Build and Run cFS Target Tutorial

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

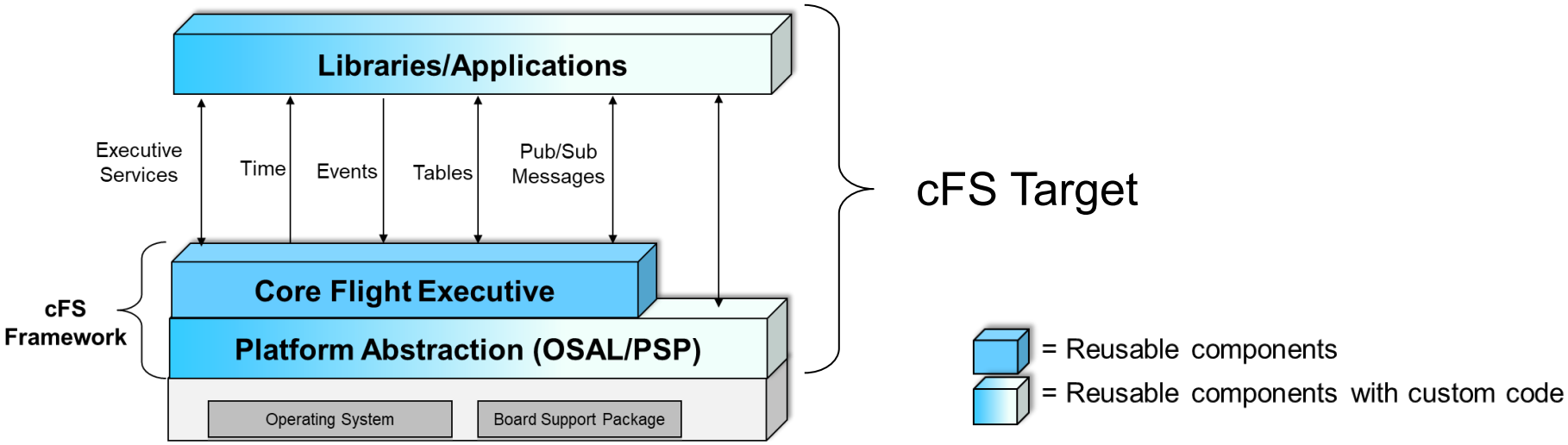
Notes

- 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

Defining and Building cFS Targets

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: ‘Operations Services App Suite’ and Demo App
├--cfe-eds-framework/
│  ├--apps/  . . . . . .  Command Ingest (ci_lab) 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 file that impacts the target build must be described

- 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.

Importing and Creating Apps

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. Basecamp style apps with Basecamp App Specs
  2. NASA Proxy App (includes a Basecamp App Spec)
  3. Apps without a Basecamp App Spec

- **Refer to the Basecamp Application Developers Guide for complete details about Basecamp style apps**

Introduction

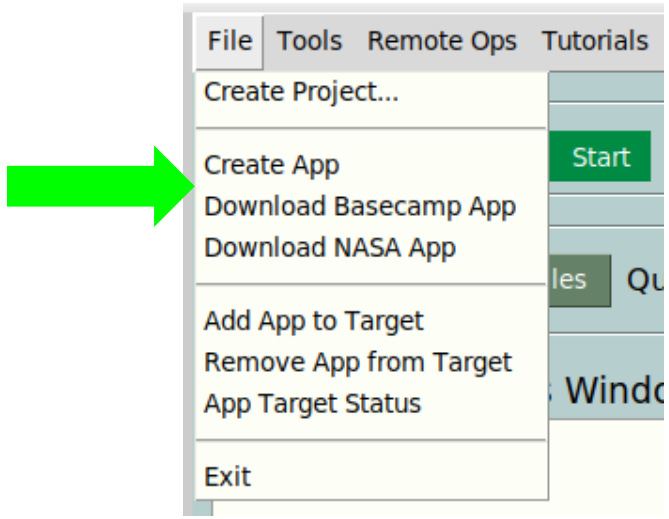
- **When cFS Basecamp is installed the default cFS Target is preconfigured so the only required action is to build the cFS Target from the command line**

