadTt for offloading to the TCS		
	telOffloadFocus	state of the focus telescope offload publishing for the TCS • IDLE - The RTC is not publishing telOffloadFocus • LOST - The RTC is publishing telOffloadFocus, but due to lost data/frame the values are set to zero. • LOCK -The RTC is publishing telOffloadFocus for offloading to the TCS	enum: (IDLE, LOST, LOCK)	
	telOffloadMag	state of the magnification telescope offload publishing for the TCS • IDLE - The RTC is not publishing telOffloadMag • LOST - The RTC is publishing telOffloadMag, but due to lost data/frame the values are set to zero. • LOCK - The RTC is publishing telOffloadMag for offloading to the TCS	enum: (IDLE, LOST, LOCK)	
	telOffloadMode	state of the telescope mode offload publishing for the TCS • IDLE - The RTC is not publishing telOffloadMode • LOST - The RTC is publishing telOffloadMode, but due to lost data/frame the values are set to zero. • LOCK -The RTC is publishing telOffloadMode for offloading to the TCS	enum: (IDLE, LOST, LOCK)	
oiwfs1Poa				
	tip	OIWFS 1 tip error	double	arcsec
	tilt	OIWFS 1 tilt error	double	arcsec
	rate	The rate at which the OIWFS 1 tip/tilt errors offloading parameter is updated.	float ($0 \leq x \leq 20$)	Hz

oiwfs2Poa				
	tip	OIWFS 2 tip error	double	arcsec
	tilt	OIWFS 2 tilt error	double	arcsec
	rate	The rate at which the OIWFS 2 tip/tilt errors offloading parameter is updated.	float ($0 \leq x \leq 20$)	Hz
oiwfs3Poa				
	tip	OIWFS 3 tip error	double	arcsec
	tilt	OIWFS 3 tilt error	double	arcsec
	rate	The rate at which the OIWFS 3 tip/tilt errors offloading parameter is updated.	float ($0 \leq x \leq 20$)	Hz
ngsState				
	enable	flag indicating is OIWFS has been configured as enabled	array[7] of enum: (NONE TT TTF)	
	fluxHigh	OIWFS high-flux flag	array[7] of boolean	
	fluxLow	OIWFS low-flux flag	array[7] of boolean	
	algo	current OIWFS gradient algorithm	array[7] of enum: (COG_STATIC, MF_COG, MF_UPDATE, MF_STATIC, BP, CUSTOM, NONE)	
	handOff	flag indicating if the detector is currently disabled for guide star hand-off	array[7] of boolean	
	acqTable	flag indicating if the acquisition and dither tables have been set in the RTC for the OIWFSs	array[7] of boolean	
sim				
	dm0	DM0 simulated connection flag. If set then commands are sent to the simulator IP/port instead of the DM0 IP/port.	boolean	
	dm11	DM11 simulated connection flag. If set then commands are sent to the simulator IP/port instead of the DM11 IP/port.	boolean	

	lgs	LGS WFS pixel simulated connection flag. If set then pixels are processed from to the simulator port instead of the real LGS WFS	boolean	
	Lgsf	LGSF simulated connection flag. If set then commands are sent to the simulator IP/port instead of the LGSF IP/port.	boolean	
	odgw	ODGW pixel simulated connection flag. If set then pixels are processed from to the simulator port instead of the real ODGW port	array[4] of boolean	
	oiwfs	The array is ordered ODGW [1 2 3 4]. OIWFS pixel simulated connection flag. If set then pixels are processed from to the simulator port instead of the real OIWFS port. The array is ordered OIWFS [1 2 3].	array[3] of boolean	
	ngs	NGS pixel simulated connection flag. If set then pixels are processed from to the simulator port instead of the real NGS port.	boolean	
	SRT-RPG	SRT-RPG simulated connection flag. If set then RTC will communicate with the simulated SRT-RPG IP/port specified in the configuration file		
	tts	TTS simulated connection flag. If set then commands are sent to the simulator IP/port instead of the TTS IP/port.	boolean	
ngsPupilCentError				
	x	shift of the NGS pupil in the x-direction due to misalignments	array[7] of double (0 ≤ x ≤ 0.5)	fraction of pupil diameter
	y	shift of the NGS pupil in the y-direction due to misalignments	array[7] of double (0 ≤ x ≤ 0.5)	fraction of pupil diameter
	rate	The rate at which this telescope pupil shift misalignment parameter is updated. This rate will be equal to minimum of the NGS frame rate and 20 Hz.	array[7] of float (0 ≤ x ≤ 20)	Hz
	time	timestamp	array[7] of double	TAI / PTP

ngsTtf				
	tip	NGS tip error	array[7] of double	Arcsec on the sky
	tilt	NGS tilt error	array[7] of double	Arcsec on the sky
	focus	NGS focus error	array[7] of double	µm of RMS wavefront error
	rate	The rate at which the NGS tip/tilt/focus errors offloading parameter is updated.	array[7] of float (0 ≤ x ≤ 20)	Hz
ngsState				
	ngsBin	width of NGS bin in pixels	enum: (1, 2, 4, 8, 16, 32)	
	fluxHigh	number of high-flux subaps (where a subap is a set of four quadrant pixels)	integer (0 ≤ x)	
	fluxLow	number of low-flux subaps (where a subap is a set of four quadrant pixels)	integer (0 ≤ x)	
	algo	current NGS gradient algorithm	enum: (COG_STATIC, GOG_UPDATE, CUSTOM, NONE)	
	opticalGain	current NGS optical gain, ordered [x, y]. If the scalar optical gain optimization algorithm is specified in the RTC Assembly configuration file, then both values in the array will be equal.	array[2] of double	
	handOff	flag indicating if the detector is currently disabled for guide star hand-off	boolean	
	telDitherFlag	flag indicating if NGS thinks the telescope is dithering or not	boolean	
	telDitherGain	gain applied to the NGS gradients while the telescope is performing a dither	float (0 ≤ x ≤ 1)	
telOffloadTt				
	x	tip mode	float	Arcsec on the sky

	y	tilt mode	float	Arcsec on the sky
	rate	the rate at which this telescope TT offloading parameter is updated	float (0 ≤ x ≤ 20)	Hz
	time	timestamp	double	TAI / PTP
telOffloadFocus				
	focus	focus mode	float	µm of WFE RMS
	rate	the rate at which this telescope focus offloading parameter is updated	float (0 ≤ x ≤ 20)	Hz
	time	timestamp	double	TAI / PTP
telOffloadMag				
	magnification	The magnification plate scale error is relative to the telescope prescription.	float (-0.01≤ x ≤0.01)	fractional error of image scale relative to the designed telescope prescription
	rate	the rate at which this telescope magnification offloading parameter is updated	float (0 ≤ x ≤ 20)	Hz
	time	timestamp	double	TAI / PTP
telOffloadMode				
	comaX	x coma mode	float	µm of WFE RMS
	commaY	y coma mode	float	µm of WFE RMS
	m1	M1 modes Discussion: The RTC computes the M1 modes (defined in the Noll Zernike basis, but	array[100] of float	nm of RMS wavefr

