SSC21-WKI-06

OpenSatKit – A Flight Software Educational Platform

David McComas
Open STEMware Foundation
15634 Thistle Downs Court Woodbine, MD 21797
443-547-0201
dmccomas63@gmail.com

ABSTRACT

OpenSatKit (OSK) provides a free educational resource originally created for spacecraft flight software (FSW) developers to adopt and use NASA's open-source core Flight System (cFS) that has expanded to include applications for space systems curriculum, STEM educators and hobbyist. In 2015 the NASA Goddard Space Flight Center (GSFC) open sourced the cFS and a spacecraft dynamic simulator called 42. In the same year Ball Aerospace open sourced their COSMOS user interface for command and control of embedded systems. OSK combines these three systems into a ground-flight-simulator software system.

The cFS is an open architecture allowing users to port the core Flight Executive (cFE) to a platform (processor/operating system) of their choice, select/configure cFS community apps and develop custom platform software/apps to complete their system. The cFS has a long successful flight heritage ranging from CubeSats to large NASA GSFC observatories and a bright future that includes NASA's Commercial Lunar Payload Systems program and the international lunar Gateway program. The cFS' open architecture and collection of open-source community apps is part of what makes it so valuable, but they can complicate system integration increasing the learning and adoption curve.

For the past few years, OSK's features, and functionality have expanded to include a cFS-based reference mission called SimSat that includes over 20 apps and a YouTube channel with more than 15 training videos. These resources capture institutional knowledge and lesson learned from years of FSW experience. While these advances have helped many organizations learn and adopt the cFS, it has created some complexity challenges of its own.

To address these challenges OSK is being reorganized to present material based on user tasks and goals. If a user wants to use the cFS for their CubeSat then the top-level tasks include acquiring/implementing a CFS platform port, identifying/integrating existing cFS apps for some of their mission's functionality and developing new platform and app-level software for the remaining functionality. OSK provides task-based activity diagrams with examples, instructional videos, demo scripts, and exercises for each activity.

OSK has expanded beyond its initial primary cFS-based FSW use case and can be applied to STEM education in general. The cFS has been ported to the low-cost Raspberry Pi processor and connected to OSK's COSMOS instantiation. A series of code-as-you-go (CAYG) exercises and videos are in production that teach users how to port the cFS to the Pi and add software for managing sensors and actuators.

This paper describes the new OSK organization and features that provide free FSW educational resources. OSK allows students to develop skills and apply them to meet their educational needs that will transfer into marketable skills as they enter the technical workplace.

INTRODUCTION

OpenSatKit[1] (OSK) was originally developed to provide a complete desktop solution for learning about and developing applications for NASA's open-source flight software (FSW) platform called the core Flight System (cFS). The cFS is a reusable FSW architecture (Figure 1) that provides a portable and extendable platform with a product line deployment model[2]. NASA manages the Platform and Service Layers collectively referred to as the cFS Framework as an open-source project on github[3]. To use the cFS for a project or

mission, a user must locate, learn, and configure existing cFS community apps as well as write and integrate their own mission-specific apps. These can be daunting tasks especially for new users.

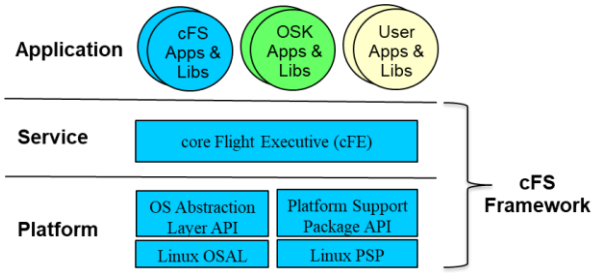

Figure 1: cFS Architecture

The cFS Framework provides many benefits including significant flight heritage, high reliability, a complete set of command and data handling functions required by most spacecraft, and an active community that maintains a suite of apps many with flight heritage and high reliability. OSK was originally developed to help users realize these benefits for their own projects. OSK combines three powerful open-source platforms (Figure 2): Ball Aerospace Corporation's COSMOS command and control platform for embedded systems[4], NASA's cFS[5], and Eric Stoneking's 42 Simulator[6] that can be run on an Ubuntu Linux platform. The initial concept was to provide an operational system that users could work with to learn about the cFS, existing apps, and develop their own apps.