- **Basecamp supports adding and removing three classes of user apps from the cFS Target**
  1. Basecamp style apps with Basecamp App Specs
  2. NASA Proxy App (includes a Basecamp App Spec)
  3. Apps without a Basecamp App Spec

- **Refer to the Basecamp Application Developers Guide for complete details about Basecamp style apps**

1 – Basecamp Style Apps (1 of 4)

- Basecamp style apps can be added to the *cfs-basecamp/usr/apps* directory using either the *File->Create App* or *File->Download Basecamp App* tools



- New Basecamp style apps can be creating by starting with an existing app

1 – Basecamp Style Apps (2 of 4)

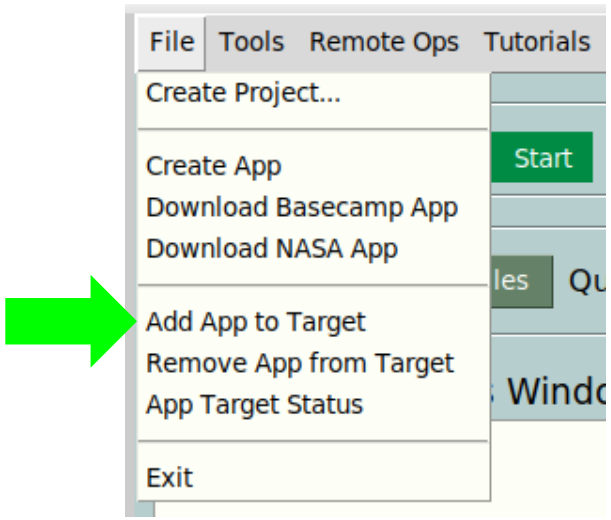
```
cfs-basecamp
├- . . .
└-usr/
  └-apps/
    └-hi_world/
      ├- CMakeLists.txt . . . . . Cmake instructions
      ├- hi_world.json  . . . . . Basecamp App Spec
      ├--eds/
      │ └- hi_world.xml . . . . . EDS command and telemetry specs
      └- fsw/
        ├-/mission_inc/
        │  └- hi_world_mission_cfg.h  . . . 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
```

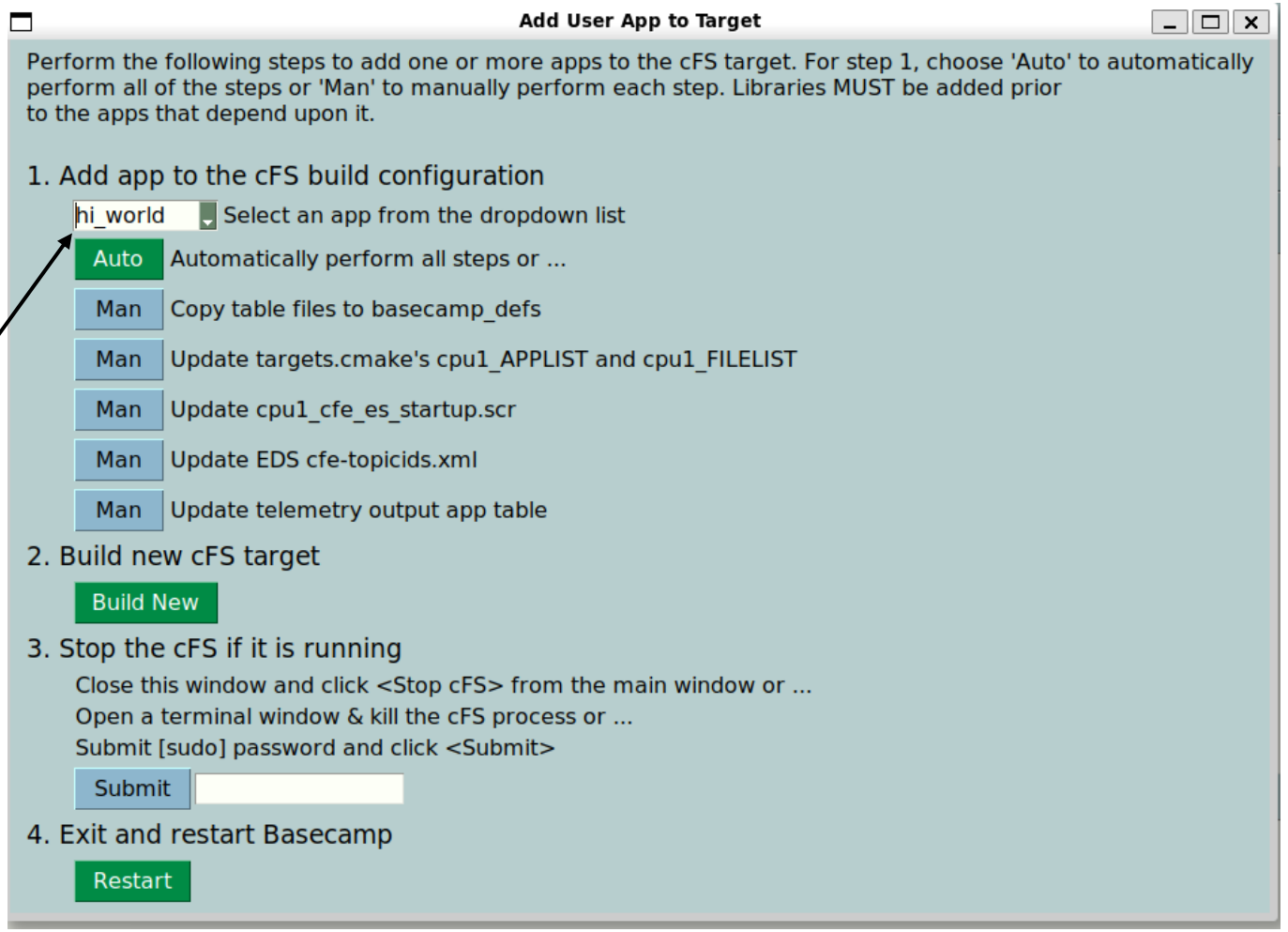
"hi_world" created using ***File->Create App*** tool

