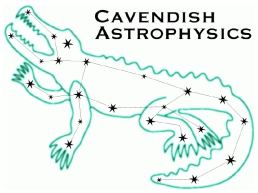
MRO Delay Line

Network Message Protocols and Telemetry/Status/Logs File Format

John Young

rev 1.2
21 June 2010



Cavendish Laboratory
JJ Thomson Avenue
Cambridge CB3 0HE
UK

Change Record

Revision	Date	Authors	Changes
0.5	2006-02-24	JSY	Initial version
0.6	2006-04-27	JSY	Revised following internal discussions. Removed code snippets — dlmsg library has its own manual
0.8	2006-05-04	JSY	Small corrections. Added Table 1
0.9	2006-07-19	EBS	Actual command names
0.10	2007-01-19	EBS	Changes to status/telemetry/commands for trolleys
0.11	2007-03-22	JSY	Agreed Trajectory message format, changes to commands for VME, more actual command names
0.12	2007-05-30	JSY	Updated status/telemetry descriptions, added MetrolError$N$ item
0.14	2008-03-27	JSY	Added note about current DL_TELEMETRY implementation
0.15	2009-04-03	JSY	Workstation status/telemetry names no longer have suffixes; all items now listed. Corrections to trolley items. Corrections to FITS format
0.16	2009-09-04	JSY	Version 2 status message format
0.17	2009-11-18	JSY	DL_LOG FITS table
1.0	2010-01-13	JSY	Updated Table 1. Removed command data from FT. Added telemetry format extension for shearcam video. Restructured document and incorporated socket connection protocol. Updated introduction for Production Software PDR, deleted appendix containing old suggestions for production software architecture
1.1	2010-06-18	EBS	Added Agilent hardware status booleans to Subsubsection C.1.2
1.2	2010-06-21	JSY	Trolley and workstation "MotorVel" status items renamed

Objective

To describe the network protocols for sending commands, status information and telemetry between the sub-systems of the delay line control system. To identify which com-

mands and telemetry/status items are sent to/from each sub-system. To describe the FITS-based file format used to record telemetry and status information and to log commands and diagnostic messages, when the delay line control system is operating in a standalone mode.

Scope of this document

This document provides a detailed description of the protocols used for inter-subsystem communication over Ethernet in the delay line control system. These communications encompass both control of the delay lines and routing of status information and diagnostic telemetry to the delay line Workstation (and hence to the ISS). We also describe the FITS-based file format used by the workstation software for recording status and telemetry, and logging commands and diagnostic messages, whose structure is closely related to the network message formats.

The same internal communication protocols will be used by the components of the delivered prototype and production software, so this document applies equally to both sets of software.

Reference Documents

RD1 Production Delay Line Control Software Architecture INT-406-VEN-xxxx

RD2 dlmsg Library Manual (distributed with the dlmsg library source code)

Applicable Documents

AD1 Trolley Electronics Design Description INT-406-VEN-0112

Acronyms and Abbreviations

API Application Programming Interface

DL Delay lines

dlmsg Cavendish delay line messaging protocol

ISS Interferometer Supervisory System

MROI Magdalena Ridge Observatory Interferometer

OPD Optical Path Difference

VME VERSAmodule Eurocard, bus type used by the delay line Metrology computer

Contents

1 Introduction

The delay line control system is a distributed, event-driven system, comprising software running on the following computers:

- A "workstation" PC (shared between all delay lines) to act as a system controller.
- A VME-bus CPU (shared between all delay lines) to read the metrology signal and hence control the Cat's-eye.
- A low-power PC104 single-board micro on each trolley, to control onboard functions with undemanding timing requirements, and to send telemetry to the Workstation.
- A rack-mounted PC connected to each shear camera, to capture camera frames and compute shear corrections which are applied to the Cat's-eye secondary tip-tilt stage.

The production workstation software will provide command and monitoring interfaces to the MRO Interferometer Supervisory System (ISS). The workstation software (both prototype and production) also provides a user interface for operating the delay line in a standalone mode.

The architecture of the production delay line control system is described more fully in RD1.

2 Control System Information Flow

The flow of information between the sub-components of the control system is shown in Figure 1. Many of the signals are transmitted as messages over Ethernet; we categorise these messages as follows:

1. Commands (& Acknowledgements)
2. (Command) Data:
   information needed in real-time to close servo loops
3. Status (includes command acknowledgements and log/fault messages)
4. Telemetry

The messaging protocols described in the following sections are summarised in Table 1.

