MASSACHUSETTS INSTITUTE OF TECHNOLOGY

HAYSTACK OBSERVATORY

WESTFORD, MASSACHUSETTS 01886

Telephone: *781-981-5400*
Fax: *781-981-0590*

14 November 2013

TO: Distribution
FROM: Alan Whitney, Chester Ruszczyk
SUBJECT: Mark 6 Command Set (Release 1.0)

Introduction

This memo documents the control program and command set for the Mark 6 VLBI data system when used in normal field recording applications. Different software must be used to read recorded data into a correlator.

1. *cplane* program

The commands detailed in this memo are implemented by a program called *cplane* to control the DIM functionality of the Mark 6 VLBI data-recording system. Playback functionality is not handled by this application.

2. Notes on Mark 6 Command set

Note the following with respect to the command set:

1. Processing of all of the commands/queries expect the VSI-S communications protocol and command/response syntax. *cplane* will also support *remote procedure calls* (RPCs).
2. Commands/queries are case *insensitive*.
3. Versions of program *cplane* with a revision date earlier than the date on this memo may not implement all commands indicated in this memo or, in some cases, may implement them in a different way.

3. VSI-S Command, Query and Response Syntax

The following explanation of the VSI-S syntax may be useful in understanding the structure of commands, queries and their respective responses. This explanation has been lifted directly from the VSI-S specification.

3.1 *Command Syntax*

Commands cause the system to take some action and are of the form

<keyword> = <field 1> : <field 2> : …. ;

where <keyword> is a VSI-S command keyword. The number of fields may either be fixed or indefinite; fields are separated by colons and terminated with a semi-colon. A field may be of type decimal integer, decimal real, integer hex, character, literal ASCII or a VSI-format time code. White space between tokens in the command line is ignored, however most character fields disallow embedded white space. For Field System compatibility, field length is limited to 32 characters except for the ‘scan label’ (see Section 6), which is limited to 64 characters.

3.2 *Command-Response Syntax*

Each command elicits a response of the form

!<keyword> = < VSI return code > :
<Mk6-specific return code> [: <Mk6-specific ASCII return info> : ..] ;

where

<keyword> is the command keyword

<VSI return code> is an ASCII integer as follows:

0 - action successfully completed
1 - action initiated or enabled, but not completed
2 - command not implemented or not relevant to this DTS
3 - syntax error
4 - error encountered during attempt to execute
5 - currently too busy to service request; try again later
6 - inconsistent or conflicting request
7 - no such keyword
8 - parameter error

<Mark6-specific return code> - Mark 6-specific ASCII return code

<Mark6-specific ASCII return info> - optional additional ASCII info relating to Mark 6 return code

3.3 *Query and Query-Response Syntax*

Queries return information about the system and are of the form

<keyword> ? <field 1> : <field 2> : …. ;

with a response of the form

!<keyword> ? <VSI return code> :
[<Mk6-specific return code> [:<Mk6-specific ASCII return info> :] ..] ;

where

<VSI return code> is an ASCII integer as follows:

0 - query successfully completed
1 - action initiated or enabled, but not completed
2 - query not implemented or not relevant to this DTS
3 - syntax error
4 - error encountered during attempt to execute query
5 - currently too busy to service request; try again later
6 - inconsistent or conflicting request
7 - no such keyword
8 - parameter error
9 - indeterminate state

Note: *A ‘blank’ in a returned query field indicates the value of the parameter is unknown.*
A ‘?’ in a returned query field indicates that not only is the parameter unknown, but that some sort of error condition likely exists.

4. Modules, Slots, Groups and Group References

Modules and Slot#'s

A Mark 6 *module* consists of 8 disks in a specially designed portable carrier. Each module is physically identified by a permanent unique identifier (called Module Serial Number' or 'MSN') printed on a barcode label on the module front panel and can be plugged into a physical *slot#* in the Mark 6 system (typically slot#'s are 1 to 4 for a system with a single expansion chassis.

The slots in a Mark 6 system are numbered *1 to n*, where *n* is the number of available slots; the slot#'s are prominently labeled on the Mark 6 chassis (see Figure 1). Each Mark 6 *system chassis* and *expansion chassis* accommodates two slots; the most usual Mark 6 system will have a system chassis with one expansion chassis for a total of four slots, though some systems may be able to support more expansion chassis.

Unlike the Mark 5 systems, each Mark 6 module is connected to the Mark 6 controller by a pair of high-speed serial-data cables, each supporting four disks. Each cable-pair is labeled with the slot# that they support, but the connection of the two cables to the module in that slot# is arbitrary; the Mark 6 system determines which cable is connected to which connector. Both data-cables must be connected to the module in the associated slot# before the keyswitch is turned to 'on', which applies power to the module; modules will be self-discovered by the Mark 6 as they come alive. Correspondingly, a module should be physically removed only after the associated group is logically 'unmounted', the keyswitch is turned to 'off', the associated red LED turns off after ~5 seconds (to allow the disks to spindown before handling), and the data cables are disconnected.

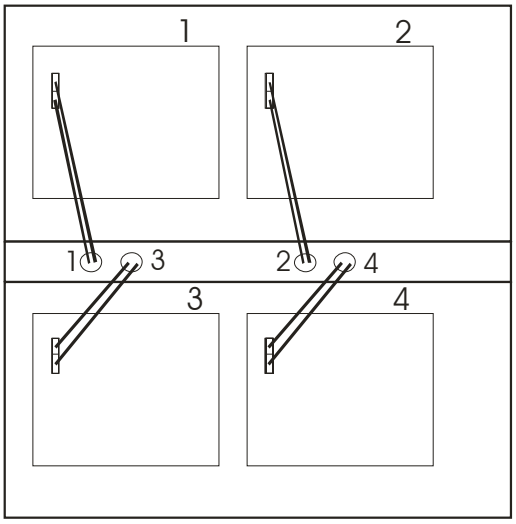

Figure 1: Schematic representation of a Mark 6 system with system chassis (top), cable-management tray (center), and expansion chassis (bottom). Four disk modules are mounted with associated data cables attached to module front panels.

Groups

A *group* consists of one of more modules designated as a set of modules that act as a single unit for recording and playback. The maximum number of modules in a group is equal to the number of slots in the Mark 6 systembeing used, typically 2 or 4 modules, but some systems may accommodate as many as 6 modules.

A group can only be created from *initialized* modules, each of which has undergone an initialization procedure that erases all data on a module and returns all its disks to a preset condition. In order to create a new group, all the initialized constituent modules must be simultaneously mounted; the group is formed by designating the slot#'s of the modules to be included in the group (subsequent mounting of the disks in a group may be to other slot#'s), or by asking the Mark 6 to create a group from *n* available qualified modules.

Once a set of modules is designated as a group, that set of modules acts as a single unit and is inseparable and unchangeable until the constituent modules are individually re-initialized.