		excluding piston, tilt, focus and coma modes) based on the DM0 commands. The M1 modes are offloaded and then rotated to the telescope M1 control system by the TCS. In LGS and NGS modes up to approximately 100 M1 modes are offloaded. In seeing limited mode, only up to the 5th order radial modes are offloaded, zero-padded up to the full vector length.		o-nt error
	rate	The rate at which this telescope TT offloading parameter is updated	float ($0 \leq x \leq 20$)	Hz
	time	timestamp	double	TAI / PTP
pipelineDelay		(Notable events are defined in Section 5.3)		
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
ngsDitherTimeout				
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
lotLost				
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
lotfLost				
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
lotDegrade				
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
loLost				
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
loDegrade				
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
loNgsLost				
	state	State of the Notable Event	boolean	

	msg	Error message describing the Notable Event	string	
loCorrectionLost				
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
hoCorrectionLost				
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
loErrVectLost				
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
hoErrVectLost				
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
ttsSenseLost				
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
dmClip				
	state	State of the Notable Event	array[3] of boolean	
	msg	Error message describing the Notable Event	array[3] of string	
lgsTtConvergeError				
	hop	Block num	int	
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
lgsFocusConvergeError				
	hop	Block num	int	
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
lgsTtCorrectionLost				
	hop	Block num	int	

	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
lgsTtSenseLost				
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
lgsFocusCorrectionLost				
	hop	Block num	int	
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
lgsPolGradLost				
	hop	Block num	int	
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
lgsDmShapeLost				
	hop	Block num	int	
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
ngsGradMissing				
	hop	Block num	int	
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
ngsGradOrder				
	hop	Block num	int	
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
ngsGradCrc				
	hop	Block num	int	
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
dmVectMissing				
	hop	Block num	int	

	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
dmVectOrder				
	hop	Block num	int	
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
dmVectCrc				
	hop	Block num	int	
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
guardRow				
	detector	Detector that triggered the Notable Event	enum: (OIWFSA, OIWFSB, OIWFSC, ODGW1, ODGW2, ODGW3, ODGW4)	
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
lowFlux				
	detector	Detector that triggered the Notable Event	enum: (LGSWFSA, LGSWFSB, LGSWFSC, LGSWFSD, LGSWFSE, LGSWFSF, NGS, OIWFSA, OIWFSB, OIWFSC, ODGW1, ODGW2, ODGW3, ODGW4)	
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
pixelMissing				

	detector	Detector that triggered the Notable Event	enum: (LGSWFSA, LGSWFSB, LGSWFSC, LGSWFSD, LGSWFSE, LGSWFSF, NGS, OIWFSA, OIWFSB, OIWFSC, ODGW1, ODGW2, ODGW3, ODGW4)	
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
pixelOrder				
	detector	Detector that triggered the Notable Event	enum: (LGSWFSA, LGSWFSB, LGSWFSC, LGSWFSD, LGSWFSE, LGSWFSF, NGS, OIWFSA, OIWFSB, OIWFSC, ODGW1, ODGW2, ODGW3, ODGW4)	
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	
pixelCrc				
	detector	Detector that triggered the Notable Event	enum: (LGSWFSA, LGSWFSB, LGSWFSC, LGSWFSD, LGSWFSE, LGSWFSF, NGS, OIWFSA, OIWFSB, OIWFSC, ODGW1,	

			ODGW2, ODGW3, ODGW4)	
	state	State of the Notable Event	boolean	
	msg	Error message describing the Notable Event	string	

5.2 SUBSCRIBED STATUS

The following table lists the status that is required from outside the RTC, it also indicates where that data is directed. It is expected that this table will be updated over time.

Table 5-6 – Subscribed Status names, description and where it is used

Status name	Desc	Block/Process/Pipeline
Zenith Angle	Telescope zenith angle	SRT
R0	Initial R0 estimate	SRT

5.3 NOTABLE EVENTS

A common occurrence throughout control systems is the generation of flags or warnings by components to indicate that some sort of exceptional, but automatic, control action must be taken. These are called “notable events”. They are not to be confused with “alarms” that have a strict interpretation, being reserved for conditions that require human operator intervention (i.e., corrective action cannot be taken automatically for an alarm). The notable event itself may simply trigger other checks to determine what precisely caused the event (e.g., by checking other published state variables), before deciding on the appropriate corrective action.

All notable events are assumed to have internal state (usually Boolean), unless specifically stated otherwise. This implies that when a condition arises that warrants a notable event, the internal notable event state will be set to true and are published. However, if on consecutive subsequent frames/iterations the condition persists, the notable event will not continue to be published repeatedly (to reduce unnecessary bandwidth usage). Conversely, if the condition that raised the notable event is no longer true, the notable event state is cleared and published again. A timestamp will be included that indicates when the notable event state transition occurred internally (i.e., it will generally have occurred before the event is actually published).

A major use of notable events, particularly in the RTC, is to inform the higher-level software that there is low flux on a detector, this is likely due to the loss of an individual guide star. In this case, the higher-level software can pause an observation and autonomously trigger a re-acquisition process for that star, which has the potential for recovery much faster than re-applying the full AO acquisition sequence.

A component may also publish notable events for a variety of other component/instrument specific reasons or for other miscellaneous unexpected faults, such as hardware communication issues, missing data packages, slow or late data transmissions, or garbled/invalid data.

A list of notable events is given in Section 5.3

5.3.1 Notable Events Publishing Rate

One issue with internal states is they are often based on a **noisy measurement;** if subsequent measurements lie close to a state transition threshold, there may be high-frequency oscillations between the true/false states. To avoid high-frequency oscillations in a notable event stream, the responsible state object will only publish a notable event at a configurable maximum frequency (e.g., 100 Hz, or every 10 ms). If the internal state of the notable event is true over the entire 10 ms period, the event will only be published if the previous published state was false. However, if over that period the state switches from false to true, then the notable event will be published as true, even if it was previously true. Finally, if the previously published state is true, and over the entire period the state is false or the state only switches from true to false, then the notable event will be published as false.

An example of a changing internal state with notable events published every 10 ms is shown in Figure 7.

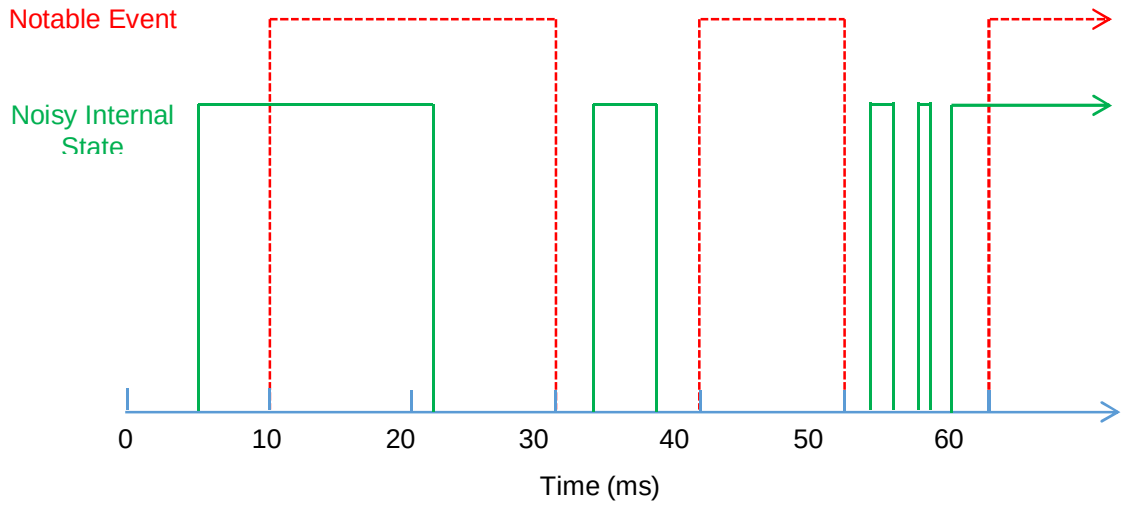

Figure 7 - Illustration of published notable events (red) with a 10 ms period, corresponding to a changing noisy internal state (green).