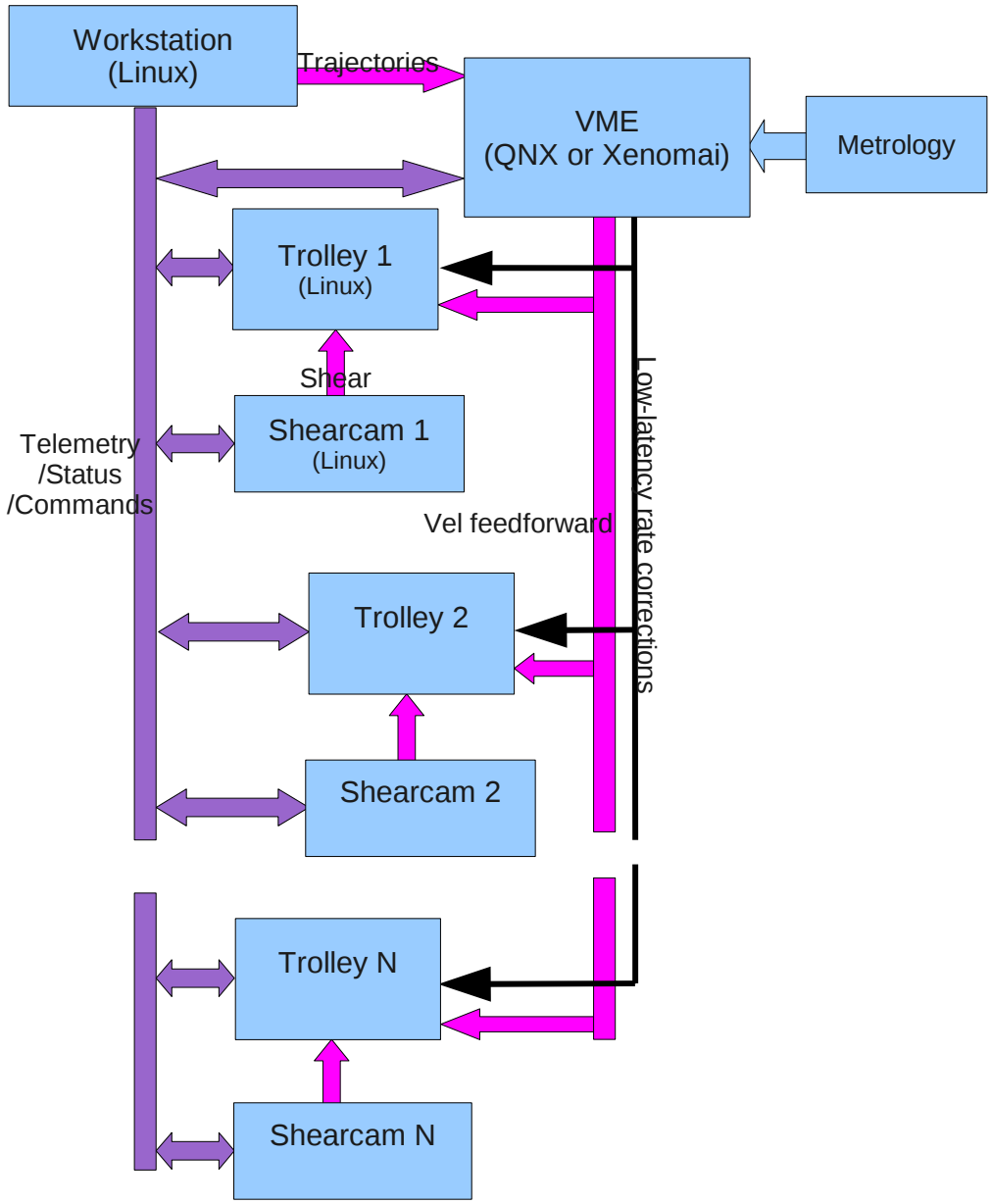


Figure 1: Inter-subsystem information flow in the delay line control system, with the delay lines operating in a standalone mode. The production delay line software will incorporate additional interfaces to the ISS and Fringe Tracker (not shown in this diagram), which will use different communication protocols to those described in this document. Telemetry, status and command message flows are shown as purple arrows. Command data streams (labelled individually) are shown in magenta. The thin black arrows represent control signals transmitted via interfaces other than Ethernet; these signals are described in AD1.

Table 1: Summary of delay line messaging protocols. The table uses the standard identifiers for delay line sub-systems listed in Table 4. Note that copies of command data are transmitted to the Workstation by the originating sub-system, as telemetry, in order to log the data. Commands are logged directly by the Workstation.

Msg. type	Source	Destination	Msg. Rate /Hz	Use of content
Status	TRLYn, VME, SHEARn, Workstation	Workstation	10–30	System-level control, displayed, recorded
Telemetry	All	Workstation	1–10	Recorded
Command	Workstation	VME, SHEARn, TRLYn	Async.	Obey command
Command Data	Workstation, VME, SHEARn	VME, SHEARn, TRLYn	1–30	Close loop

2.1 Design Aims

The messaging protocol and its C library implementation were designed with the following aims in mind:

- Provide the capability to record all control signals (hardware and software), for debugging the prototype delay line
- Use well-defined message protocols, and document these thoroughly
- Provide a flexible messaging system, that allows e.g. adding/changing signals with minimal knock-on effects
- Use the same protocols for the "prototype" and "production" control software so that the prototype and production components can be used interchangably

2.2 Connection Protocol

The protocol for establishing communication between the various computers is described in Sec. A. In brief, the Workstation acts as a server, listening for connection attempts on a pre-arranged TCP/IP port. Each sub-system connects to the server, establishing its own socket connection which it subsequently uses to transmit status and telemetry to the Workstation. The Workstation uses the same connection to send commands in the opposite direction. A separate socket connection is made for each command data stream (Sec. B.1).

2.3 Commands & Command Data

We have distinguished between commands and command data (henceforth we will refer to command data as "data" where this is unambiguous): the former are sent from the Workstation to the delay line sub-systems, usually asynchronously. The latter (e.g. shear offsets) are used to close servo loops in the system. Typically command data are transmitted at a fixed rate from one specific sub-system to another. The relevant servo loop is activated or deactivated in response to commands from the Workstation to the sub-system receiving the data; typically the data is always sent whether the system state requires it or not.

As described below, command acknowledgements are incorporated into the status messages transmitted to the Workstation. There are no explicit acknowledgements for data messages.

Commands and their corresponding acknowledgements are logged to the same file as telemetry and status information (defined in the next section) — see Sec. D. Command data are logged by sending a copy of the data from the originating sub-system to the Workstation as telemetry.

We give more details on the protocols for transmitting commands and data in Sec. B.

2.4 Telemetry & Status

We define telemetry to consist of "measurements" made primarily for debugging the delay lines. We treat measurements from all physical sensors in the delay line system as potential telemetry data. Telemetry can also include values of variables within the delay line control software.

Detailed lists of the telemetry items identified thus far are given in Sec. C.1. We envisage that telemetry is digitised and buffered locally before being transmitted (in "chunks" at typically one-second intervals) over the Ethernet to the Workstation, where it is buffered prior to possible archiving, processing and display.

We define "status" to consist of information used by the control system and by the user in controlling the delay lines. By this definition status includes:

- Information (mostly boolean) about the state of a sub-system, which typically changes in response to commands.
- Command acknowledgments, to provide near-immediate feedback on whether each command was accepted.
- Information about whether the delay line is performing acceptably (e.g. OPD jitter)
- Information that should be displayed in real time (e.g. trolley position)

Status messages are sent at regular intervals (every $\sim$0.1 s). The message format allows these to contain arbitrary boolean and numerical status items.

