PALM-3000
Message Types Document
(Commands, Automations, Telemetry, and Status)
Version 0.2

Document Owner: Rick Burruss

Contributions by:
J. Angione, A. Bouchez, S. Guiwits, D. Hale, J. Roberts, M. Troy, T. Truong

Revision Sheet

Revision No.	Date	Revision Description
0.0	03/10/09	Initial Document (RB)
0.1	03/27/09	First set of revisions (RB)
0.2	05/04/09	Second set of revisions (RB)

Contents

1 GENERAL INFORMATION

1.1 Purpose

This document presents the Message Types for the PALM-3000 adaptive optics system. It contains all of the message types to be used by the P3K external devices and subsystems. The message types are separated into commands, status structures, and telemetry structures. This document also details all of the automations and the automation algorithms used by the P3K instrument.

1.2 Acronyms and Abbreviations

ACAM	Acquisition Camera component
AO	Adaptive Optics
AOCA	Adaptive Optics Command Dispatch and Automations component
AODB	Adaptive Optics Database components
AODP	Adaptive Optics Database Publisher
AODS	Adaptive Optics Database Subscriber
AODD	Adaptive Optics Device Driver component
GUI	Graphical User Interface
HODM	High Order Deformable Mirror
HWFP	High Order Wave Front Processor component
IDL	Interactive Data Language scripting language
LODM	Low Order Deformable Mirror
P3K	PALM-3000
PALAO	The original NGS AO system at Palomar commissioned in December 1999
PALM-3000	The visible light AO upgrade to PALMAO
PALMAO	Upgrades to PALAO, particularly after the April 2003 upgrade
TTM	Tip/tilt Mirror

1.3 Related Products

- P3K Software Architecture Document; T. Truong
- P3K TWiki Pages at: http://www.oir.caltech.edu/twiki_oir/bin/view.cgi/Palomar/Palm3000/WebHome

2 STATEMENT OF NEED

2.1 Introduction

The P3K control software messaging types, error codes, and automations need to be clearly defined in order to present the messaging design from a high level perspective to the development team. A clear messaging design will help the P3K development meet the software requirements, preserve the software architecture, and clarify communication between components.

2.2 Message Types

The primary function of defining message types is to distinguish and enumerate system level messaging for the P3K instrument. P3K message types are divided into three main classes:

- Commands
  - Commands will be issued from external devices (GUI, Pharo, Science Instruments, IDL) over socket connections to the P3K Command Dispatcher (AOCA). If the command exists in the automations command list, then the automation is started. Otherwise the command is published for the appropriately subscribed sub-system components.

- Status Structures
  - Each of the "low level" components will ***publish*** a structure of status fields appropriate to its state. The HWFP structure will also include all of the requested telemetry data. The AODP component ****may**** extract the HWFP status information into a unique status structure, and then publish both the status and telemetry structures separately (see HWFP_stat).

- External Structures
  - These status structures will be maintained by AOCA and delivered to external components over a standard Tcp/IP socket connection

- Info Structures
  - Each of the "low level" components will ***publish*** a small status structure containing a short list of non-changing values unique to the component. These information structures will only be published ONCE at system startup, and will contain useful information like firmware version, linux kernel version, and software version numbers.

- Telemetry Structures
  - The HWFP and ACAM components will publish telemetry data. The HWFP structure will also include its status fields appropriate to its state. The AODP component **may** extract the HWFP telemetry data into a unique telemetry structure(s), and then publish both the status and telemetry structures separately (see HWFP_data).

2.3 Error Codes

Error codes will be defined globally for errors common to the entire P3K system. Each low level component (MOTOR, ACAM, etc) will also include error status and strings unique to that component in the interest of detail. External devices such as the GUI may display some or all of these fields as appropriate, whereas all of the information will be published. System health may be gleaned from error codes, and the system may respond to such cases, although this is not yet in any AOCA or other high level design.

2.4 Automations

Automations are defined as a set of two or more primitive P3K commands. An automation may also subscribe to and evaluate telemetry data, and execute various algorithms. An automation may publish a status structure appropriate to its state. Some automations require that the P3K system be blocked while running. The AOCA component will block all external devices from connecting to it when an automation requires blocking. Blocked automations may be canceled while they are running. AOCA will monitor the state of all the automations (idle, running, or finished), and publish a status structure with this information.

The full list of automations and their parameters are listed in section 5 below.

3 MESSAGE TYPES BREAKDOWN

3.1 Message Type : Commands

This is the list of the primitive commands that external devices such as the GUI and the Science Cameras, as well as external scripts such as IDL, will pass to the AOCA component over a socket connection. The AOCA component will publish these primitive commands.

Command Syntax:

COMMAND <Required Parameter> {Optional Parameter}

3.1.1 HWFP_cmd

These commands are published by AOCA and subscribed to by the High Order Wavefront Processor (HWFP) component. They are intended to command the real time processes for P3K, such as tip / tilt, low and high order deformable mirror control, table loading, system telemetry logging, and real time parameter settings.

Complete HWFP Syntax:

HWFP TTM {ON|OFF} {MODE = 1 : i}*

HWFP LODM {ON|OFF} {MODE = 1 : i}*

HWFP HODM {ON|OFF} {MODE = 1 : i}*

HWFP LOAD <TABLE = filename>

<TABLE> = <CENT_OFFSETS | PIX_OFFETS | HO_DM_MAP | LO_DM_MAP | PIX_GAINS | RECONST>

HWFP SET <PARAM = float>

<PARAM> = <HODM_PRO_GAIN | HODM_INT_GAIN |
LODM_PRO_GAIN | LODM_INT_GAIN | TTM_PRO_GAIN |
TTM_INT_GAIN | SUBAP_MIN_FLUX | HODM_MODE | LODM_MODE>

HWFP LOG <DATA = telem> {INTERVAL = 1 : 500}

<telem> = <(PIXELS, CENTROIDS, SUBAP_FLUX, HODM_POS,
LODM_POS, HODM_RES, LODM_RES, TTM_POS, TTM_RES, WFP_STAT)
| NONE | ALL>

* MODE is not yet defined

3.1.2 HWFS_cmd

These commands are published by AOCA and subscribed to by the Device Driver component (AODD) and the HWFP component (to know when HWFS in ON or OFF). AODD then passes the command to the High Order Wave Front Sensor (HWFS) component, which controls a SciMeasure camera similar to the PalAO WFS camera.

Complete HWFS Syntax:

HWFS {ON|OFF} {RATE = 1 : ?} {BIAS = ?} {GAIN = ? : ?} {DELAY = ?}
{CLAMP = ?} {FUNCTION = 1:16} {OFFSET = ?} {INFO}

3.1.3 ACAM_cmd

These commands are published by AOCA and subscribed to by the Device Driver component (AODD), which then passes the commands to the Acquisition Camera (ACAM) component, which controls an Imperx CCD Camera over a ??? interface.