In this figure, it is assumed that the internal state is initially false. The following bullets summarize the notable events that are published at the indicated times:

- 10 ms -> true: reports that the internal state went true at 5 ms
- 30 ms -> false: reports that the internal state went false at 22 ms
- 40 ms -> true: reports that the internal state went true at 33 ms
- 50 ms -> false: reports that the internal state went false at at 37 ms
- 60 ms -> true: reports that the internal state went true at 52 ms

Despite the oscillations in the internal state between 50 and 60 ms, it finished true, so no further notable event indicating that it temporarily went false would be published.

5.3.2 Notable Event Fields

These are events that are reported in the Global Memory by a block for the Command Handler. They are then available for the System Controller to read. These are special status called notable events.

The format for the request for notable events is the same as described in Section 5.1.

Table 5-7 – Notable Events

Event	Block	Description
ngsPixelMissing	Low order NGS processing	At least one NGS pixel UDP datagram was missing from the frame.
ngsOutOfOrder	Low order NGS processing	At least one NGS pixel UDP datagram was received out of order.
ngsBadCrc	Low order NGS processing	A bad CRC was detected in at least one NGS pixel UDP datagram.
ngsLowFlux	Low order NGS processing	Insufficient flux detected on NGS.
ngsOiwfsPixelMissing	Low order NGS processing	At least one OIWFS pixel UDP datagram was missing from the frame.
hoDMVectOutOfOrder	Temporal Filtering & Combination	At least one HO DM vector UDP datagram was received out of order.
hoDMVectBadCrc	Temporal Filtering & Combination	A bad CRC was detected in at least one HO DM vector UDP datagram.
lotLost	LO Reconstructor	LOT modes cannot be computed.
lotDegrade	LO Reconstructor	Only some of the LOT modes can be computed.
loLost	LO Reconstructor	LO modes cannot be computed.
loDegrade	LO Reconstructor	Only some of the LO modes can be computed.
loNgsLost	LO Reconstructor	LO modes, based on the Tier 0 NGS modes, cannot be computed.
loCorrectionLost	Temporal Filtering & Combination	LO error vector cannot be computed.
hoCorrectionLost	HO Reconstructor	HO error vector cannot be computed.
hoErrVectLost	HO Reconstructor	LO error vector is late
loErrVectLost	HO Reconstructor	HO error vector is late
ttsSenseLost	Closed Loop WFC	Failed to get TTS postion.
dmClip	Closed Loop WFC	DMs were excessively clipped.

lgsTtConvergeError	Closed Loop WFC	LGS TT error has not converged in the expected amount of time.
lgsTtCorrectionLost	LGS processing	Lost LGS TT correction.
lgsFocusConvergeError	LGS processing	Missing some or all LGS focus errors
lgsPolGradLost	Temporal Filtering & Combination	Missing some or all LGS POL gradients
lgsDmShapeLost	Temporal Filtering & Combination	Lost LGS DM shape feedback
lowFlux	Command Handler	Insufficient flux detected on a detecto
log	Command Handler	Includes the output of the command handler, and it based on the current debut level
overallPipelineStatus	Command Handler	A combination of all of the block/pipeline status
pipelineDelay	Command Handler	Part or all of the RTC pipeline is running slow Notable Event.

5.4 ALARMS

Alarms indicate a condition that require a human operator's intervention, or at least acknowledgement. It is assumed that alarms cannot be monitored by another component (implying that alarms cannot be used directly to remedy the situation); only a human user is intended to take action on an alarm. For conditions that can be addressed by higher-level software, notable events are used.

A common alarm that most components can raise is a watchdog alarm. The watchdog is continually sending ping messages to the Command Handler. A failure to receive this means the component is not behaving correctly and therefore a watchdog alarm is raised to indicate that there is a problem. Since alarm states must also be refreshed at least once every 9 seconds, it is expected that these updates will be triggered by the ping message from the Watchdog (thus, implying a maximum period of 9 seconds for the Watchdog).

A component may raise alarms for a variety of component/instrument specific reasons or for other miscellaneous unexpected conditions that require an operator's attention or to report faults, that prevent the normal operation of the component.

These are events that are reported as status from the Commands Handler to the System Controller. These are special status or alarm events.

The format for the request for notable events is the same as described in Section 5.1.

Table 5-8- Alarm Events that are published

Event	Block	Description
watchdog	Command Handler	Some part of the RTC has become unresponsibe
pipelineDelay	Command Handler	Part of all of the RTC pipelines is running slow (exception is when pipeline rates are changed and it is settling)
pipelineFault	Command Handler	Part or all of the RTC pipeline is not running

6. FILE FORMATS OF READ-IN FILES

The following will detail the file formats for the following files:

- Dark, Bias and Flat files for both HO and LO WFS
- Reference offset file
- SRT-RPG configuration file
- Wavefront Corrector files
- Hardware configuration file
- SRT configuration settings
- Sub-Aperature Maks file
- NCPA file

It is likely that the format of these files will evolve over time.

Table 6-1 - HEART Input file formats

Name	Description	Format
DARK_FILE	Containing 1 set of darks for all WFS's	FITS file format containing extensions for first all HO WFS, then all LO WFS Keywords in the header will detail the contents
BIAS_FILE	Containing 1 set of bias files for all WFS's	FITS file format containing extensions for first all HO WFS, then all LO WFS Keywords in the header will detail the contents
FLAT_FILE	Containing 1 set of flats for all WFS's	FITS file format containing extensions for first all HO WFS, then all LO WFS Keywords in the header will detail the contents

REF_OFFS_FILE	Reference offsets	Text file, first line contains # off offsets, following lines contains the offsets
WFC_LAB_FLAT	Path/filename to a file containing WFC Lab flat	Text file, first line contains # of actuators, following lines contains the settings for each actuator
WFC_SYS_FLAT	Path/filename to a file containing WFC system flats	Text file, first line contains # of actuators, following lines contains the settings for each actuator
WFC_DM_SHAPE	Path/filename to a file containing DM shape	Text file, first line contains # of actuators, following lines contains the settings for each actuator
SRT_CONF_FILE	Path/filename to a file containing the SRT configuration	key/value pairs: - min/max loop gain - default fudge gain for reconstr - number PSD data points number segments to average
WFC APERTURE	Path/filename to a file containing WFC aperture pupil	Fits file format
HARDWARE_FILE	Contains hardware assignments for blocks	List a block name and priority
SRT_CONF_FILE	Path/filename to a file containing the SRT configuration, which contains key/value pairs: - min/max loop gain - default fudge gain for reconstructor - number of PSD data points number of segments to average	Lists the key word, then value.
SUBAPP_MASK_FILE	Sub-aperature mask file, fits format	Fits file format
NCPA_FILE	Non-common path file, if not using then this is a file full of 1's	Text file, first line contains number of parameters in the file, the following lines contain the data

7. ADDITIONAL COMMANDS NOT YET PROMOTED

This section covers commands and status that may graduate to the external interface during the build. It depends on the need for it. The second section are some commands that are specific to certain architectures.

