OpenSatKit Enables Quick Startup for CubeSat Missions

David McComas
NASA Goddard Space Flight Center
8800 Greenbelt Road Greenbelt, MD 20771
301-286-9038
david.c.mccomas@nasa.gov

Ryan Melton
Ball Aerospace & Technologies Corp.
1600 Commerce St., Boulder, CO 80301; 303-939-6771
rmelton@ball.com

ABSTRACT

The software required to develop, integrate, and operate a spacecraft is substantial regardless of whether it's a large or small satellite. Even getting started can be a monumental task. Every satellite mission requires three primary categories of software to function. The first is Flight Software (FSW) which provides the onboard control of the satellites and its payload(s). Second, while developing a satellite on earth, it is necessary to simulate the satellite's orbit, attitude, and actuators, to ensure that the systems that control these aspects will work correctly in the real environment. Finally, the ground has to be able to communicate with the satellite, monitor its performance and health, and display its data.

OpenSatKit provides these three software components in an open source software package. It combines NASA's Core Flight System (cFS)[1,2], NASA's 42[3] spacecraft dynamics simulator, and Ball Aerospace's COSMOS[4] ground system into a system that can be deployed and operational within hours. OpenSatKit is designed to simplify the task of integrating new FSW applications and an example Raspberry Pi target is included so users can gain experience working with a low-cost embedded hardware target. All users can benefit from OpenSatKit but the greatest impact and benefits will be to SmallSat missions with constrained budgets and small software teams.

OPENSATKIT OVERVIEW

The Core Flight System (cFS) is an open flight software (FSW) architecture that provides a portable and extendable platform with a product line deployment model[1,2]. As the cFS is designed for flexibility with many tunable parameters, it can be challenging for new users to configure and deploy. The OpenSatKit 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. All of these components are freely available as open source software.

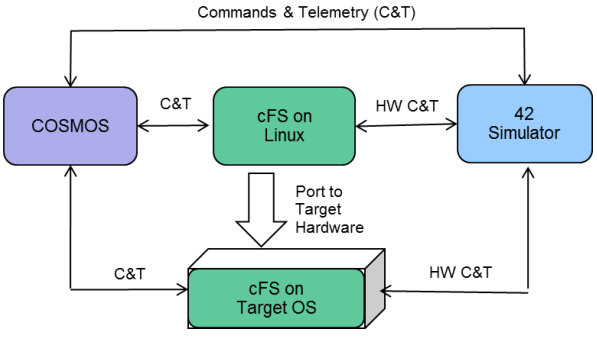

Figure 1 – Starter Kit Block Diagram

Starting with an operational flight-ground system makes the FSW developer's job much easier. Developers can focus on tailoring the kit's cFS components to their needs, adding new mission-specific applications, and porting the cFS to their target platform.

FLIGHT SOFTWARE

The cFS provides a significant portion of a mission's FSW. On recent National Aeronautics and Space Administration (NASA) Goddard Space Flight Center (GSFC) missions using source lines of code (SLOC) as a metric and excluding the operating system, the cFS has provided about a third of the FSW. 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 starter kit can also serve 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.

CORE FLIGHT SYSTEM

Figure 2 shows the cFS architecture. Two prominent features are the Application Program Interface (API)-based layers and the definition of an application as a distinct well-defined architectural component.

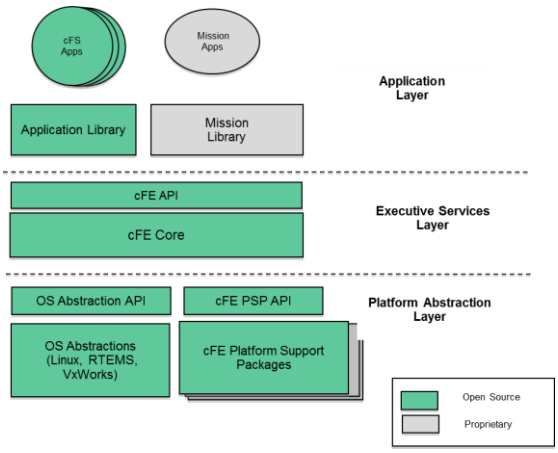

Figure 2 – cFS Layered Architecture

The cFS defines 3 layers with an API between each layer. 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 GSFC FSW projects. The core services include a Software Bus (inter-app messaging), Time Management, Event Messages (Alerts), Table Management (runtime parameters), and Executive Services (startup and runtime resource management).