Complete ACAM Syntax:

ACAM {ON|OFF} {INT_TIME = 0.0 : ?} {RATE = 1 : ?} {WINDOW = 1 (full 1x1) | 2
(full 2x2) | 3 (full 4x4) | 4 (acq 4x4) | 5 (pupil 2x2} {TEST = 0 | 1} {BACK_SUB = 0 | 1}
{BACK_FILE = file_path} {INFO}

3.1.4 HODM_cmd

These commands are published by AOCA and subscribed to by the Device Driver component (AODD), which then passes the commands to the High Order Deformable Mirror (3000 actuator DM) over the HODM “slow speed” serial interface.

Complete HODM Syntax:

HODM {ENABLE | DISABLE | RESET}

3.1.5 MOTOR_cmd

These commands are published by AOCA and subscribed to by the Device Driver component (AODD), which then passes the commands on to the MOTOR component.

Complete MOTOR Syntax:

MOVE <MOTOR> <POSITION> {VELOCITY = 0.0 : ?}

OFFSET <MOTOR | VIRTUAL_MOTOR> <OFFSET> {VELOCITY = 0.0 : ?}

HOME <MOTOR>

STOP <MOTOR>

<MOTOR> = <ACQ | FQ_RELAY | FQ_STIM | STIM | SSM1_A | SSM1_B | SSM2_A | SSM2_B | WFS_Z | WFS_LENS | WHITE_X | WHITE_Y | WHITE_Z | LASER_Z | TTM_A | TTM_B>

<VIRTUAL_MOTOR> = <IMAGE_X | IMAGE_Y | PUPIL_X | PUPIL_Y>

3.1.6 WLIGHT_cmd

These commands are published by AOCA and subscribed to by the Device Driver component (AODD), which then passes the commands on to the WLIGHT component.

Complete WLIGHT Syntax:

WLIGHT {ON|OFF} {PWR = 0 : 100}

3.1.7 TCS_cmd

These commands are passed from AOCA directly to the TCS through a socket connection opened by the AOCA "TCS CMD_task".

Complete TCS Syntax:

TCS <TCS COMMAND>

3.1.8 PHARO_cmd

These commands are passed from AOCA directly to PHARO through a NetServices socket connection wrapper, and opened by the AOCA "PHARO CMD_task". The NetServices wrapper will also parse incoming commands from PHARO into primitive AOCA commands.

Complete PHARO Syntax:

PHARO <PHARO COMMAND>

3.1.9 INST_cmd

These commands are passed from AOCA directly to a science instrument through the AOCA socket connection. *This implementation will be delayed until a need is required and defined.*

Complete INST Syntax:

INST <INST COMMAND>

3.1.10 Automation Commands

These commands are passed from AOCA directly to the called Automation "AFn start_task". See section 5 for a complete list of these commands.

3.1.11 SHUTDOWN_cmd

This command is intended to gracefully shutdown the P3K software or one of the sub system processes. SHUTDOWN with no arguments will close all processes. (should we include a RESTART command to restart individual or collections of processes? This also clarifies that low level components should run as unique processes)

Complete SHUTDOWN Syntax:

SHUTDOWN {PROCESS}

{PROCESS} = {ALL | AOCA,AODD,AODP,AODS}

3.2 Message Type : Status

3.2.1 HWFP_stat

The HWFP component will publish a structure containing ALL of the HWFP data at the frame rate specified by the HWFS frame rate. The HWFP LOG_INTERVAL parameter may be set to > 1 to slow the data output down, but the output rate will generally be high. If deemed necessary by the RTC engineer (ie: GUI display requirements), the AODP component **may** split the HWFP data into two or more structures. This status structure is an EXAMPLE of one such split option (see 3.3.1 HWFP_data below). Following is a list of necessary status variables that are typically either single values or unchanged over long periods of time.

HWFP Status Structure:

```
typedef struct {
        float ttm_int_gain          /* TTM integral gain */
        float ttm_pro_gain          /* TTM proportional gain */
        float hodm_int_gain         /* HODM integral gain */
        float hodm_pro_gain         /* HODM proportional gain */
        float lodm_int_gain         /* LODM integral gain */
        float lodm_pro_gain         /* LODM proportional gain */
        float subap_min_flux        /* subaperture minimum flux */
        int ho_mode                 /* HODM reconstruct mode TBD */
        float ho_rss_res            /* HODM root sum square for line plots */
        int lo_mode                 /* LODM reconstruct mode TBD */
```

```
        float lo_rss_res                /* LODM root sum square for line plots */
        int log_types                   /* telemetry types to log */
        int log_rate                    /* data logging rate (necessary ?) */
        int ttm_rate                    /* TTM rate */
        int ttm_on                      /* ON | OFF */
        int hodm_on                     /* ON | OFF */
        int lodm_on                     /* ON | OFF */
        char  pix_gains_file[SIZE]      /* gain table from HWFS camera */
        char  pix_offst_file[SIZE]      /* offset table from HWFS camera */
        char  cent_offst_file[SIZE]     /* centroid offset file */
        char  reconst_file[SIZE]        /* reconstructor */
        char  ho_flatmap[SIZE]          /* HODM flatmap */
        char lo_flatmap[SIZE]           /* LODM flatmap */
} HWFP_stat;
```

3.2.2 HWFS_stat

This data structure contains the status of the HWFS component.

HWFS Status Structure

```
typedef struct {
        int rate                /* camera rate (Hz) */
        int bias                /* camera bias */
        float gain              /* camera gain */
        float delay             /* camera delay(?) */
        int clamp               /* camera clamp */
        int function            /* camera function, replaces program or mode (1:16)
        int state               /* disconnected | ON | error */
        int error_status        /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} HWFS_stat
```

3.2.3 ACAM_stat

This data structure contains the status of the ACAM component.

ACAM Status Structure:

```
typedef struct {
        int int_time            /* integration time (milliseconds) */
        int rate                /* camera rate (Hz) */
        float gain              /* camera gain */
        int window              /* full | pupil | acq */
        int height              /* ACAM image height */
```

```
        int width                   /* ACAM image width */
        int start_row               /* ACAM row start */
        int start_col               /* ACAM column start */
        int back_sub                /* ON | OFF */
        char back_file{SIZE}        /* path to acam background file */
        float temp                  /* ACAM temperature */
        int state                   /* disconnected | ON | OFF | error */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} ACAM_stat
```

3.2.4 HODM_stat

This data structure contains the status of the HODM component.

