MRO Delay Line

Network Message Protocols and
Telemetry/Status/Commands Log File Format

John Young

rev 0.17
18 November 2009

Cavendish Laboratory
Madingley Road
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	WKSTN status/telemetry names no longer have suffixes. All WKSTN items now listed. Corrections to TRLY 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

Objective

To propose network protocols for sending commands, status information and telemetry between the sub-systems of the prototype delay line control system. To identify which commands and telemetry/status items are sent to/from each sub-system. To propose a file format for archiving the telemetry and status information after it has arrived at the Workstation, together with the commands issued by the Workstation.

Scope of this document

This document fleshes out the architecture of the prototype control system (outlined in previous documents), by describing in detail the flow of information between the various computers in the prototype delay line system. However, it does not provide a complete description of the software that will be delivered.

1

We describe the internal messaging protocols designed to transmit this information. The protocols encompass both control of the prototype delay line, and routing of telemetry (for debugging) back to the Workstation.

We also describe a FITS-based file format for logging of commands, status, and telemetry by the Workstation – this will be used to debug the prototype system.

As well as being a working document of the Cambridge Delay Line Team, this document should also be useful to MRO in order to:

- Provide information relevant to MRO's future decisions about which aspects of the prototype system are to be re-used for MROI operations. This could be none, some or all of:

  - The basic architecture
  - The message formats
  - The file format
  - The C code that implements messaging and logging
  - The (mostly) C code for the entire prototype control system

- If (part of) the prototype control system is re-used, assist in the design of external interfaces to the delay line (e.g. from the Interferometer Control System and Fringe Tracker). This point is discussed in Appendix A.

- If the prototype control system is not re-used, inform the design of the the control system to be used for operations

This document is likely to be revised as the prototype control system is implemented.

Summary

The protocols proposed for communication between prototype delay line sub-systems are summarised in Table 1.

1   Design Aims

The architecture and implementation described in this document were arrived at with the following aims in mind:

- Provide the capability to record all control signals (hardware and software), for debugging the prototype delay line

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	WKSTN	10–30	System-level control, displayed, logged
Telemetry	All	WKSTN	1	Logged
Command	WKSTN	VME, SHEARn, TRLYn	Async.	Obey command
Command Data	WKSTN, VME, SHEARn, FT	VME, SHEARn, TRLYn	10-200	Close loop

- Facilitate possible re-use of the architecture and/or code by MRO, both for fault diagnosis over the lifetime of the interferometer and for observing

- 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 Phases 1 (initial tests) & 2 (as delivered to MRO) of the prototype system control software

2 Introduction

The architecture for the control system of the prototype delay line is described in the presentation given at MRO by JSY (INT-406-VEN-0006) and in the "Trolley Concept Description" (document identifier pending) delivered to MRO in January 2006.

The flow of information between the components of this 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