7.1 COMMANDS

The following commands are relevant mostly to lower level commands or pyramid wavefront sensing. These commands may be promoted to official commands if the needs presents itself.

7.1.1 algoSetLgs

Description	sets the LGS gradient computation algorithm				
Type	simple				
Command Name	**algoSetLgs**				
Parameters (in)	Field	Description	Type	Units	Required
	algo	Selected LGS gradient algorithm	enum: (COG_STATIC, MF_UPDATE, MF_STATIC)		Yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true lgsState.enable[{at least one}] = true if {input enable == MF_UPDATE}, then loop.lgsDither != NONE if {input enable == MF_STATIC}, then lgsState.algo = MF_UPDATE Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

This command sets the LGS gradient computation algorithm.

When the LGS algorithm is set to MF_UPDATE, the LGS matched filter optimization is started. This algorithm can only be selected if LGS dithering is enabled (loop.lgsDither != NONE). If the previous algorithm was not MF_STATIC, then the LGS pixel processing does not have a set of matched filters. Therefore it will initially use CoG (lgsState.algo = MF_COG) and once a set of matched filters has been computed then they will be applied to the pixel processing (lgsState.algo = MF_UPDATE).

While matched filter optimization is enabled (lgsState.algo = MF_COG | MF_UPDATE) the LGS dither DLL integrators, i0 frames and matched filters will be updated.

While not optimizing (lgsState.algo != MF_COG | MF_UPDATE) the updating of optimization parameters will be suspended. Re-enabling the optimization will append new data to the previously gathered statistics; however RTC will delay the new statistic gathering so the new data lines up in dither signal phase with where the old data left off.

Enabling matched filter optimization will also trigger the updating of the MFU portion of the TWFS/MFU reference vector. However, while the telescope is preforming a dither, the RTC should be instructed by the System Controller to suspend the matched filter optimization by switching to MF_STATIC or COG_STATIC.

When the LGS algorithm is set to COG_STATIC, the LGS gradients will be generated using center of gravity. Any matched filter statistic gathering will be suspended, but not reset. This is the default state after activating the pipeline (via the pipelineActivate command).

When the LGS algorithm is set to MF_STATIC, the LGS gradients will be generated using the current matched filters. However, it will not update the current matched filters. Matched filter statistic gathering will be suspended, but not reset. This algorithm can only be selected if the previous state was MF_UPDATE.

Note that this command does not reset the LGS dither DLL integrators, the time-series of i(b) frames used to update the i(0) frame, or matched filters. To reset the LGS optimization parameters the loopParamRest command can be used.

7.1.2 algoSetNgs

<table>
<tr><td>Description</td><td colspan="5">sets the NGS gradient computation algorithm</td></tr>
<tr><td>Type</td><td colspan="5">simple</td></tr>
<tr><td>Command Name</td><td colspan="5">algoSetNgs</td></tr>
<tr><td rowspan="3">Parameters (in)</td><td>Field</td><td>Description</td><td>Type</td><td>Units</td><td>Required</td></tr>
<tr><td>algo</td><td>Selected NGS gradient computation algorithm</td><td>enum: (COG_STATIC, COG_UPDATE)</td><td></td><td>Yes</td></tr>
<tr><td colspan="5"></td></tr>
<tr><td>Parameters (out)</td><td colspan="5"></td></tr>
<tr><td>Status values affected</td><td colspan="5">Precondition:
state.cmd = READY
loop.ready = true
if {input enable == COG_UPDATE}, then loop.ngsDither != NONE
Execution:
state.cmd = BUSY
At Completion:
state.cmd = READY
ngsState.algo[] = {input algo}</td></tr>
</table>

Action and Response
This command sets the NGS gradient computation algorithm.

When the NGS algorithm is set to COG_UPDATE, the NGS optical gain optimization is started. This algorithm can only be selected if NGS dithering is enabled (loop.pwsfDither != NONE). However, while the telescope is preforming a dither, the RTC should be instructed by the System Controller to halt optical gain optimization by switching to COG_STATIC.

While optical gain optimization is enabled (i.e. ngsState.algo = COG_UPDATE), the NGS optical gain value and NCP dither DLL integrator will be updated. While not optimizing (i.e. ngsState.algo != COG_UPDATE), the updating of optimization parameters will be suspended. Re-enabling the optimization will append new data to the previously gathered statistics; however RTC will delay the new statistic gathering so the new data lines up in dither signal phase with where the old data left off.

When the NGS algorithm is set to COG_STATIC, the NGS gradients will be generated using center of gravity. This is the default state after activating the pipeline, via the pipelineActivate command.

Note that this command does not reset the NGS optical gain value or the NCP dither DLL integrator. To reset the optical gain the loopParamRest command can be used.

7.1.3 algoSetOiwfsOdgw

Note that this function may be renamed to algoSetLoNgs as part of a refactoring process to generalize the OIWFS and ODGW algorithm selection be suitable for any LO order NGS WFS.

<table>
<tr><td>Description</td><td colspan="5">sets the OIWFS/ODGW gradient computation algorithm</td></tr>
<tr><td>Type</td><td colspan="5">simple</td></tr>
<tr><td>Command Name</td><td colspan="5">algoSetOiwfsOdgw</td></tr>
<tr><td rowspan="3">Parameters (in)</td><td>Field</td><td>Description</td><td>Type</td><td>Units</td><td>Required</td></tr>
<tr><td>detector</td><td>OIWFS or ODGW detector</td><td>enum: (OIWFSA, OIWFSB, OIWFSC, ODGW1, ODGW2, ODGW3, ODGW4)</td><td></td><td>no</td></tr>
<tr><td>algo</td><td>OIWFS/ODGW gradient computation algorithm</td><td>enum: (COG_STATIC, MF_UPDATE, MF_STATIC, BP)</td><td></td><td>yes</td></tr>
<tr><td>Parameters (out)</td><td colspan="5"></td></tr>
<tr><td>Status values affected</td><td colspan="5">Precondition:
state.cmd = READY
loop.ready = true
oiwfsState.enable[{input detector}] != NONE
odgwState.enable[{input detector}] = true
if {input algo == MF_STATIC}, then oiwfsState.algo[{input detector}] = MF_UPDATEodgwState.algo[{input detector}] = MF_UPDATE
Execution:
state.cmd = BUSY
At Completion:
state.cmd = READY
oiwfsState.algo[{input detector}] = {input algo} | MF_COG
odgwState.algo[{input detector}] = {input algo} | MF_COG</td></tr>
</table>

Action and Response

This command sets the OIWFS/ODGW gradient computation algorithm.

