CFS-101 TRAINING GUIDE

NOTE: This tutorial uses the open source versions of cFE/cFS and OSAL available from https://sourceforge.net.

Table of Contents

---

1. Introduction

Welcome to the training guide for the CFS-101 self-guided tutorial. This tutorial is designed to show the steps for creating, building, and running a simple Core Flight System (cFS) -based software system. It is not designed to dive into a detailed explanation of the Core Flight Executive (cFE) framework or its included cFS applications. This is left to the documentations that are packaged with the cFS source code and information on the cFS website (cfs.gsfc.nasa.gov).

By the end of this tutorial, you will be able to create, build, and run a simple cFS-based software system that can:

- Be commanded from a ground system
- Provide input data to the flight system
- Publish and subscribe data to/from the rest of the system
- Process and monitor the data
- Downlink the flight data to a ground system

To take this tutorial, you should know how to:

- Program in C
- Use the Linux operating system (OS)

1.1. Training prerequisites

1.2. Training materials

This training guide is used with a virtual machine (VM) called CentOS-6.10-CFS-101. Log in to the VM as "dev" user with password "cfs101!!".

The development environment for this VM has been set up and configured with the appropriate system libraries, open-source version of the cFS software, and project workspace for the training.

Due to NASA IT policy, dev user is not set up as a superuser and the root password is not provided. This VM is meant to be used solely for the purpose of this training.

2. cFS Anatomy

The cFS architecture looks like THIS and THIS.

This section describes the source code's directory trees.

Either due to preferences or requirements, a cFS-based project can be set up differently from others in terms of where directories and files are located. Regardless of how the project is set up, the six top-level CFS directories must exist with their sub-directories and contents as pictured in the links below.

Note that it is not required for these top-level directories to be under the same parent directory. However, their sub-directories must be at least as shown in the pictures.

- cfe
  This directory contains source code for the cFE layer (cFE). It is a set of mission-independent, re-usable, core software services, which includes the Executive Services (ES), the Event Services (EVS), the Software Bus Services (SB), the Table Services (TBL), the Time Services (TIME), and the File Services (FS). These services run as child threads on a non-real-time OS like Linux, and as tasks on a real-time OS such as VxWorks.
- osal
  This directory contains source code for the OS Abstraction Layer (OSAL). It provides a common set of Application Programmer Interfaces (APIs) at the OS level. Using the APIs, cFS applications can be coded as OS-independent, increasing portability between different platforms.
- psp

This directory contains source code for the Platform Support Package (PSP) layer. Like the OSAL, it also provides a set of common APIs, but at the processor level.

- apps
  This directory contains source code for the cFS applications, including both re-use and mission-specific applications. Commonly referred to as "apps", these run as child threads on non-real-time OS like Linux, and as tasks on real-time OS such as VxWorks.
- cfs101_defs
  This directory contains build configuration files for `cmake`.
- tools
  This directory contains source code for the supporting tools, including both re-use and mission-specific tools.

  Unlike the other sub-directories, there's no requirement on how things are layed out in the `tools` directory. However, it is recommended that each tool has its own sub-directory. Some tools are provided as part of the cFS release.

For the CFS-101 VM, all six top-level directories are located under the same parent directory in the workspace `/home/dev/Training_workspace/CFS-101`. Note that it is okay to use symbolic links for these directories if desired. The `CFS-101` and `osal` directories are set up that way for this training. Also keep in mind that this setup might not, and does not have to, work for all cFS-based projects. For example, one project might group them as core (cfe, osal, psp) vs. mission-specific (apps, build, tools) for the purpose of version control management.

You should now traverse within the `/home/dev/Training_workspace/CFS-101` directory to get familiar with the file and sub-directory locations.

3. Getting Started With The Initial Flight System

Let's now walk through the steps to build and run a simple cFS-based system, starting with system descriptions.

If you want to reset the workspace to the intial state at any point in the tutorial, execute the following commands:

```
$ cd /home/dev/Training_workspace
$ rm -rf CFS-101.initial
$ tar zxvf ../Archives
/CFS-101.initial.tar.gz
$ rm CFS-101
```

```
$ ln -s CFS-101.initial CFS-101
```

If you want to see the final workspace with all the updates, execute the following commands:

```
$ cd /home/dev/Training_workspace
$ tar zxvf ../Archives/CFS-101.final.tar.gz
$ rm CFS-101
$ ln -s CFS-101.final CFS-101
```

3.1. System components

This VM is setup for you to build a simple CFS-based system initially. The system will utilize the following components:

- `cfe` → core services
- `osal` → OS abstraction layer providing common set of APIs
- `psp` → platform-specific
- `apps` → applications
  - `sch` → sends out scheduled commands to other apps
  - `ci` → receive commands from external sources
  - `to` → send out telemetry to external sources
  - `io_lib` → library used by the `CI` and `TO` applications
- `tools` → tools
  - `gen_app_code` → script that generates code template for a cFS application
- `cFS-GroundSystem` → application that creates ground system displays and data connections

3.2. System behaviors

In general, a cFS-based system behaves as described below. Note that this is not the ONLY way a cFS-based system can be setup. This is just a more common setup.