Special components of each status message indicate whether any commands have been received since the previous status message was issued, and for each such command, whether the command and any associated parameters are valid, and whether the command will be acted on. In this way the status message incorporates command acknowledgement(s). Any subsequent changes of state in response to a command will be indicated by (regular) status messages, i.e. changes in the status items listed in Sec. C.1.

Each status message may contain multiple log/fault notifications, each comprising:

- a log/fault type specifying the category of log/fault and its level (severity),
- a bitmask specifying which of the (up to 10) delay lines the log/fault applies to, and
- a human-readable message.

In the case of faults, the first component of the message string is a name for the fault. The log/fault types supported for status messages (see Table 9) have been chosen in order to map straightforwardly to the types and levels for logs and faults understood by the ISS.

The essential distinction between telemetry and status, as defined here, is that the former is only employed (in near-real-time or after the fact) by the user to debug the system, whereas status information is used by the control system and interactively by the user in controlling the delay line (and may also be used for debugging purposes). This is the reason for requiring status information to be sent more frequently compared with telemetry chunks.

However, status information is logged to the same file as telemetry, so that the user can correlate the information when debugging the delay line.

In the control system architecture, status and telemetry messages are only sent to the Workstation (see Figure 1). The Workstation contains all of the system-level intelligence necessary to control the delay line.

Transmission of telemetry/status from different sub-systems will be staggered (use of NTP will be sufficient to synchronise this), to avoid overloading the network.

2.5 Message Formats

Messages are encoded using the locally-written "Serialise" library , and transmitted using sockets over TCP/IP. The serialise library implements a compact yet flexible binary representation, and is described in the serialise manual. The Serialise library (described in RDxx) implements messages that are automatically self-describing in the sense that the receiving software can deduce the contained data types/lengths and their order/grouping from just the message itself.

The message formats defined in the appendix add another layer of self-description which labels each data item and provides meta-data information such as the units and timing of the measurement.

2.6 Implementation

A C library, "dlmsg", has been written to facilitate composing, transmitting, receiving and decoding telemetry/status messages. This library is described in its own manual [RD2].

The dlmsg library contains code for sending and receiving all four types of message identified in this document (status, telemetry, command, and command data). Support for concatenated status messages (Sec. C.3.1)and concatenated telemetry messages (Sec. C.2.1) is included.

All delay line sub-systems use the library, in order to reduce the overall development time and make messaging more reliable. The library hides details of the message formats from callers, so the formats can straightforwardly be modified if necessary.

The library is small (when compiled without debugging symbols, the size of the binary is approximately 40 kilobytes), and portable. It has been fully tested on the Linux and QNX Neutrino operating systems.

A Socket Initialisation Protocol

The text in this section was taken from the document "Socket Initialisation Protocol for Delay Line Computers" by Bodie Seneta (rev 1.0).

This section describes how the computers should connect to each other so that messages can be passed between them as required for a functioning delay line. We assume that the reader has a basic knowledge of C programming and the concept of interprocess communication using unix ports and sockets.

A.1 Definitions

For clarity we define the following:

Source A computer program that generates delay line messages.

Sink A computer program that receives delay line messages.

Command, command data, status, telemetry All are message types as set out above.

Controller A computer program that sends commands (a "command source"). Normally the workstation is the only controller, but if there is a test program running on another computer that sends commands then it is also a controller.

Controllee A computer program that accepts commands (a "command sink"). Controllers and controllees need not be running on separate computers.

Port1, port2 Unix ethernet port numbers. Port1≠port2. Port1 is used for status, telemetry and command messages. Port2 is used for command data messages. The current de-facto standard is to use ports 5000 and 5001.

SocketA, socketB,...,socketF Unix sockets, which should be created using the socketlib library (distributed with the serialise library described in "Serialise Network Message Protocol").

ProgamA, programB : Two programs that communicate with each other via a socket connection.

A.2 The Connection Protocol

There are actually two connection protocols. The first is for establishing connections where commands, status and telemetry information is exchanged and the second is for command data, which has simpler requirements. In both cases, the protocol is the same whether the delay line network is being initialised or a connection is being restored due to a prior fault or a deliberate disconnection.

A.2.1 Command, status and telemetry connection protocol

This protocol makes use of the fact that a given controllee only sends telemetry and status information to one other program (the controller) and only expects commands to arrive from that same program.

1. Controllers should be initialised (possibly using `CreateListenPort()` from socketlib) so that they listen for connection attempts from anywhere on socketA using port1.
2. Controllees should initiate a connection with a controller by creating socketB (possibly by using `ConnectToServer()` or `ConnectToServerWithTimeout()`) which is intended for transmission of status and/or telemetry messages to the controller on port1. Controllers do not initiate such connections[1].
3. When the controller notices that a connection is being attempted on port1, it creates socketC to receive subsequent messages from the controllee on port1 and to send any commands to it (also using port1). As port1 connection attempts arrive from other sources, further sockets are created as necessary.
4. When the controllee notices that socketC exists (that is, its attempt to create socketB was successful), it should start to listen for incoming commands from the controller using socketB.
5. The connection is now established. The controllee can use socketB to send status and telemetry information to and receive commands from the controller. The controller can use socketC to send commands to and receive telemetry and status information from the controllee.

A.2.2 Command data connection protocol

For command data, a command data sink can expect command data to arrive from anywhere, but it does not need to send anything back to the source.

1. Command data sinks should be initialised so that they listen for command data from anywhere using port2 on socketD.
2. A command data source initiates communication with a command data sink by creating socketE which is intended for transmission of command data using port2.

[1] Rationale: Under normal circumstances the controller is an always-online workstation, while controllees are added to or removed from the network as needs dictate. It is sensible for controllees to announce their presence to the controller when they appear on the network – otherwise the workstation would have to periodically poll the network to discover which controllees were currently available.

3. The command data sink reacts by creating socketF for reception of subsequent command data from this source on port2. As messages arrive from other sources, further sockets are created as necessary.

4. The connection is now established. The command data sink can receive command data on socketF. The command data source can transmit command data on socketE.