When the OIWFS/ODGW algorithm is set to MF_UPDATE, the OIWFS/ODGW matched filter optimization is started. If the previous algorithm was not MF_STATIC, then the OIWFS/ODGW pixel processing does not have a matched filter. Therefore, it will initially use CoG (i.e.

oiwfs/odgwState.algo = MF_COG) and, once the matched filter has been computed, then it will be applied to the pixel processing (oiwfs/odgwState.algo = MF_UPDATE). However, while the telescope is preforming a dither or the detector is (re-)acquiring, the RTC should be instructed by the System Controller to halt the matched filter optimization by switching to MF_STATIC or COG_STATIC.

When the OIWFS/ODGW algorithm is set to BP, the OIWFS/ODGW gradients will be generated using the brightest pixel.

When the OIWFS/ODGW algorithm is set to COG_STATIC, the OIWFS/ODGW gradients will be generated using center of gravity (CoG).

When the OIWFS/ODGW algorithm is set to MF_STATIC, the OIWFS/ODGW gradients will be generated using the current matched filters. However, it will not update the current matched filters. This algorithm can only be selected if the previous state was MF_UPDATE.

Note that this command does not clear the current OIWFS/ODGW matched filters.

7.1.4 setLgsTtCntrl

<table>
<tr><td>Description</td><td colspan="5">sets the controller type used in the LGS TT loop</td></tr>
<tr><td>Type</td><td colspan="5">simple</td></tr>
<tr><td>Command Name</td><td colspan="5">setLgsTtCntrl</td></tr>
<tr><td rowspan="2">Parameters (in)</td><td>Field</td><td>Description</td><td>Type</td><td>Units</td><td>Required</td></tr>
<tr><td>cntrl</td><td>desired LGSF FSM TT controller</td><td>enum: (INTEGRATOR, KALMAN)</td><td></td><td>yes</td></tr>
<tr><td>Parameters (out)</td><td colspan="5"></td></tr>
<tr><td>Status values affected</td><td colspan="5">Precondition:
state.cmd = READY
loop.ready = true
lgsState.enable[{at least one}] = true
loop.lgsTt[] = IDLE | INACTIVE
Execution:
state.cmd = BUSY
At Completion:
state.cmd = READY
lgsState.tt = {input cntrl}</td></tr>
</table>

Action and Response

This command sets the controller type used in the LGS TT loop.

7.1.5 lgsfFsmZero

Description	sends a zero command to all LGSF FSMs				
Type	simple				
Command Name	**lgsfFsmZero**				
Parameters (in)	Field	Description	Type	Units	Required
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true loop.lgsTt[] = IDLE | INACTIVE Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

This command sends a zero command to all LGSF FSMs.

This command is rejected if the LGS TT loop is closed.

7.1.6 loopLgsFocus

Description	enables or disables the LGS focus loop, which publishing of the LGS trombone offset (lgsFocus) to control the NCC LGS trombone				
Type	simple				
Command Name	**loopLgsFocus**				
Parameters (in)	Field	Description	Type	Units	Required
	enable	indicates whether to enable or disable the LGS focus loop	boolean		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true lgsState.enable[{at least one}] = true Execution: state.cmd = BUSY At Completion: state.cmd = READY if {input enable} = true, then loop.lgsFocus != IDLE if {input enable} = false, then loop.lgsFocus = IDLE				

Action and Response

This command enables or disables the LGS focus loop, which is publishing the LGS focus offset (lgsFocus.

While the focus error is above the LGS focus threshold (as set in the config file) the focus loop state will be ACQUIRE. Once the focus error has dropped below the threshold, the focus loop state will change to LOCK. Note that if the sodium layer estimate and, therefore, prepositioning of the LGS trombone is sufficient, then the initial focus should be below the threshold, and focus loop will be locked immediately. If the LGS focus loop is not IDLE, an enable request will be a no-op.

7.1.7 integratorControl

Description	Control the SRT integrators				
Type	simple				
Command Name	**integratorControl**				
Parameters (in)	Field	Description	Type	Units	Required
	name	Integrator name (e.g., LO Pipeline or HO Pipeline)	string		yes
	Control	Start, stop, resume or reset	enum {START | STOP | RESUME | RESET}		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

Start, stop, reset, or resume the integrator defined.

7.1.8 loopLowTier0

Description	enables or disables Tier 0, the low-order loop using the NGS to control TT and focus				
Type	simple				
Command Name	**loopLowTier0**				
Parameters (in)	Field	Description	Type	Units	Required
	enable	indicates whether to enable or disable the Tier 0 loop	boolean		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true loop.tier1 = IDLE loop.tier2[] = IDLE | INACTIVE calibWc.override = false Execution: state.cmd = BUSY At Completion: state.cmd = READY if {input enable}, then loop.lo != IDLEloop.tier0 != IDLE if {input !enable}, then loop.lo = IDLEloop.tier0 = IDLE				

Action and Response

This command enables or disables Tier 0, the low-order loop using the NGS to control TT and focus.

This command is intended to be used as an optional first step in low-order acquisition.

If the Tier 1 and/or Tier 2 loops are already closed, or are acquiring, this command cannot be executed and will return an error. If either Tier 1 or 2 loops are closed while Tier 0 is active, then a hand-off procedure is triggered and the Tier 0 loop will be opened.

Disabling the Tier 0 loop merely stops the NGS update of low-order loop and does not impact any other NGS operation.

7.1.9 loopLowTier1

Description	enables or disables Tier 1, the low-order loop using the Tier 1 detector, i.e. the LO TTF detector				
Type	simple				
Command Name	**loopLowTier1**				
Parameters (in)	Field	Description	Type	Units	Required
	enable	indicates whether to enable or disable the Tier 1 loop	boolean		yes
Parameters (out)					

Status values affected	Precondition: state.cmd = READY loop.ready = true calibWc.override = false if {loop.tier1 == OIWFS*}, then oiwfsState.acqTable[OIWFS*] = true Execution: state.cmd = BUSY At Completion: state.cmd = READY if {input enable}, then loop.lo != IDLEloop.tier0 = IDLEloop.tier1 != IDLE if {input !enable}, then loop.tier1 = IDLEloop.tier3[] = IDLE | INACTIVEloop.tier3f = IDLE | INACTIVE

Action and Response

This command enables or disables the low-order loop which applies low-order correction errors to the control path from the Tier 1 detector, i.e. the LO TTF detector.

Enabling this loop triggers the Tier 1 acquisition process that steps through the Tier 1 acquisition table. While stepping through this table, the Tier 1 loop state is set to ACQUIRE. Once the acquisition process is complete the state is set to LOCK.

If the Tier 0 loop is closed, then a hands-off procedure is performed to transition TTF control from the NGS to the Tier 1 detector. If the NGS is the Tier 1 detector, then this hands-off procedure is a no-op.

If enabled, and the Tier 1 detector is already acquiring or locked, this command is a no-op. Additionally, if the Tier 1 detector is the NGS then the ACQUIRE state is skipped since the NGS does not have an acquisition procedure. If the NGS is configured as the Tier 1 detector, disabling Tier 1 merely stops the NGS control in the low-order path and does not impact any other NGS operation.

Disabling Tier 1 will also disable Tier 3.