- At startup, it loads, starts, and initializes its components, cFE core and cFS apps, as threads or tasks.
- If system startup is successful, all the components run according to their thread priority, as pre-defined in the `cfe_es_startup.scr` file.

- When the SCH application runs, it continuously sends out Wake-Up commands to the other applications using a pre-defined scheduling table.
- When a typical cFS application runs, it continuously listens for incoming commands. It does not do any "work" until it receives a Wake-Up command. Only upon receiving a Wake-Up command can an application get to run through its processing cycle (one processing cycle per Wake-Up command).

  An application's processing cycle consists of four parts:
  - Receiving & processing commands
  - Receiving & processing input data
  - Computing output data
  - Sending out output data
- Non-typical cFS applications, like SCH, CI, and TO, are not scheduled. SCH is typically set up to run at the system's minor frame rate; whereas CI and TO can be setup to run on a scheduled or pre-defined periodic rate, or both. Mission requirements should dictate how these applications be configured to run.

For this tutorial, when you build and run your simple system initially, it doesn't do much but responding to Send-Hk command from the SCH application and No-Op and Reset commands from a ground system.

In later sections, you will be adding new applications to the system to make it do something more substantial.

3.3. Setting the build environment variables

Being able to build cFS relies on setting your build environment variables correctly.

1. Open a new terminal to build and run cFS system (from here on it will be referred to as the FSW Terminal, where cFS is built and run from).
2. Go to the project's home directory for the tutorial:

   ```
   $ cd /home/dev/Training_workspace/CFS-101
   ```
3. Set the environment variables for this terminal session by executing the following command:

   ```
   $ source setvars.sh
   ```

You can browse through the file, `setvars.sh`, and see how these build environment variables are defined. HERE is a sample of such a file.

The file has already been set properly for the tutorial, so it does not require modification. In general this file must be modified for a cFS-based project in order to reference the locations of its six top-level directories. This only needs to be done once

per project.

3.4. Building the initial flight system

From the FSW Terminal, you should be at `/home/dev/Training_workspace/CFS-101` directory.

1. Build the initial flight system with the following commands (this build is for a Linux OS):

   ```
   $ make prep
   $ make
   $ make install
   ```

   You should see build output similar to these:

   make prep OUTPUT,
   make OUTPUT,
   make install OUTPUT

3.5. Running the initial flight system

1. Also from the FSW Terminal, go to the `exe` directory for the `cpu1` build:

   ```
   $ cd /home/dev/Training_workspace/CFS-101/build/exe/cpu1
   ```

2. Take a look at the file `./cf/cfe_es_startup.scr`. It is the input data file for the system. It is used by the cFE ES to load and start up cFS libraries and applications. You can read the comments at the end of the file for more detail. HERE is an example of such a file.

   Note what is included in the `cfe_es_startup.scr` for system run can be a subset of what is included in the `/home/dev/Training_workspace/CFS-101/cfs101_defs/targets.cmake` for the `cpu1` build (look for build variable TGT1_APPLIST). There is nothing wrong with building more than what you want to run with cFS.

3. Execute the following command to run the system:

   ```
   $ ./core-cpu1
   ```

   You should see output similar to THIS.

   If an application is successfully loaded and initialized, there should be an output line acknowledging such for each cFS application/library entry in the `cfe_es_startup.scr` file. See an EXCERPT from a sample run output.

After a successful system startup, it looks as though the system is not doing anything, but actually the `SCH` application is cycling through its scheduling table, sending out scheduled commands to other parts of the system. You will see evidence of data being updated when you go through the next section.

4. To exit cFS, enter `<CTRL-C>`.

   You should see output similar to THIS.

3.6. Interacting with the initial flight system

For training purposes let's use the simple ground-system utilities that are included with the cFS release to interact with your newly built system. In this VM the ground system runs on the same computer as the flight system. In a real setup a ground system would run on a different computer that has some type of network connection to the flight computer.

3.6.1. Sending commands to the initial flight system

1. From the FSW Terminal exit and re-run the cFS executable that you built (i.e., `core-cpu1`).
2. Open another terminal to run the Ground System displays (from here on it will be referred to as the GSW Terminal).
3. Execute the following commands:

```
$ cd /home/dev/Training_workspace/CFS-101/tools/cFS-GroundSystem
$ python GroundSystem.py
```

4. From the cFS Ground System page click on the Start Command System button to bring up the Command System Main Page.

Sending `CI` commands

1. From the Command System Main Page click the Display Page button for the CI Commands entry to bring up the CI Commands (CPU1) page.
2. From there send the following commands, in the order listed, to `CI` by clicking their Send buttons:

   CI_NOOP_CC
   CI_RESET_CC
   CI_NOOP_CC

   You should see ground-system and flight-system output similar to THIS.