Group References

Multiple groups of modules may be simultaneously mounted on a Mark 6 system up to the limit of the number of available slots. In order to distinguish simultaneously mounted groups, each group is assigned a *group reference (*or *group_ref*) consisting of an unordered set of digits corresponding to the slot#'s of the constituent modules (e.g. '3' '12' or '314' or '1234' for groups of 1, 2, 3 and 4 modules, respectively). Note that the group reference will change accordingly if the group of modules is subsequently remounted to a different set of slots in the same or another Mark 6 system.

5. Scan names, Scan Labels and Linux filenames

cplane defines a 'scan' as a continuously recorded set of data. Each scan is identified by a scan name, experiment name and station code, which are normally derived from the information in the associated VEX file used in the scheduling of the experiment (see http://www.vlbi.org/vex)

The filename format for a file containing data from a single scan is typically

<exp name>_<station code>_<scan name>.<file type>

where

<exp name> - experiment name; max 16 chars (consistent with current limit)

<station code> - standard 2-character ASCII station code, or decimal numeric value corresponding to 16-bit numeric station code (see VDIF specification at http://www.vlbi.org for clarification).

<scan name> - assigned scan name (derived from VEX file or other source); max 16 chars

<file type> - identifies high-level data format within file (for example: 'vdif' and 'm5b' for VDIF and Mark5B data formats, respectively)

Maximum scan-label length, including embedded underscores and possible scan-name suffix character, is 50 characters. <exp name>, <stn code> and <scan name> may contain only standard alpha-numeric characters, except '+', '-' and '.' characters may also be used in <scan name>. All fields are case sensitive. No white space is allowed in any of these subfields. Lower-case characters in all subfields are preferred. An example Mark 6 filename is:

grf103_ef_scan123.vdif

In actuality, during the recording of a scan, the Mark 6 writes a single standard Linux file to each disk in the active group and assigns system-specific unique filenames to each of these files that are different than the prescribed Mark 6 filename. The Mark 6 keeps track of the correspondence between these filenames and the Mark 6 filename through separate auxiliary files maintained on the data disks; in this way, the group of individual files that comprise a Mark 6 file may be managed and referred to by the Mark 6 filename.

6. Command set overview

The Mark 6 set of commands and queries is designed to be operationally straightforward, as well as to provide some diagnostic information needed to solve problems when they arise. There are only four top-level commands necessary to control the Mark 6 system; seven queries are available to gather various types of information, including diagnostic information to help isolate some types of problems, particularly the identification of 'slow' disks.

Commands

input_stream: Each 'input_stream' command specifies the physical 10GigE port to which the stream is connected, the data format (VDIF, Mark5B, etc) of the packet payloads, an ASCII label that is used to identify the datastream, and the IP/MAC source address for packet filtering (if necessary). Up to four data streams of up to 4Gbps each may be defined.

mod_init: 'Initializes' a disk module by erasing all data and readies it to be assigned to a group.

group: The 'group' command is used to manage groups of modules, as follows:

- 'auto' - create a new group of *n* modules from the pool of qualified available modules
- 'new' - create a new group from a set of specified modules (by slot#)
- 'mount' – mount a group of modules
- 'open' – 'open' a specified group to enable it for recording
- 'close' – 'close' a specified group to recording, but may be later re-opened
- 'protect' –prevent further recording to a specified group until 'unprotected'
- 'unprotect' – designate a specified group as available for recording, but does not 'open' for recording
- 'erase' – erase all data from a specified group
- 'unmount' – prepare a specified group for physical removal of associated modules

record: Turn scan recording on/off or specify a start time and duration. Each scan is labeled with a unique scan name, the experiment name and station name.

Queries

A variety of queries are available to ascertain the status and health of the system, and to help provide some limited diagnostic information as necessary:

disk_info: Provides disk-by-disk module information about serial#, model, manufacturer, temperature, storage capacity, and amount of storage currently used; very useful for locating slow or dead disks.

mstat: Provides module properties and status by slot#, group, or all mounted modules. Properties include slot#, associated group (if any), extended-MSN, #disks discovered, #disks registered at initialization, total module storage and amount used. Status of each specified module is shown as 'recording', 'open', 'closed', 'initialized' 'unknown', etc., as well as 'unprotected' or 'protected' as appropriate. Also returned are the MSN(s) for any missing (i.e. unmounted) module(s) of an incomplete group, allowing missing modules to be quickly identified.

rtime: Returns amount of recording time at a specified data rate available on the currently open group. Before issuing a 'record' command, the Field System can check if space is available and take appropriate action as needed to identify a new available group.

scan_check: Reads a small portion of the last-recorded scan (default) or specified scan and reports such information as scan#, scan label, #data streams and corresponding stream labels, data format, duration of recording, total storage used, average data rate, and amount of missing data.

scan_info: Returns general information about the last-recorded scan (default) or specified scan without actually reading any of the data of the scan. Reports such things as Mark 6 system serial#, scan#, scan label, current status (pending, recording, flushing buffers, complete), start time, duration, and #data streams.

sys_info: Returns general system information such as Mark 6 system serial#, OS type and revision, *cplane* version, command set revision, available RAM, max number data disks supported, #slots in system, #input ports and port reference names.

msg: Retrieves (if available) the ASCII message associated with a specified *cplane* error return code

7. *cplane* Command/Query Summary (by Category)

7.1 System Setup and operation

7.2 Status and information queries

7.3 Processing-status indicators requested by NRAO

8. *cplane* Command Set Details

This section contains a complete description of all *cplane* commands/queries in alphabetical order.

delete – Delete recorded scan(s) from group

[command list]

Command syntax: delete = <scan>;
Command response: !delete = <return code> : <*cplane* return code> ;

Query syntax: delete?;
Query response: !delete? <return code> : <*cplane* return code> : <last scan deleted> ;

Purpose: Delete a scan from an active module

Query parameters:

Parameter	Type	Allowed values	Default	Comments
<scan>	char	scan name		Delete specified scan from mounted group (see Notes 1 & 2)

Query response parameters:

Parameter	Type	Values	Comments
<scan>	char	-	Last scan removed.

