Core Flight System (cFS) Training

Integration with COSMOS

Flight Software Systems Branch, Code 582
Goddard Space Flight Center, Greenbelt, MD
Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA’s STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- **TECHNICAL PUBLICATION.** Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- **TECHNICAL MEMORANDUM.** Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- **CONTRACTOR REPORT.** Scientific and technical findings by NASA-sponsored contractors and grantees.

- **CONFERENCE PUBLICATION.** Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- **SPECIAL PUBLICATION.** Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- **TECHNICAL TRANSLATION.** English-language translations of foreign scientific and technical material pertinent to NASA’s mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- E-mail your question to help@sti.nasa.gov

- Phone the NASA STI Information Desk at 757-864-9658

Write to:

NASA STI Information Desk Mail, Stop 148 NASA Langley Research Center Hampton, VA 23681-2199
Core Flight System (cFS) Training

Integration with COSMOS

Flight Software Systems Branch, Code 582
Goddard Space Flight Center, Greenbelt, MD
Core Flight Executive (cFS) Training

Integration with COSMOS
1. Introduction

2. cFE Services
   a) Executive Services
   b) Software Bus
   c) Event Services
   d) Time Services
   e) Table Services

3. Application Layer
   a) cFS Applications
   b) cFS Libraries

4. [Optional] Integration with COSMOS
cFS and COSMOS

- cFS has been used with several ground systems
  - ASIST
  - ITOS
  - COSMOS
- COSMOS is an open-source ground system solution
  - https://cosmosrb.com/

This module will show how to operate cFS with COSMOS
Module Agenda

• Getting Started
• Defining Commands
• Defining Telemetry
• Creating Telemetry Displays
• Basic Scripting
• Test Runner
Prerequisites
Module Prerequisites

• Have a running cFS build environment that includes the cFS sample_app
  – This is the result of completing Exercise 1 in the main cFS training package

• Have COSMOS installed on development machine
  – Installation instructions here: https://cosmosrb.com/docs/installation/
Part 1 - Setup
To setup the cFS Bundle directly from the latest set of interoperable repositories:

```bash
git clone https://github.com/nasa/cFS.git
cd cFS
git checkout bootes-rc2
git submodule init
git submodule update
```

Copy in the default makefile and definitions:

```bash
cp cfe/cmake/Makefile.sample Makefile
cp -r cfe/cmake/sample_defs sample_defs
```

Subsequent exercises assume that cFS was cloned into the home directory (“~/cFS”)

If running on a standard Linux build as a normal user, allow OSAL “permissive mode” for best effort message queue depth and task priorities.

- Open the sample_defs/default_osconfig.cmake file
- Find the “OSAL_CONFIG_DEBUG_PERMISSIVE_MODE” parameter and set it to TRUE
Part 2 – Build and Run

The cFS Framework, including sample applications, will build and run on the pc-linux platform support package (should run on most Linux distributions), via the steps described in https://github.com/nasa/cFE/tree/master/cmake/README.md. Quick-start is below:

To prep, compile, and run (from cFS directory above):

```bash
make prep
make
make install
cd build/exe/cpu1/
./core-cpu1
```

Should see startup messages and CFE_ES_Main entering OPERATIONAL state. Note the code must be executed from the build/exe/cpu1 directory to find the startup script and shared objects.
### Exercise 0 Recap

#### cFE Version
- cFE Version: v6.7.0-dev292
- cFE Creation Date: 2021-07-14
- Last Official Release: cfe v6.7.0

#### cFE Services Started
- Started Services:
  - CDS Early Init
  - CFE Early Init
  - Event Log Clear
  - CFE SB Early Init
  - CSD Space Packet Protocol v1
  - CFE TIME Early Init
  - CFE_TBL Early Init
  - CFE FS Early Init

#### Evans Port 1
- EVS Port 1/CFE_ES 1: CFE EVS V6.7.0-dev292 (Codename: Bootes), Last Official Release: cfe v6.7.0
- EVS Port 1/CFE_ES 2: cFE Versions: cfe v6.7.0-dev292, osal v5.0.0-dev247, psp v1.4.0-dev76, cfe chksm 7319
- EVS Port 1/CFE_ES 9: Mission bootes-rcl-20 sample, CFE git version: v6.8.0-rcl-1-ge5291a, OSAL git version: v5.1.0-rcl-1-gf7f39f1

#### Evans Port 2
- EVS Port 2/CFE_SB 1: cFE SB V6.7.0-dev292
- EVS Port 2/CFE_SB 14: No subscribers for MsgId 0x808, sender CFE_SB

#### Evans Port 3
- EVS Port 3/CFE_ES 1: cFE ES V6.7.0-dev292
- EVS Port 3/CFE_ES 2: cFE Versions: cfe v6.7.0-dev292, osal v5.0.0-dev247, psp v1.4.0-dev76, cfe chksm 7319
- EVS Port 3/CFE_SB 14: No subscribers for MsgId 0x808, sender CFE_ES