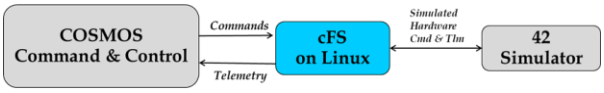

Figure 2: OSK Components

OSK uses customized COSMOS screens and scripts to provide a framework to assist users with workflows and to operate their integrated system. OSK's main screen is shown in Figure 3. Screens are used to launch context sensitive documentation, demos, tutorials, and training videos from OSK's YouTube channel[7].

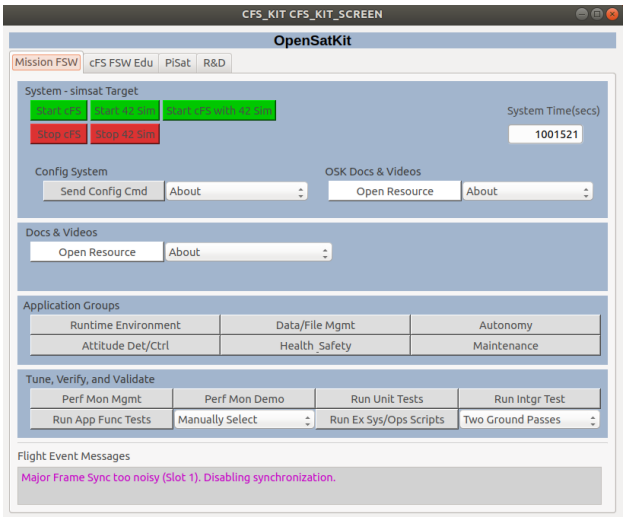

Figure 3: OSK Main Screen

After a couple of years of experiencing how OSK was being used for different purposes in varying environments, it became apparent that the original OSK concept was too vague and created new complexity challenges on top of the cFS. It also became apparent that OSK could play a role in STEM education in general as both a tool for demonstrations and for hands-on learning. In version 3.0 OSK was refactored to align with the following four use cases:

1. Support cFS-based mission FSW development.
2. cFS FSW Education.
3. Remotely control a cFS system on a Raspberry Pi.
4. Use OSK for research and development.

The next four sections describe how OSK addresses these use cases which is followed by a section on STEM educational ideas that go beyond the immediate STEM applicability of each use case.

CFS-BASED MISSION FSW

Figure 4 shows the general flow for a spacecraft FSW development effort. OSK is designed to directly support 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[8] is maintained

by the OSK project and provides links to cFS porting resources.

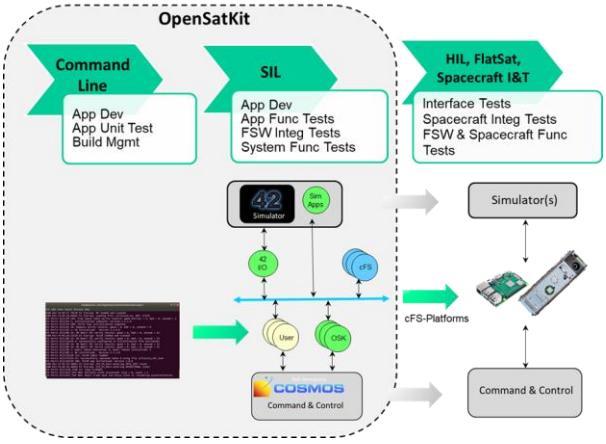

Figure 4: Mission FSW Lifecycle

To understand how OSK can help with mission FSW design the end goal needs to be defined so a design and implementation strategy can be created to take a user from OSK's starting point to their mission's needs. Since spacecraft often have a unique design creating a "one size fits all" approach is not feasible. OSK helps users create their unique mission design by defining a superset of app groups that cover most permutations of spacecraft FSW needs. Figure 5 shows OSK's generic spacecraft app model that a user can tailor to their specific mission where each circle represents an app. Note that cFS community apps shown in blue supply a large percentage of a mission's functionality.