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 – Basecamp Style Apps (3 of 4)

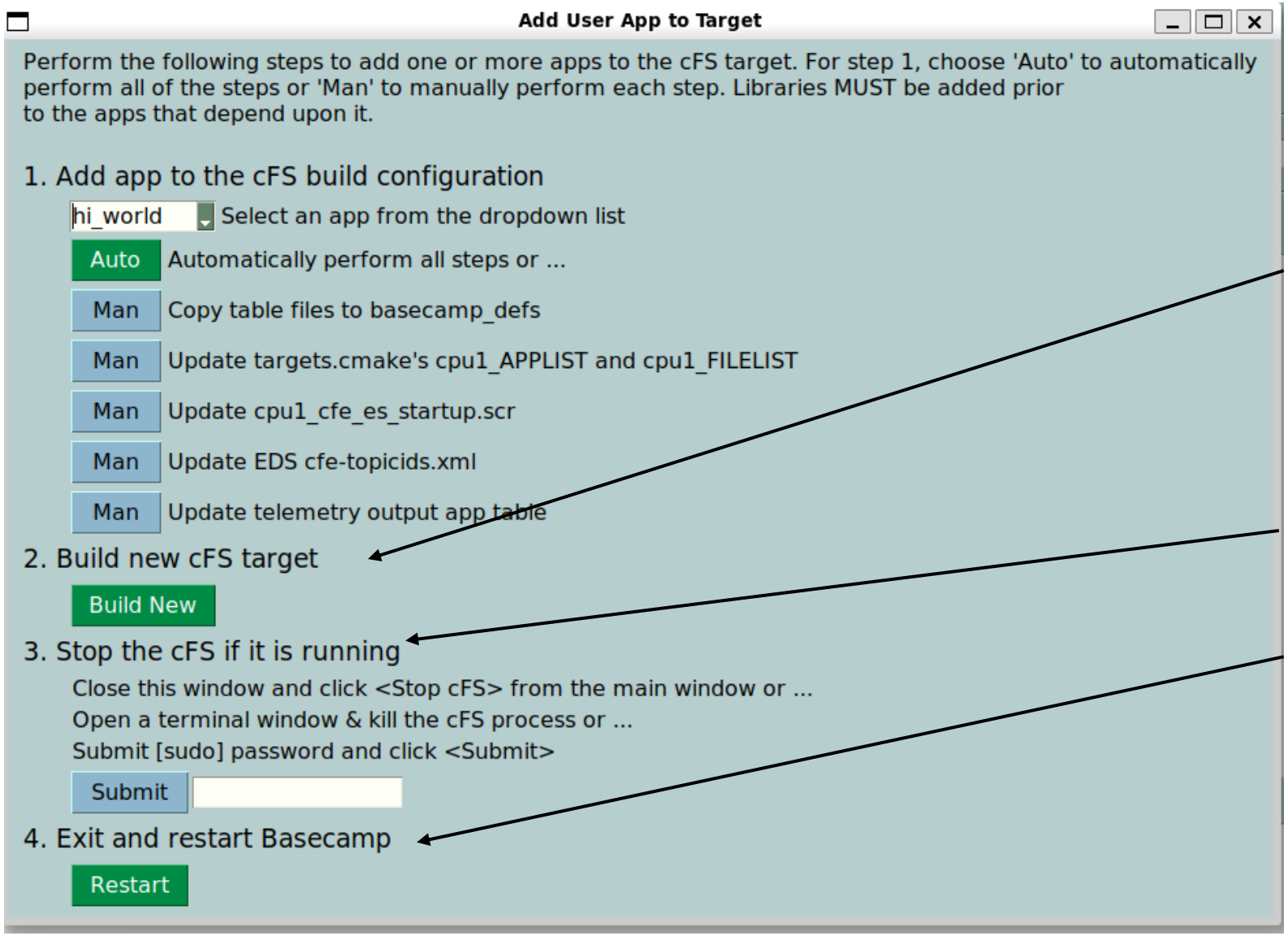
1. Select *File->Add App to Target*





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 – Basecamp Style Apps (4 of 4)



3. **Build the cFS**