#### Evans Port 4
- EVS Port 4/CFE_SB 14: No subscribers for MsgId 0x808, sender CFE_SB

#### Evans Port 5
- EVS Port 5/CFE_ES 9: Mission bootes-rcl-20 sample, CFE git version: v6.8.0-rcl-1-ge5291a, OSAL git version: v5.1.0-rcl-1-gf7f39f1

#### Evans Port 6
- EVS Port 6/CFE_ES 9: Build 202012072555 ejtimon@gs580s-582cfes

#### Evans Port 7
- EVS Port 7/CFE_SB 14: No subscribers for MsgId 0x808, sender CFE_ES

#### Evans Port 8
- EVS Port 8/CFE_SB 14: No subscribers for MsgId 0x808, sender CFE_ES

#### Evans Port 9
- EVS Port 9/CFE_ES 9: Build 202012072555 ejtimon@gs580s-582cfes

#### Evans Port 10
- EVS Port 10/CFE_SB 14: No subscribers for MsgId 0x808, sender CFE_SB

---

Note: The above log entries are sample data from the cFE system and do not represent the full log entries provided in the document.
Getting Started
Navigating COSMOS

Reference: https://cosmosrb.com/docs/structure/
Terms

• Target – a destination for commands and/or a source of telemetry
  – When communicating with cFS, each cFS app is typically a target
  – Much of COSMOS is organized around targets

• Tool – one of the main “out of the box” components of COSMOS
  – Everything on the “Launcher” screen is a tool
  – Each tool can be configured
  – The “Command and Telemetry Server” must be running in order to use the other tools

• Interface – mechanism by which COSMOS communicates with a given target
Must either auto declare (all targets) or declare each target individually. Typically needs to be modified in a real system.

Connection details. Can often be left unchanged.

Scripts that read and write log files. Can be changed or left as-is.

Default locations of log files. Can often be left unchanged.
Configure the log writer. Note that the ruby script is the same one specified in system.txt

Alternate ways of defining interfaces. Each target must be associated with an interface in order to be used. Interfaces are often customized, though there are some built-in choices.

Optional router specification

Optional background task
Objective: Create a new target for the sample app

Part 1 – Add sample_app to COSMOS

1. Navigate to the config/targets directory in COSMOS
2. Create a directory called “SAMPLE”
3. Enter the “SAMPLE” directory
4. Create a file called “target.txt”
5. Navigate to the directory “cosmosdemo/config/system”
6. Open the file “system.txt” and add the line “DECLARE_TARGET SAMPLE”
   • This tells COSMOS to look for the target you just created
7. Navigate to the directory “cosmosdemo/config/tools/cmd_tlm_server”
8. Open the “cmd_tlm_server.txt” file
9. Under the PACKET LOG WRITER section, create a LOCAL interface, add the line “TARGET sample” under it

```plaintext
INTERFACE LOCAL udp_interface.rb 127.0.0.1 1234 1235 nil nil 128 nil nil
       TARGET sample
```
• For simplicity you can delete everything in the “cmd_tlm_server.txt” file below the LOCAL interface
Part 2 – Launch COSMOS

1. Enter the main Cosmos directory and launch COSMOS with “ruby Launcher”
   • You may need to click “Update Project CRCs” when COSMOS starts up
2. Click on “Command and Telemetry Server” and click “OK” on the dialog that pops up

Successful startup confirms that Part 1 was done correctly.
Exercise 1 - Recap

---

**system.txt**

```
# Declare Targets that make up the system
# DECLARE_TARGET target_name [substitute_name]
# AUTO_DECLARE_TARGETS
# DECLARE_TARGET INST
# DECLARE_TARGET INST INST2
# DECLARE_TARGET EXAMPLE
# DECLARE_TARGET TEMPLATED
# DECLARE_TARGET DART
# DECLARE_TARGET SYSTEM
# DECLARE_TARGET SAMPLE

# Listen Hosts - IP addresses or hostnames to listen on when running the tools
```

---

**cmd_tlm_server.txt**

```
TITLE 'COSMOS Command and Telemetry Server - Demo Configuration'
# PACKET_LOG_WRITER Parameter Notes
# nil: use default log names
# true: logging enabled
# nil: Don't cycle logs based on time
# 2000000000: Create new log after 2 Billion bytes
# nil: Use the default log directory
# false: Log synchronously - more efficient
# PACKET_LOG_WRITER DEFAULT packet_log_writer.rb nil true nil 2000000000 nil false
# PACKET_LOG_WRITER_SYSTEMLOG packet_log_writer.rb system

INTERFACE LOCAL udp_interface.rb 127.0.0.1 1234 1235 nil nil 128 nil nil
TARGET SAMPLE
```
Defining Commands
• A command database defines the commands that can be sent to flight software
• COSMOS uses text-based command databases
• These databases must specify every field in a command (even those that don’t change)
Location of Command Database

- **COSMOS**
  - config
  - lib
  - outputs
  - procedures
  - tools
  - data
    - Ruby scripts for advanced processing
  - system
  - targets
    - cmd_tlm
  - tools
    - COSMOS tools
    - Log files from COSMOS runs
Relationship with cFS

- The COSMOS command database generally relies on the following files in a cFS app:
  - XX_msg.h
  - XX_msgids.h
  - XX_msgdefs.h

- Each command message structure defined in XX_msg.h should be defined in the command database.

