NASA/TM–20210000619/Rev 1

Core Flight System (cFS) Training

Integration with COSMOS

Flight Software Systems Branch, Code 582
Goddard Space Flight Center, Greenbelt, MD

March 2021

NASA STI Program ... in Profile

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

NASA/TM–20210000619/Rev 1

Core Flight System (cFS) Training

Integration with COSMOS

Flight Software Systems Branch, Code 582
Goddard Space Flight Center, Greenbelt, MD

March 2021

Level of Review: This material has been technically reviewed by technical management.

Available from

NASA STI Program
Mail Stop 148
NASA's Langley Research Center
Hampton, VA 23681-2199

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
703-605-6000

Available in electronic form at https://www.sti.nasa.gov and https://ntrs.nasa.gov

National Aeronautics and Space Administration

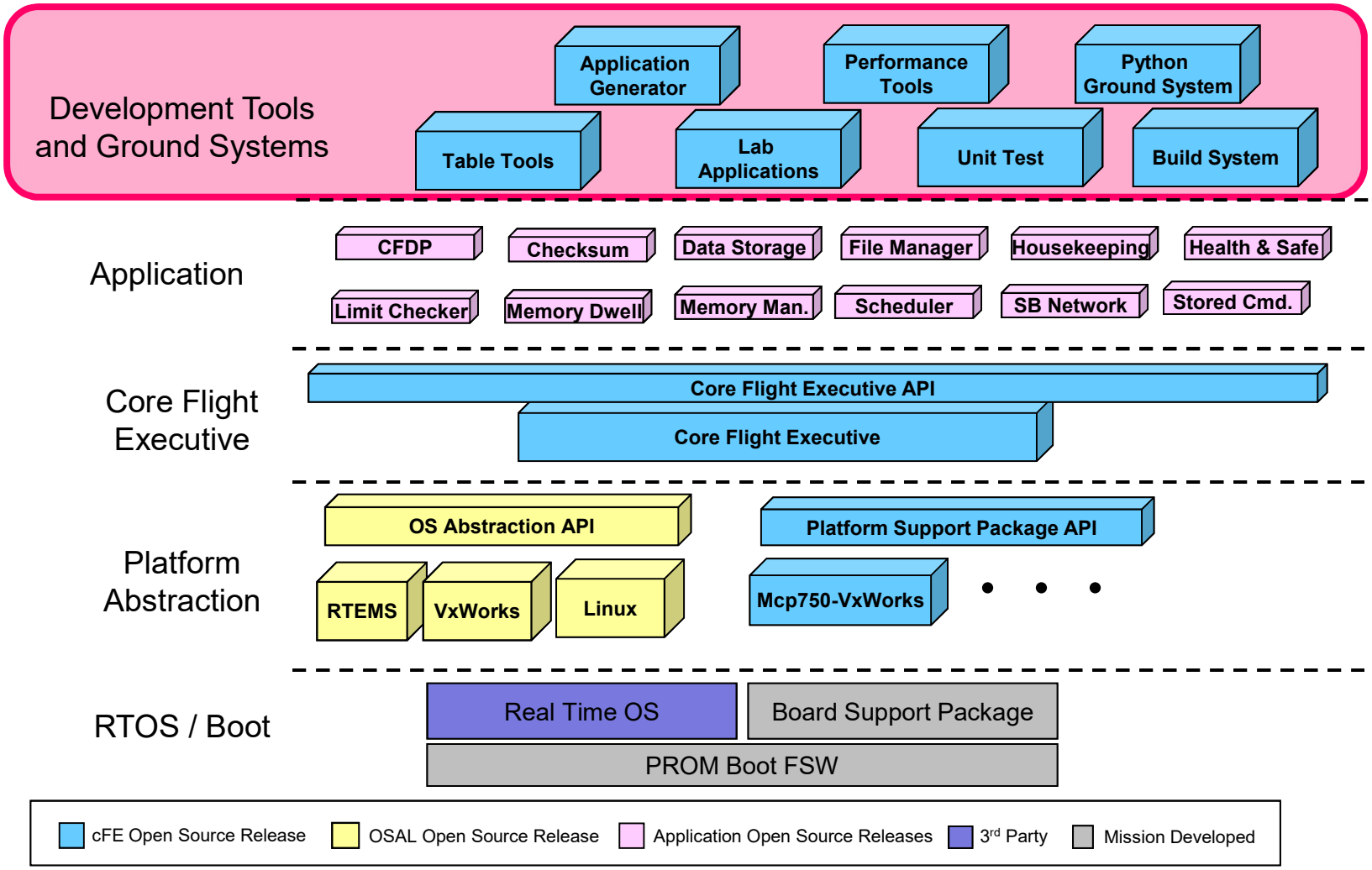
Core Flight Executive (cFS) Training

Integration with COSMOS

Course Agenda

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**

COSMOS - cFS Context



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


National Aeronautics and Space Administration
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:

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

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

Copy in the default makefile and definitions:

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

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):

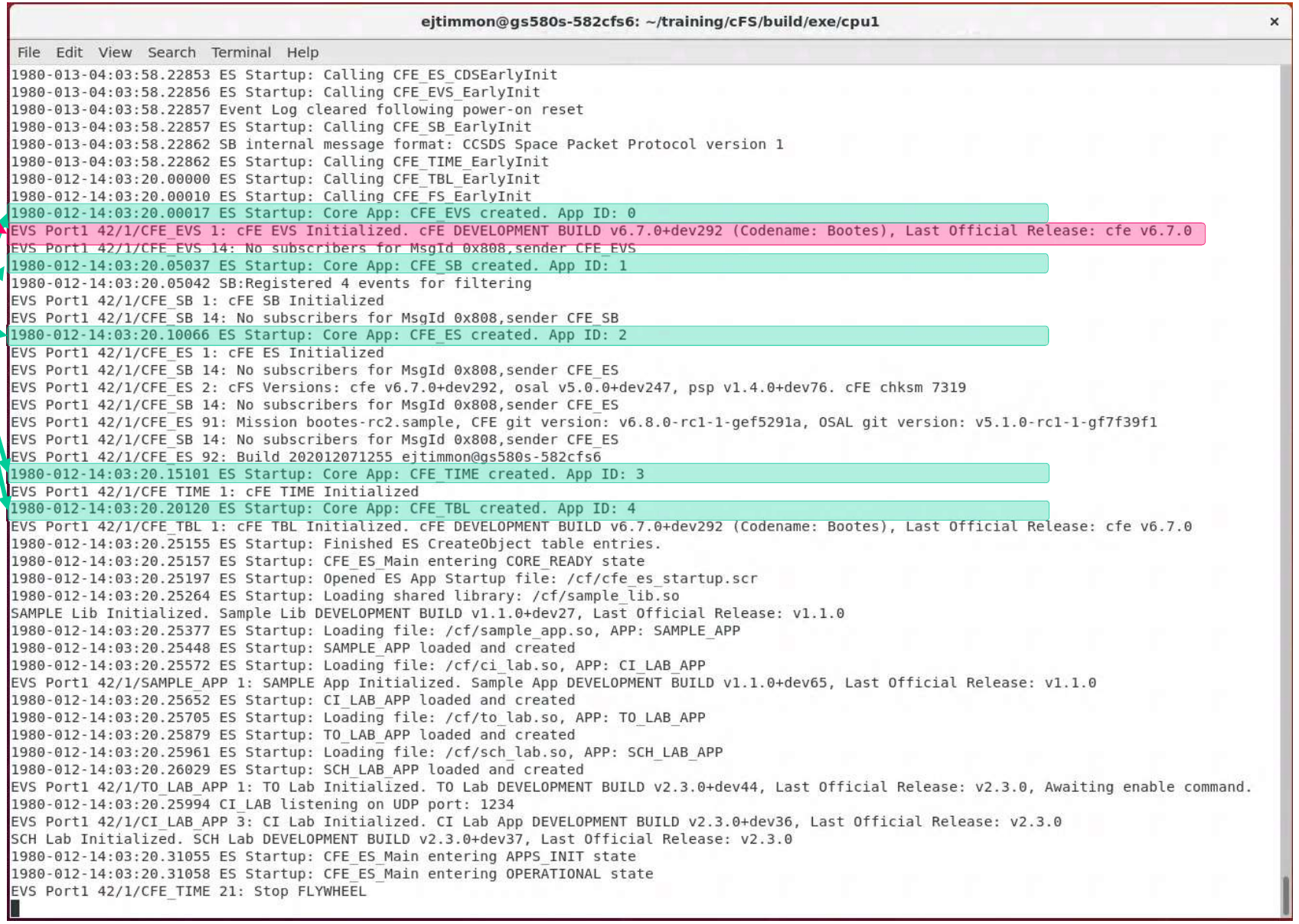
```
make prep
make
make install
cd build/exe/cpu1/
./core-cpu1
```