4. Telemetry

2.1 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.
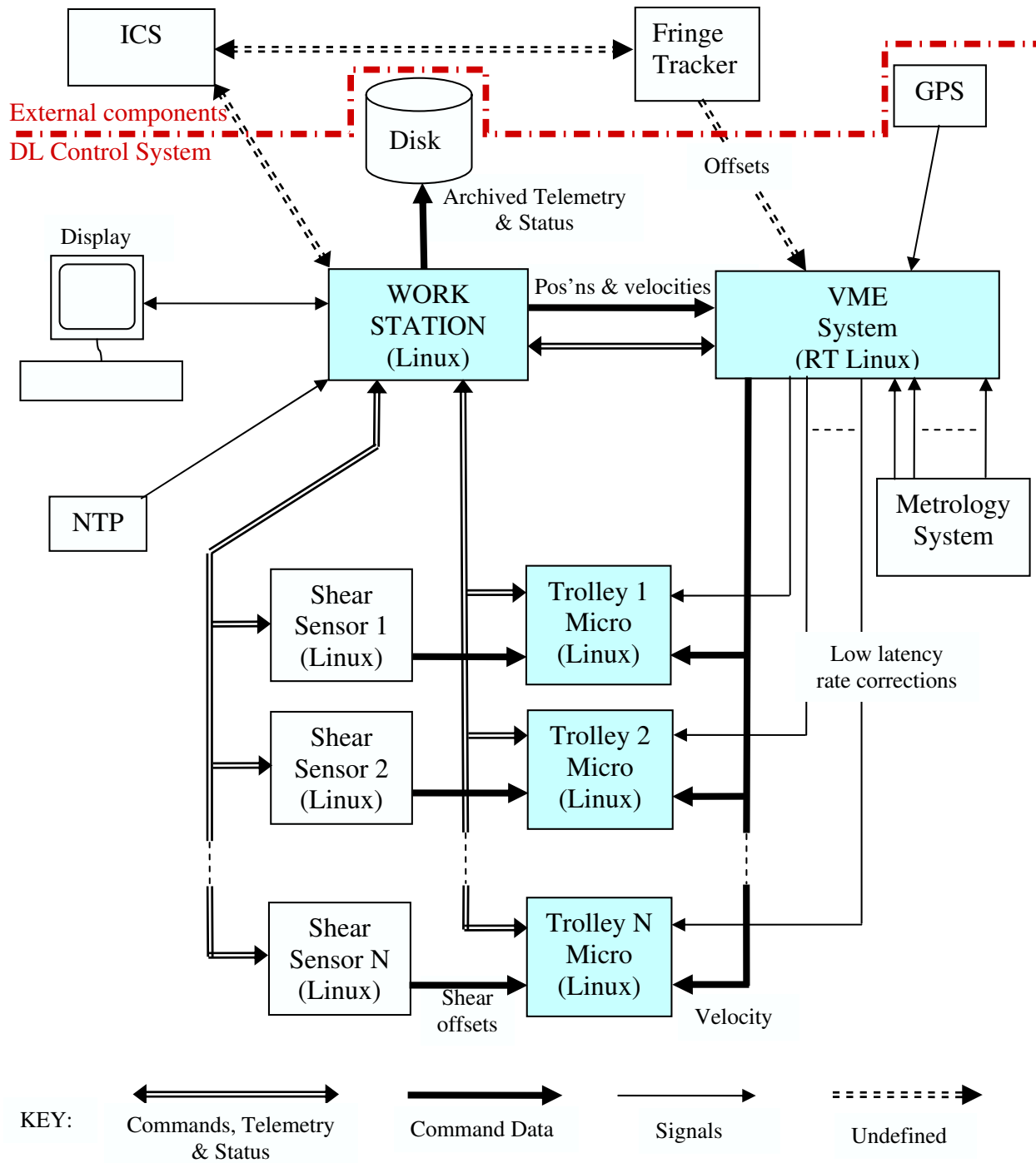
Figure 1: Information flow in the control system for the prototype delay line. Dashed lines (labelled "undefined") indicate where interfaces to MROI components external to the delay lines are envisaged, if the same architecture were to be used for operations. In this diagram, "signals" are control signals transmitted via interfaces other than Ethernet. These signals are not described any further in this document.

Commands and their corresponding acknowledgements will be logged in the same file as telemetry and status information (defined in the next section) — see Sec. 5. 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. 3.

2.2   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. 4.3. 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 ~ 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. 4.3.

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 will be logged to the same file as telemetry, so that the user can correlate the information when debugging the delay line.

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

The aim of this document is to propose self-describing message formats suitable for transmission of all commands/data (Sec. 3), plus status messages and telemetry streams (Sec. 4). A FITS-based file format for logging all of the communications is described in Sec. 5.

3 Commands & Data

In this section we distinguish between **data**, streams of information used to close servo loops; and **commands**, which are sent less regularly and for a variety of purposes. "Acknowledgements" to all commands are incorporated into the status messages sent to the Workstation. There is no explicit acknowledgement of data messages.

Each data item should also be transmitted from the originating sub-system to the Workstation as telemetry. The Workstation will transmit status and/or telemetry over a loopback connection to itself, so that the information is automatically buffered and archived, processed and/or displayed as appropriate.

