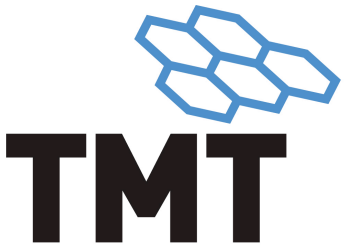

THIRTY METER TELESCOPE

TMT SOFTWARE REQUIREMENTS FOR LOW-LEVEL SUBSYSTEMS

TMT.SFT.DRD.12.001.REL05

October 15, 2012

TABLE OF CONTENTS

1 INTRODUCTION

This document is the Design Requirements Document (DRD) for software that is included in what TMT calls *low-level subsystems*. In the TMT software system a low-level subsystem is not a general term; rather, it indicates a kind of software/hardware system with specific characteristics. The characteristics are listed and discussed in Section 3, but are also listed here.

A low-level software subsystem is identified by the following characteristics:

- The software is tightly coupled to or embedded in a vendor-delivered hardware system.
- The functionality of the software is limited in scope to control of hardware and has limited capabilities.
- The functionality of the system does not require any kind of information or configuration change for different science observing modes.
- The system and software must be a standalone product (autonomous) with no dependencies on other TMT systems (other than the TMT Software Safety System).
- The software is delivered as a final product with the subsystem.
- The software is not expected to change over the lifetime of the project, or changes will be infrequent requiring considerable observatory resources and planning.

This document includes the requirements and recommendations for software executing in low-level subsystems that must be integrated into the TMT software system.

TMT prefers that system software for low-level hardware control be built and integrated using the following approaches in the order shown:

1. A Type 4 PLC/PAC system.
2. A Type 1 built with networked controllers and TMT Assemblies and HCDs.
3. As Type 2 third-party vendor provided hardware (not generally applicable to entire subsystems).
4. As a Type 3 low-level subsystem that meets the requirements of this document.

While low-level software subsystems are sometimes necessary, the low-level subsystem integration approach is not the preferred TMT approach**. Using a Type 3 software approach requires explicit permission from TMT**. Because of its characteristics, it requires this special requirements document.

This document is part of the Observatory Software Design Phase Documentation Plan (RD01).

1.1 PURPOSE

Low-level software subsystems have unique characteristics within the TMT software architecture and design. Because of their unique characteristics, design requirements are needed to ensure that the low-level software subsystem can be successfully integrated, tested, and maintained over the lifetime of the Observatory.

1.2 SCOPE

This document provides software requirements and recommendations for software that is to be developed for TMT low-level subsystems.

1.3 AUDIENCE

This document is targeted primarily towards software developers who are designing and implementing software as part of the TMT construction phase and reviewers of the TMT software system.

1.4 APPLICABLE DOCUMENTS

AP01 OSW TN003 – TMT Software Testing and Simulation, TMT.SFT.TEC.12.001.

AP02 TMT Software Management Plan, TMT.SEN.SPE.08.002 (to be revised).

AP03 TMT Environmental, Safety and Health Process Guidelines, TMT.PMO.MGT.12.009.

AP04 TMT Software Development Process, TMT.SFT.TEC.12.012 (not yet available).

AP05 OSW TN002 - TMT Guidelines for Software Safety, TMT.SFT.TEC.11.022.

AP06 OSW TN004 – TMT Software Review Deliverables, TMT.SFT.TEC.12.003.

1.5 REFERENCE DOCUMENTS

RD01 OSW Design Phase Documentation Plan (TMT.SFT.TEC.11.004).

RD02 OSW TN001-Responsibilities of OMOA Layers, TMT.SFT.TEC.11.016.

RD03 TMT Software Vision, TMT.SFT.TEC.11.005.

RD04 TMT Software Technical Architecture, TMT.SFT.TEC.11.013 (not yet available).

1.6 CHANGE RECORD

Revision	Date	Who	Modifications
DRF01	May 4, 2012	KG	Initial Draft
DRF02	May 23, 2012	KG	Updated with new requirements and modifications.
DRF03	June 15, 2012	KG	Updates based on M. Sirota comments.
REL04	July 9, 2012	KG	Additional updates based on M. Sirota comments. Converted to REL.
REL05	Oct 15, 2012	KG	Included preferred implementation choices.