   The ground system outputs information about the command message content

and the destination for every command sent. The flight system outputs acknowledgements from the applications upon receiving their commands.

3. Close the CI Commands (CPU1) page.

Sending TO commands

1. From the Command System Main Page click the Display Page button for the TO Commands entry to bring up the TO Commands (CPU1) page.
2. From there send the following commands, in the order listed, to TO by clicking their Send buttons:

   TO_NOOP_CC
   TO_RESET_CC
   TO_NOOP_CC

   You should see ground-system and flight-system output similar to THIS.

   The ground system outputs information about the command message content and the destination for every command sent. The flight system outputs acknowledgements from the applications upon receiving their commands.
3. Don't close the TO Commands (CPU1) page just yet.

Now let's see the telemetry on the ground displays.

3.6.2. Receiving telemetry from the initial flight system

1. From the cFS Ground System display, click the Start Telemetry System button to bring up the Telemetry System Main Page.

   You should expect to see no telemetry updating on the Telemetry System Main Page at this point. This is because telemetry output from the flight system hasn't been enabled yet.
2. From the TO Commands (CPU1) page click the Send button for the TO_ENABLE_OUTPUT_CC command to bring up its Parameter Dialog display.
3. From the Parameter Dialog page enter "127.0.0.1" in the input box for the cDestIp parameter and "5011" in the input box for the usDestPort parameter. Then click the Send button at the top of the display page.

   You should see ground-system and flight-system output similar to THIS.

   The ground system outputs information about the command message content and the destination for every command sent. The flight system outputs acknowledgements from the applications upon receiving their commands.
4. Close the Parameter Dialog page and the TO Commands (CPU1) page.

5. Now from the Telemetry System Main Page you should expect to see telemetry relating to packet counts update periodically as the ground system receives telemetry from the flight system. So far only the telemetry from cFE core services are being received. Later on you will add more telemetry from the cFS applications when you go through the exercise for creating new cFS applications.
6. When you are done inspecting the data clear out the desktop for the next section by closing all the Ground System displays and exit CFS running in the FSW Terminal.

3.7. What's next?

If you get to this point, CONGRATULATIONS!!! You've just built and run your first cFS-based system.

Let's take a short break before diving into the next section, where you will create new cFS applications that add more functionalities to the simple system you just built and ran.

4. Developing a New Flight System

Now that you know how to build and run a cFS-based system, let's enhance it to do more than the basics. Then you can see how data is produced and published by one application, and subscribed and consumed by another application.

Let's implement a cFS-based system that will:

- Read from a temperature sensor
- Monitor the values (i.e., increasing/decreasing/not changing)
- Determine the value range (i.e., nominal/hot/cold)

- Set range limits on user command
- Set the temperature value on user command (i.e., update the temperature value)
- Set the delta value on user command (i.e., update the value used to compute the next temperature value)

4.1. New flight system design

This system will need the following CFS applications:

- `sch` → to send scheduled messages to other applications
- `ci` → to receive user commands from the ground system
- `to` → to send out telemetry to the ground system
- `io_lib` → library used by `CI` and `TO`
- `temp_io` → to read and publish temperature data
- `temp_mon` → to monitor the temperature data by subscribing to and consuming `TEMP_IO`'s data

Your initial system already has the first four covered. You now need to add two new applications, `TEMP_IO` and `TEMP_MON`.

4.1.1. `TEMP_IO` application

This application is responsible for reading and publishing temperature data. Let's schedule it to run at 1Hz. Its three-part data processing cycle is as follows:

1. Process the following commands, when received:
    - Wake-Up and Send-Hk
    - No-Op and Reset
    - Set-Current-Temp
    - Set-Delta-Value
2. Read data from a temperature sensor (simulated)
3. Publish the new temperature value

4.1.2. `TEMP_MON` application

This application is responsible for monitoring temperature data, determining if the value is increasing/decreasing, and labeling it as nominal/hot/cold. Let's also schedule it to run @ 1Hz. Its four-part data processing cycle is as follows:

1. Process the following commands, if received,
    - Wake-Up and Send-Hk

- No-Op and Reset
- Set-Cold-Limit
- Set-Hot-Limit

2. Receive the new temperature value via data subscription
3. Compare and determine:
   - Temperature change: increased/decreased/no change
   - Temperate range: nominal/hot/cold
4. Publish the processed data

4.2. Initial creation of the new applications

The first step is to create new applications with minimal functionalities, then integrate them to the initial system and make sure they can be built and run before adding more features. Doing this incrementally can save you a lot of time if you need to isolate and debug a problem.

4.2.1. Creating the new applications from a script

You can always implement a cFS application from scratch. You can also start with a copy of an existing cFS application and modify it. Or you can use a script that comes with the cFE release to generate the initial code for your application that could be built and run as-is. Let's use the script to save you some time.