3.1 Data Dropouts

If a stream of data ceases, this is an error condition. In general, the receiving sub-system will immediately flag the error in a status message, but attempt to project forward from previous data, in order to:

- Ameliorate the immediate effects of the data loss

- Minimise the time taken to recover once the data stream is restarted

However, this projection should be performed *for a few seconds at most*, because the projected data will become increasingly inaccurate, hence using them will be worse than assuming appropriate neutral values.

The bases for projection of the various data streams are expected to be:

Velocity from VME Constant velocity

Shear offsets Maintain last tip-tilt position

Note that the code that implements this projection is not intended to be elaborate.

3.2 Provisional 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.

3.2.1 Workstation to VME 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,	D[23]	1 Hz	at which to realise 1st position
	Time interval,			between points
	Position valid flag,			
	10×position,			include intra-night offset
	10×velocity			

3.2.2 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

3.2.3 Workstation to Shear Detectors

Each command applies to the shear detector 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

3.2.4 Fringe Tracker to VME System

A possible use of the command protocol defined in this document for the external interface to the Fringe Tracker would be as follows.

Each data message applies to *all trolleys*, and will include parameter values for all (some of which may be zero/not valid).

Data Label	Values	Type	Rate	Comment
FTOffset	Incremental offset	D[10]	<200 Hz	Zero if data invalid

3.2.5 VME 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

3.2.6 Shear Detector to Trolley Micro

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

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)

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

3.3   Command/Data Message Format

The proposed 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. 4.5). 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 will be used for the two applications.

C library functions to send and receive commands and data are available in the "dlmsg" library (which is documented separately).

Table 3: Command/data message format. Note that for commands, the source identifier will always be "WKSTN".

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.

"WKSTN"	Workstation
"VME"	VME System
"TRLYn"	Trolley n micro
"SHEARn"	Shear sensor for trolley n
"FT"	Fringe tracker (if applicable)

4 Telemetry & Status

4.1 Telemetry Characteristics

The common features of the various telemetry streams are:

- Typically 16-bit samples (with exceptions, e.g. each metrology datum has 36 bits), although these may need scaling and offsetting to convert to physical units.

  The telemetry formats should accommodate integers up to 32 bits and floating point values up to 64 bits.

- Regular sampling

- Different sampling rates for different streams (10 Hz to 5 kHz)

- Sampling synchronised with a local free-running clock; times of samples not directly tied to GPS time

  - Sampling will *not* be occasionally re-synchronised with GPS time
  - Some streams will automatically be synchronised with others (as they will be digitised on the same board) – desirable to encode this fact
  - However, with some (all?) ADC boards the sampling of the streams digitised by each board will be interleaved, as the board only has a single ADC device – desirable to encode time offsets

- Samples can be stamped according to NTP time of measurement, derived from GPS time. For higher sample rates, time-stamping will be every N samples

- Each sub-system generally sends the same set of telemetry items at all times

- Streams may stop and re-start, e.g. when a sub-system is reset

Note that images from the shear sensors will be logged directly by the shear sensor computers, rather than via the Workstation. Hence the protocols described in this document will not need to handle image data.

4.2 Status Characteristics

The common features of the various status messages are:

- Contain both boolean (true or false) and numerical (e.g. OPD jitter) status items

- Sent at ~10 Hz (30 Hz from the shear sensors)

- Each sub-system generally sends the same set of status items at all times; the Workstation chooses subsets to archive and display based on user input

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.

4.3 Telemetry and Status Items

The status items sent from each sub-system to the Workstation should include *at least* the items given below. 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. 3.2 has completed successfully. Note that the status message format only allows boolean and 64-bit floating point data types.

For completeness, we also list the telemetry items identified thus far.

4.3.1 Workstation Items

The following items will be transmitted over the loopback interface, to facilitate display and logging using the mechanisms provided for sub-system data.

The Workstation will open 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
MotorVel	10	Float64	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

4.3.2   VME 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
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 WKSTN: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

4.3.3   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	Demand velocity (currently MotorVel)

Item	Sample Rate /Hz	Type	Comment
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
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.?