1.7 ACRONYMS

API Application Programmer Interface
DRD Design Requirements Document
EUI Engineering User Interface
HCD Hardware Control Daemon
LSS Low-level Software Subsystem
OSW Observatory Software
TBD To Be Determined
TMT Thirty Meter Telescope

2 INTEGRATION OF LOW-LEVEL DEVICE CONTROLLERS, HARDWARE AND SUBSYSTEMS

An important concern of the TMT software architecture is the integration of hardware and software subsystems. The Observing Mode Oriented Architecture (OMOA) adopted by TMT is a simple, layered approach that provides support for the integration of hardware. This document does not describe the OMOA layering in depth. Please see RD02, RD03 and RD04 for more information on the TMT software system, OMOA layers and their responsibilities.

The lowest layer in the OMOA software system, called the Hardware Control Layer, consists of all the controllable devices that are available for use in the observatory and the software that communicates with the hardware controllers. Software components in this layer, called Hardware Control Daemons (HCD), are responsible for communication with a hardware device or system and integrating it into the TMT system.

The possibilities for hardware integration are shown with four cases. The first two cases shown in Figure 1 represent the integration of individual hardware components. These two cases represent the preferred approach for integrating hardware.

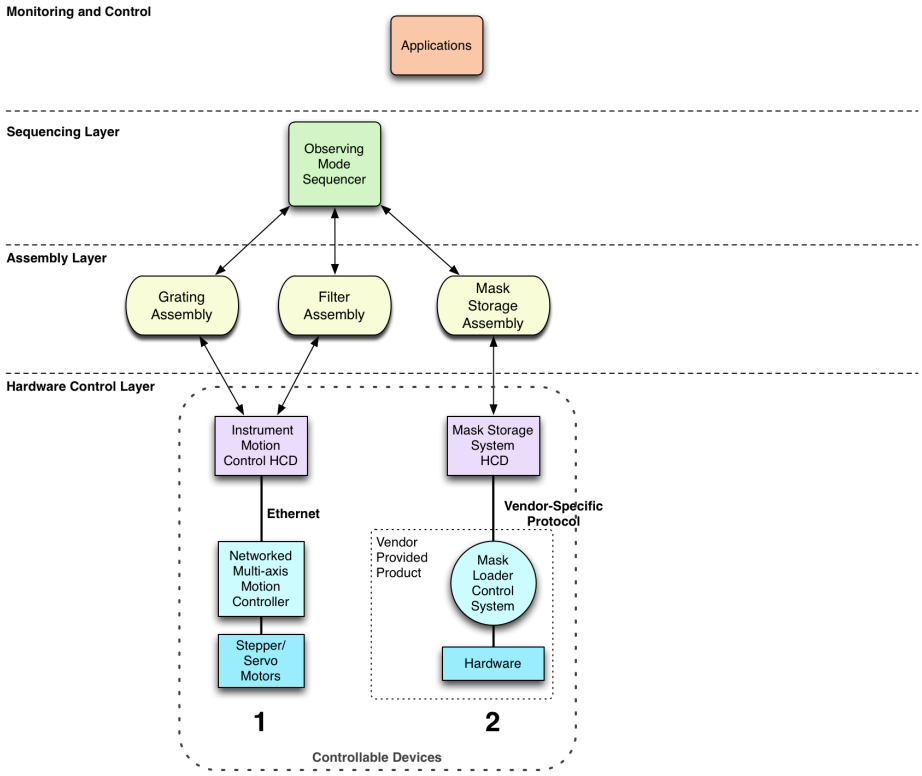

Figure 1: This application shows all OMOA layers. Hardware such as motion control is integrated through network connections to Hardware Control Daemons in the Hardware Control Layer.

Networked Controllers (1). In case 1, the HCD represents a networked piece of hardware with communication over Ethernet using TCP/IP-based protocols. The example (number 1 in Figure 1) shows a multi-axis, networked motion controller such as those provided by Galil or Delta Tau. The HCD communicates using a standard application protocol such as SSH or HTTP over TCP/IP with a task-focused API. TMT plans on providing support for specific motion controllers.