- The XX_msgids.h and XX_msgdefs.h files are used to find arguments to commands:
  - MsgID, Command Code, etc.
Defining a Command Database

- The command database file resides in the cmd_tlm folder under the target
- In the file, each command starts with a COMMAND tag
  COMMAND <Target> <Command Name> <Endianness> <Description>
- Under the COMMAND tag, each parameter is appended to the command
  - Using APPEND_ID_PARAMETER or APPEND_PARAMETER

Reference: https://cosmosrb.com/docs/command/
Review: cFE Software Bus Messages

- By default Consultative Committee for Space Data Systems (CCSDS) packets used to implement messages
- CCSDS Primary Header (Always big endian)

- CCSDS Command Packets
  - Secondary packet header contains a command function code
- CCSDS Telemetry Packets
  - Secondary packet header contains a time stamp of when the data was produced

```c
typedef struct{
    CCSDS_PriHdr_t     Pri;
    CCSDS_CmdSecHdr_t  Sec;
} CFE_SB_CmdHdr_t;

typedef struct{
    CCSDS_PriHdr_t     Pri;
    CCSDS_TlmSecHdr_t  Sec;
} CFE_SB_TlmHdr_t;
```
No-Op Example

- **Sample_app No-Op Command in cFS:**
  
  ```c
  typedef struct
  {
      uint8    CmdHeader[CFE_SB_CMD_HDR_SIZE];
  } SAMPLE_NoArgsCmd_t;
  ```

- **Sample_app No-Op Command in COSMOS:**
  
  ```
  COMMAND SAMPLE SAMPLE_NOOP BIG_ENDIAN "Sample_app NOOP Command"
  APPEND_ID_PARAMETER STREAM_ID 16 UINT 0x1882 0x1882 0x1882 ""
  APPEND_PARAMETER  SEQUENCE 16 UINT 0xC000 MAX_UINT16 0xC000 ""
  FORMAT_STRING "0x%04X"
  APPEND_PARAMETER  PKT_LEN 16 UINT 0x0001 0x0001 0x0001 ""
  FORMAT_STRING "0x%04X"
  APPEND_PARAMETER  CMD_ID   8  UINT 0          0          0         ""
  APPEND_PARAMETER  CHECKSUM 8  UINT  MIN_UINT8 MAX_UINT8 MIN_UINT8 ""
  ```

- **cFS Primary Header**
  - cFS CMD
  - Secondary Header

- **cFS CMD**
  - SAMPLE_APP_CMD_MID
    from sample_app_msgids.h

- **Sample_APP_NOOP_CC**
  from sample_app_msg.h
Other Command Parameters

- Command parameters can have types INT, UINT, FLOAT, DERIVED, STRING, BLOCK
- Parameter ranges are specified with Minimum, Maximum, and Default values
- For numbers, FORMAT_STRING specifies the input format of the number
  - Ex. "0x%04X" specifies input in hexadecimal
- Parameters can also be selected from a drop down list using the STATE tag
  
  ```
  APPEND_PARAMETER ENABLE 32 UINT 0 1 0 "Enable setting"
  
  STATE FALSE 0
  STATE TRUE 1
  ```
Exercise 2 – Part 1

Objective: Create a command database for sample_app and send commands to cFS (2 parts total)

Part 1 – Add sample_app to COSMOS

1. Navigate to the config/targets/SAMPLE directory in COSMOS
2. Inside “target.txt”, add the following line: COMMANDS sample_cmds.txt
   - Note: This file tells COSMOS the name of the file containing the command database
3. Inside the “SAMPLE” directory, create a “cmd_tlm” directory
4. Inside the “cmd_tlm” directory, create a file “sample_cmds.txt”
5. Open the file sample_cmds.txt
6. Create a command definition for each command in sample_app_msg.h
   - You should have a total of 3 commands
   - They will be similar to the No-Op command example
   - sample_app_msg.h is located in the cFS tree at apps/sample_app/fsw/src/sample_app_msg.h
Part 2 – Send commands to sample_app

1. In a different terminal window, start the cFS
   1. Leave this running, but put the window aside and return to the terminal window with COSMOS
2. Enter the main Cosmos directory and launch COSMOS with “ruby Launcher”
   1. You may need to click “Update Project CRCs” when COSMOS starts up
3. Click on “Command and Telemetry Server” and click “OK” on the dialog that pops up
4. Click on the “Cmd Packets” tab and scroll down until you see the target SAMPLE on the left
5. Click on the “View in Command Sender” button beside “SAMPLE_NOOP”
6. Click “Send” on the Command Sender window
   1. A no-op event message should show up in the cFS terminal window
The sample_cmds.txt file:

COMMAND SAMPLE SAMPLE_NOOP BIG_ENDIAN "Sample_app NOOP Command"

