SSC23-P3-29

cFS Basecamp: A Flight Software STEM Education Ecosystem

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

ABSTRACT

The open-source core Flight System (cFS) Basecamp ecosystem includes several cFS-based STEM educational projects and provides the infrastructure for users to create their own. Basecamp's tool suite and app repositories function much like a smartphone's App Store model. The initial cFS Basecamp installation includes several built-in tutorials that help users learn NASA's cFS application environment and shorten their path to productivity. Online resources describe Basecamp's goal-oriented software/hardware projects. These projects are designed so students understand how to create app-based solutions to meet a particular goal. This approach evolved after years of being engaged with teaching the cFS and learning which teaching methods were most effective.

Users begin by installing a lightweight Python GUI with minimal external dependencies. This approach helps avoid platform-specific issues so Basecamp can be used in classroom settings where students have diverse computing platforms. With Basecamp's GUI installed, students are ready to work on projects. A preinstalled demonstration app in conjunction with a self-guided tutorial helps users understand an app's command/telemetry interface and the cFS application runtime environment. A built-in app generation tool creates a "Hello World" app to help students take a first step into cFS app development. From there, they can work through Code-As-You-Go (CAYG) lessons that introduce topics. Each new topic is reinforced with hands-on exercises. These lessons are more suitable for instructor-led classes that can be held virtually or in person.

The next level of projects requires Basecamp's github app repositories. Using the GUI, students can select and install Basecamp cFS apps from github with only a few mouse clicks. For example, the General-Purpose Input/Output (GPIO) Demo project requires a cFS Raspberry Pi interface library and an app to control an LED connected to a Raspberry Pi through the GPIO connector. To implement this project, students first connect an LED to a Raspberry Pi, install Basecamp on the Pi, download/install the library/app, and rebuild/run the cFS. A second Basecamp instance installed on a separate computer can remotely control the Raspberry Pi. This is achieved by using Basecamp's MQTT Gateway app. This app utilizes the Internet of Things (IoT) MQTT messaging service that has freely available broker servers.

Basecamp's modular approach with plug 'n play cFS apps make it an ideal platform for creating STEM educational projects. These projects will help students learn valuable hardware/software skills while using NASA's award-winning flight software that has a large user base in the aerospace community.

BACKGROUND AND MOTIVATION

NASA licensed its core Flight System[1,2] (cFS) as open source in 2015 and, since then, it has played a significant role in the rapidly expanding aerospace industry. The cFS' international user base includes academic, government, and commercial organizations. During the same period of community growth, there have been organizations that evaluated the cFS and decided not to use it. Here are a few of the current cFS adoption challenges:

1. Where should someone start learning how to use the cFS?
2. How do they use the cFS to solve their project's needs?
3. What cFS processor/operating system ports are available and if needed, how do I create a new port?

4. What existing libraries/apps are available and how should they be configured/combined with project-specific libraries/apps to provide a fully functioning system?

5. Are there cFS open-source community benefits?

OpenSatKit[3] was released in 2017 to help address challenges 1, 2, 4, and 5. OpenSatKit includes a reference mission cFS target, the Ball Aerospace COSMOS ground system[4], and NASA's 42 dynamic simulator[5]. While OpenSatKit helped with the situation, it also suffers from adoption and maintenance challenges. Users install OpenSatKit as a monolithic tool so the installation can be over 20 minutes. In addition, the current version of COSMOS used in OpenSatKit, uses an old version of Ruby with an outdated graphics library that doesn't always install correctly on the latest Linux platforms so successful installations are inconsistent. OpenSatKit's dependencies on 42, COSMOS, the cFS Framework, and several NASA cFS apps makes it hard to maintain as each of these packages continue to evolve. OpenSatKit's current cFS Framework is a major release version behind NASA's most recent cFS Framework due to the timing of NASA apps that are compatible with the cFS Framework. In addition, COSMOS underwent a major architectural change and the management of COSMOS transitioned from Ball Aerospace to OpenC3[6]. At the time of this writing, OpenSatKit has not been refactored to the new COSMOS nor updated to the latest cFS Framework and app versions.

The design concepts and motivation for creating cFS Basecamp[7] were driven by OpenSatKit user feedback and by in-person experiences of using OpenSatKit for educational and aerospace project consultation purposes. Basecamp's architecture is the inverse of OpenSatKit's monolithic architecture. Basecamp includes a minimalistic Python command and control system that communicates with a cFS target that includes an app suite that provides an operational runtime environment. The default cFS target has more than enough functionality to be used for cFS training. Basecamp's fast and easy installation combined with a short learning curve means users can quickly focus on learning the cFS.

