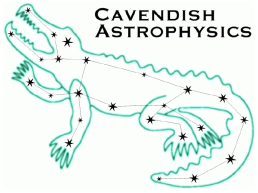
MRO Delay Line

Control Software Architecture

The Cambridge Delay Line Team

rev 0.1
4 July 2007



Cavendish Laboratory
Madingley Road
Cambridge CB3 0HE
UK

Change Record

Revision	Date	Authors	Changes
0.1	2007-07-04	JSY	Initial version

Objective

To describe the overall architecture of the control software for the prototype MROI delay line.

Scope of this document

This document provides a high-level description of the control software for the prototype delay line. We describe the sub-systems which comprise the distributed control system and their roles within the system. The protocols used for inter-subsystem communication over Ethernet (encompassing both control of the delay line and routing of diagnostic telemetry to a central server) are described in detail. We also describe a FITS-based file format for logging of sub-system commands, status, and telemetry.

More detailed functional descriptions of the software for individual sub-systems may be found in the following documents:

- Workstation Software Functional Description
- VME Software Functional Description
- Trolley Software Functional Description
- Shear Camera Software Functional Description

1 Introduction

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

- A "workstation" PC (shared between all trolleys) to act as a supervisor, and provide a user interface for testing the delay line and interrogating delay line telemetry.
- A VME-bus CPU (shared between all trolleys) 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 actions of these computers are coordinated by means of a custom network messaging protocol. This protocol defines several categories of message.

1. Commands:
   Sent by the Workstation to sub-systems in response to user input
2. (Command) Data:
   Information needed in real-time to close the servo loops described in Sec. 2
3. Status (transmitted from sub-system to the Workstation):
   - Subsystem state information
   - Delay line performance metrics
   - Information to display for the user
   - Command acknowledgements
4. Telemetry:
   Diagnostic information transmitted from sub-systems to the Workstation

1.1 Platforms

The VME system is an Intel?? processor VME-bus system running the QNX operating system (see "Metrology System and VME Hardware Design Description"). The other delay line control computers all run the GNU/Linux operating system (Linux kernel version 2.6.11 or 2.6.20). The trolley micro is a PC104-bus system with an ARM-compatible processor (see "Trolley Electronics Design Description"), the other Linux systems having standard Intel x86 architectures.

The prototype delay line code is written in ANSI C (C99), except for the code for visualisation/analysis of recorded telemetry, which is written in Matlab script (version 7). Where appropriate (e.g. for much of the Workstation code and some libraries), we have written object-oriented code in C (in some cases making use of the GLib Object System).

2 Overview of Control Loops

We now outline the servo loops involved in the operation of the trolley, and describe the roles played by the Workstation, VME, trolley, and shear camera software in closing these