Vendor Provided Hardware Product (2). In case 2, an off-the-shelf product is integrated. Similar to scenario 1, the HCD provides communication with the product. The off-the-shelf

product defines the communication protocol. The TMT requirement is for Ethernet-based protocols when available, but in some situations this may not be possible. The details of the vendor product application protocol are encapsulated in the HCD. Unlike case 1, in case 2 the controlled device is a complete, purchased, of-the-shelf component. The example here is a purchased robot system for storage of masks like the one planned for the MOBIE/WFOS instrument.

In Figure 2, parts of Telescope Controls are used to demonstrate integration cases 3 and 4. These two cases show the integration of complete subsystems that may include both software and hardware.

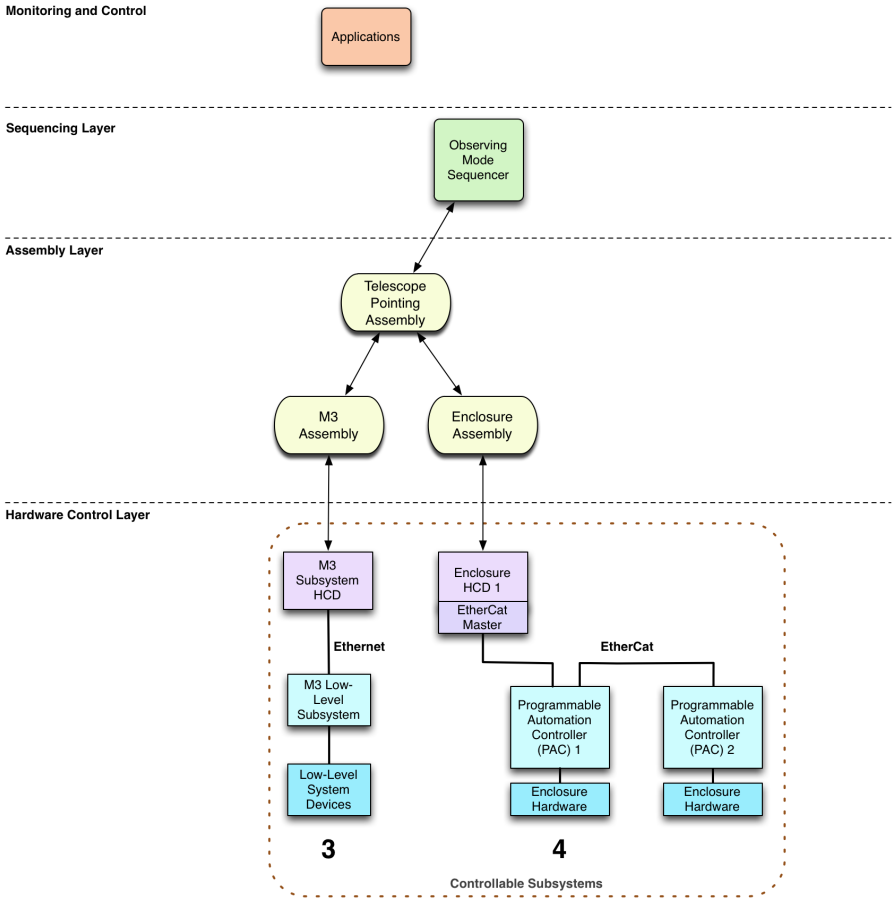

Figure 2: Integration cases 3 and 4 show the integration of entire software/hardware subsystems.

Low-level Subsystem (3). Case 3, the subject of this DRD, is the situation where a collaborator or vendor is creating an entire software/hardware subsystem for inclusion in the TMT software system. A defining characteristic of case 3 is that unique software is being written for the low-level subsystem. The subsystem is called *non-conforming* because it is not based on and does not use TMT Common Software. As with other cases, the low-level subsystem is represented by an HCD in the TMT software system. The HCD encapsulates the low-level subsystem's application protocol, which is based on TCP/IP over Ethernet. The case 3 example in Figure 2 shows the M3 control system implemented as a low-level subsystem.