OpenSatKit provides a reference mission that can be very helpful to new cFS developers because it shows how an app suite can solve a mission's functional requirements. However, experience has shown that breaking down the problem space into smaller parts and incrementally showing how to solve individual parts of the problem is what many missions need. Common questions include, "How do I create an app to manage a payload?" "How do I create an attitude control app?" Separate Basecamp projects can be created to address each question.

In 2023, Basecamp projects were successfully used during a hands-on workshop at the 2023 FSW Workshop[8] and for a University of Maryland Space Systems lecture and homework assignment. The Basecamp online resources for apps and projects will continue to mature so users can use them independent of an instructor. These resources will provide guidance for how the cFS community can contribute apps and projects. In addition, apps and projects are not limited to the aerospace community: they can be used for general STEM education as well.

CFS ARCHITECTURE AND TERMINOLOGY

This section briefly introduces the cFS application architecture to provide context for Basecamp. Figure 1 shows the cFS layered architecture. The Platform Abstraction layer includes the Operating System Abstraction Layer (OSAL) and the Platform Support Package (PSP). This layer allows the Core Flight Executive (cFE) to be ported across different processor/operating system (OS) platforms. The cFE layer provides a runtime environment for applications through five services: Executive, Time, Events, Table, and the publish/subscribe messaging system called the Software Bus. Libraries run in the context of the app that is using them. The bottom and top layers are partially shaded blue because the cFS provides some functionality that is augmented by users with project-specific functionality. The term *cFS target* refers to all the software components that are built and deployed on a single processor/OS platform.

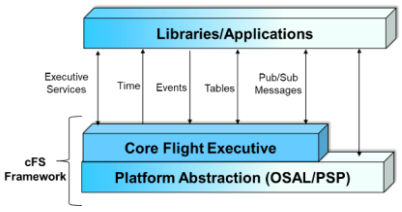

Figure 1: cFS Layered Architecture

The cFS Framework (bottom two layers) provides services to apps via function calls defined in each layer's Application Programming Interface (API). (Libraries and applications can make calls outside of the cFS Framework but, if they do, they won't be as portable.)

The cFS Framework is built as a single binary image and libraries and applications are built as individual object files. A startup script defines the order in which libs/apps are loaded. The script also defines parameters such as an app's priority and local memory requirements. App resources are acquired and released through cFS Framework APIs. Each app has its own execution thread and can spawn child tasks. Apps can be restarted and reloaded during runtime without the need to reboot.

Figure 2 shows a traditional cFS "lollipop" diagram that serves as a cFS target app context diagram. Typically, target apps are shown along with significant interfaces. It's also common practice to color code the apps to identify app heritage. The names of the apps help identify the decomposition of functional responsibilities and connecting the apps via the Software Bus emphasizes that all apps can communicate via messages.

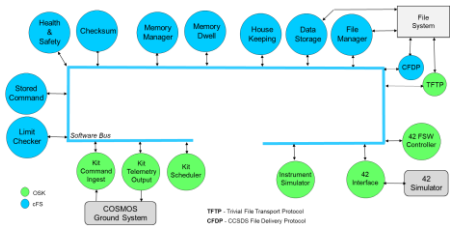

Figure 2: cFS Target App Context

CFS BASECAMP ECOSYSTEM

Figure 3 shows the components of the Basecamp ecosystem. The Basecamp box represents the Python Graphical User Interface (GUI), the tools used to build and operate cFS targets, and the tool to export the command and telemetry definitions. The cloud represents internet-based resources that include an App Exchange hosted by GitHub, instructional YouTube videos, and project webpages hosted on OpenMissionStack.com (OMS) [9].

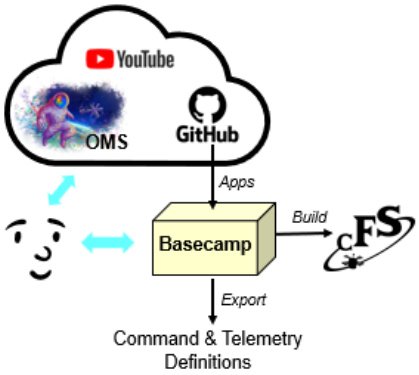

Figure 3: cFS Basecamp Components

Users engage with these resources according to their needs. Basecamp includes built-in tutorials and online videos to help users gain proficiency with Basecamp itself and to learn the cFS. The structure of Basecamp's learning material is also much different than OpenSatKit's. A majority of Basecamp's instructional material is goal-oriented lessons. For example, instead of describing a cFS Framework Application Programming Interface (API) such as the cFS Software Bus messaging system, Basecamp walks users through the process of sending a command from the ground to a particular app. This use case answers the question, "How do I send a command to an app?" and in the process of answering this question the relevance of the API is revealed to the user. Goal-oriented learning reveals the purpose of the API rather than simply describing an API's capabilities and functionality. The videos also encourage viewers to try things for themselves using Basecamp to create an immersive learning experience. There's still a need for material that can be used as a reference, but that is typically more helpful once someone has achieved some basic cFS skills.