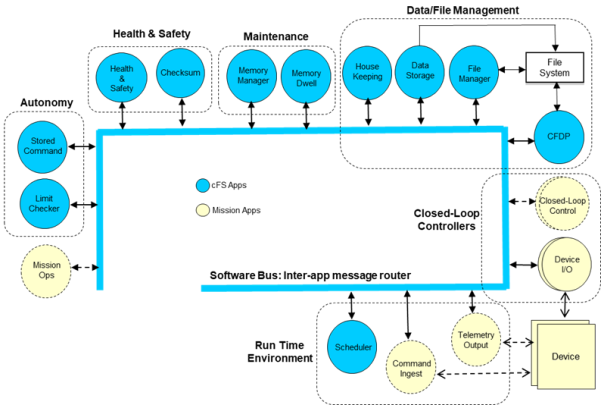

Figure 5: OSK Generic Mission App Model

Mission variability and specialization typically occurs with device interfaces, closed-loop control requirements, and mission concepts of operations. The first FSW mission design step is to create an initial app model from OSK's generic spacecraft model using mission concepts of operations, mission requirements, and the spacecraft hardware architecture often represented in the form of a block diagram. The initial goal is to create a "good enough" app model 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.

With an initial goal in mind OSK's preconfigured reference mission called SimSat shown in Figure 6 can be analyzed. The SimSat app model contains app groups like the generic spacecraft model. These groups help break down a complex FSW system into manageable pieces. In addition, cFS apps often collaborate to provide end-user functionality, unlike smartphone apps that usually operate independently. OSK contains demos and tutorials that address app groups to help users understand how cFS community apps combined with their mission-specific apps can be configured and integrated to create a complete functional system. The OSK documentation helps guide users through the FSW systems engineering process. Note that the analysis process is not always subdivided into categories that directly correlate to the app groups. For example, fault detection isolation and recovery (FDIR) involve both the Health & Safety and Autonomy app groups.

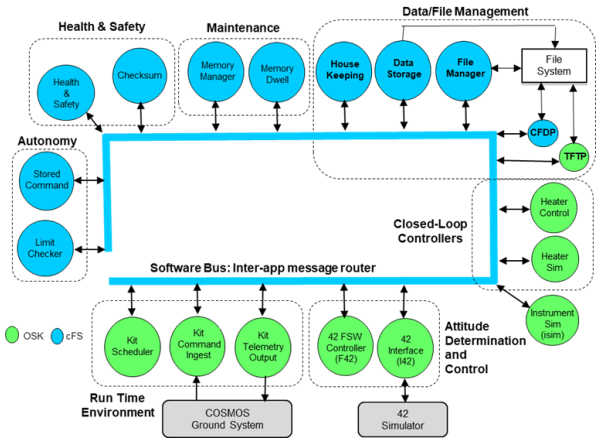

Figure 6: SimSat Reference Mission

The OSK apps shown in green in Figure 6 are not flight qualified. If a mission needs similar functionality, they can start with an OSK app and test it, or they could write their own app from scratch. The OSK apps use a common OSK a framework library that differs from the conventional cFS app design. The main difference is

that OSK apps use JSON files for parameter files and cFS apps use binary files that are managed by a cFE table service. The OSK apps with 'sim' in their name are not intended for flight. These apps provide simulated data on the software bus to support testing apps that rely on data from an external data source such as a payload. A mission will need to tailor these apps for their own simulation needs. This strategy allows ground test and operational scripts to be developed on OSK's SIL platform prior to when external hardware and/or simulators become available.

CFS FSW EDUCATION

The cFS educational material is focused on teaching the core Flight Executive (cFE) services and on developing apps. The platform APIs, Operating System Abstraction Layer (OSAL) and Platform Support Package (PSP) are used, but implementation details behind the APIs are not covered. Platform implementation details can be found at OSK's cFS community platform list[8] that contains links to platform resources maintained by community members.

Figure 7 shows the apps included in the cFS educational target. The number of apps is intentionally low to minimize complexity so the user can focus on the material being taught. The OSK apps and the cFS File Manager (FM) app are needed to provide a complete runtime and file management environment. Memory Manager and Memory Dwell are used in some demos. A user will create prototype apps as they work through exercises.