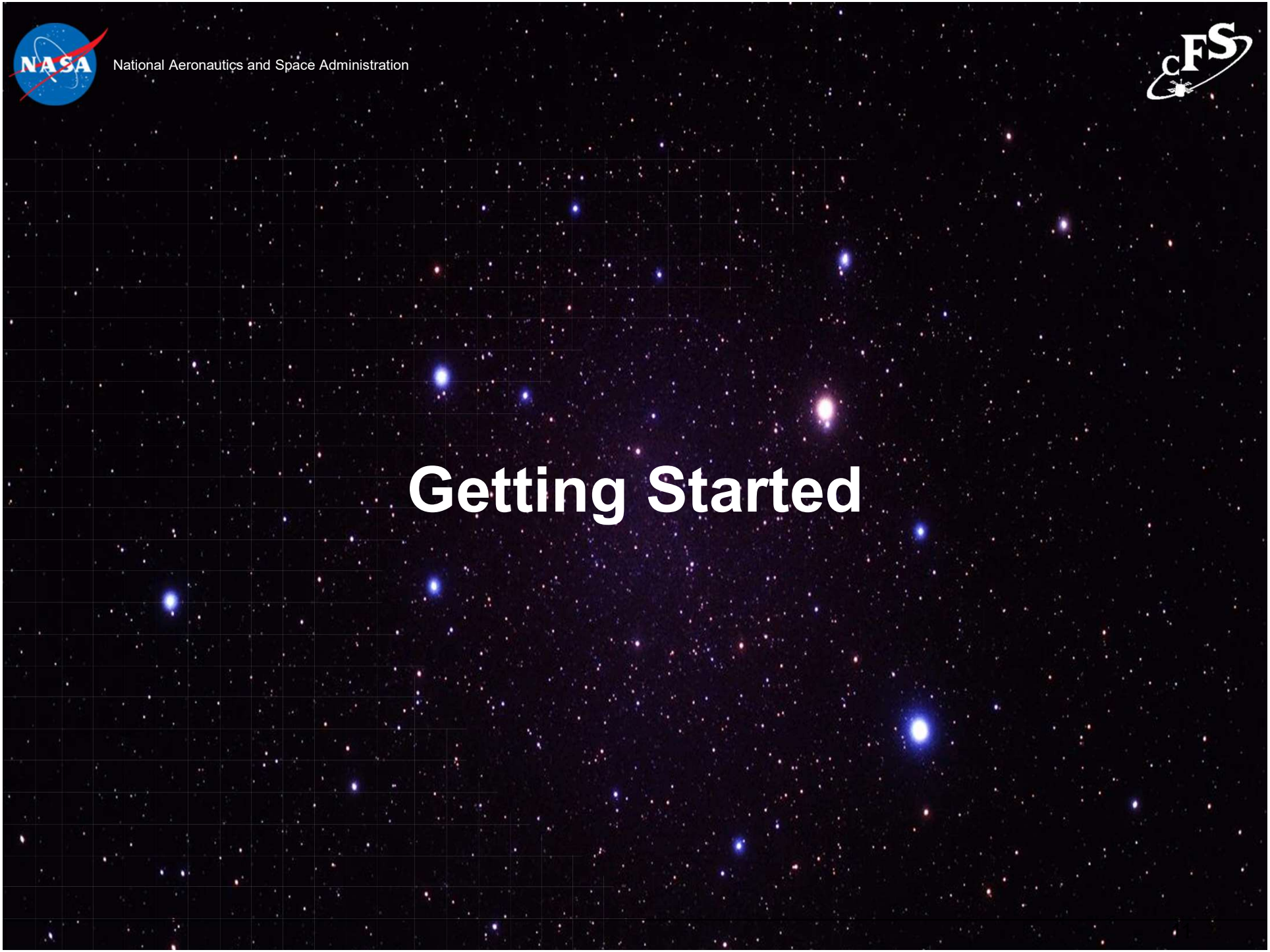
Should see startup messages and CFE_ES_Main entering OPERATIONAL state. Note the code must be executed from the build/exe/cpu1 directory to find the startup script and shared objects.

