TITLE : TCS Server Software Design Document

DOCUMENT NUMBER : 1741AE0004
ISSUE: A

SYNOPSIS : This document describes the software design of the TCS Server (TCSS) software of the TCS.

KEYWORDS : Telescope Control System, Astrometry, Pointing

PREPARED BY : Gerhard Swart

APPROVED : Gerhard Swart
SALT Systems Engineer

Kobus Meiring
SALT Project Manager Engineer

DATE : 2 December 2002
This issue is only valid when the above signatures are present.
Printed: 16/09/03 13:48
# ACRONYMS AND ABBREVIATIONS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATP</td>
<td>Acceptance Test Procedure</td>
</tr>
<tr>
<td>ATR</td>
<td>Acceptance Test Report</td>
</tr>
<tr>
<td>BMS</td>
<td>Building Management System</td>
</tr>
<tr>
<td>CDR</td>
<td>Critical Design Review</td>
</tr>
<tr>
<td>CIN</td>
<td>Code Interface Node (a LabVIEW function to interface to other SW)</td>
</tr>
<tr>
<td>ELS</td>
<td>Event Logger Software</td>
</tr>
<tr>
<td>EDS</td>
<td>Environmental Display System</td>
</tr>
<tr>
<td>HET</td>
<td>Hobby-Eberly Telescope</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output (Device)</td>
</tr>
<tr>
<td>ICD</td>
<td>Interface Control Dossier</td>
</tr>
<tr>
<td>MMI</td>
<td>Man-Machine Interface</td>
</tr>
<tr>
<td>OPT</td>
<td>Operational Planning Tool</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PDR</td>
<td>Preliminary Design Review</td>
</tr>
<tr>
<td>PFIS</td>
<td>Prime Focus Imaging Spectrograph</td>
</tr>
<tr>
<td>PI</td>
<td>Principal Investigator (Astronomer)</td>
</tr>
<tr>
<td>PIPT</td>
<td>PI Planning Tool</td>
</tr>
<tr>
<td>PLC</td>
<td>Programmable-Logic Controller</td>
</tr>
<tr>
<td>PMAS</td>
<td>Primary Mirror Alignment System</td>
</tr>
<tr>
<td>SA</td>
<td>SALT Astronomer</td>
</tr>
<tr>
<td>SALRT</td>
<td>Southern African Large Telescope</td>
</tr>
<tr>
<td>SAMMI</td>
<td>SA Machine Interface</td>
</tr>
<tr>
<td>SC</td>
<td>Software Component (e.g. part fo the TCSS)</td>
</tr>
<tr>
<td>SCAM</td>
<td>Salticam (Acquisition camera)</td>
</tr>
<tr>
<td>SCL</td>
<td>SALT Command Language (sent to TCSS)</td>
</tr>
<tr>
<td>SDB</td>
<td>Science Database</td>
</tr>
<tr>
<td>SDD</td>
<td>Software Design Document</td>
</tr>
<tr>
<td>SDP</td>
<td>Software Development Plan</td>
</tr>
<tr>
<td>SI</td>
<td>Software Item (the TCSS is a Software Item)</td>
</tr>
<tr>
<td>SO</td>
<td>SALT Operator</td>
</tr>
<tr>
<td>SOMMI</td>
<td>SO Machine Interface</td>
</tr>
<tr>
<td>SRS</td>
<td>Software Requirement Specification</td>
</tr>
<tr>
<td>STARCAT</td>
<td>Object Catalogue</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>TBC</td>
<td>To Be Confirmed</td>
</tr>
<tr>
<td>TBD</td>
<td>To Be Determined</td>
</tr>
<tr>
<td>TCS</td>
<td>Telescope Control System</td>
</tr>
<tr>
<td>TCSS</td>
<td>TCS Server</td>
</tr>
<tr>
<td>TPM</td>
<td>Telescope Pointing Machine (software for Astrometric Pointing)</td>
</tr>
<tr>
<td>VI</td>
<td>Virtual Instrument (LabVIEW function)</td>
</tr>
<tr>
<td>WEB</td>
<td>SALT web-server</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

1 Scope .................................................................................................................. 5
2 Referenced Documents .......................................................................................... 6
3 Software Context .................................................................................................... 6
   3.1 Software external interfaces ............................................................................ 6
   3.2 Development environment ............................................................................ 6
   3.3 Hardware platform ....................................................................................... 6
   3.4 TCSS commands to subsystems .................................................................... 6
   3.5 TCSS commands from SOMMI, SAMMI, Science Instruments ..................... 9
4 High-level flow diagrams and description ........................................................... 9
   4.1 TCSS modes .................................................................................................... 9
   4.2 TCSS Kernel .................................................................................................. 9
   4.3 TCSS MMI ..................................................................................................... 10
   4.4 TCSS Software Tools ................................................................................... 11
5 TCSS Kernel Software Design ............................................................................. 12
   5.1 Top-level design of the TCSS Kernel ............................................................ 12
      5.1.1 Main functional flow .............................................................................. 12
      5.1.2 Command Queues ............................................................................... 13
      5.1.3 Global Database .................................................................................. 13
      5.1.4 Error Handling .................................................................................... 15
      5.1.5 Initialisation ......................................................................................... 15
      5.1.6 Shutdown .............................................................................................. 16
   5.2 TCSS-hi ......................................................................................................... 16
      5.2.1 Monitor TCSS status ............................................................................ 16
      5.2.2 Determine TCSS mode ......................................................................... 17
      5.2.3 Execute TCSS actions ......................................................................... 18
   5.3 Command Interpreter .................................................................................... 18
   5.4 Astrometric Pointing Model .......................................................................... 20
   5.5 Geometric Pointing Model ............................................................................ 20
      5.5.1 Overview .............................................................................................. 20
      5.5.2 Accuracy requirements ......................................................................... 21
      5.5.3 Internal Interface to Geometric Pointing VI ......................................... 22
      5.5.4 Geometric Pointing model design ......................................................... 22
      5.5.5 Pointing Interface to other subsystems .................................................. 26
   5.6 TCSS-10 ........................................................................................................ 27
      5.6.1 Subsystem Controllers .......................................................................... 27
         5.6.1.1 Overview ....................................................................................... 27
         5.6.1.2 Top-level VI..................................................................................... 27
         5.6.1.3 Receive Subsystem Data ................................................................ 27
         5.6.1.4 Transmit Subsystem data ............................................................... 28
      5.6.2 Operator Interface Controllers ................................................................. 29
      5.6.3 Instrument Controllers ............................................................................ 29
      5.6.4 Local MMI Controller ............................................................................. 29

Appendix A: Definition and Management of TCSS commands ................................ 29
Appendix B: Provisional list of TCSS high-level commands .................................... 33
1 Scope

This document defines the design for the TCS Server Software item of the SALT Telescope Control System (TCS). The requirements for this software are defined in the TCSS Software Requirements Specification, document number 1741AS0001. The TCSS comprises two separate software applications that need to work together: the TCSS Kernel software (running on the TCS Server PC) and the TCSS MMI software (running remotely on any PC, but normally at least on the SO Workstation). The TCSS software is supported by the TCSS Software Tools which are used to managed changes to the software. The term “TCSS Software” or TCSS are used interchangeably in this document to refer to the TCSS Kernel software.

The purpose of this software is:

• Co-ordination of the initialisation of each SALT subsystem
• Performing the astrometric and geometric pointing calculations to point the telescope
• Controlling each subsystem’s mode and operation to achieve a co-ordinated execution of the operator’s commands
• Monitoring the safety of the telescope and preventing unsafe operation as appropriate
• Determination of the required Tracker trajectory to follow an object
• The provision of an operator interface for configuring and monitoring the TCSS

The major factors that have influenced the design of the software are the following:

• The TCSS is the “brain” of the telescope, providing the co-ordinated operation of the telescope parts, implying:
  • Its detailed design is intricately woven with the designs of each telescope subsystem
  • It will probably be the most changed piece of software on the telescope
  • Unreliability of this software would lead directly to telescope downtime and inefficiency
• The development and integration of the TCSS will be phased to optimise the development effort and allow maximum flexibility when changing subsystem interfaces. This implies:
  • The software must be modular, allowing different subsystems to be integrated piecemeal
  • The software structure and overall design must be sound, not degrading as additional functionality is added
  • Maximum use must be made of configuration files and tools to prevent erroneous software operation
• The functions performed by the TCSS are complex interactions between subsystems – this should define the required skills of software maintainers, not the complexity of the software itself.
• The development schedule is tight.

