OpenSat cFS Starter Kit User's Guide

Version 1.0 , Released August 7, 2017

Table of Contents

1.0 Introduction

This section describes the motivation for creating the starter kit and a high level introduction to each of the architectural components. If you want to jump in and start using the kit go to Section 2. If you want to use the kit to customize the Core Flight System (cFS) sections 3 and 4 explain how to manage applications and transition from the kit virtual machine to your target hardware platform, respectively. Please keep in mind the starter kit is composed of three complex products and the goal of this documentation is not to explain the details of each product. This document is written from the perspective of a flight software (FSW) developer that wants to use the cFS to control an embedded device.

Section 5 describes how to maintain the kit and delves into the starter kit's design. A typical product user's guide wouldn't include this information, but just as the kit's components are 'open architectures' the kit itself is open and may be modified by the user to be part of an operational system so this information will be helpful to those users.

1.1 Motivation

Until around 2010 the development of spacecraft flight software (FSW) was performed by large organizations that had custom proprietary solutions. FSW ran on processors lagging terrestrial processor performance by orders of magnitude often forcing software engineers to opt for performance over generalized solutions. With the exception of communications satellites many organizations produced 'one off' customized satellites to designed for a specific application. These product development lifecycles were often 5-10 years so even when FSW reuse was occurring the maturation of the reusable artifacts across products was very slow.

During this period from roughly 2005 until 2015, the National Aeronautics and Space Administration (NASA) Goddard Space Flight Center (GSFC) developed the Core Flight System (cFS) and in January 2015 the cFS was released as open source. This was great news for the aerospace community, however there were many challenges with people actually adopting the cFS for their missions. The cFS is a reusable FSW architecture that provides a portable and extendable platform with a product line deployment model. As an open architecture, the cFS can be technically challenging for new users to configure and deploy. In addition, as a government organization, it is difficult for NASA to implement an open source product business model.

This starter kit addresses these issues by providing a fully functioning flight-ground system that runs on a desktop computer. The starter kit components are shown in Figure 1. Ball Aerospace's COSMOS, a user interface for command and control of embedded systems, is used as the ground system. The cFS running on Linux provides a desktop FSW component. The 42 Simulator provides a simulation of spacecraft attitude and orbit dynamics and control. See Appendix B for details on obtaining more information on each of these components.

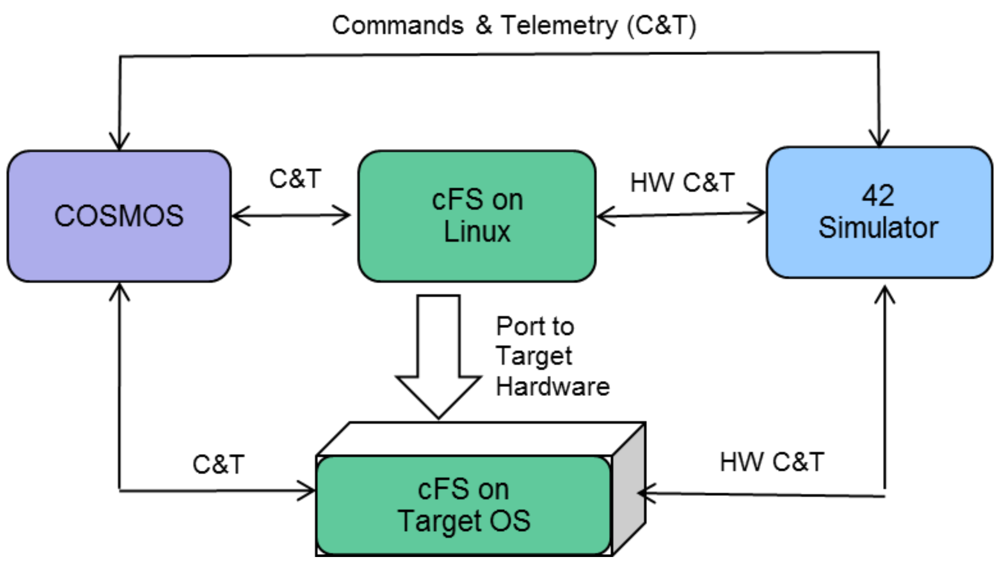


Figure 1-1 – Starter Kit Block Diagram

Starting with an operational flight-ground system makes the FSW developer's job much easier. They can focus on porting the cFS to their target platform, tailoring the kit's cFS components to their needs, and adding new mission-specific components. A future version of the kit will include a low cost commercially available target. This version describes the steps necessary for using the kit to verify that the cFS with the kit apps has successfully been ported to any target.

The starter kit also serves as a cFS training platform. It provides demonstrations to highlight common cFS features and it contains a tool for automatically creating a "Hello World" application. Since it is freely available and easy to install, it can be used as a platform for academic projects.

The cFS can have a significant impact on a mission's FSW costs. The cFS has provided about a third of the FSW on recent NASA missions using source lines of code (SLOC) as a metric and excluding the operating system from the SLOC count. Much of the functionality provided by the cFS is based on decades of FSW experience. This functionality can be very beneficial to inexperienced teams because they may not even recognize that they may need some of the functionality provided by the cFS, especially the inflight diagnostic and maintenance features.

The remainder of the introduction provides a brief description of the cFS, COSMOS, and the kit's architecture. If you are familiar with these components you can skip to Section 2 to get started with using the kit.

1.2 cFS Overview

Before jumping into the cFS architecture it's worth understanding some of the rationale behind the design. Prior to the cFS NASA GSFC FSW reuse efforts had limited success in reducing cost and schedules. Early reuse efforts used a "clone and own" approach where a new project would copy FSW components from one or more previous missions based on functional requirement similarities. This informal source-code based approach to reuse proved difficult for managers to control the scope of the changes and as a result a comprehensive verification and validation effort had to be performed for the new mission which severely limited the cost savings. In addition since FSW components were not configuration managed independent of projects, component quality did not necessarily increase because a single lineage for each component was not maintained.

