COMMAND/STATUS SET FOR ELECTRA

wht-naomi-58

Draft 2
24 July 1996
RMM

1. COMMAND COMMUNICATIONS AND THE SCRIPTING PROCESS

1.1 UI and function processes

Commands go from (UI) user interface processes (GUI and CLI and external interface) to function processes (instrument control, visualisation, etc.) and status packets flow back. Status packets can be generated at any time and UI processes should be capable of handling them at any time.

1.2 Scripting layer

The scripting layer, in order to provide sequencing control and to prevent deadlock/lockout, must be a single process which mediates the command and status packets. Although it cannot be proven that ALL command and status packets MUST flow via the single scripting process, it is suggested that this convention be adopted.

1.3 Ultimate Origin/Destination Fields

Because the General Purpose (GP) messaging protocol destination and origin fields will refer to the routing to/from the scripting process, the command body will therefore contain ultimate destination and origin fields.

1.4 Default command/status passthrough

The default action of the scripting process upon receipt of a command or status packet with an ultimate destination field other than the scripting process itself is to pass it on.

1.5 Scripting interpretation

The scripting layer may be optionally instructed to intercept a particular command template and to execute a script using the command parameters as arguments. The script would typically cause the transmission of several different command packets to be sent to function process(es). Where the originating process sets the final destination as the scripting process itself then an intercepting script must be available. If it is not then an error is logged and an error status return is sent to the originating process.

1.6 Sequential and parallel operations

The set of commands generated by the script may involve sequential and parallel sections with programmable rendezvous and actions in the event of timeout. Command packetsend operations are the only actions which can be parallelised (not parts of the scripting language). This is so that a full and consistent context is available for all script code which is executed. Parallel threads would not necessarily have information available resulting from all status packets (without a forwarding system and interlocks).

1.7 Completion Acknowledgements

The implementation of script sequencing will require acknowledgement of completion for all commands sent from the scripting process. All such commands will therefore carry a unique sequencing ID which will be echoed in the acknowledgement packet and allow an acknowledgement to be associated with the command. It is important to note that although the command extraction functions in the function process will be able to generate syntax status returns, they will not be able to generate semantic status returns and completion status returns. The script sequencer will therefore block normally (i.e. unless explicitly instructed to parallelise an operation) until it has receieved a syntax ERROR (as opposed to OK) status, a semantic error, or a completion status. A consequence of this is that function processes must keep track of command sequence ID explicitly: this is so that the corret sequence sequence ID will be placed in semantic and completion status returns. In the proposed programming interface this will actually be accomplished by keeping track of extracter handles (which are associated with sequence Ids and other extraction information). Parallel operations will simply wait for a syntax status before proceeding.

1.8 Timeouts

The scripting process will provide timeout monitoring even on passed-through commands. A timeout field is therefore provided which the originating process may fix. If the command times out (i.e. no sacknowledgement is received) then a timeout error packet is sent to the originating process. The timeout is also logged by the scripting process (as is a late acknowledgement).

1.9 Scripting process command and status ports

Separate ports are required for command and status packets. Processing of command packets is sequential while the process must remain live to status packets.

1.10 Grammar enquiry

Where appropriate library functions and programmingconventions are used for command interpretation
then it will be possible for a UI process to enquire the command set of a function process. This will permit a
standard self-configuring CLI programme, GUI or widget to provide an interface for any function process.
Special grammar probe and delve packets will allow the allowed commands and parameters to be explored.
Where a special non-primitive parameter type is retured by a probe then a DELVE can be used to
decompose it. Anything which a particular UI process cannot understand can therefore be recursively
decomposed (ultimately down to a set of primitive parameter types). The compound types are important for
status returns where a UI process can make a more specific interpretation of a group of related status
parameters. An example of this would be the group of status parameters associated with a particular class of
physical device (e.g., a wheel) which could be associated with a corresponding graphical widget or control.