APPEND_ID PARAMETER STREAM_ID 16 UINT 0x1882 0x1882 0x1882 ""
APPEND PARAMETER SEQUENCE 16 UINT 0xC000 MAX_UINT16 0xC000 ""
FORMAT STRING "0x%04X"
APPEND PARAMETER PKT_LEN 16 UINT 0x0001 0x0001 0x0001 ""
FORMAT STRING "0x%04X"
APPEND PARAMETER CMD_ID 8 UINT 0 0 0 ""
APPEND PARAMETER CHECKSUM 8 UINT MIN_UINT8 MAX_UINT8 MIN_UINT8 ""

COMMAND SAMPLE SAMPLE_RESET BIG_ENDIAN "Sample_app Reset Counters Command"

APPEND_ID PARAMETER STREAM_ID 16 UINT 0x1882 0x1882 0x1882 ""
APPEND PARAMETER SEQUENCE 16 UINT 0xC000 MAX_UINT16 0xC000 ""
FORMAT STRING "0x%04X"
APPEND PARAMETER PKT_LEN 16 UINT 0x0001 0x0001 0x0001 ""
FORMAT STRING "0x%04X"
APPEND PARAMETER CMD_ID 8 UINT 1 1 1 ""
APPEND PARAMETER CHECKSUM 8 UINT MIN_UINT8 MAX_UINT8 MIN_UINT8 ""

COMMAND SAMPLE SAMPLE_PROCESS BIG_ENDIAN "Sample_app Process Command"

APPEND_ID PARAMETER STREAM_ID 16 UINT 0x1882 0x1882 0x1882 ""
APPEND PARAMETER SEQUENCE 16 UINT 0xC000 MAX_UINT16 0xC000 ""
FORMAT STRING "0x%04X"
APPEND PARAMETER PKT_LEN 16 UINT 0x0001 0x0001 0x0001 ""
FORMAT STRING "0x%04X"
APPEND PARAMETER CMD_ID 8 UINT 2 2 2 ""
APPEND PARAMETER CHECKSUM 8 UINT MIN_UINT8 MAX_UINT8 MIN_UINT8 ""
Exercise 2 - Recap
Exercise 2 - Recap

1980-012-14:03:20.25410 ES Startup: CI_LAB_APP loaded and created
1980-012-14:03:20.25427 CI_LAB listening on UDP port: 1234
EVS Port1 42/1/CI_LAB_APP 3: CI Lab Initialized. Version 2.3.5.0
1980-012-14:03:20.25450 ES Startup: Loading file: /cf/to_lab.so, APP: TO_LAB_APP
1980-012-14:03:20.25461 ES Startup: TO_LAB_APP loaded and created
1980-012-14:03:20.25525 ES Startup: SCH_LAB_APP loaded and created
SCH Lab Initialized. Version 2.3.7.0
EVS Port1 42/1/CFE_SB 7: Duplicate Subscription,MsgId 0x0 on TO_LAB_TLMPIPE pipe,app TO_LAB_APP
EVS Port1 42/1/CFE_SB 7: Duplicate Subscription,MsgId 0x0 on TO_LAB_TLMPIPE pipe,app TO_LAB_APP
EVS Port1 42/1/CFE_SB 7: Duplicate Subscription,MsgId 0x0 on TO_LAB_TLMPIPE pipe,app TO_LAB_APP
EVS Port1 42/1/CFE_SB 7: Duplicate Subscription,MsgId 0x0 on TO_LAB_TLMPIPE pipe,app TO_LAB_APP
EVS Port1 42/1/CFE_SB 7: Duplicate Subscription,MsgId 0x0 on TO_LAB_TLMPIPE pipe,app TO_LAB_APP
EVS Port1 42/1/CFE_SB 7: Duplicate Subscription,MsgId 0x0 on TO_LAB_TLMPIPE pipe,app TO_LAB_APP
EVS Port1 42/1/TO_LAB_APP 1: TO Lab Initialized. Version 2.3.7.0 Awaiting enable command.
1980-012-14:03:20.30535 ES Startup: CFE_ES_Main entering APPS_INIT state
1980-012-14:03:20.30537 ES Startup: CFE_ES_Main entering OPERATIONAL state
EVS Port1 42/1/CFE_TIME 21: Stop FLYWHEEL
EVS Port1 42/1/SAMPLE_APP 3: SAMPLE: NOOP command Version 1.1.11.0
Defining Telemetry
Telemetry Databases

- A telemetry database defines the telemetry that can be received from the flight software.
- COSMOS uses text-based telemetry databases.
- These databases must specify every field in a telemetry packet.
- The database must also tell COSMOS how to identify the packet.
• The COSMOS telemetry database generally relies on the following files in a cFS app:
  – XX_msg.h
  – XX_msgids.h
• Each telemetry message structure defined in XX_msg.h should be defined in the telemetry database
• The XX_msgids.h file is used to find the message ID
Defining a Telemetry Database

• The telemetry database file resides in the cmd_tlm folder under the target

• In the file, each command starts with a TELEMETRY tag

  TELEMETRY <Target> <Packet Name> <Endianness> <Description>

• Under the TELEMETRY tag, each telemetry item is appended to the packet
  – Using APPEND_ID_ITEM or APPEND_ITEM