To address these challenges the Goddard's FSW Branch formed a team of senior engineers to perform a structured heritage analysis across a decade of missions. The initial funding was from non-mission sources which allowed the engineers to participate uninhibited by near-term mission schedules. The diversity of the heritage missions (single string vs. redundant string, varying orbits, different operational communication scenarios, etc.) provided valuable insights into what drove FSW commonality and variability across different missions. The team took the entire FSW life-cycle into consideration, including in-orbit FSW sustaining engineering, as they performed their analysis. Identifying system and application level variation points to address the range and scope of the flight systems domain. The goal was to enable portability across embedded computing platforms and to implement different end-user functional needs without the need to modify the source code. The cFS uses compile-time configuration parameters to implement the variation points. Figure 2 shows the results using a classic software engineering "V-model". The shaded components are cFS artifacts and the <*p*> notation indicates a parameterized artifact. This lifecycle product line approach dramatically increased the number of reusable artifacts and changed how future missions would approach their FSW development efforts.

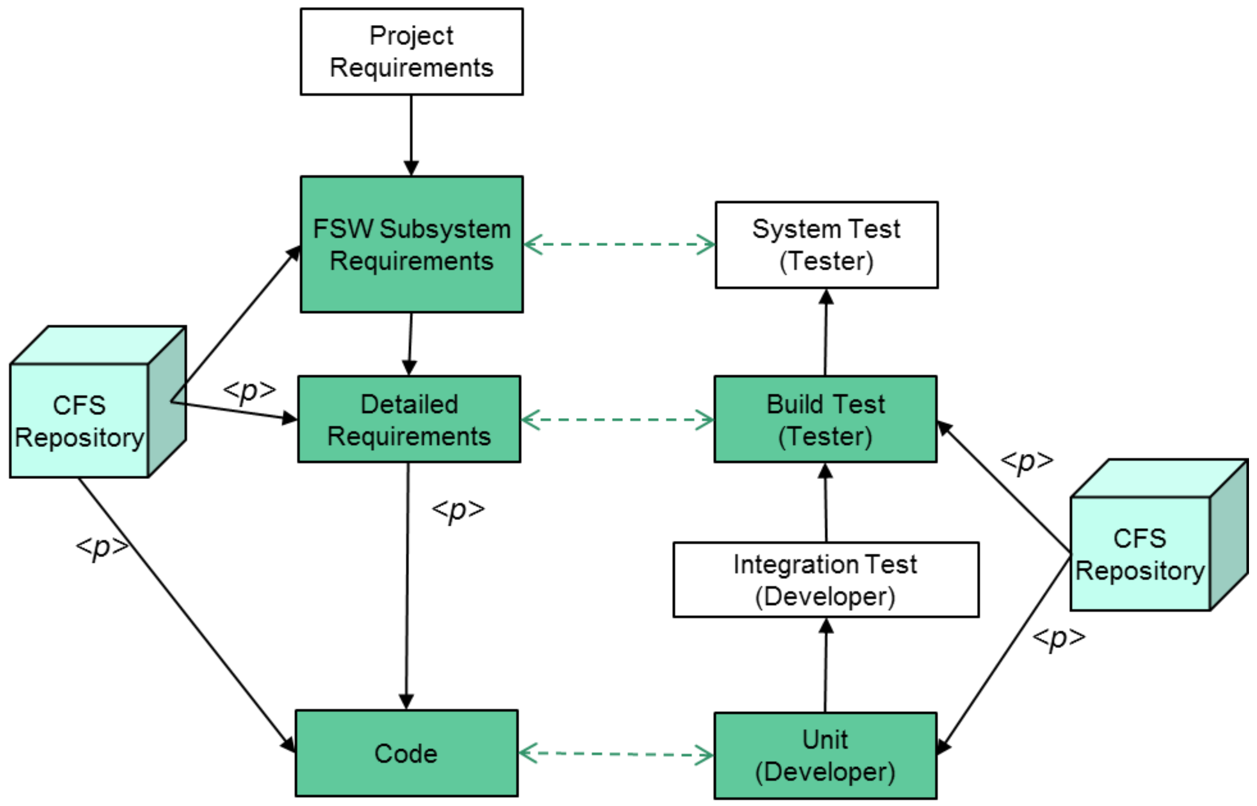


Figure 1-2 – cFS-based Project FSW Lifecycle

TBD – Describe/define the components in Figure 2

All of the artifacts in Figure 2;s *cFS Repository* boxes are available as part of the open source release. The starter kit's features primarily support the developer's integration activities. There are tools that create a "hello world" application and a corresponding unit test harness. The kit is not intended to be an Integrated Development Environment (IDE) so the user is free to use the development environment of their choosing. Once an application is mature the kit supports the integration of an application into the kit. Build testing support may be added as a future enhancement.

1.2.1 cFS Architecture

While a majority of the heritage analysis focused on FSW functional features a significant and conscious effort was made to address the cFS's architectural quality attributes such as portability, performance, scalability, interoperability, verifiability, and complexity. Figure 3 illustrates the cFS architecture and two fundamental architectural features are the Application Program Interface (API)-based layers and the definition of an application as a distinct well-defined architectural component. Applications can easily be integrated into the build system and even dynamically added/removed during runtime.

