cFS-based Mission Flight Software Workbook

These slides are a collection of notes that are slowly evolving with each OSK release into a complete workbook

OSK v3.2

Introduction (1 of 2)

OSK and this workbook can help accelerate and optimize the application development process, but it can not replace the rigor that is required to specify, design, develop, and verify robust FSW. FSW development and verification is a complex endeavor. NASA's cFS has been for "spacecraft bus" FSW on missions ranging from the 6U Dellingr CubeSat** to the 4-ton Global Precipitation Measurement (GPM) spacecraft.

Dellingr

GPM

From an app perspective, Dellingr has 17 apps (8 cFS community, 9 mission-specific) and GPM has 25 apps (11 cFS community, 14 mission-specific). There's a lot more to FSW than just apps such as device drivers, libraries, etc., but apps implement most of a mission's functional requirements and regardless of a mission's physical size, there is usually a substantial FSW application development effort. This workbook takes a FSW systems engineering approach towards defining the apps required to satisfy your mission requirements and designing an execution model for how they will work together as a system.

**SmallSat Conference online proceeding search for Dellingr:
https://digitalcommons.usu.edu/do/search/?q=dellingr&start=0&context=2746912&facet=

Introduction (2 of 2)

The core Flight Executive(cFE) is a general-purpose embedded system framework that can host apps to provide a cFS mission solution and there is a very wide range of mission architectures. In fact, the mission may not even be a space mission. This workbook targets using the cFS for spacecraft bus FSW. Even this narrower focus has considerable variability. For example, does the mission require onboard attitude determination and control (ADC)? If so, are the ADC algorithms being developed and implemented as cFS apps inhouse or is an ADC component with sensors and actuators being procured, in which case cFS apps will need to be written that interface with the ADC component.

Workbook Outline

The following outline describes the order and purpose of each workbook section:

1. **OSK-cFS System Engineering Development Processes**
   - Defines a mission FSW engineering context
   - Defines a generic spacecraft app model, a SimpleSat reference app model, and processes for using these models to develop your mission app model
2. **Apply OSK-cFS Process to an example mission**
   - Uses the models and processes defined in section 1 to develop an app model for an example mission
3. **Tune, Verify, and Validate Your System**
   - Describes cFS, COSMOS, and OSK tools that are available to help you tune, verify, and validate your system
4. **Attitude Determination and Control with the 42 Simulator**
   - Describes FSW engineering processes for developing algorithms using 42, porting the algorithms to cFS apps, and running a closed-loop system

OSK-cFS System Engineering Development Processes

The next two slides show a FSW mission lifecycle context and a cFS software engineering context.

FSW Mission FSW Context

Project & Systems

- Ops Concept
- Requirement
  - Analysis,
  - Decomposition
  - Allocation
- Build vs Buy

Subsystems

- Refine FSW Requirements
- Develop Algorithms
- Procure HW Components
- Develop Inhouse HW Components
- Select & Procure Processor Board

FSW Development

FSW-cFS Systems Engineering

Integrate & Verify FSW

Select & Configure cFS Community Apps

Develop Custom cFS Apps

Develop Custom Platform Software

Port cFE to Processor/OS

Spacecraft Integration & Test

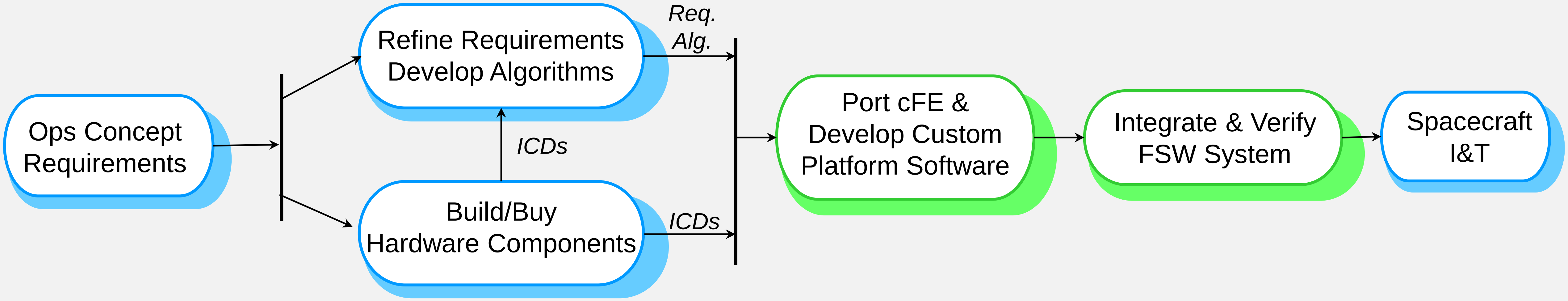
Spacecraft Operations

Supported by OSK

FSW Mission FSW Context

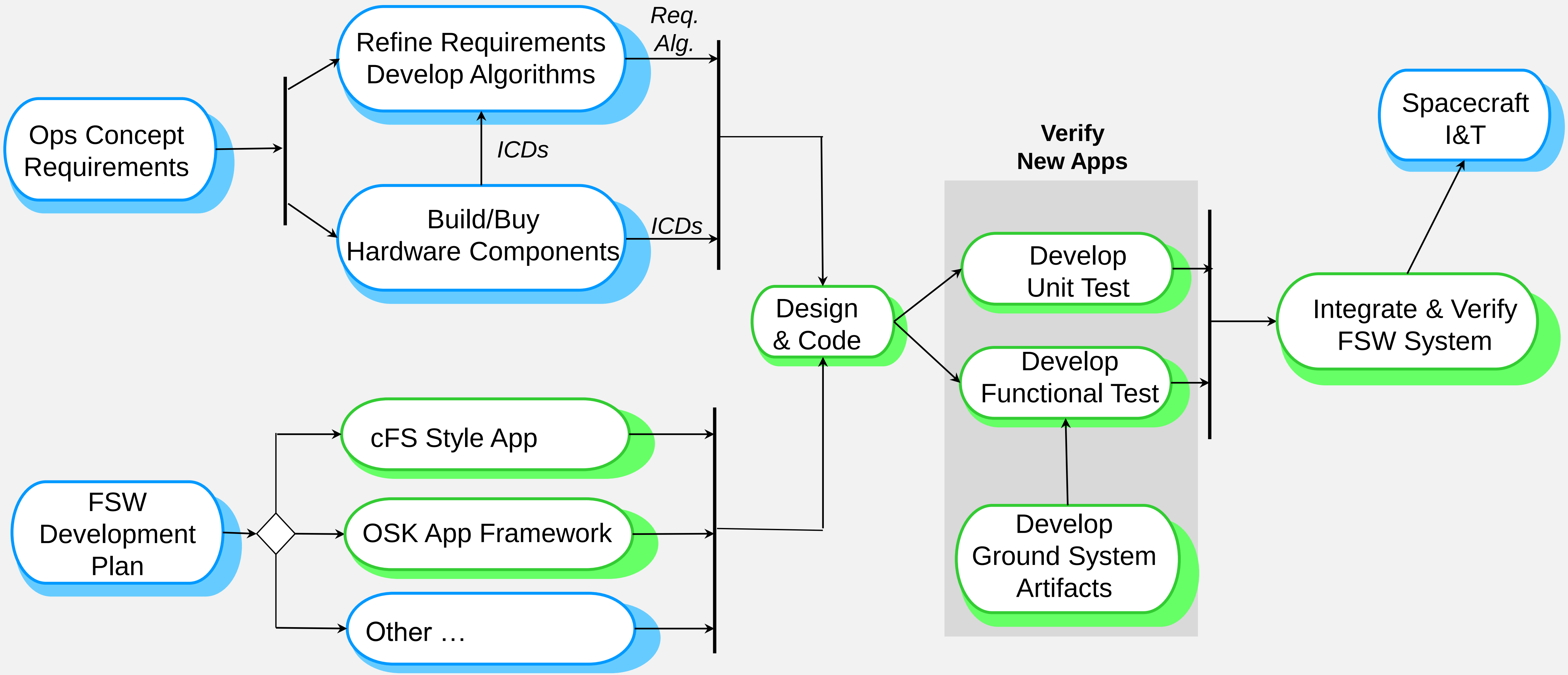
- OSK supports integrating and testing cFS community apps, OSK apps, and user mission-specific apps into a functional FSW system that runs within OSK's software-in-the-loop (SIL) environment
- Nothing precludes OSK from being used in later mission lifecycle phases, however, creating the hardware in-the-loop (HIL) interfaces, developing simulators, and migrating ground system artifacts (if COSMOS is not used) are not covered by OSK.
- These efforts are represented by the gray arrows. The green arrow pointing to the processor card is not within the OSK boundaries because porting the cFE to a hardware platform is not directly covered by OSK, however, a cFS community platform list https://github.com/OpenSatKit/cfs-platform-list is maintained by the OSK project and provides links to cFS porting resources.
- OSK is not required to develop cFS apps, however, note the following
  - You will eventually need a ground interface and test script environment
  - You can leave OSK's SimSat environment and develop new apps in a new mission or target or use OSK's Sandbox target

Porting the cFE and Developing Platform Software



- **OSK does not currently support platform development activities**
- **Preliminary plans to create a Code-As-You-Go (CAYG) cFE porting tutorials**
  - YouTube video with git repos

Developing Custom Apps



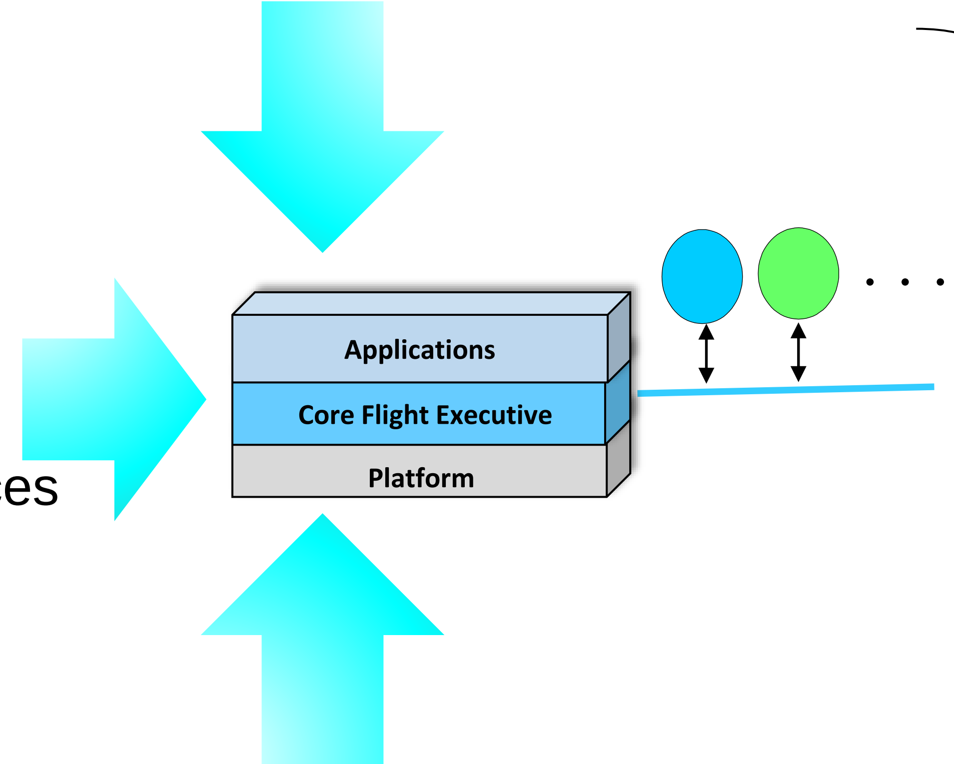
Developing New Apps

- **OSK app generator creates “hello world” app source code and COSMOS command & telemetry definition files with templates for**
  1. cFS User’s Guide Style App
     - Design and code according to the cFS Developer Guide
  2. OSK Framework App
     - Adhere to cFS API principles but design according to the OSK framework design patterns and coding idioms
- **Initial app development often done independent of COSMOS**
  – “Hello world” tool useful for learning, but often useful to start with an existing app that has a top-level design close to your needs
  – 42 Interface (I42) and 42 FSW Controller (F42) serve as high level example
- **Verify New Apps**
  – OSK does not augment cFS’ unit testing framework
  – OSK supports functional app testing and system integration testing using COSMOS Script Runner and Test Runner tools
    - Thin integration and functional test infrastructure built on top of COSMOS script API
    - Minimal example tests, more planned
  – Ground tools written as needed

cFS Software Engineering

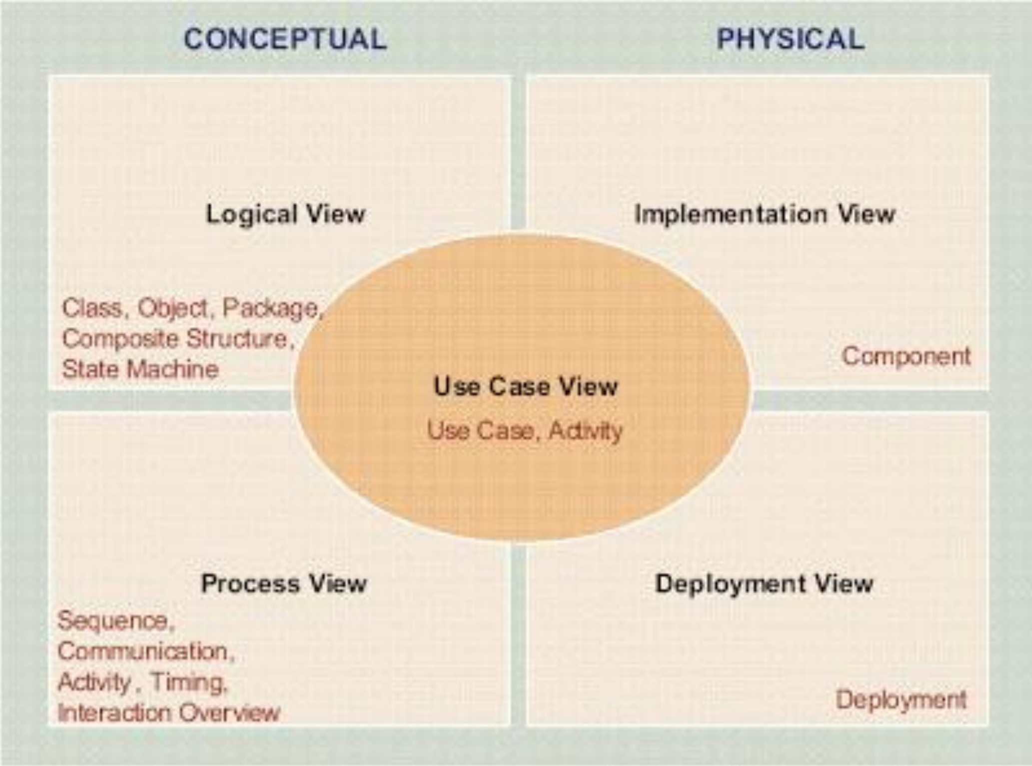
- Mission/Ops Concept
- Mission Requirements

- cFS build config
- cFS app config
- New app interfaces



- Hardware Architecture
- Operating System, Board Support, Device Drivers
- Interface Control Documents (ICDs)

4+1 View UML Diagrams [1]



Use a combination of custom notation and UML diagrams

Logical View

- cFS layered architecture
- cFS application model
- OSK object-based design

Process View

- Sequence diagrams with apps and messages
- Scheduler app-based design

Implementation View

- cFS mission tree structure and build environment
- ES startup script

Deployment View

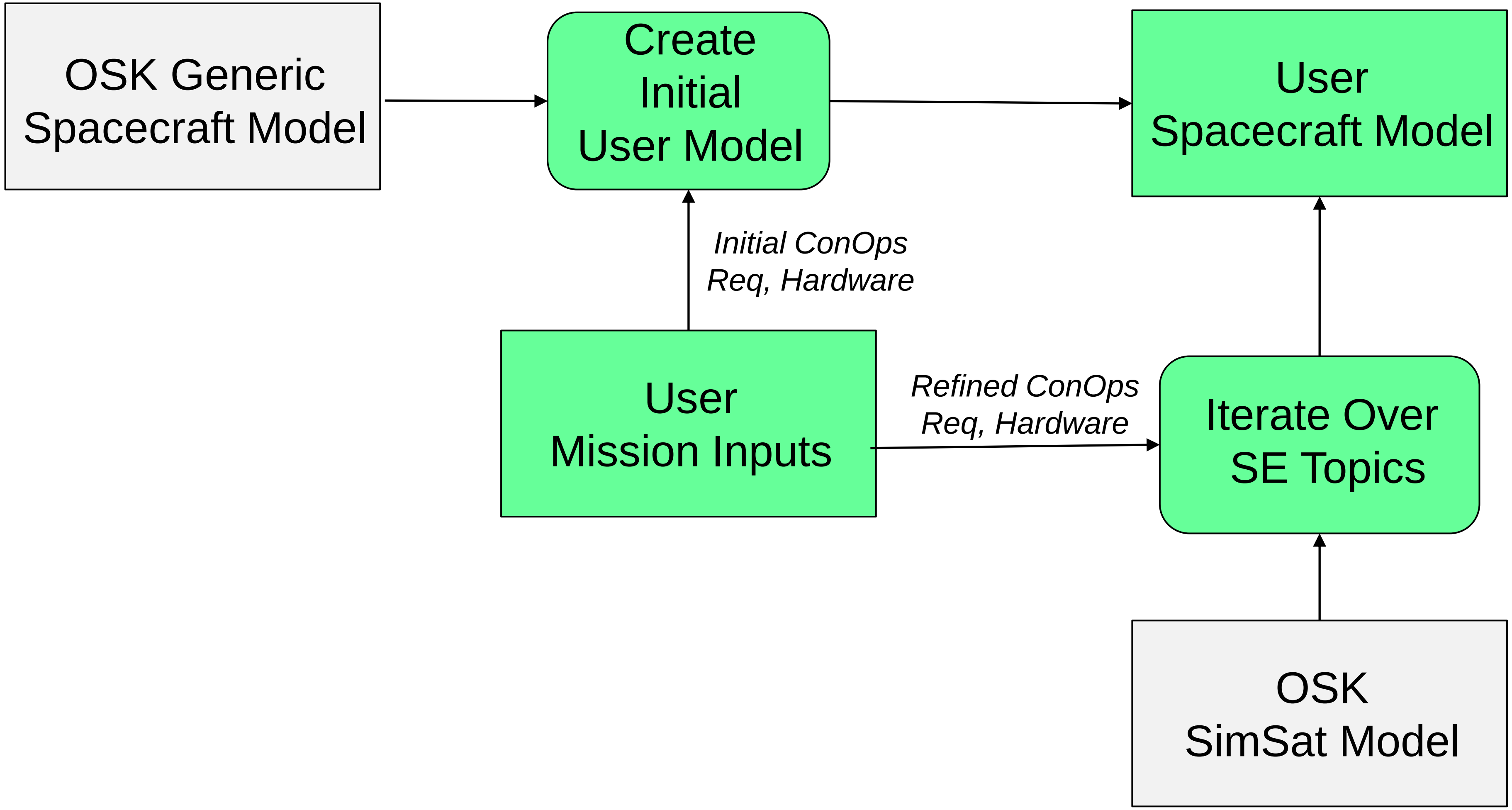
- Github submodules?
- cFS targets

Use Cases

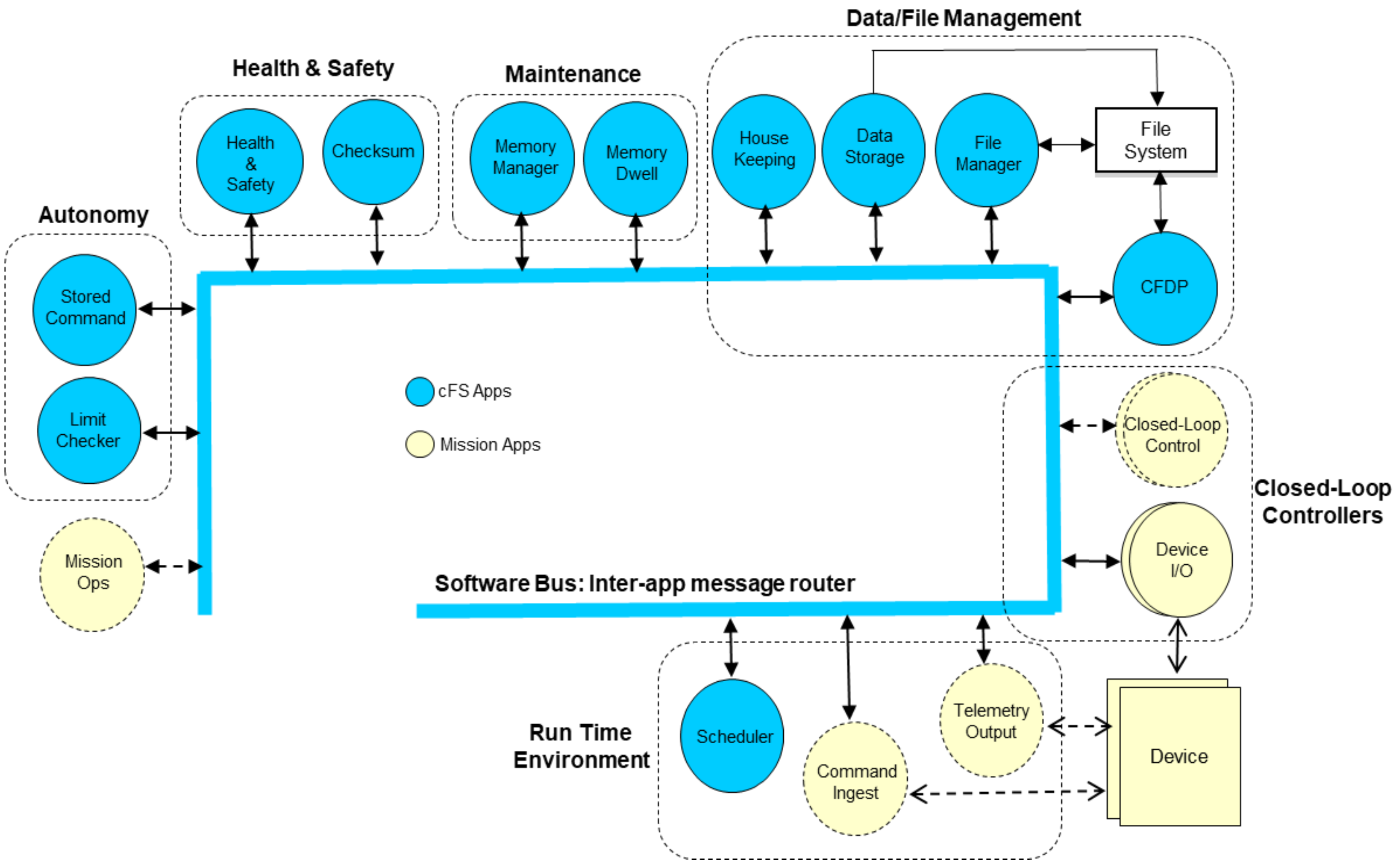
- Define driving scenarios that exercise groups of apps to achieve a mission goal