The TCSS Kernel software implements the primary functions of the TCSS and is the main subject of this document. The TCSS MMI and the TCSS Software Tools are described briefly in this document, but are subservient to the final implementation of the TCSS Kernel software.

The functional flow and design of the software is shown primarily using LabVIEW diagrams. It is assumed that the reader is conversant with this representation.

The term “thread” is used to refer to separate, independent, asynchronous executing LabVIEW loops.

This issue of the SDD has only preliminary information regarding the TCSS state machines. A significant effort is still required to optimise the defined concepts to make the software easily modifiable with changing subsystem interfaces and interactions without sacrificing implementation simplicity.
2 Referenced Documents

The following documents are referenced in this design document and are applicable to the extent mentioned herein.

- 1000AB0044 SALT LabVIEW Coding Standard
- 1000AS0040 SALT Operational Requirements
- 1700BP0009 TCS Software Development Plan
- 1700AS0001 TCS Specification
- 1741AS0001 TCSS Software Requirements Specification
- 1000AS0049 SALT Data Interface Control Dossier
- 1741AA0002 Astrometric Pointing Model
- 1741AA0003 Geometric Pointing Model
- 1000AA0058 Geometric Model and Error Corrections
- 1000AS0031 SALT Axes and Calibration definition
- 1000AA0058 Geometric Model and Error Corrections

3 Software Context

3.1 Software external interfaces.

Figure 1 shows the software external interfaces, as defined in the TCSS Specification. This diagram is shown for information purposes only, being superseded by the SALT Data ICD referenced in section 2 and sections 4 and 5 of this document. Table 1 provided a preliminary list of data items flowing via these interfaces. The Event Logging System (ELS) has the capability to monitor all the information shown.

3.2 Development environment

The TCSS will be developed in LabVIEW 6.1 for Windows, in compliance with the SALT Software Standard. Certain core modules may be developed in C, being called by the LabVIEW software. These modules are either already developed in C or have other real constraints requiring this.

3.3 Hardware platform

The TCSS software will run on the TCSS PC together with the LabVIEW Datasocket application that will be the data repository for communication between SALT subsystems. The hardware will be selected as late as possible but will conform to the minimum requirements specified in the TCSS Specification. The TCSS will connect to the outside world via a 100Mb/s Ethernet network.

3.4 TCSS commands to subsystems

The TCSS and the SALT subsystems, need to operate in a fashion that will allow a subsystem to re-start and re-synchronise itself with the rest of the telescope, without forcing the TCSS to repeat previously issued command. This means that the TCSS communicates the required state of a subsystem (as embodied in the Command cluster that is transmitted), not event information. The required state is thus communicated to each subsystem in a cyclic fashion and needs to be responded to by that subsystem.
Figure 1: TCSS external software interfaces
<table>
<thead>
<tr>
<th>Item</th>
<th>Interfacing application</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Planning Database</td>
<td>Current telescope pointing direction, pupil details and trajectory</td>
</tr>
<tr>
<td>2</td>
<td>SDB</td>
<td>Current telescope pointing direction, telescope mode/states, failure information, currently observed object ID, trajectory details.</td>
</tr>
<tr>
<td>3</td>
<td>SAMMI</td>
<td>Current telescope pointing direction, telescope mode/states, currently observed object ID, trajectory details, pointing model details, initialisation status, failure information.</td>
</tr>
<tr>
<td>4</td>
<td>OPT</td>
<td>Current telescope pointing direction, pupil details and trajectory</td>
</tr>
<tr>
<td>5</td>
<td>SOMMI</td>
<td>Current telescope pointing direction, telescope mode/states, currently observed object ID, trajectory details.</td>
</tr>
<tr>
<td>6</td>
<td>PCON</td>
<td>Current telescope pointing direction, telescope mode/states, failure information, currently observed object ID, trajectory details.</td>
</tr>
<tr>
<td>7</td>
<td>SCON</td>
<td>Current telescope pointing direction, telescope mode/states, failure information, currently observed object ID, trajectory details.</td>
</tr>
<tr>
<td>8</td>
<td>SAMMI</td>
<td>Pointing and guidance offsets, acquisition, guidance and calibration commands</td>
</tr>
<tr>
<td>9</td>
<td>SOMMI</td>
<td>Set-up, day operation, pointing and guidance offsets, acquisition, guidance, close-out, maintenance and calibration commands</td>
</tr>
<tr>
<td>10</td>
<td>PCON</td>
<td>Pointing and tracking offsets, offset tracking command</td>
</tr>
<tr>
<td>11</td>
<td>SCON</td>
<td>Pointing and tracking offsets, offset tracking command</td>
</tr>
<tr>
<td>12</td>
<td>PCON</td>
<td>High-level mode commands, information to populate the FITS header</td>
</tr>
<tr>
<td>13</td>
<td>SCON</td>
<td>Information to populate the FITS header</td>
</tr>
<tr>
<td>14</td>
<td>TC</td>
<td>Required Tracker mode, MCP enable, rqd trajectory</td>
</tr>
<tr>
<td>15</td>
<td>SDC</td>
<td>Required Structure and Dome modes, MCP enable, rqd Structure and Dome angles, shutter open/close</td>
</tr>
<tr>
<td>16</td>
<td>TPC</td>
<td>Required Payload mode, guidance/focus probe required states/positions, guidance offset requirements, fold mirror position, plate scales, ADC and moving baffle required positions</td>
</tr>
<tr>
<td>17</td>
<td>BMS</td>
<td>BMS rqd mode, A/C on/off/setpoint, ventilation open/close/setpoint, CCAS shutter open/close, lights on/off (TBC, may be via SDC)</td>
</tr>
<tr>
<td>18</td>
<td>PCON</td>
<td>PFIS actual configuration and mode, exposure status, failure information</td>
</tr>
<tr>
<td>19</td>
<td>SCON</td>
<td>SALTICAM actual configuration and mode, exposure status, failure information</td>
</tr>
<tr>
<td>20</td>
<td>MACS</td>
<td>PMAS mode/state, mirror misalignment figures-of-merit, PM alignment cmnds, failure information.</td>
</tr>
<tr>
<td>21</td>
<td>TC</td>
<td>Tracker mode/state, actual trajectory, failure information, calibration move request</td>
</tr>
<tr>
<td>22</td>
<td>SDC</td>
<td>Structure/Dome mode/state, actual angles of Dome and Structure, failure information, calibration move request</td>
</tr>
<tr>
<td>23</td>
<td>TPC</td>
<td>Payload mode/state, guide probe position, image quality, focus figure-of-merit, ADC status, moving baffle status</td>
</tr>
<tr>
<td>24</td>
<td>BMS</td>
<td>BMS mode/state, telescope temperature, failure information, a/c settings, critical environmental parameters</td>
</tr>
<tr>
<td>25</td>
<td>TPC</td>
<td>Guidance and Focus corrections</td>
</tr>
<tr>
<td>26</td>
<td>MACS</td>
<td>Require mode/state</td>
</tr>
<tr>
<td>27</td>
<td>Local MMI</td>
<td>This is an internal interface, allowing TCSS commands, configuration changes and display of internal states and variables</td>
</tr>
</tbody>
</table>

Table 1: Preliminary list of TCSS external interface data
3.5 TCSS commands from SOMMI, SAMMI, Science Instruments

The TCSS is required to accept text-based non-cyclic commands and control the whole telescope accordingly. This has the following effect:

- The TCSS will continue to operate according to the most recent commands, regardless of whether the commanding machine is still active or not (e.g. during a reboot of SOMMI).
- The user will be able to provide command-line commands
- Script-like files, combining these commands into sensible groups can be written by the operator
- The TCSS needs to parse these text commands to generate the appropriate subsystem commands
- Initially, low-level text commands (typically subsystem commands) will be received by the TCSS, but as its sophistication grows during the integration process, these will become higher-level commands and the TCSS will be required to provide the appropriate sequencing and timing of subsystem actions.