Notes:

1. Unlike previous version of Mark5 that recorded data sequentially, the Mark6 allows the deletion of any scan; the released space may be used for subsequent recording. Note, however, that depending upon the physical pattern of deleted data on member disks, maximum subsequent re-recording data rate may be somewhat diminished.
2. Group must be ‘open’ and ‘unprotected’.
3. To erase all scans in a group, use the ‘group=erase’ command.
4. An extension in planned to allow deletion of multiple scans through a ‘wild-card’ specification of <scan>.

disk_info – Get disk information about individual disks in a module (query only)

[command list]

Query syntax: disk_info? <info_type> : <slot#> ;

Query response: !disk_size? <return code> : <*cplane* return code> : <info type> : <slot#> : <eMSN> : <#discovered disks> : <#registered disks> : <#disks> : <disk1 info> : …. : <disk*n* info> ;

Purpose: Get information about individual disks in a specified module

Query parameters:

Parameter	Type	Allowed values	Default	Comments
<info_type>	char	serial | model | vendor | usage | size | temp	usage	'serial' – disk serial#'s 'model' – disk model #'s 'vendor' – vendor/manufacturer 'usage' – disk usage in GB 'size' – disk sizes in GB 'temp' – disk temperature in degC
<slot#>	int	available slot#'s	-	slot# - return info on module in specified slot#

Query response parameters:

Parameter	Type	Values	Comments
<info_type>	char	-	Information type being returned
<slot#>	int	1-9	Module slot#
<eMSN>	char	-	Extended-MSN of first (and perhaps only) module in group
<#discovered disks>	int	1-9	#disks discovered in mounted module
<#registered disks>	int	1-9	#disks registered at initialization
The following is repeated for each registered disk in module; order of disk information is always identical to order of serial number query			
<disk*x* info>	-	-	Information of type requested by <info_type>: 'serial' – disk serial# at initialization; preceded by '-' if disk is missing/dead (see Notes 1 and 2) 'model' – disk model # at initialization (see Note 3) 'vendor – disk vendor (WD, SS, HT, SG, etc) 'usage' – current disk usage in GB (see Notes 3 and 4); null if unavailable 'size' – disk size in GB at initialization (see Note 3) 'temp' – current disk temperature in degC; null if unavailable

Notes:

1. At module initialization, the serial#'s, model#'s and disk sizes of *all* disks in a module are written to *each* disk in the module; this information is then accessible even if one or more of the original disks fail (as long as at least one disk in the module is still OK).
2. For any undiscovered disks (missing/dead), the corresponding reported serial number(s) is preceded by '-'. This allows easy identification of missing/dead disks by serial number within a module.
3. All disk information is listed in the same order and position as in 'disks? serial' query so that physical disks can associated with the query-return info.
4. The Mark 6 system writes to disks in a round-robin fashion, but if a disk is not ready to write when its turn comes around, Mark 6 skips over that disk and writes to the next one instead. If this happens repeatedly to the same disk, it will accumulate noticeably less data than its properly operating neighbors. Therefore, if an examination of the amount of data recorded on individual disks across a module reveals significant disparity, one or more slow and/or improperly functioning disks should be suspected. A 'disks? serial;' query may be used to identify corresponding physical disks.

DTS_id – Get brief system information (query only)

[command list]

Query syntax: DTS_id? ;

Query response: !DTS_id ? <return code> : <system type> : <software version number> : <serial number> : <command set revision>;

Purpose: System info query backwards compatible with Mark 5 system

Monitor-only parameters:

Parameter	Type	Values	Comments
<system type>	char	Mark6	
<software version number>	char		Version number for current version of *cplane*
<serial number>	ASCII		System serial number; generally is in the form 'mark6-xxxx' where xxxx is the system serial number (see Note 1)
<command set revision>	char		Mark 6 command set revision level corresponding to this software release (e.g., '3.0.4')

Notes:

1. The 'DTS_id?' query provides legacy support for the same command on the Mark 5 systems, providing a common query to allow the user to distinguish between a Mark 5 and Mark 6 system.
2. The Serial number is of the format Mark6-XXXX, but if an expansion chassis is connected and the serial number of the expansion chassis is set the output of the serial number will be Mark6-XXXX-YYYY. Where XXXX is the main Mark6 serial number and YYY is the expansion chassis serial number if present.

DTS_id

execute – Upload an XML file to Mark 6 (NYI)

[command list]

Command syntax: execute = <action>:[<filename>];
Command response: !execute = <return code> : <*cplane* return code> ;

Query syntax: execute?;
Query response: !execute? <return code> : <*cplane* return code> : <action>:[<file>] ;

Purpose: Upload an XML schedule file Mark 6.

Query parameters:

Parameter	Type	Allowed values	Default	Comments
<action>	char	upload | append | finish	-	'upload' – name to be assigned to uploaded xml schedule file and includes first block of data 'append' – append data to schedule file, 'finish' – last block of data; write file to disk
<filename>	char		-	Name of file being uploaded and written to OS disk

Query response parameters:

Parameter	Type	Values	Comments
<action>	char	uploading | finished | error | -	'uploading' – presently uploading an xml schedule file, 'finished' – successfully uploaded an xml schedule file, 'error' – file did not upload successfully '-' - inactive
<filename>	char		Name of file being uploaded and written to OS disk

Notes:

1. The 'execute' command transmits, piece by piece, an XML ASCII file (which may, for example, be an xml-based schedule or configuration file created from VEX) to a Mark 6 through a VSI-S connection. This file may then be subsequently executed using the 'mk6cc' to initiate. Additional <actions> may be added in the future.
2. The 'upload' option specifies a filename to be created and initiates the upload process, which transfers the file piece by piece through individual VSI-S transfers.
3. An 'append' action appends to the file; there is no restriction on the number of 'append' actions that can be initiated after an 'upload' is initiated.
4. A 'finish' must be issued to close the process, indicating the last block of data and causing the specified file to be written.
5. Errors will be returned if :
    a. 'uploads' is initiated, with the state not 'null' or 'finished';
    b. 'append' is initiated with the previous state not being upload on the receipt of the first command and all subsequent states being 'upload';
    c. 'finish' issued with the prior state being 'upload' or 'append'

execute

group – Manage module groups

[command list]

Command syntax: group = <action> : <param> ;
Command response: !group = <return code> : <*cplane* return code> : <group_ref >;

Query syntax: group? ;
Query response: !group ? <return code> : <*cplane* return code>: <group1> : <group2 > [: …] ;

Purpose: Manage module groups.