Exercise 0 Recap

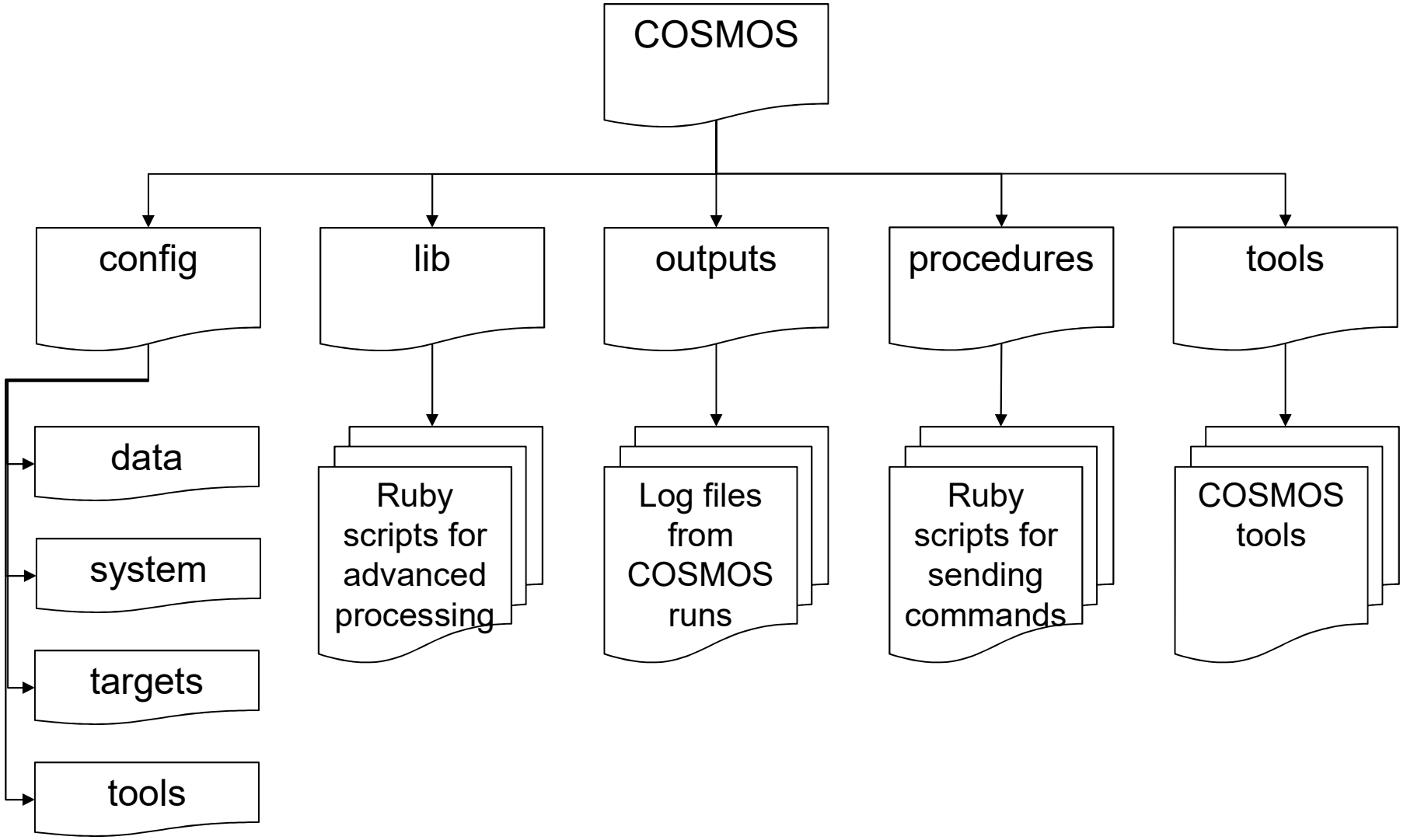
cFE Version

cFE Services Started




National Aeronautics and Space Administration
NASA
cFS
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**