2. PROGRAMMING INTERFACE

```
typedef enum {
      COMMAND;
               SYNTAX_PROBE;
               SYNTAX_DELVE;
      SYNTAX_PING;
} PragmaticVerb

/* initialisable type */

typedef struct {
      const int en;                 /* int cast from each enum value */
      const char * const keyword;   /* text keyword */
      const char * const help;          /* text keyword */
      void *expand;                 /* expansion field (so that initialisers can
be used) */
} EnumCtrl;

/* set up / shut down functions */

ExtracterHandle CmdNewExtracter  (
      char *buf;              /* protocol packet buffer */
      int len;                      /* buffer length */
);    /* returns extracter handle */


CmdStatus CmdFreeExtracter   (
      ExtracterHandle handle;       /* extracter handle */
);    /* returns status */

/* pragmatic verb get function */
CmdStatus CmdGetPragmaticVerb (
      ExtracterHandle handle;       /* parser handle */
```

```
      PragmaticVerb *enPtr;        /* result pragmatic verb enumerate */
);    /* returns status */

/* if this function returns a probe or delve then no actions should be carried
out and no */
/* variables set. The extracter system checks thepragmatic verb when the
extracter is    */
/* initialised and will not set primitive types */

/* primitive parameter get functions */

CmdStatus CmdGetEnum (
      ExtracterHandle handle;      /* parser handle */
      EnumCtrl enCtl[];      /* array of enums, ids and help fields */
      CmdFlag flag;                /* indicates when LAST parameter of cmd is
extracted */
      int *enPtr;            /* result enumerate (cast to int) pointer */
);    /* returns status */

CmdStatus CmdGetInt (
      ExtracterHandle handle;      /* extracter handle */
               char *id ;               /* name of paramter (DTM: tag) */
               char *hlp;               /* descriptive help string for
parameter */
      int min;                     /* minimum value */
      int max;                     /* maximum value */
      CmdFlag flag;                /* indicates when LAST parameter of cmd is
extracted */
      int *intPtr;                 /* result int pointer */
);    /* returns status */

CmdStatus CmdGetFloat (
      ExtractHandle handle;        /* Extracter handle */
      char *id    ;                /* name of paramter (DTM: tag) */
               char *hlp;               /* descriptive help string for
parameter */
      float min;             /* minimum value */
      float max;             /* maximum value */
      CmdFlag flag;                /* indicates when LAST parameter of cmd is
extracted */
      float *floatPtr;             /* result float pointer */
);    /* returns status */

CmdStatus CmdGetString (
      ExtracterHandle handle;      /* extracter handle */
      char *id    ;                /* name of paramter (DTM: tag) */
               char *hlp;               /* descriptive help string for
parameter */
      int maxlen;            /* limit on string size */
      CmdFlag flag;                /* indicates when LAST parameter of cmd
extracted */
      char * strPtr;               /* result string pointer */
      int * actuallen;             /* actual string size */
);    /* returns status */

The proposed implementation for all the primitives and the general fields
described in the next section are
```

tagged parameters within DTM headers. A proposed extension is to transmit tupples of preset types using
the data section. The header would then carry type, rank, and dimensionality information.

3. GENERAL COMMAND FIELDS

3.1 The command infrastructure fields

Ultimate-Origin
Ultimate-Destination
Pragmatic-Verb
   Can be: COMMAND STATUS GRAMMAR-PROBE GRAMMAR-DELVE COMMAND-PING
Sequence-ID
   set by command; echoed by status
Timeout
   For commands, probes, delves and pings

3.2 The command fields

Command-Verb
Command-Parameters

4. GENERIC COMMAND SET

Command verbs:

StatusRegister
    register for status packets to be sent to the ultimate origin.
    allowed parameters: ALL, or specific category which may be obtained by grammar probes.

StatusPing
    force a status to be sent to the ultimate origin for ALL or a particular category of status.
    The status returns may be UNDEFINED if they are not meaningful in the current context.