HODM Status Structure:

```
typedef struct {
        int state                   /* standby | normal | test */
        struct temp                 /* array of all temperatures */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
  } HODM_stat
```

3.2.5 MOTOR_stat

This data structure contains the status of the MOTOR component. There will be as many motor structures as there are motors, each to be published separately as a different data item.

MOTOR Status Structure:

```
typedef struct {
        char name                   /* motor name */
        float position              /* motor position (microns or degrees) */
        float velocity              /* requested velocity */
        bool home                   /* at home? */
        int state                   /* disconnected | moving | stopped | error */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
  } MOTOR_stat
```

3.2.6 WLIGHT_stat

This data structure contains the status of the WLIGHT component.

WLIGHT Status Structure:

```
typedef struct {
```

```
        int state                   /* ON | OFF */
        int pwr                     /* 0 : 100 */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} WLIGHT_stat
```

3.2.7 TCS_stat

This data structure is pulled from the TCS by AOCA via the TCS Status_task every 1 second, and then published. This structure must also be pushed by AOCA to the Pharo Netservices wrapper every one second.

TCS Status Structure:

```
typedef struct {
        char  utc[16];              /* time tag; "ddd hh:mm:ss.s" */
        char  ra[12];               /* hh:mm:ss.ss */
        char  dec[12];              /* [+/-]dd:mm:ss.s */
        char  ha[12];               /* [E/W]hh:mm:ss.s */
        char  raw_dec[12];          /* [+/-]dd:mm:ss.s */
        char  raw_ha[12];           /* [E/W]hh:mm:ss.s */
        int   telescope_id;         /* 200 */
        int   conn_status;          /* AO-TCS connection status */
        int   tel_stable;           /* 0 = unstable; 1 = stable */
        float airmass;              /* telescope airmass */
        float cass_ring_angle;      /* telescope cassegrain ring angle (degrees) */
        float focus;                /* telescope focus (mm) */
        float tubelength;           /* telescope tube length (mm) */
        float ra_offset;            /* in arcsec */
        float dec_offset;           /* in arcsec */
        float ra_track_rate;        /* in arcsec/hr */
        float dec_track_rate;       /* in arcsec/hr */
} TCS_stat
```

3.2.8 Auto_stat

This data structure is a list of states for each of the available automations. An automation state is either “idle”, “finished”, or “running”. The state parameter is a float from 0 to 1, such that if it is “running”, the percentage to “finished” is known. This structure is maintained by AOCA and published every one second (?).

Auto Status Structure:

```
typedef struct {
        float af1                   /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af2                   /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af3                   /* 0.0 : 1.0 (idle | running (%) | finished) */
```

```
        float af4                   /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af5                   /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af6                   /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af7                   /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af8                   /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af9                   /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af10                  /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af11                  /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af12                  /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af13                  /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af14                  /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af15                  /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af16                  /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af17                  /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af18                  /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af19                  /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af20                  /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af21                  /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af22                  /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af23                  /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af24                  /* 0.0 : 1.0 (idle | running (%) | finished) */
        float af25                  /* 0.0 : 1.0 (idle | running (%) | finished) */
} Auto_stat
```

3.3 Message Type : External Status (not published)

3.3.1 PHARO_stat

This data structure must be pushed to the Pharo Netservices wrapper every one second, to be used in the Pharo FITS header. The TCS_stat structure (above) is also pushed to Pharo every one second. This structure is not published by AOCA, it will be sent across the Tcp/IP socket connection. Since the Pharo control software will not be modified, the structure and variable names will not change. AO_SCP stands for AO status, configuration, and performance.

PHARO Status Structure:

```
typedef struct {
        float    wfs_delta_x;       /* offset from WFS center to PHARO center (asec) */
        float   wfs_delta_y;
        float   fsm_int_gain;               /* FSM servo gains */
        float   fsm_prop_gain;
        float   dm_int_gain;                /* DM servo gains */
        float   dm_prop_gain;
        boolean fsm_on;                     /* low order on/off */
```

```
        boolean dm_on;                  /* high order on/off */
        int     conn_status;            /* AO controller connection status */
        int     fsm_rate;               /* FSM update rate */
        int     wfs_cam_rate;           /* WFS camera readout rate */
        int     log_data;               /* data types currently being logged */
        char    reconst_name[NAME_LEN]; /* reconstructor name */
        float   ao_quality;             /* AO quality measure (TBD) */
} AO_SCP;
```

3.3.2 INST_stat

A generic instrument status structure for instruments that connect to AOCA outside of the Netservices protocol. A longer list of motors may be required (ie: SWIFT)?

INST Status Structure:

```
typedef struct {
        float ttm_int_gain              /* TTM integral gain */
        float ttm_pro_gain              /* TTM proportional gain */
        float hodm_int_gain             /* HODM integral gain */
        float hodm_pro_gain             /* HODM proportional gain */
        float lodm_int_gain             /* LODM integral gain */
        float lodm_pro_gain             /* LODM proportional gain */
        float subap_min_flux            /* subaperture minimum flux */
        int ho_mode                     /* HODM reconstruct mode TBD */
        float ho_rss_res                /* HODM root sum square for line plots */
        int lo_mode                     /* LODM reconstruct mode TBD */
        float lo_rss_res                /* LODM root sum square for line plots */
        int ttm_rate                    /* TTM rate */
        int ttm_on                      /* ON | OFF */
        int hodm_on                     /* ON | OFF */
        int lodm_on                     /* ON | OFF */
        char  pix_gains_file[SIZE]      /* gain table from HWFS camera */
        char  pix_offst_file[SIZE]      /* offset table from HWFS camera */
        char  cent_offst_file[SIZE]     /* centroid offset file */
        char  reconst_file[SIZE]        /* reconstructor */
        char  ho_flatmap[SIZE]          /* HODM flatmap */
        char lo_flatmap[SIZE]           /* LODM flatmap */
        int hwfp_int_time               /* integration time (milliseconds) */
        int hwfp_rate                   /* camera rate (Hz) */
        float ssm_1a_pos                /* SSM_1A position */
        float ssm_2a_pos                /* SSM_2A position */
        float ssm_1b_pos                /* SSM_1B position */
        float ssm_2b_pos                /* SSM_2B position */
```

} INST_stat;

3.4 Message Type : Info Status

3.4.1 AODD_info

This status structure will be published once at system startup, and will contain hardware info, version numbers, and other system relevant information that we would like to keep track of but only need published once. The linux version information comes from the Cass cage AODD computer.