Appendix A defines the command language that will be used and provides an overview of the proposed scheme for managing it to ensure that only valid commands are issued to the TCSS. This language has been named the SALT Command Language (SCL).

4 High-level flow diagrams and description

4.1 TCSS modes

The TCSS implements the telescope modes indicated in the TCSS Specification referenced in section 2.

4.2 TCSS Kernel

The TCSS Kernel is a multi-threaded application which can conceptually be divided into four parts which are detailed separately in section 5:

- **TCSS-hi** is the software that performs the high-level, co-ordinated control of the telescope. It includes the TCSS state machine and error checking and sends appropriate commands to the TCSS-lo software, to control each subsystem to the required state. This software runs in one thread, at 5Hz.
- **TCSS-lo** is the software that performs the low-level control of each of the subsystems, each running in its own independent thread at a rate appropriate for that subsystem (typically 5Hz). This software communicates with the appropriate subsystem using the ICD-defined clusters and monitors each subsystem for failures, reporting these to TCSS-hi. Additional TCSS-lo threads perform the bi-directional data communication with the MMI’s and Instruments.
- The Command Interpreter performs the conversion of commands from SOMMI, SAMMI and the science instruments. Its output is cluster commands that are sent to TCSS-hi or TCSS-lo as appropriate.
- The Pointing Models (comprising the Astrometric Model and the Geometric Model) are essentially subroutines called by other parts of the software, performing conversions between various axes.

The communication inside the TCSS kernel occurs in several ways, depending on the type of information:

- **Independent Queues** are used for the following information:
  - Commands from to TCSS-hi and TCSS-lo
  - Errors detected by TCSS-lo threads and sent to TCSS-hi
  - Astrometric Pointing model requests
  - Geometric Pointing model requests
  - Events that are sent to the TCSS state machine

- A shared memory area is used for general status information (Global Database) where each parameter is normally “owned” by one of the software modules and thus not subject to modification from several sources:
  - Information shared between TCSS-lo modules (e.g. ambient temperature)
• Information shared between TCSS-lo and TCSS-hi modules (e.g. current telescope direction)
• LabVIEW wired data connections for:
  o Initialisation data
  o Some queue references
  o Information flow inside a particular thread

Figure 2 illustrates these relationships in a diagrammatic form.

### 4.3 TCSS MMI

The TCSS MMI provides the Salt Operator and maintenance staff with a window to view the internal operation of the TCS Kernel software and the data communicated by the SALT subsystems.

The TCSS MMI will operate on the same principle as the other subsystem MMI’s, allowing multiple instances to be opened although only one instance will actually be in control. The layout and style of the MMI will be similar to that of the SDC and Tracker, with a fixed upper and left-hand display section and tab selectors to access other pages.

The following information will be shown on the fixed panes:
• TCSS mode
• Status of comms between TCSS kernel and TCSS MMI applications
• Time, Date
• Current status of each subsystem controlled by the TCSS (failure status, source of control and

![TCSS MMI Diagram](image-url)
mode

- Summary of TCSS internal failure states
- Commands being implemented by TCSS-hi

The following information should be grouped into appropriate sub-pages:
- TCSS state machine and mode control, including the ability to manually force certain modes
- TCSS pointing information
- TCSS detailed internal status showing the status of each TCSS-lo software module and commands in their queues
- TCSS global database information – selected global variables
- Manual commands – the ability to send text-based commands to the TCSS and subsystems
- Set-up and Calibration – changing of INI files (parameters are identified in section 5.1.5)

4.4 TCSS Software Tools

The following tools are envisaged:
- The Script Configuration Tool is used to define the valid keywords for the SALT Command Language (SCL) based on the SALT ICD cluster type definitions
- The Command Generator is used by the user (or the person configuring the software) to write the text commands.
- The State Table Tools are seen as a possible way of simplifying the definition of the state tables used to co-ordinate the operation of the subsystems (TCSS-hi) and also the state tables used to control the subsystems (TCSS-lo).

Details of these tools will follow the implementation of the TCSS Kernel software (see Appendix A).
5 TCSS Kernel Software Design

5.1 Top-level design of the TCSS Kernel

5.1.1 Main functional flow

Figure 3 below shows the functional flow of the TCSS Kernel as implemented in LabVIEW. It should be noted that this is a conceptual representation which lacks some technical detail and thus does not represent the actual LabVIEW code.

The outside loop is used for re-initialising the software after a software-controlled “re-start” while the inner loop forms the TCSS-hi software preceded by the INIT TCSS VI and followed by the Stop TCSS VI. Only two of the TCSS-lo VI’s are shown (Dome Control and Tracker Control). TCSS MMI interface software and the Command Interpreter are also not shown.

The TCSS-hi loop will execute at a rate determined by the INI file read by the initialisation software. Initialisation data is read and passed to all the top-level VI’s in both TCSS-hi and TCSS-lo. Within TCSS-hi, information that is required by successive iterations is contained in the “loop data” wire which is passed serially through its VI’s. The lower wire contains a reference to the error queue, where all errors are placed for response by the error handler (part of Monitor TCSS status). The same error queue is used for the whole TCSS kernel.

The details of the command queues flowing from TCSS-hi to TCSS-lo, the Global Database, Error Handling, Initialisation and Shutdown are described in the sections that follow.
5.1.2 Command Queues

“Commands” passing between the various threads of the TCSS Kernel are sent via queues, to ensure that no messages are lost due to the possibly asynchronous operation of these threads.

The following command queues have been defined:

- TCSS Command queue: This contains the SCL commands originating from outside the TCSS kernel which must be processed by the TCSS kernel.
- Dome Command queue: This contains the cluster commands that need to go to the “Dome Controller” part of the TCSS-low software, originating from the “Execute TCSS actions” part of TCSS-hi or the command interpreter.
- Structure Command queue (similar to the Dome Command queue)
- Tracker Command queue (similar to the Dome Command queue)
- Payload Command queue (similar to the Dome Command queue)
- BMS Command queue (similar to the Dome Command queue)
- PMAS Command queue (similar to the Dome Command queue)
- SDB Command queue (similar to the Dome Command queue)

When the receiving software has processed a command, it is removed from the queue. Over-and-above the actual commands, the queues will contain the status of each command, which would be one of the following:

- New
- Waiting to execute
- Being executed
- Completed (transient only)
- Failed

Details of the command “management” will be defined during the implementation phase.

5.1.3 Global Database

The purpose of the Global Database is to store non-command data that is exchanged between the various TCSS kernel threads.

The Global variables will not make use of the LabVIEW “Global” type, but rather use an un-initialised shift-register in a VI. An example of such a VI is shown in Figure 4. The variable has the following features to avoid race conditions:

- Before writing, a unique token must be obtained by first reading the variable
- Calling software with a valid token will be able to update the variable once
- Data can be read at any time without a token

A simplified version of this VI could be used for variables that do not have the danger of “race conditions”.
The content of each variable will be determined according to the demands of writing and reading the data and may be any of the valid LabVIEW data types.
5.1.4 Error Handling

Two types of errors can exist inside the TCSS kernel:

- LabVIEW VI errors: these will typically be wired from one VI to the next, in the traditional LabVIEW coding style with final output error cluster being placed on the error queue. This means that some of the downstream VI’s will not execute if an upstream error occurs.
- Errors from custom VI’s and system errors (e.g. dome shutter won’t close) are immediately placed on the error queue, but downstream VI’s will execute normally.