The Software Bus provides a publish-and-subscribe Consultative Committee for Space Data Systems (CCSDS) [6] 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. New applications can easily be integrated into the build system and even dynamically added/removed during runtime.

As shown in Figure 2 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.

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 3 shows the minimal context for a user app on a single processor system. Three 'kit' apps provide the higher level services.

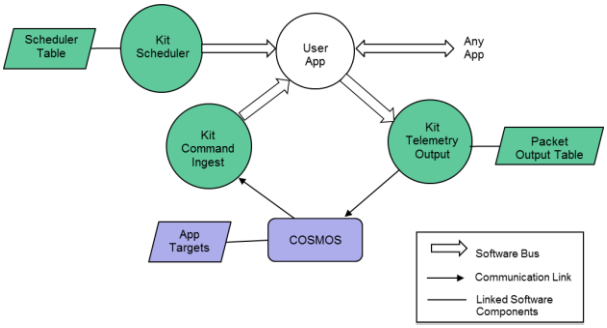

Figure 3 – User Application Context

Apps must have the ability to receive commands from and send telemetry to the ground systems. 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 also generates one or more telemetry packets and sends 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 control mechanism for when to send time-based housekeeping telemetry.

42 SIMULATOR

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 4 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.

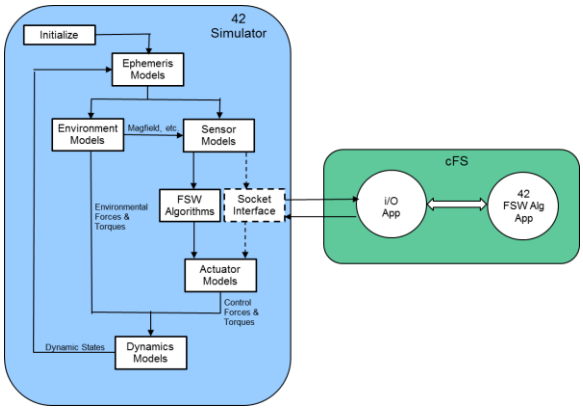

Figure 4 – 42 Simulator

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 4.

COSMOS

Ball Aerospace COSMOS is an open source command and control system that can be used to test and operate any embedded system, from a single board to a complete satellite. COSMOS is made up of a collection of 15 tools that provide functionality such as 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.

The following diagram shows how the COSMOS system is organized. At the heart of the real-time system is the Command and Telemetry Server. All commands and telemetry packets flow through the server to/from the other modularly designed tools that make up COSMOS. This ensures that all interaction with the targets (typically a satellite and a set of GSE hardware), is logged.