The next level of cFS education addresses project-oriented problems such as, "How do I create a cFS app to manage a spacecraft payload?" Basecamp's App Repo and online project webpages address these types of questions. Basecamp includes libraries and apps that can be installed on Basecamp's cFS target with a couple of mouse clicks. Users can assemble new projects or go to Basecamp's project webpages to download preassembled projects that provide solutions to these types of problems.

The App Repo and project assembly processes are feasible because Basecamp uses a NASA cFS toolchain called *cfe-eds-framework*[10] that was released as open source in 2021. This toolchain was developed under a technological effort and allows app command/telemetry definitions to be defined once using CCSDS Electronic Data Sheets (EDS) [11] and the toolchain produces flight and ground code from the single definition. Basecamp leverages the toolchain so apps can automatically be added and removed from the cFS target. Efforts are underway at NASA to add an EDS toolchain to the cFS. EDS application command/telemetry definitions are already included with the current cFS Framework.

The use of EDS to define application interfaces and the inclusion of a runtime app suite as part of the standard Basecamp distribution allow the flight and ground software to be architected as a single system. Figure 4 illustrates how the EDS app specs provide single source definitions used by the build process to generate and propagate artifacts for both the ground and flight systems. When an app is added to Basecamp, the app's command and telemetry screens are generated using the EDS-defined Python bindings, so the user does not have to manually input any definitions nor modify python code.

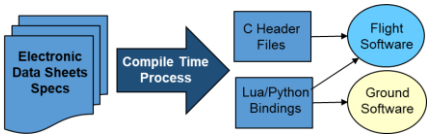

Figure 4: EDS Toolchain Artifacts

PYTHON GUI

Basecamp's Python GUI assists users with using the App Exchange, managing a cFS target, running a cFS target, and accessing built-in educational resources. The GUI uses PySimpleGUI[12] which helps minimize dependencies and simplify installation. The main screen is shown in Figure 5. Section A provides mechanisms for user input, and status on the current target. Parts of the section will be described in more detail later. Section B displays print statements and events messages from a running cFS target. Section C displays ground events. Figure 6 shows parts of Section A that allow a user to build, start, and stop a cFS target. Information about the cFS target is displayed as well as the communication status between the cFS target and the GUI.

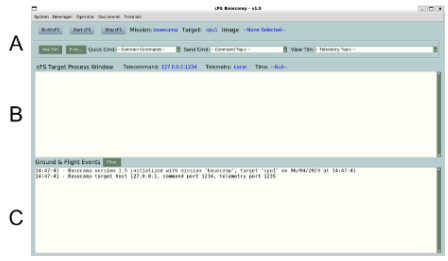

Figure 5: Basecamp Main Screen

Figure 6: cFS Target Management Fields

The *Send Cmd* and *View Tlm* drop down menus located in Section A are shown in Figures 7 and 8 respectively. They provide access to all telecommand, and telemetry messages defined for a cFS target. The EDS definitions use the term *Topic* to identify a message independent of a cFS target. A separate Software Bus message identifier is used within a cFS target deployed on a processor/OS platform.

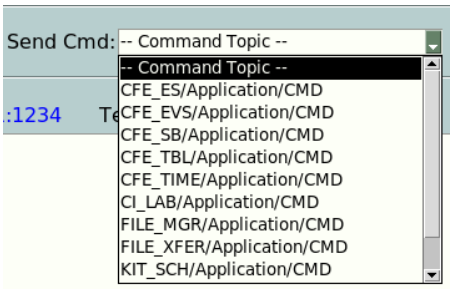

Figure 7: Send Command Menu

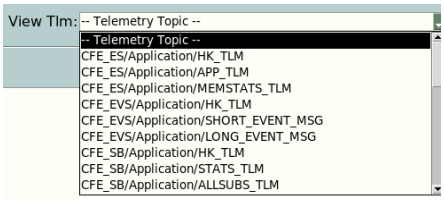

Figure 8: View Telemetry Menu

Figure 9 shows the File Manager app's *"Send File Information"* command screen as an example command that requires user input. Figure 9 shows File Manager's "File Information" telemetry that is sent in response to the command.