Each error will have either the standard LabVIEW error code or a custom error code, defined in the INI file. The error description and code contained in the INI file will also have an associated flag, indicating the seriousness of the error. Errors considered “Major Failures” will place the TCSS in “Major Failure” mode, disallowing telescope operation via the TCSS.

An error handler in Monitor TCSS status VI processes the error queue. It will initiate the appropriate actions depending on the errors that are reported.

Errors will be reported to SOMMI and the appropriate messages displayed to the operator.

5.1.5 Initialisation

At initialisation the following sequence of events will occur:

- The "TCSS Main.INI" file will be loaded from disk and used to initialise the TCSS. This file will contain the following information:
  - Error codes and descriptions
  - URL’s for communicating to each of the subsystems, SDB, SOMMI, SAMMI and the science instruments
  - Error detection parameters such as timeout values, maximum and minimum limits
  - Loop iteration times for each of the TCSS Kernel threads
  - Names of each of the command queues
  - The state transition tables for TCSS-hi and each subsystem controller
- Command queues will be set up and flushed.
- The Geometric Pointing model VI and the Astrometric Pointing VI (described in sections 5.4 and 5.5) will be called and instructed to initialise using separate INI files containing only the pointing model set-up parameters.

If the TCSS receives an “Initialise” mode command, it will first shut down, prior to re-initialising, hence the need for the outer loop in Figure 3.

Figure 5 shows the functional flow of the initialisation process.
This VI initialises the TCSS without requiring all the subsystems to initialise. It goes through the following steps:
- Define and flush (in case of re-start) command queues for each subsystem
- Set all LabVIEW variables to their default
- Read initialisation values for TCSS and subsystem controllers from INI file
- Set up INI information for all subsystems
- Set initial values for TCSS loop information

Figure 5: Initialisation of the TCSS Kernel

5.1.6 Shutdown

Shutting down the TCSS will perform the following sequence of events:
- Flush all the command queues and command all the controlled subsystems into “Ready” mode.
- Close all file references that may still be open
- Report “Shutdown” to the SOMMI and SAMMI

5.2 TCSS-hi

The top-level of the TCSS-hi software has already been shown in the previous section. This software forms the brains of the TCSS kernel, being responsible for the control of co-ordinated telescope functions. Three main VI’s have been identified at this stage, but this grouping of functionality may change during the implementation.

5.2.1 Monitor TCSS status

This software has the following functions:
- To monitor the state of each of the subsystem controllers and the subsystems they control, reporting any errors via the error queue and generating the appropriate control events for the state machine.
- Receiving the TCSS-hi commands from the Command Interpreter and generating the appropriate events for the state machine.
- Monitoring the execution of the commands received and updating the command status. Commands that have been implemented are removed from the queue.
- Monitoring the execution of TCSS-hi commands received and reporting their execution status to the command interpreter.
- To publish information from TCSS-hi that is required by the other threads to the Global
Database

- Handle errors in the command queue

Figure 6 shows the functional flow of the Monitor TCSS Status VI.

![Functional flow of the Monitor TCSS Status function](image)

The “Update GDB – TCSS” VI is possibly not required, but reserves a place for transferring information from the TCSS-hi thread loop data to the Global Database for use by other threads. It is probably more appropriate to perform this function in the “Execute TCSS Actions” VI described in section 5.2.3.

The “Generate TCSS Events” VI places events onto the TCSS Event queue (destined for the state machine in the “Determine TCSS mode” VI) based on two sources:

- It receives the TCSS Command Queue from the command interpreter and will generate an appropriate event for every new command
- It evaluates certain conditions based on the Global Database and the Loop data and generates events that are used by the state machine (e.g. completion of a Track).

The “Detect TCSS Errors” VI independently monitors the Global Database and Loop Data and will place error information on the error queue (see 5.1.4) and also place error events on the TCSS Event queue if appropriate.

If possible, the relationship between variable values, TCSS mode and the generated event should be an integrated part of the state table, read in from an INI file during initialisation.

### 5.2.2 Determine TCSS mode

The precise operation of this part of the software has not yet been defined in detail. If possible, this will be based on the state machine that is being developed for the Payload Computer.

The following features need to be built into this software:

- It will receive events from various parts of the software via the Event Queue.
- The response to a particular event will depend on the present mode of the software
- An event can result in one or more of the following:
  - Changing the mode of the TCSS
  - Triggering action(s) by TCSS-hi
  - Triggering action(s) by one or more of the subsystem controllers
  - Triggering action(s) by the pointing models
  - Triggering action(s) by the command interpreter
- The relationship between the current mode, events, new mode and actions should be in a table that is read during initialisation. If possible this should be linked to the table used in “Generate TCSS Events” (see 5.2.1) and to SCL “sequence files” that were debugged as low-level commands from SOMMI (See Appendix A).
- Due to the potential complexity of the actions that are possible, a “state table tool” may be appropriate. See 4.4.
Actually translating the action list into commands for each of the other threads is accomplished in “Execute TCSS actions” described in the next section.

5.2.3 Execute TCSS actions

This software has also not been defined in great detail, relying heavily on the precise implementation of the state machine. The following features need to be included:

- It receives the list of actions to perform from the preceding VI and then translates these into the appropriate queued commands to the other software threads. Part of this process is calling the Astrometric and Geometric Pointing Models, to calculate the required subsystem positions, based on the actions to be performed. Some of these positions may be pre-defined (e.g. Tracker and Structure positions for Primary Mirror alignment). The pointing direction will often not be only a point, but a time-series trajectory of points.
- The queued commands need to be identical to the output of the command interpreter outputs for non-TCSS-hi commands (i.e. based on the cluster type definitions in the ICD) as they will in fact be placed on the same queues.
- Synchronisation of command execution between various threads will require a “command manager” that is part of this software. (NOTE: the issue of synchronisation can be handled at various levels in the software and the most appropriate implementation still needs to be defined).
- Commands issued to other threads but not successfully implemented, need to be reported and appropriate events generated.
- If it is found to be more appropriate, then this software will perform the “Update GDB – TCSS” function described in 5.2.1).

5.3 Command Interpreter

The purpose of this software is to accept SCL commands and send the appropriate cluster-based commands to the various TCSS kernel threads. The syntax of the received commands is defined in Appendix A. As the keywords have been extracted from the ICD cluster definitions, updating those clusters from the SCL commands and indicating which cluster controls have changed, should not prove difficult. The ICD does not yet define the TCSS-hi commands, so a provisional list is presented in Appendix B.

The command interpreter performs the following functions:

- Receive SCL commands via the TCSS command queue (the actual reception of those commands from the outside world is handled by the appropriate TCSS-lo threads).
- Determine if a command can be executed by evaluating the following conditions:
  - Must it wait for a preceding command to finish or time to expire?
  - If this is a TCSS-lo command, may the command be implemented?
- Convert the SCL command to the appropriate cluster command, grouping sequential commands to the same cluster so that only one cluster command is issued, implementing all the SCL commands for that cluster
- Send the cluster commands to the appropriate threads for implementation via their appropriate command queues
- Monitor the execution status of each of the issued cluster commands (using serial number)
- Report the status of all SCL commands for transmission to their sources

Figure 7 shows a typical low-level command cluster, indicating the appropriate status information that is used to manage that command. Figure 8 shows a possible way of decoding the SCL commands, changing the value of the cluster command to that specified by the SCL command. This is not an optimal implementation and serves only to demonstrate that it can be done without major effort.
Figure 7: A typical cluster command inside the TCSS kernel
5.4 Astrometric Pointing Model

The prime purpose of the Astrometric Pointing Model is to calculate the required telescope pointing direction (local Azimuth and Elevation) to point the telescope at a given the position in the sky. This essentially requires the conversion of position in an astronomical co-ordinate and time system, to a local co-ordinate and time system. Intermediate and inverse transforms are also required.