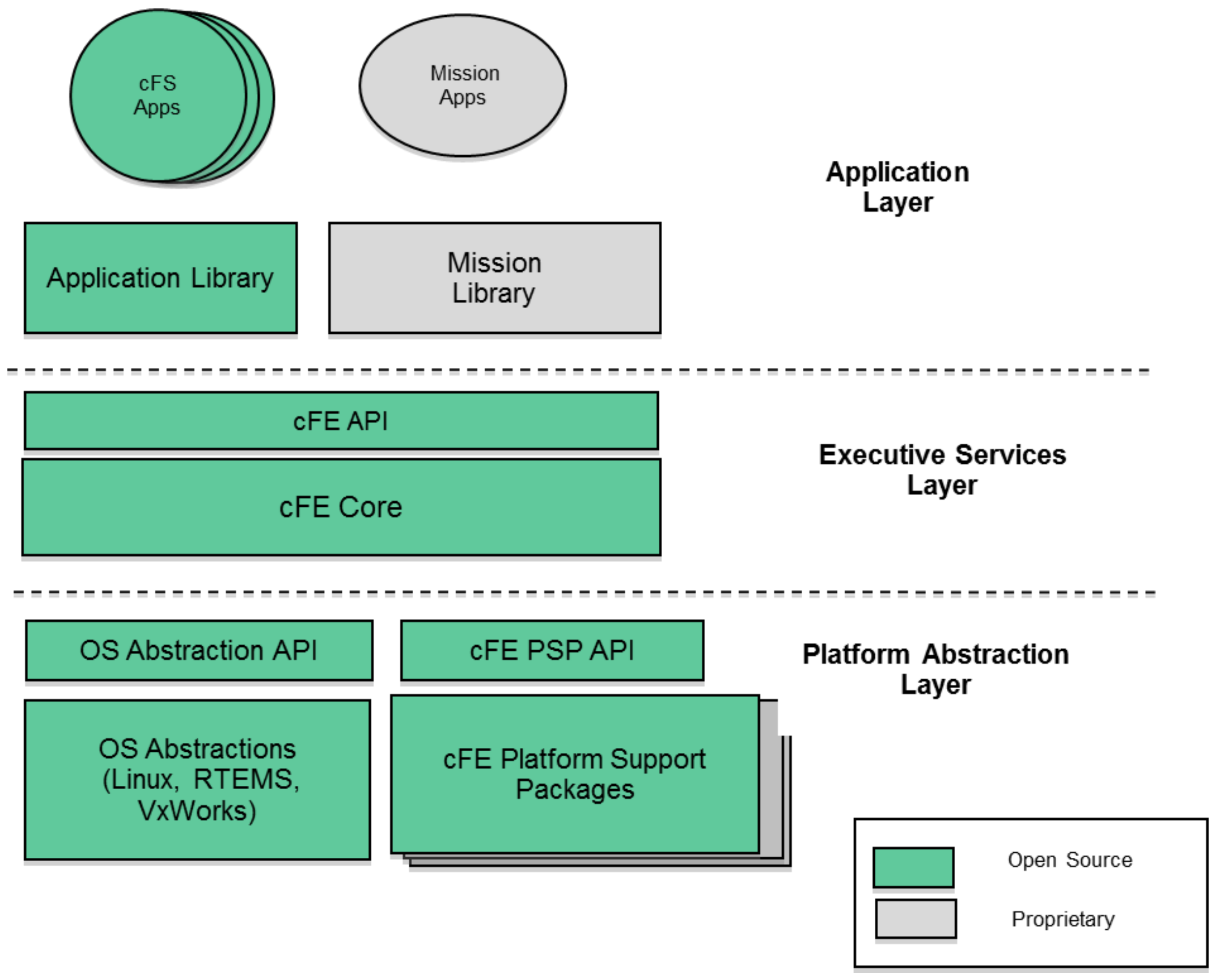


Figure 1-3 – cFS Layered Architecture

The cFS defines 3 layers with an API between each layers. Layer 1 supports portability by decoupling the higher levels from hardware and operating system implementation details. All access to the platform is controlled through two APIs: the Operating System Abstraction Layer (OSAL) and the Platform Support Package (PSP).

Layer 2 contains the core Flight Executive (cFE) that provides five services that were determined to be common across most FSW projects. The core services include a Software Bus (messaging), Time Management, Event Messages (Alerts), Table Management (runtime parameters), and Executive Services (startup and runtime). The Software Bus provides a publish-and-subscribe CCSDS standards-based inter application messaging system that supports single and multi-processor configurations. Time Management provides time services for applications. The Event Message service allows applications to send time-stamped parameterized text messages. Four message classes based on severity are defined and filtering can be applied on a per-class basis. Tables are binary files containing groups of application defined parameters that can be changed during runtime. The table service provides a ground interface for loading and dumping an application's tables. Executive Services provides the runtime environment that allows applications to be managed as an architectural component. All of the services contain tunable compile-time parameters allowing developers to scale the cFE to their needs.

The APIs in Layers 1 and 2 have been instrumental in the cFS' success across multiple platforms and the cFE API has remained unchanged since the launch of the Lunar Reconnaissance Orbiter in 2009. The APIs, their underlying services, and the cFS build tool chain provide the architectural infrastructure that make applications an explicit architectural component. A cFS compliant application will run unchanged regardless of the host platform. The application layer contains thread-based applications as well as libraries (e.g. linear algebra math library) which can be shared among multiple applications.

As shown in Figure 3 all of the source code has been released as open source. The code is managed by a multi-NASA Center configuration control board (CCB) that ensures that the application context will evolved in a controlled manner.

1.2.2 cFS Application Context

The application layer is where the bulk of the cFS scalability and extendibility occurs. Users create new missions using a combination of existing cFS compliant apps (partial or complete reuse) and new mission-specific apps. Just as the cFE provides common FSW services there is a set of apps that provide common higher level functional services. Figure 4 shows the minimal context for a user app on a single processor system. Three 'kit' apps provide the higher level services. The details of why they're kit apps is explained in Section TBD.

