Small Satellite Research Laboratory
Franklin College of Arts and Sciences
UNIVERSITY OF GEORGIA

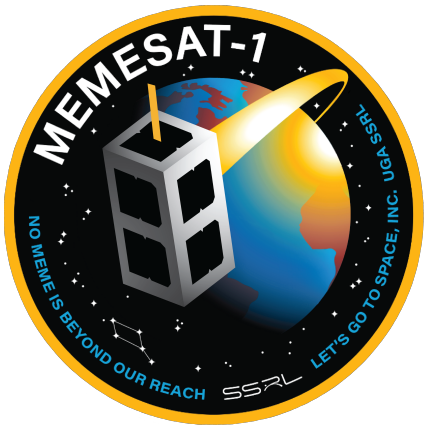
Software Architecture Documentation

Prepared for
The Small Satellite Research Laboratory
Fall 2024 - Spring 2025

Principal Investigator
Dr. Deepak Mishra

The University of Georgia
Geography-Geology Department
Athens, Georgia
United States of America

Prepared by
The MEMESat-1 CDH Team
University of Georgia
Athens, Georgia
United States of America



Contents

1 List of Acronyms

BBS Bulletin Board System
CDH Command and Data Handling
EPS Electrical Power System
GDS Ground Data System
FSM Finite State Machine
FSW Flight Software
MCU Microcontroller Unit
MS-1 MEMESat-1
OBC On-Board Computer
OSAL Operating System Abstraction Layer
OTA Over-The-Air
OTAU Over-The-Air Updater
TLM Telemetry
UART Universal Asynchronous Receiver-Transmitter
UHF Ultra High Frequency

2 Revision History

Changes	**Authors**	**Version**
[2022-02-22] Removed cFS content and planned F Prime sections.	Starks, Michael	0.0.0
[2023-02-23] Added F Prime File Manager section	Mallory, Reese	0.0.1
[2022-02-23] Wrote System Manager and Radio components	Starks, Michael	0.1.0
[2022-02-23] Elaborated on File Manager	Mallory, Reese	0.1.1
[2022-02-24] Added Ground Station section	Chitgopkar, Sudhan	0.1.2
[2022-03-09] Added caption to images. Added small build section	Merchant, Saba Starks, Michael	0.2.0
[2022-10-31] Updated document to reflect changes to scheduler and addition of TlmChan	Starks, Michael Hammond, Aiden	0.3.0

Changes	Authors	Version
[2022-10-31] Approved for CDR	Lassiter, Caroline	1.0.0
[2023-09-11] Updating for IMR-2	Beattie, Olivia	1.1.0
[2024-03-18] Updated to reflect revised data flow requirements. Cleaned up document for feasibility review.	Beattie, Olivia Lemus, Samuel	1.2.0
[2024-04-05] Approved for Feasibility	Garon, Isaac	2.0.0
[2025-04-05] Reviewing for IMR-3	Lemus, Samuel	2.1.0

3 Purpose

This document describes the Software Architecture for the MEMESat-1 mission. This document details the high-level functionality of the F' Software Suite.

4 Glossary

General

1. Apps - Discrete applications that perform a specific function.
2. Topology - The collection of components that comprise the flight software.
3. Hardware Component - Software representation of a piece of hardware.
4. Subsystem - Groupings of components that form a discrete logic unit responsible for a certain area of satellite behavior (EPS, COMM, ADCS). These can be thought of as machines for the purpose of logical abstraction.
5. System - MEMESat-1.

Operational States

6. Mode - The highest operational state of our satellite. Each mode has a unique goal for the satellite to achieve. Transitions between modes are defined by the state of the satellite's power and the presence of anomalies.

Logical Abstraction

7. Black Box - A phrase used in computer science to describe a device, system, or object which is viewed only in terms of its inputs and outputs. With black boxes the internal implementation is irrelevant.
8. Interface - Interfaces go hand in hand with black boxes. They are a defined way of interacting with a device, system, or object and can be thought of as the wall we build around our box to make it opaque. As long as an implementation fulfills the requirements of its interface from an outside perspective that implementation becomes a black box.