7.1.10 loopLowTier2

Description	enables or disables Tier 2, the low-order loop using the Tier 2 detectors, i.e. the LO TT/TTF detectors				
Type	simple				
Command Name	**loopLowTier2**				
Parameters (in)	Field	Description	Type	Units	Required
	enable	indicates whether to enable or disable the Tier 2 loop	boolean		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true calibWc.override = false if {loop.tier2[] == ODGW*}, then odgwState.acqTable[ODGW*] = true				

	if {loop.tier2[] == OIWFS*}, then oiwfsState.acqTable[OIWFS*] = true Execution: state.cmd = BUSY At Completion: state.cmd = READY if {input enable}, then if {at least one mode.tier2[] != NONE}, then loop.lo != IDLEloop.tier0 = IDLEloop.tier2[] != IDLE if {input !enable}, then loop.tier2[] = IDLE | INACTIVEloop.tier3[] = IDLE | INACTIVEloop.tier3f = IDLE | INACTIVE

Action and Response

This command enables or disables the low-order loop which apples low-order correction errors to the control path from the Tier 2 detectors, i.e. the LO TT/TTF detectors.

Enabling this loop triggers the Tier 2 acquisition process that steps through the Tier 2 acquisition tables. While stepping through these tables, the Tier 2 loop is set to ACQUIRE. Once the acquisition process is complete the state is set to LOCK.

If enabling the Tier 2 loop but the detector is inactive (mode.tier2[] = NONE), then that detector is skipped and the corresponding Tier 2 state will be INACTIVE. If a Tier 2 detector is already acquiring or locked, this command is a noop for that detector and will proceed to the next Tier 2 detector.

If the Tier 0 loop is closed, then a hands-off procedure is performed to transition TT/TTF control from the NGS to the Tier 2 detector(s). If the NGS is the Tier 2 detector, then this hands-off procedure is a no-op and will proceed to the next Tier 2 detector. If NGS in configured as a Tier 2 detector then disabling Tier 2 merely stops the NGS control in the low-order path, and does not impact any other NGS operation.

Disabling Tier 2 will also disable Tier 3.

7.1.11 loopLowTier3

Description	enables or disables Tier 3, the low-order loop using the Tier 3/3F detectors, i.e. the LOT detectors				
Type	simple				
Command Name	**loopLowTier3**				
Parameters (in)	Field	Description	Type	Units	Required
	enable	indicates whether to enable or disable the Tier 3 loop	boolean		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true loop.tier1 != IDLE loop.tier2[] != IDLE calibWc.override = false				

	if {loop.tier3[] == ODGW*}, then odgwState.acqTable[ODGW*] = true if {loop.tier3[] == OIWFS*}, then oiwfsState.acqTable[OIWFS*] = true if {loop.tier3f == OIWFS*}, then oiwfsState.acqTable[OIWFS*] = true Execution: state.cmd = BUSY At Completion: state.cmd = READY if {input enable}, then loop.lo != IDLEloop.tier3[] != IDLEloop.tier3f != IDLE if {input !enable}, then loop.lo != IDLEloop.tier3[] = IDLE | INACTIVEloop.tier3f = IDLE | INACTIVE

Action and Response

This command enables or disables the low-order loop which applies low-order correction errors to the control path from the Tier 3 and Tier 3F detectors, i.e., the LOT detectors.

Enabling this loop triggers the Tier 3 and Tier 3F acquisition process that steps through the acquisition tables. While stepping through these tables, the Tier 3 loop state is set to ACQUIRE. Once the acquisition process has been completed the state is set to LOCK.

Tier 3 can only be closed after Tier 1 and all active Tier 2 loops have been started, however Tier 3 acquisition will not start until after Tier 1 & 2 detectors have completed their acquisition (loop.tier1 = LOCK and loop.tier2[] = LOCK | INACTIVE). Therefore, if there are no active Tier 2 detectors, only Tier 1 must be acquired before closing Tier 3 loops.

If enabling is requested and a Tier 3 or 3F detector is inactive (mode.tier3[] = NONE or mode.tier3f = NONE), that detector is skipped and the corresponding Tier state will be INACTIVE. If a Tier 3 detector is already acquiring or locked, this command is a no-op for that senor and will proceed to the next Tier 3 detector.

Enabling the Tier 3 loop will set the LO path filter to a high-pass filter, and the LOT path filter to a low-pass filter. Conversely, disabling Tier 3 will set the low-order mode filter to an all-pass filter, and the low-order truth filter to be an all-stop filter.

7.1.12 paramConfigSetSRT

Description	Set or save SRT configuration parameters, used for engineering purposes				
Type	simple				
Command Name	**paramConfigSetSRT**				
Parameters (in)	Field	Description	Type	Units	Required
	set	If set, this will set the SRT parameter, otherwise it will save it to file	boolean		yes
	param	Configuration parameter identifier	string		yes
	value	New value for the specified parameter as a string	string		
	file	Filename to write to when save is selected			
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

Set/save SRT configuration parameters, including reconstructor parameters and reconstructor mode (i.e. MCAO, GLAO, or LTAO mode).

7.2 COMMANDS SPECIFIC TO ARCHITECTURE

7.2.1 loopLgsDither

Description	enables or disables LGS dithering				
Type	simple				
Command Name	**loopLgsDither**				
Parameters (in)	Field	Description	Type	Units	Required
	enable	indicates whether to enable or disable the LGS Dithering loop	enum: (CP, NCP, NONE)		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true lgsState.enable[{at least one}] = true Execution:				

	state.cmd = BUSY At Completion: state.cmd = READY loop.lgsDither = {input enable} if {input enable == NONE and lgsState.algo == MF_COG | MF_UPDATE}, then lgsState.algo = COG_STATIC | MF_STATIC

Action and Response

This command enables or disables LGS dithering.

If non-common path (NCP) dithering is selected, then the RTC assumes that LGSF has, or will be, instructed to dither the LGS FSMs. The RTC will then listen to the LGS FSM sensed positions stream from the LGSF and estimate the dither signal for optimization and removal from the LGS WFS gradients. If the LGS TT loop is open (i.e. loop.lgsTt[] == IDLE), then the RTC will send the current LGSF FSM command, as stored in the FSM integrator or Kalman filter, to trigger the LGSF FSM sensed position response.

If common path (CP) dithering is selected, then the CP LGS dither signal (tip/tilt) is applied to DM0. The CP dither signal is applied to the actuator commands sent to the DM. If the High and Low-Order loops are stopped (i.e. loop.ho = IDLE and loop.lo = IDLE) before or during the application of the CP LGS dither signal, the dither signal is superimposed on the current DM0 shape. This is taken from what is stored in the main wavefront corrector integrator. This dither signal is completely independent of the NGS dither signal described in the loopNgsDither command.

If NONE is selected, then the RTC will not dither DM0 and it will ignore the dither signal applied to the LGSF FSMs. If the LGS WFS gradient optimization algorithm is set to matched filter optimization (i.e. lgsState.algo == MF_COG | MF_UPDATE) and the dithering is disabled, then the LGS WFS gradient optimization is halted (i.e. lgsState.algo == COG_STATIC | MF_STATIC).