   a. This uses the 'make topicids' command option that is described in Lesson 2

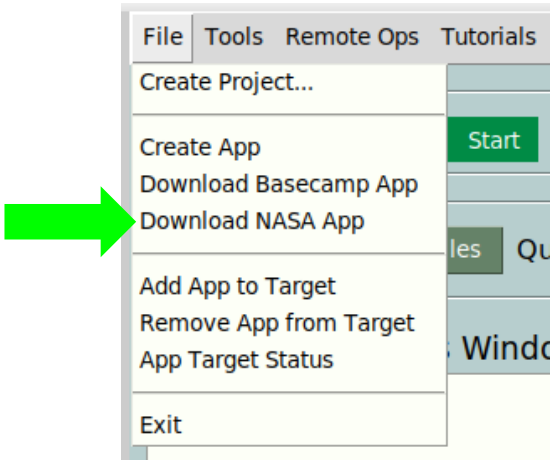
4. **Stop the cFS if it is running**

5. **Restart Basecamp's GUI**
   a. This causes Basecamp to use the new python libraries created by the cFS build process

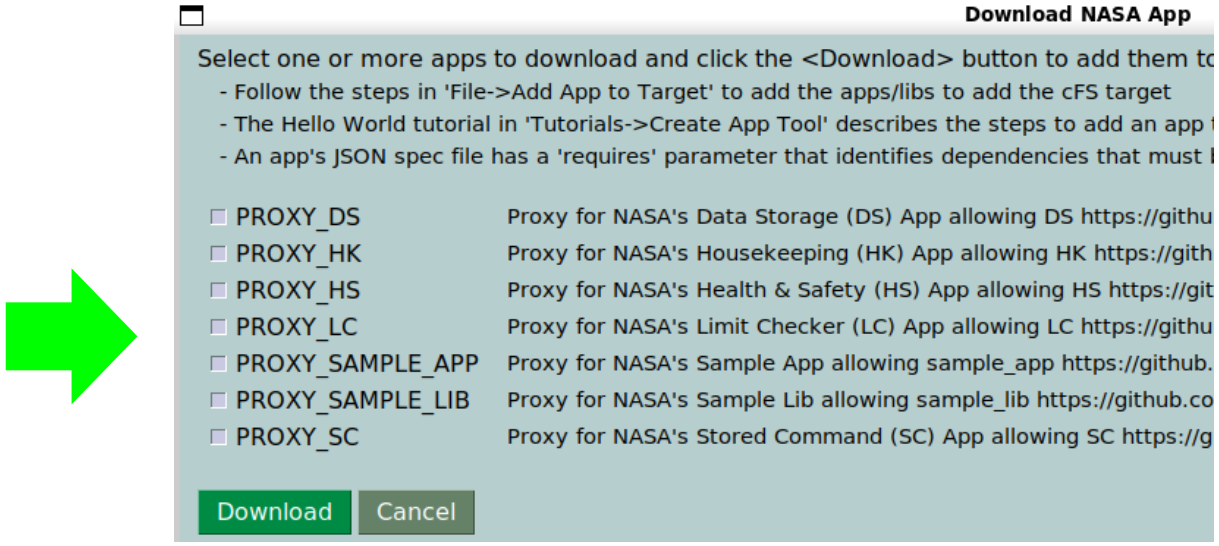
2 – NASA Proxy App (1 of 2)