AODD Info Structure:

```
typedef struct {
        char version{SIZE}              /* p3k software version number */
        char version_date{SIZE}         /* p3k revision date */
        char linux_version{SIZE}        /* cass cage linux version */
        char hwfs_info{SIZE}            /* HWFS firmware info */
        char acam_info{SIZE}            /* ACAM firmware info */
        struct {                        /* firmware info struct for all motors & controllers */
                char aerotech{SIZE}
                char newport{SIZE}
        } motor_info
} AODD_info
```

3.4.2 AOCA_info

This status structure will be published once at system startup, and will contain version numbers and other system relevant information that we would like to keep track of but only need published once. The linux version information comes from the Data room telemetry AOCA computer.

AOCA Info Structure:

```
typedef struct {
        char version{SIZE}              /* p3k software version number */
        char version_date{SIZE}         /* p3k revision date */
        char linux_version{SIZE}        /* telemetry linux version */
        char aoca_info{SIZE}            /* aoca version number */
        struct {                        /* list of all the current automations */
                char af1{SIZE}
        } auto
} AOCA_info
```

3.4.3 GUI_info

This status structure will be published once at system startup, and will contain version numbers and other system relevant information that we would like to keep track of but only need published once. The linux version information comes from the Data room telemetry computer.

AOCA Info Structure:

```
typedef struct {
        char version{SIZE}          /* p3k software version number */
        char version_date{SIZE}     /* p3k revision date */
        char linux_version{SIZE}    /* telemetry linux version */
        char gui{SIZE}              /* gui version number */
} AOCA_info
```

3.5 Message Type : Telemetry

3.5.1 HWFP_data

The HWFP component will publish a structure containing ALL of the HWFP data at the frame rate specified by the HWFS frame rate. The HWFP LOG_INTERVAL parameter may be set to > 1 to slow the data output down, but the output rate will generally be high. If deemed necessary by the RTC engineer (ie: GUI display requirements), the AODP component **may** split the HWFP data into two or more structures. This data structure is an EXAMPLE of one such split option (see 3.2.1 HWFP_stat above). The following telemetry data is required for system operation, but will likely be arranged differently, and/or may be split further into smaller data structures.

HWFP Data Structure:

```
typedef struct {
        unsigned short pixels           /* HWFP pixel data */
        float centroids                 /* HWFP centroids */
        float ttm_a_res                 /* TTM residuals */
        float ttm_b_res
        struct {                        /* TTM positions */
                float ttm_a
                float ttm_b
        } pos
        float hodm_residuals            /* HODM residuals */
        float hodm_positions            /* HODM positions */
        float lodm_residuals            /* LODM residuals */
        float lodm_positions            /* LODM positions */
        float subap_flux                /* HWFP subaperture flux */
} HWFP_data
```

3.5.2 ACAM_data

The Acquisition Camera (ACAM) will publish the CCD image data at the rate specified by the integration time, which can be as fast as 10 Hz. This data will be published in a structure separate from the ACAM status structure.

ACAM Data Structure:

```
typedef struct {
        float pixels[height][width]        /* ACAM pixel data */
} ACAM_data
```

4 ERROR CODES

Following is a list of error codes to be used globally by the P3K system (first pass is cut&paste from ao3 with only a few modifications. This list needs to be better defined). A more detailed list of states and errors will be handled by the individual lower level components.

(* Should we monitor system health with a process?)

```
/* non critical errors */
#define SYNTAX_ERROR                200
#define INVALID_TABLE_ID            201   /* Table id is unknown by system */
#define INVALID_CMD_ID              202   /* command id is unknown by system */
#define INCORRECT_FILE_FORMAT       204   /* Error detected in file format */
#define FILE_NOT_FOUND              205   /* Could not find file */
#define NOT_IMPLEMENTED             206   /* funtion is not implemented yet */
#define INVALID_COMMAND             208
#define OUT_OF_RANGE                218   /* Parameter out of range. */
#define WFP_LOAD_FAILED             219   /* WFP failed to acknowledge table load */
/* critical errors */
#define OUT_OF_MEMORY               250   /* Attempt to allocate memory failed */
#define INVALID_POINTER             251
#define OPEN_PIPE_FAILED            254   /* */
#define INITIALIZATION_ERROR    259    /* controller is not initialized before being used *
```

5 AUTOMATIONS

5.1.1 AF1 (TTM ON)

Description

This automation is run whenever an external device requests to close the TTM high order loop. The system state is checked to make sure it is safe to close the TTM loop.

Command Syntax

TTM ON {MODE = 1 : i}

Procedure

1. turn hwfs camera on? (HWFP sync?)
2. check subap flux?
3. check if motor or telescope moving?

Block? **NO**
Status? **NO** External devices will receive TTM state from published HWFP data

5.1.2 AF2 (TTM OFF)

Description
This automation is run whenever an external device requests to open the TTM high order loop. The system state is checked to make sure it is safe to open the TTM loop.

Command Syntax

TTM OFF

Procedure

1. check lodm or hodm on, depending on mode?
2. <placeholder>
3. <placeholder>

Block? **NO**
Status? **NO** External devices will receive TTM state from published HWFP data

5.1.3 AF3 (LDM ON)

Description
This automation is run whenever an external device requests to close the LODM high order loop. The system state is checked to make sure it is safe to close the LODM loop.

Command Syntax

LDM ON {MODE = 1 : i} * MODE not yet defined

Procedure

1. check for motors or telescope moving?
2. check if TTM is on

3. check if HODM is on, depending on mode?

Block? **NO**
Status? **NO** External devices will receive LODM state from published HWFP data

5.1.4 AF4 (LDM OFF)

Description
This automation is run whenever an external device requests to open the LODM high order loop. The system state is checked to make sure it is safe to open the LODM loop.

Command Syntax

LDM OFF

Procedure
1. check if hodm is on, depending on mode?
2. <placeholder>
3. <placeholder>

Block? **NO**
Status? **NO** External devices will receive LODM state from published HWFP data

5.1.5 AF5 (HDM ON)

Description
This automation is run whenever an external device requests to close the HODM high order loop. The system state is checked to make sure it is safe to close the HODM loop.

Command Syntax

HDM ON {MODE = 1 : i} * MODE not yet defined

Procedure
1. check for motors or telescope moving?
2. check if TTM is on
3. check if LODM is on, depending on mode?

Block? **NO**
Status? **NO** External devices will receive HODM state from published HWFP data

5.1.6 AF6 (HDM OFF)

Description
This automation is run whenever an external device requests to open the HODM high order loop. The system state is checked to make sure it is safe to open the HODM loop.

Command Syntax

HDM OFF

Procedure

1. check if hodm is on, depending on mode?
2. <placeholder>
3. <placeholder>

Block? **NO**
Status? **NO** External devices will receive HODM state from published HWFP data

5.1.7 AF7 (TTM_OFFLOAD)

Description
This automation is run whenever an external device requests to offload TTM residual error to the telescope. This automation may be set to ON by default during system startup.