1. Open a new terminal to create the applications (from here on it will be referred to as the App Terminal, where application code is created).
2. Go to the `gen_app_code` directory and execute the following command to get information on its usage.

   ```
   $ cd /home/dev/Training_workspace/CFS-101/tools
   /gen_app_code
   $ python gen_app_code.py
   ```

   You should see output like THIS.
3. Execute the following command to generate code for both `temp_io` and `temp_mon`:

   ```
   $ python gen_app_code.py MISSION CFS-101 OWNER "Jane Smith"
   OUTDIR /home/dev/Training_workspace/CFS-101/apps APPS
   temp_io temp_mon
   ```

   You should see output similar to THIS.
4. Go to the `apps` directory and verify that two new directories, `TEMP_IO` and `TEMP_MON`, were added:

```
$ cd /home/dev/Training_workspace/CFS-101/apps
$ ls
```

Feel free to take a look at the created directories and generated files at this point.

4.2.2. Updating the new applications' generated files

You need to make some updates to the generated files since the script uses default values that should be unique across all apps:

1. From the App Terminal go to directory `/home/dev/Training_workspace/CFS-101/apps/temp_mon/fsw/platform_inc`.
2. Open the file `temp_mon_msgids.h` and replace its MID values LIKE SO.

Now the MID values for both `TEMP_IO` and `TEMP_MON` are unique across the system.

4.2.3. Adding `cmake` file for the new applications

The current version of the `gen_app_code.py` script does not generate build files for `cmake`, so for now you have to manually add it to the two new applications. This step will go away when the newer version of the script is distributed.

1. Go to the `apps` directory and make a copy of `CMakeLists.txt` file from the `CI` directory:

```
$ cd /home/dev/Training_workspace/CFS-101/apps
$ cp ./ci/CMakeLists.txt ./temp_io
$ cp ./ci/CMakeLists.txt ./temp_mon
```

2. Edit the file `./temp_io/CMakeLists.txt` LIKE SO.
3. Similarly, edit `./temp_mon/CMakeLists.txt` LIKE SO.

4.2.4. Adding header file for cFE-GroundSystem tool

This section is only necessary if the cFE-GroundSystem tool is used to build your ground system. Since this training uses cFE-GroundSystem tool to build ground displays, this section is required until the tool is upgraded to handle parsing of multiple header files.

1. From the FSW Terminal, go to directory `/home/dev/Training_workspace`

`/CFS-101/apps/temp_io/fsw/src`.

2. Create the file `temp_io_msg_for_grnd.h` by copying from `temp_io_msg.h`.

```
$ cp temp_io_msg.h temp_io_msg_for_grnd.h
```

3. Edit the file `temp_io_msg_for_grnd.h` LIKE THIS.
4. Go to directory `/home/dev/Training_workspace/CFS-101/apps/temp_mon/fsw/src`.
5. Create the file `temp_mon_msg_for_grnd.h` by copying from `temp_mon_msg.h`:

```
$ cp temp_mon_msg.h temp_mon_msg_for_grnd.h
```

6. Edit the file `temp_mon_msg_for_grnd.h` LIKE THIS.

4.3. Updating TO's subscription table and SCH's scheduling table

4.3.1. Adding new data subscriptions to TO for downlink

`TO`'s main role is to send telemetry from the flight system. It uses a subscription table to track messages it needs to send. You need to add two subscription entries for `TEMP_IO` and `TEMP_MON` data messages to that table:

1. From the App Terminal, go to directory `/home/dev/Training_workspace/CFS-101/apps/to`.
2. Edit the file `./fsw/tables/to_config.c` LIKE SO to add `TEMP_IO` and `TEMP_MON` messages to `TO`'s subscription table.
3. Edit the file `./CMakeLists.txt` LIKE THIS to include paths to the `TEMP_IO` and `TEMP_MON` header files for `TO`'s build its table.

4.3.2. Scheduling the new applications with SCH

In order for the new applications to do any "work", `SCH` has to send Wake-Up commands to them. Similarly, in order for the applications to send its telemetry, `SCH` has to send Send-Hk commands to them. You need to add new scheduling entries for `TEMP_IO` and `TEMP_MON` to perform these commands.

1. From the App Terminal, go to directory `/home/dev/Training_workspace/CFS-101/apps/sch`.
2. Edit the file `./fsw/tables/sch_def_msgtbl.c` LIKE THIS to add `TEMP_IO` and `TEMP_MON` messages to `SCH` message table.
3. Edit the file `./fsw/tables/sch_def_schtbl.c` LIKE SO to schedule `TEMP_IO` and `TEMP_MON` to run at 1Hz.

4. Edit the file `./CMakeLists.txt` LIKE THIS to include paths to `TEMP_IO` and `TEMP_MON` header files for `SCH`'s build of its tables.