A.3 The Disconnection Protocol

In the operational delay line, it is an error condition for any program to break an established socket connection, even when this is deliberate, and it should be reported as such. Hence a disconnection can be handled in much the same way whether one of the parties deliberately disconnected or there was a system failure such as a network problem. The disconnection protocol is the same for all kinds of source and sink:

1. ProgramA ceases communication with programB by ceasing to write to and/or read from the corresponding socket. It then destroys the socket itself.

2. ProgramB notices communication with programA has stopped. It might do this by timing out if it was expecting data from programA, or by noticing that messages to programA fail to be sent, or by receiving a "hangup" signal from the socket (this might take several minutes).

3. ProgramB then stops transmitting via the socket in question and destroys that socket. The disconnection process is now complete.

B Commands & Data

This section gives details of the command and command data protocols outlined in Section 2.3 above.

B.1 Command/Data Lists

In the listings that follow, the types of command parameters and data values are indicated by means of the type codes understood by the Serialise library. A key to these type codes is given in Table 2.

B.1.1 Workstation to VME Metrology System

Each command applies to the trolleys specified as part of the command, by means of the "trolley mask". This is an integer word (transmitted as an array of unit length) where, if bit i is set, the command applies to trolley i. If bit i is *unset*, the state of trolley i remains unchanged.

Each "Trajectory$N$" data message applies to the trolley N specified in the message label.

The UTC in Trajectory$N$ shall be an integer number of seconds. The velocities in Trajectory$N$ shall be calculated by differencing the transmitted positions; they should not be the instantaneous velocities for the same times as the positions.

Command	**Parameters**	**Type**		**Comment**
Follow	Trolley mask	H[1]		Start OPD loop
Idle	Trolley mask	H[1]		Stop OPD loop, zero carriage velocity
Datum	Trolley mask	H[1]		Slew to datum, zero metrology
FringeTrackOn	Trolley mask	H[1]		Apply FT offsets
FringeTrackOff	Trolley mask	H[1]		Don't apply FT offsets
ResetFTOffset	Trolley mask	H[1]		Set total FT offset to zero
Data Label	**Values**	**Type**	**Rate**	**Comment**
Trajectory$N$	UTC, Time interval, Position valid flag, 10×position, 10×velocity	D[23]	1 Hz	at which to realise 1st position between points include intra-night offset

B.1.2 Workstation to Trolley Micros

Each command applies to the trolley whose micro it is sent to.

Command	**Parameters**	**Type**	**Comment**
DoNothing	–		For testing
SteeringOn	–		Steering servo on
SteeringOff	Steering position	D[1]	
TipTiltOn	–		Tip-tilt servo on
TipTiltOff	Tip, tilt positions	D[2]	
FocusPos	Position, timeout	D[2]	
FocusOffset	Offset, timeout	D[2]	
DirectSlew	Velocity	D[1]	To drive trolley when VME down
DirectSlewOff	Velocity	D[1]	Drive trolley but let VME override

B.1.3 Workstation to Shear Sensors

Each command applies to the shear sensor it is sent to.

Command	Parameters	Type	Comment
SetFiducial	X, Y	D[2]	Current fiducial returned in status
LogVideoOn	M	H[1]	Save one image in every M
LogVideoOff	–		Stop saving images

B.1.4 VME Metrology System to Trolley Micros

Each data message applies to the trolley whose micro it is sent to.

Data Label	Values	Type	Rate	Comment
Slew	Carriage velocity	D[1]	10 Hz	
Track	Velocity	D[1]	10 Hz	Velocity for feed forward
(Rate demand)	Cat's eye error signal		5 kHz	Over low-latency link

B.1.5 Shear Sensor to Trolley Micro

Each data message applies to the trolley whose micro it is sent to.

Data Label	Values	Type	Rate	Comment
TipTiltOffset	Tip, Tilt offsets	D[2]	30 Hz	Current position in telemetry

B.2 Command/Data Message Format

The message format is given in Table 3. We have assumed the use of DFB's "Serialise" library, to match what has been agreed for telemetry/status messages (see Sec. 2.5). The format makes use of the fact that serialise automatically encodes the data type of each message component, to allow the array of command parameters/data values to take the most appropriate data type (integer or floating point) for each command.

More details on serialise may be found in the serialise manual, entitled "Serialise Network Message Protocol". For convenience, the serialise type codes are reproduced in Table 2.

As defined, the message format can be used for both commands and data, but distinct identifier strings and independent message version numbers are used for the two applications.

C Telemetry & Status

This section gives details of the telemetry and status protocols outlined in Section 2.4 above.

Table 2: Type codes (format specifiers) supported by the serialise library. Please refer to the serialise manual for more details.

s	null-terminated string
i	32-bit integer
d	64-bit float
C	array of characters
B	array of 8-bit integers
H	array of 16-bit integers
I	array of 32-bit integers
L	array of 64-bit integers (not yet implemented)
F	array of 32-bit float
D	array of 64-bit float
(	start tuple (group)
)	end tuple (group)
o	pre-encoded object (serialising)/"any" object (deserialising)

Table 3: Command/data message format.

Type	Item	Description
(	Message Start	Mandatory for serialise
	Identifier	
s	"MRO_DL"	Common to all delay line messages
s	"CMD"/"DATA"	Identifies type of message
i	Message version	Incremented when command/data format changed
	Body	
s	Source Identifier	See Table 4
i	Tag	Incremented by source when command/data message sent
s	Command/data label	e.g. "SteeringOn", "TiptiltOffset"
H/I/L/F/D	1d parameter/value array	Omitted if command takes no parameters
)	Message End	Mandatory for serialise

Table 4: Sub-system identifiers used in delay line messages.

"SYSTEMn"	Workstation system controller for trolley n
"WKSTN"	Workstation (not trolley-specific)
"VME"	VME Metrology System
"TRLYn"	Trolley n micro
"SHEARn"	Shear sensor for trolley n
"FT"	Fringe tracker (if applicable)

C.1 Telemetry and Status Items

The thinking behind these choices is that the status messages should contain sufficient information for the Workstation to conclude whether any of the commands in Sec. B.1 has completed successfully. Note that the status message format only allows boolean and 64-bit floating point data types.