Command Syntax

TTM_OFFLOAD ON|OFF PROMPT {GAIN = 0.0 : 1.0} {TIME_AVG = ? : ?} {THRESH = ? : ?}

(ao3 time average duration is 3 seconds)
(ao3 threshold is 0.15")

Procedure

1. subscribe to HFWP to get TTM state, and when TTM is ON (check stimulus state?)
    a. determine the average TTM position (or residual error) over TIME_AVG
    b. convert to ra and dec arcseconds
    c. if abs(ra) || abs(dec) > 2.5, error (ao3 threshold)
    d. if abs(ra) || abs(dec) >= THRESH
        i. if PROMPT, prompt the user if they want to offload
            1. if YES, offload = offload * GAIN
            2. send af7_cmd to the telescope
        ii. else
            1. offload = offload * GAIN

2. send af7_cmd to the telescope

e. loop back to a.

2. watch for TTM OFF

Block? **NO**
Status? **YES**

```
typedef struct {
        int running                 /* 0 | 1 */
        float gain                  /* ttm_offload gain */
        int time_avg                /* duration to gather ttm positions (sec */
        float threshold             /* lower threshold for avg ttm pos offloads (arcsec) */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} AF7_stat
```

5.1.8 AF8 (FOC_OFFLOAD)

Description
This automation is run whenever an external device requests to offload residual focus error to the telescope. This automation may be set to ON by default during system startup.

Command Syntax

FOC_OFFLOAD ON|OFF PROMPT {MODE = 1 : i} {GAIN = 0.0 : 1.0} {TIME_AVG = ? : ?} {THRESH = ?:?}

(ao3 time average duration is 7 seconds)
(ao3 threshold is 0.03 mm)

Procedure

1. subscribe to HFWP to get LODM and HODM state, and depending on MODE, when LODM or HODM are ON (check stimulus state?)
    a. depending on MODE determine the average LODM and/or HODM positions (or residual error) over TIME_AVG
    b. extract focus and convert to millimeters of telescope focus
    c. if focus > 0.8, error (ao3 threshold)
    d. if focus >= THRESH
        i. if PROMPT, prompt the user if they want to offload
            1. if YES, offload = offload * GAIN
            2. send af8_cmd to the telescope
        ii. else
            1. offload = offload * GAIN
            2. send af8_cmd to the telescope

e. loop back to a.

2. watch for LODM and/or HODM OFF

Block? **NO**
Status? **YES**

```
typedef struct {
        int running                 /* 0 | 1 */
        int mode                    /* RTC mode */
        float gain                  /* ttm_offload gain */
        int time_avg                /* duration to gather ttm positions (sec) */
        float threshold             /* lower threshold for avg ttm pos offloads (arcsec) */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} AF8_stat
```

5.1.9 AF9 (STIM)

Description
Select a stimulus source

Command Syntax

STIM SKY | WLIGHT | LASER

CANCEL STIM

Procedure

1. if SKY
    a. move wlight fold mirror out
    b. turn wlight off
2. if WLIGHT
    a. move wlight fold mirror in
    b. turn wlight on
3. if LASER
    a. move wlight fold mirror in
    b. turn wlight off

Block? **YES**
Status? **YES**

```
typedef struct {
        int running                 /* 0 | 1 */
```

```
        int stim_pos                /* SKY | WLIGHT | LASER */
        int cancel                  /* 0 | 1 */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} AF9_stat
```

5.1.10 AF10 (BACKGROUND)

Description
Take HWFS background data (do we want ability to see this image from GUI?)

Command Syntax

BACKGROUND {RA = 0.0 : 1000.} {DEC = 0.0 : 1000.} {CAM = BOTH | HWFS | ACAM} {HWFS_TIME_AVG = 1 : 60} {ACAM_TIME_AVG = 1 : 60}

CANCEL BACKGROUND

Procedure

1. subscribe to AF9_stat to get stimulus position
2. subscribe to HFWS to get frame rate
3. subscribe to HWFP_stat to get TTM, LODM, and HODM state
4. if wlight is on and loops are off
    a. turn white light off
    b. determine frames necessary to reach TIME_AVG duration (for both)
    c. subscribe to HWFP_data (and/or ACAM_data) to get subap_flux
    d. average the frames, write this data to disk, and publish
    e. turn wlight on and signal completion
5. if stim = sky
    a. parse ra and dec values
    b. send the PT command to the TCS (wait for move DONE)
    c. determine frames necessary to reach TIME_AVG duration (for both)
    d. subscribe to HWFP_data (and/or ACAM_data) to get subap_flux
    e. average the frames, write this data to disk, and publish
    f. send the –PT command to the TCS (wait for move DONE)
    g. signal completion

Block? **YES**
Status? **YES**

```
typedef struct {
        int running                 /* 0 | 1 */
        float ra                    /* telescope right ascension (arcseconds) */
        float dec                   /* telescope declination (arcseconds) */
```

```
        float hwfs_time_avg          /* duration to gather background HWFS data (sec) */
        float acam_time_avg          /* duration to gather background ACAM data (sec) */
        int camera                   /* BOTH | HWFS | ACAM */
        char hwfs_bg_file            /* path to HWFS_background file */
        float hwfs_background[data]  /* HWFS_background pixel data */
        char acam_bg_file            /* path to ACAM_background file */
        float acam_background[data]  /* ACAM_background pixel data */
        int cancel                   /* 0 | 1 */
        int error_status             /* global enumerated value */
        char status_string{SIZE}     /* detailed freeform comment */
} AF10_stat
```

5.1.11 AF11 (LDM_REG)

Description
Register the LODM actuators to the HWFS subapertures

Command Syntax

LDM_REG COARSE | FINE | BOTH {THRESHOLD = ? : ?} {COUNTER = 1 : 15}

CANCEL LDM_REG

Procedure

1. subscribe to AF9 to get stimulus position
2. turn TTM ON (AF1)
3. if wlight is on
    a. if COARSE || BOTH
        i. publish HWFP LOAD HODM_MAP = last wlight pokemap
        ii. run the coarse registration algorithm until threshold or counter is met (return error)
    b. if FINE || BOTH
        i. publish HWFP LOAD HODM_MAP = last wlight finemap
        ii. run the fine registration algorithm until threshold or counter is met (return error)
    c. signal completion
4. if stim = sky
    a. if COARSE || BOTH
        i. publish HWFP LOAD HODM_MAP = last telescope pokemap
        ii. run the coarse registration algorithm until threshold or counter is met (return error)
    b. if FINE || BOTH
        i. publish HWFP LOAD HODM_MAP = last telescope finemap

ii. run the fine registration algorithm until threshold or counter is met (return error)

c. signal completion

Block? **YES**
Status? **YES**

```
typedef struct {
        int running                 /* 0 | 1 */
        int type                    /* 1 = coarse | 2 = fine | 3 = both */
        float threshold             /* registration completion threshold */
        int counter                 /* number of registration iterations to run */
        int cancel                  /* 0 | 1 */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} AF11_stat
```

5.1.12 AF12 (HDM_REG)

Description
Register the HODM actuators to the HWFS subapertures (does this depend on HWFS_LENS being used?) (will there be various modes to worry about?)