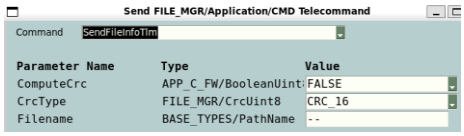

Figure 9: Example Command Input Screen

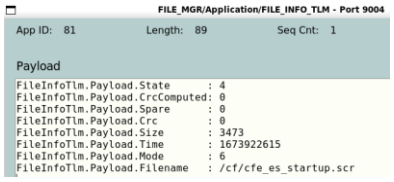

Figure 10: Example Telemetry Screen

The static contents of the menus and screens shown in Figures 7 through 10 are retrieved from the Python libraries generated by the EDS toolchain. Therefore, when a new cFS target is built, no new Python coding is required. The only required action is to restart the Basecamp GUI so it uses the new cFS Target Python libraries.

OPS SERVICE APPS

The default Basecamp cFS target comes preconfigured with several libraries and applications that provide a fully functional operational runtime environment. These apps are maintained by the Basecamp development team so they will always be compatible with the latest NASA cFS Framework that is released in the cfe-eds-framework project. This avoids OpenSatKit's issue of trying to maintain a default cFS target app suite containing non-OSK apps that is compatible with the latest NASA cFS Framework release.

The Ops Service app suite includes:

Command Ingest: Receives telecommands from external interfaces and publishes command messages on the Software Bus.

Telemetry Output: Receives telemetry messages from the Software Bus and sends them to an external interface.

File Manager: Provides a ground command/telemetry interface for managing onboard directories and files.

File Transfer: Transfers files between flight and ground using a file transfer protocol implemented in both the flight and ground systems.

Scheduler: Periodically send messages using a time-based message scheduler table.

Having a standard suite of ops service apps combined with EDS allows the flight and ground software to be architected as a single system. The File Browser tool shown in Figure 11 highlights the integrated ground-flight perspective of Basecamp. The left pane lists the files in the current ground system working directory. Most operations can be performed using the default ground file server directory. The right pane shows the current flight software working directory that defaults to the directory where the cFS apps reside that are loaded during cFS initialization.

Common file management operations can be performed by right clicking within the ground or flight panes. To send a file from the flight to ground, the user simply highlights the file in the flight windowpane, right clicks, and selects *Send to Ground* from the menu. The file transfer process is automatically invoked, and the ground directory listing is updated after the file transfer is complete.

Figure 11: File Browser

APP REPOS

The Developer menu shown in Figure 12 lets users download apps from GitHub repositories and add/remove apps from the cFS target. The Download app screen is shown in Figure 13. The contents of this screen are generated from the GitHub repositories. The *Add App* menu item launches a screen that guides users through the steps for adding apps to the cFS target and rebuilding it. The user does not have to edit any files.

Commented [BT1]: This needs a figure number

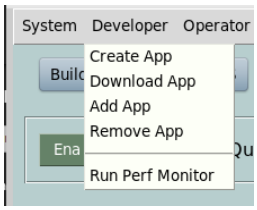

Figure 12: App Repo Menu

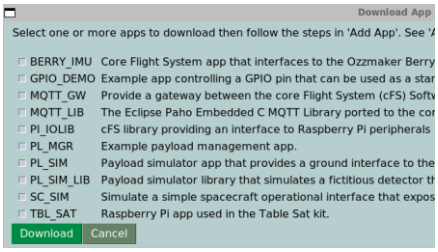

Figure 13: Download App Screen

Automated app integration is made possible because command and telemetry packets are defined using EDS, and each app has a JSON specification file and a JSON runtime initialization file. All Basecamp apps use Basecamp's application framework library called APP_C_FW that provides utilities to process the JSON initialization files. APP_C_FW is important because it allows apps to be developed with a common design that can be utilized by Basecamp tools. It's somewhat analogous to smartphone app frameworks but on a much smaller scale.