C.1.1 Workstation Items

The following items are transmitted over the loopback interface, so that they are logged etc.

The Workstation opens a separate socket connection (to itself over the loopback interface) for each trolley, hence the trolley number is not needed as a suffix to each item name.

Item	Sample Rate /Hz	Type	Comment
OpdFollow	10	Bool	
OpdIdle	10	Bool	
OpdDatum	10	Bool	
OpdDirectSlew	10	Bool	
PosEndLimit	10	Bool	
NegEndLimit	10	Bool	
Track	10	Bool	
TrackInSpec	10	Bool	Track and Error, Jitter in spec
FTrackOn	10	Bool	
FocusOn	10	Bool	
SteeringOn	10	Bool	
TipTiltOn	10	Bool	
FollowCurrent	10	Bool	Current trajectory in force
Pos	10	Float64	from VME
Error	10	Float64	from VME
Jitter	10	Float64	from VME
FTOffset	10	Float64	from VME
CarrVelDem	10	Float64	VelDem from trolley
CoarsePos	10	Float64	from trolley
FocusPos	10	Float64	from trolley
Roll	10	Float64	from trolley
SteeringPos	10	Float64	from trolley
ShearSigX	10	Float64	from shearcam
ShearSigY	10	Float64	from shearcam
TipTiltXPos	10	Float64	from shearcam
TipTiltYPos	10	Float64	from shearcam
HourAngleNow	10	Float64	Transmitted earlier
PosDemNow	10	Float64	
VelDemNow	10	Float64	
IntraNightOffset	10	Float64	
		Telemetry	
PosDem	10	Float64	Position demand
VelDem	10	Float64	Velocity demand
IntraNightOffset	10	Float64	
TxUtc	1	Float64	Transmission time
HourAngle	1	Float64	For sidereal trajectory

C.1.2 VME Metrology Items

Since the VME system deals with all trolleys, there are equivalent status and telemetry items for each trolley. In the list below, N stands for the number of the relevant trolley; status and telemetry for all trolleys are sent over a single socket connection.

Item	Sample Rate /Hz	Type	Comment
			Status
Idle$N$	10	Bool	i.e. obeying 'Follow off'
Track$N$	10	Bool	False if slewing
DatumSeek$N$	10	Bool	True until metrology zeroed
DatumFound$N$	10	Bool	Set after successful datum seek
FTrack$N$	10	Bool	True if applying o/s from Fringe Tracker
MetPcbHot$N$	10	Bool	True if Agilent board is overheating
MetSignalBN	10	Bool	Agilent "B" signal is present
MetSignalAN	10	Bool	Agilent "A" signal is present
MetError$N$	10	Bool	True if any Agilent board error flags set
MetSampling$N$	10	Bool	Hardware sample pending
MetInterpReset$N$	10	Bool	Interpolator reset line status
MetInterpUnlock$N$	10	Bool	Interpolator has lost lock
MetOverflow$N$	10	Bool	Hardware position counter overflowed
MetGlitch$N$	10	Bool	Out-of-bounds acceleration detected
MetNegVbN	10	Bool	No transitions within 12.8μs on "B" input
MetPosVbN	10	Bool	Two transitions within 26.7ns on "B" input
MetNegVaN	10	Bool	No transitions within 12.8μs on "A" input
MetPosVaN	10	Bool	Two transitions within 26.7ns on "A" input
Pos$N$	10	Float64	Instantaneous metrology value
Error$N$	10	Float64	Mean OPD error over 0.1s window
Jitter$N$	10	Float64	Peak-to-peak OPD error over 0.1s window
MetState$N$	10		Metrology state XXX define for Agilent/Zygo
FTOffset$N$	10	Float64	Total FT offset
DatumPos$N$	10	Float64	Metrology just prior to reset at datum
			Telemetry
InterpPos$N$	5000	Float64	Interpolated Workstation PosDem
Metrology$N$	5000	Float64	Metrology position
MetrolError$N$	5000	Float64	Position error
RateDem$N$	5000	Float64	Cat's-eye error signal as transmitted /volt
VelDem$N$	10	Float64	Carriage demand velocity
FTIncr$N$	<200	Float64	Incremental FT offset

C.1.3 Trolley Micro Items

Item	Sample Rate /Hz	Type	Comment
			Status
SteeringOn	10	Bool	
TipTiltOn	10	Bool	
FocusOn	10	Bool	
Idle	10	Bool	
Track	10	Bool	
DirectSlew	10	Bool	Obeying slew override from workstation
PosEndLimit	10	Bool	In positive limit
NegEndLimit	10	Bool	In negative limit
VelDem	10	Float64	Carriage demand velocity
SteeringPos	10	Float64	
Roll	10	Float64	
TiptiltXPos	10	Float64	
TiptiltYPos	10	Float64	
FocusPos	10	Float64	
Temp	10	Float64	Roving temperature
CoarsePos	10	Float64	Odometer reading
DiffPos	10	Float64	Differential position sensor
			Telemetry
CoilDrive	5000	Float32	Cat's-eye drive current
DiffPos	5000	Float32	Differential position
DiffVel	5000	Float32	Differential velocity
Loop1	5000	Float32	Output volts from RF link
Loop2	5000	Float32	Output volts from metrology loop-shaping stage of pre-amp
SteeringDem	10	Float32	Steering demand
MotorVel	100	Float32	Motor velocity
MotorDemI	100	Float32	Motor demand current
MotorI	100	Float32	Motor current
MotorPos	100	Float32	Motor position
CatsAccelX	5000	Float32	Cat's-eye acceleration in X
CatsAccelY	5000	Float32	Cat's-eye acceleration in Y
CarrAccelX	5000	Float32	Carriage acceleration in X
CarrAccelY	5000	Float32	Carriage acceleration in Y
VPri	100	Float32	Primary supply voltage
V+5	10	Float32	"+5V" actual voltage
V-5	10	Float32	"-5V" actual voltage
V+12	10	Float32	"+12V" actual voltage
V-12	10	Float32	"-12V" actual voltage

Continued on next page

Item	Sample Rate /Hz	Type	Comment
VStore	10	Float32	Onboard storage voltage
TFocus	10	Float32	Focus stage temp.
TPriCell	10	Float32	Primary mirror cell temp.
TCarrF	10	Float32	Carriage front temp.
TCarrR	10	Float32	Carriage rear temp.
RfSig	10	Float32	Low latency link signal strength

XXX power usage?