5 Components and Decentralized Structure

The architecture is based on NASA's Fprime (F') software framework. A detailed explanation of the F' architecture can be found here.

5.1 A Brief on Software Systems Architecture

The software system is designed to break down the software into modules. Each module provides separation of function, the definition of interfaces, behavioral characteristics, testing at the unit level, and ownership. These modules are also known as components. All external functions of the component are defined as an interface for interacting with the component, the external-facing interface is also known as a port. The system structure allows for separation logic that a module only interacts with the layer below it.

5.2 Components

Each component contains a discrete portion of the system's logic. The component architecture implies usage patterns, as well as usage constraints. Components encapsulate behavior - they are localized to one compute context interacting with other components through ports.

Ports are the point of interconnection between Components and encapsulate typed interfaces in the architecture. Each port is defined in a specific type and can only connect Components by defining ports of the same type. Directionality of the ports need to be specified by the user to represent where the data originates and where it goes.

5.3 Topology: F' Application

Components are instantiated at runtime and then connected through the ports to other components in the system. There should be no code dependencies between the components, only dependencies on port types.

5.4 Application Component

This layer implements mission-specific functionality. It encompasses the system logic and uses peripherals and other components to perform the detailed behavior of the application.

5.5 Manager Component

This layer focuses on controlling the peripheral by using the interface driver at the abstract level. The manager does not know how it is used or that it is used. It only knows the driver that is used to talk to its peripherals and how to translate high-level commands into driver messages.

5.6 Driver Component

This layer is responsible for the particular hardware interface. It is written to interact with only that type of device.

5.7 Application Manager Driver Pattern

The F' design pattern contains 3 layers: application, manager, and driver. Each layer defines components for only that layer's functionality.

6 Components

The following section of this document describes all of the F' components that will be on the on-board computer (OBC).

6.1 System Manager

The System Manager is a collection of components that acts as an observer, overseeing system operation. The components of the system manager house the satellite's state machine, and schedule low-frequency tasks, lower than 1 Hertz. Telemetry data will be gathered from EPS and Radio at regular intervals and stored separately from the payload in the Telemetry Database bundled with F'. The telemetry information will be used downlinked every 60 seconds to maintain accurate health information. Telemetry data will be monitored for errors to transition the satellite into safe mode until ground station operators can evaluate the state of the satellite and resume normal operation. Error logging will have six modes during operation that will be tracked separately for each system.

ACTIVITY_LO	described as miniscule info messages; these conventionally come from background tasks.
ACTIVITY_HI	can be compared to info messages; these typically come from foreground tasks.
WARNING_LO	less-severe warning events
WARNING_HI	high-severity warning events, although the system can still function
FATAL	fatal events indicating that the system **must** reboot.
COMMAND	events tracing the execution of commands.

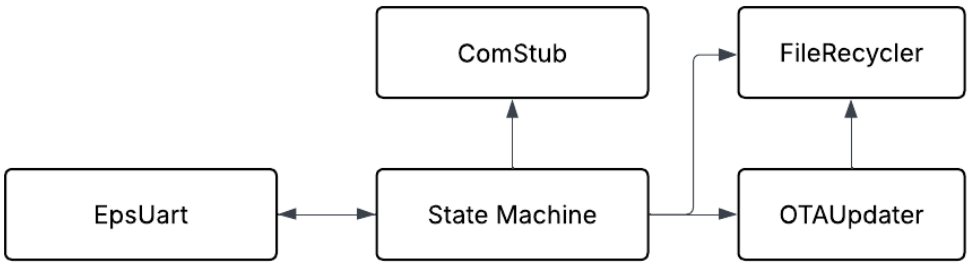
If there is a change in mode needed, the System Manager will convey the mode change to the components. The System Manager will act as the software to tie the satellite subsystems together and control the operation of the components.