Reference: https://cosmosrb.com/docs/telemetry/
**HK Example**

**Sample_app housekeeping telemetry in cFS:**

```c
typedef struct {
    uint8 TlmHeader[CFE_SB_TLM_HDR_SIZE];
    SAMPLE_HkTlm_Payload_t Payload;
} OS_PACK SAMPLE_HkTlm_t;
```

**Sample_app housekeeping packet in COSMOS:**

```plaintext
TELEMETRY SAMPLE SAMPLE_HK BIG_ENDIAN "Sample_app housekeeping telemetry"
APPEND_ID_ITEM STREAM_ID 16 UINT 0x0883 ""
APPEND_ITEM   SEQUENCE 16 UINT ""
APPEND_ITEM   PKT_LEN 16 UINT ""
APPEND_ITEM   SECONDS 32 UINT ""
APPEND_ITEM   SUBSECS 16 UINT ""
APPEND_ITEM   CMD_CNT 8  UINT "Command Counter"
APPEND_ITEM   CMD_ERRS 8  UINT "Command Error Count"
APPEND_ITEM   SPARE 16 UINT "Spares"
```

`SAMPLE_APP_HK_TLM_MID` from `sample_app_msgids.h`
Objective: Create a telemetry database for sample_app and get telemetry from cFS (4 parts)

Part 1 – Add the sample_app HK packet

1. Navigate to the config/targets/sample directory in COSMOS
2. Open the target.txt file and add the line “TELEMETRY sample_tlm.txt”
3. Inside the “cmd_tlm” directory, create a file “sample_tlm.txt”
4. Add the definition for the sample_app housekeeping packet
Part 2 – Add a target for TO_Lab

*This is necessary because we need to enable telemetry in cFS before we will see it in COSMOS.*

1. Navigate to the config/targets directory in COSMOS
2. Create a directory called “TO_LAB”
3. Enter the “TO_LAB” directory
4. Create a file called “target.txt”
5. Inside “target.txt”, add the following line: COMMANDS to_lab_cmds.txt
6. Inside the “TO_LAB” directory, create a “cmd_tlm” directory
7. Inside the “cmd_tlm” directory, create a file “to_lab_cmds.txt”
8. Navigate to the directory “cosmosdemo/config/tools/cmd_tlm_server”
9. Open the “cmd_tlm_server.txt” file
10. Under the LOCAL interface, add the line “TARGET TO_LAB”
11. Navigate to the directory “cosmosdemo/config/system”
12. Open the “system.txt” file and add “DECLARE_TARGET TO_LAB”
Part 3 – Add a command database for to_lab

1. Navigate back to the “config/targets/to_lab/cmd_tlm” directory in COSMOS
2. Open the file “to_lab_cmds.txt”
3. Create a command definition for the TO_LAB_EnableOutput_t command
   - This definition is located in to_lab_msg.h (located in the cFS directory under apps/to_lab/fsw/src/)
Part 4 – Send commands/receive telemetry from cFS

1. Enter the main Cosmos directory and launch COSMOS with “ruby Launcher”
   1. You may need to click “Update Project CRCs” when COSMOS starts up
2. On the Launcher window, click on “Command and Telemetry Server” and click “OK” on the dialog that pops up
3. On the Launcher window, click on “Command Sender”
4. In the drop-down list beside target, select TO_LAB. The command field should automatically update to “TO_LAB_ENABLE”. Click Send.
   1. An event message should appear in the cFS window
   2. On the “Tlm Packets” tab of the “Command and Telemetry Server” window, the count of “SAMPLE_HK packets should be incrementing.
5. Click on the “View in Packet Viewer” button beside “SAMPLE_HK”
   1. At this point, the “CMD_CNT” field should be “1” if you are still running the same cFS instance as in Exercise 1, or 0 if you restarted cFS
6. Send a SAMPLE_APP NOOP command as in Exercise 1
   1. The “CMD_CNT” field in the packet viewer should increment by 1
Exercise 3 - Recap

The sample_tlm.txt file:

```
TELEMETRY SAMPLE SAMPLE_HK BIG_ENDIAN "Sample_app housekeeping telemetry"
APPEND_ID_ITEM STREAM_ID 16 UINT 0x0883 ""
    FORMAT_STRING "0x%04X"
APPEND_ITEM SEQUENCE 16 UINT ""
    FORMAT_STRING "0x%04X"
APPEND_ITEM PKT_LEN 16 UINT ""
APPEND_ITEM SECONDS 32 UINT ""
APPEND_ITEM SUBSECS 16 UINT ""
APPEND_ITEM CMD_ERRS 8 UINT "Command Counter"
APPEND_ITEM CMD_CNT 8 UINT "Command Error Count"
APPEND_ITEM SPARE 16 UINT "Spares"
```
The to_lab_cmds.txt file:

```
COMMAND TO_LAB TO_LAB_ENABLE BIG_ENDIAN "TO_Lab enable telemetry"

APPEND_ID_PARAMETER STREAM_ID 16 UINT 0x1880 0x1880 0x1880 ""
APPEND_PARAMETER SEQUENCE 16 UINT 0xC000 MAX_UINT16 0xC000 ""
   FORMAT_STRING "0x%04X"
APPEND_PARAMETER PKT_LEN 16 UINT 0x0001 0x0001 0x0001 ""
   FORMAT_STRING "0x%04X"
APPEND_PARAMETER CMD_ID 8 UINT 6 6 6 ""
APPEND_PARAMETER CHECKSUM 8 UINT MIN_UINT8 MAX_UINT8 MIN_UINT8 ""
APPEND_PARAMETER DEST_IP 128 STRING "127.0.0.1" "Destination IP"
```
### Exercise 3 - Recap