Programmable Logic Controller (PLC)/Programmable Automation Controller (PAC) (4). With case 4, a software/hardware subsystem is constructed from a commercial PLC or PAC. This case is like 3, but in case 4 the software is written using tools provided by the PLC/PAC

vendor. The communication protocol is typically determined by the vendor product. The figure shows an example enclosure control system consisting of multiple PAC/PLC chassis connected via a network fieldbus like EtherCat. In this example, the protocol is specified by the use of EtherCat. TMT is working towards supporting one or more standards in this area. Example systems are National Instrument hardware or Allen-Bradley PLCs. EtherCat is a deterministic Ethernet-based fieldbus choice.

2.1 HARDWARE CONTROL DAEMONS

The OMOA Hardware Control Layer is populated with software components called Hardware Control Daemons (HCD). Each HCD manages one hardware controller, low-level subsystem, or PAC/PLC system as discussed in the previous section.

To understand the functionality required in a low-level system, it is necessary to understand the role and function of the HCD. The following are characteristics and responsibilities of HCDs.

- Each HCD acts as proxy representing the remote hardware system in the TMT software system.
- At this layer in the software each HCD and its hardware is autonomous; any required synchronization with other hardware is handled in a higher layer. In cases 3 and 4, the subsystem itself cannot depend on or control other TMT systems.
- Each HCD acts as an adapter and provides a uniform software interface and feature set focused on hardware control to the layers above.
- HCDs for multi-channel hardware controllers must support access to all channels, multiplex access to the controller and support multiple asynchronous client requests and responses.
- HCDs monitor the status and state of their hardware controllers and generate telemetry events, health and alarms in the TMT software system.
- HCDs will often convert input user or engineering units to units appropriate for the hardware.
- HCDs encapsulate the protocol and transport needed by a hardware controller. This may require vendor libraries.
- The HCD provides a way to integrate an external system. An HCD can implement a vendor-specific communication protocol allowing testing at a low level while providing uniform integration with the final system.

The HCD provides a suitable location for device simulation allowing end-to-end system testing without hardware presence (AP01). The Hardware Control Daemons are *always executing,* and each can be accessed at any time by the software layers above.

3 LOW-LEVEL SUBSYSTEM DESCRIPTION

A low-level software subsystems is one of the expected types of software systems that must be integrated into the TMT software system as described in case 3 of Figure 2.

To be classified as a low-level subsystem the software/hardware system must satisfy the following characteristics:

- The software is tightly coupled to a vendor-delivered hardware system.
- The functionality of the software is limited in scope and may have limited capabilities.
- The system and software must be a standalone product with no dependencies on other TMT systems.
- The software is delivered as a final product with the subsystem.
- The software is stable and is not expected to change over the lifetime of the project. Or changes will be infrequent requiring considerable observatory resources and planning.

The following points are typically true for software for a low-level system.

- The low-level software is not using TMT Common Software.
- The low-level software may be built using tools and languages that are not part of the TMT standards.
- The low-level software subsystem communicates with the TMT software system through a protocol that created by and is unique to the low-level subsystem.

A low-level subsystem, as described in scenario 3 in Section 2, is unique in that it is the only scenario where the communication protocol between the low-level system and the HCD is a design decision for the subsystem. In the other three cases, the application protocol is established by the purchased vendor product or TMT. It is the only scenario where a software component is designed without the benefit of the TMT Common Software.

Due to the uniqueness of case 3, special requirements are required.

While low-level software subsystems are sometimes necessary, the low-level subsystem integration approach is not the preferred TMT approach.

3.1 EXTERNAL INTERFACES

The context of a low-level software subsystem is shown in the following figure. The low-level software subsystem is at the center of the figure.

