

Build and Run cFS Target Tutorial

Objectives

- Describe how to use cFS Basecamp to build and run a core Flight System (cFS) Target

Lessons

1. Define what a cFS Target is and how to build one
2. Describe how to run a cFS Target
3. Introduce Application Specifications and describe how they are used in Basecamp's automated build and runtime environments

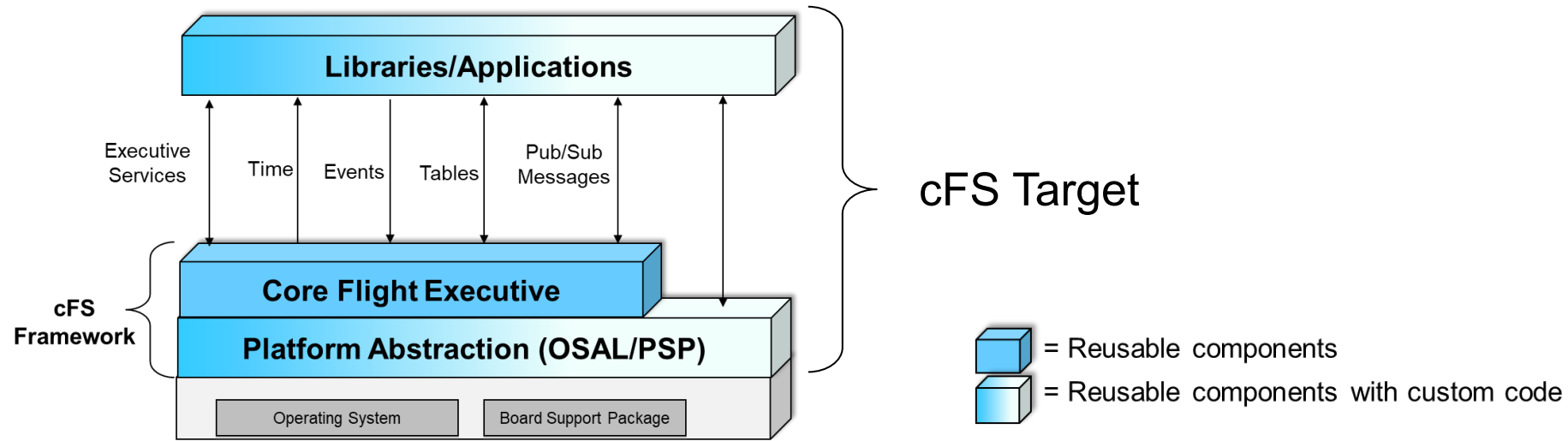
Notes

- Basecamp's demo app APP_C_DEMO is used as an example throughout this tutorial
 - APP_C_DEMO is part of the default cFS Target so the examples can be referenced
- Basecamp is a cFS application development and runtime platform and does not address porting the cFS to different processor/operating system platforms.
- <https://github.com/cfs-tools/cfs-platform-list> provides a list of community supported cFS ports.

Lesson 1

Build a cFS Target

What is a cFS Target?



**A *cFS Target* is the cFS Framework built as an executable
and
A collection of libraries and applications built as object files that are loaded
during the cFS initialization process**

cFS Target Source Directories**

- Highlighted directories contain source files for the *cFS Target*
- Additional build related directories are listed
- Note “apps” directories include both libraries and applications

cfs-basecamp

--apps/	Preinstalled Basecamp apps: ‘ <i>Operations Services App Suite</i> ’ and Demo App
--cfe-eds-framework/		
--apps/	<i>Command Ingest (ci_lab)</i> used in ‘Operational Service App Suite’
--build/	Output directory containing the artifacts produced by the build process
--cfe/	cFS Framework , includes Electronic Data (EDS) Sheet base type definitions
--basecamp_defs/	Top-level cmake files that define the cFS Target
--eds/	EDS configurations and Topic ID definitions
--libs/		
--osal/		
--psp/		
--tools/	Electronic Data Sheet build tools
--usr/		
--apps/	User apps added to Basecamp

** Apps can reside outside of Basecamp’s directory

Target Platform Configuration (1 of 2)

- Before describing the steps to create a target, one mission configuration that impacts the target build must be addressed
- Mission configurations are defined in the *config.xml* EDS file

```
cfs-basecamp
├--apps/
└--cfe-eds-framework/
    ├──apps/
    ├──cfe/
    └-- basecamp_defs/
        └-- eds/config.xml
```

Target Platform Configuration (2 of 2)

```
<Package name="CFE_MISSION" shortDescription="cFE mission configuration">
  <!--
    Note on header type selection - this should be defined to the bare
    (non-qualified) EDS type name. The MSG EDS file and associated
    code assume that the message type is defined in the CCSDS namespace
  -->
  <!-- Select CCSDS v1 headers only (no APID qualifiers) -->
  <Define name="MSG_HEADER_TYPE" value="SpacePacketBasic" />
  <!-- Select CCSDS v2 headers (with APID qualifiers) -->
  <!-- <Define name="MSG_HEADER_TYPE" value="SpacePacketApidQ" /> -->

  <!-- Use 16-bit subseconds field for a 48-bit timestamp in TLM messages -->
  <Define name="MSG_TELEMETRY_SUBSECONDS_TYPE" value="uint16"/>
  <Define name="DATA_BYTE_ORDER" value="bigEndian"/>
```

In the CFE_MISSION package, set the host platform's memory endian setting to match your processor's memory type **

- Set DATA_BYTE_ORDER to either "littleEndian" or "bigEndian" (default) **

** There's a bug in Basecamp's version of the EDS Toolchain that causes issues with cFE file header reads/writes. Big endian seems to work even on little endian Intel machines so big endian is the default. See Basecamp issue #121.