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

A. Use the *File->Download NASA 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 App to Target* and follow the same steps as a Basecamp style app

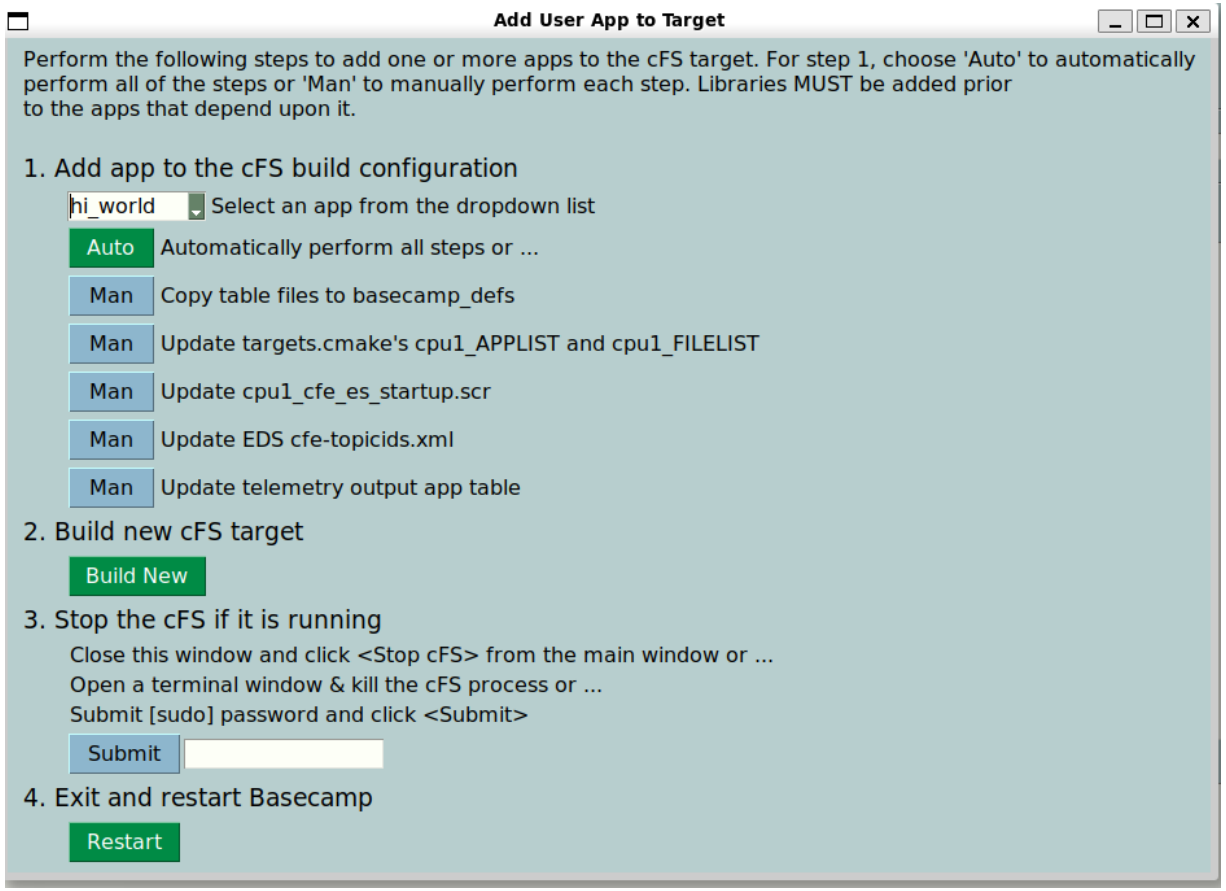
Notes

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)