The “Telescope Pointing Machine” developed by Jeffrey Percival and used at the WIYN telescope, provides the required functionality and accuracy, and is already mature software. This software has been developed in “C” and will be called in LabVIEW using a Code Interface Node. This software can perform the transforms and calculations:

- FK4 Precession to B1950
- FK5 Precession to J2000
- IAU 1980 Ecliptic to FK5 Equatorial
- IAU 1958 Galactic to FK4 B1950
- FK4 B1950 to FK5 J2000
- Heliocentric parallax
- Geocentric parallax
- Light deflection
- Aberration
- Precession from FK5 J2000 to date
- Nutation
- Earth’s rotation
- (HA, Dec) to (Az, El)
- Refraction
- WHAM co-ordinate system

The software is a table-driven state machine that provides fast and rigorous vector-based calculation of not only object position but also velocity. For details refer to the TPM Specification referenced in section 2.

The Astrometric Pointing VI (which contains the CIN that connects to the TPM software), should run once when called, and have the following interface:

- Cluster of independent input variables input
- Enumerated type showing function to perform input
- Cluster containing dependent variables input
- Cluster containing dependent variables output
- Error cluster

5.5 Geometric Pointing Model

5.5.1 Overview

The purpose of the Geometric Pointing Model is to calculate the required positions and orientations of each of the SALT subsystems to position the image of an object at the required location in the appropriate focal plane. This model converts the required pointing Azimuth and Elevation (outputs from the Astrometric Pointing model) into position commands for the Structure, Dome, Tracker, ADC and moving baffles, taking into account the position in the focal plane where the image should be and the imperfections in the telescope. The inverse of these calculations are also required.

Figure 9 provides a summary of the functional and data flow of the Geometric Pointing Model. It can be
seen that there are four pointing values that are added to command the tracker to the required position:

- The theoretical required position of the Tracker relative to the Top Hex (Ideal Tracker Frame), calculated from Azimuth and Elevation taking into account the geometry and misalignments of the pier, structure and primary mirror
- The correction for the misalignment caused by the Tracker Payload due to its misalignment, and optical effects
- An operator offset introduced during acquisition to remove the final pointing residual
- The guidance offset, which will take out all residual errors while tracking an object across the sky

The residual errors removed by the operator and the guidance system must be used as default offsets for the next time the telescope is pointed and must be accumulated in a suitable fashion to allow fine-tuning of the pointing model in an off-line way.

5.5.2 Accuracy requirements

The TCSS Specification requires a minimum accuracy for the Geometric Pointing model of 8" with a goal of 2", if one assumes that the Astrometric Pointing model is indeed as accurate as predicted.

This is means that:

- When viewing the acquisition image after pointing “blindly” at an object of known Elevation and Azimuth, the RMS of the radial distance from the telescope “boresight” to the centroid of the object’s image should be less than or equal to 8” on the sky (with a goal of 2”).

Figure 9: High-level flow diagram of the Geometric Pointing Model
This is a statistical value, calculated on the basis of at least 50 pointing instances under varying environmental conditions. The first pointing of an evening can be used as a “calibration” point and is thus not used in the calculation.

A second implicit requirement is that the image should appear at very nearly the same location in each of the various focal planes. This is required so that the image is not lost when switching the fold mirror. A value of 0.3” is set as goal here (TBC: depending on the mechanical constraints in the payload). This places a higher accuracy requirement on some of the pointing terms. The precise requirement and contribution of each error will have to be analysed as part of this implementation.

5.5.3 Internal Interface to Geometric Pointing VI

The operation of the Geometric Pointing VI should be similar to the Astrometric Pointing VI and the interface should thus also have the following interface:

- Cluster of independent input variables input
- Enumerated type showing function to perform input
- Cluster containing dependent variables input
- Cluster containing dependent variables output
- Error cluster

5.5.4 Geometric Pointing model design

The Geometric Pointing model needs to correct for the geometry and misalignments shown in Table 2 below. Figure 10 shows the preliminary data flow through the pointing model (some of the corrections shown are actually performed in other subsystems, as described in 5.5.5). The shaded blocks in both Figure 10 and Table 2 indicate areas where simple models will be implemented at first and only expanded to more complex models if required to meet the specified pointing accuracy. The software should implement these terms as independent entities that can be adjusted and calibrated separately and there should be a “global top-up” model that is used to correct for the residual errors (the last item in Table 2).

A growth path is to use the star catalogue information contained in the STARCAT part of the TCS, to perform field recognition of the acquisition image in the same way that the Star Tracker works. This will provide accurate on-sky pointing accuracy and could lead to improved telescope efficiency. This functionality is not planned for telescope “first light”.

A growth path is to use the star catalogue information contained in the STARCAT part of the TCS, to perform field recognition of the acquisition image in the same way that the Star Tracker works. This will provide accurate on-sky pointing accuracy and could lead to improved telescope efficiency. This functionality is not planned for telescope “first light”.
<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Dependence on other variables</th>
<th>Type*</th>
<th>Calibration method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Structure North misalignment</td>
<td>None</td>
<td>Offset</td>
<td>Theodolite survey, star tracker</td>
</tr>
<tr>
<td>2</td>
<td>Dome North misalignment</td>
<td>None</td>
<td>Offset</td>
<td>Theodolite survey</td>
</tr>
<tr>
<td>3</td>
<td>Pier Global tilt</td>
<td>Structure Azimuth</td>
<td>Axis rotation</td>
<td>Theodolite survey, star tracker</td>
</tr>
<tr>
<td>4</td>
<td>Pier flatness</td>
<td>Structure Azimuth</td>
<td>LUT</td>
<td>Theodolite survey, star tracker</td>
</tr>
<tr>
<td>5</td>
<td>Base wedge design tilt</td>
<td>None</td>
<td>Axis rotation</td>
<td>Design drawing</td>
</tr>
<tr>
<td>6</td>
<td>Truss design thickness</td>
<td>None</td>
<td>Axis movement</td>
<td>Design drawing</td>
</tr>
<tr>
<td>7</td>
<td>Truss/base wedge misalignment</td>
<td>None</td>
<td>Axis rotation</td>
<td>Theodolite, adjust Truss relative to Top Hex centre (ITF origin)</td>
</tr>
<tr>
<td>8</td>
<td>Truss/base wedge warping</td>
<td>Temperature, structure azimuth</td>
<td>Model or LUT</td>
<td>Monitoring mirror angle with temperature and structure az angle</td>
</tr>
<tr>
<td>9</td>
<td>Structure design misalignment</td>
<td>None</td>
<td>Axis rotation</td>
<td>Design drawing</td>
</tr>
<tr>
<td>10</td>
<td>Structure misalignment</td>
<td>None</td>
<td>Axis rotation</td>
<td>Theodolite survey, star tracker</td>
</tr>
<tr>
<td>11</td>
<td>Structure warping, bending</td>
<td>Temperature, structure azimuth, Tracker position,</td>
<td>Model or LUT</td>
<td>Surveying and using star tracker under varying temp conditions and tracker positions</td>
</tr>
<tr>
<td>12</td>
<td>M1 misalignment (mirror relative to Truss). Could be combined with item 7.</td>
<td>None</td>
<td>Axis rotation</td>
<td>Theodolite, adjust M1 segment relative to Top Hex centre (ITF origin)</td>
</tr>
<tr>
<td>13</td>
<td>Calculation of Tracker Commands</td>
<td>None</td>
<td>Axis rotation</td>
<td>None</td>
</tr>
<tr>
<td>14</td>
<td>SAC design image scale</td>
<td>None</td>
<td>Angular conversion</td>
<td>Sagem acceptance test documents</td>
</tr>
<tr>
<td>15</td>
<td>SAC image translation/defocus and axis-symmetric distortion</td>
<td>Tracker Tip/Tilt</td>
<td>LUT or model</td>
<td>Information provided by SAGEM</td>
</tr>
<tr>
<td>16</td>
<td>SAC mounting misalignment w.r.t. other payload parts and mounting ring</td>
<td>None</td>
<td>Axis rotations</td>
<td>Survey Payload in the lab</td>
</tr>
<tr>
<td>17</td>
<td>Fixed structure flexure</td>
<td>Tracker Tip/Tilt</td>
<td>LUT or model</td>
<td>?</td>
</tr>
<tr>
<td>18</td>
<td>Moving baffle installation misalignment</td>
<td>Non</td>
<td>Offset in X,Y</td>
<td>Survey in lab</td>
</tr>
<tr>
<td>19</td>
<td>ADC design effect on image location</td>
<td>Object Elevation, Tracker position</td>
<td>Model</td>
<td>Design drawing</td>
</tr>
<tr>
<td>20</td>
<td>ADC misalignment effect on image</td>
<td>Object Elevation, Tracker position</td>
<td>Model or LUT</td>
<td>Measure in lab</td>
</tr>
<tr>
<td>21</td>
<td>Rotating structure misalignment</td>
<td>None</td>
<td>Axis rotation</td>
<td>Survey in lab and on Tracker, finalise with guidance tests on sky (rho dependence)</td>
</tr>
<tr>
<td>22</td>
<td>Rotating structure flexure</td>
<td>Tracker Tip/Tilt/Rho</td>
<td>LUT or model</td>
<td>?</td>
</tr>
<tr>
<td>Description</td>
<td>Dependence on other variables</td>
<td>Type*</td>
<td>Calibration method</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-------------------------------</td>
<td>--------------------------------</td>
<td>--------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Fold mirror image inversion</td>
<td>None</td>
<td>Conversion</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Fold mirror image distortion and translation</td>
<td>Station in use, mirror in use</td>
<td>Conversion</td>
<td>Survey in lab</td>
<td></td>
</tr>
<tr>
<td>PFIS mounting misalignment</td>
<td>None</td>
<td>Axis rotation</td>
<td>Survey and finalise with guidance on-sky (rho dependence)</td>
<td></td>
</tr>
<tr>
<td>FIF mounting misalignment</td>
<td>None</td>
<td>Axis rotation</td>
<td>Survey and finalise with guidance on-sky (rho dependence)</td>
<td></td>
</tr>
<tr>
<td>SALTICAM mounting misalignment</td>
<td>None</td>
<td>Axis rotation</td>
<td>Survey and finalise with guidance on-sky (rho dependence)</td>
<td></td>
</tr>
<tr>
<td>SALTICAM optics image distortion</td>
<td>None</td>
<td>Axis rotation</td>
<td>Survey and finalise with guidance on-sky (rho dependence)</td>
<td></td>
</tr>
<tr>
<td>Global residual correction</td>
<td>Structure Azimuth, Tracker X, Tracker Y, Tracker Rho, Temperature, field position, focal station</td>
<td>Statistically adjusted model (e.g. Kalman filter)</td>
<td>Each operator adjustment can be used to adjust this as can the guidance corrections. Initially this adjustment will be off-line</td>
<td></td>
</tr>
</tbody>
</table>