XXX Loop3 etc.?

C.1.4 Shear Sensor Items

Item	Sample Rate /Hz	Type	Comment
	Status		
FiducialX	30	Float64	
FiducialY	30	Float64	
ShearSigX	30	Float64	w.r.t. fiducial
ShearSigY	30	Float64	w.r.t. fiducial
XValid	30	Bool	
YValid	30	Bool	
LoggingOn	30	Bool	Shear logging on
	Telemetry		
ShearX	30	Float32	w.r.t. fiducial
ShearY	30	Float32	w.r.t. fiducial
ConfidenceX	30	Float32	
ConfidenceY	30	Float32	

C.2 Telemetry and Status Protocols

We refer to the Workstation as the "server", with the VME Metrology CPU, trolley micros, and shear sensor computers as "clients". The server-side software allows any number of clients to connect.

Clients send "chunks" of telemetry at 1 Hz (or faster if this is more convenient), and status messages at 10 Hz (or faster). Each telemetry chunk may contain many data samples. Multiple chunks of telemetry (each containing a different signal) may be concatenated into a single message.

Each status message contains a heterogeneous set of numerical and boolean values. In the simplest variation of the message format, these have a common timestamp. However, it is

permissible to concatenate several status units (each of which can contain multiple items) in a single message, each unit having an independent timestamp.

Messages are transmitted from client to server over a TCP/IP socket connection. The server listens on a pre-arranged TCP/IP port, and can accept connections from multiple clients to that port.

In the current implementation, when recording is active the server logs all telemetry and status received to a single file on disk. The log file format is described in Sec. D.

C.2.1 Telemetry Format Details

Each telemetry message contains separate identifier, header and data components. The identifier identifies the category (telemetry or status) and version of the message. The intention is that all delay line network messages have equivalent identifiers, encoded using serialise. The header contains sufficient information to allow decoding and interpretation of the data part that follows.

The format of a telemetry message is enumerated in Table 6. A key to the type codes may be found in Table 2.

For chunks of length 5000 samples, the "sample index" for consecutive chunks would be 0, 5000, 10000, ... This allows missing data to be identified. The sample index would normally be reset whenever the stream is reconfigured.

Heterogeneous telemetry information may be combined in a single message by joining multiple header/data units onto the identifier. In other words, given that the basic message enumerated in Table 6 has the format (I(H)D) (where I stands for the identifier, H for the header items and D for the data part), a concatenated message is constructed as (I(H)D(H)D...). Header/data units may appear in any order.

C.3 Extension to telemetry message format for shear camera video

A proposed modification of the telemetry message format that allows time-series of images to be transmitted to the Workstation is shown in Table 7.

The data array that follows the header and optional metadata can be any valid FITS image array, with any number of dimensions (though typically one – TIME – or 3 – X, Y, TIME), in column-major storage order w.r.t. the dimensions in the message header. The last (slowest) dimension is assumed to be the time dimension.

Colour images (which are not needed for the current monochrome shear cameras) could be transmitted as 4d arrays (X, Y, RGB, TIME).

The key-value metadata need only be transmitted in the first message in a sequence (with the same ConfigId).

Table 6: Current telemetry message format. Further header/data units may be added to the end of the message, as described in the text.

Type	Item	Description
(	Message Start	Mandatory for serialise
	Identifier	
s	"MRO_DL"	Common to all delay line messages
s	"TELE"	Identifies type of message
i	Message version	Incremented when this format changed
	Header	
(	Header Start	
s	Client Identifier	See Table 4
i	Client ConfigId	Incremented when client's set of streams changed or stream(s) reconfigured
i	Secondary Client Identifier	Synchronous streams share same value
i	Time offset /μs	Relative offset from other streams with same secondary client identifier
s	Stream Identifier	Label, e.g. "Rel_pos"
i	Nominal sample rate /Hz	
i	No. of samples in chunk	Not req'd to decode
s	Type code for data	"H"/"I"/"L"/"F"/"D". Not req'd to decode
s	Units	e.g. "mm"
i	Sample index	Index of chunk's 1st sample (see below)
d	UTC of chunk's 1st sample	In Unix Time. "Time offset" is included.
)	Header End	
	Data	
H/I/F/D	1d data array	Samples in time order
)	Message End	Mandatory for serialise

The metadata will be decoded, buffered, and written verbatim to the relevant HDU of FITS logfiles. This section of the message would generally be omitted for non-image telemetry, and for image telemetry would contain keywords conforming to the WCS FITS conventions that specify the world coordinates of the image pixels (for shear camera images this could be shear in X and Y with respect to the current fiducial, as a function of time). The display orientation should follow the FITS convention i.e. the image pixel (1,1) is displayed in the lower left corner, the X coordinate increases to the right in the image, and the Y coordinate increases in the upward direction.

Table 7: Proposed new version of telemetry message format that can be used to transmit image telemetry. Differences from the current format are highlighted. As with the current message format, further header/data units may be added to the end of the message as described in the text.

Type	Item	Description
(	Message Start	Mandatory for serialise
	Identifier	
s	"MRO_DL"	Common to all delay line messages
s	"TELE"	Identifies type of message
i	Message version	Incremented when this format changed
	Header	
(	Header Start	
s	Client Identifier	See Table 4
i	Client ConfigId	Incremented when client's set of streams changed or stream(s) reconfigured
i	Secondary Client Identifier	Synchronous streams share same value
i	Time offset /μs	Relative offset from other streams with same secondary client identifier
s	Stream Identifier	Label, e.g. "Rel_pos"
i	Nominal sample**/frame** rate /Hz	
I	**Chunk array dimensions**	See text
s	Type code for data	"H"/"I"/"L"/"F"/"D". Not req'd to decode
s	Units	
i	Sample**/frame** index	Index of chunk's 1st sample (see below)
d	UTC of chunk's 1st sample**/frame**	In Unix Time. "Time offset" is included.
)	Header End	
	Optional Metadata	
(	**Metadata Start**	
ss	**Keyword, Value**	
...		
)	**Metadata End**	
	Data	
H/I/F/D	1d data array	See text
)	Message End	Mandatory for serialise

C.3.1 Status Format Details

Each status message contains separate identifier, command acknowledgement, header and data components. The format of a status message is enumerated in Table 8.