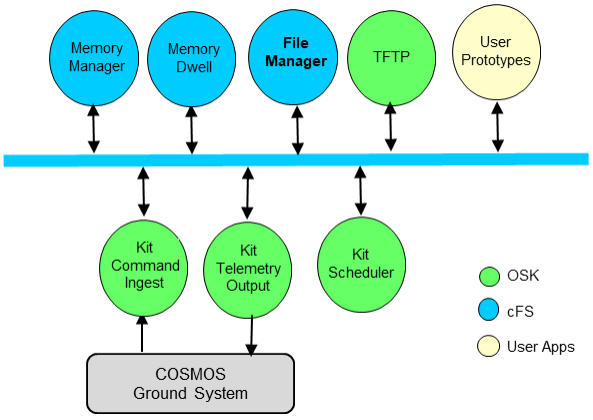

Figure 7: cFS Education Configuration

There are two types of educational resources: cFE services and application development. The cFE services learning resources are accessed via one screen per service as shown in Figure 8's Executive Service screen.

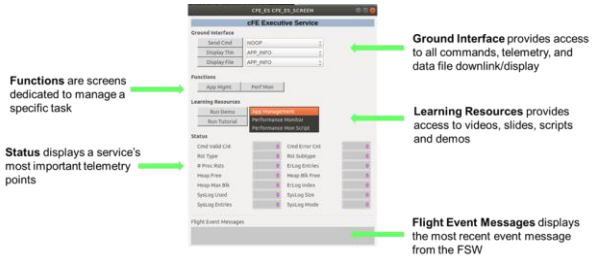

Figure 8: Example cFS Service Screen

The application development resources are more sophisticated. They begin with using a tool to create a "Hello World" app that includes both the FSW source code and COSMOS command and telemetry definitions. Figure 9 shows the screen that steps the user through the workflow for creating a "Hello World" app. The figure on the left side identifies where files are located and whether they are automatically generated or manually edited by the user. When a user completes the six steps, they will be able to send commands to the new app and display the app's telemetry using the COSMOS Command Sender and Telemetry Viewer tools, respectively.

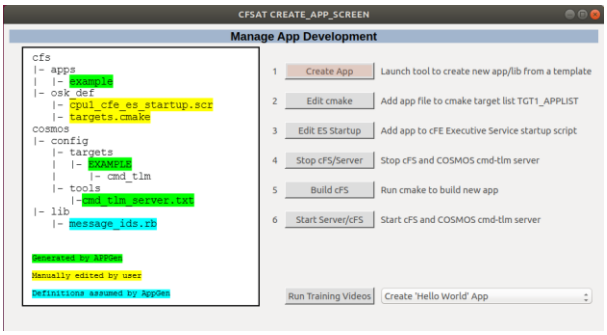

Figure 9: Create App Workflow Screen

The app creation task can be followed by hands-on exercises where the student progressively augments the Hello World app with more features. This workflow uses a combination of COSMOS screens and PDF files to guide the user through the exercises as they manually make code changes using the editor of their choice.

PI-SAT

A separate Pi-Sat Distribution[9] is maintained that can be installed on multiple versions of the Raspberry Pi. The distribution contains instructions for how to configure the Raspbian operating system so the Pi can be controlled by OSK's COSMOS ground system over WiFi. Figure 10 shows OSK's main Pi-Sat screen, a custom Pi-Sat sensor data screen, and two example Pi-Sat configurations. The top configuration is a low-cost CanaKit[10] that uses a Raspberry Pi 3B and the lower configuration with the Pi-Sat display is a custom

configuration developed by Alan Cudmore at the NASA GSFC that uses a Pi Zero.