**NOTES**: The type of correction applied is shown here for information only, as an initial guess.

- **LUI** – Look-up table
- **Axis rotation** – Euler transform of position and/or rotation
- **Conversion** – simple ax + b calculation
- **Model** – more complex numeric calculation

**Table 2: Geometric Pointing correction terms**
Figure 10: Flow diagram of the Geometric Pointing model (including some non-TCSS parts)
5.5.5 Pointing Interface to other subsystems

The "SALT Axes and Calibration definition" document referenced in section 2 defines the co-ordinate systems in which data is communicated between the various subsystems and thus also defines what the output co-ordinate systems of the Geometric Pointing model should be. The definitions in that document which will be implemented in the ICD, will override those described below and shown in Figure 10.

The communication is normally with respect to an "Ideal" subsystem and can be summarised as follows:

- **TRACKER**: Communication is in the Ideal Tracker Frame (ITF), which is defined relative to the actual Top Hex onto which the Tracker is mounted.
- **PRIMARY MIRROR**: Communication is in the Telecentric Co-ordinate Frame (TCF), defined relative to the Primary Mirror truss, and the vertex of the central segment.
- **DOME**: Communication is in the Ideal Dome Frame (IDF), which is defined relative to the top of the ring wall.
- **STRUCTURE**: Communication is in the Structure Base Frame (SBF), which is defined relative to the top of the actual pier.
- **PAYLOAD COMPUTER**: This depends on the part of the payload being referred to:
  - **ADC**: Communication is in the Ideal ADC Frame (IAF), which means that the TCSS Geometric Pointing model excludes the calibration of the ADC positioning stages, but includes ADC misalignment and optical effects.
  - **MOVING BAFFLE**: Communication is in the Ideal Baffle Frame (IBF), which means that the TCSS Geometric Pointing model excludes the calibration of the Moving Baffle positioning stages, but includes Moving Baffles misalignment.
  - **PFIS GUIDANCE SYSTEM**: Communication is in the Ideal PFIS Frame (IPF), which is the physical position of an object in the focal plane, corrected for all effects, measured relative to the PFIS mounting frame (i.e. including PFIS mounting misalignment w.r.t. the payload but excluding PFIS internal misalignment or flexure – these would be negligible for pointing the guidance system). This excludes the position calibration of the guide probe positioning stage and any optical misalignment inside the guidance system.
  - **FIF GUIDANCE SYSTEM**: Communication is in the Ideal FIF Frame (IFF), which is the physical position of an object in the focal plane, corrected for all effects, measured relative to the FIF mounting frame (i.e. including FIF misalignment w.r.t. the payload but excluding FIF internal misalignment or flexure – these would be negligible for pointing the guidance system). This excludes the position calibration of the guide probe positioning stage and any optical misalignment inside the guidance system.
  - **SALTICAM GUIDANCE SYSTEM**: Communication is in the Ideal SALTICAM Frame (ISF), which is the physical position of an object in the focal plane (after the SALTICAM Optics), corrected for all effects, measured relative to the SALTICAM mounting frame (i.e. including SALTICAM misalignment w.r.t. the payload but excluding SALTICAM internal misalignment or flexure – these would be negligible for pointing the guidance system). This excludes the position calibration of the guide probe positioning stage and any optical misalignment inside the guidance system but includes the optical misalignment introduced by the SALTICAM optics.
  - **PFIS**: Communication is in the Ideal PFIS Frame (IPF), which is the physical position of an object in the focal plane, corrected for all effects, measured relative to the PFIS mounting frame (i.e. including PFIS misalignment w.r.t. the payload but excluding PFIS internal misalignment or flexure). This information would also be used for slit manufacture, via the PFIS subsystem.
  - **FIF**: Communication is in the Ideal FIF Frame (IFF), which is the physical position of an object in the focal plane, corrected for all effects, measured relative to the FIF mounting frame (i.e. including FIF misalignment w.r.t. the payload but excluding FIF internal misalignment or...
flexure). This excludes the position calibration of the FIF position stages.

- SALTICAM: Communication is in the Ideal SALTICAM Frame (ISF), which is the physical position of an object in the focal plane (after the SALTICAM Optics), corrected for all effects, measured relative to the SALTICAM mounting frame (i.e. including SALTICAM misalignment w.r.t. the payload but excluding SALTICAM internal misalignment or flexure).

The positions of objects and guidance errors are reported in these terms, except the guidance and focus errors that is sent directly to the Tracker from the Payload and SALTICAM which need to be converted to Ideal Tracker Frame using a small-angle approximation.

5.6 TCSS-lo

5.6.1 Subsystem Controllers

5.6.1.1 OVERVIEW

The purpose of the subsystem controllers are to receive the low-level cluster commands from the Command Interpreter and TCSS-hi and to relay this information to the appropriate subsystem at the appropriate communication rate. They also need to report the status of each command received, removing completed commands from the queue. The communication integrity with the subsystem and the subsystem health are also monitored and reported via the error queue.