Each message incorporates a structure for zero, one or more command acknowledgements, containing the following items:

- No. of commands received since previous status sent [Int]

For each command received, in order of receipt, the following items:

- Source of command [string]
- Command Tag (incremented at source each time any command sent) [Int]
- Parse flags [Boolean values encoded as Byte array]:
  - Command understood
  - Parameters in range (TRUE if no parameters)
  - Command will be (or has been) obeyed

If there are no numeric (boolean) status items, the NumLabel and NumUnits (BoolLabel) component(s) shall be an empty tuple (), and the NumVal (BoolVal) component shall be omitted (I am presuming serialise does not allow a zero-length array).

Multiple status units, each with an independent timestamp, may be concatenated to form a single message. Given that the basic message enumerated in Table 8 has the format (IA(H)D) (where I stands for the identifier, A for the command acknowledgements, H for the header items and D for the data part), a concatenated message is constructed as (IA(H)D(H)D...). Header/data units should appear in time order.

D Log File Format

The prototype and production versions of the workstation software have the capability to record telemetry and status information, outgoing commands and log/fault messages to disk files.

The format used for standalone recording is based on FITS binary tables, as Matlab has a built-in capability to read these (they can also be read into C, Python and IDL programs using third-party libraries), and the Cambridge team has relevant experience of writing FITS from C.

Table 8: Status message format. Further header/data units may be added to the end of the message, as described in the text.

Type	Item	Description
(	Message Start	Mandatory for serialise
	Identifier	
s	"MRO_DL"	Common to all delay line messages
s	"STAT"	Identifies type of message
i	Message version	Incremented when this format changed
	Acknowledgements	
i	No. commands received	since previous status sent
	For each command received:	
(	Acknowledgement start	
s	Source of last command	
i	Command Tag	Incremented at source when command sent
B[3]	Parse flags	See text
)	Acknowledgement end	
	⋮	
	Header	
(	Header Start	
s	Client Identifier	See Table 4
i	Client ConfigId	Incremented when set of status sent by client changes
i	No. log/faults to follow	
	For each log/fault:	
(	Log/fault start	
i	Log/fault type	See Table 9
i	Trolley mask	Same meaning as in commands
s	Human-readable message	For faults, starts with fault name followed by colon
)	Log/fault end	
	⋮	
(NBool×s)	BoolLabel	Labels for boolean status items, encoded as tuple of strings
(NNum×s)	NumLabel	Labels for numeric status items, encoded as tuple of strings
(NNum×s)	NumUnits	Units for numeric status items, encoded as tuple of strings
d	UTC timestamp for status	In Unix Time
)	Header End	
	Data	
B[NBool]	BoolVal	Boolean status items, encoded as Int8 array
D[NNum]	NumVal	Numeric status items, encoded as Float64 array
)	Message End	Mandatory for serialise

Table 9: Log/fault type codes used in status messages.

Value	Enumeration	Maps to MROI LogType	Maps to MROI LogLevel	Comment
1	DL_LOG_VERBOSE	INFO	FINER	
2	DL_LOG_DEBUG	INFO	FINE	
3	DL_LOG_CONFIG	INFO	CONFIG	related to configuration, especially at startup
4	DL_LOG_INFO	INFO	INFO	
5	DL_LOG_FAULT	FAULT	WARNING	
6	DL_LOG_SEVERE_FAULT	FAULT	SEVERE	

D.1 FITS Primer

FITS binary tables are part of the core FITS standard (which is in widespread use in astronomy), and provide a framework (meta-format) for storing heterogeneous data in a compact binary form. The latest version of the FITS standard is version 2.1b.

FITS files consist of any number of header/data units (HDUs), each of which represents an image, binary table, or ASCII table, together with associated metadata. Headers are always encoded in ASCII, and contain a set of keywords and associated values. Certain keywords have special meanings according to the FITS standard (for example they describe the structure of the data part of the HDU), but other application-specific keywords can be included. For historical reasons, the first HDU can only contain an image (which can be of zero size).

D.2 Structure of Telemetry/Status/Logs FITS File

Telemetry, status, command logs and log/fault messages (for all trolleys) are saved to the same file. A FITS log file as defined in this document may contain any number of DL_TELEMETRY (Sec. D.4), and DL_STATUS (Sec. D.5) tables, plus one DL_CMD table (Sec. D.6) and one DL_LOG table (Sec. D.7. Generally the tables appear in time order (with typically several tables of each type for the same time interval), with groups of status and telemetry tables interleaved, but any table ordering is valid.

Typically command logs and log/fault messages (perhaps filtered) are always recorded, whereas recording of status and telemetry — which has a higher data-rate — is activated by the user for a pre-set period (several tens or hundreds of seconds).

The table formats are not designed to accommodate changes in the set of telemetry/status items sent by a client during a recording. Clients should not do this anyway!

In the sections that follow, serialise typecodes (Table 2) are used to indicate the data types of keyword values/columns (FITS defines its own, different, typecodes).

D.3 Primary Header

The primary header (header of the first HDU) of the file contains no application-specific keywords, and no mandatory keywords that it would be useful to read.

D.4 DL_TELEMETRY Table

A FITS log file as defined in this document may contain any number of DL_TELEMETRY tables. The telemetry streams in each table are those that are time-synchronised with each other (as indicated by the secondary client identifiers and time offsets in the telemetry messages), and thus there will be at least one table per active client for each time interval.