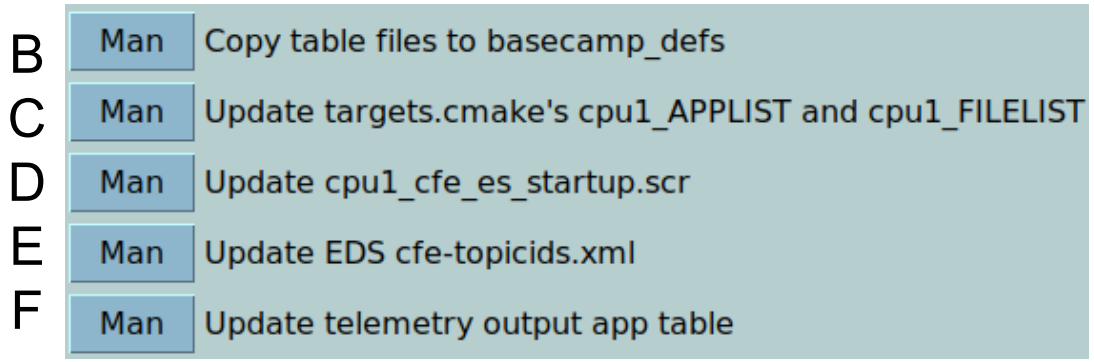
A. Select *File->Add App to Target*

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



***Creating an app spec may be less work than these steps and is the recommended approach*

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



Steps

B. Since there's no app spec then probably not a Basecamp style app and no files need to be copied

C. Update *targets.cmake*[1]

D. Update *cpu1_cfe_es_startup.scr*[1]

E. Update *cfe-topicids.xml*[1]

F. Update *cpu1_kit_so_pkt_tbl.json*[1]

- 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

1. *How to integrate apps is describe in the next section*

Integrating and Building Targets

Integrate Apps into a cFS Target

- 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

- **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

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

```
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-topicids.xml (1 of 2)

Add the app's telecommand Topic ID defined it its EDS spec to cfe-topicids.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-topicids.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-topicids.xml (2 of 2)

Add the app's telemetry Topic IDs defined it its EDS spec to cfe-topicids.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-topicids.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.json[1] app_c_hist_tbl.json[2] …)
```

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")
```

Step 3 – Add App to CFE_ES's Startup Script