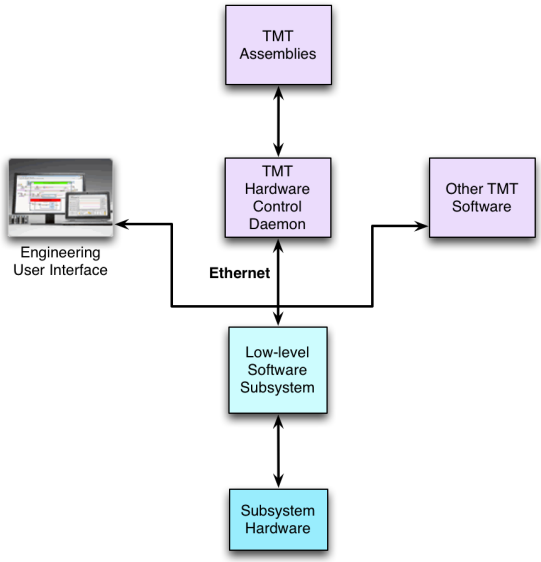

Figure 3: Low-level software subsystem context diagram

The low-level software subsystem communicates via an application protocol with a TMT Hardware Control Daemon. Other TMT software may also communicate directly with the low-level software subsystem.

In most all cases, the low-level software subsystem communicates with hardware via hardware-specific protocols.

An Engineering User Interface or Other TMT Software may connect directly to the low-level software subsystem. The communication network is Ethernet. The Engineering User Interface must understand and use the low-level subsystem protocol.

3.2 TERMINOLOGY

- For the purposes of this document it will be assumed that the HCD or other TMT software systems exchange *messages*.
- The set of capabilities supported by the low-level system is called its *Application Programming Interface (API).*

The phrase *TMT software systems* shall be used to refer to TMT produced programs that interact with the low-level subsystem using its API to exchange messages.

A TMT software system software component exchanging messages with the low-level system is called a *client*.

4 REQUIREMENTS FOR LOW-LEVEL SUBSYSTEMS

This section includes requirements on low-level subsystem software.

The requirements of this document pertain only to low-level subsystems (case 3 of Figure 2**).**

In the following, a low-level subsystem or low-level software subsystem is referred to as LLS.

4.1 SYSTEM CONSTRAINTS

This section includes requirements that constrain the low-level software subsystem and protocol.

[REQ-3-OSW-LLS-0010] The LLS software should be targeted for deployment on standard TMT computer hardware. Specifics of this hardware are TBD.

Discussion: Some systems may need unique computer equipment. If necessary this should be demonstrated during system reviews.

[REQ-3-OSW-LLS-0020] The LLS software should be targeted for deployment on the standard TMT operating system. The current requirement is for a Unix-like operating system (e.g., Linux or Solaris)

Discussion: Details of TMT standard operating systems are TBD, but the Unix-like requirement will be retained. Most TMT systems do not require a real-time operating system. TMT will standardize on a solution for real-time systems.

[REQ-3-OSW-LLS-0030] The LLS shall be compatible with TMT network standards.

Discussion: *Details of the TMT network standards are TBD. The baseline is for 10 GbE.*

[REQ-3-OSW-LLS-0040] The LLS shall be compatible with TMT environmental and survival standards and requirements (see [REQ-1-ORD-1050] through [REQ-1-ORD-1550]).

[REQ-3-OSW-LLS-0050] The LLS shall not use nor depend on any version of the Windows operating system for its intended use during TMT operations.

Discussion: In some cases a Windows operating system may be used to develop a low-level system and this is acceptable, but it can not be necessary to run the software during operations.

4.2 TIME ACCESS REQUIREMENTS

The following requirements pertain to low-level subsystems that require and use accurate and precise time.

[REQ-3-OSW-LLS-0100] The LLS shall use the TMT network Time Bus for access to precision time. For the most precise time, the LLS shall use the standard, supported TMT Time Bus card. The TMT Time Bus is based on Precision Time Protocol, IEEE-1588-2002.

[REQ-3-OSW-LLS-0110] The LLS shall be fully compatible and operate properly during any leap-second events without requiring human intervention.

4.3 SOFTWARE SUBSYSTEM DESIGN REQUIREMENTS

The design of the low-level software system must follow the requirements in this section.

4.3.1 Lifecycle Design Requirements

[REQ-3-OSW-LLS-0200] The LLS computing system shall be able to go from the power off state to being fully-operational and able to receive and execute commands from the TMT software systems without human involvement.

[REQ-3-OSW-LLS-0210] The LLS computing system shall be able to go from the power off state to being fully-operational and able to receive and execute commands in 30 seconds or less.