6.2 State Machine

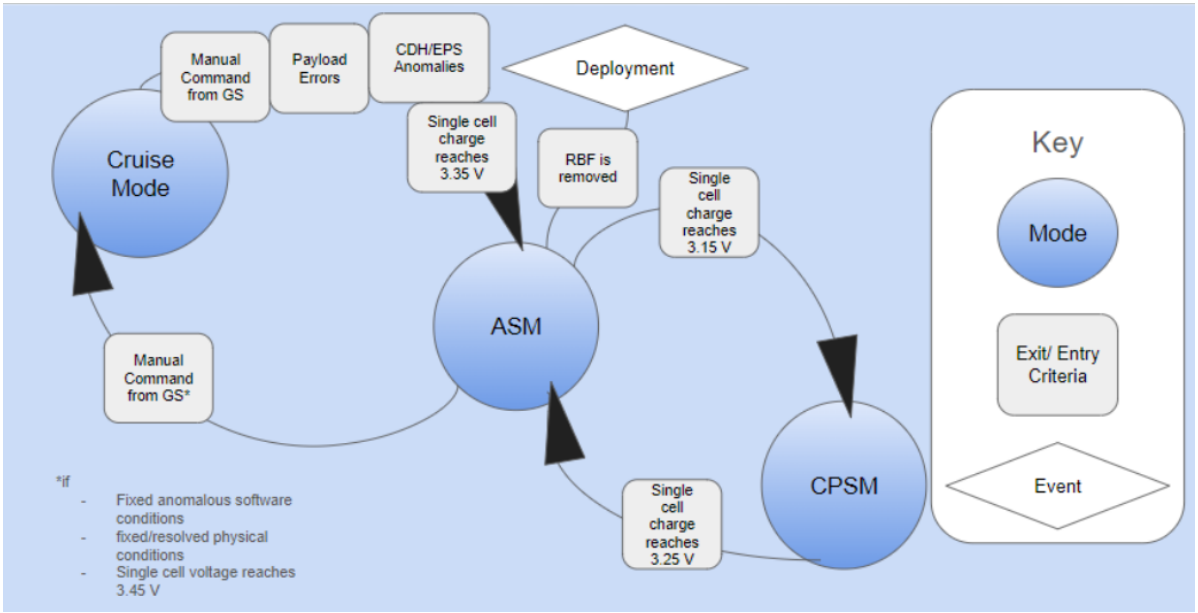
The State Machine on MEMESat-1's OBC is responsible for keeping track of the state of the system and controlling the transitions between operational modes. The state machine has

5 states: Cruise Mode (CM), Anomalous Safe Mode (ASM), Critical Power Mode (CPM), Restart, and Shutdown. The system can only enter CM from a command from the ground station; while in CM the satellite will operate in nominal conditions and will maintain all functionalities of the payload. The state machine will transition to ASM based on the criteria in the Diagram below. When the system is in ASM, communications will be limited to the ground station only, and the transmission of images and HAM radio messages will be halted. Health beacons will be sent every 60 seconds in CM and ASM. If the system goes into CPM based on the criteria in the diagram below, the OBC will shut down and the satellite will be controlled by the Electrical Power System (EPS) microcontroller until conditions allow the system to return to a stable state.