Mission FSW Systems Engineering Steps

1. **Create an initial app model from OSK's generic spacecraft model using mission concepts of operations, mission requirements, and the spacecraft hardware architecture often in the form of a block diagram**
   - Initial goal is to create a "good enough" architecture based on the maturity of the information at hand
   - Designing FSW is a very iterative process with top-down and bottom-up technical and non-technical forces at work
   - Trades are often made throughout the requirements analysis and spacecraft design phases that impact FSW. These forces are both technical and non-technical concerns may also influence decisions that impact the technical design
2. **Analyze OSK's SimSat mission app model to understand what capabilities exist within OSK**
3. **Work through system engineering topics and app groups to design new apps and understand how to configure cFS community apps**
4. **Work through spacecraft lifecycle to determine the need for**
   - Different versions of apps for different test environments
   - Simulation apps that can serve
5. **Determine how you want to use OSK in the spacecraft lifecycle**
   - Create OSK mission target
   - Plan migration to PIL and other environments
6. **If OSK will be used in a verification role then develop test artifacts as needed**
   - FSW validation should occur in a high-fidelity test environment


OSK Generic
Spacecraft Model
Create
Initial
User Model
User
Spacecraft Model
Initial ConOps
Req, Hardware
User
Mission Inputs
Refined ConOps
Req, Hardware
Iterate Over
SE Topics
OSK
SimSat Model

OSK Generic Spacecraft Model



OSK Generic Spacecraft Model

- **Show how generic model can be tailored for different missions. First focus on CubeSats, larger mission can extrapolate.**
- **Three main tailoring areas**
  - Device I/O
  - Closed loop control needs
  - Mission ops
- **Examples**
  - COTS vs inhouse ADCS
  - Payload control (closed vs open loop) and data management
  - Need for a mission manager app or ACS mode manager type app coupled with autonomy app group to achieve con ops
- **In order to work through the steps an example user mission is needed to show how to create a user model and then create a plan for how to migrate from SimSat to the user model needs**

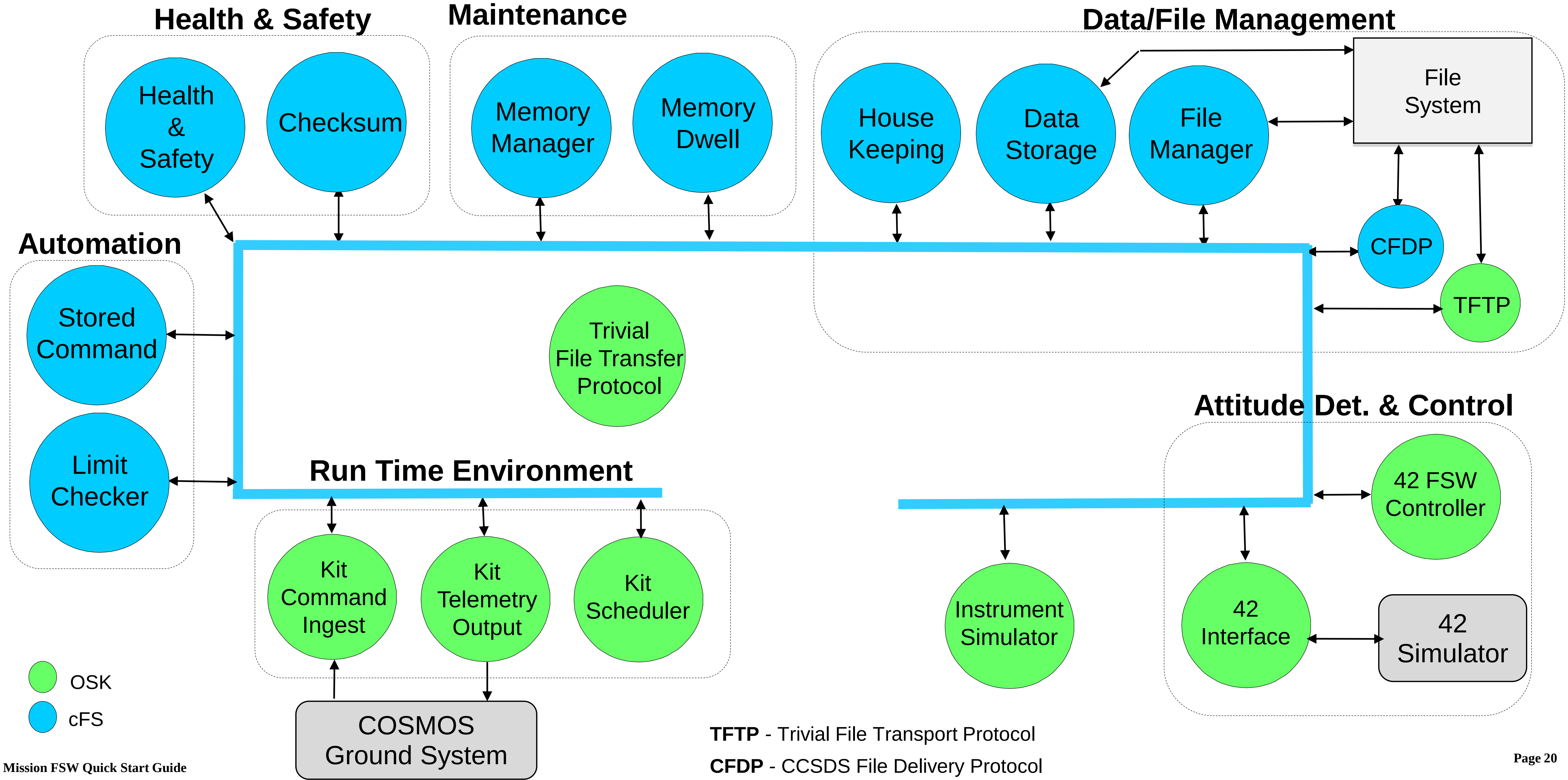
SimSat Overview

- **Objectives**
  - SimSat is a fictional spacecraft that provides a reference mission context
  - Provide a complete application suite illustrating
    - What apps are required to meet a mission's requirements
    - How they are configured and integrated as a system (not all apps configured yet)
    - Current app suite includes command & data handling (C&DH) apps
  - Provide example scripts
    - Integration test script
    - Operational script
  - Provide context for training exercises

- **SimSat is not integrated with the 42-simulator**
  - Closed-loop 42-Simulator example scenario is not related to SimSat

- **Default OSK app configuration is for a fictitious satellite called SimpleSat (SimSat)**
  - The cFS can be used for many different types of embedded systems. A spacecraft was chosen due to the increased usage of the cFS on CubeSats
- **SimSat provides a reference mission to provide context to**
  - Illustrate what applications are required and how they are configured and integrated as a system to meet the requirements
  - Demonstrate an example integration test script
  - Demonstrate an operational script
- **This does not include**
  - Porting SimSat to a new platform
  - Integrating hardware devices
- **SimSat is a**
  - Low Earth Orbit (LEO) satellite with one nadir-pointing science instrument
  - The instrument has
    - A detector that produces 10 bytes of data per second
    - A power the following sequence: Apply power, wait for instrument initialization (~20s), and command to enable science
  - The science team requires

SimSat Applications (1 of 4)



TFTP - Trivial File Transport Protocol

CFDP - CCSDS File Delivery Protocol

SimSat Applications (2 of 4)

- **The previous slide shows a cFS "bubble" chart where each app is a bubble and they communicate via messages on the software bus.**
  - The blue cFS apps are reusable open source apps that are available on https://github.com/nasa/xx where 'xx' is the abbreviated app name
  - The green OSK apps were written specifically for OSK
  - The external COSMOS and 42 interfaces use UDP and TCP respectively
- **Apps are designed to perform a dedicated function with clear interfaces and they operate in groups to achieve higher level mission objectives**

- **Runtime Environment Apps**
  - **Kit Command Ingest (KIT_CI)** receives CCSDS command packets from COSMOS and sends them on the Software Bus
  - **Kit Telemetry Output (KIT_TO)** reads CCSDS telemetry packets from the Software Bus and sends them to COSMOS
  - **Kit Scheduler (KIT_SCH)** contains tables that define when to send messages on the Software Bus
    - Apps can use these messages to perform synchronous activities, e.g. sending their housekeeping status packet

SimSat Applications (3 of 4)

- **Data/File Management**
  - **File Manager (FM)** provides a ground interface for performing common directory and file operations
  - **Data Storage (DS)** reads packets from the software bus and writes them to files according to table-defined
  - **Housekeeping (HK)** creates new telemetry packets from pieces of other telemetry packets. The new packets are written to the SB and can be stored and/or telemetered.
  - **Trivial File Transfer Protocol (TFTP)** transfers files between the flight and ground COSMOS. There's an open source CCSDS File Delivery Protocol (CFDP) app that will be added in a future release.

- **Autonomy**
  - **Limit Checker (LC)** monitors one or more telemetry values and start stored command relative time sequences (RTSs) in response to limit violations
  - **Stored Command (SC)** Provides services to execute preloaded, table-defined command sequences at predetermined absolute or relative time intervals

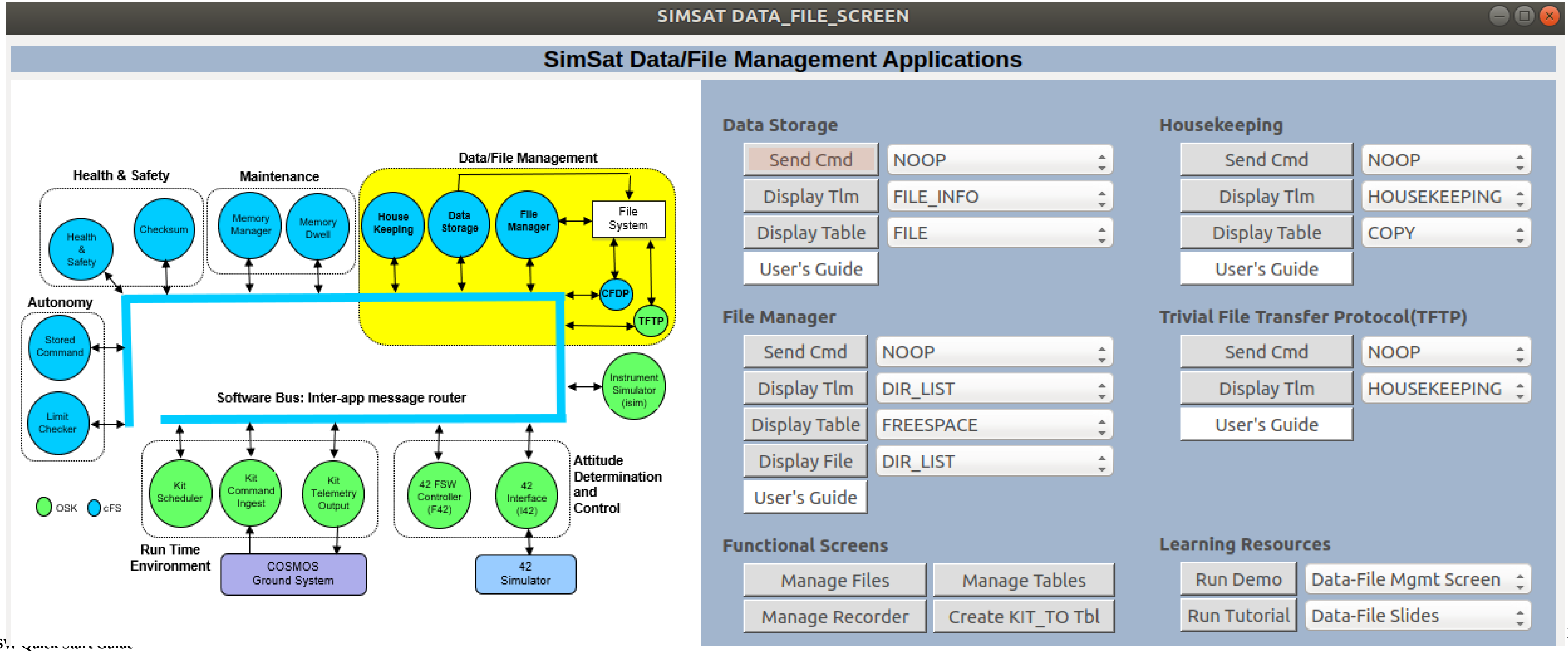
- **Attitude Determination and Control Apps**
  - **42 Interface (I42)** manages a TCP/IP connection to 42 and transfers actuators/sensor packets to/from 42
  - **42 FSW (F42)** Implements the “ThreeAxisFsw” attitude control algorithm defined in 42

- **Maintenance**
  - **Memory Dwell (MD)** creates telemetry packets containing contents of memory location specified in dwell tables
  - **Memory Manager (MM)** provides read/write access to memory

- **Health & Safety**
  - **Checksum (CS)** monitors checksums across table-defined static code/data regions and reports errors
  - **Health & Safety (HS)** monitors table-defined application check-in and event messages and reporting errors and/or starting a RTS to address the issue

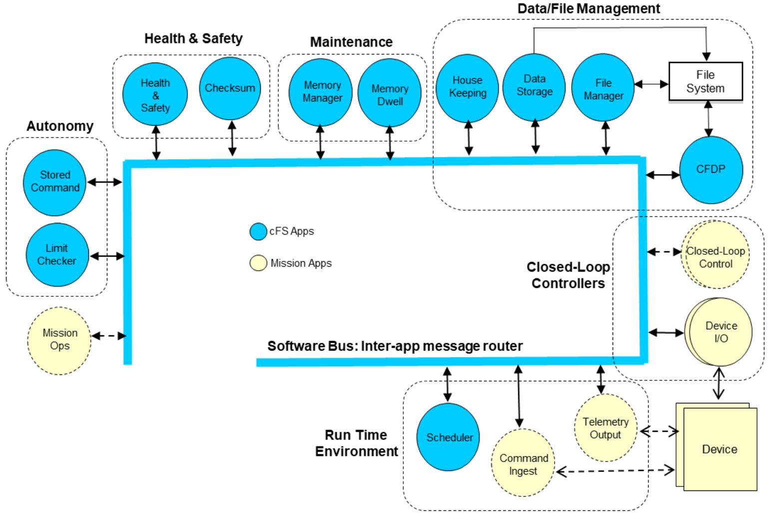
SimSat Application Group Screens

- Each screen highlights the pre-configured SimSat apps that participate in achieving a particular user goal
- Similar organization to the cFE service screens: App ground interface, Functional screen link and a Learn section
- Some demos exist, but not all apps configured for SimSat and no tutorials have been created

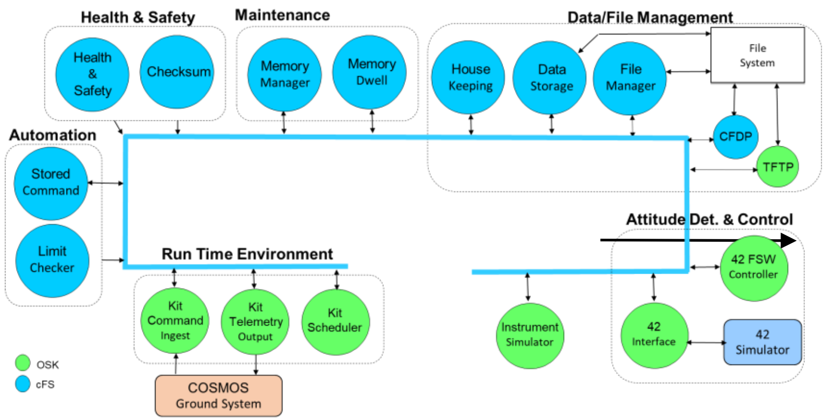


FSW Systems Engineering Process

Generic Spacecraft Model



SimSat Reference Mission

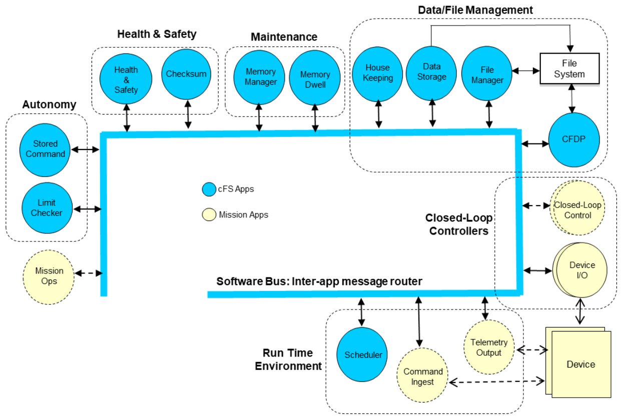


Mission Inputs

ConOps
Req, Hardware

Iterate over FSW System Engineering Topics

Mission Model

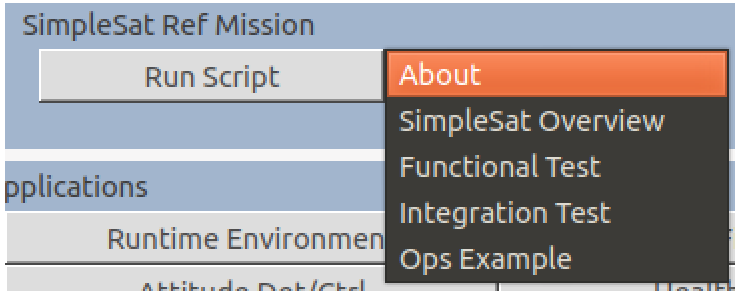


- Workbook style documentation steps users through FSW system engineering (SE) topics
- Applications addressed in small groups that collaborate to provide end-user functionality