Command parameters:

Parameter	Type	Allowed values	Default	Comments
<action>	char	auto | new | mount | open | close | protect | unprotect | erase | unmount	-	auto – create a group from available eligible modules new – create a group from specified module(s) mount – mount a group of modules open – open a group for recording close – close the open group, if any protect – write-protect a group; automatically 'closes' group unprotect – write-enable a group erase – erase all data (must be preceded immediately by and unconditionally by 'unprotect' command) unmount – logically unmount a group in preparation for physical removal
<param>	int	See comments	-	For 'auto' – *#modules* from which to create new group of modules (slot#'s unspecified). For 'new' – *group_ref* of new group of modules (e.g. '12', '1234', etc.) For 'mount' – *group_ref* of new group of modules (e.g. '12', '1234', etc.) For 'open' – *group_ref* to be opened for recording (e.g. '12' for group with modules in slot#s 1 & 2). For 'close' – null For 'protect' – *group_ref* to be write-protected. For 'unprotect' – *group_ref* to be 'unprotected' (i.e. write-enabled) For 'erase' – *group_ref* to be erased For 'unmount' - *slot#* or *group_ref* to be physically unmounted; error if attempt to unmount an open group

Query response parameters:

Parameter	Type	Values	Comments
<group_ref>	int	#modules | group_ref	#modules – For 'auto' create group_ref – For an existing group, *group_ref* must include all modules in group; order of slot#'s in group_ref is irrelevant (i.e. '12' is equivalent to '21')

Rules:

a. A new group of one or more modules can only be assembled from initialized (i.e. empty, unprotected) module(s).
b. A successfully created new group will be automatically 'mounted'.
c. Once created, a group of *n* modules is referenced by an *n*-digit number *group_ref* with the module slot#'s as unordered digits (e.g. '3', '12','314',1234').
d. Only one group may be 'open' at any given time.
e. A successful 'protect' command automatically closes the affected group, but leaves the group available for reading (such as 'scan_check').
f. A 'protected' group must be 'unprotected' before it can be opened for recording.
g. An 'erase' command must be immediately and unconditionally preceded by an 'unprotect' command (even if the group is already 'unprotected').
h. A 'unmount' command ensures that the specified group is quiescent before physical removal of the associated modules; an attempt to 'unmount' a currently open group will be rejected and cause an error return.

group

group_members – Get group eMSN members (query only)

[command list]

Query syntax: group_members? <slot> ;

Query response: !group_members? <return code> : <*cplane* return code > : <eMSN of slot> : [<eMSN2>] : [<eMSN3>] : [<eMSN4>];

Purpose: Get a groups eMSN members, by querying a specific module in a slot

Monitor-only parameters:

Parameter	Type	Values	Comments
<slot>	int	1-4	Slot with module being queried
<eMSN of slot>	ASCII		The eMSN of the slot being queried
[<eMSN2>]	ASCII		Get additional groups eMSN members (see Note 1)
[<eMSN3>]	ASCII		Get additional groups third eMSN members if it exists
[<eMSN3>]	ASCII		Get additional groups forth eMSN members if it exists

Notes:

1. If the module belongs to a group of greater than 1, the additional eMSN will be displayed. If the module belongs to a group of one, only the slots eMSN will be provided, or a "-" if it is not initialized into a group yet.

group_members

gsm –Set/get Group State Mask (NYI)

[command list]

Command syntax: gsm = <group_ref> : <GSM> ;
Command response: !gsm = <return code> : <*cplane* return code> ;

Query syntax: gsm? <group_ref>;
Query response: !gsm? <return code> : <*cplane* return code> :
<group_ref> : <GSM> : <eMSN1 [: eMSN2 : ...] ;

Purpose: Set/get Group State Mask (GSM): logs the last significant group operation; this is an NRAO-specific command.

Command parameters:

Parameter	Type	Allowed values	Default	Comments
<group_ref>	int	group_ref	-	
<GSM>	char	recorded | played | erased |unknown | error	none	To be used only if automatically-set GSM parameter is to be overwritten. Requires a preceding 'vol_cmd= <protect>:<group_ref>; command. Current value of GSM is ignored.

Query parameters

Parameter	Type	Allowed values	Default	Comments
<group_ref>	int	group_ref	All groups	Target group_ref

Query response parameters:

Parameter	Type	Values	Comments
<group_ref>	int	group_ref	Group reference
<GSM>	char	recorded | played | erased | unknown | error	recorded – last significant operation was record or a record-like function played – last significant operation was playback erased – last significant operation was erase unknown – ? error – error occurred; for example, a failure during one of the significant operations above
<eMSNn>	char	-	eMSN(s) of modules in group

Notes:

1. Normally, the setting of the GSM parameter happens automatically whenever a record, play, or erase command is issued. However, the <gsm=...> command is provided to manually overwrite the current GSM parameter. This command requires a preceding 'protect=off' and affects only the active module. A 'gsm=…" command ignores the current value of the GSM (see 'gsm_mask' command).
2. The GSM logs the last significant group operation. It is designed to distinguish between groups waiting to be correlated, have been correlated, or have no data (erased) and ready to be recorded. The GSM is saved on all modules in the group in the same area as the permanent MSN. The scan_check command does not affect GSM.
3. The 'gsm' and 'gsm_mask' commands were requested by NRAO and are designed primarily for use at their correlator.
4. If no modules are inserted, an error code 6 is returned.

gsm_mask – Set mask to enable changes in Group State Mask (NYI)

[command list]

Command syntax: gsm_mask = <erase_mask_enable> : <play_mask_enable> : <record_mask_enable>;
Command response: !gsm_mask = <return code> : <*cplane* return code> ;

Query syntax: gsm_mask? ;
Query response: !gsm_mask? <return code> : <*cplane* return code> :
<erase_mask_enable> : <play_mask_enable> : <record_mask_enable> ;

Purpose: Set mask to enable changes in Group State mask (GSM); this is an NRAO-specific command.

Command parameters:

Parameter	Type	Allowed values	Default	Comments
<erase_mask_enable>	int	0 | 1	1	0 – disable an erase operation from modifying the GSM. 1 – enable erase operation to modify the GSM.
<play_mask_enable>	int	0 | 1	1	0 – disable a play operation from modifying the GSM. 1 – enable play operation to modify the GSM.
<record_mask_enable>	int	0 | 1	1	0 – disable a record operation from modifying the GSM. 1 – enable record operation to modify the GSM.

Query parameters: None

Query response parameters:

Parameter	Type	Values	Comments
<erase_mask_enable>	int	0 | 1	
<play_mask_enable>	int	0 | 1	
<record_mask_enable>	int	0 | 1	

Notes:

The GSM mask is intended to prevent accidental changes in the GSM. When a module is at a station, the GSM setting of 1:0:1 will disable a play operation from modifying the GSM. Likewise, at a correlator one might want to disable the record_mask_enable.

gsm_mask

input_stream – Define input data stream(s)

[command list]

Command syntax: input_stream = <action> : [<stream_label>] : [<data_format>] : [<payload_size>]:[<payload_offset>] :[<psn_offset>]:[<interface_ID>]: [<filter address>] ;

Command response: !packet = <return code> : <*cplane* return code> ;

Query syntax: input_stream? [<stream_label>] ;

Query response: !input_stream? <return code> : <*cplane* return code> : <stream_label$_1$> : <data format1> : <payload_size1>:<payload_offset$_1$> :<psn_offset$_1$>:<interface_ID$_1$> : [<filter address$_1$>] [port#$_1$] : <stream_label$_2$> : <data format$_2$> : <payload_size$_2$>:<payload_offset$_2$> :<psn_offset$_2$>:<interface_ID$_2$> : [<filter address$_2$>] [port#$_2$] :…;