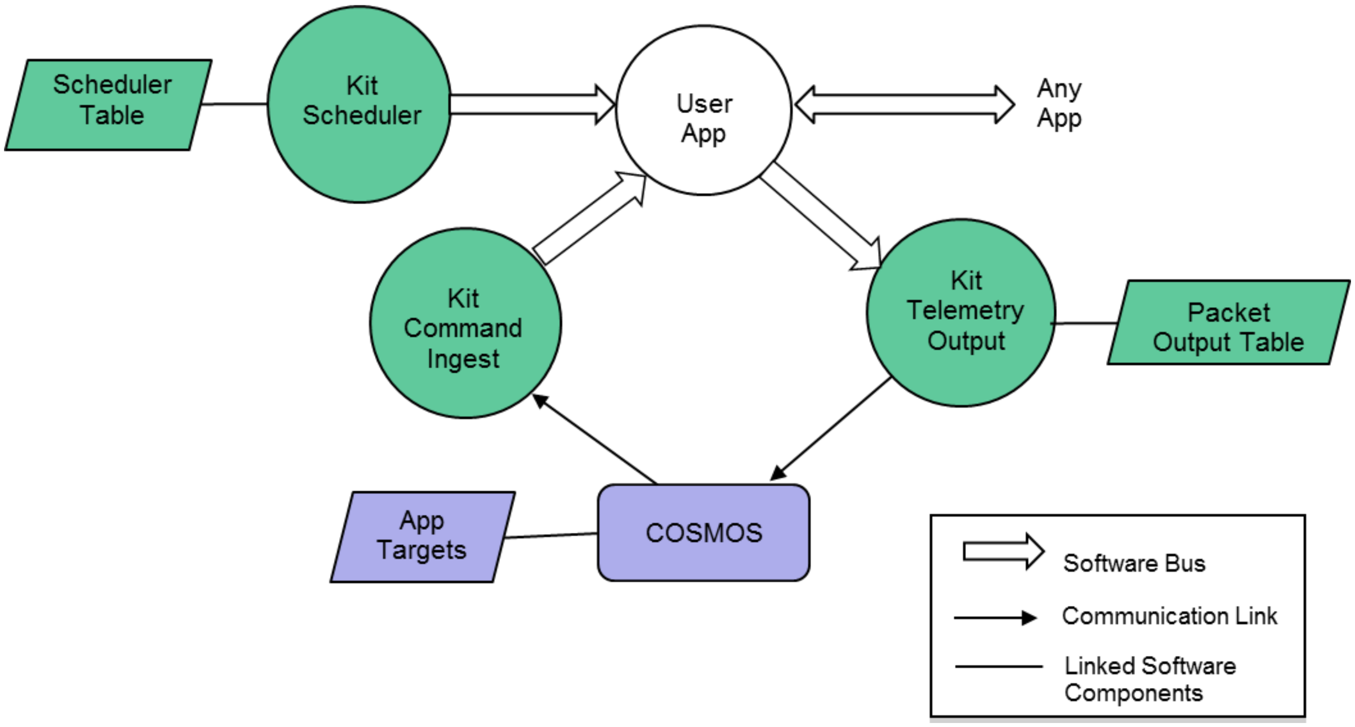


Figure 1-4 – User Application Context

Apps must have the ability to receive commands from and send telemetry to the ground system. The Command Ingest app receives commands from the ground and sends them on the software bus. The software bus uses the command message identifier to route the command to the app that has subscribed to the message id. An app will also generate one or more telemetry packets and send them on the software bus. The Telemetry Output app uses a table to determine which message ids to subscribe to and how often to forward them to the ground system.

Users have multiple mechanisms for how to control the execution of an application. The scheduler app provides a time synchronized mechanism for scheduling application activities. The Scheduler app uses a table to define time slots for when to send a message that users can use to initiate an activity. Activities can be scheduled to occur faster or slower than 1 second. Even if an app's execution is data driven (.i.e. pends for one or more data packets to start its execution) it is often convenient to use the scheduler as a means to send time-based housekeeping telemetry.

1.3 COSMOS Overview

Ball Aerospace's COSMOS is a freely available open source command and control system for operations and test of an embedded system. A set of 15 applications provide automated procedures, real-time and offline telemetry display and graphing, logged data analysis and Comma Separated Variables (CSV) extraction, limits monitoring, command and telemetry handbook creation, and binary file editing. COSMOS scripting offers the full power of the Ruby programming language allowing operators to send commands, verify telemetry, read and write files, access the network, and even send an email upon completion. Advanced debugging functionality allows for single-stepping through procedures, setting breakpoints, and complete logging of all script and user interaction with the system. Detailed data visualization allows for custom screen creation, line and x-y plotting of data, and easy creation of custom 3d visualizations. Offline data analysis and data extraction capabilities make narrowing down anomalies easy.

This user's guide describes the components of COSMOS that are relevant to using the starter kit. For a complete description of COSMOS refer to the documentation at http://cosmosrb.com.

TBD – Targets, CmdTlmServer, TlmViewer, ScriptRunner

1.4 42 Simulator Overview

42 is an open source software package that simulates spacecraft attitude and orbital dynamics and control. 42 is design to be both powerful and easy to configure and run. It supports multiple spacecraft anywhere in the solar system and each spacecraft is a multi-body model that can be a combination of rigid and flexible bodies. 42 consists of a dynamics engine and a visualization front end. The two components can run on the same processor, different processors, or just the dynamics can be run without visualization.

Figure 5 shows the processing flow of the 42 simulation models. The Ephemeris Models determine object (spacecraft, sun, earth, etc.) positions and velocities in a particular reference frame. This information is input to the Environmental Models that computes the forces and torques exerted on each object. The ephemeris and environmental data is read by the Sensor Models. The FSW algorithms read the sensor data, estimate states, run control laws, and output actuator commands. The Actuator Models compute control forces and torques. The forces and torques from Environmental Models and Actuator Models are input the Dynamics Model that integrates the dynamic equations of motion over a time step. The new states are fed back to the Ephemeris Models and the simulation process is repeated.