How to Integrate Apps into a cFS Target (1 of 2)

App integration and build steps

1. Add the app's message topic IDs to the target
2. Add the app to the target's cmake build environment
3. Add the app to the Executive Service's startup script
4. Update Operations Services App Suite tables to accommodate the app
5. Build the new cFS Target

How to Integrate Apps into a cFS Target (2 of 2)

- Basecamp's GUI automates the processes of integrating an app into a cFS Target if the app includes a Basecamp application specification (App Spec)
- Apps without a Basecamp App Spec can be integrated manually; however, all apps must include an EDS file that defines an app's command and telemetry messages
- Basecamp provides an 'app proxy' feature that allows NASA style apps (including binary tables) to be automatically integrated into a target
- This tutorial first describes the required steps for integrating an app independent of a Basecamp app spec and then describes the automated, semi-automated and manual integration processes

Step 1 – Add Message Topic IDs to the Target**

9/3

cfs-basecamp

```
--apps/
--cfe-eds-framework/
  |--apps/
  |--build/
    |-- exe/cpu1/core-cpu1 . . . . . Core Flight Executive (cFE) executable
    |-- exe/cpu1/cf/*. * . . . . . "Compact Flash (CF)" contains app/lib object files, app default table files
  |--cfe/
  |--basecamp_defs/
    |--eds/
      |-- cfe-topicids.xml . . . . . Defines a mission's telecommand and telemetry message Topic ID values
      |--global_build_options.cmake . . Defines app source search path
      |--mission_build_custom.cmake . . Defines compiler options
      |--targets.cmake . . . . . Defines the target name cpu1, the apps/libs in the cpu1 target, and
      |                               files to copy from basecamp_defs/ to build/exe/cpu1/cf/
      |--cpu1_cfe_es_startup.scr . . . . Defines the apps/libs that are loaded when core-cpu1 runs
    |--libs/
  |--Makefile . . . . . Controls the build process; GNU-make wrapper that calls the CMake tools
  |--osal/
  |--psp/
  |--tools/
  |--usr/
    |--apps/
```

**** Lesson 3 and the Basecamp App Dev Guide describe how to define app EDS messages and their Topic IDs**

Update cfe-topicsids.xml (1 of 2)

18

Add the app's telecommand Topic ID defined in its EDS spec to cfe-topicsids.xml

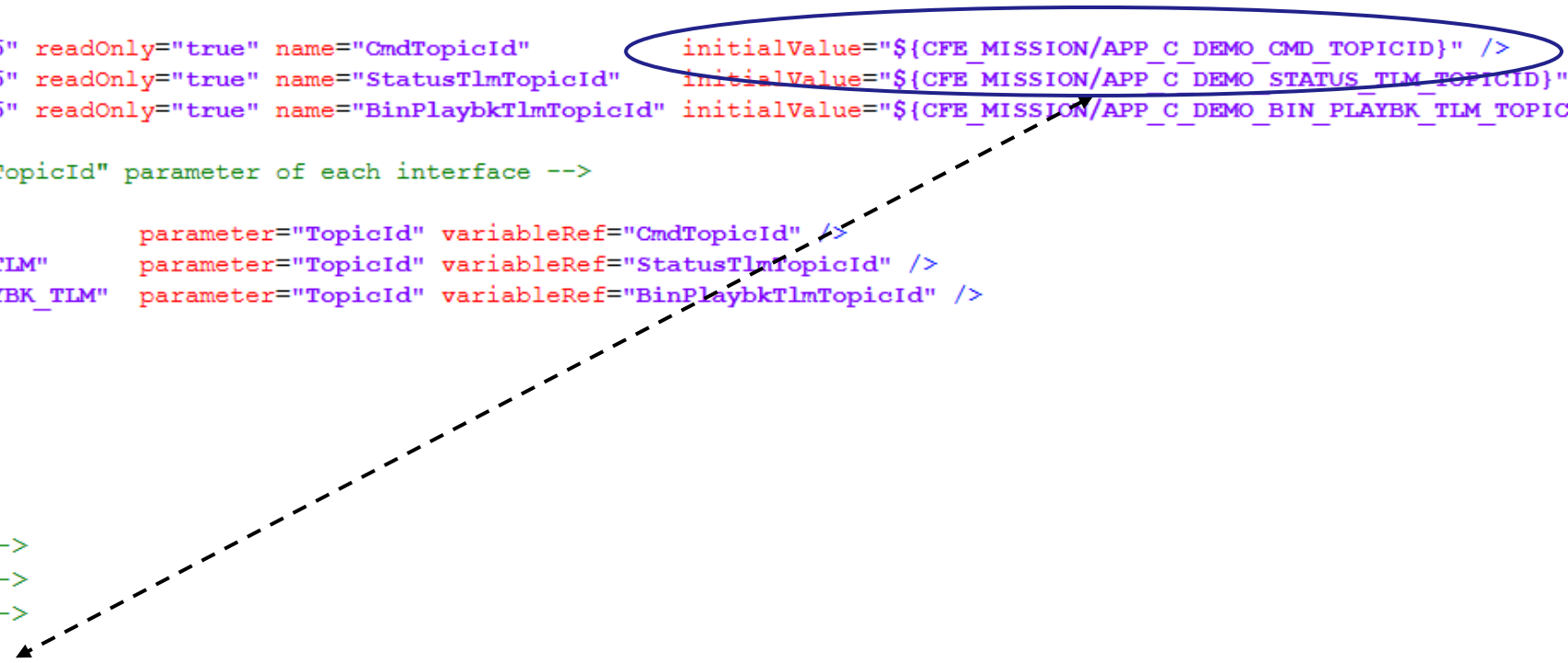
app_c_demo.xml

```
<Implementation>
  <VariableSet>
    <Variable type="BASE_TYPES/uint16" readOnly="true" name="CmdTopicId"           initialValue="${CFE_MISSION/APP_C_DEMO_CMD_TOPICID}" />
    <Variable type="BASE_TYPES/uint16" readOnly="true" name="StatusTlmTopicId"      initialValue="${CFE_MISSION/APP_C_DEMO_STATUS_TLM_TOPICID}" />
    <Variable type="BASE_TYPES/uint16" readOnly="true" name="BinPlaybkTlmTopicId"  initialValue="${CFE_MISSION/APP_C_DEMO_BIN_PLAYBK_TLM_TOPICID}" />
  </VariableSet>
  <!-- Assign fixed numbers to the "TopicId" parameter of each interface -->
  <ParameterMapSet>
    <ParameterMap interface="CMD"           parameter="TopicId" variableRef="CmdTopicId" />
    <ParameterMap interface="STATUS_TLM"    parameter="TopicId" variableRef="StatusTlmTopicId" />
    <ParameterMap interface="BIN_PLAYBK_TLM" parameter="TopicId" variableRef="BinPlaybkTlmTopicId" />
  </ParameterMapSet>
</Implementation>
```