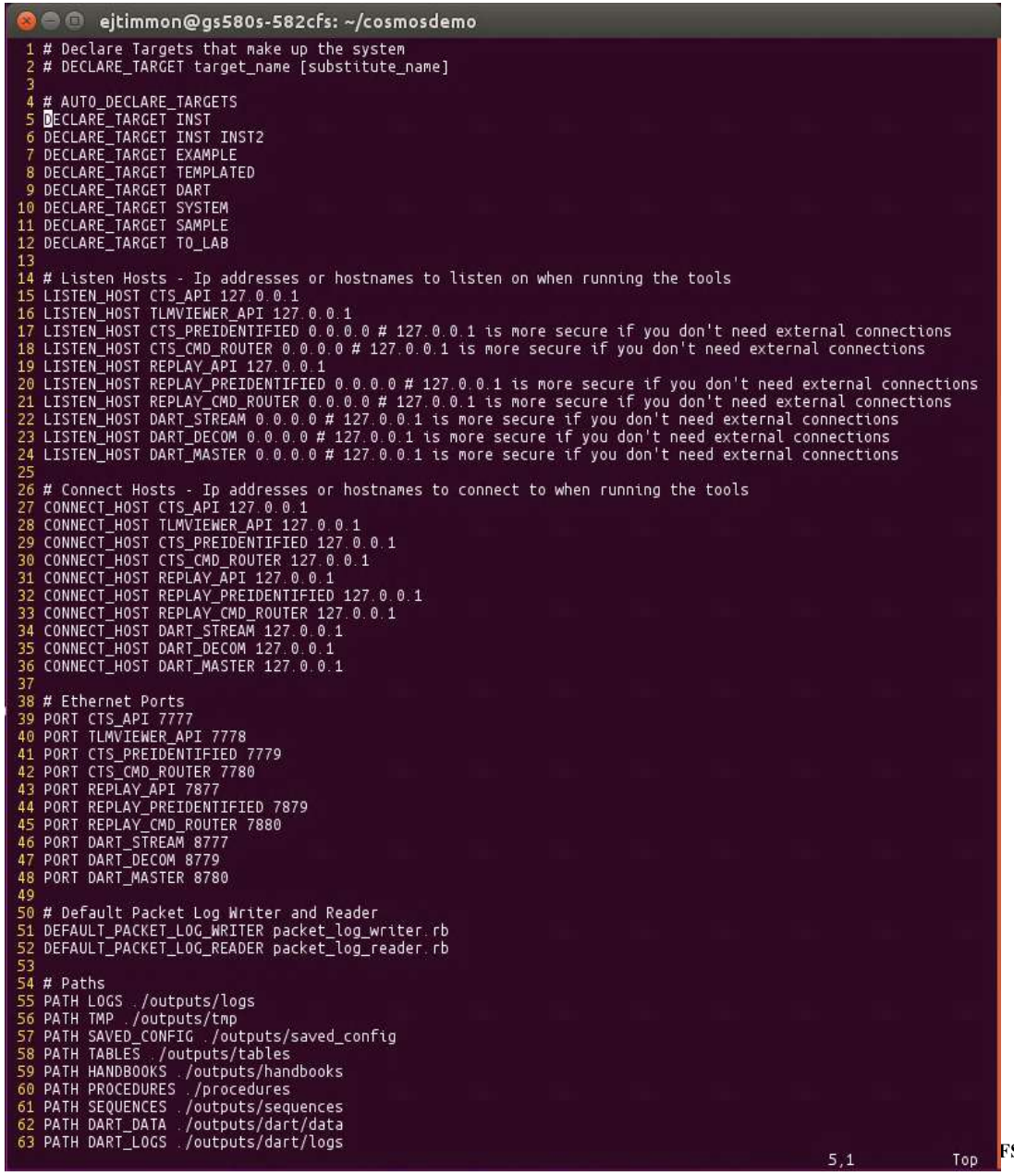
System Configuration (system.txt)

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.



Command/Telemetry Server Configuration (cmd_tlm_server.txt)

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

```
1  TITLE 'COSMOS Command and Telemetry Server - Demo Configuration'
2
3  # PACKET_LOG_WRITER Parameter Notes
4  # nil:use default log names
5  # true: logging enabled
6  # nil: Don't cycle logs based on time
7  # 2000000000: Create new log after 2 Billion bytes
8  # nil: Use the default log directory
9  # false: Log synchronously - more efficient
10 PACKET_LOG_WRITER DEFAULT packet_log_writer.rb nil true nil 2000000000 nil false
11 # PACKET_LOG_WRITER SYSTEMLOG packet_log_writer.rb system
12
13 # Explicitly declare these interfaces since we're using name substitution
14 INTERFACE_TARGET INST cmd_tlm_server.txt # Use cmd_tlm_server.txt in targets/INST
15 INTERFACE_TARGET INST2 cmd_tlm_server2.txt # Use cmd_tlm_server2.txt in targets/INST
16
17 # Here is an example of declaring the interface directly
18 INTERFACE SYSTEM_INT cmd_tlm_server_interface.rb
19   TARGET SYSTEM
20   DISABLE_DISCONNECT
21   # LOG SYSTEMLOG
22   # DONT_LOG
23   # DONT_CONNECT
24   # DONT_RECONNECT
25   # RECONNECT_DELAY 15.0
26   # LOG_RAW
27
28 # Auto interface the rest of the targets by using their cmd_tlm_server.txt file
29 AUTO_INTERFACE_TARGETS
30
31 ROUTER INST_ROUTER tcpip_server_interface.rb 2055 2055 10.0 nil LENGTH 32 16 7
32   OPTION LISTEN_ADDRESS 127.0.0.1
33   ROUTE INST_INT
34   # DONT_CONNECT
35   # DONT_RECONNECT
36   # DISABLE_DISCONNECT
37   # RECONNECT_DELAY 15.0
38   # LOG_RAW
39
40 BACKGROUND_TASK example_background_task.rb
41   STOPPED
42 BACKGROUND_TASK limits_groups.rb 5 # Initial delay to allow interfaces to connect
```