"Explore cFS/SimSat" Tab

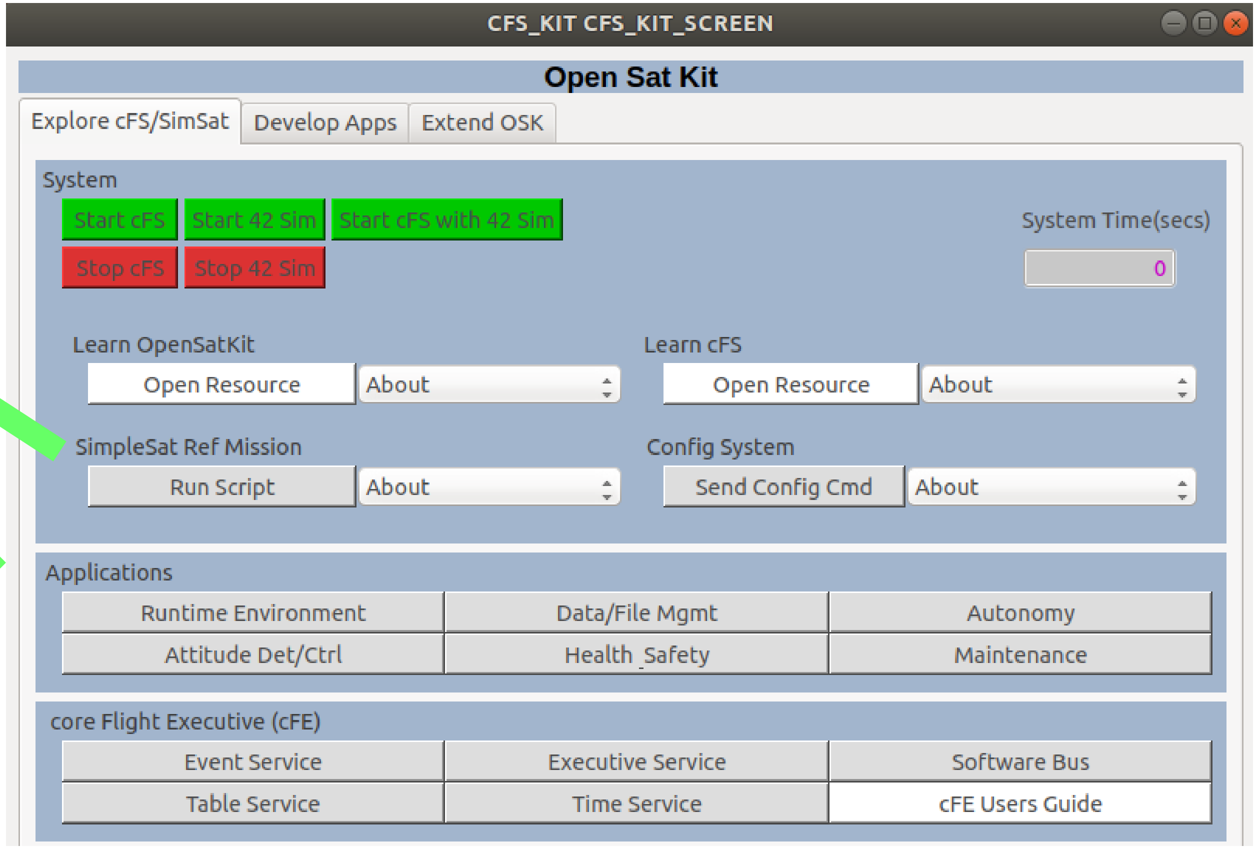
1. SimSat reference mission slides and scripts
2. Launch application functional group screen (next slide)

1 SimSat Reference Mission



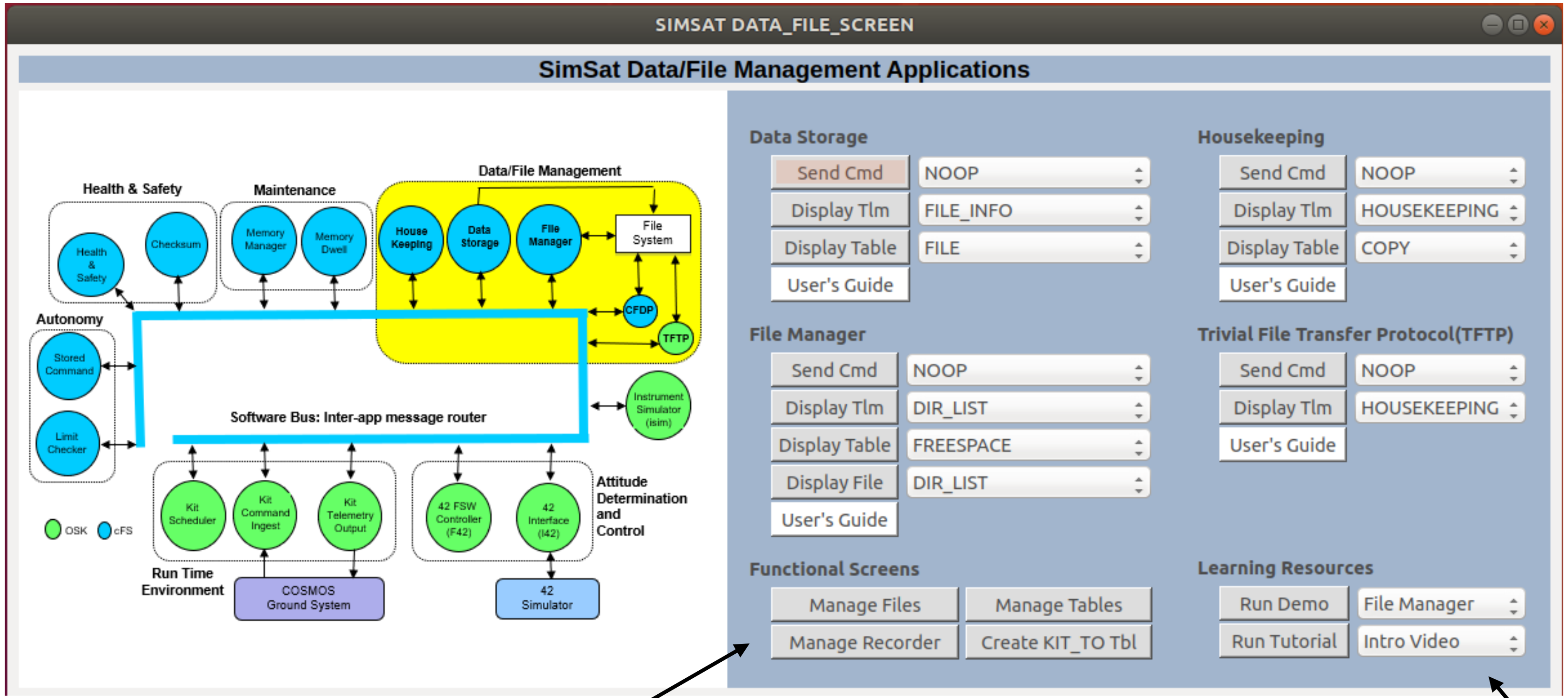
2 Application Group

(next slide)



SimSat App Screens

Each functional application group screen uses the following layout



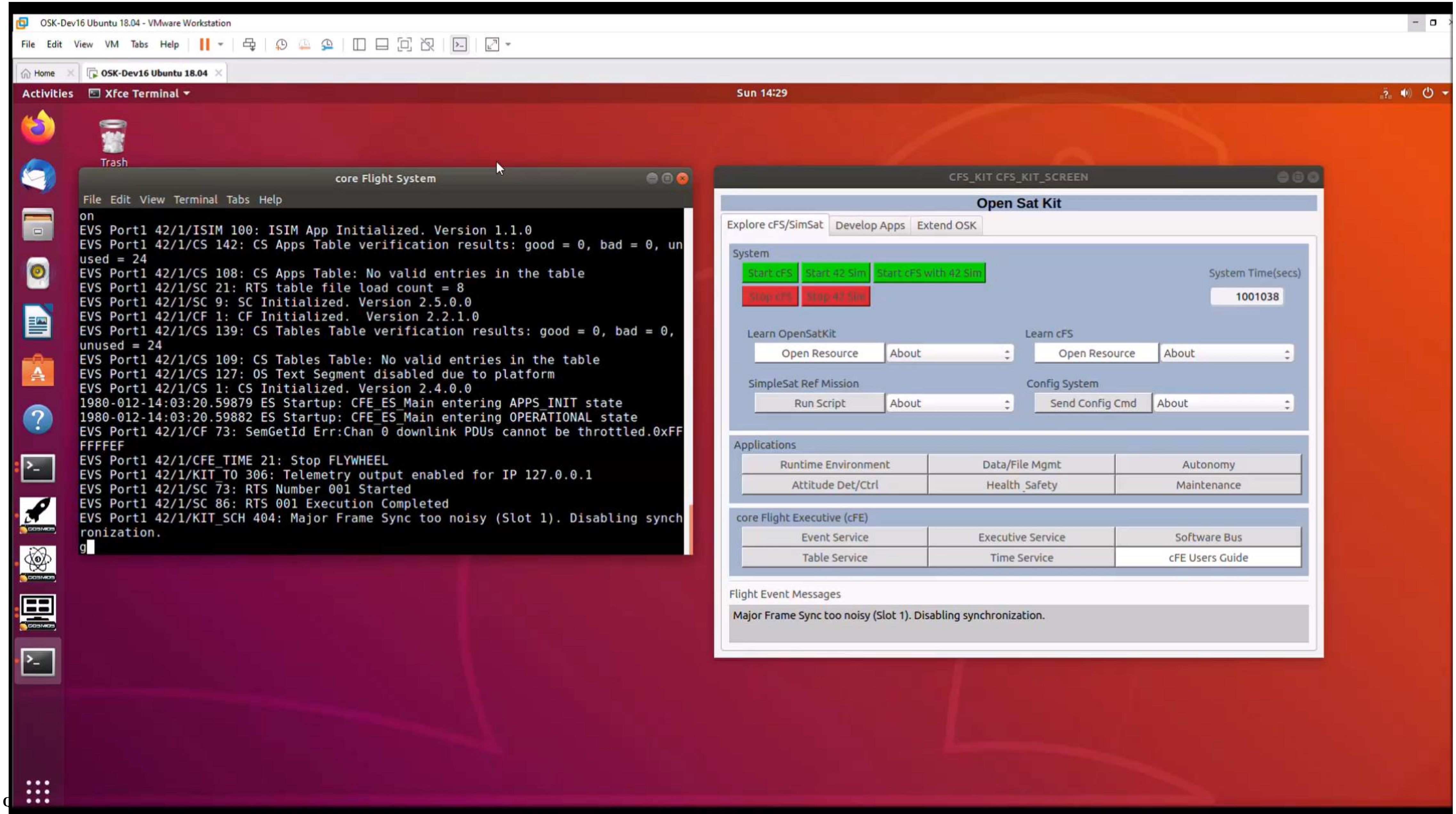
Complete interface to each app

- All commands
- All telemetry packets
- "Display Table" – Dump, transfer and display table in COSMOS Table Manager
- "Display File" – Issue app's command to create a file, then transfer and display binary file in COSMOS Table Manager

Functional screens combine commands and telemetry from one or more apps that work together to perform a related tasks.

Launch videos, demos (pre-defined screen sequences) and tutorials (slides and/or scripts)

File-Data Storage Demo



System Engineering Topics

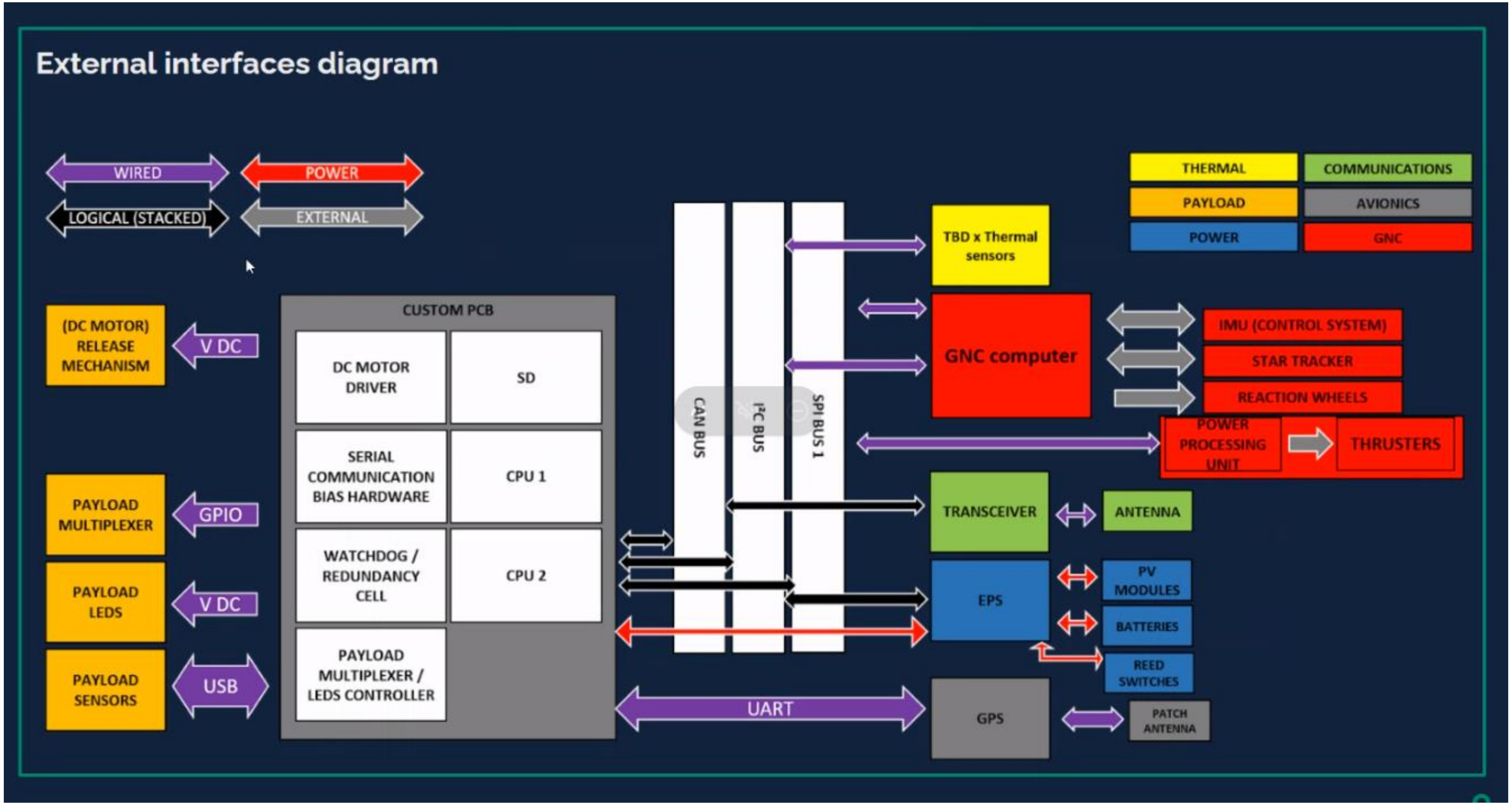
- **System Engineering topics do not have a one-to-one correlation with app groups**
- **ConOps, mission requirements, and a hardware block diagram are required inputs**
  - This is an iterative process so reapply these inputs
  - Spacecraft lifecycle can be considered a process input and iteratively examined to determine what's needed
- **Goal is to manage complexity by working through topics**
  - Topics are not 100% orthogonal (non-overlapping)
  - Create a ConOps and show how it impacts different apps groups
- **SE Topics**
  1. Device I/O
  2. Close-loop control
  3. Mission Ops & autonomy
  4. Data-File management
  5. Runtime Environment
  6. FDIR
  7. Interface & control apps
  8. Maintenance
  9. Time
  10. Parameters and configuration
- **V&V Topics**
  1. Unit tests
  2. App functional tests
  3. Integration tests
  4. System and ops tests

Apply the OSK-cFS System Engineering Development Processes

Mission Analysis

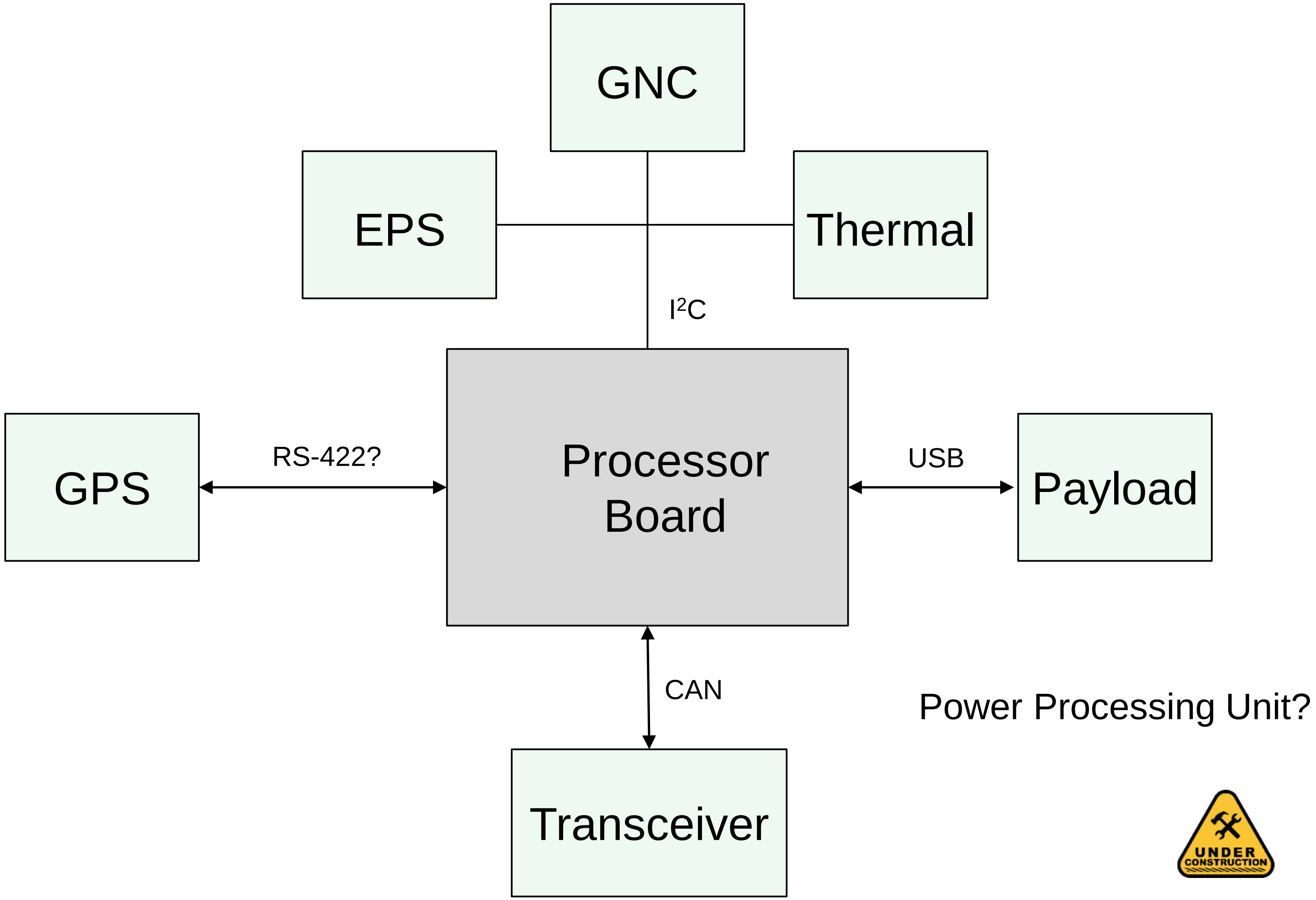
- Success criteria
- Driving requirements