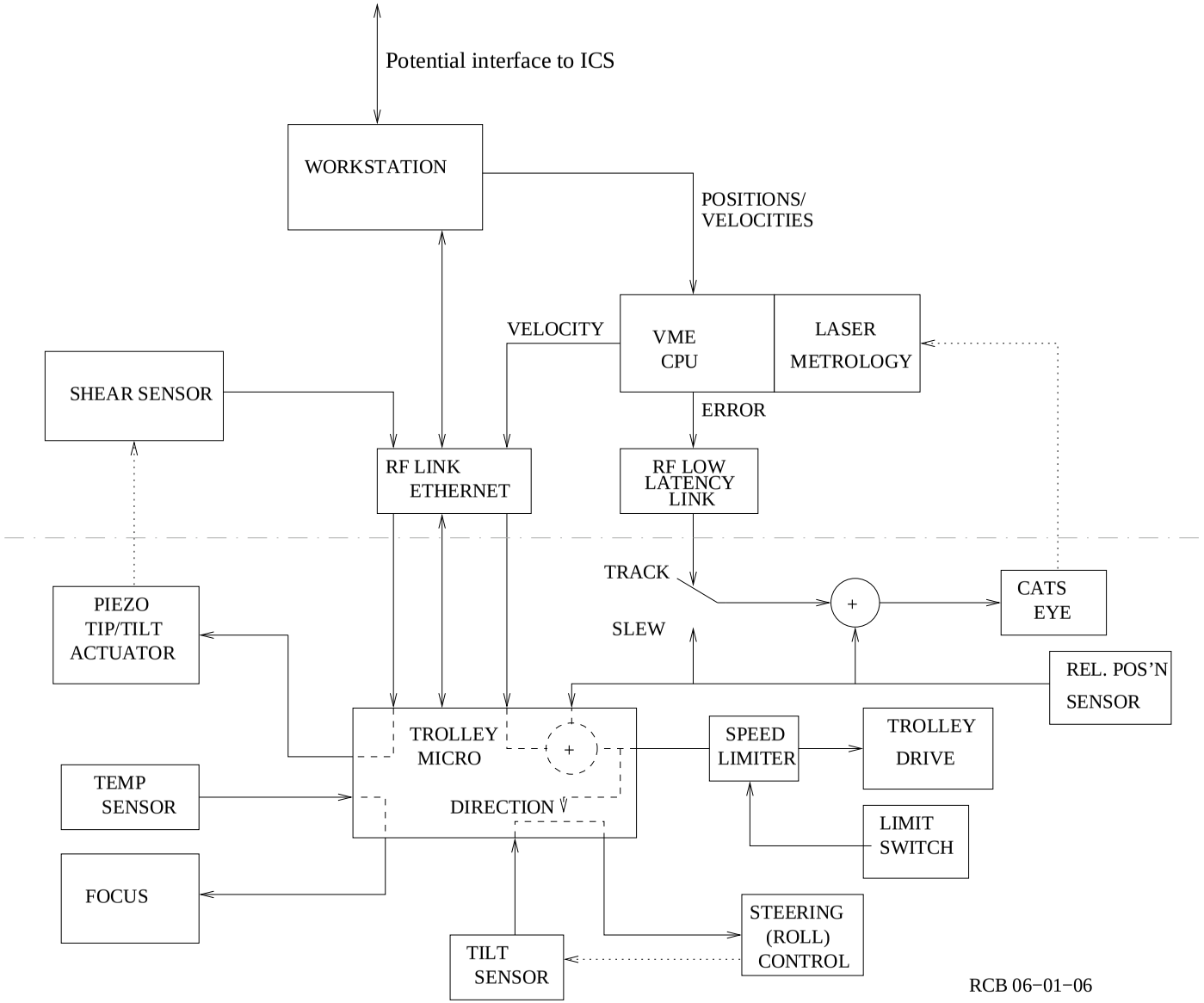


Figure 1: Overview of the control system for the prototype trolley, showing the origin of the signals used to control the active elements on the trolley. Components in the lower half of the diagram (except part of the speed limit switch) are physically located on the trolley. Each dotted line is drawn between an active element and a sensor whose output is *indirectly* affected by actuating the element. Dashed lines indicate connections made in software running on the onboard trolley micro. The "rel. pos'n sensor" measures the relative position of the Cat's-eye with respect to the carriage.

loops. Cross-references are provided to the more detailed descriptions in other documents.

The servo loops are shown conceptually in Figure 1, which includes both the loops entirely contained on the trolley and those relying on signals from external components, transmitted to the trolley via the RF data links.

2.1 Focus Control

This loop is only activated occasionally to perform axial translation of the Cat's-eye secondary mirror, in order to compensate thermally-induced changes in the length of the Cat's-eye tube. The loop is closed around an encoder mounted on the secondary stage under control of the trolley micro.

2.2 Steering

This is really a misnomer. The trolley is not being steered as it can only move axially along the pipe, so this is really roll control.

Mostly this is achieved by having the trolley centre of gravity below the centre line. This is a robust control – it is then almost impossible to capsize a trolley – but it is not very accurate, and accuracy is needed for two reasons. Firstly the axes of the secondary tip/tilt stage have to be kept matched to the beam shear sensor, and secondly the trolley wheels have to follow relatively narrow tracks to cross the pipe joints where the surfaces are aligned.