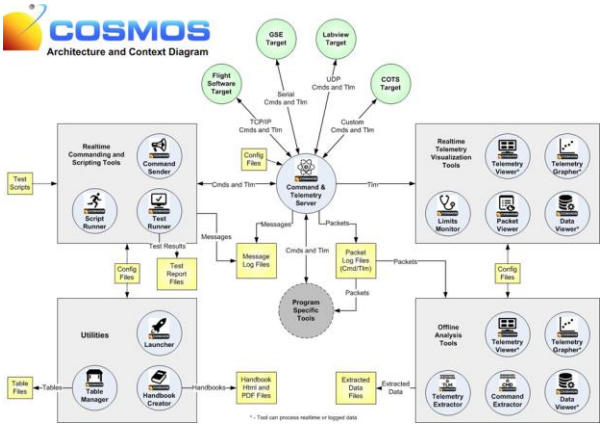

Figure 5 – COSMOS Architecture

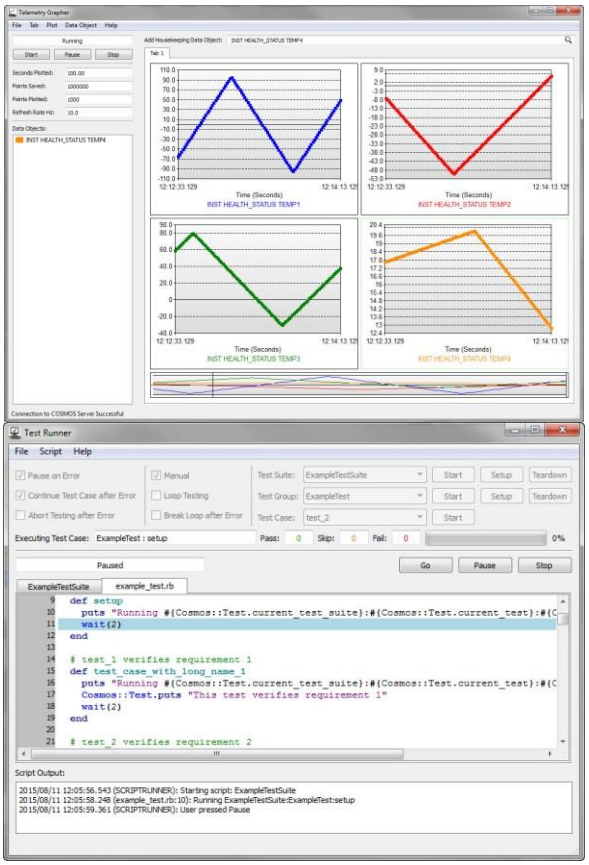

Figure 6 – COSMOS Tools

For OpenSatKit, COSMOS has been preconfigured to communicate with the cFS and the 42 Simulator over TCP/IP. Out of the box, you have a set of COSMOS test procedures ready to execute, telemetry screens to display data from both systems, and the ability to monitor and analyze all the telemetry in the system. The full COSMOS functionality is ready to go and you'll immediately have a working user interface for the system you are designing.

For more information, please see http://cosmosrb.com.

OPENSATKIT FEATURES

The OpenSatKit is distributed with instructions for creating a Linux virtual machine[1]. OpenSatKit is started by launching COSMOS from the cfs-kit/cosmos directory. A customized COSMOS Launcher GUI appears that is the standard COSMOS Launcher with the addition of a cFS Starter Kit button as shown in Figure 7. When you click the cFS Starter Kit icon COSMOS' Command and Telemetry Server and Telemetry Viewer tools are launched since they are required by the kit. The OpenSatKit main page shown in Figure 8 is also opened.

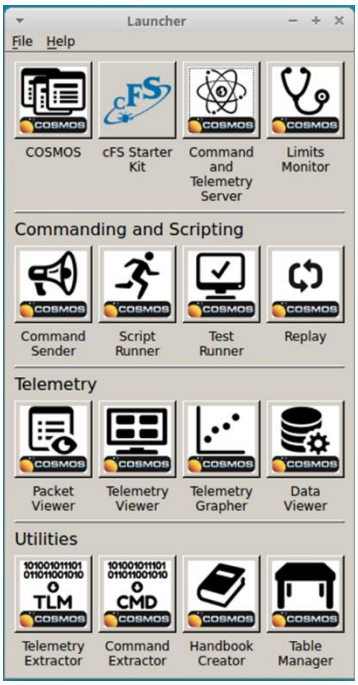

Figure 7 – Custom COSMOS Launcher

Figure 8– Starter Kit Main Page

The main page layout reflects the primary goals of the kit: provide a complete cFS system to simplify cFS' learning curve, simplify application development and integration into a cFS system, and assist in porting the cFS to a new platform. 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 most of the Home tab's functions.

The Home tab is divided into four sections: System, cFS-Functions, Kit-Tools, and Event Messages. The System section allows the user to start the cFS and perform some simple system level operations to ensure that the system is functioning properly. Each button in the cFS-Function section opens a command and telemetry page that allows the user to focus on a particular cFS functional activity that requires one or more apps. For example, the File Management page (see Figure 9) is used to manage onboard directories/files using the File Manager (FM) app and transfer files between COSMOS and the cFS using the Trivial File Transfer Protocol (TFTP) app. The Demo tab contains a demo for each of these functional areas. The cFS-Function pages and corresponding demos help user's conquer the cFS learning curve. In addition the page definitions and underlying Ruby scripts provide examples that users can build upon for their mission-specific applications.