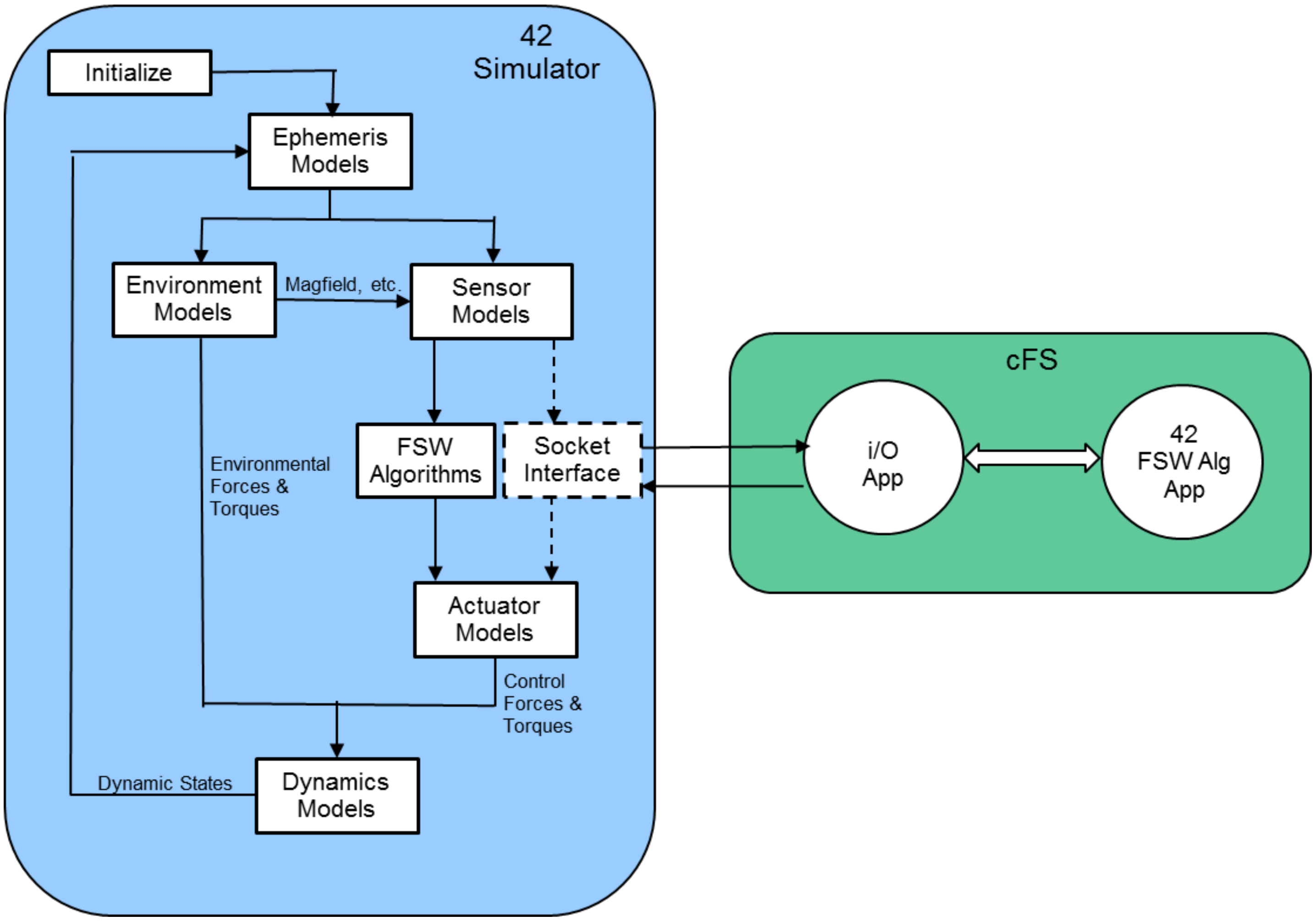


Figure 1-5 – 42 Simulator

The dashed Socket Interface box in Figure 5 has been added to the 42 simulator for the OpenSatKit and replaces the FSW Algorithm box. The FSW Algorithm App running on the cFS implements the 42 FSW algorithms. The I/O App communicates with the new 42 Socket Interface to transfer sensor and actuator data between 42 and the cFS platform. 42 is command line driven which allows it to be controlled by and external program such as COSMOS. This control is not shown in Figure 5.

1.5 PiSat Target

TBD – Describe PiSat

2.0 Using Starter Kit Features

This section describes the starter kit features and how to use them. Everything in the kit is open source so if there's a feature you like you can use the underlying implementation to customize a solution to meet your needs. Note the kit is typically run in a desktop VM and it is not intended to be configured to meet real-time embedded FSW requirements. Therefore it isn't helpful, for example, to create a scheduler table that schedules an app at 10 Hz. The kit assists in functionally integrating apps and Section 5 outlines steps for transitioning from the kit to an embedded system.

2.1 Launching the Starter Kit

After you have installed the starter kit following the instructions in Appendix B you will have the following directory structure as shown in Figure 2-1.

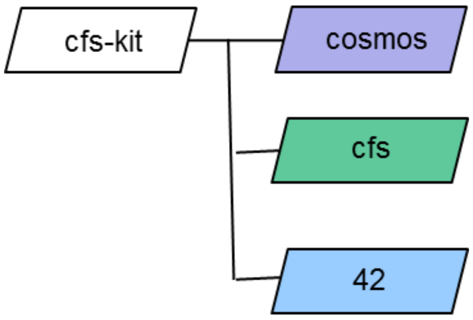


Figure 2-1 –Starter Kit Directory Structure

Start the kit by performing the steps below. When you install the kit for the first time COSMOS is launched as the final installation step so you can start with step 4 in this situation.

1. Create a terminal window by entering
    - ***CTRL-ALT-T***
2. By default you will be in in your home directory. Change the directory to the COSMOS base directory by entering
    - ***cd cfs-kit/cosmos***
3. Start COSMOS by entering
    - ***ruby Launcher***
    - This starts the COSMOS Tool Launcher window as shown on the left side of Figure 2-2.
4. Start the starter kit by clicking on the
    - ***cFS Start Kit*** button
    - This starts the COSMOS Command and Telemetry Server and Telemetry View tools. The server is needed to connect to the cFS and Telemetry Viewer is sued to launch the kit's the kit's main window as shown on the right side of Figure 2-2.

5. Start the cFS by clicking on the green
   - ***Start cFS*** button
   - This creates a new terminal window as shown in Figure 2-3 and starts the cFS within in the window. A series of startup messages are displayed. The startup window details are

At this point you have a running system and can start to explore the kit's features.