5.6.1.2 TOP-LEVEL VI

Figure 11 below shows the top-level VI for the control of the Dome and can be considered typical. It receives initialisation information from the INIT TCSS VI and thereafter continues in its own loop until it receives a “TCSS shutdown” command. Note that this would normally not coincide with the shutdown of the subsystem being controlled.

![Figure 11: A typical top-level VI for a subsystem controller](image)

The sequence of functions in the dome controller resemble those of TCSS-hi as described in section 5.1.1 with the addition of the Receive and Transmit VI’s that exchange data with the subsystem being controlled and are defined in the paragraphs that follow.

5.6.1.3 RECEIVE SUBSYSTEM DATA

This software receives the data from the subsystem being controlled, using the ICD-defined Rx VI’s, as shown in figure 12 below. Communication failures are reported to TCSS-hi via the error queue. The “Monitor Subsystem” VI will ensure that appropriate events are generated to respond locally to the failure.
5.6.1.4 Transmit Subsystem Data

Figure 13 shows the transmission of data to the Dome subsystem using the ICD-defined Tx VI. This is the typical implementation for all subsystems.
5.6.2 Operator Interface Controllers

The interfaces to the SOMMI, and SAMMI are implemented on separate threads, each performing the following functions:

- Arbitrate between multiple instances of SOMMI and SAMMI to ensure that there is only one “master” of each at any one time.
- Receive the SCL command sequence using an ICD-defined cluster and if the data validity is set, pass the individual commands to the command interpreter.
- Report back the progress of every command in the TCSS-hi and TCSS-lo queues (using the unique command serial number)
- Transmit information required by SOMMI and SAMMI for display purposes, from the Global database

5.6.3 Instrument Controllers

These are identical to the Operator Interface Controllers, except that the commands received from them will only be allowed when certain pre-defined conditions have been met (e.g. when that instrument is in use). The information reported is also limited to the necessary status information and information required to populate the FITS header of the science data.

It is not anticipated that these “controllers” will have any control over the science instruments as such, but rather will allow the instruments to have limited control of the telescope. The following specific commands are envisaged at this stage (TBD):

- Offset the image in the focal plane by a certain amount
- Follow a time-synchronised square-wave positional offset of a certain period (for sky subtraction)

If it is a requirement that the instrument point the telescope at a specific RA and Dec, it is recommended that this occur via SOMMI/SAMMI and the Operation Planning Tool.

5.6.4 Local MMI Controller

This software will communicate with the TCSS local MMI. It will perform the following functions:

- Transmit information to multiple local MMI’s using pre-defined clusters at the appropriate update rate (typically 5Hz).
- Enable the receipt of control information from one local MMI if the TCSS receives a “maintenance mode” command on its SOMMI interface.
- Hand-over control from one local MMI to another when control is released
- Receive SCL and other commands from the local MMI
Appendix A: Definition and Management of TCSS commands

A.1 Introduction

The text-based commands going to the TCSS originate from SOMMI, SAMMI, PFIS, HRS and possibly SALTICAM. The language that makes up these commands needs to be simple and allow easy implementation in LabVIEW, which is not a "text-based" language and does thus not naturally allow scripting. Standard command parsers are also not appropriate.

There will be two types of commands to the TCSS:

• Low-level commands: These are actually commands that are sent directly to the telescope subsystems and contain no inter-subsystem co-ordination. These commands end up in a change in the cyclic "cluster" state commands which are sent to the subsystems from the TCSS. The TCSS adds minimal intelligence to these commands, and converts the text-based command into a cluster commands

• High-level commands: These commands are actually to the TCSS rather than one of the particular subsystems, and are used to co-ordinate the operation between subsystems and to perform "system-level" functions like calculating pointing information. The TCSS high-level software (called TCSS-hi) provides this inter-subsystem co-ordination and triggers the appropriate subsystem actions. See Appendix B for a provisional list of high-level commands.

These text commands make up the SALT Command Language (SCL).

As telescope subsystems will be modified during the life of the telescope and because the level of telescope automation will increase, the SCL will be required to change with time without requiring major software changes. Low-level commands will be used during commissioning and "sequence files" built up in SOMMI. These will later be combined into single high-level commands with the sequencing taking place in the TCSS-hi software.

A.2 Syntax of the SALT Command Language (SCL)

In order to simplify the software and facilitate upgrade-ability, it is essential that SCL is defined according to the command clusters that are sent to the SALT subsystems. If this is not done, the interpretation of these commands is no longer just text parsing, requiring more than the minimum intelligence.

As all SCL commands will end up in cluster commands to the subsystems, the cluster definitions should be used to define the language (see section A.3 below). Although this is not the most efficient implementation (in terms of the number of keywords or parameters), this one-to-one relationship gives the appropriate simplicity in the software. Although the communication of commands to the TCSS will be text-based, a set of command cluster will be defined, from which the appropriate high-level SCL commands can be derived.

The following syntax is proposed (but may alter during implementation):

```plaintext
<Subsystem name>.<parameter name>=<parameter value[,parameter value][,...][&flag]
```

Items enclosed in "<>" are mandatory and those in "[]" are optional. Commands are separate by carriage returns (or line feeds TBC). The "." separator between keywords, the ":" separator between parameters the "=" and "&" are mandatory. Spaces are ignored within and between key words. Commands are not case sensitive. The keywords are defined as follows:
Subsystem name: The name of subsystem that is the destination of the information, being one of
the following (note this is NOT necessarily the name of the computer doing the controlling although it
often is):
• Tracker
• Payload
• Dome
• Structure
• PMAS
• BMS
• TCSS
• PFIS
• SCAM
• SDB
• TCSS

Parameter name: These keywords differ per subsystem and are the variable names of items in the
command clusters flowing from the TCSS to the subsystem controlling computers. They are derived
directly from the cluster type definitions in the ICD. In the long term, these can be rationalised to
become more similar. The “validity” and “timestamp” of the clusters are considered “protocol” and are
thus NOT part of SCL.

Parameter value: This is the value of the parameter, in the correct type of the command cluster. As
clusters are not allowed inside the “ICD clusters” only the following types are valid:

- NUMERIC: The internal LabVIEW convention applies here.
- STRING: Any string is allowed except the “flag” indicator, & which is a reserved character.
- ENUMERATED and RING: the appropriate “string” value is the parameter, derived from the list
  that can be selected (not case sensitive).
- ARRAY: Some or all of the array values can be changed and three syntaxes are supported:

  A list of comma separated Parameter values are entered (as per the standard syntax): the
  array length is determined by the number of parameters listed and they become the values of
  the array elements. This can only be used for 1-D arrays.

  When changing some or all of the array elements the Parameter value for an array is defined
  as follows, and multiple parameter values are allowed:
  
  \[
  \langle \text{Index of first dimension}[[\text{Index of second dimension}]]\ldots\rangle; \langle \text{Array element value} \rangle
  \]

  If the whole array needs to be set to set to a specific single value the following syntax applies
  and only one parameter is allowed:

  \[
  \langle \text{Start Index of first dimension}>:<\text{Finish Index of first dimension}>
  
  \langle[[\text{Start Index of second dimension}:<\text{Finish Index of second dimension}]
  
  \langle[\ldots];\langle \text{Array element value} \rangle \rangle \ (\text{all on one line})
  \]

- BOOLEAN: The Parameter value is TRUE or FALSE.
&Flag: This allows the addition of additional information that increases the flexibility of the
interpretation of the command by the TCSS. The following flags are foreseen that this time:

&W[time duration] Wait until the previous commands to this subsystem are finished before
proceeding with this command. If followed by a number, this represents the number of seconds
to wait after finishing the previous commands, before proceeding with this command.
&T<time duration> Do not implement this command until the defined number of seconds
have passed since issuing this command.
&A Abort all unfinished commands to this subsystem and implement this one.