6.2.1 Block Diagram



6.2.2 FSM Diagram



6.3 Scheduler

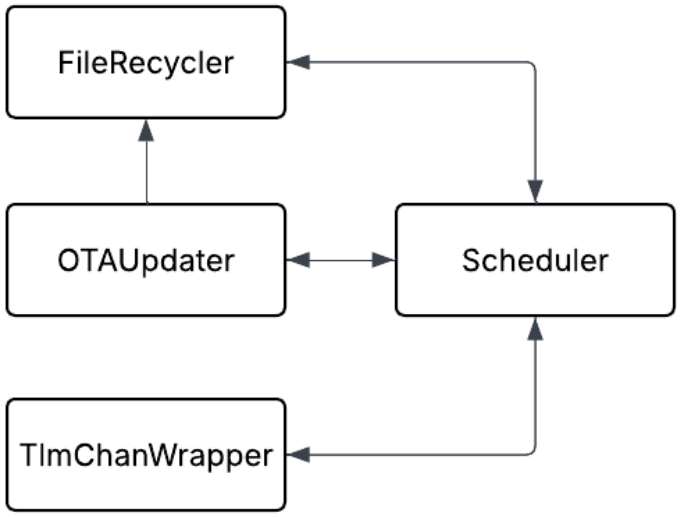
The Scheduler is the task delegator of the system; alongside the rate groups defined in F Prime architecture, the component causes actions to occur on a schedule that is defined using the

format below. The Scheduler is used to program tasks that happen at a rate slower than 1 Hz (the limit of the rate groups). The Scheduler works by using a libcron 'tick' to invoke a port connected to the desired component. Each component connected to the Scheduler is instantiated through a 'preamble' function predefined by F' to allow an active component's initialization functionality.

6.3.1 Format

```
+ ---------------------- second (0-59)
| + ---------------------- minute (0-59)
| | + ---------------------- hour (0-23)
| | | + ---------------------- day of month (1-31)
| | | | + ---------------------- month (1-12 or JAN-DEC)
| | | | | + ---------------------- day of week (0-6 or SUN-SAT)
| | | | | |
| | | | | |
* * * * * *
```

6.3.2 Block Diagram



The Scheduler is connected to the FileRecycler and TlmChanWrapper components to schedule the deletion of files and the downlink of telemetry data, respectively.

6.4 Telemetry Database

F Prime's Telemetry Channel (TlmChan) component will be used as the telemetry database for the satellite. The component stores the serialized form of telemetry values from other components and implements the storage as a table accessed by the telemetry ID. The data can be individually read back or periodically pushed to another component for transporting

out of the system. Svc::TlmChan is an implementation of the Svc::TlmStore component in the Svc/Tlm directory.

TlmChan has two alternate implementations. One performs a linear lookup to find the telemetry entry in the table based on the telemetry ID. This is more space efficient but has slower performance because the table is traversed each time. The second implementation uses the telemetry ID as an index in the table. This implementation is faster but can be space inefficient if disjointed telemetry IDs exist.

The TlmChan component has an input port that receives channel updates from other components in the system. Those components interact with an implementation class generated by F' and take an autocoded type that can be serialized and sent to the output port of the component. That output port connects TlmRecv, and values are stored in an internal double-buffered table as generic data, and a flag is set when a new value is written to the channel entry.

The telemetry database is interacted with through a telemetry channel wrapper component (TlmChanWrapper) that is connected to the Scheduler. The wrapper acts as a medium between the TlmChan component pre-defined by F Prime and the Scheduler component in order to preserve the integrity of the predefined code while also creating a small layer of abstraction with regards to our custom scheduler. The TlmChan component has a maximum number of telemetry message types that can be stored, with a default max of 50; this will be modified as development continues.

6.5 File Recycler

The File Recycler component acts as an important guard against excess data. The component makes use of the Scheduler Component in order to delete files at a determined interval that is controllable from the ground station. Commands on the ground station can be given to delete files in four separate ways: instructing the component to delete a certain number of files beginning from oldest to newest, truncating the storage to a certain number of files, deleting files created before a certain time, or deleting files created after a certain time. Each method determines file creation time via each file's time of last status changes, which only updates by writing or setting inode information1. The component will handle the deletion of data for both memes and messages from HAM radio users.

6.5.1 Block Diagram

The FileRecycler is connected to the Scheduler to ensure that the functionalities of the component can be executed on a fixed interval. The component is connected to the State Machine to receive information on when the system is in CPM; while in this mode the component should lock access to all files.