Command Syntax

HDM_REG COARSE | FINE | BOTH {THRESHOLD = ? : ?} {COUNTER = 1 : 15}

CANCEL HDM_REG

Procedure

1. subscribe to AF9 to get stimulus position
2. turn TTM ON (AF1)
3. if wlight is on
    a. if COARSE || BOTH
        i. publish HWFP LOAD HODM_MAP = last wlight pokemap
        ii. run the coarse registration algorithm until threshold or counter is met (return error)
    b. if FINE || BOTH
        i. publish HWFP LOAD HODM_MAP = last wlight finemap
        ii. run the fine registration algorithm until threshold or counter is met (return error)
    c. signal completion
4. if stim = sky
    a. if COARSE || BOTH

i. publish HWFP LOAD HODM_MAP = last telescope pokemap
ii. run the coarse registration algorithm until threshold or counter is met (return error)

b. if FINE || BOTH
i. publish HWFP LOAD HODM_MAP = last telescope finemap
ii. run the fine registration algorithm until threshold or counter is met (return error)

c. signal completion

Block? **YES**
Status? **YES**

```
typedef struct {
        int running                 /* 0 | 1 */
        int type                    /* 1 = coarse | 2 = fine | 3 = both */
        float threshold             /* registration completion threshold */
        int counter                 /* number of registration iterations to run */
        int cancel                  /* 0 | 1 */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} AF12_stat
```

5.1.13 AF13 (LDM_FLAT)

Description
Generate a LODM flatmap

Command Syntax

LDM_FLAT {CLEAR = 0 : 1} {ITERATE = 1 : 10} {TIME_AVG = 1 : 60}

CANCEL LDM_FLAT

Procedure

1. subscribe to AF9 to get stimulus position
2. turn TTM ON (AF1) and LODM ON (AF3)
3. if wlight is on
   a. if CLEAR = 1 load the midrange lodm_flatmap(?), else load current wlight lodm_flatmap
   b. for iter = 1 : ITERATE
      i. average dm_positions over TIME_AVG period
      ii. save the average to disk
      iii. open the LODM loop (primitive?)
      iv. coadd the average positions to the starting flatmap

v. close the LODM loop

c. write the final result to the current wlight flatmap
d. publish (load) this flatmap
e. use this flatmap to generate wlight pokemap and finemap
f. signal completion

4. if stim = sky
    a. if CLEAR = 1 load the midrange lodm_flatmap(?), else load current telescope lodm_flatmap
    b. for iter = 1 : ITERATE
        i. average dm_positions over TIME_AVG period
        ii. save the average to disk
        iii. open the LODM loop (primitive?)
        iv. coadd the average positions to the starting flatmap
        v. close the LODM loop
    c. write the final result to the current telescope flatmap
    d. publish (load) this flatmap
    e. use this flatmap to generate wlight pokemap and finemap
    f. signal completion

Block? **YES**
Status? **YES**

```
typedef struct {
        int running                 /* 0 | 1 */
        int clear                   /* 0 | 1 */
        int iterate                 /* number of flatmap iterations */
        float time_avg              /* duration to average over dm positions (sec) */
        char flat_file              /* path to final flatmap file */
        float flatmap[data]         /* flatmap */
        int cancel                  /* 0 | 1 */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} AF13_stat
```

5.1.14 AF14 (HDM_FLAT)

Description
Generate a HODM flatmap

Command Syntax

```
HDM_FLAT {CLEAR = 0 : 1} {ITERATE = 1 : 10} {TIME_AVG = 1 : 60}
```

CANCEL HDM_FLAT

Procedure

5. subscribe to AF9 to get stimulus position
6. turn TTM ON (AF1) and HODM ON (AF3) (LODM?)
7. if wlight is on
    a. if CLEAR = 1 load the midrange hodm_flatmap(?), else load current wlight hodm_flatmap
    b. for iter = 1 : ITERATE
        i. average dm_positions over TIME_AVG period
        ii. save the average to disk
        iii. open the HODM loop (primitive?)
        iv. coadd the average positions to the starting flatmap
        v. close the HODM loop
    c. write the final result to the current wlight flatmap
    d. publish (load) this flatmap
    e. use this flatmap to generate wlight pokemap and finemap
    f. signal completion
8. if stim = sky
    a. if CLEAR = 1 load the midrange hodm_flatmap(?), else load current telescope hodm_flatmap
    b. for iter = 1 : ITERATE
        i. average dm_positions over TIME_AVG period
        ii. save the average to disk
        iii. open the HODM loop (primitive?)
        iv. coadd the average positions to the starting flatmap
        v. close the HODM loop
    c. write the final result to the current telescope flatmap
    d. publish (load) this flatmap
    e. use this flatmap to generate wlight pokemap and finemap
    f. signal completion

Block? **YES**
Status? **YES**

```
typedef struct {
        int running             /* 0 | 1 */
        int clear               /* 0 | 1 */
        int iterate             /* number of flatmap iterations */
        float time_avg          /* duration to average over dm positions (sec) */
        char flat_file          /* path to final flatmap file */
        float flatmap[data]     /* flatmap */
        int cancel              /* 0 | 1 */
        int error_status        /* global enumerated value */
```

```
        char status_string{SIZE}        /* detailed freeform comment */
} AF14_stat
```

5.1.15 AF15 (HWFS_LENS)

Description
Set the system to use one of four HWFS lenslet arrays

Command Syntax

HWFS_LENS 8 | 16 | 32 | 64

CANCEL HWFS_LENS

Procedure

1. move the HWFS_LENS motor to the appropriate position
2. set the necessary HWFP parameters
3. <placeholder>

Block? **YES**
Status? **YES**

```
typedef struct {
        int running                     /* 0 | 1 */
        int lens                        /* 8 | 16 | 32 | 64 */
        int cancel                      /* 0 | 1 */
        int error_status                /* global enumerated value */
        char status_string{SIZE}        /* detailed freeform comment */
} AF15_stat
```

5.1.16 AF16 (NGS_ACQ)

Description
Center a target on the HWFS camera

Command Syntax

NGS_ACQ

CANCEL NGS_ACQ

Procedure

1. GUI should prompt the user to click on the target in the ACAM display
2. get the X,Y coordinates from the ACAM click
3. subscribe to AF9 to get stimulus position
4. if wlight
    a. publish image_x and image_y to move SSMs to move wlight to the HWFS cam
5. if sky
    a. get cass ring from TCS_stat
    b. calculate ra and dec, publisha TCS PT command to center the target on the HWFS cam
6. TTM ON (?)