For accurate roll control an active trim system is included on the trolley. An electronic tilt sensor measures the roll angle of the trolley. This is digitised and read by the onboard micro, which controls a servo motor to adjust the tracking of the un-powered rear wheel to correct any error. This loop also needs an input from the drive motor controller as the sense of the correction required depends on the direction of travel. This low frequency loop is entirely contained on the trolley, and is described in the document "Trolley Electronics Design Description".

2.3 Shear Control (Secondary Tip/tilt)

This is an "always-on" low frequency loop closed via the Ethenet RF link and onboard micro. The shear sensor in the BCA uses a small fraction of the metrology laser light to measure any shear, and sends correction signals to the tip/tilt stage in the secondary mount on the Cat's-eye. This loop is described in the document "Shear Camera Software Functional Description".

2.4 Optical Path Delay (Cat's-eye and Carriage Control)

There are two main modes of operation for these loops. In either case the position of the Cat's-eye is measured by the laser metrology system, and the relative position of the Cat's-eye and the carriage is measured by an onboard sensor. However, the measurements are used in somewhat different ways in the two modes.

The OPD loop is described in more detail in XXX.

2.4.1 Tracking

This is the most critical mode, used when recording science data, and involves two stages to the servo loop:-

Cat's-eye The Cat's-eye position is measured by the laser metrology system and compared (by the VME CPU) with the current demanded position (interpolated from positions sent slightly in advance from the Workstation – see Sec. B.1.1). The resulting error signal is sent via the dedicated low latency RF link to the trolley (bypassing the onboard micro), where it is amplified and used directly to drive the Cat's-eye voice coil actuator.

Two small additional signals derived from the Cat's-eye/carriage relative position sensor are also applied, in hardware (we call this "trolley management"). The first one is a proportional term used to reduce the effective stiffness of the "wishbone" leg flexure pivots. The second is a velocity term to offset dynamic drag caused by the voice coil. In addition electronic travel limits for the Cat's-eye are set by the relative position sensor; these clamp the voice coil drive signal if the limits are exceeded.

Carriage The carriage drive motor is directly controlled by the relative position sensor to keep the carriage centred under the Cat's-eye so the "wishbone" legs are upright. This is important for best noise rejection. A demanded velocity term sent via the Ethernet RF link (Sec B.1.5) is added to reduce tracking error.

2.4.2 Slewing

This mode is used for rapid re-positioning of the trolley. The Cat's-eye voice coil is driven directly by the relative position sensor to hold it fixed relative to the carriage. The drive motor can then be ramped up to full speed until the desired position is approached and then slowed before reverting to tracking mode. This is accomplished by the VME CPU reading the laser metrology and sending velocities to the trolley micro (Sec B.1.5).

3 Control System Information Flow

The flow of information between the components of the control system is shown in Figure 2. Many of the signals are transmitted as messages over Ethernet; we categorise these messages as follows:

1. Commands
2. (Command) Data:
   information needed in real-time to close servo loops
3. Status (includes command acknowledgements)
4. Telemetry

The messaging protocol described in the following sections is summarised in Table 1.

3.1 Design Aims

The messaging protocol was designed with the following aims in mind:

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

3.2 Connection Protocol

The protocol for establishing communication between the various computers in 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).