For a detailed description on the design of the File Recycler component, refer to the FileRecycler Component Overview.docx.

6.6 Eps Uart

The component is responsible for sending UART communications between the EPS microcontroller (MCU) and the OBC to relay telemetry and events about the status of the satellite. The component utilizes the packet format defined below to parse and packetize the messages sent between the OBC and MCU. When a message is received the input port checks the serial receive status for an error; If there is no error, the component verifies the length and content of the packet before passing it into the parse function. The message is then parsed, based on the packet format described below, and the received data is distributed to the other components onboard the satellite. When a message is being sent, the data is packetized using the packet format described, and is then sent to the EPS.

The component is connected to the State Machine and uses the LinuxUARTDriver to communicate with the EPS MCU. The main function of the component is to receive data from the EPS microcontroller and use it to set system telemetry values for downlinks as Health Beacons. The State Machine component utilizes this data to monitor system health and initiate a state change if an anomaly is detected. The component is responsible for communicating changes in the state of the system of the satellite to the EPS in case of CPM.

For a detailed description on the design of the EPS_UART component, refer to the EPS_UART Documentation.

6.7 Authentication

The Authentication component is responsible for verifying all communications between the satellite and the ground station. All uplinks are authenticated to ensure that no one but the ground station can send data to the satellite, and all downlinks are authenticated to guarantee that all information sent to the ground station only comes from the satellite, untampered. Additionally, all commands sent to the satellite are encrypted to protect the satellite from being intercepted or commanded by anyone other than the ground station. The openSSL library is used for all encryption and hashing algorithms.

For a detailed description on the design of the Authentication component, refer to the Authentication Component Overview.docx.

It is of the occasional opinion that the Authentication component functions should be a subfunction of the Radio Interface component.

6.8 Storage

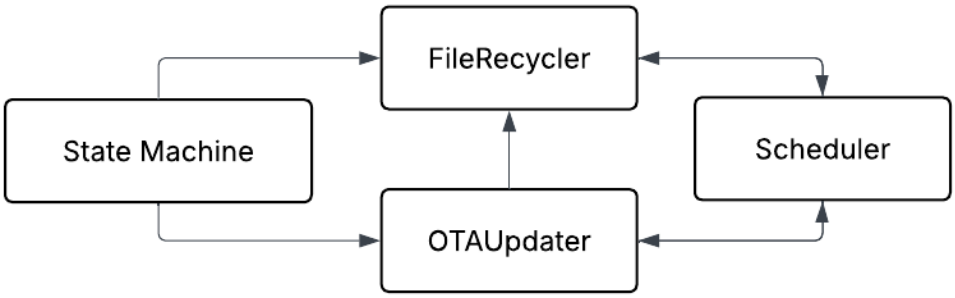
The Storage component in MEMESat-1's OBC is responsible for storing user submitted memes and messages from HAM Radio users on the satellite's file system. The component monitors the number of memes and messages stored onboard the satellite to ensure that the maximum threshold is not exceeded. Each meme or message received is designated a unique index that determines its file name. The Storage component encapsulates memes and messages as custom F' types, allowing for serialization and deserialization from raw byte buffers for file storage. The component relies on the F' Operative System Abstraction Layer (OSAL) for file system interactions, using a structured storage format with unique file names and message serialization.

6.8.1 Block Diagram

6.9 Over-The-Air Updater

The Over-The-Air Updater (OTAU) component is responsible for updating the flight software on MEMESat-1. The OTAU component will be used to update the flight software on the satellite in the event of a bug or error in the code. This component should also retain the ability to flash the STM with a new image in a scenario where this is needed. The OTAU's core responsibilities are segement file tracking and culmination for any set of data which cannot be uplinked in a single pass.