[REQ-3-OSW-LLS-0220] It shall be possible for TMT software systems to remotely remove power from the LLS computer.

Discussion: To enable this, the LLS may require remotely accessible power control.

4.3.2 Hardware Requirements

[REQ-3-OSW-LLS-0300] The LLS should use networked hardware controllers and avoid using board-based controller boards unless absolutely necessary.

Discussion: The LLS is acting in the same role as the HCD in scenario 1 of Figure 1. Reliance on specific boards makes long-term maintenance of the software and hardware system more difficult.

[REQ-3-OSW-LLS-0310] The communication between the TMT software systems and the LLS shall be based on Ethernet technologies.

4.4 APPLICATION PROTOCOL REQUIREMENTS

The requirements in this section pertain to the design and implementation of the protocol used to exchange messages between a TMT client program and the LLS.

[REQ-3-OSW-LLS-0400] Protocol design choices must be made in collaboration with TMT and TMT must authorize protocol choices.

[REQ-3-OSW-LLS-0405] The communication protocol between the TMT software systems and the LLS shall be based on TCP/IP.

[REQ-3-OSW-LLS-0410] The communication protocol shall provide a periodic heartbeat at 1Hz that minimally demonstrates the LLS is online and operating properly.

[REQ-3-OSW-LLS-0415] The communication protocol must support connection and reconnection without human involvement.

[REQ-3-OSW-LLS-0420] Connection and reconnection to the system should not impact the system's proper operation.

[REQ-3-OSW-LLS-0425] The protocol between the TMT software systems and the LLS shall properly support multiple, concurrent messages between the LLS and one or more clients.

[REQ-3-OSW-LLS-0430] The protocol between the TMT software systems and the LLS shall be bi-directional allowing messages to flow in both directions simultaneously.

[REQ-3-OSW-LLS-0435] The LLS shall acknowledge that commands have been received with a positive or negative acknowledgment.

[REQ-3-OSW-LLS-0440] The protocol between the TMT software systems and the LLS shall not depend on polling by the client for proper operation.

[REQ-3-OSW-LLS-0445] The LLS protocol shall not limit the functionality of the LLS, the number of simultaneous clients, nor number or types of messages that can pass between the TMT software systems and the low-level subsystem.

[REQ-3-OSW-LLS-0450] All numeric values used in the protocol should be of the same type (e.g., double, integer, etc.) if possible to minimize conversion errors.

[REQ-3-OSW-LLS-0455] The protocol should be based on available, well-tested communication libraries. Under no circumstances should the subsystem protocol be written from scratch based on low-level operating system calls (i.e., sockets).

Discussion: It is not necessary to make choices on implementation libraries in 2012. But in 2012, library examples that satisfy 0445 are: ZeroMQ (http://www.zeromq.org/) or an implementation of AMQP (http://www.amqp) such as RabbitMQ (http://www.rabbitmq.com). A REST-based HTTP protocol is appropriate for some systems.

[REQ-3-OSW-LLS-0460] The LLS developer shall deliver a client side library that allows full use of the LLS API.

Discussion: The client library is used during testing and during operations.

4.5 SUBSYSTEM FUNCTIONAL REQUIREMENTS

The following are requirements on the functionality of the low-level subsystem and the API used by TMT client software to interact with the low-level subsystem.

4.5.1 Lifecycle Functional Requirements

[REQ-3-OSW-LLS-0600] It shall be possible for TMT software systems to **halt** the LLS software using the LLS API.

Discussion: For the this requirement, halt means safely stop all LLS software while leaving the operating system running. (It may be necessary for some systems to safely park associated hardware prior to halting LLS software.)

[REQ-3-OSW-LLS-0610] It shall be possible for TMT software systems to **shutdown** the LLS using the LLS API.

Discussion: For the this requirement, shutdown means safely park any associated hardware, safely halt all software, and then power down the computer. (Shutdown of the OS may also be necessary.)

[REQ-3-OSW-LLS-0620] The LLS shutdown task should take no more than 30 seconds.

[REQ-3-OSW-LLS-0630] It shall be possible for TMT software systems to **soft restart** the LLS software system using the LLS API.