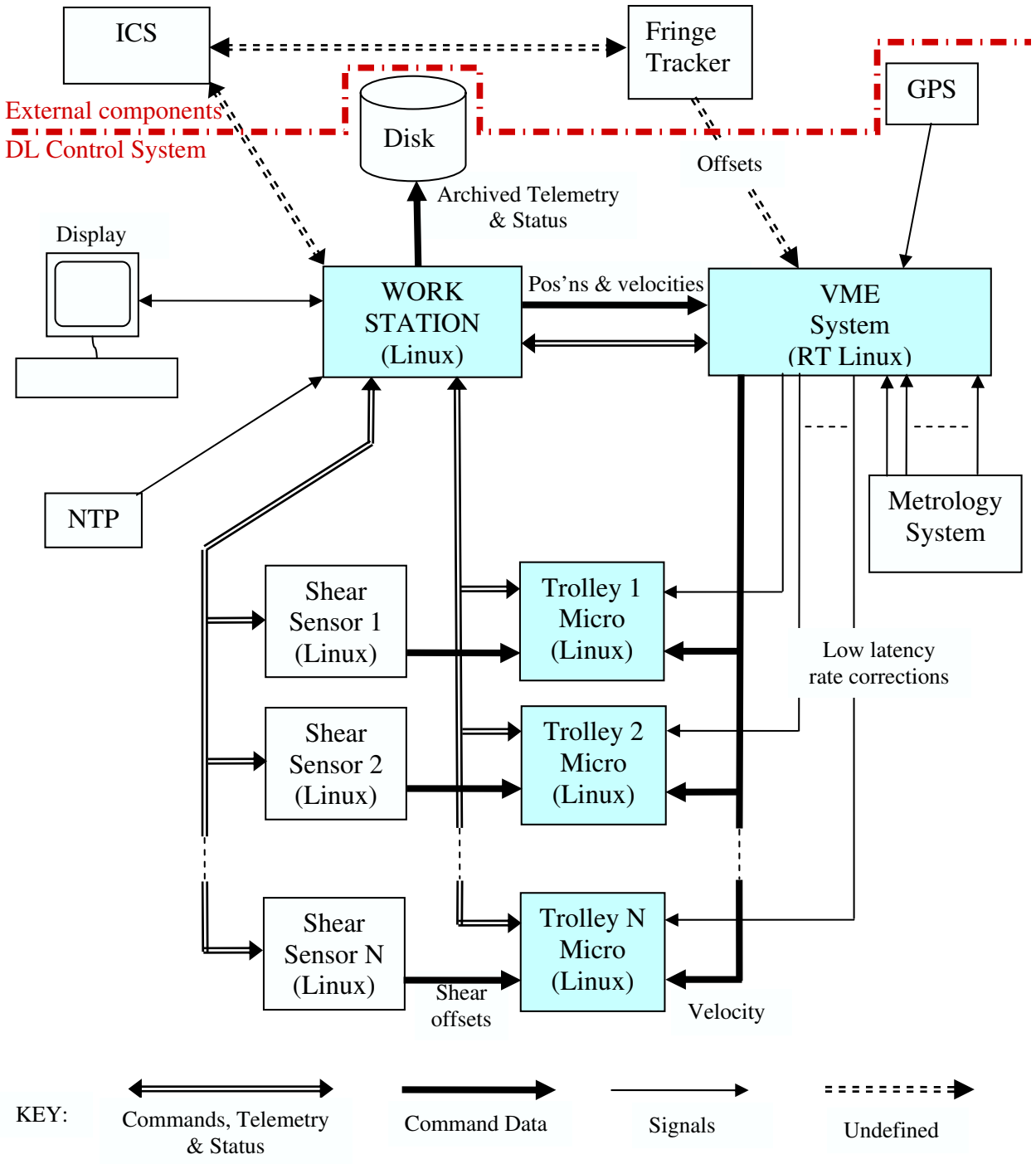


Figure 2: Information flow in the control system for the prototype delay line. Dashed lines (labelled "undefined") indicate where interfaces to MROI components external to the delay lines are envisaged, if the same architecture were to be used for operations. In this diagram, "signals" are control signals transmitted via interfaces other than Ethernet. These signals are not described any further in this document.

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

3.3 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 (see Sec. 2). 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 in 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.

3.4 Telemetry & Status

We define telemetry to consist of "measurements" made primarily for diagnosing problems with 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 implemented thus far are given in Sec. C.1. 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 optional 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.

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 (see Sec. D), so that the user can correlate the information when debugging the delay line.