4.4. Building the new flight system

1. From the FSW Terminal, go to directory `/home/dev/Training_workspace/CFS-101`.
2. Add `TEMP_IO` and `TEMP_MON` to the `cpu1` build by editing the file `./cfs101_defs/targets.cmake`, LIKE SO.
3. Also add them to the startup script by editing the file `./cfs101_defs/cpu1_cfe_es_startup.scr`, LIKE THIS.
4. Re-build the initial flight system with the following commands:

```
$ rm -rf build
$ make
$ make install
```

Expect to see similar build output as the one in section titled *Building the initial system*, but with additional entries for `TEMP_IO` and `TEMP_MON`.

See sample output for make OUTPUT and make install OUTPUT.

4.5. Running the new flight system

1. From the FSW Terminal, go to the directory `/home/dev/Training_workspace/CFS-101/build/exe/cpu1`.
2. Run the new system by executing the following command:

```
$ ./core-cpu1
```

Expect to see similar run output as the one in section titled *Running the initial system*, but with additional entries relating to the `TEMP_IO` and `TEMP_MON` applications.

HERE is a sample output. Can you find where `TEMP_IO` and `TEMP_MON` are loaded and then initialized?

4.6. Interacting with the new flight system

First you need to add a few things to the ground system for the new applications.

4.6.1. Adding ground commands for the new applications

Adding commands for the new applications is quite simple with the `cFE-GroundSystem` tool.

Creating `TEMP_IO` commands

1. From the GSW Terminal, go to directory `/home/dev/Training_workspace/CFS-101/tools/cFS-GroundSystem/Subsystems/cmdGui`.
2. Edit the file `CHeaderParser-hdr-paths.txt` LIKE THIS.
3. Execute the following command to generate the command file for `TEMP_IO`:

```
$ python CHeaderParser.py
```

4. Enter the text highlighted in green LIKE SO.

Creating `TEMP_MON` commands

1. From the same directory, `cmdGui`, re-edit the file `CHeaderParser-hdr-paths.txt` LIKE SO.
2. Re-execute the script:

```
$ python CHeaderParser.py
```

3. Enter the text highlighted in green LIKE SO.

Adding new applications to the command pages

1. From the same directory, add the new command files to the command pages by editing the file `command-pages.txt` LIKE THIS.

4.6.2. Sending commands to the new flight system

1. From the FSW Terminal, exit and re-run the cFS executable, `core-cpu1`.
2. Close out all the Ground System displays from the desktop.
3. From the GSW Terminal, re-start the Ground System displays by executing the following commands:

```
$ cd /home/dev/Training_workspace/CFS-101/tools/cFS-GroundSystem
$ python GroundSystem.py
```

4. From the cFS Ground System page, click on the Start Command System button to bring up the Command System Main Page.

Sending `TEMP_IO` commands

1. From the Command System Main Page, click the Display Page button for the TEMP_IO Cmds (CPU1) entry to bring up `TEMP_IO` command page.

2. From there, send the following commands, in the order listed, to `TEMP_IO` by clicking the Send buttons:

   TEMP_IO_NOOP_CC
   TEMP_IO_RESET_CC
   TEMP_IO_NOOP_CC

   You should see ground-system and flight-system output similar to THIS.

   The ground system outputs information about the command message content and the destination for every command sent. The flight system outputs acknowledgements from the applications upon receiving their commands.

Sending `TEMP_MON` commands

1. From the Command System Main Page click the Display Page button for the TEMP_MON Cmds (CPU1) entry to bring up the `TEMP_MON` command page.
2. From there, send the following commands, in the order listed, to `TEMP_MON` by clicking the Send buttons:

   TEMP_MON_NOOP_CC
   TEMP_MON_RESET_CC
   TEMP_MON_NOOP_CC

   You should see ground-system and flight-system output similar to THIS.

   The ground system outputs information about the command message content and the destination for every command sent. The flight system outputs acknowledgements from the applications upon receiving their commands.
3. Close all of the Ground System displays.

4.6.3. Adding telemetry from the new applications to ground system

Like adding commands, adding telemetry for the new applications is quite simple with the `cFS-GroundSystem` tool.

Creating `TEMP_IO` telemetry

1. From the GSW Terminal go to directory `/home/dev/Training_workspace/CFS-101/tools/cFS-GroundSystem/Subsystems/tlmGUI`.
2. Create a new file called `cfs-temp-io-hk-tlm.txt` and edit it LIKE THIS.
3. `cfs-temp-io-hk-tlm.txt` mirrors the definition of `TEMP_IO_HkTlm_t` defined in `/home/dev/Training_workspace/CFS-101/apps/temp_io/fsw/src/temp_io_msg_for_grnd.h`.
4. Save and close the file.