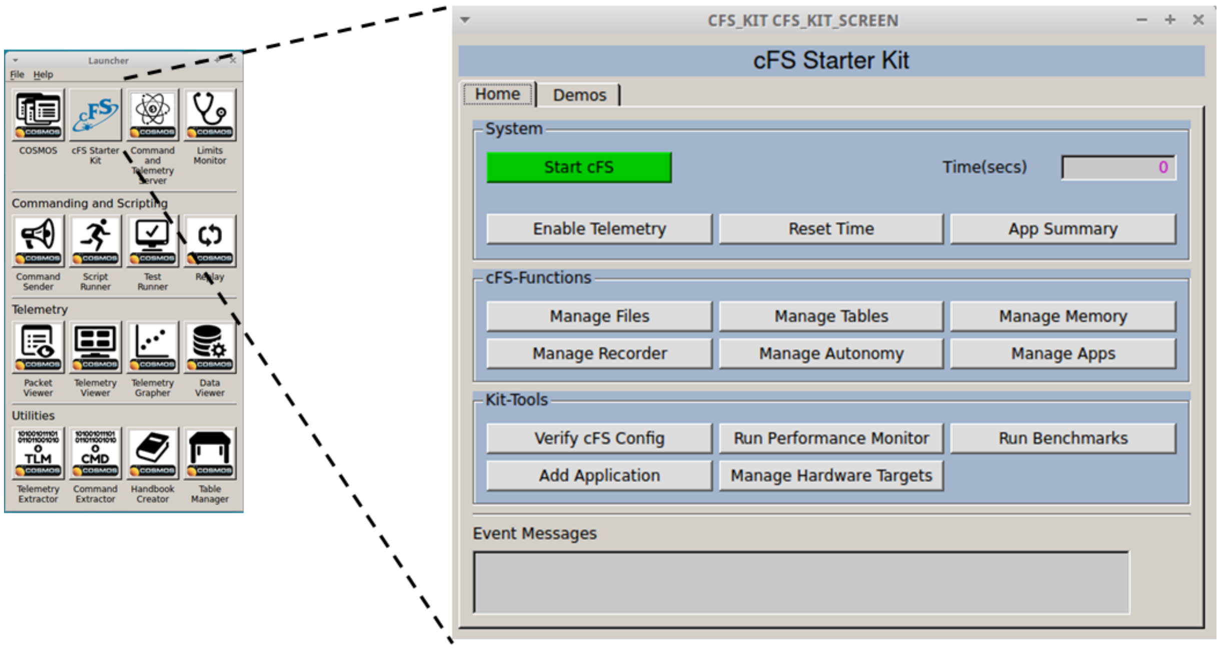
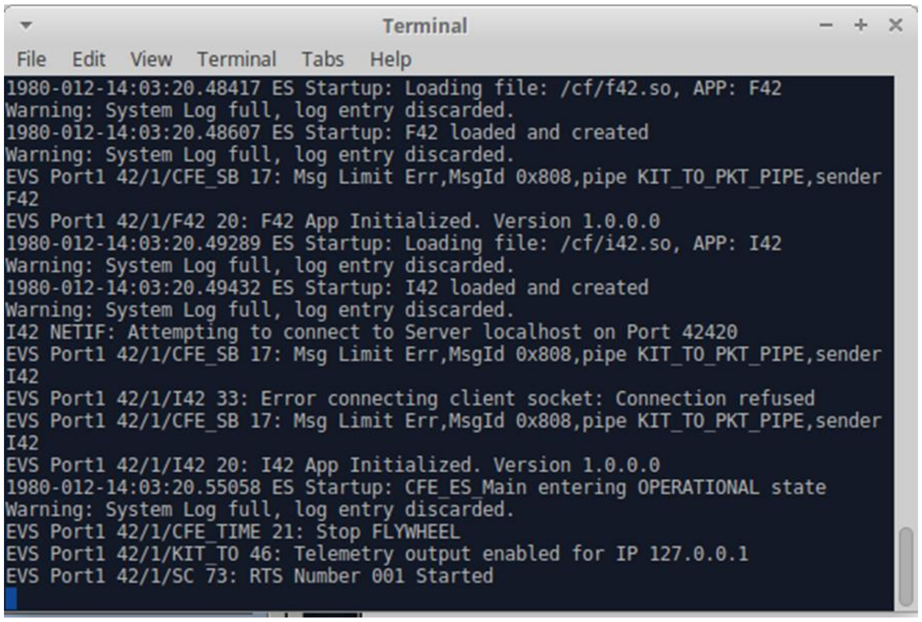


Figure 2-2 –Launcher



```
1980-012-14:03:20.48417 ES Startup: Loading file: /cf/f42.so, APP: F42
Warning: System Log full, log entry discarded.
1980-012-14:03:20.48607 ES Startup: F42 loaded and created
Warning: System Log full, log entry discarded.
EVS Port1 42/1/CFE_SB 17: Msg Limit Err,MsgId 0x808,pipe KIT_TO_PKT_PIPE,sender
F42
EVS Port1 42/1/F42 20: F42 App Initialized. Version 1.0.0.0
1980-012-14:03:20.49289 ES Startup: Loading file: /cf/i42.so, APP: I42
Warning: System Log full, log entry discarded.
1980-012-14:03:20.49432 ES Startup: I42 loaded and created
Warning: System Log full, log entry discarded.
I42 NETIF: Attempting to connect to Server localhost on Port 42420
EVS Port1 42/1/CFE_SB 17: Msg Limit Err,MsgId 0x808,pipe KIT_TO_PKT_PIPE,sender
I42
EVS Port1 42/1/I42 33: Error connecting client socket: Connection refused
EVS Port1 42/1/CFE_SB 17: Msg Limit Err,MsgId 0x808,pipe KIT_TO_PKT_PIPE,sender
I42
EVS Port1 42/1/I42 20: I42 App Initialized. Version 1.0.0.0
1980-012-14:03:20.55058 ES Startup: CFE_ES_Main entering OPERATIONAL state
Warning: System Log full, log entry discarded.
EVS Port1 42/1/CFE_TIME 21: Stop FLYWHEEL
EVS Port1 42/1/KIT_TO 46: Telemetry output enabled for IP 127.0.0.1
EVS Port1 42/1/SC 73: RTS Number 001 Started
```

Figure 2-3 –cFS Terminal Window

2.2 Feature Overview

The main page layout reflects the primary goals of the kit: provide a complete cFS system to simplify the cFS learning curve, simplify the cFS deployment, simplify application development and integration into a cFS system, and assist in porting the cFS to a new platform. The cFS is a complex system so not every cFS feature is covered by the kit. A conscious effort was made to limit the kit's complexity while supporting enough cFS functionality to allow a new user to successfully use the cFS with pre-configured applications. Note all cFS commands and telemetry are accessible from the COSMOS Command and Telemetry Server tool menus.

The main page has two tabs: Home and Demo. The Home tab provides buttons to perform all of the kit's functions. The Demo tab provides pre-configured demonstrations for many of the Home tab's functions. The Home tab is divide into four sections: System, cFS-Functions, Kit-Tools, and Event Messages. The System section is described in Table 2-1. The cFS Functions and kit tools are described in their own subsection. The Event Message window displays the last event message sent by the FSW.

Button/Field	Description
Start cFS	Start the cFS in a terminal window
Time	cFE Executive Service's housekeeping telemetry time seconds value. Should start incrementing after the cFS is started
Enable Telemetry	Telemetry Output's telemetry to COSMOS should be enabled when the cFS is started with the cFS button. If the cFS is running and telemetry has not been enabled this button provides a convenient way to start telemetry.

Reset Time	Resetting time command is a quick and convenient way to show COSMOS and the cFS are communicating properly.
App Summary	Opens a page showing all of the apps in the kit with their housekeeping telemetry (see Section 3) sequence counters and command counters. This provides a quick look capability to show that all fo the apps are running.