Exercise 1 – Create a new target

Objective: Create a new target for the sample app

Part 1 – Add sample_app to COSMOS

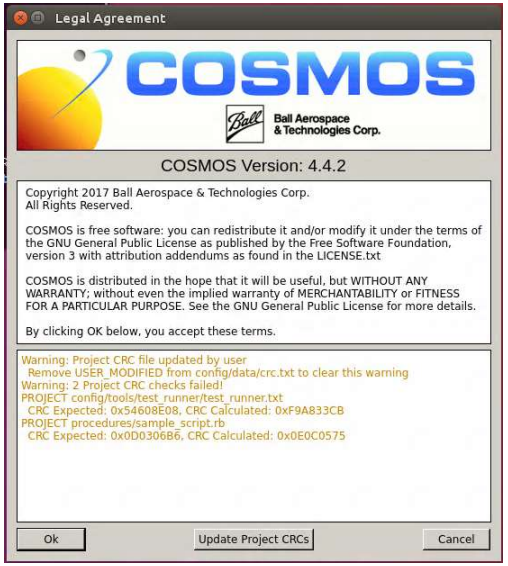
1. Navigate to the config/targets directory in COSMOS
2. Create a directory called “SAMPLE”
3. Enter the “SAMPLE” directory
4. Create a file called “target.txt”
5. Navigate to the directory “cosmosdemo/config/system”
6. Open the file “system.txt” and add the line “DECLARE_TARGET SAMPLE”
   - This tells 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

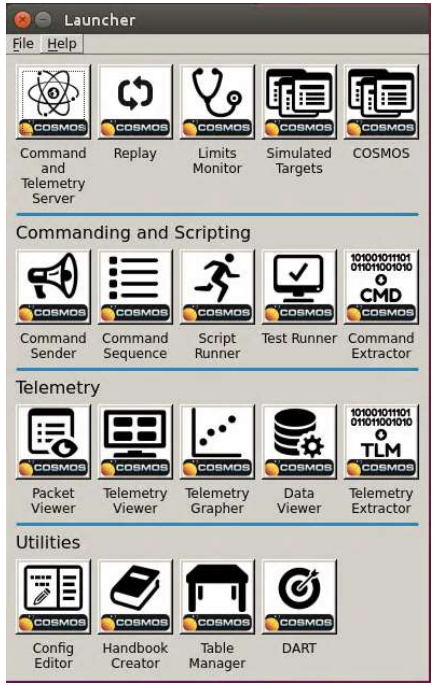
```
INTERFACE LOCAL udp_interface.rb 127.0.0.1 1234 1235 nil nil 128 nil nil
                  TARGET sample
```

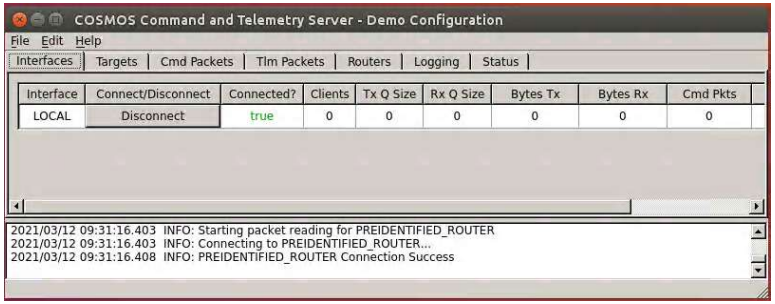
   - For simplicity you can delete everything in the “cmd_tlm_server.txt” file below the LOCAL interface