cfe-topicsids.xml

```
<!-- ##### -->
<!-- ## Development App ## -->
<!-- ##### -->

<Define name="APP_C_DEMO_CMD_TOPICID" value="${CFE_MISSION/TELECOMMAND_BASE_TOPICID} + 28"/>
<Define name="PROTO_CMD_TOPICID" value="${CFE_MISSION/TELECOMMAND_BASE_TOPICID} + 29"/>
<Define name="SAMPLE_APP_CMD_TOPICID" value="${CFE_MISSION/TELECOMMAND_BASE_TOPICID} + 30"/>
<Define name="SAMPLE_APP_SEND_HK_TOPICID" value="${CFE_MISSION/TELECOMMAND_BASE_TOPICID} + 31"/>
```



Update cfe-topics.xml (2 of 2)

Add the app's telemetry Topic IDs defined in its EDS spec to cfe-topics.xml

app_c_demo.xml

```
<Implementation>
  <VariableSet>
    <Variable type="BASE_TYPES/uint16" readOnly="true" name="CmdTopicId" initialValue="{CFE_MISSION/APP_C_DEMO_CMD_TOPICID}" />
    <Variable type="BASE_TYPES/uint16" readOnly="true" name="StatusTlmTopicId" initialValue="{CFE_MISSION/APP_C_DEMO_STATUS_TLM_TOPICID}" />
    <Variable type="BASE_TYPES/uint16" readOnly="true" name="BinPlaybkTlmTopicId" initialValue="{CFE_MISSION/APP_C_DEMO_BIN_PLAYBK_TLM_TOPICID}" />
  </VariableSet>
  <!-- Assign fixed numbers to the "TopicId" parameter of each interface -->
  <ParameterMapSet>
    <ParameterMap interface="CMD" parameter="TopicId" variableRef="CmdTopicId" />
    <ParameterMap interface="STATUS_TLM" parameter="TopicId" variableRef="StatusTlmTopicId" />
    <ParameterMap interface="BIN_PLAYBK_TLM" parameter="TopicId" variableRef="BinPlaybkTlmTopicId" />
  </ParameterMapSet>
</Implementation>
```

cfe-topics.xml

```
<!-- ##### -->
<!-- ## Development App ## -->
<!-- ##### -->

<Define name="APP_C_DEMO_STATUS_TLM_TOPICID" value="{CFE_MISSION/TELEMETRY_BASE_TOPICID} + 35"/>
<Define name="APP_C_DEMO_BIN_PLAYBK_TLM_TOPICID" value="{CFE_MISSION/TELEMETRY_BASE_TOPICID} + 36"/>
<Define name="PROTO_HK_TLM_TOPICID" value="{CFE_MISSION/TELEMETRY_BASE_TOPICID} + 37"/>
<Define name="SAMPLE_APP_HK_TLM_TOPICID" value="{CFE_MISSION/TELEMETRY_BASE_TOPICID} + 38"/>
<Define name="SAMPLE_APP_TBL_FILE_TOPICID" value="{CFE_MISSION/TELEMETRY_BASE_TOPICID} + 39" shortDes
```

Step 2 – Add App to Target's CMake

cfs-basecamp

```
--apps/
--cfe-eds-framework/
  |--apps/
  |--build/
    |-- exe/cpu1/core-cpu1 . . . . . Core Flight Executive (cFE) executable
    |-- exe/cpu1/cf/*. * . . . . . "Compact Flash (CF)" contains app/lib object files, app default table files
  |--cfe/
  |--basecamp_defs/
    |--global_build_options.cmake . . Defines app source search path
    |--mission_build_custom.cmake . . Defines compiler options
    |--targets.cmake . . . . . Defines the target name cpu1, the apps/libs in the cpu1 target, and
    |                               files to copy from basecamp_defs/ to build/exe/cpu1/cf/
    |--cpu1_cfe_es_startup.scr . . . Defines the apps/libs that are loaded when core-cpu1 runs
  |--libs/
  |--Makefile . . . . . Controls the build process; GNU-make wrapper that calls the CMake tools
  |--osal/
  |--psp/
  |--tools/
--usr/
  |--apps/
```

target.cmake

A. Add the new target app's filename (no extension) to the `cpu1_APPLIST`

```
SET(cpu1_APPLIST ci_lab kit_to kit_sch file_mgr file_xfer app_c_demo)
```