Table 1 – Home Page System Buttons and Fields

2.2.1 Manage Files

TBD

2.2.2 Manage Tables

TBD

2.2.3 Manage Memory

TBD

2.2.4 Manage Recorder

TBD

2.2.5 Manage Autonomy

TBD

2.2.6 Manage Applications

TBD

2.2.7 Verify cFS Configuration

TBD

2.2.8 Run cFS Performance Monitor

TBD

2.2.9 Run Benchmarks

TBD

2.2.10 Run 42 Simulator

TBD

2.2.11 Add Application

TBD

2.2.12 Manage Hardware Targets

TBD

2.3 Pre-installed Applications

2.3.1 Kit Applications

TBD - Describe

Benchmark
Heater Control
Heater Simulation
Kit Command Ingest
Kit Scheduler
Kit Telemetry Output
TFTP

2.3.2 cFS Applications

TBD - Describe

Checksum
Data Storage
File Manager
Health & Safety
Limit Checker
Memory Dwell
Memory Manager
Stored Command

3.0 Managing Applications

TBD

3.1 cFS Application Model

TBD
Housekeeping telemetry
Event initialization message

3.2 Removing Applications

TBD

3.3 Creating New Applications

TBD
Yes. I'll use file manager as an example. You can replace an existing app's entries like checksum (CS) since you want be needing it. Let me know fi you have questions. I took a half day today so I'll be leaving after my 11:30am meeting but I can check email later today.

1. kit_sch_schtbl.xml: Scheduler table entry defines frequency of message to be sent. You can use 1Hz. The "msg_id" is the index into the message table.

```
 <!-- FM -->
 <slot id="1" entry="3" enable="true" frequency="5" offset="0" msg_id="13" />
```

2. kit_sch_msgtbl.xml: The scheduler's message table is a little quirky because you have to byte flip the entries. You can keep seq-seq & length the same as below. Just need to set entry id and MID

```
 <!-- FM_SEND_HK_MID 0x188D(6285) => 0x8D18(36120), 0xC000(48152) => 0x00C0(192),
0x0001 => 0x0100(256) -->
 <entry  id="13"  stream-id="36120" seq-seg="192" length="256" />
```

3. kit_to_pkttbl.xml: TO is straight forward. Buffer limit says how many can back up in TO"s queue. 4 is fine.

```
 <!-- FM_HK_TLM_MID 0x088A -->
 <entry  stream-id="2186" priority="0" reliability="0" buf-limit="4"/>
```

Unit testing

3.4 Searching for Existing Applications

TBD

4.0 Transitioning to your Platform

TBD - Address systems engineering activities. More in depth than a checklist but keep details to a what must be done and references source for how to achieve them.

TBD – Use an example mission. SHwo end goal and hightlight lifecycle challenges.

4.1 Porting to a new platform

TBD

4.2 PiSat

4.3 Configuring the cFE

Startip log & messages
Time

4.4 Creating your application suite

Ops concepts

4.5 Development process

TBD

4.6 Ground system

TBD

4.7 Systems Topics

5.0 Starter Kit Design and Maintenance

This section describes the kit's design. It provides enough information for someone to maintain and expand the kit. If this section gets too large it will be split into a separate document. It is included with the user's guide because most users are developers and they may want more information even if they're not maintaining the kit.

5.1 COSMOS Architecture

The COSMOS directory structure is shown in Figure D-1.

5.1.1 Ruby Gems

The application layer is where the bulk of the cFS scalability and extendibility occurs. Users

5.2 Component Releases

This sections outlines the steps needed to update a particular component of the starter kit.

5.2.1 cFE

Tools
Copy docs

Appendix A - Acronyms

API	Application Programming Interface
cFE	Core Flight Executive
C&DH	Command and Data Handling
CCSDS	Consultative Committee for Space Data Systems
cFS	Core Flight Software System
CM	Configuration Management
CMD	Command
COTS	Commercial Off-The-Shelf
CPM	CFS Performance Monitor
CPU	Central Processing Unit
DCR	Discrepancy/Change Request
EDAC	Error Detection and Correction
EDS	Electronic Data Sheet
EEPROM	Electrically-erasable Programmable Read-Only Memory
ES	Executive Services
ETU	Engineering Test Unit
EVS	Event Services
FC	Function Code
FDC	Failure Detection and Correction
FSB	Flight Software Branch
FSW	Flight Software
HW	Hardware
ICD	Interface Control Document
I&T	Integration & Test
MET	Mission Elapsed Time
OS	Operating System
OSAL	Operating System Abstraction Layer
PID	Pipe Identifier
RTOS	Real-Time Operating System
SB	Software Bus Services
STCF	Spacecraft Time Correlation Factor
T&C	Telemetry and Command
TAI	International Atomic Time
TBD	To Be Determined

TBL	Table Services
TIME	Time Services
TLM	Telemetry
URL	Universal Resource Locator
UTC	Coordinated Universal Time
UTF	Unit Test Framework
VDD	Version Description Document
VM	Virtual Machine

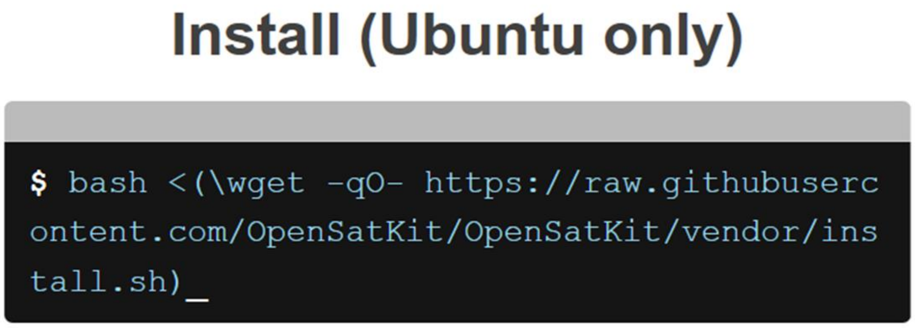
Appendix B – Online Resources

B.1 First Time Kit Installation

The current kit installation script is designed to run on Ubuntu. Due to limited resources we wanted to keep the installation simple, robust, and easy to maintain. A single platform significantly lowers the resources required to verify the installation.

Perform the following steps to install the kit:

1. Create an Ubuntu host platform. A virtual machine is the typical solution. The osboxes.org website provides freely available Ubuntu images for Virtual Box and VMWare.
    a. TBR – Check if auto login allows cFS to be launched without a prompt so telemetry can be enabled.
2. From within the Ubuntu platform go to https://opensatkit.github.io and you'll see the following information.