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

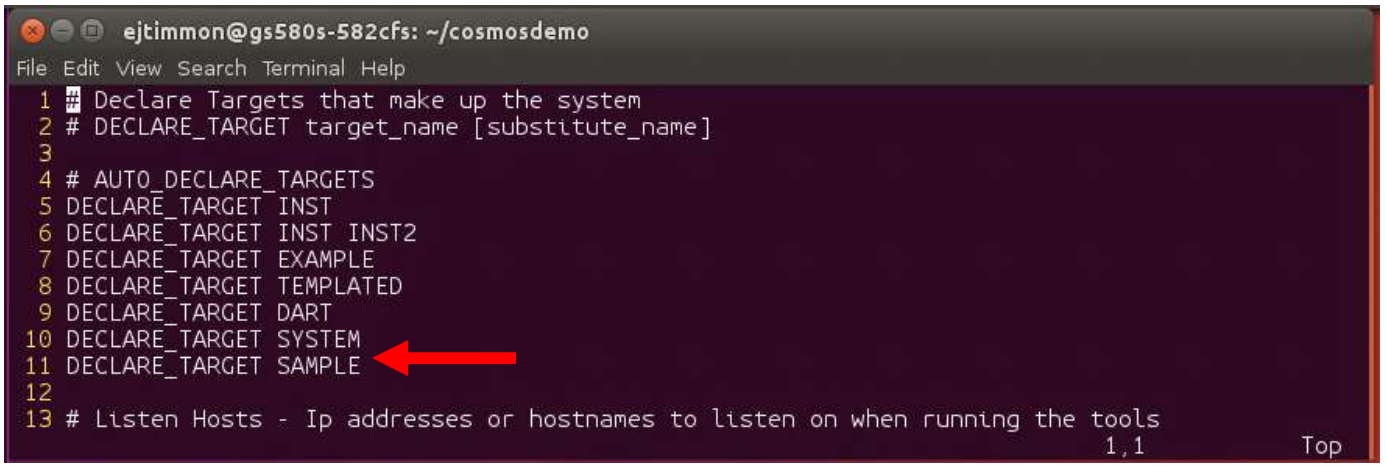






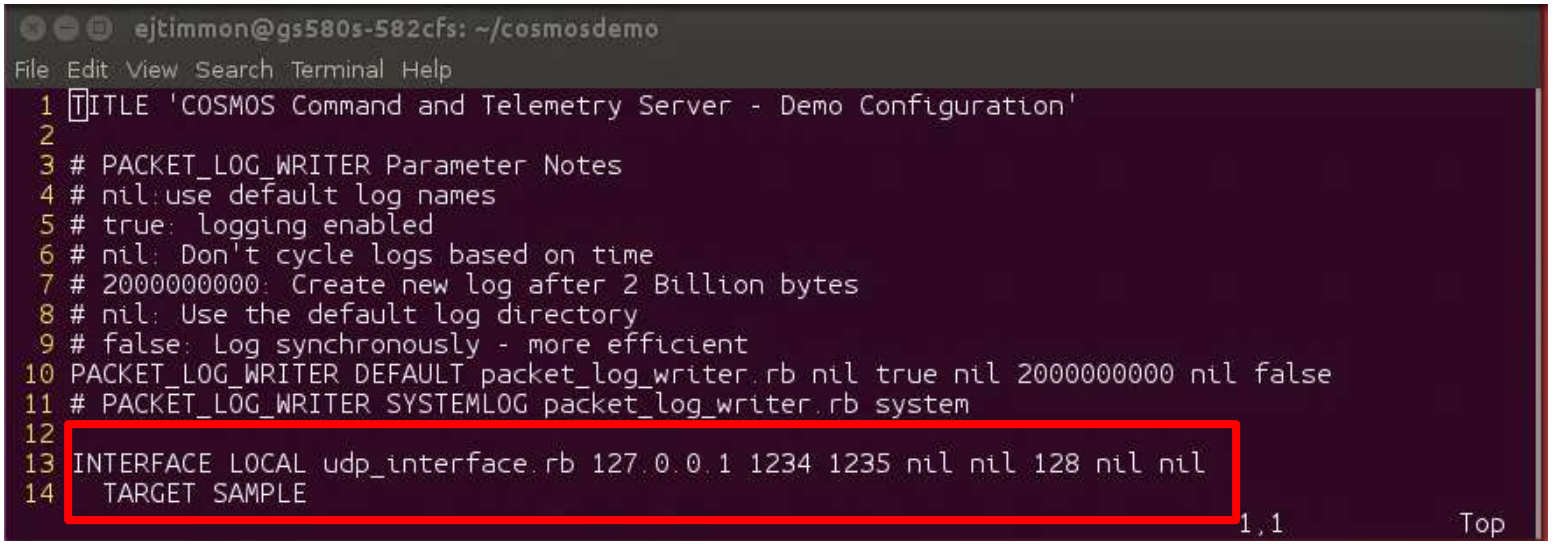
Successful startup confirms that Part 1 was done correctly.