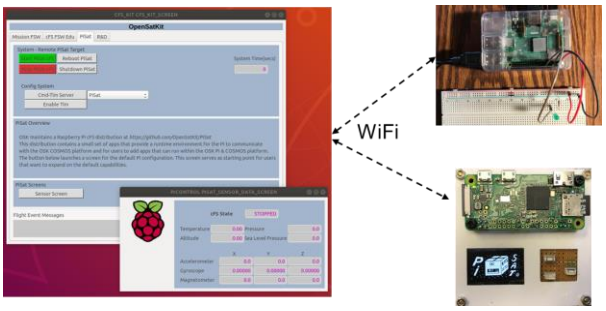

Figure 10: OSK Pi-Sat Configuration

The Pi-Sat cFS app suite shown in Figure 11 contains a minimal set of OSK apps that provide a cFS app runtime environment and file management/transfer services. cFS apps are intentionally not used to avoid the use of binary tables. All OSK apps use JSON tables. This distribution includes a Pi I/O library written in C so users can write new cFS apps in C or C++.

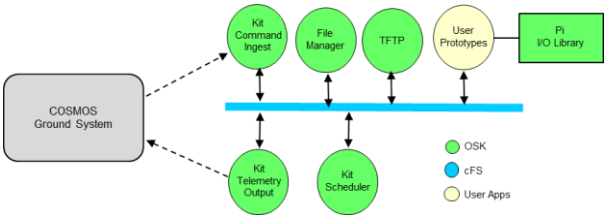

Figure 11: Pi-Sat cFS App Suite

RESEARH & DEVELOPMENT

OSK's build system creates a target named "sandbox" that contains a minimal set of OSK apps that provide a cFS app runtime environment and file management/transfer services. These apps and all OSK/cFS libraries are loaded during startup. The R&D apps are not loaded during startup.

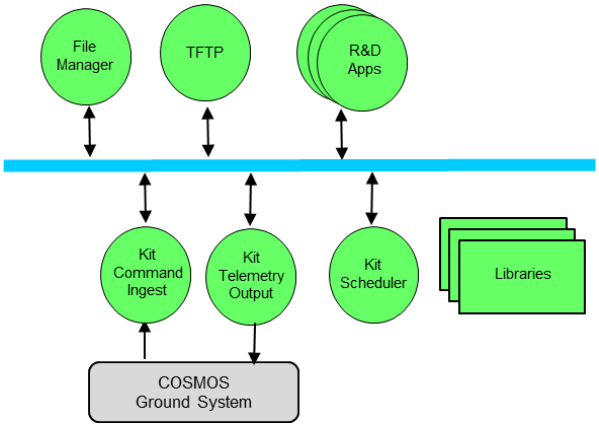

Figure 11: R&D cFS App Suite

The OSK screens shown in Figures 12 and 13 provide interfaces for adding and removing R&D apps during runtime, respectively. OSK takes advantage of the cFS ability to add and remove apps during runtime which is an unusual feature for FSW systems.

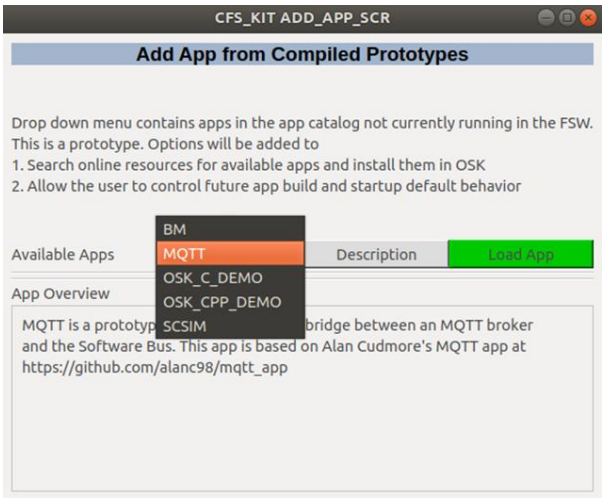

Figure 12: R&D Add App

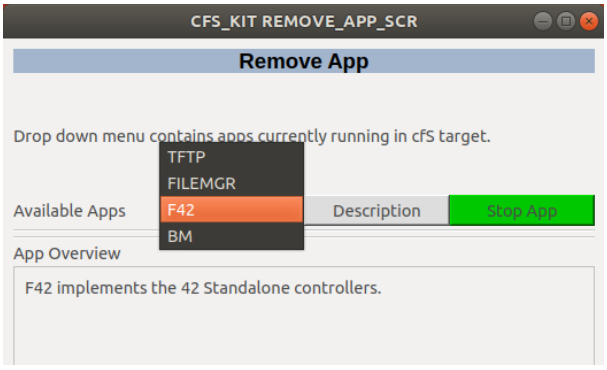

Figure 13: R&D Remove App

STEM EDUCATION

This section highlights the STEM education aspects of OSK's four use cases and identifies additional STEM education opportunities for educators.