Keywords defined by the FITS Standard

Keyword	Type	Value
DATE-OBS	s	Start UTC of table data as *yyyy-mm-dd*T*hh:mm:ss[.sss]*
DATE	s	UTC when file written as *yyyy-mm-dd*T*hh:mm:ss*
TTYPEn	s	Name of column (field) n (=stream identifier or "UTC")
TUNITn	s	Units for column n

Other Keywords

Keyword	Type	Value
TBL_VER	s	Version number of table definition
CLID	s	Client identifier for streams in this table
SEC_CLID	i	Secondary client identifier for streams in this table
REFSTRM	i	Number of column containing reference stream data
SMPRATEn	i	Nominal sample rate for column n /Hz
TIMOFFn	i	Time offset from reference stream for column n /μs
DATE-NOM	s	Start UTC of recording as *yyyy-mm-dd*T*hh:mm:ss[.sss]*
UTC-NOM	d	Start UTC of recording as Unix Time

Columns

Each table "cell" shall contain a one-dimensional *array* of telemetry samples. The array lengths for different columns are chosen such that the data in each row of the table spans

the same time interval for all columns. The data type of each column shall match that used in the telemetry messages for that stream.

A single column (of double precision type) with TTYPE="UTC" contains periodic timestamps: each cell in this column contains an array (perhaps of length 1) giving the UTC timestamps (in Unix Time, i.e. seconds since midnight Jan 1, 1970) for the so-called *reference stream*. One such timestamp comes from each telemetry network message. The column containing the data for the reference stream is identified by the REFSTRM keyword, and will be the most rapidly-sampled stream in the table. If several streams have equal-fastest sampling, one will be chosen arbitrarily as the reference stream to which the timestamps apply.

The table is normally structured such that each row corresponds to a single chunk of telemetry for the reference stream. For two synchronised streams A and B with sample rates of 5 kHz and 10 Hz respectively, the table might be arranged as follows (where (...) represents a single cell):

(1×UTC) (5000×A) (10×B)
(1×UTC) (5000×A) (10×B)
(1×UTC) (5000×A) (10×B)
etc.

Note that prior to April 2008, the workstation software wrote DL_TELEMETRY tables which did not fully conform to the above definition. Rather than write a client's telemetry streams to several DL_TELEMETRY tables, grouped by synchronicity (i.e. SEC_CLID), all streams were written to a single table. Hence the TIMOFFn values were not necessarily meaningful.

D.5 DL_STATUS Table

A FITS log file may contain any number of DL_STATUS tables, perhaps with different timespans. The status items from different clients shall be stored in separate tables. Both boolean and numeric values (from the same client) are stored in each table.

Keywords defined by the FITS Standard

Keyword	Type	Value
DATE-OBS	s	Start UTC of measurement as *yyyy-mm-dd*T*hh:mm:ss[.sss]*
DATE	s	UTC when file written as *yyyy-mm-dd*T*hh:mm:ss*
TTYPEn	s	Name of column (field) n (=status item label or "UTC")
TUNITn	s	Units for column n

Other Keywords

Keyword	Type	Value
TBL_VER	s	Version number of table definition
CLID	s	Client identifier for status items in this table
DATE-NOM	s	Start UTC of recording as *yyyy-mm-dd*T*hh:mm:ss[.sss]*
UTC-NOM	d	Start UTC of recording as Unix Time

Columns

The table shall have columns of FITS logical type with a single boolean status value per table cell, and columns of double precision type with a single numeric status value per table cell. The timestamps (Unix Time) for the status information are contained in a single double precision column with TTYPEn of "UTC".

If a client packages its status items for each interval in more than one message unit (in order to encode the epochs of measurement more precisely), there will be one FITS row per unit, and some of the values in the table will be NULL, encoded as per the FITS standard (zero-valued byte for logical columns, IEEE NULL for double precision columns).

Log/fault messages are written to a separate FITS table (Sec. D.7).

Command acknowledgements are stored in the columns listed below. If the number of acknowledgements in a status message exceeds the number of status units in the message, an extra table row shall be included for each further command. Besides the acknowledgement columns, other columns in these rows should contain the same values (including timestamp) repeated.

Column	Type	Value
ICMD	i	Index of command in interval since last status message
CMDSRC	s	Source of command
CMDTAG	i	Command tag
PFLAGS	FITS logical[3]	Parse flags

D.6 DL_CMD Table

This table is used to log commands sent by the Workstation only.

Keywords defined by the FITS Standard

Keyword	Type	Value
DATE-OBS	s	Start UTC of log as *yyyy-mm-dd*T*hh:mm:ss[.sss]*
DATE	s	UTC when file written as *yyyy-mm-dd*T*hh:mm:ss*
BLANK	i	Code for NULL in integer arrays

Other Keywords

Keyword	Type	Value
TBL_VER	s	Version number of table definition
CMDSRC	s	

Columns

Column	Type	Value
UTC	d	Timestamp (Unix Time)
DEST	s	Destination for command
CMDTAG	i	Command tag
CMD	s	Command label
IPAR	I[n]	Integer parameter values
FPAR	D[m]	Floating point parameter values

The dimensions n and m of the parameter arrays should be chosen to be sufficient for all possible commands. Array cells that are superfluous for a particular command shall contain NULL values encoded as per the FITS standard (for IPAR the value of BLANK in the header, for FPAR the IEEE NULL value).

D.7 DL_LOG Table

This table is used to record human-readable log and fault messages. These may be received from clients in status messages or generated within the workstation software.

Keywords defined by the FITS Standard

Keyword	Type	Value
DATE-OBS	s	Start UTC of messages as *yyyy-mm-dd*T*hh:mm:ss[.sss]*
DATE	s	UTC when file written as *yyyy-mm-dd*T*hh:mm:ss*

Other Keywords

Keyword	Type	Value
TBL_VER	s	Version number of table definition

Columns

Column	Type	Value
UTC	d	Timestamp (Unix Time)
CLID	s	Source of message
TYPE	s	Category of log/fault message (see Table 9)
TRLYMASK	FITS logical[10]	T for affected delay lines
TIME-OBS	s	UTC timestamp as *hh:mm:ss[.sss]*
MESSAGE	s	Human-readable message. For FAULT and SEVERE FAULT types, the message begins with the fault name followed by a colon.