In the prototype control system architecture, status and telemetry messages are only sent to the Workstation (see Figure 2). The Workstation contains all of the system-level intelligence necessary to control the delay line.

3.5 Message Formats

Inter-subsystem messages are encoded using the locally-written "Serialise" library , and transmitted using sockets over TCP/IP. The Serialise library (described in its manual "Serialise Network Message Format") implements a compact yet flexible binary representation. 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 formats defined in the appendix add another layer of self-description which labels each data item and, for telemetry and status, provides meta-data such as the units and timing of each measurement.

A C library, "dlmsg", has been written to facilitate composing, transmitting, receiving and decoding delay line network messages (see manual "dlmsg Library"), and is used by all of the prototype delay line control software components.

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 values have not yet been defined but 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()`) 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 3.3 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 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.

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 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

B.1.4 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

B.1.5 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

B.1.6 Shear Detector 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 command/data message format is given in Table 3. The format makes use of the fact that the Serialise library 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.

The same message format is used for both commands and data (reducing the amount of coding), but distinct identifier strings and independent message version numbers are used for the two applications.

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. 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

C Telemetry & Status

This section gives details of the telemetry and status protocols outlined in Section 3.4 above.

C.1 Telemetry and Status Items

The thinking behind the choice of status items 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.

Since the Workstation deals with all trolleys, there are equivalent items for each trolley. In the list below, N stands for the number of the relevant trolley.

Item	Sample Rate /Hz	Type	Comment
	Telemetry		
PosDem$N$	10	Float64	Position demand
VelDem$N$	10	Float64	Velocity demand

C.1.2 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.

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
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	Std. dev. of OPD error over 0.1s window
MetState$N$	10		Metrology state XXX define
FTOffset$N$	10	Float64	Total FT offset
			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

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
(Limit switches)	10		XXX define
VelDem	10	Float64	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
			Telemetry

Continued on next page

Item	Sample Rate /Hz	Type	Comment
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.?

C.1.4 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	

C.2 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 current implementation, the server logs all telemetry and status received to a single file on disk. The log file format is described in Sec. D.

C.3 Telemetry and Status Message Formats

Telemetry and status messages are encoded using the "Serialise" library , and transmitted using sockets over TCP/IP.

The message formats (really flexible meta-formats) described here add another layer of self-description to that provided by serialise. Each data item is accompanied by a label and by meta-data 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.*

C.3.1 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.

Each future change to this message format will require an increment of the version number.

C.3.2 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:

- 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]

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)

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

- 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.

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.

D Log File Format

The format is based on FITS binary tables. Matlab (used to implement the data analysis software) has a built-in capability to read these. They can also be read into C, Python and IDL programs using third-party libraries.

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/Commands FITS File

Telemetry, status and command logs (for all trolleys) are saved to the same file, although status and/or telemetry may be omitted, under the user's control. A FITS log file as defined in this document may contain any number of DL_TELEMETRY (Sec. D.4), and

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	Error severity	See Table 8
s	Error message	Empty string if no error
(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: Error severity codes used in status messages.

Value	Meaning	Comment
0	No error	
1	Warning	
2	Error	
3	Fatal	Client will cease execution

DL_STATUS (Sec. D.5) tables, plus one DL_CMD table (Sec. D.6). 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.

The timespan *of the file* may be a pre-set period of a few seconds, or else recording may continue until stopped by the user. If the set of telemetry/status items sent by any client changes (as indicated by the relevant ConfigId in the network messages), a new telemetry/status table (with different columns) is started and the previously active one is closed (this type of client behaviour is discouraged).

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 measurement as *yyyy-mm-dd*T*hh:mm:ss[.sss]*
DATE	s	UTC when file written as *yyyy-mm-dd*T*hh:mm:ss[.sss]*
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

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 seconds *relative to DATE-OBS*) 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 is 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.

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[.sss]*
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

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 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).

The following columns are also present, to store error information:

Column	Type	Value
SEVERITY	i	Error severity
ERRORMSG	s	Error message

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	B[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[.sss]*
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
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).