The *cFS-based Mission FSW* resources can be used by educational institutions developing CubeSat missions. OSK provides documentation, training videos, and FSW examples that cover FSW system engineering topics that are not well documented in the aerospace industry. In fact, many CubeSat mission development efforts run into difficulty because FSW was not treated with the same engineering rigor as other subsystems. Some FSW system engineering topics include:

- Creating a synchronous system that be verified with repeatable tests.
- Analyzing and implementing a realtime/stored telemetry management strategy.
- Creating highly parameterized applications with ground support tools that simplify FSW tuning during spacecraft I&T and operations.
- Creating a robust and sustainable FSW system that includes FDIR and in-orbit updates.

OSK and the cFS provide hands-on opportunity for students to learn about specifying and developing requirements-based testing. OSK does not currently include this curriculum, but the pieces are in place to allow an educator to create their own curriculum. NASA GSFC's cFS applications contain functional requirements, test scenarios, and the test scripts that are used to verify an app and trace to the functional requirements. These test scripts are written for the Advanced Spacecraft Integration and System Test (ASIST) ground system[11] that is not freely available to students. Students could be assigned the tasks of learning a NASA cFS app's requirements, test scenarios, and test scripts, e.g., File Manager[12]. Then they would use OSK to develop new functional tests that cover the requirements and execute them within OSK's SimSat functional test suite.

OSK's *Pi-Sat* code repo and integration with COSMOS instructions is specifically designed for STEM education. Students learn how to build and run the cFS on a Raspberry Pi and about remote operations by interfacing to the cFS via the COSMOS ground system. The Raspberry Pi has many hardware peripheral options and OSK's instructions describe how to augment the default cFS and COSMOS configurations to include new student cFS applications that interface to a hardware component. Educators can also create their own Pi-Sat configurations like the ones shown in Figure 10 that can be used for demonstrations. The Pi-Sat unit in Figure 10 with the display has been used in classroom demonstrations ranging from grade school through college.

OSK's *cFS FSW Education* and *R&D* targets are designed to teach cFS FSW engineering concepts, software engineering concepts in general, and provide a platform for exploratory prototyping. For example, the R&D app suite includes a protype Message Queuing Telemetry Transport (MQTT) app that can be used with an MQTT broker such as HiveMQ[13] to demonstrate publishing messages from an MQTT broker onto the cFS software bus using the MQTT app. MQTT is a message protocol that is suitable for restricted networks with low bandwidth and high latency and is a popular choice for Internet of Thing (IoT) devices.

A final STEM consideration is a curriculum for porting the cFS. As previously stated OSK does not directly support this activity and a git repo with porting resources is maintained by the OSK project[8]. The recommendation is for educators interested in teaching embedded software systems to develop educational material for porting the cFS to an embedded processor using a realtime operating system. Once the cFS is ported, a workflow like OSK's Pi-Sat workflow could be used that allows students to interface to the embedded system using COSMOS and OSK's command and telemetry apps. Students could then add customized interfaces and apps to learn what goes into creating a fully functional embedded system that is remotely controlled.

FUTURE WORK

OSK continually evolves based on user needs. Completing the app group and individual app demos and training videos is an ongoing effort. In addition, the "code-as-you-go" hands on exercises will be expanded to cover both the cFS style and OSK style apps and include more complex app features. If there's demand for some of the OSK apps to be fully verified for use in flight, then unit and functional tests will be developed. A future STEM educational consideration related to OSK enhancements is the possibility of collaborating with educators to create curriculums that would include student assignments that produce OSK contributions. This would provide the students with opportunities to experience working on a collaborative open-source project.