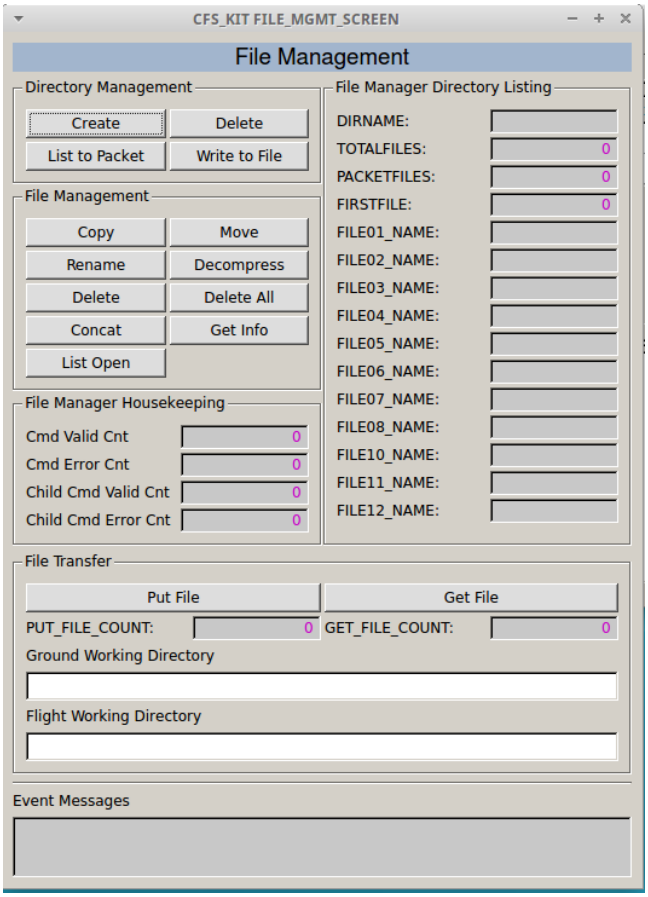

Figure 9– File Management Page

The Kit-Tools section provides tools that assist the user with verifying a platform, evaluation a platform's performance, integrating additional applications to the kit, and porting the cFS to a new target. The current kit includes a Raspberry Pi target.

Summary

OpenSatKit is a freely available open source toolkit that provides users with a complete cFS hosted on a Linux platform. The cFS is a very mature and highly reliable FSW system that has been used on several NASA Class B missions[5]. We hope that this kit will greatly benefit SmallSat missions that often have constrained budgets and small software teams. In additional we'd like to continue to grow the cFS community in an open manner allowing additional assets to be developed and shared among cFS-based missions.

Acknowledgments

The authors would like to acknowledge and thank the cFS community for all the hard work and dedication in maturing the cFS and contributing ideas, applications and tools.

References

1. "Core Flight System" *Retrieved from* http://coreflightsystem.org *June 12th 2017.*
2. "National Aeronautics and Space Administration, Flight Software Systems Branch, cFS Overview" *Retrieved from* http://cfs.gsfc.nasa.gov/Introduction.html. *June 12th 2017.*
3. Stoneking, Eric, "42 Simulator" *Retrieved from https://sourceforge.net/projects/fortytwospacecraftsimulation/ June 12th 2017.*
4. Melton, Ryan, "Ball Aerospace COSMOS" *Retrieved from* http://cosmosrb.com/ *June 12th 2017.*
5. National Aeronautics and Space Administration, Online Directives Information System, Software Engineering Requirements NPR-7150.2B, http://nodis3.gsfc.nasa.gov/displayDir.cfm?t=NPR&c=7150&s=2
6. "The Consultative Committee for Space Data Systems" *Retrieved from* https://public.ccsds.org/default.aspx *June 12th 2017*