Block? **YES**
Status? **YES**

```
typedef struct {
        int running                 /* 0 | 1 */
        int clickX                  /* ACAM X click position */
        int clickY                  /* ACAM Y click position */
        int cancel                  /* 0 | 1 */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} AF16_stat
```

5.1.17 AF17 (NGS_SEEING)

Description
Calculate the seeing of a target in the ACAM

Command Syntax

NGS_SEEING {INT_TIME}

CANCEL NGS_SEEING

Procedure
1. GUI should prompt the user to click on the target in the ACAM display
2. read the ACAM int_time
3. set the ACAM INT_TIME
4. get the ACAM data
5. centroid the target at the click position and measure the fwhm (publish)
6. set the ACAM int_time back
7. this automation needs more work

Block? **YES**
Status? **YES**

```
typedef struct {
        int running                 /* 0 | 1 */
        int clickX                  /* ACAM X click position */
        int clickY                  /* ACAM Y click position */
        int int_time                /* ACAM integration time used for seeing (ms) */
        float seeing                /* measured seeing value (arcseconds) */
        int cancel                  /* 0 | 1 */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} AF17_stat
```

5.1.18 AF18 (SAVE)

Description
Save system settings to a file

Command Syntax

SAVE <FILENAME>

Procedure

1. Subscribe to "everything" needed for a system restore (does this need to be running at system startup?)
2. write this out to a file on disk

Block? **NO**
Status? **YES**

```
typedef struct {
        int running                 /* 0 | 1 */
        int filename                /* path to save filename */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} AF18_stat
```

5.1.19 AF19 (RESTORE)

Description
Restore system settings from a file (default system file will be DEFAULT)

Command Syntax

RESTORE <FILENAME>

CANCEL RESTORE

Procedure

1. read FILENAME
2. publish everything

Block? **YES**
Status? **YES**

```
typedef struct {
        int running                 /* 0 | 1 */
        int filename                /* path to restore filename */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} AF19_stat
```

5.1.20 AF20 (RESET)

Description
Reset a system component or process

Command Syntax

RESET <AOCA,AODD,AODP,AODS,ACAM,MOTOR,HWFS,WLIGHT>

RESET <MOTOR>

<MOTOR> = <ACQ | FQ_RELAY | FQ_STIM | STIM | SSM1_A | SSM1_B | SSM2_A | SSM2_B | WFS_Z | WFS_LENS | WHITE_X | WHITE_Y | WHITE_Z | LASER_Z | TTM_A | TTM_B>

CANCEL RESET

Procedure

1. a MOTOR reset may be a combo of HOME and reset to DEFAULT
2. this needs more thought
3. <placeholder>

Block? **YES**

Status? **YES**

```
typedef struct {
        int running                 /* 0 | 1 */
        int item                    /* list of reset targets */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} AF20_stat
```

5.1.21 AF21 (STAR_SETUP)

Description
Set system parameters based on star magnitude and seeing

Command Syntax

STAR_SETUP <MAG = -1.0 : 15> <SEEING = 0.5 : 2.0>

Procedure

1. publish HWFP params and tables based on magnitude and seeing lookup tables
2. <placeholder>
3. <placeholder>

Block? **NO**
Status? **NO**

5.1.22 AF22 (LOG_RATE)

Description
Convert a given log rate into the log interval required by the HWFP LOG_INTERVAL command parameter.

Command Syntax

LOG_RATE <RATE = 1 : 2000>

Procedure

1. subscribe to HWFS_stat to get camera rate
2. convert the rate into integer interval between frames
3. publish HWFP LOG_INTERVAL = interval

Block? **NO**
Status? **NO**

5.1.23 AF23 (SHUT_ALL)

Description
Shutdown everything

Command Syntax

SHUT_ALL

Procedure

1. move hardware to safe settings
2. when hardware is done, issue a SHUTDOWN ALL
3. <placeholder>

Block? **YES**
Status? **NO**

5.1.24 AF24 (FLEX)

Description
Activate non common path flexure compensation using small SSM moves from a lookup table

Command Syntax

FLEX ON | OFF {RATE 1.0 : 60.0} {GAIN 0.0 : 1.0}

CANCEL FLEX

Procedure

1. from a lookup table apply SSM corrections at RATE x GAIN
2. <placeholder>
3. <placeholder>

Block? **NO**
Status? **YES**

```
typedef struct {
        int running             /* 0 | 1 */
        float rate              /* Rate at which to apply flexure corrections (sec) */
        float gain              /* Gain applied to the correction */
        int cancel              /* 0 | 1 */
```

```
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} AF24_stat
```

5.1.25 AF25 (APRC)

Description
Run the APRC scripts to generate the best science flatmaps and centroid offsets

Command Syntax

APRC <ITER = 1 : 20>

CANCEL APRC

Procedure

1. run the APRC scripts to reduce the non common path error to a minimum, generating new flatmaps and centroid offsets
2. <placeholder>
3. <placeholder>

Block? **YES**
Status? **YES**

```
typedef struct {
        int running                 /* 0 | 1 */
        int iter                    /* iteration number */
        float wfs_error             /* hwfs error estimate after aprc is complete (nm) */
        int pharo_focus             /* focus offset for Pharo (um) */
        int cancel                  /* 0 | 1 */
        int error_status            /* global enumerated value */
        char status_string{SIZE}    /* detailed freeform comment */
} AF25_stat
```

6 FULL P3K COMMAND LIST

COMMAND <Required Parameter> {Optional Parameter}

HWFP TTM {ON|OFF} {MODE = 1 : i}

HWFP LODM {ON|OFF }{MODE = 1 : i}

HWFP HODM {ON|OFF} {MODE = 1 : i}

HWFP LOAD <TABLE = filename>

<TABLE> = <CENT_OFFSETS | PIX_OFFETS | HO_DM_MAP | LO_DM_MAP | PIX_GAINS | RECONST>

HWFP SET <PARAM = float>

<PARAM> = <HODM_PRO_GAIN | HODM_INT_GAIN | LODM_PRO_GAIN | LODM_INT_GAIN | TTM_PRO_GAIN | TTM_INT_GAIN | SUBAP_MIN_FLUX | HODM_MODE | LODM_MODE>