#### COSMOS Command and Telemetry Server - Demo Configuration

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>Targets</th>
<th>Cmd Packets</th>
<th>Tlm Packets</th>
<th>View Raw</th>
<th>View in Packet Viewer</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST2</td>
<td>IMAGE</td>
<td>0</td>
<td>View Raw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INST2</td>
<td>MECH</td>
<td>0</td>
<td>View Raw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INST2</td>
<td>PARAMS</td>
<td>0</td>
<td>View Raw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAMPLE</td>
<td>SAMPLE_HK</td>
<td>22</td>
<td>View Raw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM</td>
<td>LIMITS_CHANGE</td>
<td>0</td>
<td>View Raw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM</td>
<td>META</td>
<td>0</td>
<td>View Raw</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**2020/07/24 14:24:54.646** ERROR: LOCAL - Unknown 36 byte packet starting: 0884C03E01D9E4F0F018000000000000000000000000000000000

**2020/07/24 14:24:54.145** ERROR: LOCAL - Unknown 16 byte packet starting: 0880C03E0009E4F0F0B72D0100000000

#### Packet Viewer: Formatted Telemetry with Units

**Target:** SAMPLE

**Description:** Sample_app housekeeping telemetry

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>PACKET_TIMESERIES</em></td>
<td>1595619115.75321</td>
</tr>
<tr>
<td><em>PACKET_TIMEFORMATTED</em></td>
<td>2020/07/24 15:31:55.753</td>
</tr>
<tr>
<td><em>PACKET_TIMEFORMATTED</em></td>
<td>2020/07/24 15:31:55.753</td>
</tr>
<tr>
<td><em>RECEIVED_TIMESERIES</em></td>
<td>1595619115.75321</td>
</tr>
<tr>
<td><em>RECEIVED_TIMEFORMATTED</em></td>
<td>2020/07/24 15:31:55.753</td>
</tr>
<tr>
<td><em>RECEIVED_COUNT</em></td>
<td>3</td>
</tr>
<tr>
<td>STREAM_ID:</td>
<td>0x08B3</td>
</tr>
<tr>
<td>SEQUENCE:</td>
<td>0x02A2</td>
</tr>
<tr>
<td>PKT_LEN:</td>
<td>9</td>
</tr>
<tr>
<td>SECONDS:</td>
<td>2655522560</td>
</tr>
<tr>
<td>SUBSECS:</td>
<td>2304</td>
</tr>
<tr>
<td>CMD_CNT:</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Command Sender

**Target:** TO_LAB

**Command:** TO_LAB_ENABLE

**Description:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value or State</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKT_LEN:</td>
<td>0x0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMD_ID:</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHECKSUM:</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEST_IP:</td>
<td>'127.0.0.1'</td>
<td></td>
<td>Destination IP</td>
</tr>
</tbody>
</table>
Exercise 3 - Recap

```plaintext
SAMPLE Lib Initialized. Version 1.1.4.0
1980-012-14:11:03 25216 ES Startup: SAMPLE_APP loaded and created
1980-012-14:11:03 25248 ES Startup: Loading file: /cf/cli_lab.so, APP: CI_LAB_APP
1980-012-14:11:03 25256 ES Startup: CI_LAB_APP loaded and created
1980-012-14:11:03 25288 ES Startup: Loading file: /cf/to_lab.so, APP: TO_LAB_APP
1980-012-14:11:03 25295 ES Startup: TO_LAB_APP loaded and created
1980-012-14:11:03 25370 ES Startup: Loading file: /cf/sch_lab.so, APP: SCH_LAB_APP
1980-012-14:11:03 25440 ES Startup: SCH_LAB_APP loaded and created
EVS Port 1 42/1/SAMPLE_APP 1: SAMPLE App Initialized. Version 1.1.11.0
1980-012-14:11:03 25571 CI_LAB listening on UDP port: 1234
EVS Port 1 42/1/CI_LAB_APP 3: CI Lab Initialized. Version 2.3.5.0
EVS Port 1 42/1/CFE_SB 7: Duplicate Subscription, MsgId 0x0 on TO_LAB_TLM_PIPE pipe, app TO_LAB_APP
EVS Port 1 42/1/CFE_SB 7: Duplicate Subscription, MsgId 0x0 on TO_LAB_TLM_PIPE pipe, app TO_LAB_APP
EVS Port 1 42/1/CFE_SB 7: Duplicate Subscription, MsgId 0x0 on TO_LAB_TLM_PIPE pipe, app TO_LAB_APP
EVS Port 1 42/1/CFE_SB 7: Duplicate Subscription, MsgId 0x0 on TO_LAB_TLM_PIPE pipe, app TO_LAB_APP
EVS Port 1 42/1/TO_LAB_APP 1: TO Lab Initialized. Version 2.3.7.0 Awaiting enable command.
SCH Lab Initialized. Version 2.3.7.0
1980-012-14:11:03 30502 ES Startup: CFE_ES_Main entering APPS_INIT state
1980-012-14:11:03 30504 ES Startup: CFE_ES_Main entering OPERATIONAL state
EVS Port 1 42/1/CFE TIME 21: Stop FLYWHEEL
EVS Port 1 42/1/TO_LAB_APP 3: TO telemetry output enabled for IP
EVS Port 1 42/1/SAMPLE_APP 3: SAMPLE: NOOP command. Version 1.1.11.0
```
Creating Telemetry Displays
Telemetry Displays