Creating `TEMP_MON` telemetry

1. Similarly, create a new file called `cfs-temp-mon-hk-tlm.txt` and edit it LIKE THIS.
2. `cfs-temp-mon-hk-tlm.txt` mirrors the definition of TEMP_MON_HkTlm_t defined in `/home/dev/Training_workspace/CFS-101/apps/temp_mon/fsw/src/temp_mon_msg_for_grnd.h`.
3. Save and close the file.

Adding new applications to the telemetry pages

1. Add the new telemetry files to the telemetry pages by editing the file `telemetry-pages.txt` LIKE SO.

4.6.4. Receiving telemetry from the new flight system

1. Start up the Ground System displays again from the directory `/home/dev/Training_workspace/CFS-101/tools/cFS-GroundSystem`:

   ```
   $ python GroundSystem.py
   ```

2. From the cFS Ground System display click the Start Telemetry System button to bring up the Telemetry System Main Page.

   You should expect to see no telemetry updating in the Telemetry System Main Page at this point. This is because telemetry output from the flight system hasn't been enabled yet.

3. In the Command System Main Page click the Display Page button for the TO Commands (CPU1) entry to open the TO Commands (CPU1) page.
4. From there, click the Send button for the TO_ENABLE_OUTPUT_CC command to bring up its Parameter Dialog display.
5. Enter "127.0.0.1" in the input box for the cDestIp parameter and "5011" in the input box for the usDestPort parameter. Then click the Send button at the top of the display page.

   You should see ground-system and flight-system output similar to THIS.

   The ground system outputs information about the command message content and the destination for every command sent. The flight system outputs acknowledgements from the applications upon receiving their commands.

6. Now from the Telemetry System Main Page you should expect to see telemetry updates periodically as the ground system receives telemetry from the flight system.

Checking out application-specific telemetry

1. Close the Telemetry System Main Page.
2. From the cFS Ground System display select "127.0.0.1" from the Selected IP Address pull-down menu at the top of the display.
3. Click the Start Telemetry System button to re-open the Telemetry System Main Page.
4. Click on the Display Page button for the Temp Monitor entry to bring up Temp Monitor telemetry page.
5. From the Command System Main Page bring up TEMP_MON Cmds (CPU1) and send a few TEMP_MON_NOOP_CC commands.

   You should expect to see its Command Counter value increased by the number of commands you just sent.

   If you send a TEMP_MON_RESET_CC command, all counter values will be reset to 0.
6. You can apply steps 4 and 5 on `TEMP_IO` if you want to see its telemetry updated.
7. At this point feel free to play with the system. When you are done, clear out the desktop for the next section by closing all the Ground System displays and exit the cFS running in the FSW Terminal.

4.7. What's next?

If you get to this point, CONGRATULATIONS AGAIN!!! You've just added new applications to your initial cFS-based system.

By now you have seen how cFS applications are created and integrated into the system. If you take a closer look at the application's generated code, there are "TO-DO" comments that should guide you where you can add code to enhance your application. There is also a generated `ReadMe.txt` file in the `src` directory of the application that describes how a cFS application works.

You can stop here and venture out on your own at this point. However, if you want to see in details how to enhance an application, you should continue with the tutorial. In the next section, you will add more functions to the `TEMP_IO` and `TEMP_MON` applications.

Take another coffee break before diving into it. You deserve it!

5. Making Enhancements to the New Flight System

Now that you successfully integrated the two new applications, `TEMP_IO` and `TEMP_MON`, to the cFS-based system, you can now focus on just the code that needs to be added to the new applications to cover the rest of the system's functionalities.

5.1. Modifying `TEMP_IO`

Let's summarize `TEMP_IO`'s functions again:

1. It can process the following commands:
    a. Wake-Up and Send-Hk
    b. No-Op and Reset
    c. Set-Current-Temp
    d. Set-Delta-Value
2. It reads data from a temperature sensor (simulated)
3. It publishes the new temperature
4. It runs at 1Hz

The initial application code, generated by the script in the previous section, takes care of items 1a and 1b. Item 4 is also done when you add a Wake-Up command entry to the `SCH`'s scheduling table. Item 2 will be a simulated sensor read by adding the current temperature value to a settable delta value.

You need to add more code to cover the rest of the functions, starting with additional commands.

5.1.1. Adding more commands

1. From the App Terminal, go to directory `/home/dev/Training_workspace`

`/CFS-101/apps/temp_io/fsw/src`.

2. Edit the file `temp_io_msg.h` LIKE SO to add command codes for the new command messages.
3. Edit the file `temp_io_private_types.h` LIKE SO to add data structures for the new command messages.
4. Edit the file `temp_io_app.h` LIKE THIS to define some local variables to be used for generating simulated sensor data.
5. Edit the function TEMP_IO_ProcessNewAppCmds() in the source file `temp_io_app.c` LIKE THIS to handle command responses.

As you can see from the modified code, TEMP_IO_SET_CURRENT_TEMP_CC command will set the current temperature value to the commanded value. Similarly, TEMP_IO_SET_DELTA_VALUE_CC command will set the current delta value to the commanded value.