Spacecraft Block Diagram

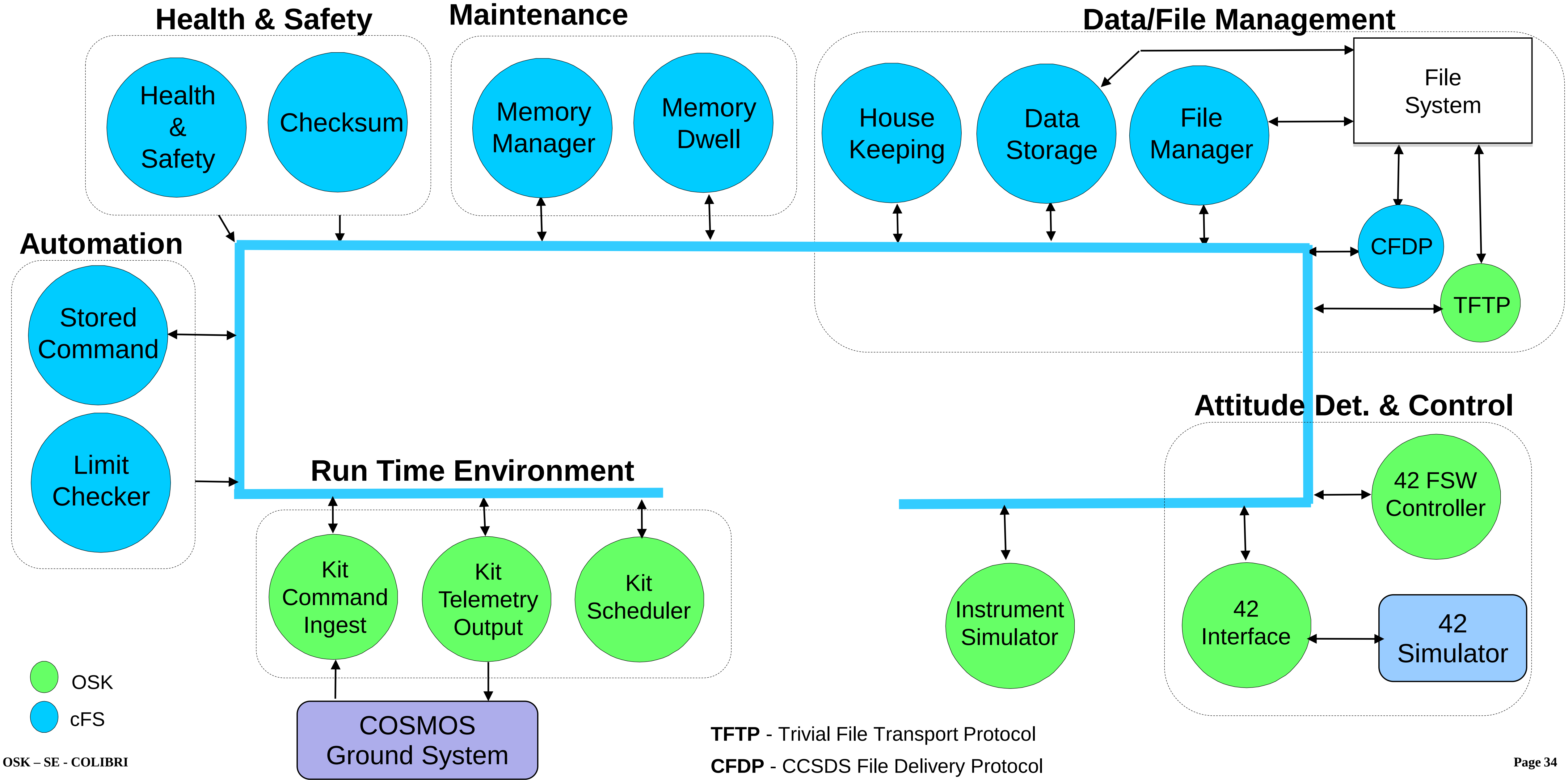


Systems engineer maintain a spacecraft block diagram that has a version number

FSW Hardware Context



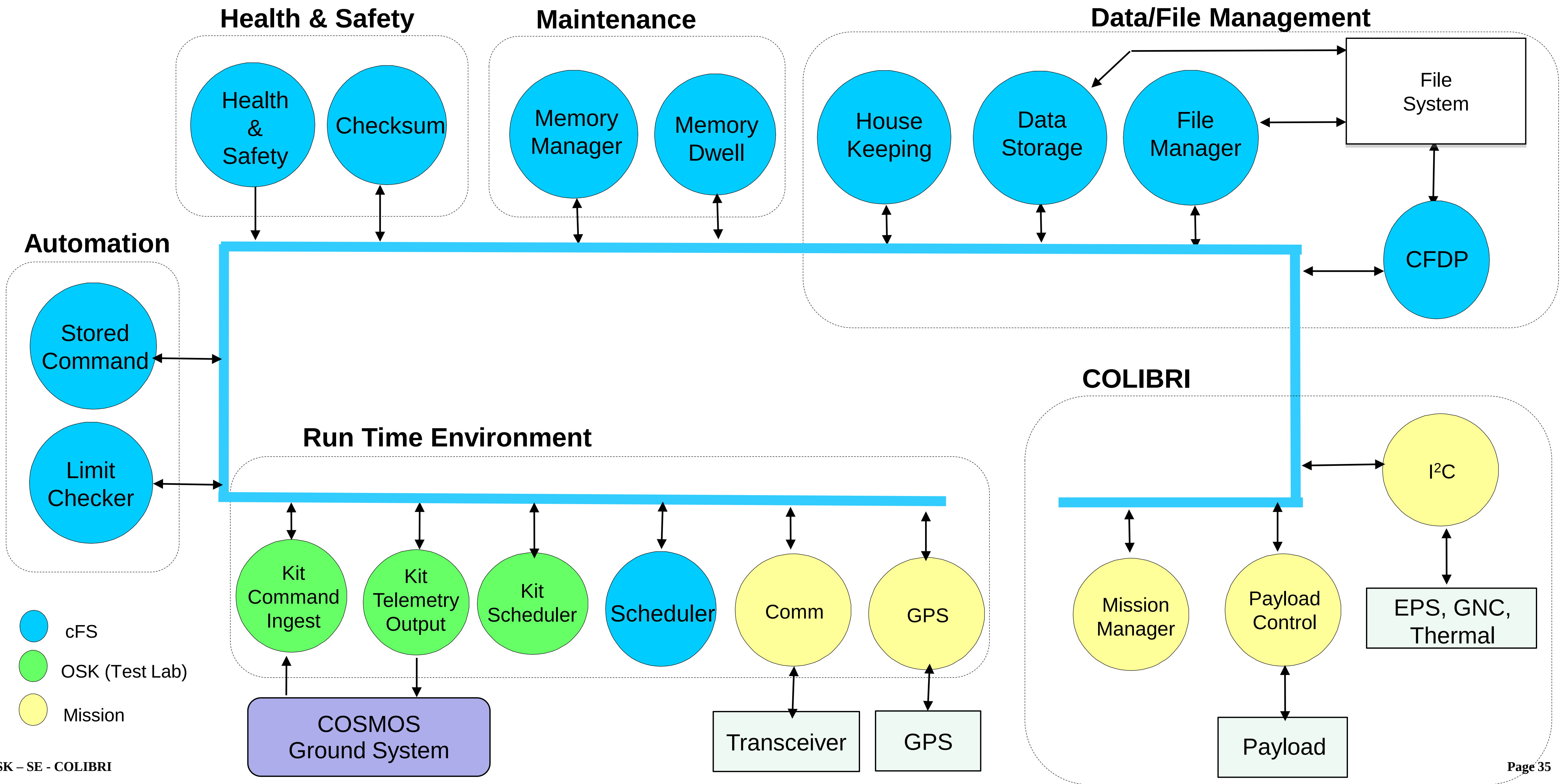
This context diagram should be synchronized with the system block diagram



TFTP - Trivial File Transport Protocol

CFDP - CCSDS File Delivery Protocol

Draft Mission Applications



Design Approach (1 of 3)

- **Decompose into goal-oriented manageable subsystems**
  1. TBD Mission (Mission Manager and Automation App Groups)
  2. Runtime Environment App Group
  3. Data-File Management App Group
  4. Fault Detection Isolation and Recovery (FDIR) (Health & Safety and Automation App Groups)
  5. FSW Sustaining Engineering (Maintenance and Automation App Groups)

Design Approach (2 of 3)

- **For each subsystem**
  - Identify new applications, often based on top-down functional requirements and interfaces (trace req.)
  - Message-based app designs (analogous to object-oriented class design)
    - Identify each app's context (what it knows about other components in the system)
    - Messages identify **what** an app can do rather than **how** it should do something
    - Use scenarios & sequences to help create and verify designs
  - Identify cFE service and cFS app features and how they can be used/configured to meet your needs
  - Iterate design from different perspectives: top-down, bottom-up, interfaces, control flow, data flows, etc.
  - Find existing apps that are close to what is needed by the new app
    - cFS design patterns exists they just aren't documented (see OSK Runtime Environment slides)
  - Think through lifecycle phases: Development, Spacecraft Test, On-orbit checkout, sustaining engineering
- **Maintain critical resource budgets**
  - Identify critical resources
    - CPU, memory (all types), telemetry (realtime & stored), onboard recorder, interfaces, message IDs, etc.
  - Estimate, measure, and track each critical resource
  - Be judicious on what you track and to what level of detail based on available engineering resources and perceived risk

Design Approach (3 of 3)

- **Interface Design Considerations**
  - What data is needed onboard for onboard processing? How often? Does it need to be synchronized with other onboard processing?
    - What data integrity checks should be performed prior to using the data onboard?
    - What potential faults should be monitored and how detected?
  - Is additional data required for the ground? If so, how often?
  - What data is needed in realtime and what can be stored?
    - Think through testing vs operational needs
    - Note you want to eventually “test as you fly” so don’t become dependent on test data
  - What data is required for diagnostics? Does the device support on-orbit software updates?
  - How will the interface be managed in each test configuration?
  - Is there a need for direct ground-to-device command or an override capability?
- **Ground Tools**
  - What tools exists? What are required? What are nice to have?
  - The cFS started at NASA/Goddard where ground system tools were built into existing ground systems and these tools have not be separated and made public

Software Development Process Recommendations

- **Assign individuals to apps or groups of apps and to system engineering artifacts**
  - Ownership and accountability promote more in-depth questioning
- **Select a common application design for new apps** (See notes page)
  - cFS App Developer’s Guide style
  - OSK Framework style
  - Others exists
    - Probably best to use
- **Hold code reviews**
  - Improves code, requirements, and alignment of code to the requirements
  - Distributes expertise across the team
- **Develop Unit and Build Test**
  - If possible, have an independent system tester

Driving Uses Cases

- **Launch & Separation**
- **“Day in the Life”**
  - Time Management
  - Science Ops
  - Payload Control
  - Ground Contacts
- **Processor reset behavior**

Mission Mode Diagram

1 –Mission: Mission Manager, SC, and LC

- **Mission Manager**
  - Context diagram
  - Identify main functional requirements
- **Initial design thoughts**
  - Define a mission mode state diagram that will implemented by MM
  - Identify commanded and autonomous mode transitions
  - An advantage of using MM rather than Limit Checker and stored commands is that MM can check for condition prior to performing a mode transition
  - How will eclipse
- **Absolute Time Command (ATS) Sequence**
- **Relative Time Command Sequences**
  -

1 –Mission: I^2C App

- **Context diagram**
- **Identify main functional requirements**
- **Scheduler table driven? Pend for wakeup message to gather I/O**

1 – CALIBRI Mission: Payload Control App

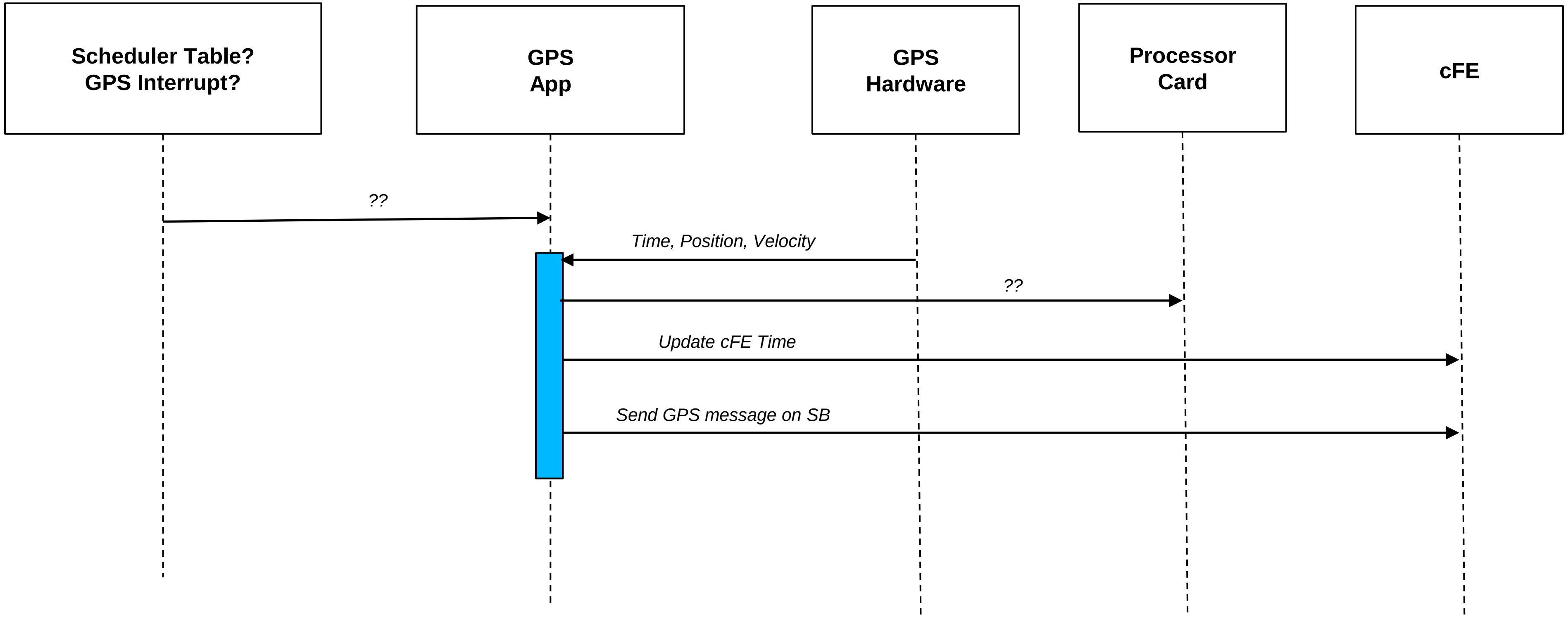
- **Context diagram**
- **Identify main functional requirements**
- **Data driven – Pend for data packets needed to run control algorithm**
  - Couple of strategies to consider
- **Design algorithms with maximum number of table-defined parameters**

1 –Mission: Sequence Diagrams

2 – Runtime Environment

- **Time Management**
  - When is GPS powered?
  - Can GPS be synchronized with SBC clock?
- **GPS App**
  - Scheduler or interrupt driven
- **Comm App(s)**
  - Interrupt/data driven
- **Transition to Scheduler App**
- **Scheduler Table design**
- **Design synchronous system**
- **Tune the system (performance monitor tool)**

2 – Time Management Scenario



How can the processor clock be correlated GPS?

3 – Data-File Management

- Telemetry Budget
- File Transfer

4 – Fault Detection Isolation and Recovery

- **Limit checker and stored command apps implement the FDIR**
- **Other**

5 – FSW Sustaining Engineering

- **Since it's a short mission with a 6-month duration I recommend focuses on the most likely scenario of needing to upload a new library or application**
  - This only requires cFE services
- **I recommend having the Memory Manager and Memory Dwell apps as part of the default FSW, however I would scale the time put into using them based on available FSW engineers**

Other Potential Topics

- **Freezing Versions**
  - COSMOS
  - cFS
  - Apps
- **Tuning System**
- **Debugging**
- **V&V configurations**
- **Event filtering**
- **Designing a scheduler table**
- **Event log buffer management**
  - Local event logs
- **Create a synchronous system**
  - Consistent operations
  - Test repeatability (FSW lab vs thermal vac). Need to know whether thermal if conditions caused the erroneous behavior

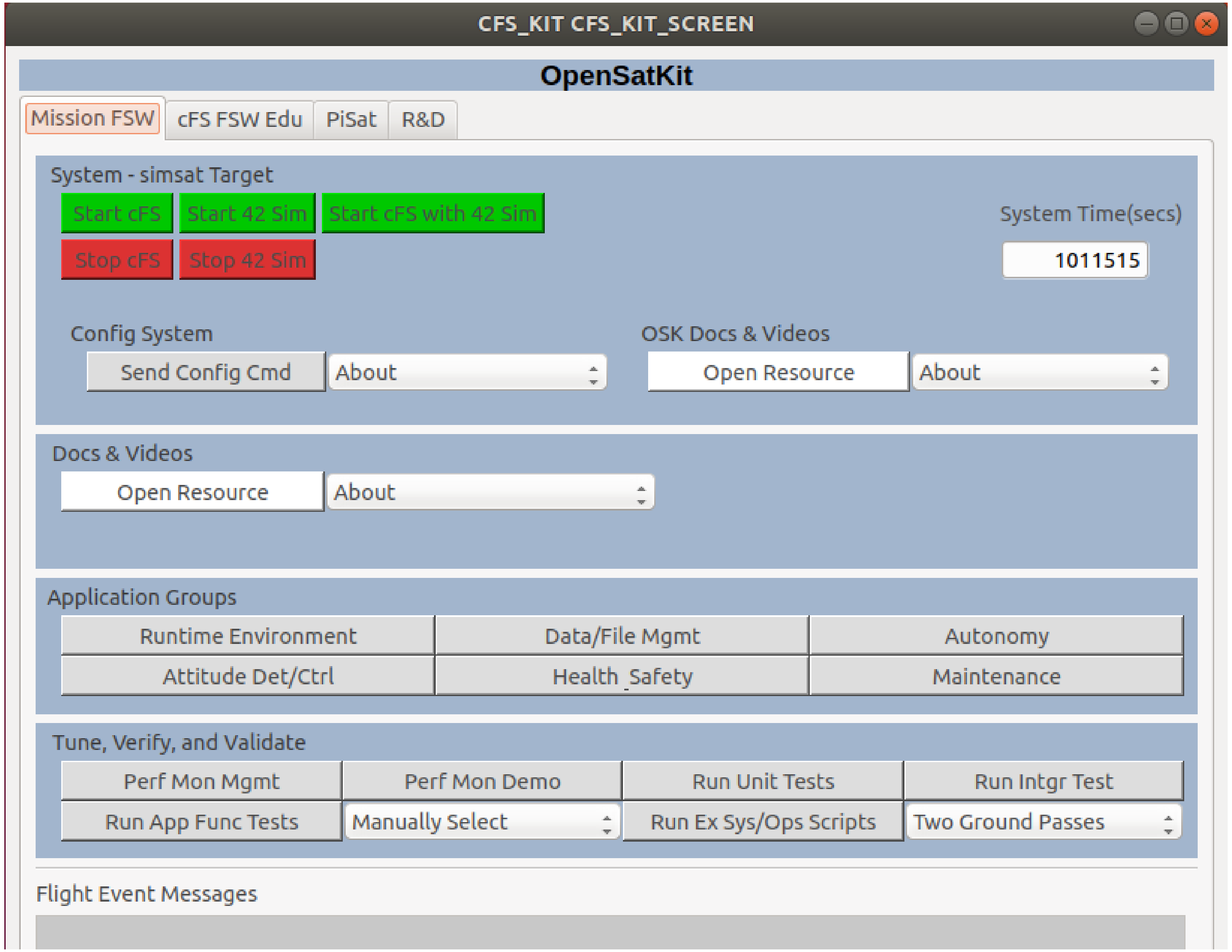
System Engineering Topics

- **SE Topics**
  1. Device I/O
  2. Close-loop control
  3. Mission Ops & autonomy
  4. Data-File management
  5. Runtime Environment
  6. FDIR
  7. Interface & control apps
  8. Maintenance
  9. Time
  10. Parameters and configuration
- **V&V Topics**
  1. Unit tests
  2. App functional tests
  3. Integration tests
  4. System and ops tests

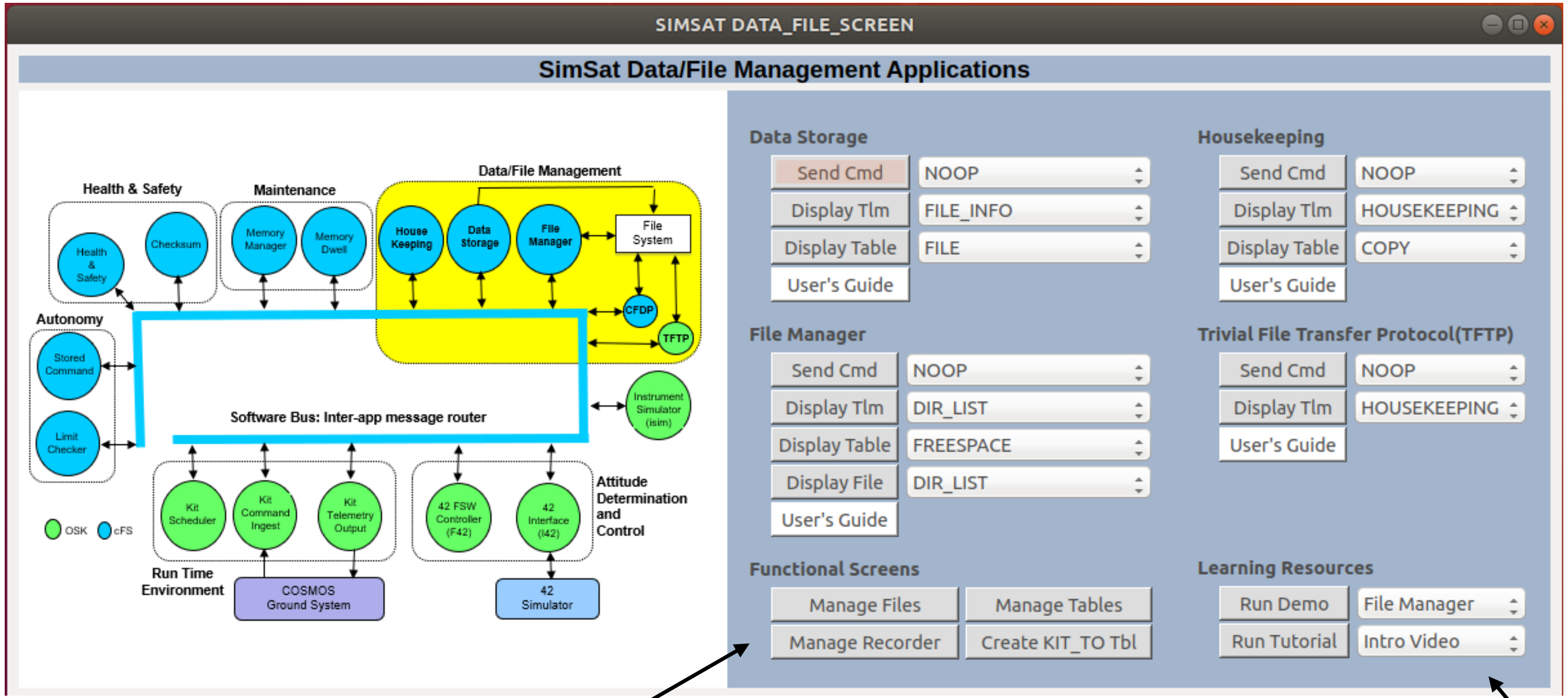
1. What data must be downlinked to meet mission goals?
2. What are the contact frequencies, durations, and data rates?
3. What parts of operations need to be automated?
4. What level of fault detection, isolation, and recovery (FDIR) is necessary?

- Bottom-up

1. What processor card is being used and is a realtime operating system required?
2. What device interfaces does the FSW need to manage and how are they connected?
3. Make versus buy decisions. Some top-down decisions impact bottom-up design.