B. Add files that need to be copied from the `cfe-eds-framework/basecamp_defs/` directory to the `cfe-eds-framework/build/exe/cpu1/cf` to `cpu1_FILELIST`

```
SET(cpu1_FILELIST cfe_es_startup.scr app_c_demo_ini.json1 app_c_hist_tbl.json2 ...)
```

Notes

1. All Basecamp apps have a JSON initialization file that define runtime parameters
2. The demo app has a histogram parameter table called *app_c_hist.json*
3. cFE style apps do not typically have any files that need to be copied, binary table files are copied from an app's directory to `cfe-eds-framework/build/exe/cpu1/cf` as part of the build process

How to Include Apps Outside of Basecamp Directories

global_build_options.cmake allows users to append the **MISSION_MODULE_SEARCH_PATH** that is used to locate app source directories

```
cfs-basecamp
└─ cfe-eds-framework/
   └─ basecamp_defs/
      ├── global_build_options.cmake . . . Defines app source search path
      ├── mission_build_custom.cmake . . . Defines compiler options
      ├── targets.cmake . . . . . Defines the target name cpu1, the apps/libs in the cpu1 target, and
      └── cpu1_cfe_es_startup.scr . . . . Defines the apps/libs that are loaded when core-cpu1 runs
```

C. Add a relative path to the directory containing your app

```
list(APPEND MISSION_MODULE_SEARCH_PATH
    "../usr/apps"
    "../apps"
    "../..../mydir/example_app")
```

•
•

Slide 15

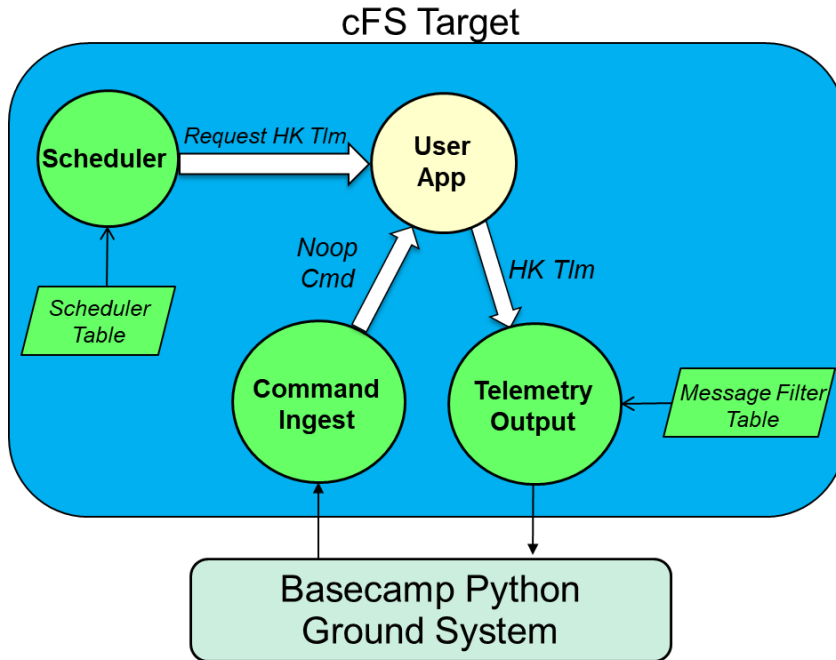
cfe_es_startup.scr

16

- cfe_es_startup.scr defines which libraries and apps are loaded during initialization
- The filename in basecamp_defs is cpu1_cfe_es_startup.scr to associate it with target cpu1 and the cpu1_ prefix is removed when the file is copied to the build/exe/cf directory

Object Type	Object Filename	Main Entry Symbol	cFE Name	Priority	Stack Size
CFE_LIB,	cfe_assert,	CFE_Assert_LibInit,	ASSERT_LIB,	0,	0,
CFE_LIB,	app_c_fw,	APP_C_FW_LibInit,	APP_C_FW,	0,	0,
CFE_APP,	app_c_demo,	APP_C_DEMO_AppMain,	APP_C_DEMO,	80,	32768,
CFE_APP,	ci_lab,	CI_Lab_AppMain,	CI_LAB_APP,	60,	16384,
CFE_APP,	kit_to,	KIT_TO_AppMain,	KIT_TO,	20,	32768,
CFE_APP,	file_mgr,	FILE_MGR_AppMain,	FILE_MGR,	80,	16384,
CFE_APP,	file_xfer,	FILE_XFER_AppMain,	FILE_XFER,	80,	16384,
CFE_APP,	kit_sch,	KIT_SCH_AppMain,	KIT_SCH,	10,	32768,

Operations Services App Suite Tables



- Basecamp's **Scheduler** and **Telemetry Output** apps are part of the *Operations Services App Suite* that provide functionality to support apps to run as part of an flight-ground operational system
- The Scheduler app send messages on the Software Bus at fixed intervals to signal apps to perform a particular function
- Telemetry Output receives messages from the Software Bus and sends them to an external destination