6.9.1 Block Diagram



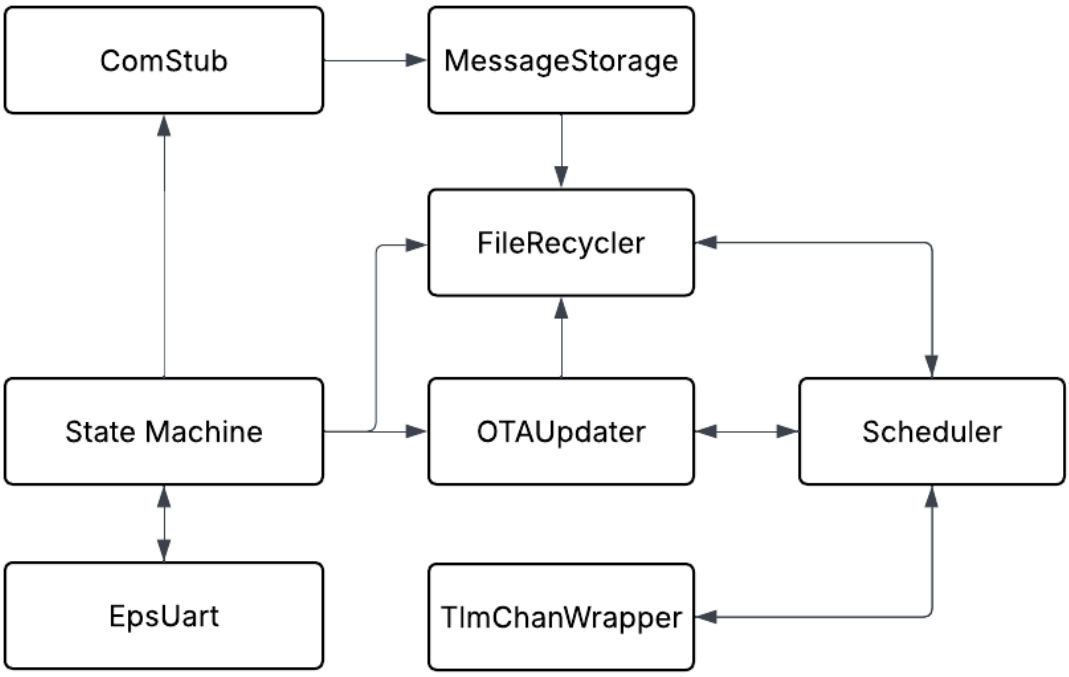
6.10 File Uplink

The FileUplink component is an active F' component that manages the uplink of files to the spacecraft. The component is responsible for receiving file packets, assembling them into files, and storing them in the on-board nonvolatile storage. For a detailed description on the functionalities and design of the component, refer to Svc::FileUplink Component.

6.11 File Downlink

The FileDownlink component is an active F´ component that manages spacecraft file downlink. This component is used by the scheduler component, which enqueues files using the SendFile port. Operators can also use this component by using the SendFile and SendPartial commands. When a file downlink initiated by a port is completed, the FileComplete port broadcasts, allowing components to detect when a previously enqueued file downlink has been completed. For a detailed description on the functionalities and design of the component, refer to Svc::FileDownlink Component.

6.11.1 Encompassing System Topology



7 Radio Subsystem

The MEMESat-1 OBC communicates with the radio over serial UART. The radio interface component will handle transferring the data between the physical radio hardware and the flight software. Incoming data will be parsed and sent to the respective component; outgoing data will be framed for transmission and sent to the radio buffer to be downlinked to the ground station. The radio subsystem will be responsible for framing the data before transmission. All data transmission will use CCSDS for framing ensuring compatibility with COSMO. F Prime supports two types of uplinks: command and raw. Private Com uplinks will contain commands for the flight system, while unencrypted raw uplinks will be payload data. Command uplinks will be sent to the command dispatcher. Raw uplinks will be sent to the file manager for appropriate storage.

7.1 Radio Interface

The radio interface intends to be a software defined component within F Prime that interacts with the physical radio hardware by inserting commands into the radio serial buffer. The radio interface then should pass the raw data received to the Packet Storage Component to handle successive state changes.