CFS_KIT CFS_KIT_SCREEN
OpenSatKit
Mission FSW
cFS FSW Edu
PiSat
R&D
System - simsat Target
Start cFS
Start 42 Sim
Start cFS with 42 Sim
Stop cFS
Stop 42 Sim
System Time(secs)
1011515
Config System
Send Config Cmd
About
OSK Docs & Videos
Open Resource
About
Docs & Videos
Open Resource
About
Application Groups
Runtime Environment
Data/File Mgmt
Autonomy
Attitude Det/Ctrl
Health_Safety
Maintenance
Tune, Verify, and Validate
Perf Mon Mgmt
Perf Mon Demo
Run Unit Tests
Run Intgr Test
Run App Func Tests
Manually Select
Run Ex Sys/Ops Scripts
Two Ground Passes
Flight Event Messages

Each functional application group screen uses the following layout



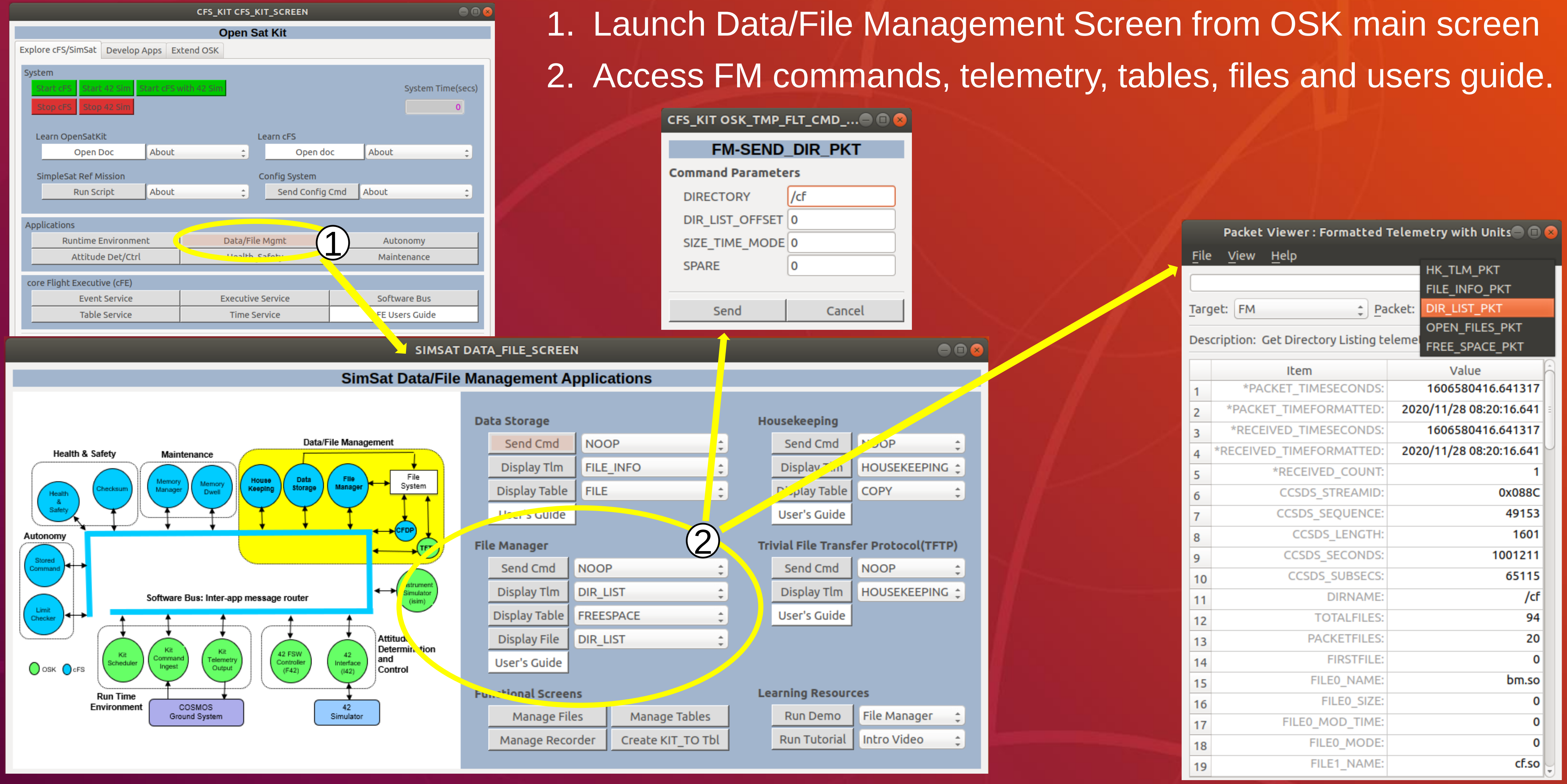
Complete interface to each app
- All commands
- All telemetry packets
- "Display Table" – Dump, transfer and display table in COSMOS Table Manager
- "Display File" – Issue app's command to create a file, then transfer and display binary file in COSMOS Table Manager

Functional screens combine commands and telemetry from one or more apps that work together to perform a related tasks.

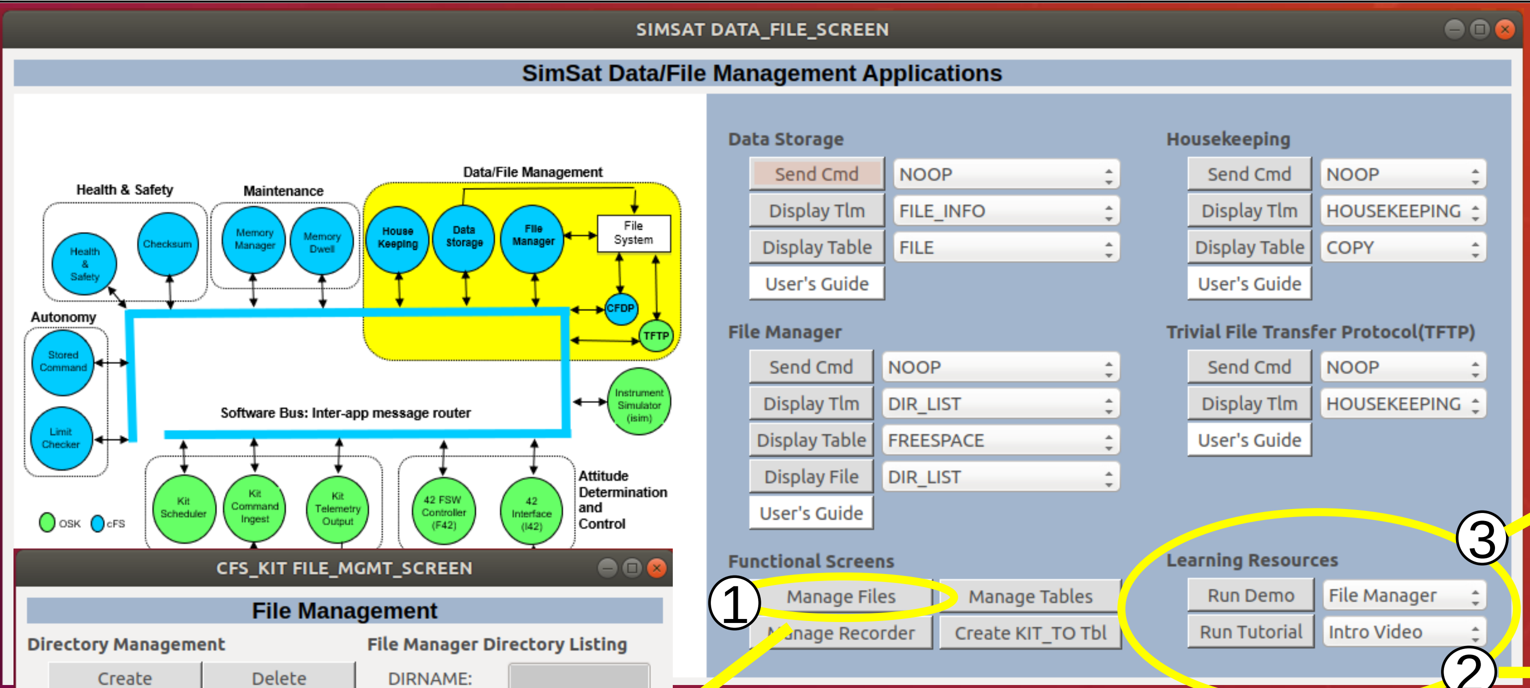
Launch videos, demos (pre-defined screen sequences) and tutorials (slides and/or scripts)

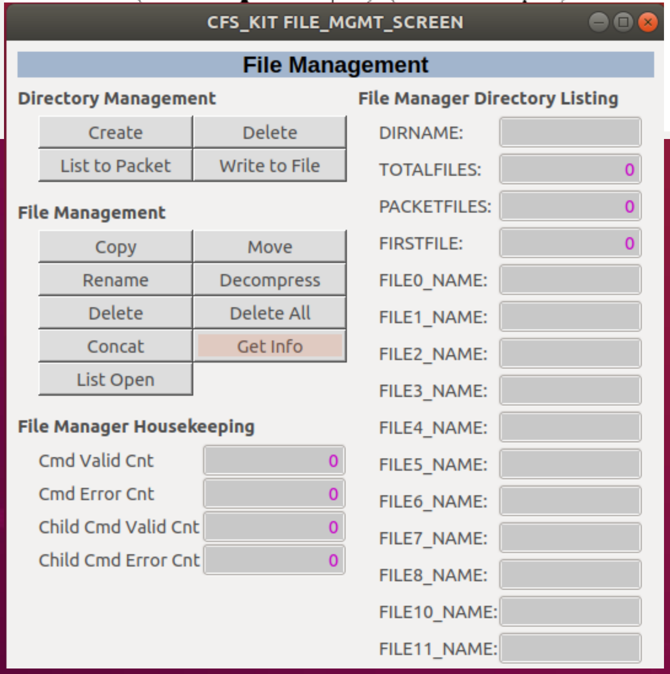
App Interactive Resources (1 of 2)

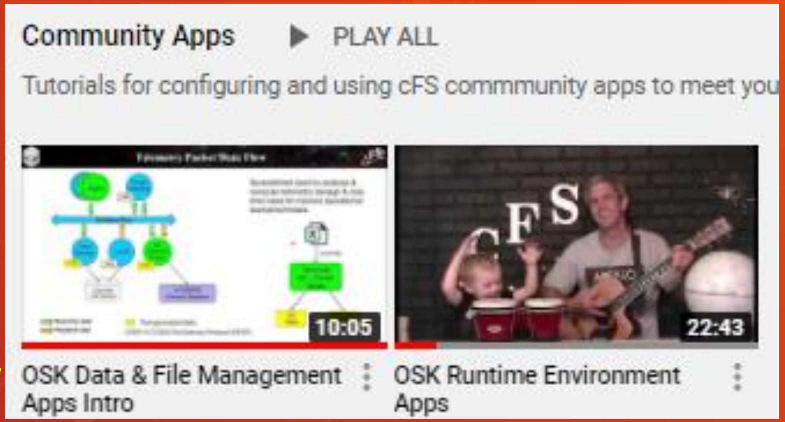
1. Launch Data/File Management Screen from OSK main screen
2. Access FM commands, telemetry, tables, files and users guide.

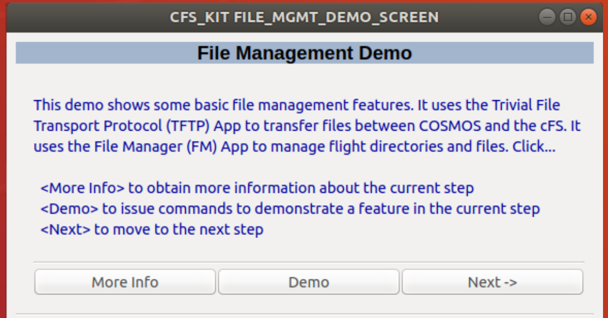


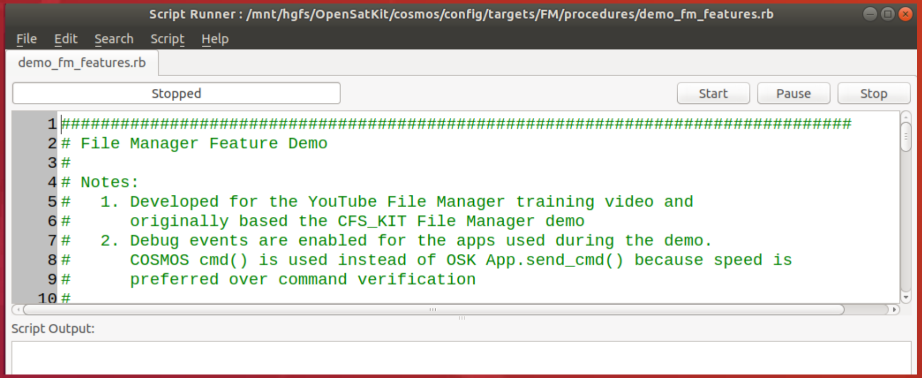
App Interactive Resources (2 of 2)





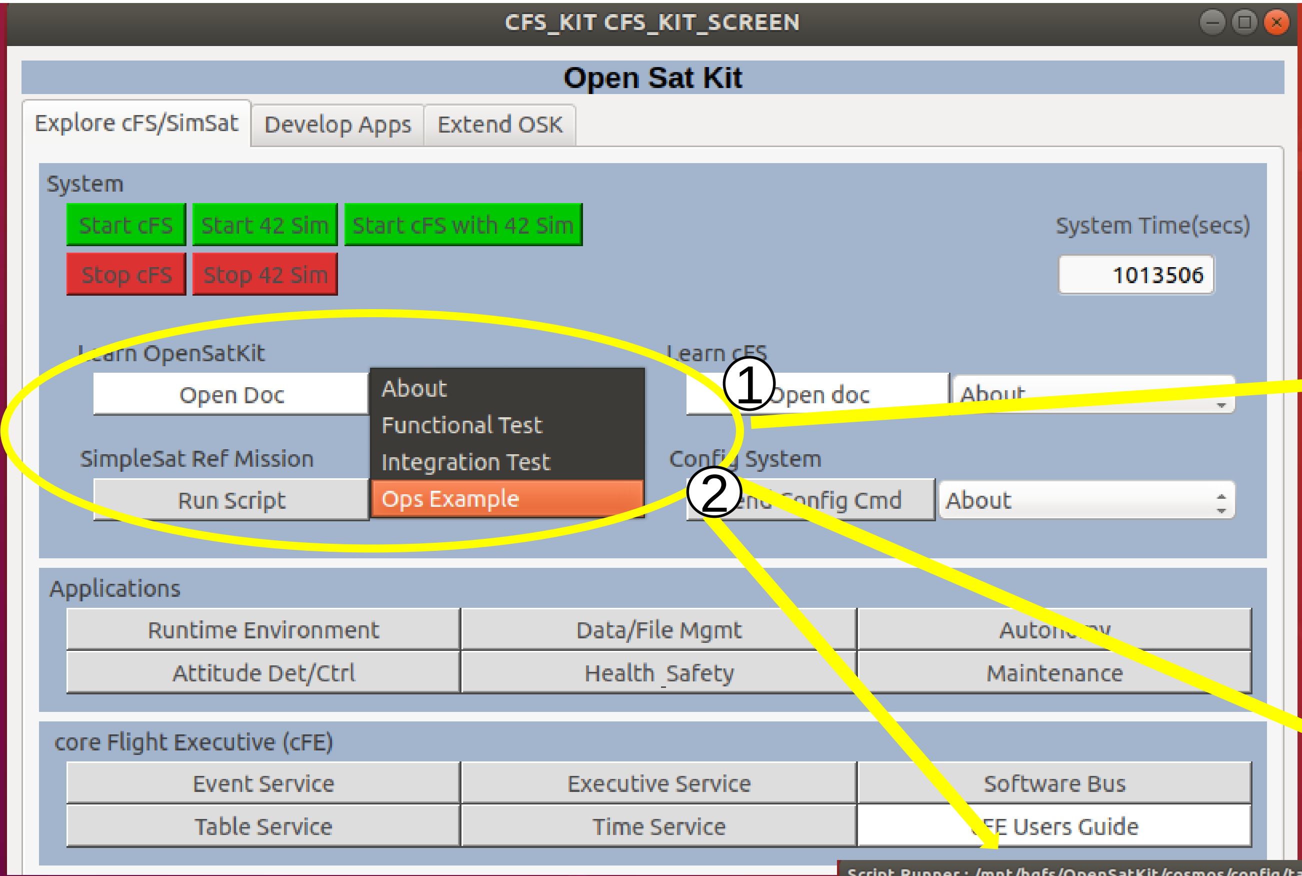




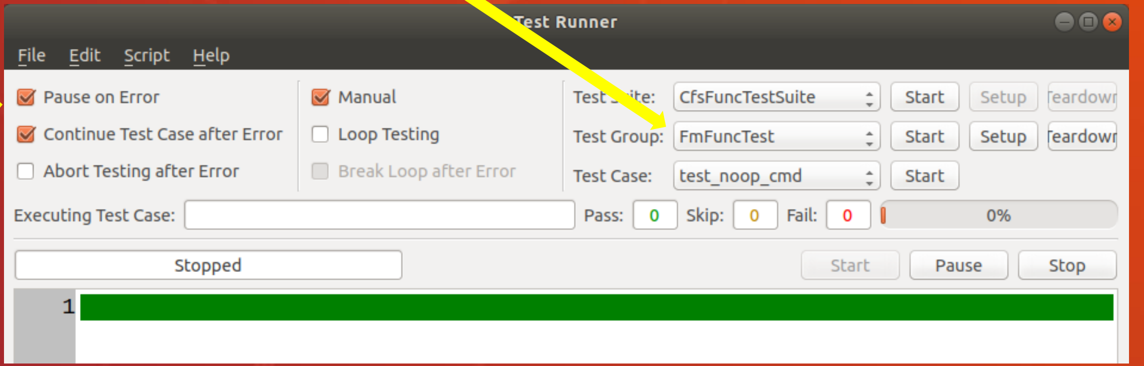


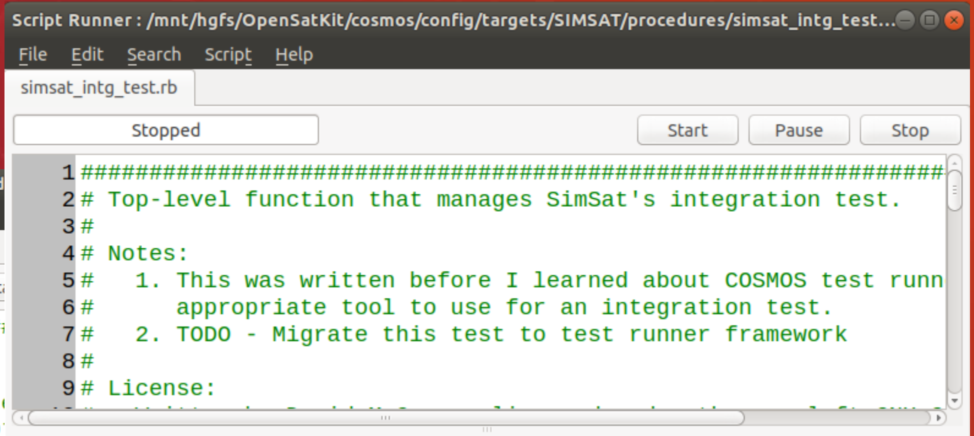
1. FM summary page with HK and directory listing
2. FM feature demo
3. YouTube Tutorials