Examples (based on the ICD of 26 November 2002):
  o dome.dome mode = ready &w will place the dome in “Ready mode” when all the
    previous commands to the TCSS are completed.
  o Dome.DomeAzAngle=12.3 will rotate the dome to the position 12.3 degrees East of
    North.
  o Dome. e-stop pushed = TRUE will inform the SDC that an Emergency Stop button
    has been pushed and that the software should respond appropriately
  o PMAS. Piston = 31;1.3e-6, 45;-3.4e-6 will move Primary Mirror segments 31
    1.3micron away from the Truss and segment 45 3.4micron towards the Truss, leaving
    all the other segments at their current position.
  o PMAS. Truss temperature = 12.3, 13.12, 10.053, 13, 12.5 will inform the MACS
    computer of the current temperature of all five Truss sensors
  o PMAS. Piston = 0:90; 0.000001 will move all the Primary Mirror segments 1micron
    away from the Truss
  o TCSS.reset=SDC will cause the TCSS to issue a reset command to the SDC computer
    (this example is not from the current ICD).
  o TCSS.2darray = 0:44 | 0:12 ; 10.00000 sets all the elements of a 2-D array with 45
    columns and 13 rows to a value of 10:00000 (row/column sequence is TBD).

Comments can be added on a line of their own, starting with the % character.

A.3 Maintenance of SALT Command Language

Figure A1 below illustrates a mechanism to do three things:
  a. To ensure that SCL contains only commands that can be understood by the TCSS and sent to the
     subsystems as appropriate (script configuration tool).
  b. To ensure that despite the large number of possible parameters that can be used, the user (or any
     software that can generate these text commands) will only send valid commands (command
     generator)
  c. To simplify the control software changes resulting from changes in the interface between
     subsystems (state table tool).

The Script Configuration Tool is used to define the valid keywords for the SALT Command Language
(SCL) based on the SALT ICD cluster type definitions. An INI file read by this tool, defines which
subsystem names are associated with which clusters, and also defines the flags and syntax. The
output of this tool is an XML file (TBD) defining the SCL, which in turn is used by the Command
Generator.

The Command Generator is used by the user (or the person configuring the software) to write the
text commands. These commands would normally be grouped together in Sequence Files, which
contain the set of commands required to achieve a particular telescope end result (e.g. point to a
designated star). The commands contained in these sequence files would be sent to the TCSS as a
group and interpreted as defined in section A.2.

The State Table Tools are seen as a possible way of simplifying the definition of the state tables
used to co-ordinate the operation of the subsystems (TCSS-hi) and also the state tables used to
control the subsystems (TCSS-lo). It is suggested that possible states and commands are bounded by
the definition of the ICD clusters and that this tool would allow the software developer to alter
subsystem checks and interactions without undue changes in the software code.

Figure 1A: Software tools to manage the software impact of ICD changes
### Appendix B: Provisional list of TCSS high-level commands

The table below is a very preliminary list of the high-level commands that need to be implemented by TCSS-hi. This will be developed during commissioning.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Subsystem actions required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Re-start TCSS</td>
<td>TCSS in Shutdown, INIT and then Ready mode, all subsystems into ready-mode</td>
<td></td>
</tr>
<tr>
<td>Re-start entire telescope</td>
<td>TCSS in Shutdown, INIT and then Ready mode, all subsystems into ready-mode</td>
<td>(or bypassing shutdown if not required on a subsystem)</td>
</tr>
<tr>
<td>Prepare for observing</td>
<td>Turn off aircon, open louvers, open shutter, turn off lights, check all subsystems ready</td>
<td></td>
</tr>
<tr>
<td>Close after observing</td>
<td>Close louvers, close shutter, turn on aircon</td>
<td></td>
</tr>
<tr>
<td>Pause observing</td>
<td>Close louvers and shutter</td>
<td></td>
</tr>
<tr>
<td>Resume observing</td>
<td>Open louvers and shutter</td>
<td></td>
</tr>
<tr>
<td>Observe sidereal RA/Dec</td>
<td>Rotate structure to optimum angle (auto raise and lower), rotate dome to structure angle, move Tracker to best position to start track, start tracking object at sidereal rate when object within Tracker field-of-regard.</td>
<td></td>
</tr>
<tr>
<td>Observe non-sidereal RA/Dec/ephemeris</td>
<td>Rotate structure to optimum angle (auto raise and lower), rotate dome to structure angle, move Tracker to best position to start track, start tracking object at specified rate when object within Tracker field-of-regard.</td>
<td></td>
</tr>
<tr>
<td>Co-ordinated move to Az/El, Tracker position</td>
<td>Rotate structure to required angle (auto raise and lower), rotate dome to structure angle, move Tracker to specified position, do not track object.</td>
<td></td>
</tr>
<tr>
<td>Co-ordinated move to RA/Dec, Tracker position</td>
<td>Rotate structure to required angle (auto raise and lower), rotate dome to structure angle, move Tracker to specified position, do not track object.</td>
<td></td>
</tr>
<tr>
<td>Prepare for Mirror Alignment (Position 1 or Position 2)</td>
<td>Rotate structure to CCAS (auto raise and lower), rotate dome to structure angle, open shutter, move Tracker to specified position (e.g. Top left), turn on CCAS instrument</td>
<td></td>
</tr>
<tr>
<td>Start Mirror Alignment</td>
<td>Perform measurement of first part of Primary Mirror</td>
<td></td>
</tr>
<tr>
<td>Next step in Mirror Alignment</td>
<td>Perform measurement of second part of Primary Mirror</td>
<td></td>
</tr>
<tr>
<td>Accept new alignment values</td>
<td>Accept new mirror alignment values</td>
<td></td>
</tr>
<tr>
<td>Guide to object RA/Dec/station</td>
<td>Move appropriate guide probe to required RA/Dec</td>
<td></td>
</tr>
<tr>
<td>Guidance ON/OFF/station</td>
<td>Tracker implement guidance corrections from appropriate guide probe</td>
<td></td>
</tr>
<tr>
<td>Tracking ON/OFF/rate</td>
<td>Calculate trajectory, Tracker follow trajectory or stop</td>
<td></td>
</tr>
<tr>
<td>Autofocus ON/OFF/station</td>
<td>Tracker implement focus corrections from appropriate focus probe</td>
<td></td>
</tr>
<tr>
<td>Guide offset</td>
<td>Payload move guide probe to achieve telescope offset</td>
<td></td>
</tr>
<tr>
<td>Track offset</td>
<td>TCSS add offset to pointing angle</td>
<td></td>
</tr>
<tr>
<td>Focus offset</td>
<td>Payload add offset to focus probe</td>
<td></td>
</tr>
<tr>
<td>System Maintence Mode</td>
<td>Place all subsystems into maintenance mode</td>
<td></td>
</tr>
<tr>
<td>TCSS maintenance Mode</td>
<td>Place TCSS in maintenance mode</td>
<td></td>
</tr>
<tr>
<td>Get Pointing data RA, Dec, Epoch</td>
<td>Uses TCSS to calculate the pointing angles and return the appropriate subsystem required positions</td>
<td></td>
</tr>
</tbody>
</table>

Table B1: High-level TCSS commands
The following issues need to be resolved concerning the high-level commands:

- How will day-time Instrument Calibration be handled? A possible method is to have SOMMI record the track information during an observation and then re-play it to the TCSS with a “calibration” flag set to prevent the Tracker moving and to enable fast movement of the moving baffle. The user could edit the recorded file to optimise the operation.

- A second issue is the balance between high and low-level commands. The level of intelligence needed in the TCSS state machine depends on this issue. It seems that a reasonable solution is to have a powerful “command filter” that will allow certain subsystem commands through, even when some “high-level” commands are being processed. (e.g. switching a fold mirror could be a low-level command that should not take place while guidance is taking place but should be allowed during the execution of a “Observe sidereal RA/Dec” command. Doing this may alleviate the need for high-level co-ordinated commands.