7.2.2 loopTwfs

Description	enables or disables the TWFS loop which updates the LGS WFS TWFS/MFU reference vector based on NGS measurements				
Type	simple				
Command Name	**loopTwfs**				
Parameters (in)	Field	Description	Type	Units	Required
	cntrl	indicates whether to enable or disable the TWFS loop	boolean		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true lgsState.enable[{at least one}] = true Execution: state.cmd = BUSY At Completion: state.cmd = READY				

	if {input enable}, then loop.twfs != IDLE if {input !enable}, then loop.twfs = IDLE

Action and Response

This command enables or disables the TWFS loop which updates the LGS WFS TWFS/MFU reference vector based on NGS measurements.

If enabled, the TWFS portion of the TWFS/MFU reference vector is updated based on NGS measurements. If the TWFS loop is not IDLE, an enable request will be a no-op. Note that the MFU portion of the TWFS/MFU reference vector is updated independently, which is enabled when LGS MF are being built/updated (i.e. lgsState.algo = MF_COG | MF_UPDATE).

Disabling the TWFS loop does not reset the TWFS/MFU reference vector stored in the internal integrator, but merely stops any further TWFS updates from NGS measurements. To reset the TWFS/MFU reference vector, use the command loopParamReset.

Disabling the TWFS reference vector merely stops updating the TWFS reference vector and does not impact any other NGS operation.

7.2.3 loopNgsDither

Description	enables or disables NGS dithering				
Type	simple				
Command Name	**loopNgsDither**				
Parameters (in)	Field	Description	Type	Units	Required
	enable	indicates whether to enable or disable the NGS Dithering loop	enum: (CP, NCP, NONE)		yes
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true ngsState.enable[{at least one}] = true Execution: state.cmd = BUSY At Completion: state.cmd = READY loop.ngsDither = {input enable} if {input enable == NONE and ngsState.algo == MF_COG | MF_UPDATE}, then ngsState.algo = COG_STATIC | MF_STATIC				

Action and Response

This command enables or disables NGS dithering.

If non-common path (NCP) dithering is selected, then the RTC assumes that NCC has, or will be, instructed to dither the NGS FSM. The RTC will then listen to the NGS FSM sensed position stream from the NCC and estimate the dither signal for optimization and removal from the NGS gradients.

If common path (CP) dithering is selected, then the CP NGS dither signal is applied to DM0. The CP dither signal is applied to the actuator commands sent to the DM. If the High and Low-Order loops are stopped (loop.ho=IDLE & loop.lo=IDLE) before or during the application of the CP dither signal, the dither signal is superimposed on the current DM0 shape, as stored in the main wavefront corrector integrator. This dither signal is completely independent of the LGS dither signal described in the loopLgsDither command.

If NONE is selected, then the RTC will not dither DM0 and will ignore the NGS FSM position stream. If the NGS gradient optimization algorithm is set to optical gain optimization (oiwfsState.algo == COG_UPDATE) and the dithering is disabled, then the NGS gradient optimization is halted (oiwfsState.algo == COG_STATIC).

7.2.4 offloadOiwfsPoa

<table>
<tr><td>Description</td><td colspan="5">enables or disables offloading to the OIWFS POAs</td></tr>
<tr><td>Type</td><td colspan="5">simple</td></tr>
<tr><td>Command Name</td><td colspan="5">offloadOiwfsPoa</td></tr>
<tr><td rowspan="4">Parameters (in)</td><td>Field</td><td>Description</td><td>Type</td><td>Units</td><td>Required</td></tr>
<tr><td>enable</td><td>indicates whether to enable or disable the OIWFS POA offloading loop. The array is ordered.</td><td>Array[3] of boolean</td><td></td><td>yes</td></tr>
<tr><td>time</td><td>timestamp indicating when the offload stage should change</td><td>double</td><td>TAI / PTP</td><td>no</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Parameters (out)</td><td colspan="5"></td></tr>
<tr><td>Status values affected</td><td colspan="5">Precondition:
state.cmd = READY
loop.ready = true
Execution:
state.cmd = BUSY
At Completion:
state.cmd = READY
if {input enable}, then loop.oiwfsPoa[] != IDLE
if {input !enable}, then loop.oiwfsPoa[] = IDLE</td></tr>
</table>

Action and Response

This command enables or disables offloading to the OIWFS POAs.

Normally the POA offload will only be activated for OIWFSs in Tier 1 and 2. However, to facilitate calibration procedures, the POA offloading can be enabled whenever the corresponding OIWFS is sending pixels to the RTC.

If enabled, and the OIWFSs have not acquired their guide stars, the acquisition process is started. If the guide stars are already acquired, or are acquiring as a result of Tiers 1 or 2 being enabled, then the acquisition process is allowed to complete.

When OIWFS POA offloading is enabled, the RTC will publish the corresponding POA offloading parameter (oiwfsAPoa, oiwfsBPoa, or oiwfsCPoa). Whenever the offloading state (loop.oiwfsPoa[]) transitions into the ACTIVE state, the corresponding LPF will be reset.

Disabling the POA offloading merely stops any offloading sent to the POA, and does not impact any other OIWFS operation.

7.2.5 ngsTelDither

<table>
<tr><td>Description</td><td colspan="5">sets or clear the flag indicating that the telescope is performing a dither</td></tr>
<tr><td>Type</td><td colspan="5">simple</td></tr>
<tr><td>Command Name</td><td colspan="5">ngsTelDither</td></tr>
<tr><td rowspan="4">Parameters (in)</td><td>Field</td><td>Description</td><td>Type</td><td>Units</td><td>Required</td></tr>
<tr><td>dither</td><td>indicates whether the telescope is dithering or not</td><td>boolean</td><td></td><td>yes</td></tr>
<tr><td>gain</td><td>gain applied to the NGS gradients while the telescope is performing a dither</td><td>double</td><td></td><td>no</td></tr>
<tr><td>time</td><td>timestamp indicating when the specified dither flag takes effect</td><td>double</td><td>TAI / PTP</td><td>no</td></tr>
<tr><td>Parameters (out)</td><td colspan="5"></td></tr>
<tr><td>Status values affected</td><td colspan="5">Precondition:
state.cmd = READY
loop.ready = true
Execution:
state.cmd = BUSY
At Completion:
state.cmd = READY
oiwfsState.telDitherFlag = {input dither}
if {input gain is specified}, then oiwfsState.telDitherGain = {input gain}
else oiwfsState.telDitherGain = {gain from config file}</td></tr>
</table>

Action and Response

This command sets or clear the flag indicating that the telescope is performing a dither.

While the telescope is dithering, the NGS gradients will be scaled by a ≤1 gain. When the telescope is not dithering the gradient gain will be unity.

The gain input can only be specified when the dither input is set to true, otherwise the command will be rejected.

If the gain input is not specified, then the NGS telescope dither gradient gain from the RTC configuration file is used, otherwise the specified value is used instead of the value from the configuration file.