5. SPECIFIC COMMAND AND STATUS SET

The specific commands for the function processes areillustrated as

<dest> <verb> <parameters...>

<dest> indicates the ultimate destination function process and can be
RTCS Real time control system
Opt Optimisation
Mech Mechanism
Vis Visualisation/Reduction data SOURCE
Script Scripting layer
Proc Process creation and control
<Reduce> Instantiation of reduction data SINK process of given type
<Plot> Instantiation of plot data SINK process of given type

<Plot> and <Reduce> can be different programmes each of which may be multiply instantiated.

Command sources must therefore replace <Plot> and <Reduce> with the Ids obtained from Process
creation system.

Some commands below allow parameter specification in physical (e.g. volts) or encoder units.
Other unit translations should be handled by the UI/Scripting processes.

Real-Time Group

RTCS GlobalTiltLoop Open|Close
RTCS SegmentTiltLoop Open|Close ALL|Rowcol|ID|Rowcol-tuple|ID-tuple
RTCS Reconstructor Open|Close
RTCS SetWFSAlgorithm <algorithm> <algorithm-specific-parameters...>
RTCS SetReconsAlgorithm <algorithm> <algorithm-specific-parameters...>
RTCS LoadZonalMatrix <filename>
RTCS SaveZonalMatrix <filename>
RTCS LoadWFS-ModeMatrix <filename>
RTCS SaveWFS-ModeMatrix <filename>
RTCS LoadMode-DMMatrix <filename>
RTCS SaveMode-DMMatrix <filename>
RTCS OpenLoopMode <Mode-specifier; int> +|-|= <amplitude; float>
RTCS ClosedLoopMode <Mode-specifier; int> +|-|= <amplitude; float>
RTCS LoadWFSOffsets <filename>
RTCS SaveWFSOffsets <filename>
RTCS SetWFSOffsetAs <subap-specifier; int> <x|y> +|-|= <offset in arcsec; float>
RTCS SetWFSOffsetPix <subap-specifier; int> <x|y> +|-|= <offset in pixels; int>
RTCS LoadDMPattern <filename>
RTCS SaveDMPattern <filename>
RTCS SetDMSegmentAs FSM|<segment-specifier; int> x|y|z
   On|Off|+|-|= <offset in arcsec (z-microns); float>
RTCS SetDMSegmentDac FSM|<segment-specifier; int> x|y|z
   On|Off|+|-|= <offset in DAC units; int>
RTCS EnableAutoOpt On|Off

Script AllLoops Open|Close
Script MeasureGains All|FSM||<segment specifier;int > <filename>

Mechanism Group

Mech AlignSource Laser|ArtificialStar
Mech AlignBS
Mech Acquisition In|Out
Mech WFSStageAs x|y|z +|-|= <offset in arcsec; float>
Mech WFSStageDAC x|y|z +|-|= <encoder offset; int>
Mech WFSND <0-N; int>
Mech OpticalFilter <Integer:0-N>|FilterToken

Script ClearForAstronomy
Script SetForAcquisition
Script SetForAlignment

<Plot> Group - generic

<Plot> Dchange
      dimensions/type about to change
<Plot> Thresh <Min; float> <Max; float>

```
      set absolute data threshold values
<Plot> Title <title; string>
<Plot> Plimits <Min; float>  <Max; float>
               display limit values (must be within Thresh limits)
<Plot>  Xlabel <x label; string>
<Plot>  Ylabel <y label; string>
<Plot>  Polygon <name; string> <npoints; int> <x1..; float> <y1..; float> <>
<Plot>  Remploy <polygon name; enum>
<Plot>  Ping
      may be subsumed into pragmat ping
<Plot> Dformat <data format; enum>
<Plot> Grab
      dump to postscript
<Plot> SetCross <number; int> <size; float> <<x,y coords; 2xnumber int DATA
TUPLE>
<Plot> Dcross
      remove

Vis group
 TBD
Opt group
 TBD
Script group
 TBD
Proc group
TBD
<Reduce> group
TBD


TO READERS: please comment on any aspect of architecture/ programming interface.
Please add any comments to filled-in OR TBD command groups above.
Please add status types and values for processes.
Please consider where compound status types might be appropriate.
```