Purpose: Select input data stream

Command parameters:

Parameter	Type	Allowed values	Default	Comments
<action >	char	add | delete | commit	-	'add' - add a new input stream definition 'delete' – delete existing specified <stream_label> 'commit' – commit to the existing set of stream declarations. See Note 1 & 2.
<stream_label>	char	16 chars max	-	User-specified label for input data stream (16 chars max); example 'RDBE1'. See Note 3.
<data_format>	char	4 chars max	vdif	'vdif' – VDIF data format; 'm5b' – Mark 5B data format Other data formats may be specified. By default, a lower-case version of this parameter is used as the Mark 6 filename suffix unless overridden by a specification in the 'record=' command. See Note 4.
<payload_size>	int	64 < X < 9000	8224	The length of the payload received (bytes) to be recorded. See Note 5.
<payload_offset>	int	0 <= X < 256	42	The headers of the received packet to exclude from the recorded data. See Note 6.
<psn_offset>	int	0<= X < 9000	0	The offset from the start of the receive packet to the packet serial number. See Note 7.
< interface_ID >	char	16 chars max	-	System-defined interface name (example: 'ETH01' for Ethernet port 1)
< filter address >	int	IP or MAC address	-	Specify exclusive IP or MAC address from which will be recorded: For IP connection: IP address (i.e. 192.68.1.34) For MAC connection: MAC address (i.e. 01.23.45.67.89.ab; must use '.' as delimiter)
<port#>	int	port number	-	Destination port# to which data stream is addressed

Query parameters:

Parameter	Type	Allowed values	Default	Comments
<stream label>	int	-	All stream labels	Specified stream label

Query response parameters:

Parameter	Type	Values	Comments
<stream_label >	char	16 chars max	User-specified stream label
<data format>	char	4 chars max	Data format

input_stream

<payload_size>	int	64 < X < 9000	The length of the payload received (bytes) to be recorded. See Note 5.
<payload_offset>	int	0 <= X < 256	The headers of the received packet to exclude from the recorded data. See Note 6.
<psn_offset>	int	0<= X < 9000	The offset from the start of the receive packet to the packet serial number. See Note 7.
< interface_ID >	char	16 chars max	System-defined interface name
< filter address >		IP or MAC address	IP/MAC address for data filtering
<port#>	int	-	Destination port#

Notes:

1. The only required field for the command is <action>. If the action is add or delete, the remaining optional parameters must be provided. If the action is commit, no other parameters are required.
2. Multiple simultaneous input data streams may be specified (one at a time) and types may be intermixed.
3. The stream label is the identifier for a specified 10GigE data stream; each input data stream must have a separate stream label.
4. Each defined 'input stream' is recorded to a separate set of Linux files.
5. Payload size defines the packet length in bytes to be recorded after the headers are stripped off. This assumes that the first word contains the start of the Mark5B or VDIF header.
6. Payload offset is the number of bytes into the received payload that should be removed and not written to disk. If a VDIF payload is encapsulated directly into a layer 2 frame, the offset will be 0. If using VTP format, than the UDP/IPv4/Sequence number should be accounted for and skipped.
7. If the PSN_offset indicates the location of the packet serial number (PSN) to be compliant with VTP and also to guarantee the ordering of data stored to disk. A value of "0" implies the data plane will not look for a PSN or guarantee order of the data. A non-zero value will indicate the byte offset position, from the start of the payload received,

input_stream

list – List scans recorded on a mounted group (query only)

[command list]

Query syntax: list? [<group_ref>] ;

Query response: !list?<return code> : <*cplane* return code> : <group_ref>:
<$scan_1$> : <$length_1$>:<creation $time_1$> : …
<$scan_n$> : <$length_n$>:<creation $time_n$>;

Purpose: List scans recorded on specified group

Query parameters:

Parameter	Type	Allowed values	Default	Comments
<group_ref>	Char	-	Active group module	Lists the scans recorded on specified group. See Notes 1 &2

Query response parameters:

Parameter	Type	Values	Comments
<group_ref >	char	16 chars max	Module group reference listing is valid for
<$scan_x$>	char	4 chars max	Scan name
< length >	int		Length of the scan in Gbytes bytes
< creation time >	ASCII		Date when the scan was created

Notes:

1. If recording is not currently active: A list of scans will be obtained by reading from the modules(s) of the requested group, which may be open or closed.
2. If recording is active:
    a. a ‘list’ request for a ‘closed’ group will unconditionally return with an error code ‘5’.
    b. a ‘list’ request for an ‘open’ group will be serviced by listing the scans cached in memory.

m6cc – Execute the mark6 command and control application (TBD)

[command list]

Command syntax: m6cc = <config_file>;
Command response: !m6cc = <return code> ;

Query syntax: m6cc? ;

Query response: !m6cc ? <return code> : <*cplane* return code>
: <config_file> : <state> ;

Purpose: Execute the Mark6 command and control application using a specified XML configuration file describing actions.
Command parameters:

Parameter	Type	Allowed values	Default	Comments
<config_file>	Ascii			An existing configuration file uploaded to the Mark6.

Query response parameters: (returns information for most recently initialized module)

Parameter	Type	Allowed values	Comments
<config_file>	Ascii		Name of the file being processed, or the last file processed.
<state>	char	active | complete | partial | error	'active' – a configuration file is being processed and executed. 'complete' – the configuration file was successfully processed through completion. 'partial' – the configuration file was processed and terminated before completion. 'error' – error condition exist e.g. the file does not exist.

Notes:

m6cc

mod_init – Initialize a disk module and assign a Module Serial Number (MSN)

[command list]

Command syntax: mod_init = <slot#> : <#disks> : <MSN> : [<type>] : [<new>];
Command response: !mod_init = <return code> ;

Query syntax: mod_init? ;

Query response: !mod_init ? <return code> : <*cplane* return code>
: <slot#> : <eMSN1> : <#disks discovered> ;

Purpose: Initialize a disk module by assigning it a unique 'permanent' Module Serial Number (MSN).

Command parameters:

Parameter	Type	Allowed values	Default	Comments
<slot#>	int	1-9		Physical slot# of module to be initialized
<#disks>	int	1-16	-	#disks installed in module. See Note 1
<MSN>	char		-	Permanent 8-character 'Module Serial Number' to be assigned to the connected disk module (example:'MPI_0153'). MSN format rules are enforced; see Note 2. The module capacity and disk manufacturer are extracted and appended to the MSN to create the 'extended-MSN' ('eMSN') [example 'MPI_0153/16/WD'); see Notes 1 and 3.
<type>	char	sg | raid	sg	The format structure that the disk module will be created in: 'sg' – scatter gather resilient format disk module 'raid' – Raid 0 formatted disk module
<new>	char	new | null	null	Required if MSN of previously initialized module is being changed. See Note 2.