7.2.6 guideStarHandOff

<table>
<tr><td>Description</td><td colspan="5">disables or (re-)enables the NGS or a OIWFS or a ODGW a guide star for a hand-off procedure during non-sidereal tracking</td></tr>
<tr><td>Type</td><td colspan="5">simple</td></tr>
<tr><td>Command Name</td><td colspan="5">guideStarHandOff</td></tr>
<tr><td rowspan="3">Parameters (in)</td><td>Field</td><td>Description</td><td>Type</td><td>Units</td><td>Required</td></tr>
<tr><td>detector</td><td>select the detector used</td><td>enum: (NGS, OIWFSA, OIWFSB, OIWFSC, ODGW1, ODGW2, ODGW3, ODGW4)</td><td></td><td>yes</td></tr>
<tr><td>disable</td><td>indicates whether to disable or (re)enable the detector for guide start hand-off</td><td>Boolean</td><td></td><td>yes</td></tr>
<tr><td>Parameters (out)</td><td colspan="5"></td></tr>
<tr><td>Status values affected</td><td colspan="5">Precondition:
state.cmd = READY
loop.ready = true
oiwfsState.enable[{input detector}] != NONE
odgwState.enable[{input detector}] = true
Execution:
state.cmd = BUSY
At Completion:
state.cmd = READY
if {input detector == NGS} then oiwfsState.handOff = {input disable}
oiwfsState.handOff[{input detector}] = {input disable}
odgwState.handOff[{input detector}] = {input disable}</td></tr>
</table>

Action and Response

This command disables or (re-)enables the NGS, an OIWFS, or an ODGW guide star for a hand-off procedure for during non-sidereal tracking.

When disabled, the RTC temporarily ignore the detector stream, treating it as a loss of measurement, without raising alarms. When (re-)enabled, the RTC will attempt to acquire the guide star following the standard procedure.

7.2.7 rtsDelete

Description	deletes data stored on the RTS, including PSFR data and nightly telemetry				
Type	simple				
Command Name	**rtsDelete**				
Parameters (in)	Field	Description	Type	Units	Required
	psfr	Delete all PSFR data that is older than the specified number of hours. A value of zero will delete all PSFR data (TBC)	integer	hours	no
	telem	Delete all nightly telemetry data that is older then the specified number of hours. A value of zero will delete all nightly telemetry data (TBC)	integer	hours	no
	tag	Delete tagged data stored on the RTS (TBC)	string	TBD	no
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = false Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

This command deletes data stored on the RTS, including PSFR data and nightly telemetry.

7.2.8 activateSRT

Description	This will activate different areas of the SRT for engineering purposes				
Type	simple				
Command Name	**activatesSRT**				
Parameters (in)	Field	Description	Type	Units	Required
	enable	indicates whether to enable or disable the SRT	boolean		yes
	val	Generic field to be used for debugging	string		
Parameters (out)					
Status values affected	Precondition: state.cmd = READY loop.ready = true Execution: state.cmd = BUSY At Completion: state.cmd = READY				

Action and Response

This will activate a portion of the SRT for debugging purposes during build. This may include processing of frames independent of the HRT.

7.3 STATUS

The following are status values that may be promoted to the external interface, but currently are not.

Table 7-1 – Status structure specific to architecture

Status name	Desc	Block/Process/ Pipeline
Tier1	Tier 1 detector assignment	enum: (NGS, OIWFSA, OIWFSB, OIWFSC)
tier2	Tier 2 detector assignment. The array is ordered Tier 2 [A B].	array[2] of enum: (NONE, NGS, OIWFSA, OIWFSB, OIWFSC, ODGW1, ODGW2, ODGW3, ODGW4)
tier3	Tier 3 detector assignment. The array is ordered Tier 3 [A B C D].	array[4] of enum: (NONE, OIWFSA, OIWFSB, OIWFSC, ODGW1, ODGW2, ODGW3, ODGW4)
tier3f	Tier 3F detector assignment	enum: (NONE, OIWFSA, OIWFSB, OIWFSC)
psfrDataReady	An event signaling the PSFR that new data is ready to be processed	
m1Scallop	Primary mirror scalloping mode	
odgwImage	ODGW image size and rotation. Arrays are ordered as ODGW [1 2 3 4].	

	The size is defined by the widths of the elliptical Gaussian image, while the rotation defines the angle between the ellipse σ1 axis and the local probe coordinate system X axis. The ellipticity information is indicative of ADC errors and is published as a diagnostic or potentially as an error signal to adjust the ADC.	
oiwfsState	OIWFS state. Arrays are ordered OIWFS [A B C].	
odgwState	ODGW state. Arrays are ordered ODGW [1 2 3 4].	
oiwfsImage	OIWFS image size and rotation. The size is defined by the widths of the elliptical Gaussian image, while the rotation defines the angle between the ellipse σ1 axis and the local probe coordinate system X axis. Arrays are ordered OIWFS [1 2 3].	
oiwfs1Poa	OIWFS 1 POA offset (o_{POA}). The RTC publishes OIWFS residual probe position error values when performing probe offloading. Discussion: The primary use case for probe offloading is observing modes for which low-frequency TT truth is determined exclusively from ODGWs; residual errors reflect drifts in the OIWFS probes with respect to the IRIS Imager in this case. Discussion: Probe offloading will be used for calibration purposes so that the RTC can directly position the probes over calibration sources itself. Discussion: Probe offloading may be useful while performing closed-loop dithering with the probes. If there is a large error between the TCS position demands and the actual guide star locations, residual pointing errors may saturate the TTS. In this case, this offloading mechanism may be used to “self-guide” the probes, keeping the loops closed. Discussion: The parameter oiwfs1Poa will be updated at the specified rate. When the rate parameter changes all subsequent updates of oiwfs1Poa will occur at that new rate. A rate of 0 Hz means the RTC will no longer send subsequent offload parameter updates to IRIS. However at some point in future, the rate parameter may change again to a non-zero value, and the offload parameter updates will resume at the new rate. This rate will depend on the steady-state rate of low-order TT rate in the RTC.	

oiwfs2Poa	OIWFS 2 POA offset (o_{POA}). The RTC publishes OIWFS residual probe position error values when performing probe offloading. Discussion: See discussion and full description of equivalent attributes in oiwfs1Poa.	
oiwfs3Poa	OIWFS 3 POA offset (o_{POA}). The RTC publishes OIWFS residual probe position error values when performing probe offloading. Discussion: See discussion and full description of equivalent attributes in oiwfs1Poa.	
oiwfsState	OIWFS state. Arrays are ordered OIWFS [A B C].	

Table 7-2 – Notable Events specific to architecture

Event	Block	Description
NGS injected dither signal timeout	Low order NGS processing	Injected dither signal is late
OIWFS<A..C> pixel datagram out of order	Low order NGS processing	At least one OIWFS pixel UDP datagram was received out of order.
OIWFS<A..C> pixel datagram CRC error	Low order NGS processing	A bad CRC was detected in at least one OIWFS pixel UDP datagram.
OIWFS<A..C> low-flux	Low order NGS processing	Insufficient flux detected on OIWFS.
OIWFS<A..C> guard row	Low order NGS processing	Guide star detected in gruad row.
missing ODGW<1..4> pixel datagram	Low order NGS processing	At least one ODGW pixel UDP datagram was missing from the frame.
ODGW<1..4> pixel datagram out of order	Low order NGS processing	At least one ODGW pixel UDP datagram was received out of order.
ODGW<1..4> pixel datagram CRC error	Low order NGS processing	A bad CRC was detected in at least one ODGW pixel UDP datagram.
ODGW<1..4> low-flux	Low order NGS processing	Insufficient flux detected on ODGW.
ODGW<1..4> guard row	Low order NGS processing	Guide star detected in gruad row.
missing HO DM vector datagram	High order reconstruction	At least one HO DM vector UDP datagram was missing from the frame.