- Provides a way to create custom telemetry displays that can be used in place of the packet viewer
- Can display values in “human readable” format
- Can display all data types
- By default, will automatically show staleness by graying out fields
- Provides a number of different layout tags for formatting pages
- Built-in widgets allow custom limit highlighting, graphing, and trending
- Interactive widgets can be tied to ruby scripts to initiate actions

Reference: https://cosmosrb.com/docs/screens/
Location of Telemetry Displays

COSMOS

- config
  - data
  - system
  - targets
  - tools
- lib
  - Ruby scripts for advanced processing
- outputs
  - Log files from COSMOS runs
- procedures
  - Ruby scripts for sending commands
- tools
  - COSMOS tools

XX screens
Exercise 4

Objective: Create a telemetry display for sample_app

1. Navigate to the config/targets/SAMPLE directory in COSMOS
2. Create a directory called “screens”
3. Inside the “screens” directory, create a file “sample_screen.txt”
4. Develop a screen that displays the sample_app housekeeping packet
   • Try to experiment with different layouts
   • Try changing the background color of the screen
5. Enter the main Cosmos directory and launch COSMOS with “ruby Launcher”
   1. You may need to click “Update Project CRCs” when COSMOS starts up
6. On the Launcher window, click on “Command and Telemetry Server” and click “OK” on the dialog that pops up
7. On the Launcher window, click on “Telemetry Viewer”
8. Click on “Show Screen” beside “SAMPLE_SCREEN”
Exercise 4 - Recap

```
SCREEN AUTO AUTO 0.5

VERTICAL

TITLE "CCSDS HEADER"
SETTING BACKCOLOR 54 95 98

HORIZONTAL

VERTICALBOX
    SECTIONHEADER "Primary Header"

    LABELFORMATVALUE SAMPLE SAMPLE_HK STREAM_ID "0x%04X" FORMATTED
    LABELFORMATVALUE SAMPLE SAMPLE_HK SEQUENCE "0x%04X" FORMATTED
    LABELFORMATVALUE SAMPLE SAMPLE_HK PKT_LEN "%d"

END

VERTICALBOX
    SECTIONHEADER "Secondary Header"

    LABELFORMATVALUE SAMPLE SAMPLE_HK SECONDS "%d"
    LABELFORMATVALUE SAMPLE SAMPLE_HK SUBSECS "%d"

END

END

TITLE "Data"
SETTING BACKCOLOR 54 95 98

VERTICALBOX
    LABELFORMATVALUE SAMPLE SAMPLE_HK CMD_CNT "%d"
    LABELFORMATVALUE SAMPLE SAMPLE_HK CMD_ERRS "%d"

END

END
```

cFS Training- Page 51
Exercise 4 - Recap

Telemetry Viewer

CCSDS HEADER
- Primary Header
  - STREAM_ID: 0x0883
  - SEQUENCE: 0xC190
  - PKT_LEN: 9

Secondary Header
- SECONDS: 911085312
- SUBSECS: 4096

Data
- CMD_CNT: 1
- CMD_ERRS: 0
Basic Scripting
COSMOS Scripts

- COSMOS provides the ability to develop Ruby scripts
- Ruby scripts can reference any defined commands and telemetry
- This is useful for testing and repeated onboard operations