Figure 14 shows an excerpt from File Manager's EDS XML file for the *"Send File Information"* command shown in Figure 9. Figure 15 shows the *"cfs"* object portion of File Manager's JSON spec file. This defines default configurations for integrating the app into the build system and for loading it during the cFS initialization.

```
<ContainerDataType name="SendFileInfoTlm_Payload">
  <EntryList>
    <Entry name="Filename"    type="BASE_TYPES/PathName"
    <Entry name="ComputeCrc" type="APP_C_FW/BooleanUint8"
    <Entry name="CrcType"     type="CrcUint8"
  </EntryList>
</ContainerDataType>
```

Figure 14: File Manager EDS Excerpt

```
"cfs": {
    "cfe-type": "CFE_APP",
    "obj-file": "file_mgr",
    "entry-symbol": "FILE_MGR_AppMain",
    "name": "FILE_MGR",
    "priority": 80,
    "stack": 16384,
    "load_addr": 0,
    "exception-action": 0,
    "app-framework": "osk",
    "tables": ["file_mgr_ini.json"]
},
```

Figure 15: Download App Screen

Basecamp's App Repo approach has several benefits. For STEM education settings, it allows students to focus on the problem being solved by the apps and not be concerned with the underlying plumbing. For new flight software developers, they can start using and learning the cFS and eventually learn the underlying build and deploy system when they have a better understanding of the cFS' architecture. The *Add App* screen has manual options that help teach each of the app integration steps. The App Repo also helps educational projects that use one or more apps to easily be created and maintained. Lastly, Basecamp's standardized app spec allows the user community to contribute apps and projects which could have a substantial impact on the breadth and depth of STEM educational material.

REMOTE OPS

Initially, Basecamp is installed and run in a Linux environment so the Basecamp GUI and cFS target are running on the same Linux platform. This environment is suitable for learning the cFS and for many software-only projects. If you need to control a remote cFS target, you need to configure four items.

First, the GUI that will be controlling the remote target must use a Python library that was generated from a cFS build that has the same app suite as the remote target. This ensures the same EDS definitions are used by the local GUI and the remote cFS target.

Second, the ability to start/stop the remote target is required. Directly starting a remote target is outside the scope of Basecamp but is an option in many situations. Basecamp provides the capability for the GUI to communicate with a second Basecamp Python tool called Remote Ops. Figure 16 shows Basecamps local remote ops screen. This screen provides commands for starting/stopping a cFS target. For this to be helpful, the Remote Ops python process must be started on the remote target as part of the boot sequence.

Figure 16: Remote Ops Target Control

The remote ops python process communication uses an MQ Telemetry Transport (MQTT)[13] broker. MQTT is the de-facto messaging protocol standard for the Internet of Things (IoT) and MQTT messaging brokers are freely available.

Third, there needs to be a communications path from the local GUI to the remote cFS target for telecommands. If a direct network connection can't be made, then Basecamp's MQTT gateway app MQTT_GW[14] can be used. The local GUI can be configured to send commands to an MQTT broker. The MQTT_GW app running on the remote target converts MQTT messages to cFS command messages and publishes them on the Software Bus.

Fourth, there needs to be a communications path from the remote cFS target to the local GUI for telemetry. If a direct network connection can't be made, then the MQTT_GW app is required on both the local and remote cFS targets. They provide a telemetry path from the remote cFS target through an MQTT broker to the local cFS target where the MQTT_GW app publishes the telemetry messages on the Software Bus. The local Telemetry Output app sends the messages to the Python GUI.

TOPIC-BASED LEARNING

Topic-based learning means Basecamp's ecosystem includes learning resources that are targeted to teaching a single topic at a time. Whenever possible the topics are goal-oriented so as a user works towards a goal, they learn how it is achieved. Basecamp's Python GUI and cFS target contain material so a user can immediately start engaging with the cFS. The default cFS target includes a demo app called APP_C_DEMO that contains enough functionality so users can step through a series of learning exercises to learn how to control an app from the GUI. Basecamp comes with built-in tutorials as shown in Figure 17. The *"Basecamp Feature Overview"* tutorial relies heavily on APP_C_DEMO.

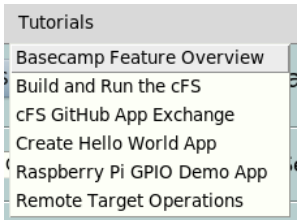

Figure 17: Built-in Tutorials

The built-in tutorials are intended to get a user started. The OpenMissionStack website[9] and OpenSatKit's YouTube channel[15] contain articles and videos that are updated over time. Lessons, whether an article or video, are focused and short in duration so users can incrementally learn as they have time.