8 Ground Data System (GDS)

The Ground Data System allows operators to control and monitor their satellite's flight software, and acts as a medium for data transmission from the ground to the satellite (and vice versa). The default software provided by JPL, written in JavaScript and Python, works out of the box and requires proper GDS and port specifications. The software, which can be controlled via GUI or CLI, allows for multiple action types:

Commanding - The command list is generated using the deployment dictionary on start up. This should define all possible commands to the satellite, the number of arguments each command takes, and the argument type. After this has been sufficiently defined, it is simple to use the GUI to select a particular dictionary and command, which will be "." delimited. The GUI keeps command execution simple after a command has been defined, using a button for execution and color-coding successful and unsuccessful command executions.

Channel Handling - The GUI also displays a table of all received telemetry data. Importantly, this table only contains cells for which data exists and has been propagated. By default, any telemetry that does not have any associated data is left entirely invisible. Though this may be changed in settings, this is an important aspect of the GDS to note.

Event Handling - The Events pane of the GUI acts as a means of viewing all previously executed commands. Importantly, "Events are sent out of the system via the Svc::ActiveLogger component and components defining commands should hook up the log port to it. If console logging is desired, the text log port can be hooked up to the Svc::PassiveConsoleTextLogger component." Events are also color-coded based on severity, as follows:

Severity	Color	Description
DIAGNOSTIC		Debug events not typically send to the GDS
COMMAND	GREEN	Events produced by the command dispatcher to aid in tracing actual command execution.
ACTIVITY_LO	GRAY	Low priority informational events typically tracking background process actions.
ACTIVITY_HI	BLUE	High priority informational events typically tracking ground-commanded foreground actions
WARNING_LO	YELLOW	Low priority non-critical warning events.
WARNING_HI	ORANGE	High priority critical warning events.
FATAL	RED	Critical failure event typically resulting in embedded system restart.

Uplinking - Uplinking is "dependent on the usage of the FileUplink components and a file system implementation", and the process broadly consists of (1) staging and (2) uplink. Files are uploaded to a staging area before final uplink. Upon uplink, all relevant files are added to the outgoing queue, and uplinking begins. During uplink, progress can be monitored for any unforeseen bugs and problems. Importantly, a 32mb file size limit exists on uplinking.

Downlinking - Files are downlinked using the fileDownlink.FileDownlink_SendFile command. During downlink, users may track packet status (via progress bar), with local download available after successful downlink completion.

Log viewing - There are a variety of logs F' displays to users in the GDS GUI, defined per their documentation below:

- ThreadedTCP.log: log from the GDS middleware server linking comm to the GDS actual
- channel.log: log of all channels received by the GDS
- command.log: log of all commands sent by the GDS
- event.log: log of all events received by the GDS

9 Build System

9.1 F Prime

F Prime has a dedicated build tool to reduce the complexity of the build process. Before development begins, the project's auto-coded files must be generated. Once the build structure is generated, the user can build the project using the build command of fprime-util. Users can also build unit tests by passing the "–ut" flag during the build process. Until the mission topology is solidified, please defer to the F' documentation for instructions on how to build F' topologies.

9.2 Buildroot

The FSW will run on a minimal Linux kernel based on the Debian OS from the Raspberry Pi Foundation. The OS will be created using Buildroot to get all of the necessary functionalities (U-boot as the bootloader, C++ support, GDB for debugging). The OS will be stored in the first partition of the eMMC storage, and the rest of the module will be allocated to storage partitions for memes and messages from HAM radio users.

9.3 Zephyr RTOS

The EPS MCU firmware will be built using the Zephyr RTOS framework. The Zephyr RTOS is a small, scalable, and open-source real-time operating system designed for resource-constrained devices. It provides a lightweight kernel, device drivers, and networking stacks that are suitable for embedded systems. The firmware will be developed in C and will utilize the Zephyr APIs for hardware abstraction and peripheral control.