- An app's telemetry message must be added to *Telemetry Output's Message Table* in order to be sent to the ground system
- *Scheduler* app's *Scheduler Table* does not need to be updated if an app uses pre-defined periodic messages

Update the Telemetry Output Table (1 of 2)

- The kit telemetry output (KIT_TO) app uses a packet filter table defined in `cpu1_kit_so_pkt_tbl.json` to determine which telemetry messages are read from the software bus and sent to an external systems over a UDP port
- The table has two JSON objects that must be updated
 1. The ***topic-id-map*** maps a Topic ID name to an index in the packet-array
 2. The ***packet-array*** contains filter parameters that define how often a packet is output

Update the Telemetry Output Table (2 of 2)

```
"topic-id-map": {  
  "start": true,  
  
  "APP_C_DEMO_STATUS_TLM_TOPICID" : "topic-id-40",  
  "APP_C_DEMO_BIN_PLAYBK_TLM_TOPICID" : "topic-id-41",  
  "APP_DEV_HK_TLM_TOPICID" : "topic-id-42",  
  "SAMPLE_APP_HK_TLM_TOPICID" : "topic-id-43",  
  "TEST_HK_TLM_TOPICID" : "topic-id-44",  
  
  "end": true  
},
```

```
"packet-array": [  
  {  
    "packet": {  
      "name": "APP_C_DEMO_STATUS_TLM_TOPICID",  
      "topic-id-40": 2147,  
      "topic-id": 2147,  
      "forward": false,  
      "priority": 0,  
      "reliability": 0,  
      "buf-limit": 4,  
      "filter": { "type": 2, "X": 1, "N": 1, "O": 0 }  
    }  
  },  
  {  
    "packet": {  
      "name": "APP_C_DEMO_BIN_PLAYBK_TLM_TOPICID",  
      "topic-id-41": 2148,  
      "topic-id": 2148,  
      "forward": false,  
      "priority": 0,  
      "reliability": 0,  
      "buf-limit": 2,  
      "filter": { "type": 2, "X": 1, "N": 1, "O": 0 }  
    }  
  },  
]
```

- The Topic ID strings must match the names in an app's EDS spec
- Basecamp's automated app integration populates the *topic-id* value
- Demo App's parameter settings except the topic-id can be used
 - The *filter* settings cause every message received to be output
 - Manual updating the *topic-id* value is discussed in the next section

Scheduler App Tables

28
/ 3

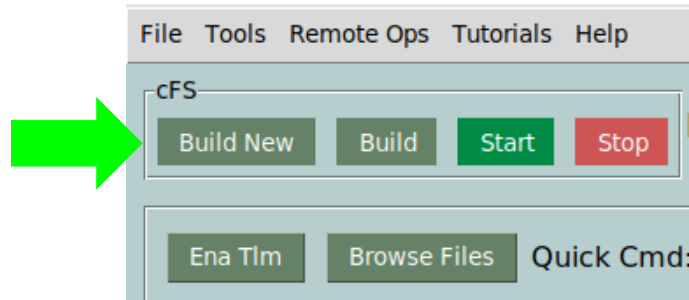
- The kit scheduler (KIT_SCH) app uses a scheduler table defined in `cpu1_kit_sch_schtbl.json` to determine when to send messages on the software bus
- Basecamp's default scheduler table divides a second into four 250ms slots and each slot can have up to 15 messages sent
- The following periodic messages are provided by Basecamp and available for apps to use
 - BC_SCH_1_HZ_TOPICID
 - BC_SCH_2_HZ_TOPICID
 - BC_SCH_4_HZ_TOPICID
 - BC_SCH_2_SEC_TOPICID
 - BC_SCH_4_SEC_TOPICID
 - BC_SCH_8_SEC_TOPICID
- If an app uses a predefined periodic message to perform a periodic function like sending status telemetry, then the scheduler app table does not need to be updated
 - This is the recommended approach

Step 5 – Build the New cFS Target

When a new app is added to a cFS Target a clean must be performed which can be done from Basecamp's main window or from the command line

A. To build using the GUI on the main select <Build New>

- The build status is displayed in the *cFS Target Process Window*



- If the build fails, go to the Linux window used to launch Basecamp to see the error messages

B. To build from the command line open a new terminal window and enter the following commands

```
cd cfs-basecamp/cfe-eds-framework
make distclean
make SIMULATION=native prep
make topicids
```

cFS Target Integration and Build Options

Introduction

- **When cFS Basecamp is installed the default cFS Target is preconfigured so the only required action is to build the cFS Target from the command line**
- **Basecamp supports adding and removing three classes of user apps from the cFS Target**
 1. User Basecamp style apps with Basecamp App Specs
 2. User NASA Proxy App (includes a Basecamp App Spec)
 3. User apps without a Basecamp App Spec
- **The remainder of this tutorial describes how to integrate each app class**
- **Refer to the Basecamp Application Developers Guide for complete details about Basecamp style apps**

1 – User Basecamp Style Apps (1 of 3)