PROJECT-BASED LEARNING

Most missions have a payload that needs to be managed so developers ask, "How do I use the cFS to manage a payload?". Basecamp projects use customized cFS targets that are designed to help teach how to solve complex problems that are broader in scope than a single topic. For instance, the Payload Sim project provides a software-only project that teaches a method for how to design and write an app to interface to a payload and write the payload's data to files. Since it is software-only, users can start learning the cFS app strategy before their mission-specific hardware is available. In addition, the Payload Sim project teaches another development strategy where non-flight libraries and apps can be used to simulate an external component before it is available to the flight software team.

Payload Sim only answers part of the initial problem, because embedded in the first question is, "How do I use the cFS to interface to an external device?" Basecamp has the Raspberry Pi General-Purpose

Input/Output (GPIO) Demo project to help answer that question. This project uses a low-cost Raspberry Pi to host the cFS target. The target includes a hardware interface library and an app to control an LED that is connected to the Raspberry Pi's GPIO header.

Each Basecamp project is documented on the OpenMissionStack website with links to videos hosted on OpenSatKit's YouTube channel and, if needed, a link to a GitHub project repo[16]. The GitHub project repo is optional because some projects are only intended to be assembled by the user accessing Basecamp's App Repo so a repository with the completed project is unnecessary and would only create a maintenance burden.

Basecamp's apps and projects are meant to serve as a starting point for the cFS community. The real utility of Basecamp will be realized if the user community contributes apps and projects. Note that Basecamp GUI's built-in tutorials are discovered when the GUI starts so a project can replace the built-in tutorials making it self-documented. Projects can be designed as part of a curriculum or to retain institutional knowledge to address a common concern with university led CubeSats that often suffer from student turnover and struggle with retaining acquired technical knowledge.

FUTURE WORK

First, the Basecamp development team should continue to create new apps, videos, and projects. At least one advanced project like a rover should be maintained by the Basecamp team.

Second, the Basecamp team should mature artifacts that enable the community to create content. This includes formalizing and documenting the app spec so people can publish compliant apps. Also, training material should be developed to support teachers so they can learn how to use Basecamp for developing apps and projects to meet their needs.

Finally, the ability to easily transition from Basecamp to fully functioning ground systems like OpenC3's COSMOS needs to be developed. The general solution is to have an EDS-to-XTCE (XML Telemetric and Command Exchange) converter. However, this would be a significant undertaking. A more viable and immediate solution would be to provide instructions for customizing the EDS tool chain to export command and telemetry definitions formatted for a particular ground system.

SUMMARY

The cFS Basecamp ecosystem provides articles, videos, apps, and projects intended to serve as educational material for the aerospace community and for STEM education. It uses NASA's cFS that has decades of flight heritage. Basecamps standardized app design with app specs combined with its use of EDS provide the infrastructure to automate the process of integrating an app to a cFS target. This same infrastructure allows the cFS community to contribute apps and projects so educational content can grow as the community grows.

Acknowledgments

The authors would like to acknowledge and thank Joe Hickey for maintaining a cFS distribution with an EDS toolchain[10]. Basecamp would not be possible without his efforts. In addition, they would like to thank the PySimpleGUI[12] and NASA cFS[1,2] teams for creating and maintaining outstanding products.

References

1. NASA Goddard Space Flight Center, Flight Software Systems Branch, core Flight System Overview, https://cfs.gsfc.nasa.gov/Introduction.html
2. NASA core Flight System open-source repository, https://github.com/nasa/cFS.
3. OpenSatKit project, https://github.com/OpenSatKit/OpenSatKit/wiki
4. Ball Aerospace COSMOS, https://github.com/BallAerospace/COSMOS
5. NASA 42 Dynamic Simulator, https://github.com/ericstoneking/42
6. OpenC3 COSMOS, https://github.com/OpenC3/cosmos
7. cFS Basecamp, https://github.com/cfs-tools/cfs-basecamp
8. 2023 FSW Workshop, https://flightsoftware.org/workshop/FSW2023
9. OpenMissionStack, https://openmissionstack.com/
10. Joe Hickey cFS Distribution with EDS, https://github.com/jphickey/cfe-eds-framework
11. 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
12. PySimpleGUI, Python GUIs for Humans https://pysimplegui.readthedocs.io/en/latest/
13. MQ Telemetry Transport (MQTT), https://mqtt.org/

14. cFS Basecamp MQTT Gateway App, https://github.com/cfs-apps/mqtt_gw
15. OpenSatKit YouTube Training Video Channel, https://www.youtube.com/channel/UC2wfvAIkrrgyC4ITwL3zokg
16. cFS Basecamp Project Repos, https://github.com/cfs-projects