App System Scripting Resources

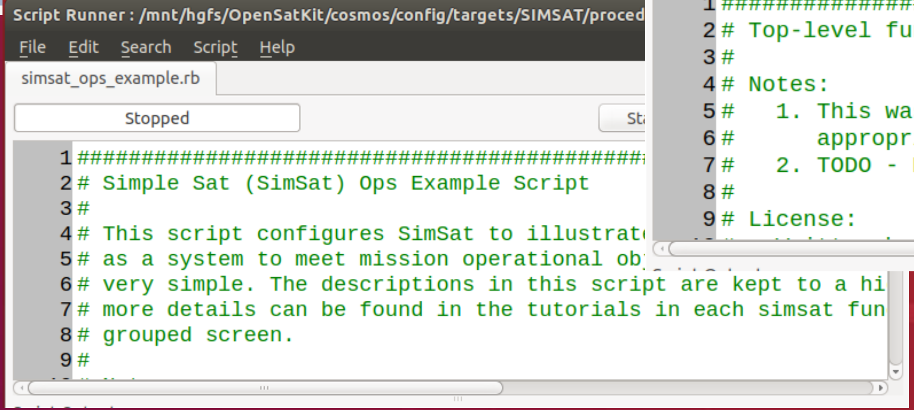


1. Functional Test Suites contain Test Groups for each app that run in the COSMOS Test Runner

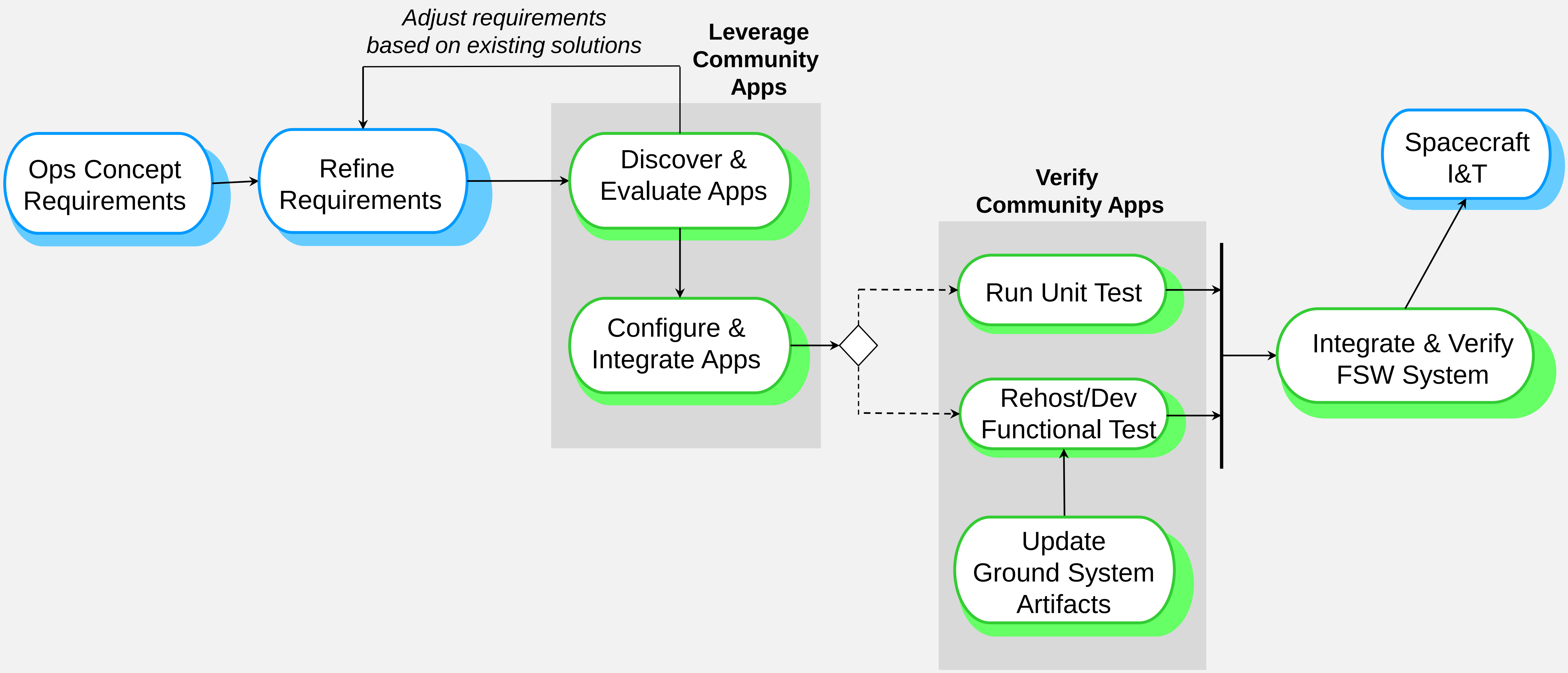




2. The Integration Script and Ops Example use the COSMOS Script Runner



Select and Configure cFS Community Apps



Select and Configure cFS Community Apps

- **Leverage Community Apps**
  - OSK's SimSat application groups provide a system context for users to learn how apps work together to achieve a goal
    - Slides, YouTube videos, and built-in demos
    - Example operational script
    - The above resources are not complete, and the plan is to continually update them
  - Prototype tools for generating default Data Storage, KIT_TO, and Housekeeping table source files
    - Links to tools from application group screens

- **Verify Community Apps**
  - NASA app unit tests can be run as needed
  - OSK apps do not currently have unit tests
  - OSK supports functional app testing and system integration testing using COSMOS Script Runner and Test Runner tools
    - Thin integration and functional test infrastructure built on top of COSMOS script API
    - No functional tests exist

OSK SimSat NASA cFS Libs & Apps

Name	Description	Cmd, Tlm, Tbl Defined	Video	Notes
CFDP (CF)	Implements the CCSDS File Delivery Protocol (CFDP)	Partial	No	Compiled, but not loaded by default. Downlink flow control semaphore not implemented
CFS_LIB	Utility functions	N/A	No	
Checksum (CS)	Compute checksums across table-defined memory blocks	All	No	
Data Storage (DS)	Read packets from the software bus and store them in files	All	Group	
File Manager (FM)	Provide basic directory & file management services	All	Group Solo	
Housekeeping (HK)	Combine parts of packets into new packets	All	Group Solo	
Health & Safety (HS)	Provide system health & safety checks	All	No	
Limit Checker (LC)	Monitor values in packets and activate stored commands as needed	All	No	
Memory Dwell (MD)	Create telemetry packets containing user defined memory locations	All	No	
Memory Manager (MM)	Provide interface to read/write (load/dump) memory locations.	All	No	Use with caution!
Software Bus Network (SBN)	Software Bus Network	None	No	Compiled, but not configured or loaded by default.
Stored Command (SC)	Manage absolute time and relative time command sequences		No	

OSK SimSat Custom Libs & Apps

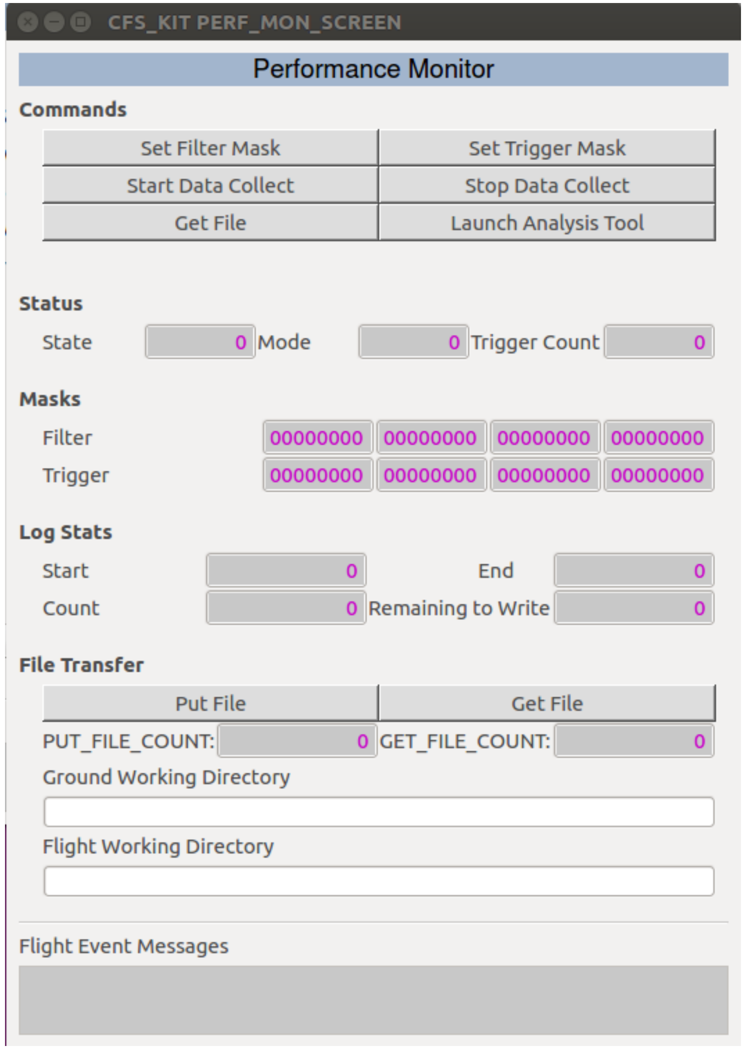
Name	Description	Cmd, Tlm, Tbl	Video	Notes
expat_lib	XML Parser	n/a	No	Only used by demo apps
osk_c_fw	Application framework for apps written in C	n/a	No	Used for all OSK SimSat apps
osk_42_lib	Utilities to manage 42's interface	n/a	No	Allows default options to be used for 42's socket interface code generation utility & 42's acapp to be used unchanged
F42	Manages execution of 42's "acapp" controller	All	No	
I42	Manages sensor/actuator interfaces for F42	All	No	
ISIM	Simulates an instrument for SimSat demos	All	No	
KIT_CI	Ingest commands over UDP socket and send them on software bus	All	Group	Functionally similar NASA's lab_ci
KIT_SCH	Send table-defined messages on the software bus	All	Group	Schedule algorithm similar to sch_lab. Message and scheduler table implemented in JSON.
KIT_TO	Receive telemetry packets from the software bus and output them over a UDP socket	All	Group	Output JSON filter table determines packet output rates. OSK table tool can generate the table for a CSV file.
TFTP	Implement Trivial File Transport Protocol	All	No	Default file transfer protocol. Doesn't preclude use of CF app.

OSK cFS Additional Libs & Apps

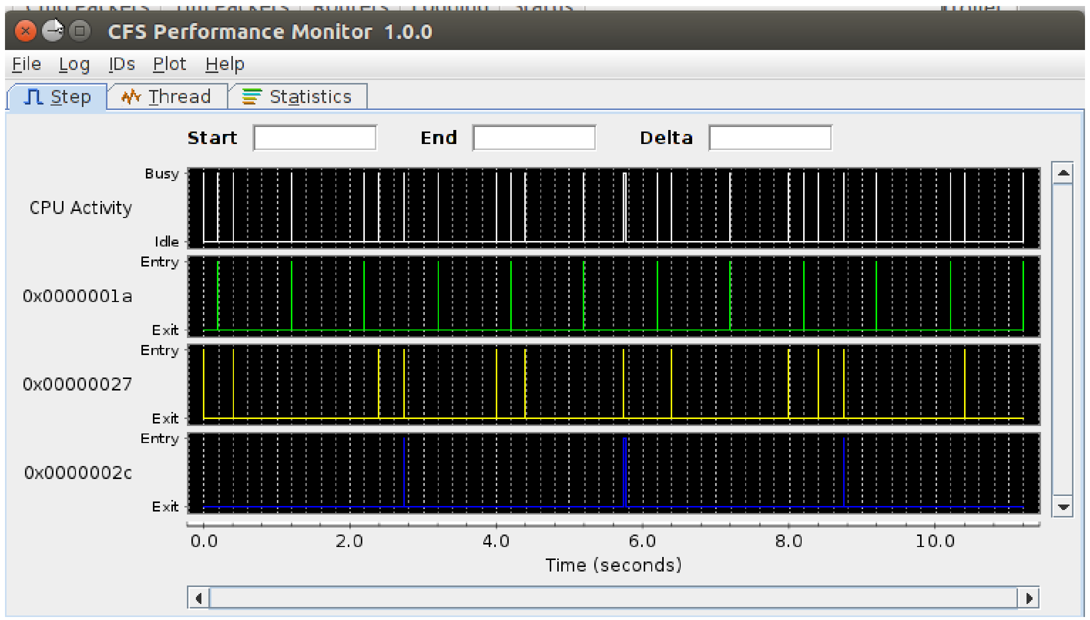
Name	Description	Cmd, Tlm, Tbl Defined	Video	Notes
osk_cpp_lib	Application framework for apps written in C++	n/a	No	Prototype only used by C++ demo app
mqtt_lib	MQTT utilities	n/a	No	Unchanged from Alan Cudmore's repo: https://github.com/alanc98/mqtt_lib
Benchmark (BM)	Provides Dhrystone, Whetstone, and custom cFS measurements	All	No	Prototype needs review and documentation.
File Manager (FileMgr)	Same functions as NASA's FM	All	No	Refactor of NASA's FM app using osk_c_fw. Investigates moving most config parameters to a runtime startup file.
Heater Control (HC)	Simple bang-bang controller for demo purposes	All	No	Used in Limit Checker demo
Heater Sim (HSIM)	Simple heater simulation to provide feedback to HC app	All	No	Used in Limit Checker demo
MQTT	Bridge between a MQTT broker and the software bus	All	No	Early prototype with partial MQTT implementation. Refactor of Alan Cudmore's app from repo: https://github.com/alanc98/mqtt_app.
osk_c_demo	Sample app to illustrate how to use osk_c_fw	All	No	Not all osk_c_fw features demonstrated. Does not include JSON initialization configuration file.
osk_cpp_demo	Sample app to illustrate how to use osk_cpp_fw	All	No	Preliminary prototype.

Tune, Verify, & Validate

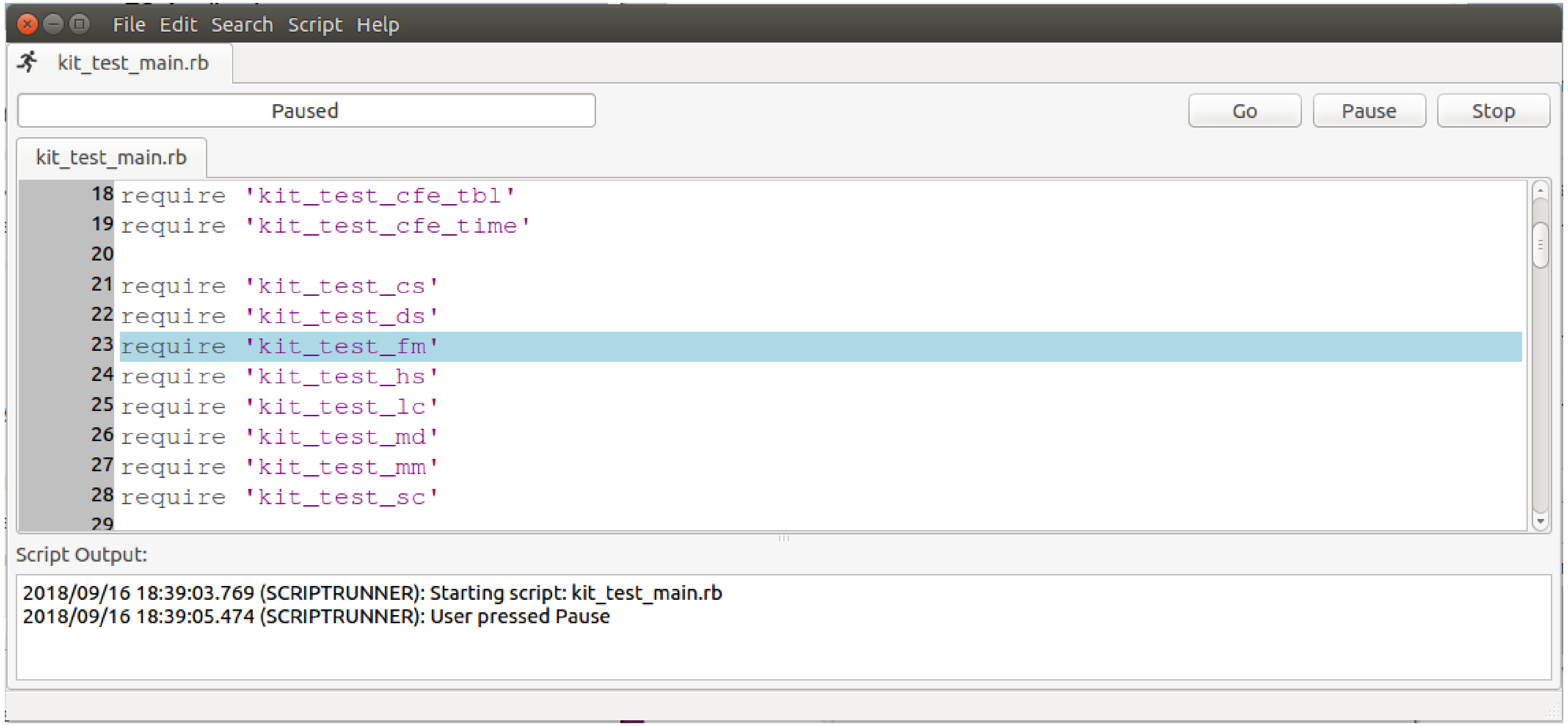
Performance Monitor Tool



- Capture FSW performance data using screen
- Download file and <Launch Analysis Tool>

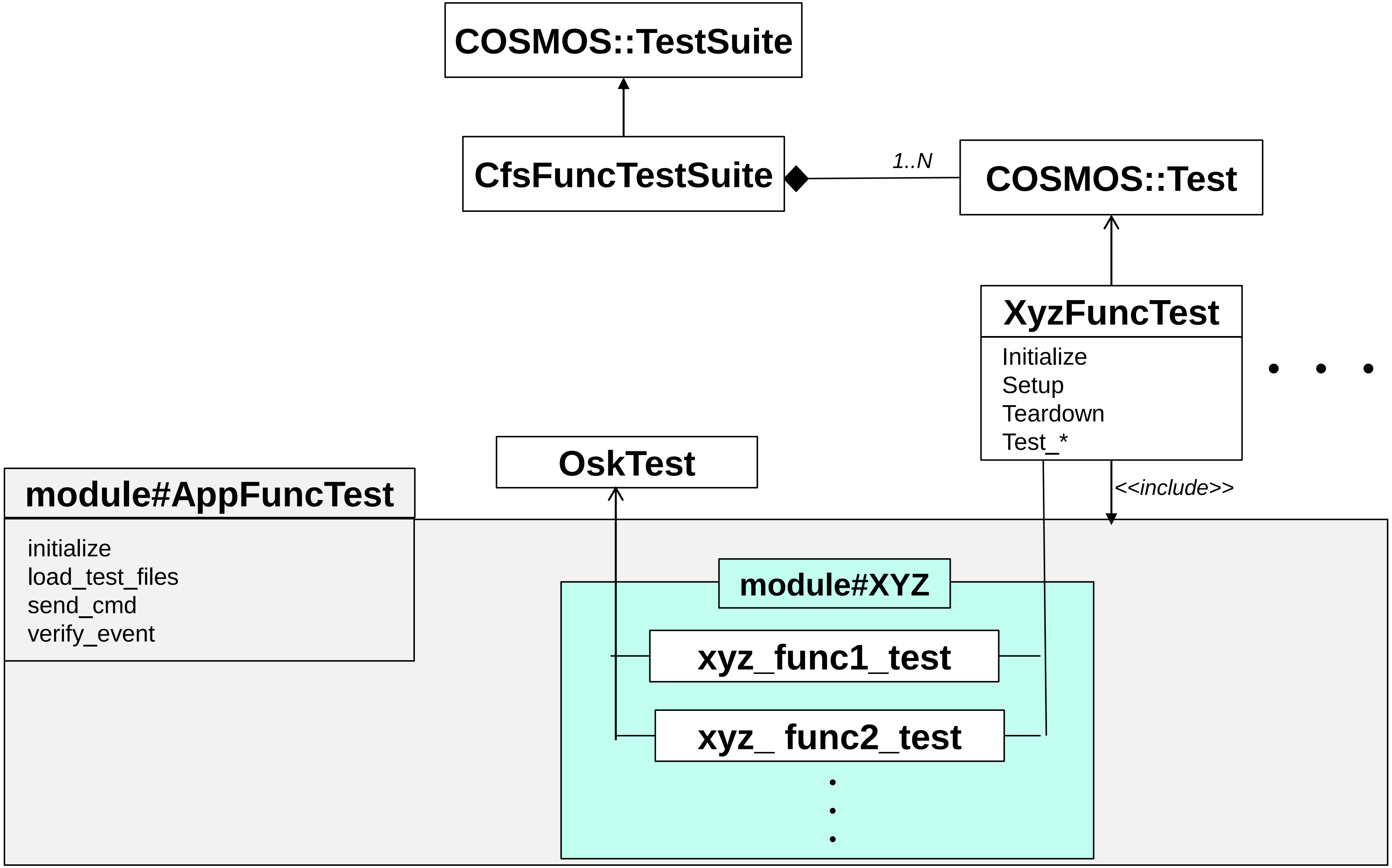


SimSat Integration Script



- **Runs test script using Script Runner**
- **Issues Noop command to every application and verifies telemetry response**