HWFP LOG <DATA = telem> {INTERVAL = 1 : ?}

<telem> = <(PIXELS, CENTROIDS, SUBAP_FLUX, HODM_POS, LODM_POS, HODM_RES, LODM_RES, TTM_POS, TTM_RES, WFP_STAT) | NONE | ALL>

HWFS {ON|OFF} {RATE = 1 : ?} {BIAS = ?} {GAIN = ? : ?} {DELAY = ?} {CLAMP = ?} {FUNCTION = 1:16} {OFFSET = ?} {INFO}

ACAM {ON|OFF} {INT_TIME = 0.0 : ?} {RATE = 1 : ?} {WINDOW = 1 (full 1x1) | 2 (full 2x2) | 3 (full 4x4) | 4 (acq 4x4) | 5 (pupil 2x2} {TEST = 0 | 1} {BACK_SUB = 0 | 1} {BACK_FILE = file_path} {INFO}

HODM {ENABLE | DISABLE | RESET}

MOVE <MOTOR> <POSITION> {VELOCITY = 0.0 : ?}

OFFSET <MOTOR | VIRTUAL_MOTOR> <OFFSET> {VELOCITY}

HOME <MOTOR>

STOP <MOTOR>

<MOTOR> = <ACQ | FQ_RELAY | FQ_STIM | STIM | SSM1_A | SSM1_B | SSM2_A | SSM2_B | WFS_Z | WFS_LENS | WHITE_X | WHITE_Y | WHITE_Z | LASER_Z | TTM_A | TTM_B>

<VIRTUAL_MOTOR> = <IMAGE_X | IMAGE_Y | PUPIL_X | PUPIL_Y>

WLIGHT {ON|OFF} {PWR = 0 : 100}

TCS <TCS COMMAND>

PHARO <PHARO COMMAND>

INST <INST COMMAND>

SHUTDOWN {PROCESS}

{PROCESS} = {ALL | AOCA,AODD,AODP,AODS}

TTM ON {MODE = 1 : i}

TTM OFF

LDM ON {MODE = 1 : i}

LDM OFF

HDM ON {MODE = 1 : i}

HDM OFF

TTM_OFFLOAD ON|OFF PROMPT {GAIN = 0.0 : 1.0} {TIME_AVG = ? : ?}
{THRESH = ? : ?}

FOC_OFFLOAD ON|OFF PROMPT {MODE = 1 : i} {GAIN = 0.0 : 1.0} {TIME_AVG = ? : ?}
{THRESH = ?:?}

STIM SKY | WLIGHT | LASER

BACKGROUND {RA = 0.0 : 1000.} {DEC = 0.0 : 1000.} {TIME_AVG = 1 : 60}

LDM_REG COARSE | FINE | BOTH {THRESHOLD = ? : ?} {COUNTER = 1 : 15}

HDM_REG COARSE | FINE | BOTH {THRESHOLD = ? : ?} {COUNTER = 1 : 15}

LDM_FLAT {CLEAR = 0 : 1} {ITERATE = 1 : 10} {TIME_AVG = 1 : 60}

HDM_FLAT {CLEAR = 0 : 1} {ITERATE = 1 : 10} {TIME_AVG = 1 : 60}

HWFS_LENS 8 | 16 | 32 | 64

NGS_ACQ

NGS_SEEING {INT_TIME}

SAVE <FILENAME>

RESTORE <FILENAME>

RESET <AOCA,AODD,AODP,AODS,ACAM,MOTOR,HWFS,WLIGHT>

RESET <MOTOR>

<MOTOR> = <ACQ | FQ_RELAY | FQ_STIM | STIM | SSM1_A | SSM1_B | SSM2_A | SSM2_B | WFS_Z | WFS_LENS | WHITE_X | WHITE_Y | WHITE_Z | LASER_Z | TTM_A | TTM_B>

STAR_SETUP <MAG = -1.0 : 15> <SEEING = 0.5 : 2.0>

LOG_RATE <RATE>

SHUT_ALL

FLEX ON | OFF {RATE 1.0 : 60.0} {GAIN 0.0 : 1.0}

APRC <ITER = 1 : 20>

7 P3K MESSAGING TYPES TABLE

ID	Message Type	Parameters	Publishers	Subscribers
1	HWFP cmd	TTM, LODM, HODM, tables, servo, RTC params	AOCA	HWFP, GUI
2	HWFP status	RTC params	HWFP	GUI, AOCA
3	HWFP telem	residuals, centroids, pixels, actuators,…	HWFP	GUI, AOCA
4	HWFS cmd	framerate, bias, gain,	AOCA	AODD, GUI, HWFP
5	HWFS status	Framerate bias, gain	AODD	GUI, AOCA
6	ACAM cmd	on/off, integ, frame rate, gain	AOCA	AODD, GUI
7	ACAM status	Integ, framerate, gain	AODD	GUI, AOCA
8	ACAM telem	Acquisition image data	AODD	GUI, AOCA
9	HODM cmd	startup sequence, driver modes, load maps	AOCA	AODD, GUI
10	HODM status	State, errors	AODD	GUI, AOCA
11	MOTOR cmd	home, move, offset…	AOCA	AODD, GUI
12	MOTOR status	moving, position, errors…	AODD	GUI, AOCA
13	Wlight cmd	on/off, pwr	AOCA	AODD, GUI
14	Wlight status	State, errors	AODD	GUI, AOCA
15	TCS cmd	PT, foc_offset	AOCA (socket)	TCS (socket)
16	TCS status	Telescope params	TCS (socket), AOCA	AOCA (socket)
17	Auto status	Idle, running (%), finished	AOCA	GUI
18	PHARO cmd	Pharo cmds in and out	Pharo (Netsvc), AOCA (Netsvc)	AOCA (Netsvc), Pharo (NetSvc)
19	PHARO status	AO_SCP data	AOCA (NetSvc)	Pharo (NetSvc)
20	INST cmd	Generic science inst cmds	AOCA	Inst
21	INST status	Generic AO data	AOCA	Inst
22	SHUTDOWN cmd	Shutdown system or components	AOCA	HWFP, AODD, GUI
23	AFn cmd	automation start, parameters, …	AOCA (tasks)	AOCA
24	AFn status	Status, error codes	AFn	GUI
25	AF10 background	HWFS background sky data	AF10 (AOCA)	HWFP
26	AF13 ldm flatmap	LODM flatmap	AF13 (AOCA)	HWFP
27	AF14 hdm flatmap	HODM flatmap	AF14 (AOCA)	HWFP
28	AODD info	Firmware, version numbers	AODD	none
29	AOCA info	version numbers, automation list	AOCA	none
30	GUI info	Version numbers	GUI	none