Query response parameters: (returns information for most recently initialized module)

Parameter	Type	Allowed values	Comments
<slot#>	int	1-9	Physical slot#
<#disks discovered>	int	1-16	#disks in associated module
<extended MSN>	char	-	Example: 'MPI_0153/16/4/WD'; see Notes 3 and 4.

Notes:

1. The 'mod_init=..' command is normally issued only when a module is first procured or assembled, when the disks are changed or upgraded, or when a multi-module group is dissolved. All connected disks in the specified slot# will be initialized; all data on the module will be erased. The number of disks specified must exactly match the number of disks discovered by the Mark 6 controller in the specified slot#. An error will be returned if #disks specified are different from number of disks discovered by the Mark 6 controller.

mod_init

2. For a module that has been previously initialized: Unless the 'change' parameter is specified, the specified MSN must match the existing module MSN or the initialization request will fail. This procedure is intended to help prevent inadvertent changes to the MSN.
3. The format of the **Module Serial Number** ('MSN' for short – see Note below) is enforced as follows:
   MSN must be 8 characters in length with format 'ownerID_serial#' where:
   a. ownerID – 2 to 5 upper-case alphabetic characters (A-Z). The 'ownerID' must be registered with Jon Romney at NRAO (jromney@nrao.edu) to prevent duplicates. Numeric characters are not allowed. Any lower-case characters will automatically be converted to upper case.
   b. serial# - numeric Module Serial Number, with leading zeroes as necessary to make the MSN exactly 8 characters long; minimum two digits, three recommended . Alphabetic characters are not allowed in the serial#.

   Note: 'MSN' is equivalent to VSN or 'Volume Serial Number' in the Mark 5 world.
4. '**Extended-MSN**' ('eMSN') is of the form 'MSN/capacity(TB)/max data rate(Gbps)/disk manufacturer'. *cplane* queries the disks to compute the capacity of the module in TB and the disk manufacturer (e.g. 'WD', 'SG', 'HT', 'SS', for Western Digital, Seagate, Hitachi, Samsung, for example); if mixed-disk models, <disk type> will be set to 'MX'; these fields are then appended to the to create the e-MSN. Mixed disk models and sizes are supported by the Mark 6, but not encouraged, particularly mixed disk sizes.
5. The replacement of any disk(s) in an initialized module requires re-initialization of the module with the 'mod_init=' command before data can be recorded. All existing data will be lost.

msg – Get ASCII message associated with *cplane* return code(query only)

[command list]

Query syntax: msg? <*cplane* return code> ;
Query response: !msg ? <return code> : <*cplane* return code> : <*cplane*-specific ASCII message> ;

Purpose: Get ASCII error message associated with specified *cplane* return code

Query parameters:

Parameter	Type	Allowed values	Default	Comments
<*cplane* return code>	int	-	-	*cplane* numerical return code

Query response parameters:

Parameter	Type	Values	Comments
<*cplane* return code>	int		<*cplane* return code>
<error message>	literal ASCII		Associated ASCII error message, if any

Notes:

1. The 'error?' query is used to retrieve an explanation of *cplane*-specific return codes (usually errors).
2. <return code> 8 (parameter error) returned if specified <*cplane* return code> does not exist.

mstat – Get module status (query only)

[command list]

Query syntax: mstat? [<type>];

Query response: !mstat? <return code> : <*cplane* return code> :
<group_ref1> : <slot#> : <eMSN1> : <#disks discovered> : <#disks nominal> : <#GB remaining> :
<#GB total> : <status1> : <status2> :<type>
[: <group_ref2> : <slot#> : <eMSN2> : <#disks discovered> : <#disks nominal> : <#GB remaining> :
<#GB total> : <status1> : <status2> : <type>]
[:…] ;

Purpose: Get module status

Query parameters:

Parameter	Type	Allowed values	Default	Comments
<type of query>	int/ char	null | all <slot#> | <group_ref> | open	open	all|null returns information on all mounted modules. <slot#> returns information module in specified slot#. group_ref or partial group_ref (i.e. one or more mounted modules of multi-module group) 'open' returns information on all modules in currently open group (if any).

Query response parameters Returns following information about specified group(s)

Parameter	Type	Values	Comments
<group_ref>	int	int	Standard group_ref if complete group is mounted (e.g. '2' or '12' or '1234'); Trailing zero assigned for every 'unmounted' module of multi-module group (e.g. '10' for two-module group associated with module in slot#1, but only module in slot #1 is mounted; '1400' for 4-module group associated with modules in slots 1 & 4, but only two modules mounted) Zero if module is not assigned to a group (i.e. <status1> is 'initialized' or 'unknown')
<slot#>	int	0-9	Physical slot# for mounted module. Zero for 'unmounted' module of multi-module group.
<eMSN>	char	-	Extended Module Serial Number (see Notes for 'mod_init' command)
<#disks discovered>	int	-	Number of disks in module discovered; zero if 'missing' module
<#disks registered>	int	-	Number of disks registered
<#GB remaining>	int		#GB remaining unrecorded on group
<#GB total>	int		#GB total storage in group
(continued)			

<status1>	char	recording | open | closed | mounted | incomplete | unmounted | initialized | unknown	recording – associated group is recording (open – associated group is complete and open closed – associated group is closed mounted – associated group is mounted incomplete – associated group has one or more unmounted modules unmounted – unmounted module of a multi-module group initialized – initialized, empty, unprotected module unassigned to a group unknown – status of module is unknown
<status2>	char	ready | recording | flushing | protected | unprotected | null	ready – associated group is ready to record (<status1> must be 'open') recording – associated group is recording (<status1> must be 'open') flushing – associated group is flushing buffers to disks (<status1> must be 'open') protected – associated group is protected (<status1> may be 'closed' or 'incomplete') unprotected – associated group is unprotected, i.e. write enabled (<status1> may be 'open', 'closed', or 'incomplete') null – applies to all 'initialized', 'unmounted' or 'unknown' modules
<type>	char	sg | raid	The format of the disk module. scatter gather (sg) or raid0 (raid)

Notes:

1. Status of all mounted modules is returned, as well as information about unmounted modules in incomplete groups.
2. Every module within a group contains information about all modules in the group; this allows the 'mstat' command to return complete identifying information for all unmounted modules within a group.
3. The information returned by the 'mstat' command is usefully presented in tabular form. The following table illustrates an example (for illustrative purposes, shows more slots than are normally available):