At the time of this writing, unit tests in general have not been addressed by OSK. Users can build and run the NASA cFS app unit tests at the command line, but an integrated unit test framework has not been configured.

After NASA officially releases the cFS Bundle release named Caelum (aka 7.0) and the NASA GSFC apps are upgraded to run with the Caelum API changes, OSK will be updated with Caelum, with the updated apps, and unit testing will be added as part of OSK's SimSat verification.

After the Caelum release another potential evolutionary step would be the maturation and integration of technologies that would facilitate the packaging, distribution, and integration of cFS community apps. The international Consultative Committee for Space Data Systems (CCSDS) organization maintains a standard called Electronic Data Sheets (EDS)[14] which are a formal specification of a device, system, or software interface in a machine-readable format. The current cFE release contains EDS specifications for the cFE apps and Joe Hickey maintains a cFS distribution with an EDS toolchain that generates FSW artifacts from EDS specifications[15]. These artifacts include FSW header files and libraries that can be used to provide symbolic command and telemetry bindings for scripting languages such as Python and Lua. OSK could play a role in standardizing how cFS uses EDS by providing a platform for prototyping and demonstrating EDS-based FSW development and test workflows to the cFS community.

A longer-term potential OSK architectural design change would include containerized components that can be cloud-based. COSMOS 5.0, in Beta release is making a significant architectural departure from 4.x which is used by OSK. COSMOS 5 is "s highly scalable, cloud native, command and control software system … [that] runs across a set of containers managed by Docker."[4]. If OSK defines the flight-ground and flight-simulator interfaces, then in theory solutions other than COSMOS and 42 could be provided so OSK itself would become a framework that could have multiple instantiations.

SUMMARY

The cFS is a mature and reliable open source FSW platform maintained by NASA. The cFS has been used on several NASA Class B missions and has recently gained traction within the CubeSat community. OSK is an open-source resource that evolved from a platform for learning about and developing applications for the cFS to a multi-faceted platform that supports developing cFS-based mission FSW, learning the cFS, remotely controlling a cFS system on a Raspberry Pi, and developing prototype R&D apps. This evolution makes OSK a resource for both the aerospace community and STEM educators.

Acknowledgments

The author would like to acknowledge and thank the COSMOS, cFS, and 42 Simulator teams for their dedication to the maintenance of and continual enhancements to their respective open-source projects.

References

1. OpenSatKit project, https://github.com/OpenSatKit/OpenSatKit/wiki
2. NASA Goddard Space Flight Center, Flight Software Systems Branch, core Flight System Overview, https://cfs.gsfc.nasa.gov/Introduction.html.
3. NASA core Flight Executive open-source repository, https://github.com/nasa/cFE.
4. Ball Aerospace COSMOS project, http://cosmosrb.com/
5. NASA core Flight System open-source repository, https://github.com/nasa/cFS.
6. Eric Stoneking, 42 Simulator, https://github.com/ericstoneking/42
7. OpenSatKit YouTube Training Video Channel, https://www.youtube.com/channel/UC2wfvAIkrrgyC4ITwL3zokg
8. OpenSatKit cFS Community Platform List, https://github.com/OpenSatKit/cfs-platform-list
9. OpenSatKit Raspberry Pi cFS Distribution Repo, https://github.com/OpenSatKit/pi-sat
10. CanaKit Electronic Kits, Module & Parts, https://www.canakit.com/
11. NASA Goddard Space Flight Center, Ground Software Systems Branch, Advanced Spacecraft Integration and System Test Overview, https://nasa-asist.gsfc.nasa.gov/index.html.
12. NASA GSFC File Manager Test Artifacts, https://github.com/nasa/FM/tree/master/test_and_ground
13. HiveMQ Enterprise MQTT Broker, http://www.hivemq.com/demos/websocket-client/
14. CCSDS XML Specifications for Electronic Data Sheets for Onboard Devices and Software Components, 2015, http://cwe.ccsds.org/fm/Lists/Projects/DispForm.aspx?ID=269
15. Joe Hickey cFS Distribution with EDS. https://github.com/jphickey/cfe-eds-framework.