Discussion: For the this requirement, it is necessary to halt the LLS software and restart the software through initialization until ready for use. The OS is not stopped and the hardware is not power cycled.

[REQ-3-OSW-LLS-0635] The LLS soft restart task should take no more than 30 seconds.

[REQ-3-OSW-LLS-0640] It shall be possible for TMT software systems to remotely remove power from the LLS computer using the LLS API.

Discussion: To enable this, the LLS may require remotely accessible power control. This may happen without halting the software. It is assumed this will not park hardware nor halt the LLS software.

[REQ-3-OSW-LLS-0650] It shall be possible for TMT software systems to **hard restart** the LLS using the LLS API.

Discussion: For the this requirement, hard restart means park hardware, shutdown software, shutdown the computer, remove power from the computer, and reapply power to the computer causing the computer to enter the start/initialization process.

[REQ-3-OSW-LLS-0660] The LLS hard restart task should take no more than 60 seconds.

4.5.2 Configuration Functional Requirements

[REQ-3-OSW-LLS-0700] The LLS software shall be easily configurable to run on any properly configured TMT computer system. Any configuration of the LLS should not require changes to the source code, recompilation, or client calls through the LLS API.

Discussion: The goal here is that the software not have embedded parameters that must be changed in order to replace the computer it runs on.

[REQ-3-OSW-LLS-0710] The LLS API shall support the loading of all run-time system parameters or configuration files from the TMT software system during LLS startup and initialization.

Discussion: Many systems will require static parameters or lookup tables that are used for proper operation of the LLS. These values will be stored in TMT databases and loaded into the subsystem at any time. The API should support the ability to initialize itself with new values for parameters or configuration files written from the TMT databases.

[REQ-3-OSW-LLS-0720] The LLS API shall support the reading of all system parameters or configuration files from the TMT software system at any time for storage in TMT databases.

Discussion: It should be possible for TMT software to gather all operational configuration parameters needed for run-time operation in order to store them in the Configuration Service.

4.5.3 Command Functional Requirements

In the following *normal operations* indicates that the system is fully initialized and operating as it is designed to operate.

[REQ-3-OSW-LLS-0800] The LLS shall be available to receive commands from TMT software systems at all times when it is in the normal operational state.

[REQ-3-OSW-LLS-0810] The LLS shall support receiving commands from multiple TMT software systems when in normal operation.

Discussion: The goal is to ensure that an Engineering User Interface program, a logging program and the TCS HCD can all operate at the same time.

[REQ-3-OSW-LLS-0820] The LLS should return command errors to the TMT software systems.

[REQ-3-OSW-LLS-0830] It shall be possible for the TMT software systems to set and get all public writable subsystem parameters using the LLS API.

Discussion: Here parameters means any values that describe the state or status of the subsystem or any hardware it controls.

[REQ-3-OSW-LLS-0840] It shall be possible for the TMT software systems to get values for any public read-only subsystem parameter using the LLS API. The API should support getting 1, several, or all values with a single API call.

[REQ-3-OSW-LLS-0850] It shall be possible for a TMT software system to determine if actions (such as hardware activities) due to commands are executing or completed using the LLS API.

4.5.4 Status Functional Requirements

Status pertains to values that describe the state of the low-level subsystem or hardware it controls or monitors. The subsystem state consists of one or more status items.

[REQ-3-OSW-LLS-0900] The LLS shall monitor the values of important parameters and make them available as status items.

[REQ-3-OSW-LLS-0910] The LLS shall allow one or more status items to be retrieved in a single command from a TMT software system using the LLS API.

Discussion: A publish/subscribe implementation is also a possibility for notifying clients of status changes.

[REQ-3-OSW-LLS-0920] The LLS status retrieval response should minimally indicate the status item name, value and units for each status item retrieved.

[REQ-3-OSW-LLS-0930] The LLS shall provide status that allows a TMT software system to determine if any controlled hardware is in motion or idle.

[REQ-3-OSW-LLS-0940] The LLS shall provide a heartbeat status item to any client that notifies the client that it is able to communicate at the rate of 1Hz.

4.5.5 Logging Functional Requirements

[REQ-3-OSW-LLS-1000] The LLS API shall allow the option of logging subsets of low-level subsystem status or state parameters.