Functional Test Scripts



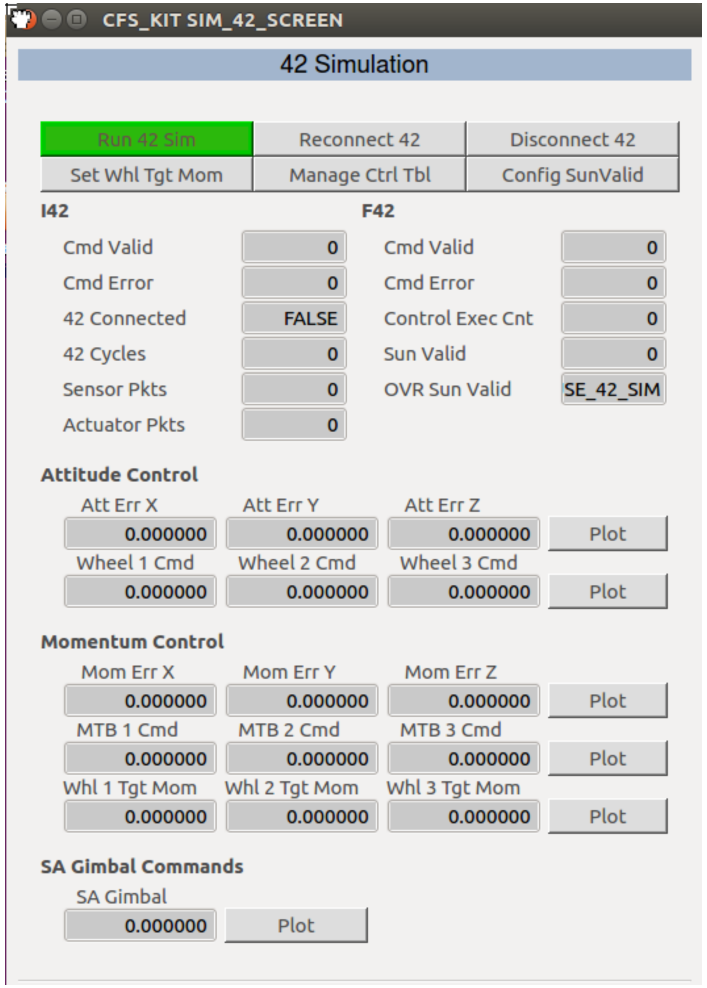
SimSat Operation Script Example

Running with 42 Simulator

Needs updates since v2.4

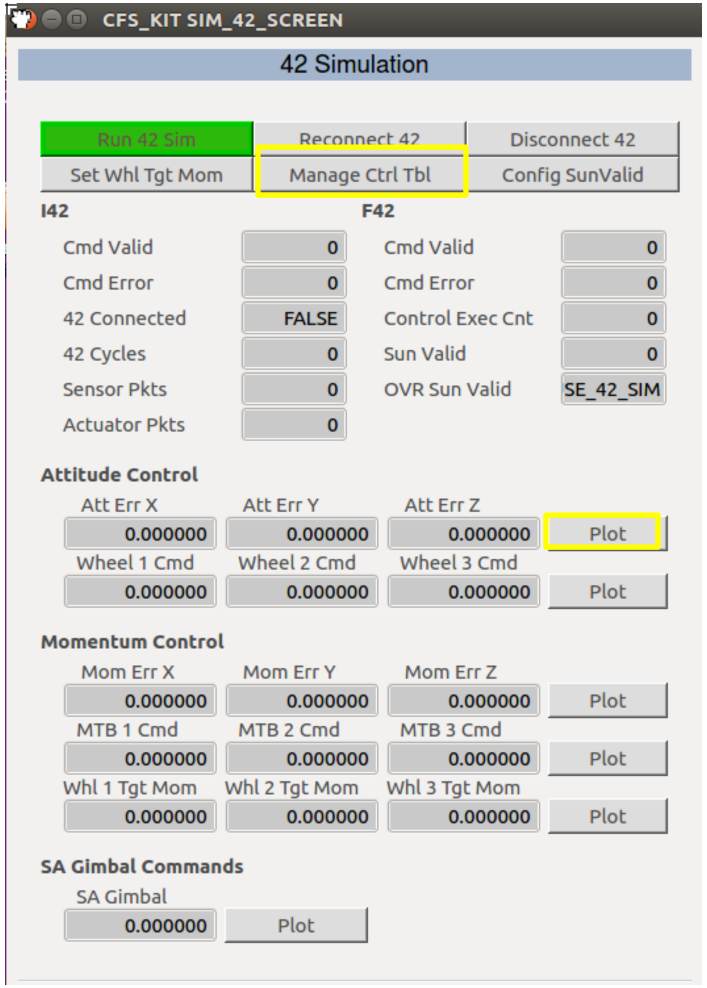
Merge with apply systems engineering processes

Starting the Simulation



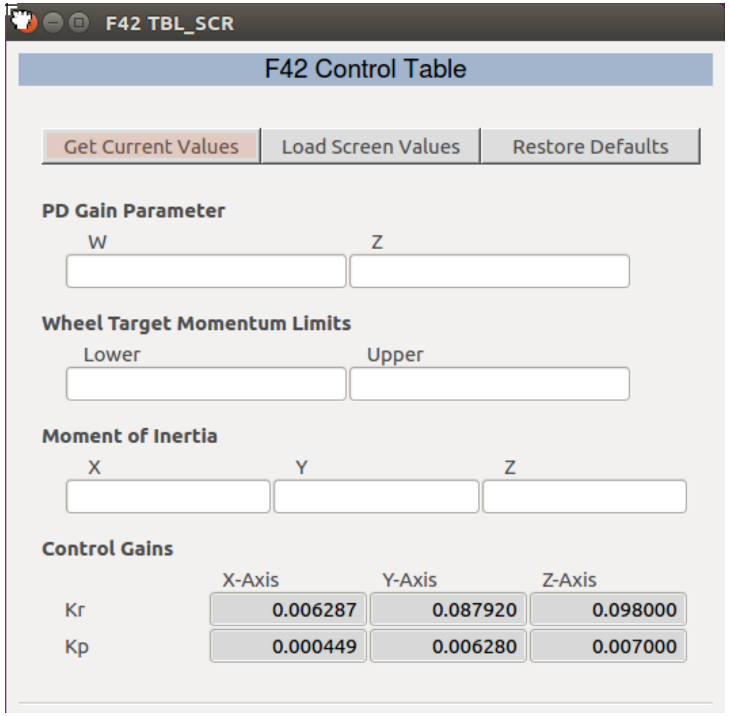
- Select <*Run 42 Sim*> which will start the 42 simulator in a new terminal window.
- The 42 configuration files used in the simulation are located in directory *OpenSatKit/42/OSK*
- The simulation takes a while to initialize

Preparing 42 Simulation

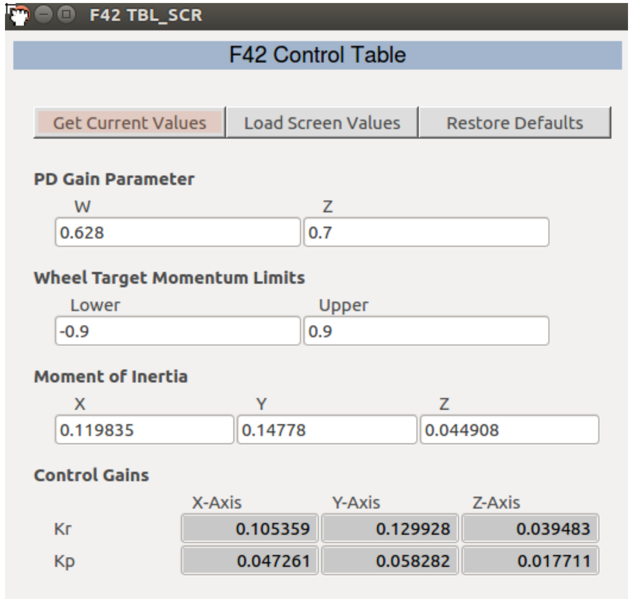


- **From the kit main page on the previous slide select <42 Simulator> and the screen to the left will appear.**
- **The 2nd row of buttons allow you to change the behavior of the control algorithms running in the FSW and are described on the next slides**
- **Before running the sim you will open some additional windows that will be used for your class exercise**
  - Manage Control Table
  - Plot Attitude Errors

Managing Control Table

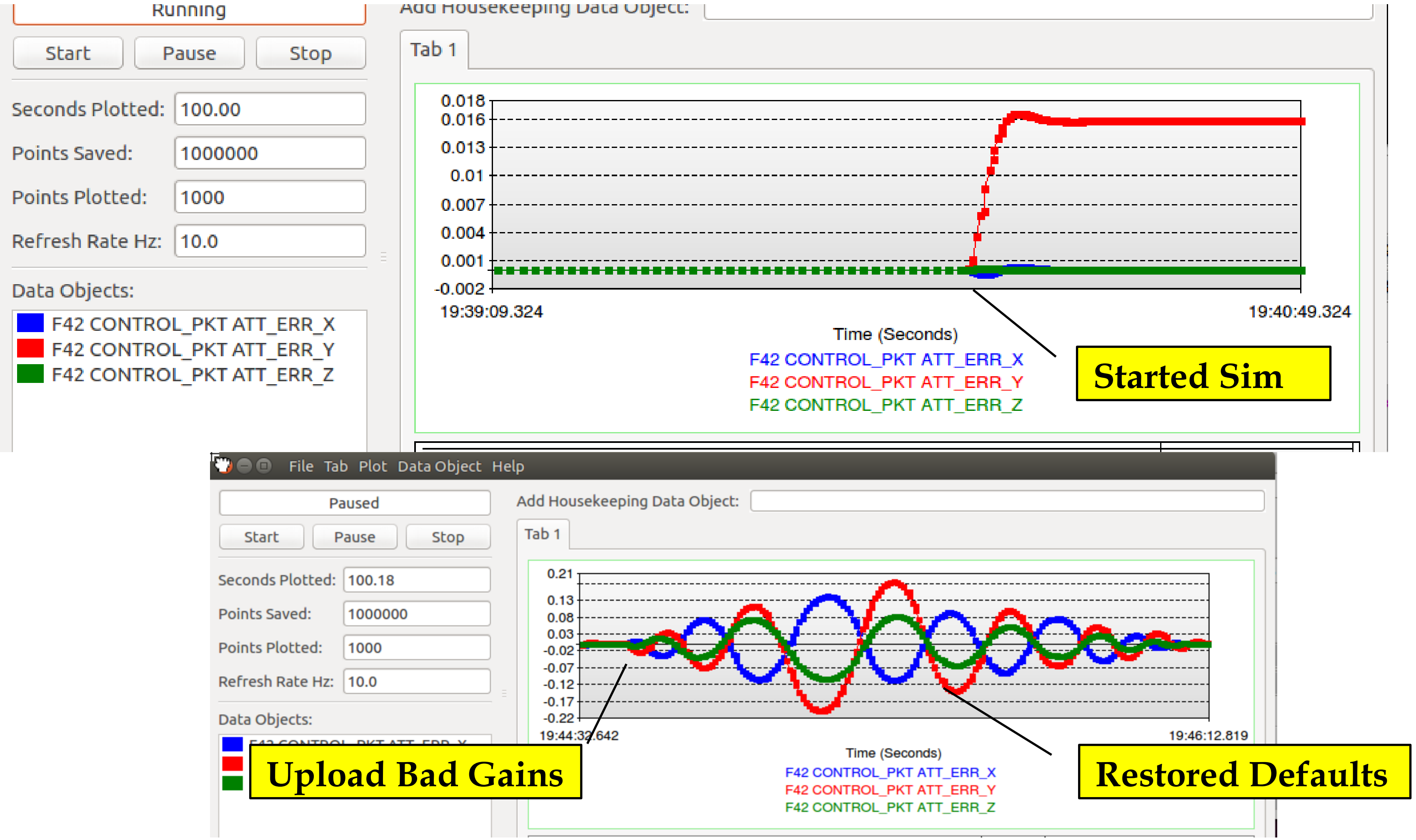


- Selecting <***Manage Control Table***> on the 42 Sim screen produces the screen to the left.
- Select <***Get Current Values***> and it will populate the screen with the current control table values. This takes a little time because it is transferring a file from flight to ground
- Edit the screen as desired and click <***Load Screen Values***> to replace the current control table values
- The defaults can be restored by clicking <***Restore Defaults***>



Plot Attitude Errors

- **Selecting <Plot> button next to the attitude errors produces the screen below**

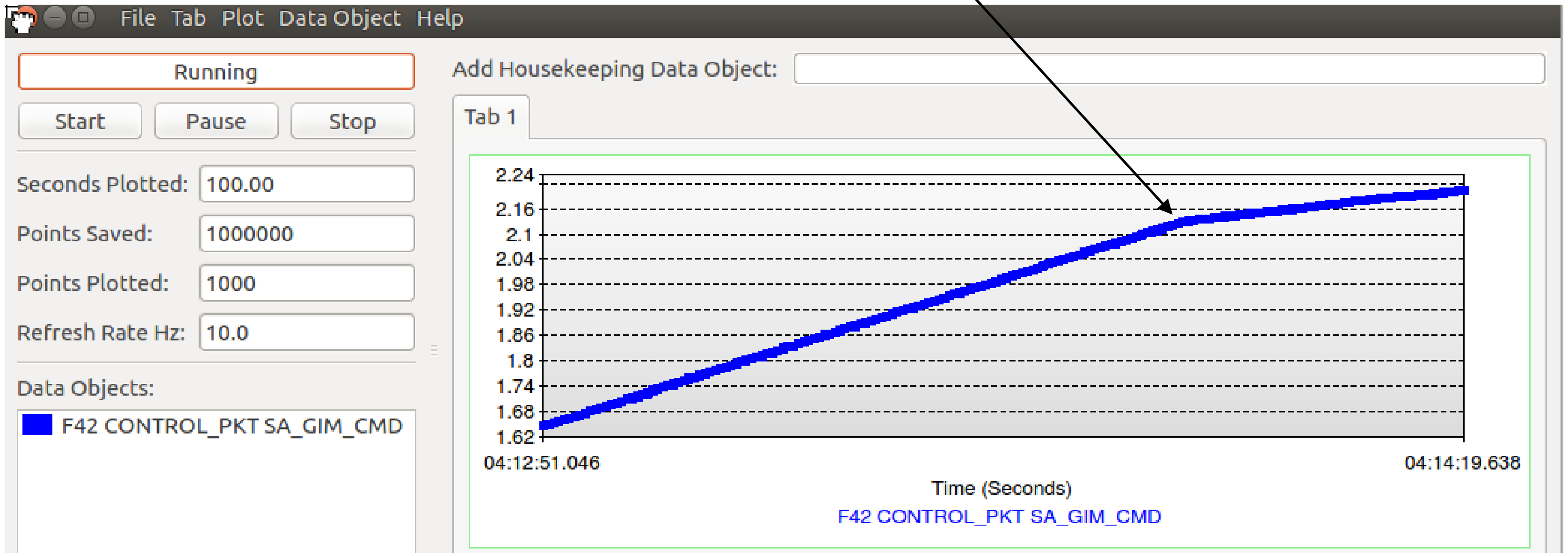


Additional Configuration Options

- **The kit includes two additional configuration options that can be manipulated**
  1. Wheel target Momentum
  2. Sun Valid Configuration

Configure Sun Valid

- Selecting <*Config SunValid*> to override the current sun valid flag
- The plot below shows gimbal command
  - The linear portion had a valid sun and the bend occurred when the SunValid was overridden to false.



Sim Termination

1. Click ***<Disconnect 42>*** to end a 42 simulation that is running with the FSW

2. To terminate the flight software click on the terminal window with the FSW messages and then enter ctrl-c

3. Each of the cosmos windows will need to be closed individually. If you close the COSMOS TlmViewer window first it prompt you to close all of the telemetry screens at once.

Appendix A

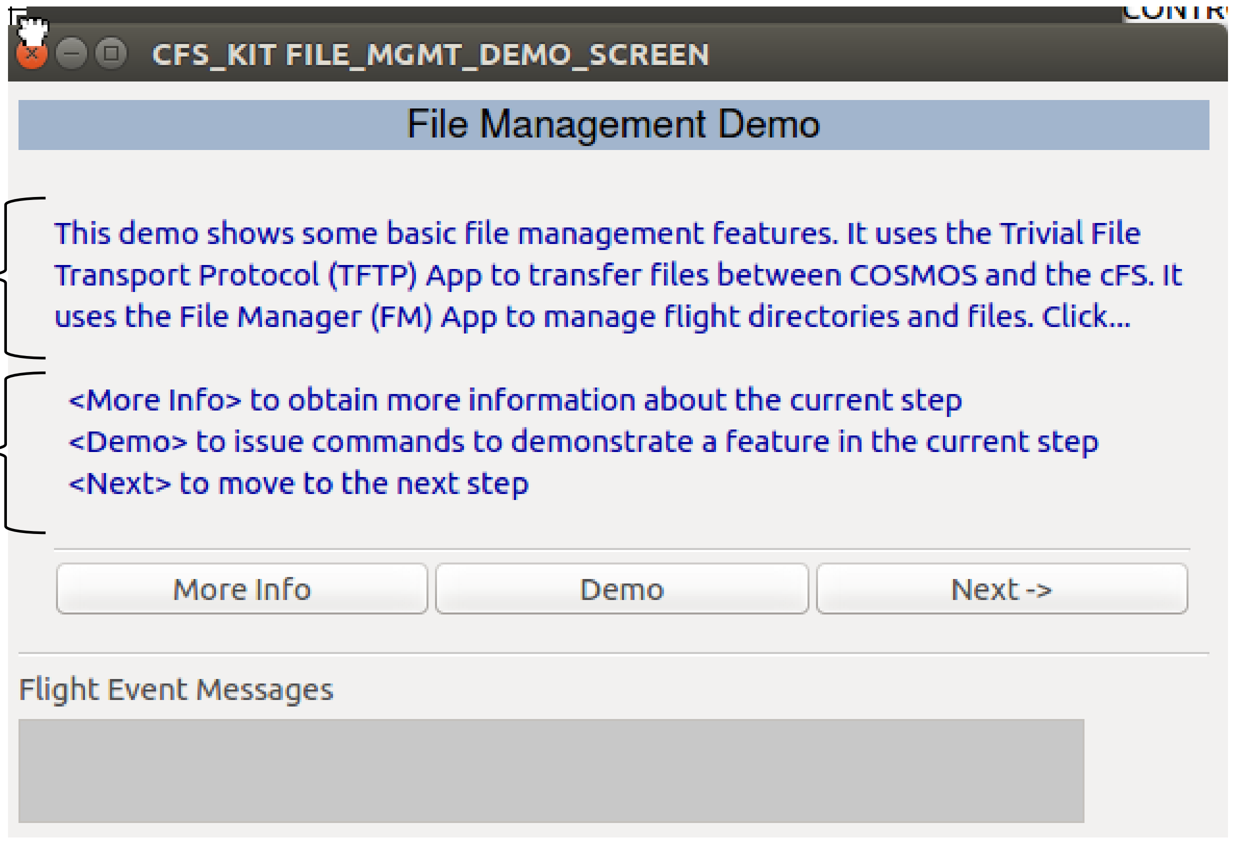
Demos

Screen-based Demos: FM Example (1 of 2)

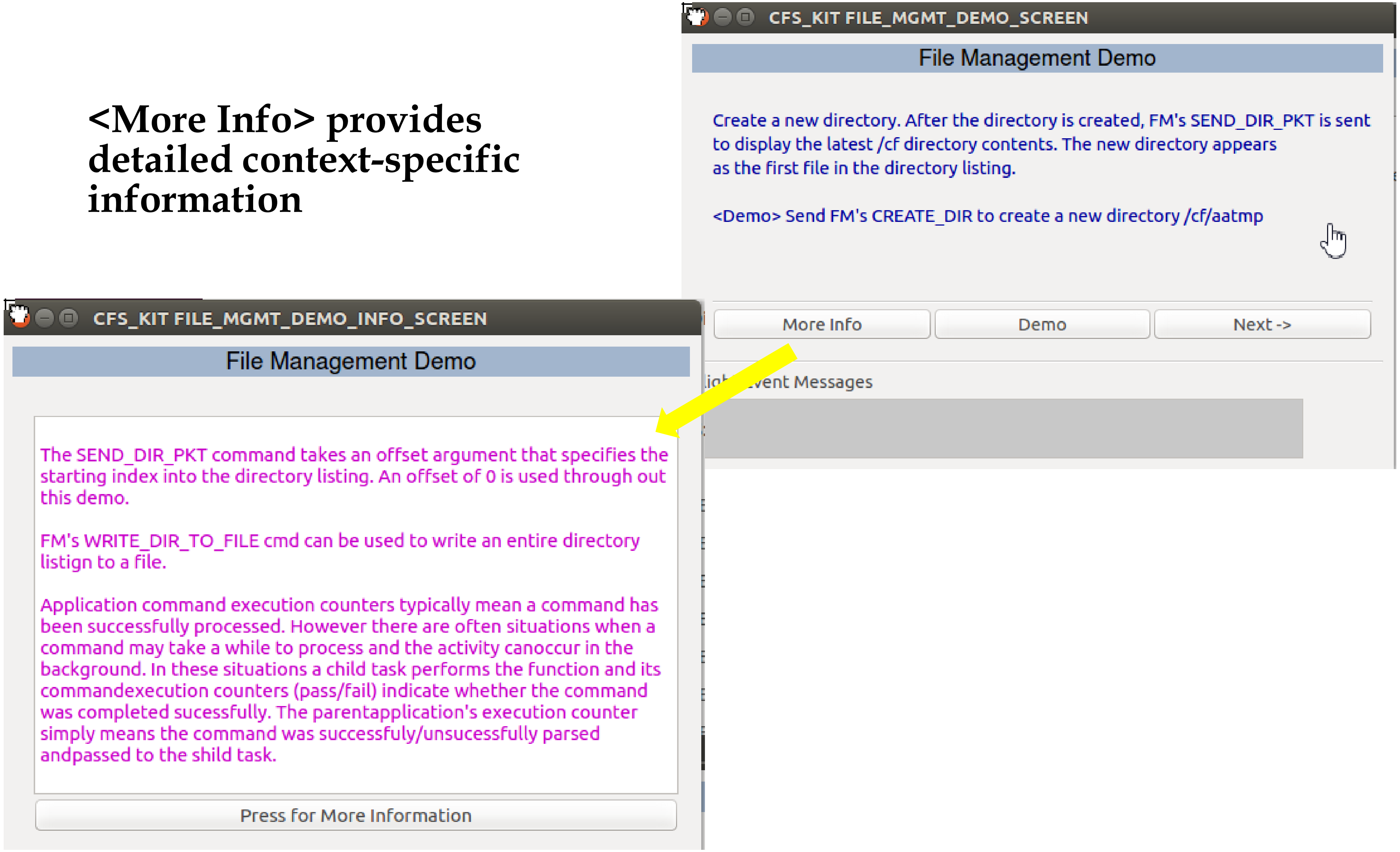
Each demo follows a common user screen configuration