5.1.2. Reading data

To simulate sensor data read, `TEMP_IO` generates the new temperature value by adding the current temperature value to the current delta value.

This is done at the beginning of every Wakeup cycle for `TEMP_IO`, which is currently at 1Hz.

1. Edit the following functions in the file `temp_io_app.c` LIKE SO:
    a. TEMP_IO_InitData() to initialize local variables
    b. TEMP_IO_ProcessNewData() to compute simulated sensor values

5.1.3. Publishing data

To publish temperature data for other apps to consume, you need to modify `TEMP_IO`'s code as follows.

This is done at the end of every Wakeup cycle for `TEMP_IO`.

1. Edit the file `temp_io_private_types.h` LIKE THIS to add data for publication using the TEMP_IO_OutData_t data structure.
2. Edit the function TEMP_IO_SendOutData() in the source file `temp_io_app.c` LIKE THIS to publish the data.

5.2. Modifying `TEMP_MON`

Let's summarize `TEMP_MON`'s functions again:

1. It can process the following commands:
    a. Wake-Up and Send-Hk

   b. No-Op and Reset
   c. Set-Cold-Limit
   d. Set-Hot-Limit
2. It gets the new temperature value and saves the current value as the old value
3. It compares and determines if the value has increased/decreased, compares and determines which range, nominal/hot/cold, the value falls in
4. It publishes temperature direction and range
5. It runs at 1Hz

The initial application code, generated by the script in the previous section, takes care of items 1a and 1b. Item 5 is also done when you add a Wake-Up command entry to the `SCH`'s scheduling table.

You need to add more code to cover the rest of the functions, starting with additional commands.

5.2.1. Adding more commands

1. From the App Terminal, go to directory `/home/dev/Training_workspace/CFS-101/apps/temp_mon/fsw/src`.
2. Edit the file `temp_mon_msg.h` LIKE SO to add command codes for the new command messages.
3. Edit the file `temp_mon_private_types.h` LIKE SO to add data structures for the new command messages.
4. Edit the file `temp_mon_app.h` LIKE SO to declare variables used to track data.
5. Edit the function TEMP_MON_ProcessNewAppCmds() in the source file `temp_mon_app.c` LIKE THIS to handle command responses.

5.2.2. Computing data

To determine temperature direction and range, `TEMP_MON` looks at the previous and current temperatures as well as the temperature limits.

This is done at the beginning of every Wakeup cycle for `TEMP_MON`, which is currently at 1Hz.

1. Edit the file, `temp_mon_msg.h`, LIKE THIS to define some macro constants used in determining range and direction.
2. Edit the following functions in the file `temp_mon_app.c` LIKE SO:
   a. Include header files, `temp_io_msgids.h` and `temp_io_private_types.h`
   b. TEMP_MON_InitData() to initialize the local variables

c. TEMP_MON_InitPipe() to subscribe to `TEMP_IO` data

d. TEMP_MON_ProcessNewData() to determine temperature range and direction

5.2.3. Publishing data

To publish data for downlink to the ground system, you need to modify `TEMP_MON`'s code as follows.

This is done when `TEMP_MON` receives a TEMP_MON_SEND_HK_MID command message from the `SCH`.

1. Edit the file `temp_mon_msg.h` LIKE THIS to add data for publication using the TEMP_MON_HkTlm_t data structure.
2. Edit the function TEMP_MON_ReportHousekeeping() in the source file `temp_mon_app.c` LIKE THIS to publish the data.

5.3. Building the enhanced flight system

1. From the FSW Terminal, go to directory `/home/dev/Training_workspace/CFS-101`.
2. Execute the following commands:

```
$ rm -rf build
$ make
$ make install
```

Expect to see similar output like THIS and THIS.

5.4. Running the enhanced flight system

1. From the FSW Terminal go to directory `/home/dev/Training_workspace/CFS-101/build/exe/cpu1`.
2. Execute the following command:

```
$ ./core-cpu1
```

Expect to see similar output to THIS. Can you spot what else is being output from your FSW Terminal?

5.5. Interacting with the enhanced flight system

5.5.1. Adding more ground commands

You need to add the new ground commands for `TEMP_IO` and `TEMP_MON` applications.