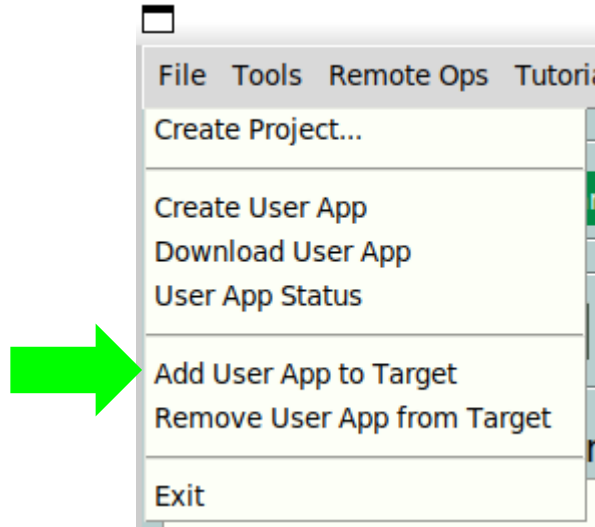
cfs-basecamp

```
|-- . . .
|--usr/
    |--apps/
        |--hi_world/
            |-- CMakeLists.txt . . . . . Cmake instructions
            |-- hi_world.json . . . . . Basecamp App Spec
            |--eds/
                |-- hi_world.xml . . . . . EDS command and telemetry specs
            |-- fsw/
                |--/mission_inc/
                    |-- hi_world_mission_cfg.h . . . Mission scope configurations
                |--/platform_inc/
                    |-- hi_world_platform_cfg.h . . . Platform scope configurations
                |--src/
                    |-- app_cfg.h . . . . . App integration configurations
                    |-- hi_world_app.h
                    |-- hi_world_app.c
                |--tables/
                    |-- hi_world_ini.json . . . . . Runtime parameters loaded during initialization
```

If an app is located in the cfs-basecamp/usr/apps directory and contains a Basecamp App Spec and an EDS Spec, Basecamp's automated integration and build system can be used

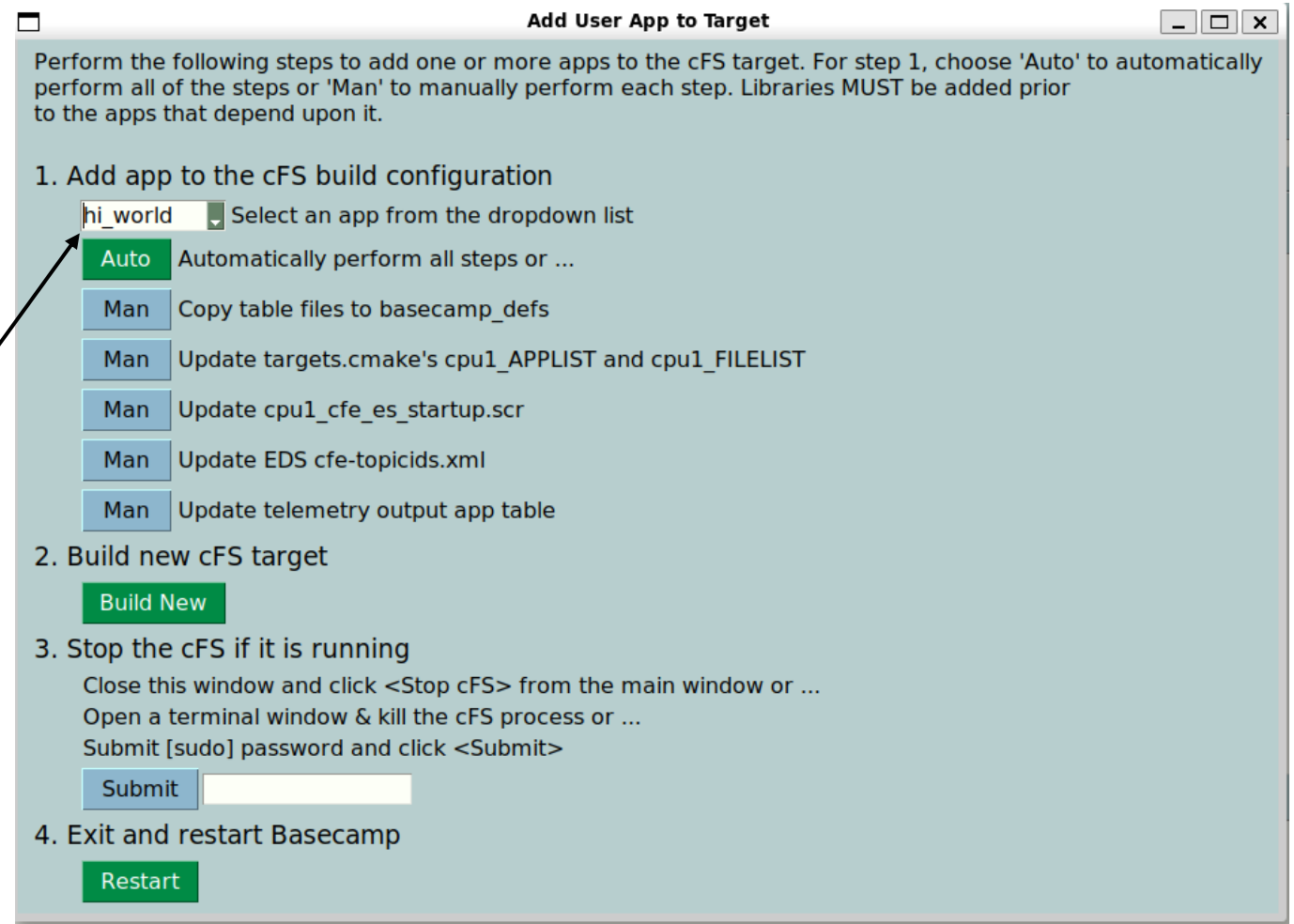
1 – User Basecamp Style Apps (2 of 3)

1. Select *File->Add User App to Target*



2. Select your user app from the dropdown menu and click <Auto>

- a. The autonomous option is recommended for normal user operations. The manual options are for educational situations.