Description of current step

Button usage description



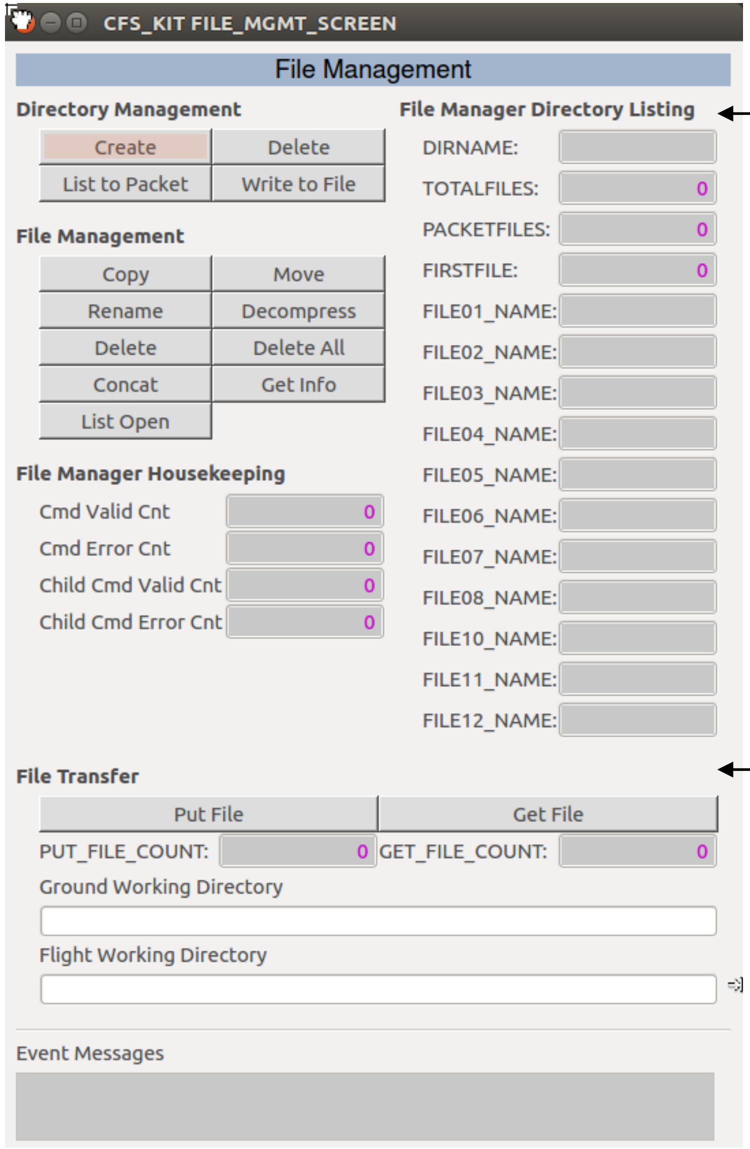
<More Info> provides detailed context-specific information



Appendix B

Application Functional Screens

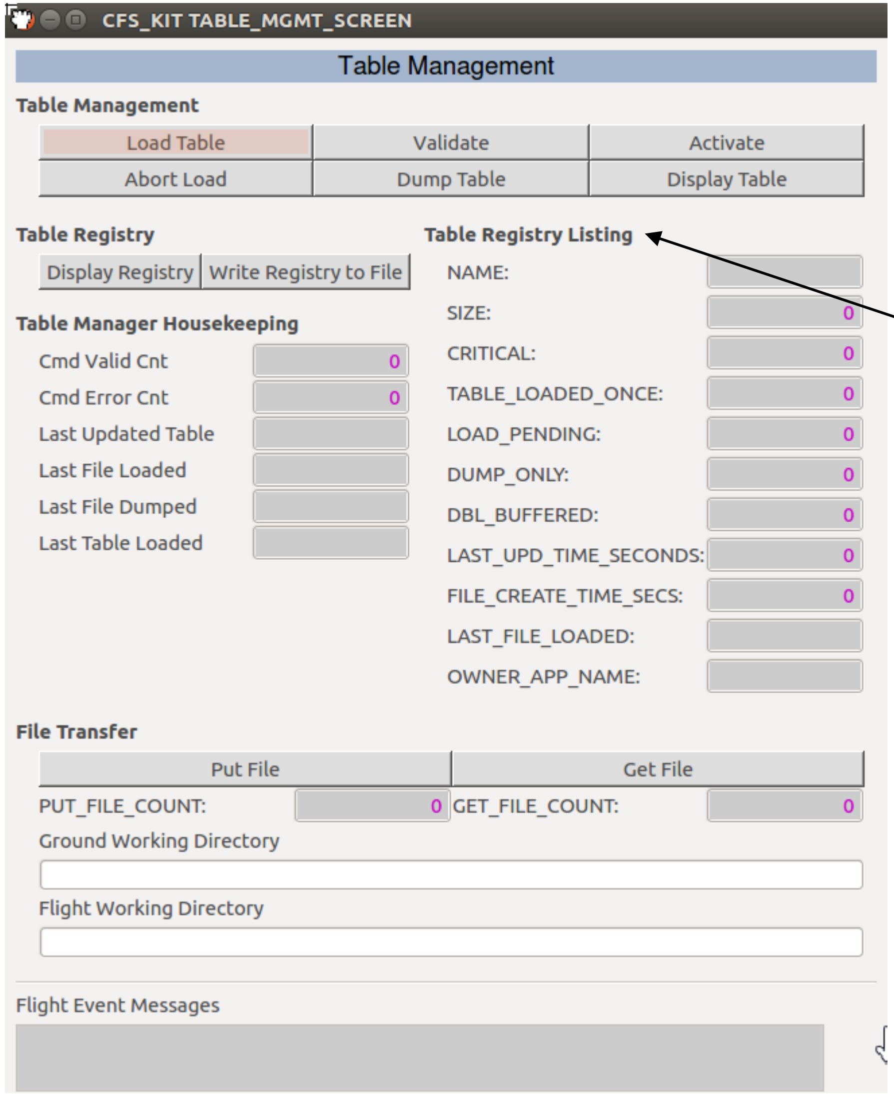
File Management



- <List to Packet> commands File Manage (FM)
  - To send a directory listing
  - The command uses a directory listing alphabetical “offset” to determine which file to start with in the listing
- OSK uses the verbs ***list*** and ***send*** to indicate information is sent in a telemetry packet.
- ***Write*** is used when information is written to a file

- <List to Packet> commands File Manage (FM)
  - To send a directory listing
  - The command uses a directory listing alphabetical “offset” to determine which file to start with in the listing

Table Management

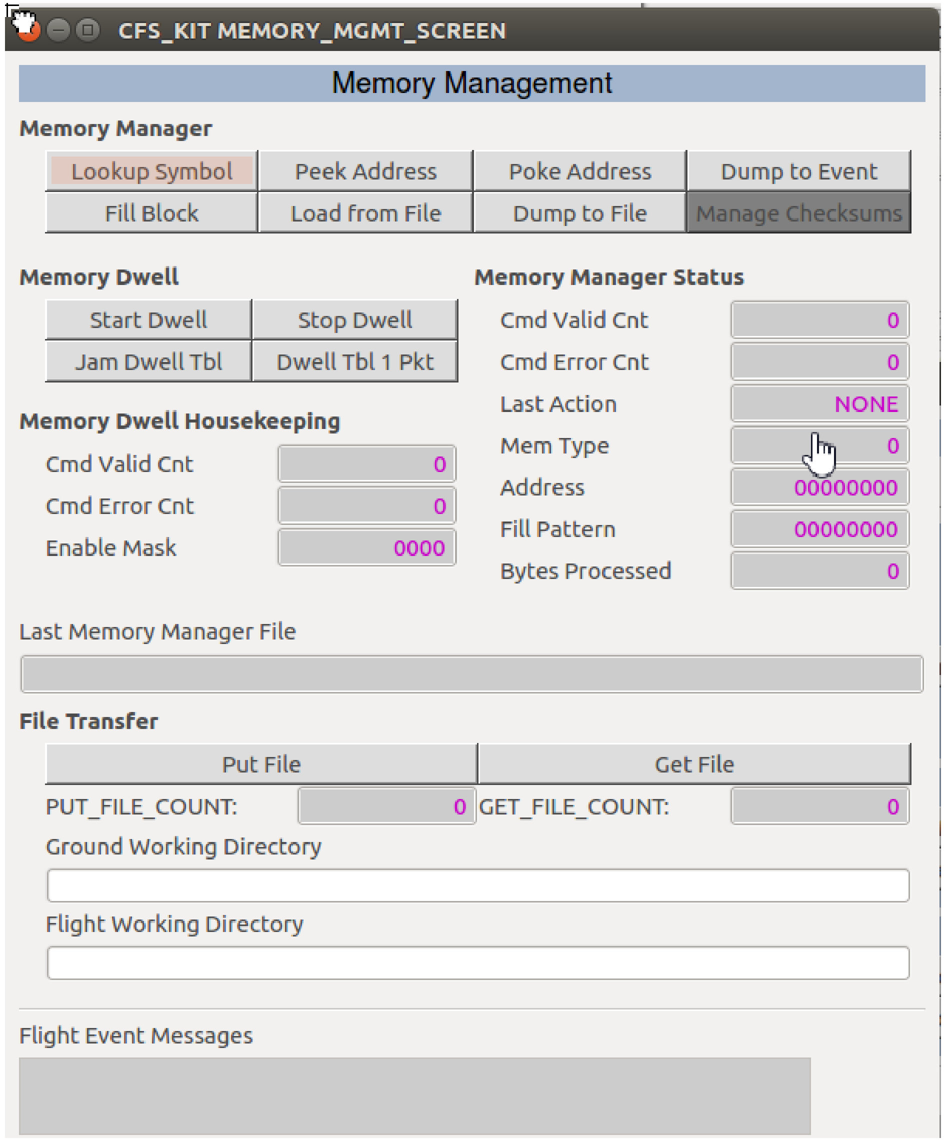


- Load a new FSW table
  <*Put File*> transfers file from ground to flight
  <*Load Table*> into table buffer
  <*Validate*> table via app validation function
  <*Activate*> new table

- <*Display Registry*> sends a table's registry information in a telemetry packet

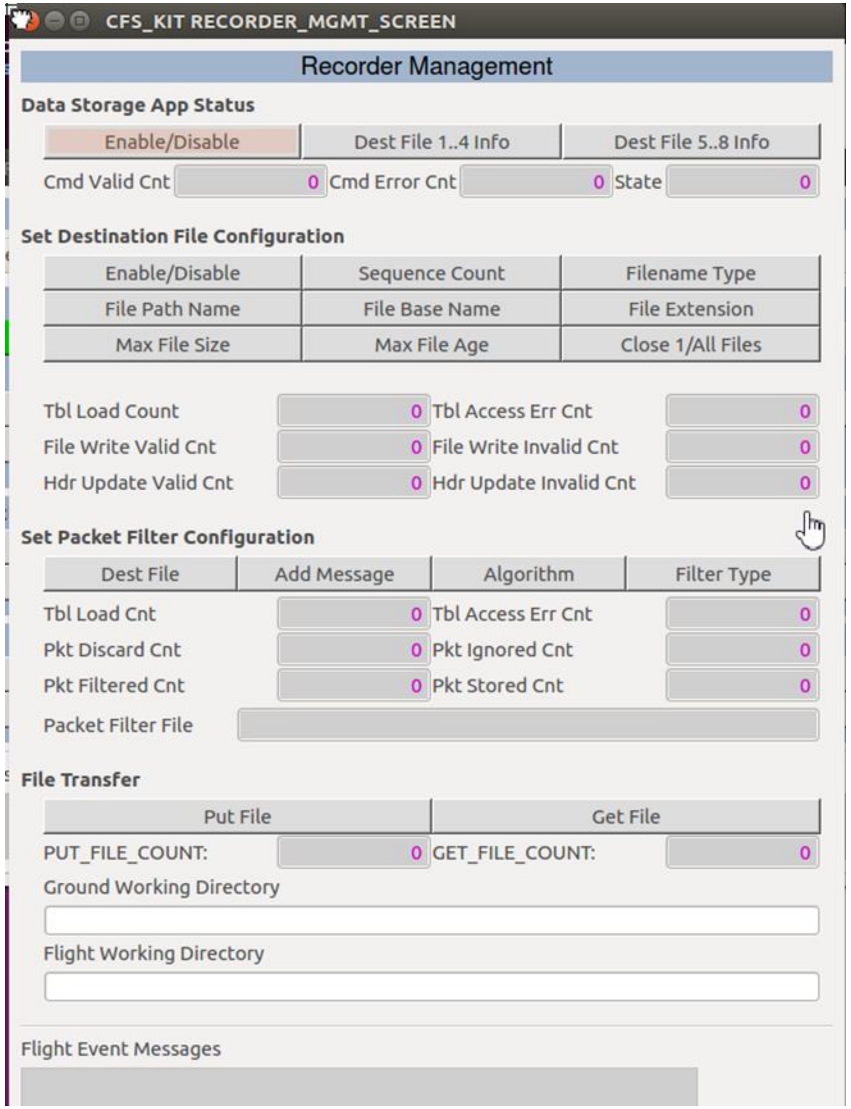
- Dump and display FSW table
  <*Dump Table*> to onboard file
  <*Get File*> transfers file from flight to ground
  <*Display Table*> launches COSMOS Table Manager to view file. Requires binary file definition.

Memory Management

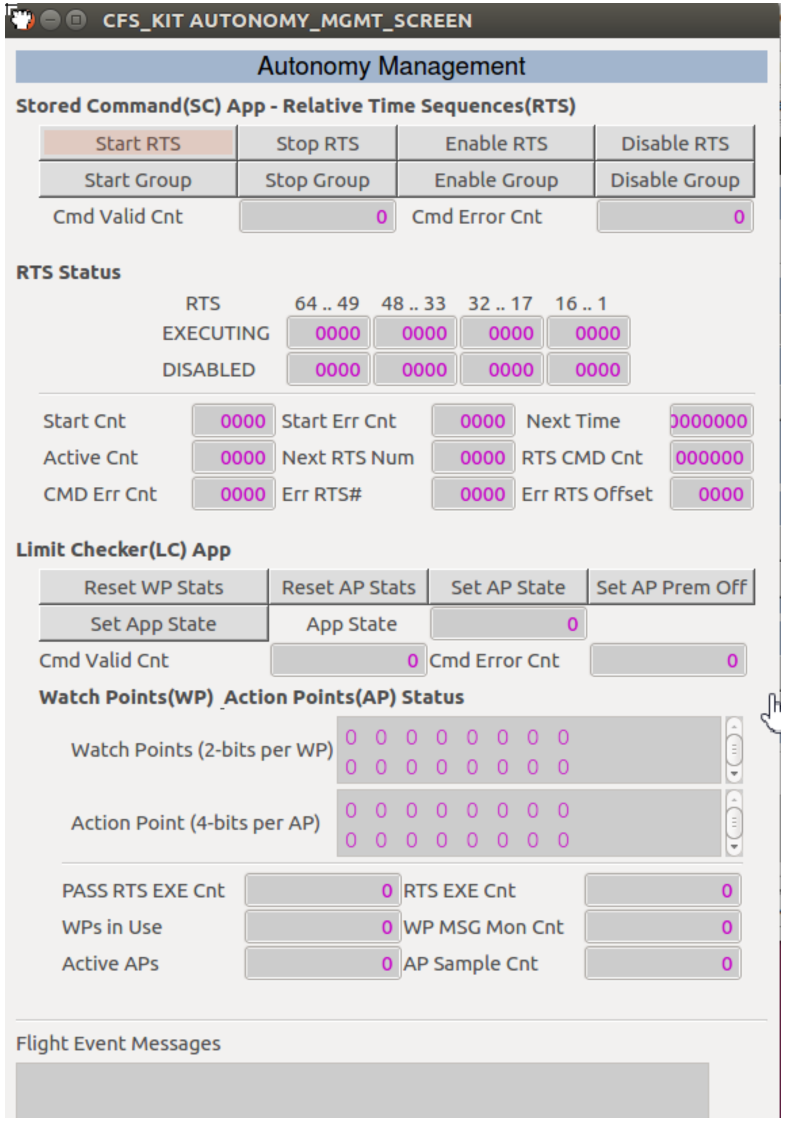


- Memory Manager (MM) and Memory Dwell (MD) apps are typically used for inflight maintenance.
- MM commands allow direct access to any memory location
- MD generates telemetry packets that contain the contents of table-specified memory locations
  - Only 1 dwell table telemetry packet is defined
  - *<Jam Dwell Table>* allows the dwell table to be loaded without using the table load service
- The FSW can easily be corrupted using memory manager
- The memory management demo is a good place to start since it demonstrates MM and MD using safe memory locations

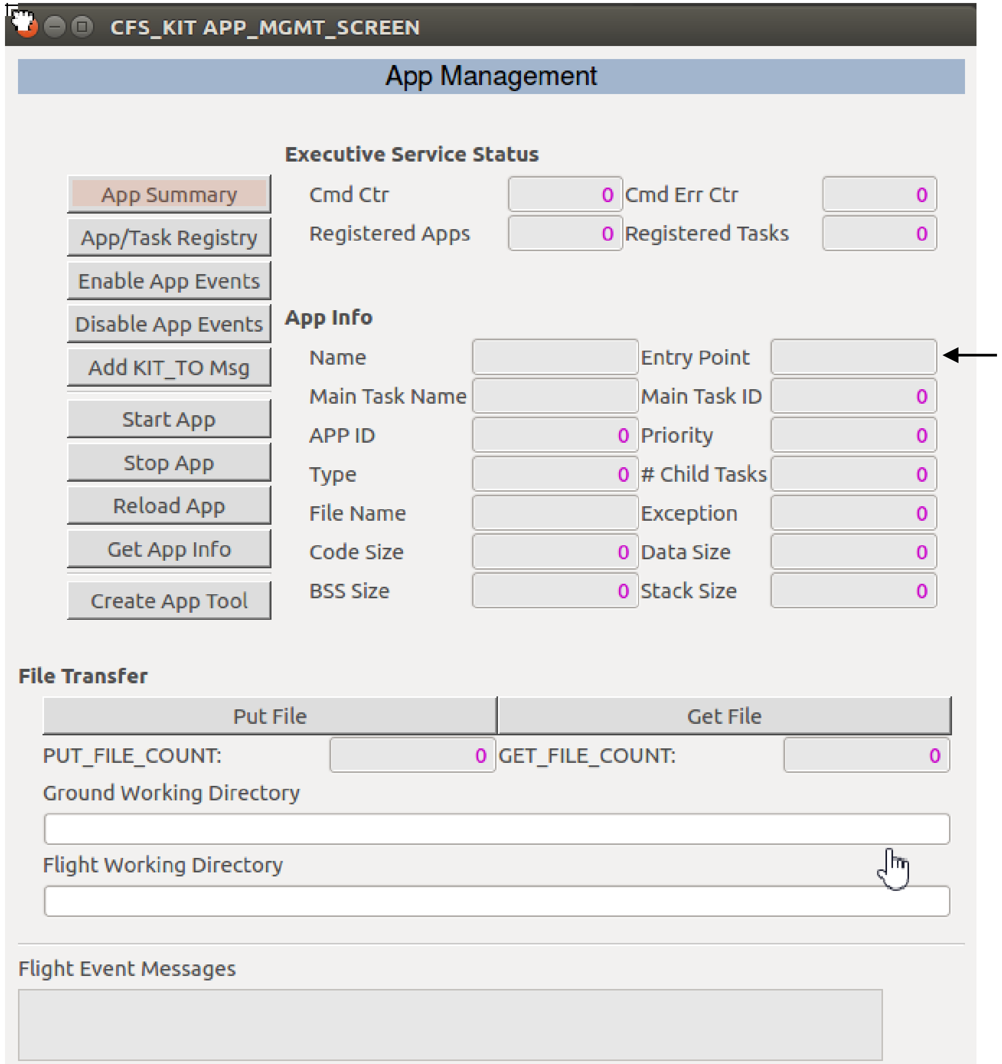
Recorder Management



Autonomy Management



Application Management



- *<Get App Info>* commands cFE executive services to send a telemetry packet with the command-specified app

- *<App/Task Registry>* commands cFE executive services to write app or task information to a file that can be transferred to ground via a *<Get File>*

Appendix C

References

References

1. **4+1 View of UML Diagrams,** http://everspring79.blogspot.com/2008/09/uml-41-view-materials.html
   - There are many articles on the "4+1 Architecture" model and on mapping UML to the 4+1 model. They don't all agree. This one seems reasonable and the purpose for including the model is to encourage the high-level thought processes and not argue detailed points. The cFS design is not documented using any single method.
2. **NASA Operational Simulator for Small Satellites,** http://www.stf1.com/NOS3Website/Nos3MainTab.php