Install (Ubuntu only)

```
$ bash <(\wget -qO- https://raw.githubusercontent.com/OpenSatKit/OpenSatKit/vendor/install.sh)_
```

3. Open a terminal window on your Ubuntu platform. Copy the bash line from the website into your terminal window and the run the shell script. This script will take from 30-60 minutes to complete depending upon your internet connection and host machine performance. There are some prompts that must be answered.
4. After the script successfully runs COSMOS is automatically launched. Refer to Section 2 for how to use the kit features.

In order to use the PiSat (see TBD for obtaining a PiSat) port forwarding must be enabled in the VM.

B.2 Updating the Kit

TBD

B.3 COSMOS Resources

http://cosmosrb.com

B.4 42 Resources

TBD

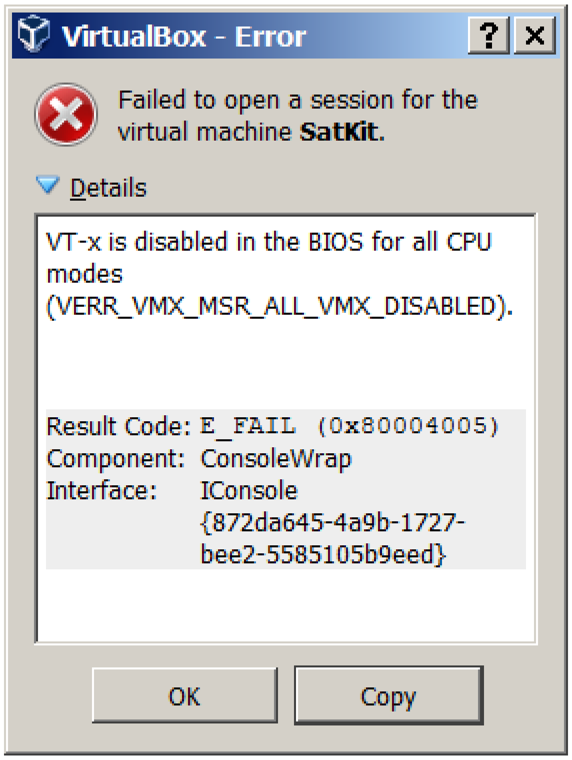
B.5 PiSat

TBD

Appendix C – User FAQs

C.1 Installation Issues

1. When I try to start a virtual machine either after I importing a .ova file or when I try to run any virtual machine, I get an error such as



Failed to open a session for the virtual machine **SatKit**.

VT-x is disabled in the BIOS for all CPU modes (VERR_VMX_MSR_ALL_VMX_DISABLED).

```
Result Code: E_FAIL (0x80004005)
Component:   ConsoleWrap
Interface:   IConsole
             {872da645-4a9b-1727-
             bee2-5585105b9eed}
```

   a. This particular error is due to the virtual technology extensions (VT-x) being disabled in your BIOS. You can enable VT-x by performing the following steps
      i. Reboot your PC. As soon as the first logo appears immediately press the F2 key
      ii. Locate the VT-x settings (menu system specific to your bios) and change it to enabled

2. When I try to update COSMOS by running "bundle update cosmos" from my COSMOS project folder I get a 'permission denied' error in a 'dot' folder such as .rbenv or .gem when a gem is being installed. This typically occurs in a VM. It is best not to install the gem as root so the easiest solution is to change the file permissions for the particular 'dot' directory and all of its

children. The first line below shows the general format and the second line is a specific example.

   a. sudo chown -R username:group ~/.xxx
   b. sudo chown -R vagrant:vagrant ~/.rbenv
   c.

C.2 COSMOS-cFS Connections Issues

1. When I start the cFS I get an error box stating failed to establish connection.
   a. Press the 'Enable Telemetry" button on the main screen.
   b. If this doesn't work then try restarting the COSMOS Command & Telemetry Server. The cFS can remain running when you do this.

2. After I start the cFS it seems to run fine, and I see event messages in the terminal window, but I don't see any telemetry
   a. Look at cFS startup messages. If there are kit_ci and/or kit_to socket bind error 98 then a cFS process is still running
   b. Open a new terminal window (ctrl-alt-t on an Ubuntu VM)
   c. Issue a 'pgrep core' command. If more than one process ID shows up then you have an orphan cFS running.
   d. Issue a sudo kill 'xxxxx' to end the old process where xxxxx is the process ID

C.4 COSMOS

1. After I change an Embedded Ruby script I don't see it take effect.
   a. The results of the ERB processing are cached. You either have to modify the file, or delete the cache from the outputs/tmp folder.

C.3 cFS

1. I modified ../cf/cfe_es_startup.scr and my changes disappeared.
   a. When you build the cFS with cmake and then do a 'cmake install' it copies the cfe_es_startup.scr from the root cmake directory into the target CPU's boot directory. Therefore if you modify the target startup script it will get overwritten when you install a new build.
2. When I start the cFS a terminal window opens and the cFS looks like it starts but nothing updates.
   a. First confirm it's not a connection issue. See FAQ C.1.1.
   b. Look through the startup messages and if you see messages stating tables were not loaded then the problem is most likely that the default tables in the /cf directory were

deleted during the cFS build process. These tables include the scheduler tables and the telemetry output table.

Appendix D – Naming Conventions

D.1 Command & Telemetry Database

FM HK_TLM_PKT
State processor endian at top of packet and don't repeat for appended data parameters
I followed the command mnemonics abbreviations in the HTML. However I made CAPS and used underscores rather than the CamelBack.

I used "write" when writing info to a file and "send" when sending a telemetry packet with info.

Descriptive comments. Start command descriptions with a verb. No period at the end of single sentence comments.

D.2 Abbreviations

ADDR	Address
ADJ	Adjust
APP	Application
ATP	Absolute Time Processor
ATS	Absolute Time Sequence
BIN	Binary
CLR	Clear
CMD	Command
CNT	Count
CONT	Continue
CTR(S)	Counter(s)
DEST	Destination
DIAG	Diagnostic
DIS	Disable
ENA	Enable
ERR	Error
EXE	Executing
EVT	Event
FLTR	Filter
FMT	Format
IDX	Index
MON	Monitor
MSG	Message
NUM	Number
PARAM(S)	Parameter(s)
REG	Register, Registry
RST	Reset
RTS	Relative Time Sequence
SEQ	Sequence

STATS	Statistics
SUB	Subtract
SYS	System
TLM	Telemetry