Creating new TEMP_IO commands

1. From the FSW Terminal go to directory `/home/dev/Training_workspace/CFS-101/apps/temp_io/fsw/src`.
2. Edit the file `temp_io_msg_for_grnd.h` LIKE SO.
3. From the GSW Terminal go to directory `/home/dev/Training_workspace/CFS-101/tools/cFS-GroundSystem/Subsystems/cmdGui`.
4. Edit the file `CHeaderParser-hdr-paths.txt` LIKE THIS.
5. Execute the following command to generate the new command file for `TEMP_IO`:

   ```
   $ python CHeaderParser.py
   ```

6. Enter the input, highlighted in green, LIKE SO.

Creating new TEMP_MON commands

1. From the FSW Terminal go to directory `/home/dev/Training_workspace/CFS-101/apps/temp_mon/fsw/src`.
2. Edit the file `temp_mon_msg_for_grnd.h` LIKE SO.
3. From the GSW Terminal go to directory `/home/dev/Training_workspace/CFS-101/tools/cFS-GroundSystem/Subsystems/cmdGui`.
4. Edit the file `CHeaderParser-hdr-paths.txt` LIKE THIS.
5. Execute the following command to generate the new command file for `TEMP_MON`:

   ```
   $ python CHeaderParser.py
   ```

6. Enter the input, highlighted in green, LIKE SO.

5.5.2. Sending commands to the enhanced flight system

Now see if you can send new commands to the modified system and get data feedbacks. The steps are exactly like those in the section *`Sending commands to the new flight system`*.

1. Changes in `TEMP_IO` command displays:
   a. TEMP_IO Cmds (CPU1) page has additional commands
   b. Parameter Dialog for TEMP_IO_SET_CURRENT_TEMP_CC command is new
   c. Parameter Dialog for TEMP_IO_SET_DELTA_VALUE_CC command is new
2. Changes in `TEMP_MON` command displays
   a. TEMP_MON Cmds (CPU1) page has additional commands

   b. Parameter Dialog for TEMP_MON_SET_COLD_LIMIT_CC is new
   c. Parameter Dialog for TEMP_MON_SET_HOT_LIMIT_CC is new

3. Now send some of the new commands to the enhanced flight system.

   From the TEMP_IO Cmds (CPU1) page:

   a. Send the TEMP_IO_SET_CURRENT_TEMP_CC command to set the current temperature to 101
   b. Send the TEMP_IO_SET_DELTA_VALUE_CC command to set the delta value to 10

      When you set the current temperature expect to see from the FSW output that the emperature value has jumped to the set value before it continues to increase by 1 again. Then, when you set the delta value, temperature value should be increasing by the newly set delta value.

4. And from the TEMP_MON Cmds (CPU1) page:

   a. Send the TEMP_MON_SET_COLD_LIMIT_CC command to set cold limit at -10
   b. Send the TEMP_MON_SET_HOT_LIMIT_CC command to set hot limit at 200

You should see output similar to THIS and THIS.

5.5.3. Receiving telemetry from the enhanced flight system

1. Go to directory `/home/dev/Training_workspace/CFS-101/tools/cFS-GroundSystem/Subsystems/tlmGUI`.
2. Edit the file `cfs-temp-mon-hk-tlm.txt` LIKE THIS to describe temperature data being downlinked.
3. Send the TO_ENABLE_OUTPUT_CC command to `TO` to enable telemetry output (see *`Receiving telemetry from the new flight system`*).

Checking out application-specific telemetry

1. From the cFS Ground System display select 127.0.0.1 from the Selected IP Address pull-down menu at the top of the display.
2. Click the Start Telemetry System button to bring up the Telemetry System Main Page.
3. From there, click on the Display Page button for the Temp Monitor entry to bring up the Temp Monitor telemetry page.
4. Now send `TEMP_IO` and `TEMP_MON` commands to the flight system and watch `TEMP_MON`'s telemetry get updated.

a. Send the TEMP_IO_SET_CURRENT_TEMP_CC command to reset the current temperature value.

b. Send the TEMP_IO_SET_DELTA_VALUE_CC command to change the update rate of the current temperature. If the delta value goes from positive to negative, or vice versa, the temperature direction should eventually change as well.

c. Send the TEMP_MON_SET_COLD_LIMIT_CC command to change the cold threshold value. When the current temperature is less than the cold threshold value, the temperature range should change.

d. Send the TEMP_MON_SET_HOT_LIMIT_CC command to change the hot threshold value. When the current temperature is greater than the hot threshold value, the temperature range should change.

5.6. What's next?

The primary objective of this training is to give you a feel for how new capabilities can be quickly and easily added to the system just by using the cFE framework, its applications, and tools. Hopefully this training has achieved that. :-)

There are many things you can do from this point on. You can expand the system further by either adding more functionality to the existing applications or add new applications. How about reconfiguring some of the existing applications? For example, `SCH` application is running at 100Hz by default. Try to configure it to run at 40Hz instead. Which header and/or source files do you have to modified? While you're at it, look at the detailed design document and user's guide for the `SCH` application. These should be at `/home/dev/Training_workspace/CFS-101/apps/sch/docs`.

If you're interested in learning more about the cFE framework itself, take a look at `/home/dev/Training_workspace/CFS-101/cfe/docs`.

You might also be interested in the Questions & Answers section as well.

6. The End?

Thank you for your interest and happy coding!!!

7. Questions & Answers

Click HERE for answers to some common cFS questions.

Last updated 2019-01-30 15:09:49 CST