Reference: https://cosmosrb.com/docs/scripting/
Location of Scripts

COSMOS

config

lib

outputs

procedures

tools

data

system

targets

tools

Ruby scripts for advanced processing

Log files from COSMOS runs

Ruby scripts for sending commands

COSMOS tools
Exercise 5

Objective: Write and execute a simple script for the sample_app

1. Navigate to the cosmos/procedures directory in COSMOS
2. Create a file called sample_script.rb
3. Write a script that sends a sample_app no-op command and receives the housekeeping telemetry
4. Enter the main Cosmos directory and launch COSMOS with “ruby Launcher”
5. On the Launcher window, click on “Command and Telemetry Server” and click “OK” on the dialog that pops up
6. On the Launcher window, click on “Script Runner”
7. Click “File” → “Open”, navigate to the procedures directory and then select “sample_script.rb”
8. When the script loads, click “Start” on the script runner window
Exercise 5 - Recap
Exercise 5 - Recap

```ruby
prompt("Sending enable telemetry")
cmd("TO_LAB TO_LAB_ENABLE with DEST_IP 127.0.0.1")

value = combo_box("Ready to send a No·Op?", 'Yes', 'No')
case value
when 'Yes'
  cmd("SAMPLE", "SAMPLE_NOOP")
tval = tlm("SAMPLE SAMPLE_HK CMD_CNT")
wait (2)
prompt("# commands " + tval.to_s)
when 'No'
alert("Exiting")
end
```
Test Runner
• The COSMOS test runner builds on the scripting capability to create organized, repeatable test suites
  – Useful for functional, system, and regression testing
• Provides a pass/fail test summary and detailed test logs
• Tests can be organized and run by “test case”, “test group”, and “test suite”
  – A Test Case is a single test.
  – A Test Group is a collection of related tests cases.
  – A Test Suite is a collection of test cases and/or test groups.
Location of Test Runner

COSMOS

- config
  - data
  - system
  - targets
  - tools

- lib
  - Ruby scripts for advanced processing

- outputs
  - Log files from COSMOS runs

- procedures
  - Ruby scripts for sending commands

- tools
  - COSMOS tools
Exercise 6

Objective: Write a simple test for the sample_app

This will adapt the simple Ruby script from Exercise 4 into a COSMOS test suite and test case

1. Navigate to the cosmos/procedures directory in COSMOS
2. Create a file called sample_test.rb
3. Write a test that sends a sample_app no-op command, receives the housekeeping telemetry, and verifies that the sample_app command counter incremented by 1
   • This script should have a test suite and a test case
   • Look at example_text.rb for an example of syntax
4. Open the file “config/tools/test_runner/test_runner.txt” and add the line “LOAD_UTILITY ‘sample_test’”
5. Enter the main Cosmos directory and launch COSMOS with “ruby Launcher”
6. On the Launcher window, click on “Command and Telemetry Server” and click “OK” on the dialog that pops up
7. On the Launcher window, click on “Test Runner”
8. Select “SampleTestSuite” from the drop down list beside “Test Suite”
   • The name might be different depending on your exact test script
   • The “Test Group” and “Test case” fields should auto-populate

9. Click on “Start” next to the Test Suite

10. Optionally update the “OPERATOR_NAME” field in the dialog that appears and click “Start Test”
    • At this point the test should run, showing the real-time execution of the script in the Test Runner window
    • At the end of the test, a “Results” window will appear with a summary of the tests that passed and failed
    • A detailed output from the script can be found in the “Script Output” panel at the bottom of the Test Runner window
Exercise 6 - Recap

test_runner.txt

```
# eejmmon@gs580s-582cfs: ~/cosmosdemo

1 LOAD Utility 'example_test'
2 LOAD Utility 'sample_test'
3 LINE_DELAY 0
4 # RESULTS_WRITER 'results_writer.rb'
5 ALLOW_DEBUG
6 PAUSE_ON_ERROR TRUE
7 CONTINUE_TEST_CASE_AFTER_ERROR TRUE
8 ABORT_TESTING_AFTER_ERROR FALSE
9 MANUAL TRUE
10 LOOP_TESTING FALSE
11 BREAK_LOOP_AFTER_ERROR FALSE
12 CREATE_DATA_PACKAGE
13 AUTO_CYCLE_LOGS
14 COLLECT_METADATA
15 # DISABLE_TEST_SUITE_START
16 # DISABLE_TEST_GROUP_START
17 # IGNORE_TEST_SUITE EmptyTestSuite
18 # IGNORE_TEST EmptyTest
19 # MONITOR_LIMITS
20 # PAUSE_ON_RED

"config/tools/test_runner/test_runner.txt" 20L, 463C 20,1 All
```
Exercise 6 - Recap

sample_test.txt

```ruby
load 'cosmos/tools/test_runner/test.rb'
class SampleCmdTest < Cosmos::Test
  def initialize
    super()
  end

  def test_sample_noop
    puts "Running #{Cosmos::Test.current_test_suite} #{Cosmos::Test.current_test} #{Cosmos::Test.current_test_case}"
    cmd("TO_LAB TO_LAB_ENABLE with DEST_IP 127.0.0.1")
    puts "Getting initial command count"
    wait(2)
    initcnt = tln("SAMPLE SAMPLE_HK CMD_CNT")
    wait(1)
    puts "Sending no-op command"
    cmd("SAMPLE SAMPLE_NDOP")
    wait(1)
    expcnt = initcnt + 1
    wait_check("SAMPLE SAMPLE_HK CMD_CNT == #\{expcnt\}", 5)
  end
end

class SampleTestSuite < Cosmos::TestSuite
  def initialize
    super()
    add_test('SampleCmdTest')
  end
end
```
Exercise 6 – Recap

Test Runner Interface:
- Test Suite: SampleTestSuite
- Test Group: SampleCmdTest
- Test Case: test sample noop

Script Output:
2020/07/27 11:36:50.234 (sample_test.rb:23): CHECK: SAMPLE SAMPLE_HK CMD_CNT -- 1 success with value -- 1 after waiting 0.00292465 seconds