4.3.4 Shear Detector 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	

4.4 Telemetry and Status Protocols

We refer to the Workstation as the "server", with the VME CPU, trolley micros, and shear sensor computers as "clients". The server-side software shall allow 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 initial implementation, the server will log all telemetry and status received to a single file on disk. An interface to Matlab will be written, allowing a simple Matlab script to extract the data corresponding to a particular telemetry stream/status item from this single file. A proposed log file format is described in Sec. 5.

Possible enhancements to the server-side software could include near-real-time display and analysis of telemetry, as well as selective archiving.

4.5 Message Formats

4.5.1 Rationale

Messages are encoded using DFB's "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. Serialise 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 format (really a flexible meta-format) described here adds another layer of self-description which labels each data item and provides ancillary information such as the units and timing of the measurement. Hence additional telemetry or status items can be added to the messages, and properly written receiving code will log/display the new items as appropriate *with no modifications to the code.*

4.5.2 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.

Once we have agreed the message format, the version number will be set to unity and each further change will require an increment of this number.

4.5.3 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 7.

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

Table 6: 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 seconds since the Unix epoch. "Time offset" is included.
)	Header End	
Data		
H/I/F/D	1d data array	Samples in time order
)	Message End	Mandatory for serialise

- 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 7 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.

4.5.4 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.

5 Log File Format

The proposed format 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 JSY has experience of calling the cfitsio library from C to write such files.

5.1 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).

5.2 Structure of Telemetry/Status/Commands 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

Table 7: 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 8
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 seconds since the Unix epoch
)	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 8: Log/fault type codes used in status messages.

Value	Meaning	Comment
1	VERBOSE	
2	DEBUG	
3	CONFIG	related to configuration, especially at startup
4	INFO	
5	FAULT	
6	SEVERE FAULT	

DL_TELEMETRY (Sec. 5.4), and DL_STATUS (Sec. 5.5) tables, plus one DL_CMD table (Sec. 5.6) and one DL_LOG table (Sec. 5.7. Generally the tables will 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).

5.3   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.

5.4   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 will normally be 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.

5.5 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).

24

Log/fault messages are written to a separate FITS table (Sec. 5.7).

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

5.6 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	"WKSTN"

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).

5.7 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 8)
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.

A   Possibilities for interfacing the delay line control system to the Interferometer Control System

In this section we make some suggestions as to how a delay line control system with the internal communications architecture described in this document could be interfaced with the top-level Interferometer Control System (ICS). We are not necessarily advocating either of the approaches described here; rather they are presented as a possible starting point for discussions.

We outline two possible ways of interfacing the delay line control system (DLCS) to the ICS. Adoption of either strategy would allow the majority of the delivered code to be reused. Of course, other approaches have their own advantages, which would need to be weighed against any advantages of re-using Cambridge code.

1. "Opaque approach": Treat delay line as "black box", Workstation communicates with ICS via some protocol to be defined.

2. "Transparent approach": Workstation forwards commands from ICS, and forwards status and telemetry to ICS, using the protocols defined here.

A.1   Opaque Approach

This strategy is the equivalent to that being adopted for the Unit Telescopes. An interface layer, running on the Workstation, would be added to the DLCS. System-level commands would be sent from the ICS, and corresponding status information (as defined in Section 2.2) transmitted in the reverse direction. Communication between the ICS and the DLCS interface layer would involve a different protocol (yet to be defined) to those described in this document.

Telemetry could either be transmitted to the ICS, or archived on the Workstation. In the latter case the tools provided with the DLCS would be used to inspect telemetry, and hence debug any problems with the delay line.

A.2   Transparent Approach

In this approach the ICS would take over some of the functionality of the Workstation (at least the provision of the system-level intelligence that coordinates actions of the delay line sub-systems). The Workstation would forward command messages from the ICS to the appropriate delay line sub-systems, and forward status and (perhaps) telemetry messages originating from sub-systems to the ICS. The protocols defined in this document could be used with minimal changes.

A variation on this strategy would be to allow the ICS to communicate with delay line sub-systems directly.