[REQ-3-OSW-LLS-1010] The LLS API shall allow logging information to pass from the low level subsystem to a TMT software system.

4.6 USER INTERFACE REQUIREMENTS

[REQ-3-OSW-LLS-1100] The LLS shall provide a user interface that allows full engineering-level monitoring and control of the LLS. This user interface shall be called the *Engineering User Interface* (EUI).

[REQ-3-OSW-LLS-1110] The LLS EUI shall be implemented as a Graphical User Interface.

[REQ-3-OSW-LLS-1120] The LLS EUI shall use the LLS-provided client library to communicate with the LLS using the LLS protocol.

[REQ-3-OSW-LLS-1130] The LLS EUI shall be usable from any computer that has access to the LLS through the TMT network.

[REQ-3-OSW-LLS-1140] The LLS EUI shall be usable at any time including during normal, operational use of the LLS.

[REQ-3-OSW-LLS-1150] The LLS EUI shall allow all necessary engineering-oriented control of the LLS including shutdown, startup, restart (both soft and hard), and initialization.

[REQ-3-OSW-LLS-1160] The use of the LLS EUI shall not impact performance of the system at any time.

[REQ-3-OSW-LLS-1170] The LLS EUI shall use TMT user interface standards unless they are inadequate.

Discussion: It may be possible for the LLS to use all or a subset of the TMT user interface standards. For instance, it may only be possible to use the recommended toolkit. TMT user interface standards are not yet defined.

5 SOFTWARE DEVELOPMENT AND TESTING

Low-level subsystem software is part of the TMT software system and must follow the processes and testing strategies of other software projects that are part of TMT.

5.1 SOFTWARE DEVELOPMENT PROCESS

[REQ-3-OSW-LLS-2000] The LLS software development process must comply with the TMT Software Management Plan (AP02).

[REQ-3-OSW-LLS-2010] The LLS software development process must comply with the TMT Software Development Process (AP04).

[REQ-3-OSW-LLS-2230] The LLS must follow the TMT software safety process (AP05).

5.2 SOFTWARE TESTING AND SIMULATION

[REQ-3-OSW-LLS-2100] Software testing of the LLS must follow the approaches of AP01.

[REQ-3-OSW-LLS-2110] The LLS must provide a suite of unit tests that test the components within low-level subsystem software system.

[REQ-3-OSW-LLS-2115] The LLS must provide a full suite of acceptance tests to test public functionality of the LLS using the client library.

[REQ-3-OSW-LLS-2120] The LLS must provide a suitable simulation mode controllable through the LLS API that allows specific components within the subsystem to be simulated as well as the entire LLS.

Discussion: Simulation means that the low-level software system appears to be operating properly even if the controllable hardware is not present or broken (see AP01).

6 DOCUMENTATION AND REVIEWS

Low-level systems are required to provide the same design documentation as all TMT software projects. Documentation is tied to the review process.

[REQ-3-OSW-LLS-2200] The LLS must provide documentation and participate in reviews according to the TMT review process and review deliverables for software (AP06).

[REQ-3-OSW-LLS-2210] The LLS must provide all source code, any build system and other files for the subsystem. Any special development tools required to build the software must be provided upon delivery.

[REQ-3-OSW-LLS-2220] The LLS communication protocol must be fully documented using formal methods such as a Backus-Naur Form (BNF).

7 GUIDELINES AND RECOMMENDATIONS

The following are recommendations for designers of low-level software systems. While not requirements at this time, if these guidelines are not followed, it is necessary to make a very good case during reviews for not following them during the review process.

- Any libraries used in the LLS should be based on open-source over off-the-shelf software. All libraries must include source code.

7.1 USER INTERFACE STANDARDS

TMT will define standards for operations user interfaces. These standards will be supported by the TMT software engineering support team. These standards will include toolkits and platforms that satisfy the TMT requirements for remote use and standardized look.

At this time, these standards are TBD. User interface approaches change rapidly and it is too early in the project to make decisions.

User interfaces may be browser-based. Browser-based tools are the suggested approach to be used for planning purposes at this time.