1 –User Basecamp Style Apps (3 of 3)

28
/ 3

Add User App to Target

Perform the following steps to add one or more apps to the cFS target. For step 1, choose 'Auto' to automatically perform all of the steps or 'Man' to manually perform each step. Libraries MUST be added prior to the apps that depend upon it.

1. Add app to the cFS build configuration

hi_world

Select an app from the dropdown list

Auto

Automatically perform all steps or ...

Man

Copy table files to basecamp_defs

Man

Update targets.cmake's cpu1_APPLIST and cpu1_FILELIST

Man

Update cpu1_cfe_es_startup.scr

Man

Update EDS cfe-topics.xml

Man

Update telemetry output app table

2. Build new cFS target

Build New

3. Stop the cFS if it is running

Close this window and click <Stop cFS> from the main window or ...
Open a terminal window & kill the cFS process or ...
Submit [sudo] password and click <Submit>

Submit

4. Exit and restart Basecamp

Restart

3. Build the cFS

- This uses the 'make topicids' command option that is described in Lesson 2

4. Stop the cFS if it is running

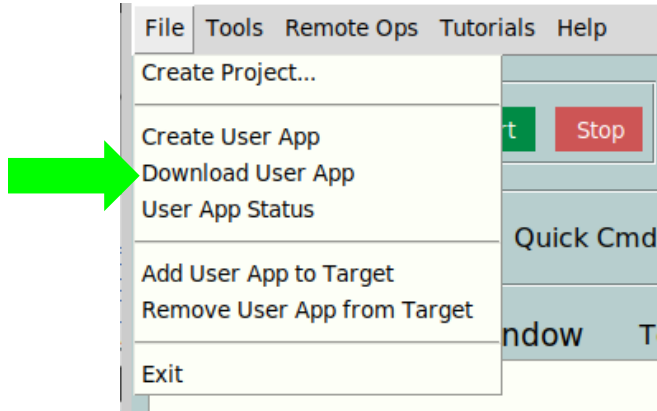
5. Restart Basecamp's GUI

- This causes Basecamp to use the new python libraries created by the cFS build process

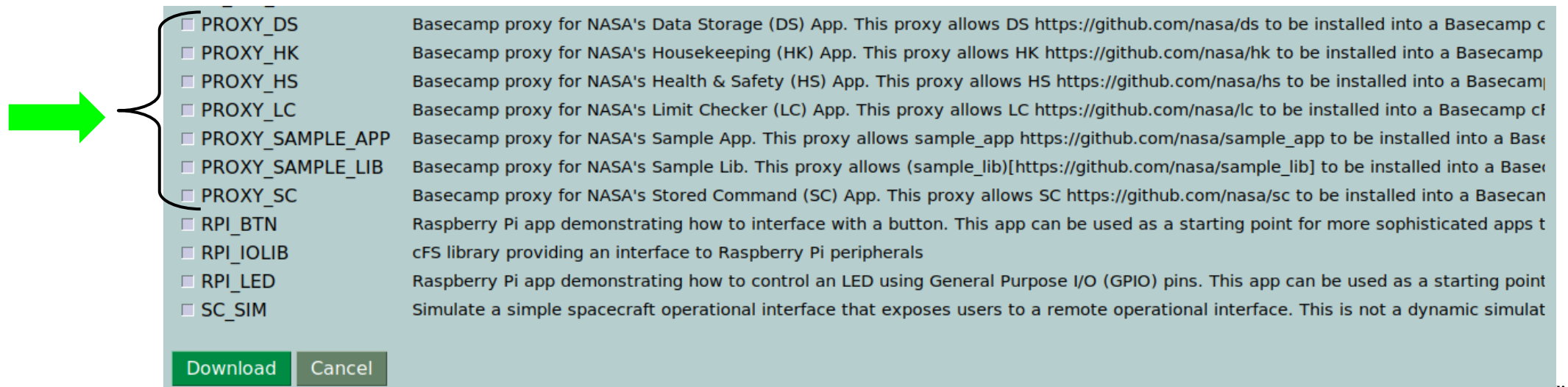
2 – User NASA Proxy App (1 of 2)

Perform the following steps to download and integrate a NASA app into a cFS Target

A. Use the *File->Download User App* menu to get a list of proxy apps



B. Select an app and click <Download>



2 – User NASA Proxy App (2 of 2)