Group	Slot #	eMSN	#disks found	#disks registered	GB (remaining)	GB (total)	Status1	Status2	Type	Comments
12	1	ABC_0450/8/4/WD	8	8	7800	8000	open	unprotected	sg	Group 12 is ready to record.
12	2	ABC_0458/8/4/WD	8	8	7799	8000	open	unprotected	sg	
-	3	XYZ00123	8	8	16000	16000	initialized	(null)		Available to be assigned to a group
-	4	XYZ00124	8	8	16000	16000	initialized	(null)		Available to be assigned to a group
56	5	QRS_0900/16/4/SS	8	8	3	16000	closed	protected	sg	Group 56 is full, closed and protected, and may be physically unmounted
56	6	QRS_0901/16/4/SS	8	8	4	16000	closed	protected	sg	
7-	7	QRS_0902/16/4/SS	8	8	8765	16000	incomplete	unprotected	sg	Slot 7 is part of an incompete 2-module group.
8	8	CDE_0321/16/4/HT	8	8	16000	16000	closed	unprotected	raid	Single-module group. Full.

record – Turn recording on|off; assign scan label

[command list]

Command syntax: record = <action> : [<duration>] : <data_size> : [<scan_name>] : [<experiment_ name>] : [<station_code>] ;

Command response: !record = <return code> : <*cplane* return code> ;

Query syntax: record? ;

Query response: !record ? <return code> : <*cplane* return code> : <status>: <group> : <scan number> : <scan name> ;

Purpose: Turn recording on|off or specify UT start time.

Command parameters:

Parameter	Type	Allowed values	Default	Comments
<action>	char	on | off | start_time		'on' immediately starts recording; 'off' stops recording ; See Note 5 'start_time' specifies UT start time in VEX time format (##y##d##h##m##s)'. e.g. 12y215d12h45m30s, See Notes 1-4.
<duration>	int	seconds		If specified, recording will stop after this amount of time; 'record=off' command may be used to stop recording before specified duration is complete.
<data size>	int	GBytes	-	Size of the data packet expected to be recorded in this scan; used to judge whether sufficient space exists on present 'ready' volume. See Note 3.
<scan_name>	char	32 chars max		Relevant only for 'record=on' command. If not specified, the current Mark 6 system UT time will be used to create a scan name in the format 'scan_xxxx', where 'xxxx' is a Mark6 generated scan serial number
<experiment_name>	char	8 chars max	expxx	Relevant only for 'record=on' command; once specified, <experiment name> will be remembered on subsequent 'record=on|time' commands; 'record=on|time' with <experiment name> will reset default.
<station_code>	char	8 chars max	stnxx	Relevant only for 'record=on' command; once specified, <station name> will be remembered on subsequent 'record=on|time' commands; 'record=on|time' with <station name> will reset default.

Query parameters: None

Query response parameters:

Parameter	Type	Values	Comments
<status>	char	off | pending | recording | flushing	'off' – recording currently inactive 'pending' – awaiting timed start 'recording' – capturing data to RAM and recording to disk 'flushing' – flushing RAM buffers to disk after data capture has stopped
<group>	int		Group to which current recording (or last recording) was made
<scan number>	int		Mark6-assigned scan serial number within group
<scan name>	char		Scan name –see Section 6 for definition of scan name.

Notes:

1. Recording always takes place to the ‘open’ group.
2. VEX time format is ‘##y###d##h##m##s’, but may be truncated from left or right as long as it is unambiguous at the time it is issued (e.g. ‘30m10.5s’ will start at specified time within the next hour).
3. While waiting for ‘time’ recording to start, status of ‘open’ group is not allowed to change.
4. An error will be returned if specified ‘start time’ plus the ‘duration’ has passed; If the ‘start time’ has passed, but the ‘end time’ has not, the recording will capture data from when the command is received (on an integer second), till the ‘end time’ or when ‘record=off’ is issued.
5. A ‘record=<off>’ command will disable ‘start_time’ recording if issued before ‘start_time’; If issued during an active recording, the data received up till that time will be captured; If issued after the ‘end_time’ and error will be returned.
6. During recording, commands to the system that may interfere with the recording capability of the disks (such as queries for ‘disk_info?’, ‘scan_info’, ‘scan_check’, etc) may be rejected.
7. If the <data size> parameter is specified, *cplane* will first check the present ‘open’ group for sufficient space. If <data size> is specified and insufficient space is available, an error will be returned. During or after recording (until the next scan is started), the ‘scan_info?’ query may be used to return information about the scan
8. For recording performance reasons, once a scan is recorded, there is no mechanism to delete it (without initializing modules within group).

record

rtime – Get remaining recording time on open group (query only)

[command list]

Query syntax: rtime? [<data rate>];
Query response: !rtime ? <return code> : <*cplane* return code> :
<group> : <data rate> : <remaining time> : <GB remaining> : <GB total> ;

Purpose: Get remaining recording time of open group for specified data rate.

Query parameters:

Parameter	Type	Allowed values	Default	Comments
<data rate>	real	Mbps	-	Usually will be data rate of next scan to be recorded

Query response parameters:

Parameter	Type	Values	Default	Comments
<data rate>	real	Gbps	-	Recording rate assumed in calculation of <remaining time>
<remaining time>	int	seconds	-	Approximate remaining record time for currently 'open' group at specified data rate.
<GB remaining>	int	GB		GB remaining on open group
<GB total storage>	int	GB		GB total available storage on open group

Notes:

1. If an 'rtime?' query is issued during recording and <data rate> is not specified, the Mark 6 system estimates the actual aggregate recording data rate and returns the <remaining time> estimate accordingly.

scan_check – Quick check of recorded data (query only) (NYI)

[command list]

Query syntax: scan_check? [<scan name|scan#>] : [<group_ref>] ;
Query response: !scan_check ? <return code> : <*cplane* return code>: <group_ref> : <scan#> : <scan label> : <#streams> : <stream label1> : <status1> : <data format1> : <start time1> : <duration1> : <datasize1> : <stream rate1> : <missing bytes1> : <stream label2> : <status2> : <data format2> : <start time2> : <duration2> : <datasize2> : <stream rate2> : <missing bytes2> : <stream label*n*> :…: <missing bytes*n*>] ;
Purpose: Quick-look check of recorded data from completed scan
Query parameters:

Parameter	Type	Allowed values	Default	Comments
<scan name|scan#>	char/int	-	Last recorded scan	See Note 2.
<group_ref>	int	-	open group	group_ref or partial group_ref (i.e. one or more mounted modules of multi-module group)

Query response parameters:

Parameter	Type	Values	Comments
<group_ref>	int	-	Group reference
<scan#>	int		Mark6-assigned sequential scan# within the group
<scan label>	char	-	Scan label (see Section 6)
<#streams>	int	-	#streams of recorded data
<stream labelx>	char	-	Stream label as defined by 'input_stream=' command
<statusx>	char	OK | time? | data?	'OK' – data frames and data appear to be normal 'time?' – error decoding time in data frames 'data? – sampled data do not appear to be statistically random; see Note 5
<data formatx>	char	-	Data format ('vdif', 'm5b', etc)
<start timex>	time	UT time	Time tag decoded from first data-frame in file, in format ##y###d##h##m##ss
<durationx>	time	seconds	Decoded time interval between first and last data-frames of scan
<data sizex>	real	GB	Total data recorded (GB) from data tream; see Note 6
<stream data ratex>	real	Gbps	Inferred stream data rate (Gbps) based on starting/ending time tags and <file size>; see Note 6.
<#missing bytes>	int	int	#bytes missing from recording

scan_check

Notes:

1. ‘record?’ status must be ‘off’ when doing scan_check; attempting scan_check during recording will return an error.
2. ‘scan#’ is the Mark6-assigned sequential scan# within the group.
3. ‘group_ref’ may be a full group reference or partial group reference (i.e. one or more mounted modules of multi-module group)
4. <data format> is taken from scan metadata recorded in the group directory; supported suffixes are ‘vdif’ and ‘m5b’ (plugins may be developed to support other data formats).
5. ‘scan_check’ examines a small amount of data near the scan start and stop points; the data are decoded according to the specified format.
6. Statistical randomness of data is based on a small sample of data at both beginning and end of scan; not authoritative, but should be looked at.
7. Mark 6 records entire Ethernet frames, which is taken into account in calculating <stream data rate>.

scan_check

scan_info – Get scan information (query only) (NYI)

[command list]

Query syntax: scan_info? [<scan name|scan#>] : [<group_ref>] ;
Query response: !scan_info ? <return code> : <*cplane* return code> : <group_ref> : <scan#> : <scan label> : <status> : <start time> : <duration> : <#data streams> : ,<system performance code> ;

Purpose: Get summary information on scan

Query parameters:

Parameter	Type	Allowed values	Default	Comments
<scan name|scan#>	char/int	-	Last recorded scan	See Note 2.
<group_ref>	int	-	open group	group_ref or partial group_ref (i.e. one or more mounted modules in multi-module group)

Query response parameters:

Parameter	Type	Values	Comments
<group_ref>	int	-	Group reference
<scan#>	char	-	Mark6-assigned sequential scan# within the group
<scan label>	int	-	Scan label (see Section 6)
<status>	char	recording | pending | flushing | complete	'recording' – capturing data to RAM and recording to disk 'pending' – awaiting timed start 'flushing' – flushing RAM buffers to disk after data capture has stopped 'compete' – all data recorded to disk
<start time>	time	UT time	Recording UT start time in format '##y###d###h##m##ss', where '#' is a digit.
<duration>	int	seconds	Recording duration in seconds
<#data streams>	int	-	# of data streams in scan; corresponds to # of files per disk (one file per data stream)
<system performance code>	int	-	Code indicating system performance. See Note 3

Notes:

1. The 'scan_info?' may be rejected if the system is too busy during active recording.
2. 'scan#' is the Mark6-assigned sequential scan# within a group.
3. 'quality code' is intended to be a indication of the performance of the Mark 6 on the basis on *cplane* internal metrics; for example, excessive buffer overflow or unequal distribution of data to disks (indicative of one or more 'slow' disks); actual codes and meanings TBD.

scan_info

status – Get detailed Mark 6 system status (query only)

[command list]

Query syntax: status? ;
Query response: !status ? <return code> : <*cplane* return code>: <status word> ;

Purpose: Get detailed Mark 6 system status

Query parameters: None

Query response parameters: (TBD by software author)

Parameter	Type	Values	Comments
<status word>	hex	-	Bit 0 – (0x0001) system 'ready' Bit 1 – (0x0002) error message(s) pending; (message may be appended); messages may be queued; error is cleared by this command. See also 'error?' query Bit 2 – (0x0004) one or more 'delayed-completion' commands are pending. Also set whenever any data-transfer activity, such as recording. Bit 3 – (0x0008) one or more 'delayed-completion' queries are pending --- Bit 4 – (0x0010) record 'on' Bit 5 – (0x0020) media full (recording halted) Bit 6 - (0x0040) burst mode Bit 7 - (0x0080) recording can't keep up; some lost data --- Bit 8 - (0x0100) dplane is operational Bit 9 - (0x0200) dplane is configured to accept data Bit 10 – (0x0400) not used Bit 11 – (0x0800) not used --- Bit 12 – (0x1000) Bank 1 selected Bit 13 – (0x2000) Bank 1 ready Bit 14 – (0x4000) Bank 1 media full or faulty (not writable) Bit 15 – (0x8000) Bank 1 write protected --- Bit 16 – (0x10000) Bank 2 selected Bit 17 – (0x20000) Bank 2 ready Bit 18 – (0x40000) Bank 2 media full or faulty (not writable) Bit 19 – (0x80000) Bank 2 write protected --- Bit 20 – (0x100000) Bank 3 selected Bit 21 – (0x200000) Bank 3 ready Bit 22 – (0x400000) Bank 3 media full or faulty (not writable) Bit 23 – (0x800000) Bank 3 write protected --- Bit 24 – (0x1000000) Bank 4 selected Bit 25 – (0x2000000) Bank 4 ready Bit 26 – (0x4000000) Bank 4 media full or faulty (not writable) Bit 27 – (0x8000000) Bank 4 write protected

status

sys_info – Get Mark 6 configuration details (query only)

[command list]

Query syntax: sys_info? ;

Query response: !sys_info ? <return code> : <*cplane* return code> :
<system type> : <Mark 6 S/N> : <OS type/rev> : <*cplane* version number> :
<command set revision> : <available RAM> :<#data disks supported> :
<#Ethernet input ports> : <portref1> : <portspeed1>:<address1>:<state1> :…. : <portref*n*> :
<portspeed*n*> ;

Purpose: Get Mark 6 configuration details

Query parameters: None

Query response parameters:

Parameter	Type	Values	Comments
<system type>	char	Mark6	
<Mark 6 serial number>	char		Assigned system serial number; generally in the form 'Mark5-4xxx'
<OS type/revision>	char		example: 'Ubuntu 10.10'
<*cplane* version number>	char		Version number of *cplane*
<command set revision>	char		Mark 6 DIM command set revision level corresponding to this software release (e.g. '1.0')
<available RAM>	int		Available burst-mode RAM (GB)
<#data disk supported	int		max# data disks supported on system
<#Ethernet input ports>	int		#Ethernet ports available for data input
<portref*x*>	char		Port reference name to be used in 'input_stream" command (e.g. 'eth0')
<portspeed*x*>	int	1 | 10	Nominal Ethernet speed (Gbps) (e.g. '1' or '10')
<address>	ascii	xx.xx.xx.xx	IPv4 Address assigned to interface
<state>	char	up | down	State if the network interface. See Note 1.

Notes:

1. If the Ethernet interface is down, the operator must log-in as root and configure and/or re-establish the Ethernet connection

sys_info