```
cfs-basecamp
├-apps/
├-cfe-eds-framework/
│  ├-apps/
│  ├-build/
│  │  ├- exe/cpu1/core-cpu1 . . . . . .
│  │  └- exe/cpu1/cf/*.* . . . . . .
│  ├-cfe/
│  ├-basecamp_defs/
│  │  ├-eds/
│  │  │  └- cfe-topicids.xml . . . . . .
│  │  ├-global_build_options.cmake . .
│  │  ├-mission_build_custom.cmake . .
│  │  ├-targets.cmake . . . . . .
│  │  │
│  │  └-cpu1_cfe_es_startup.scr . . . .
│  ├-libs/
│  ├-Makefile . . . . . .
│  ├-osal/
│  ├-psp/
│  └-tools/
└-usr/
   └-apps/
```

Core Flight Executive (cFE) executable

"Compact Flash (CF)" contains app/lib object files, app default table files

Defines a mission's telecommand and telemetry message Topic ID values

Defines app source search path

Defines compiler options

Defines the target name cpu1, the apps/libs in the cpu1 target, and files to copy from *basecamp_defs/* to *build/exe/cpu1/cf/*

Defines the apps/libs that are loaded when core-cpu1 runs

Controls the build process; GNU-make wrapper that calls the CMake tools

cfe_es_startup.scr

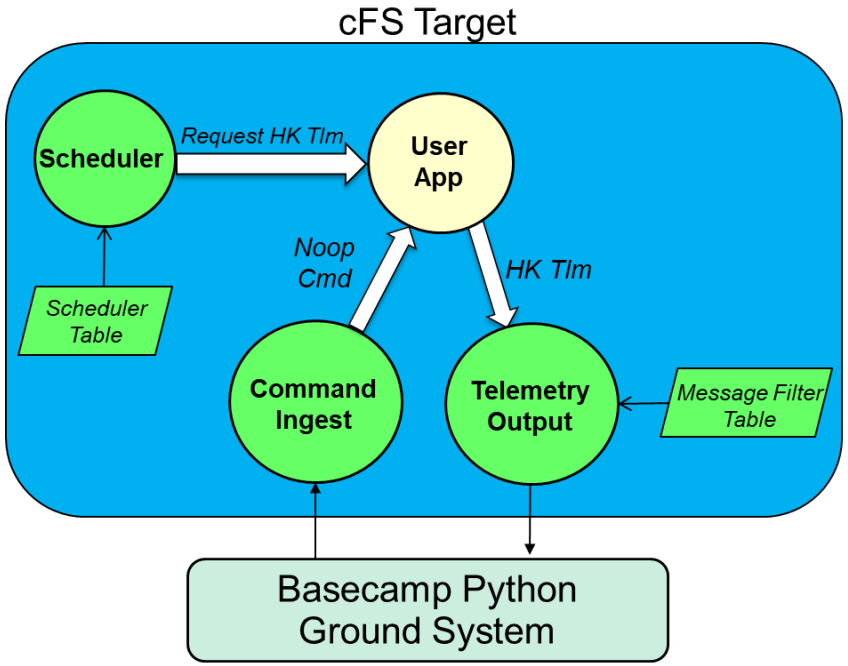
- cfe_es_startup.scr defines which libraries and apps are loaded during initialization
- The filename in basecamp_defs is cpu1_cfe_es_startup.scr to associate it with target cpu1 and the cpu1_ prefix is removed when the file is copied to the build/exe/cf directory

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,

Step 4 – Update *Operations Services App Suite* Tables

```
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/
|  |  ├--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_kit_to_pkt_tbl.json. . . . Defines how often each telemetry message is output
|  ├--libs/
|  ├--Makefile  . . . . . . . . . . . . Controls the build process; GNU-make wrapper that calls the CMake tools
|  ├--osal/
|  ├--psp/
|  └--tools/
└--usr/
   └--apps/
```

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

- **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 <u>not</u> need to be updated**
  - This is the recommended approach

Scheduler App Tables

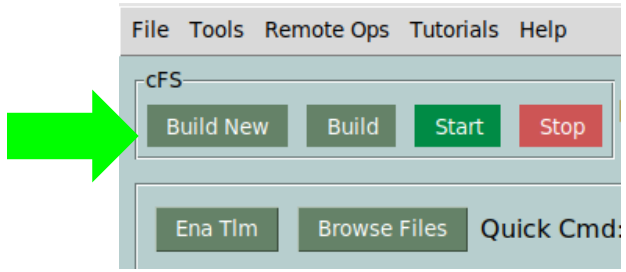
- **The kit scheduler (KIT_SCH) app uses a scheduler table defined in cpu1_kit_sch_schtbl.json to determine when to send messages on the software bus**
- **Basecamp's default scheduler table divides a second into four 250ms slots and each slot can have up to 15 messages sent**
- **The following periodic 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 <u>not</u> 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 build must be performed which can be done from Basecamp's main window or from the command line

A. To build using the GUI 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
```

Target Build Considerations

- **Perform a <*Build New*> when**
  - An app is added or removed from a target
  - Files are added or removed from an app
  - Any changes to topic IDs or to json tables that define topic IDs (from the command line ‘make topicids’ must be run)

- **<*Build*> can be used be used for routine code changes and is much faster than <Build New>**

- **If any app’s EDS definitions are changed that impact command or telemetry messages then the Python GUI must be restarted after the build completes**

- **If a build fails then the python libraries used by the GUI won’t be available**
  - If the GUI is stopped then it can’t be restarted until a successful build is done from the command line