C. Use File->*Add User App to Target* and follow the same steps as a Basecamp style app

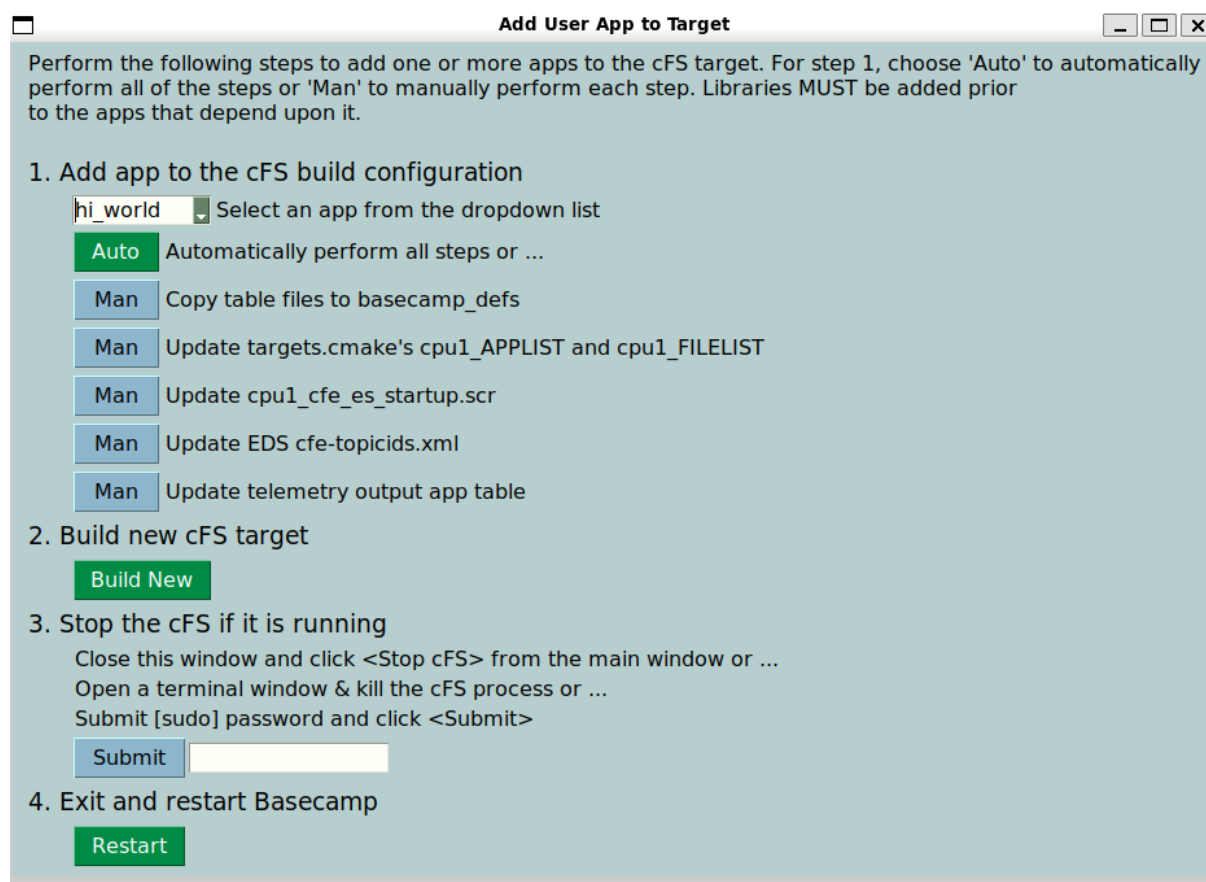
Notes

1. Basecamp's proxy app is added to the cfs-basecamp/usr/apps but will not show up in the *Add User App to Target* window's dropdown list
2. The proxy app provides a Basecamp App Spec for the NASA app and overwrites any files that need to be modified to make the NASA app work with Basecamp's cFS version
3. Proxy app github repositories can be viewed at <https://github.com/orgs/cfs-apps/repositories>
4. The technical details are contained in the Basecamp App Developer's Guide

3 – User Apps without a Basecamp App Spec** (1 of 2)

A. Select *File->Add User App to Target*

- The manual steps will be followed
- A simple text editor is opened when a file needs to be changed



The screenshot shows a window titled "Add User App to Target" with standard window controls (minimize, maximize, close). The window contains the following text and controls:

Perform the following steps to add one or more apps to the cFS target. For step 1, choose 'Auto' to automatically perform all of the steps or 'Man' to manually perform each step. Libraries MUST be added prior to the apps that depend upon it.

1. Add app to the cFS build configuration

hi_world Select an app from the dropdown list

Auto Automatically perform all steps or ...

Man Copy table files to basecamp_defs

Man Update targets.cmake's cpu1_APPLIST and cpu1_FILELIST

Man Update cpu1_cfe_es_startup.scr

Man Update EDS cfe-topicsids.xml

Man Update telemetry output app table

2. Build new cFS target

Build New

3. Stop the cFS if it is running

Close this window and click <Stop cFS> from the main window or ...

Open a terminal window & kill the cFS process or ...

Submit [sudo] password and click <Submit>

Submit

4. Exit and restart Basecamp

Restart

***Creating an app spec may be less work than these steps*

3 – User Apps without a Basecamp App Spec** (2 of 2)

Man	Copy table files to basecamp_defs
Man	Update targets.cmake's cpu1_APPLIST and cpu1_FILELIST
Man	Update cpu1_cfe_es_startup.scr
Man	Update EDS cfe-topicids.xml
Man	Update telemetry output app table

- B. Since there's no app spec then it's probably not a Basecamp style app and no files need to be copied
- C. Update *targets.cmake* as described in integration Step 2
- D. Update *cpu1_cfe_es_startup.scr* as described in integration Step 3
- E. Update *cfe-topicids.xml* as described in integration Step 1
- F. Update *cpu1_kit_so_pkt_tbl.json* as described in integration Step 4
 - You will need to compute your app's messages topic ID values
 - In *cfe-topicids.xml* locate a topic ID name near your app and compute the relative offset
 - In *cpu1_kit_so_pkt_tbl.json* find the same reference topic ID name and add the offset to the reference topic ID name's value to determine your app's topic ID value