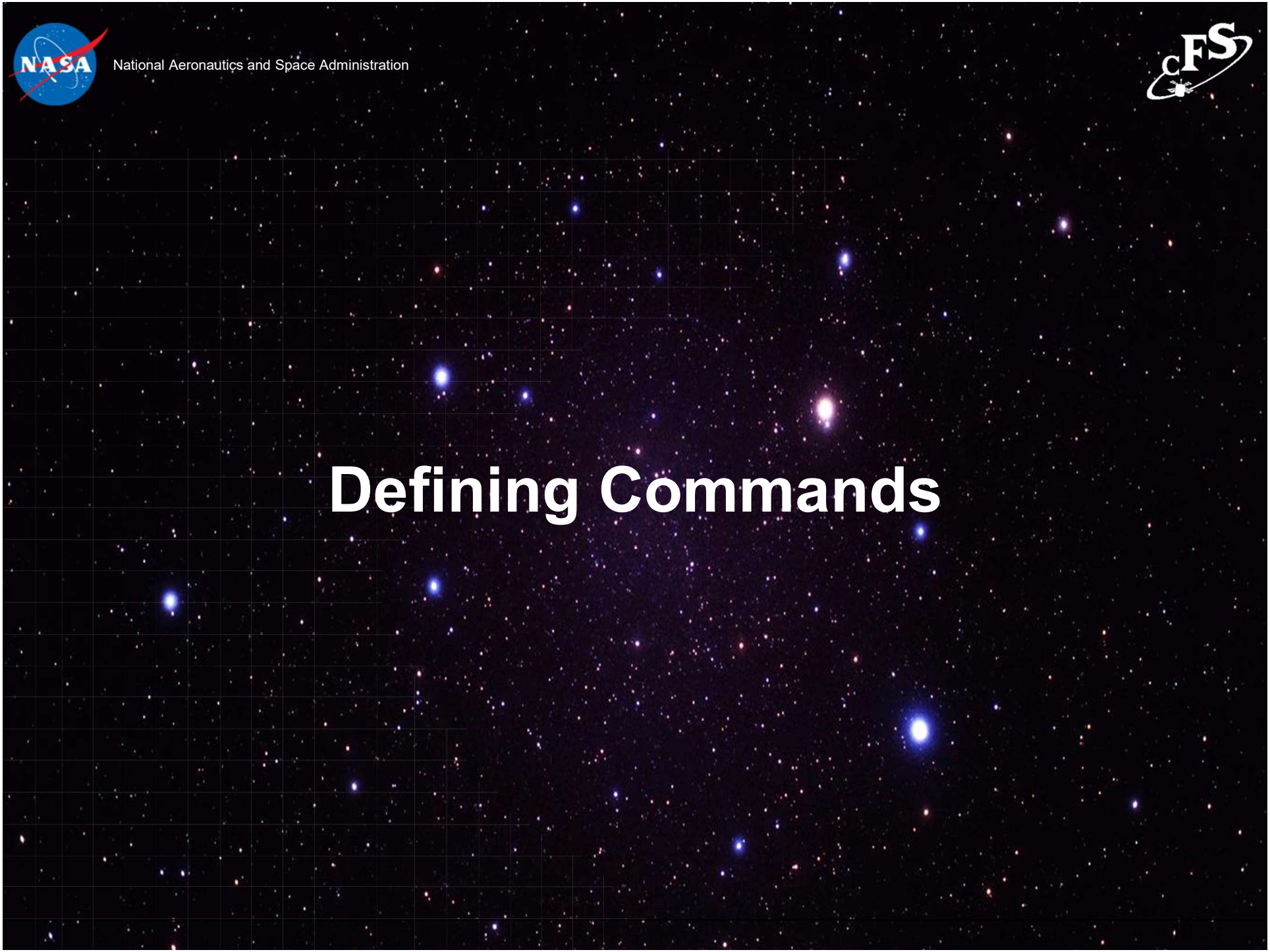
Exercise 1 - Recap



system.txt

cmd_tlm_server.txt

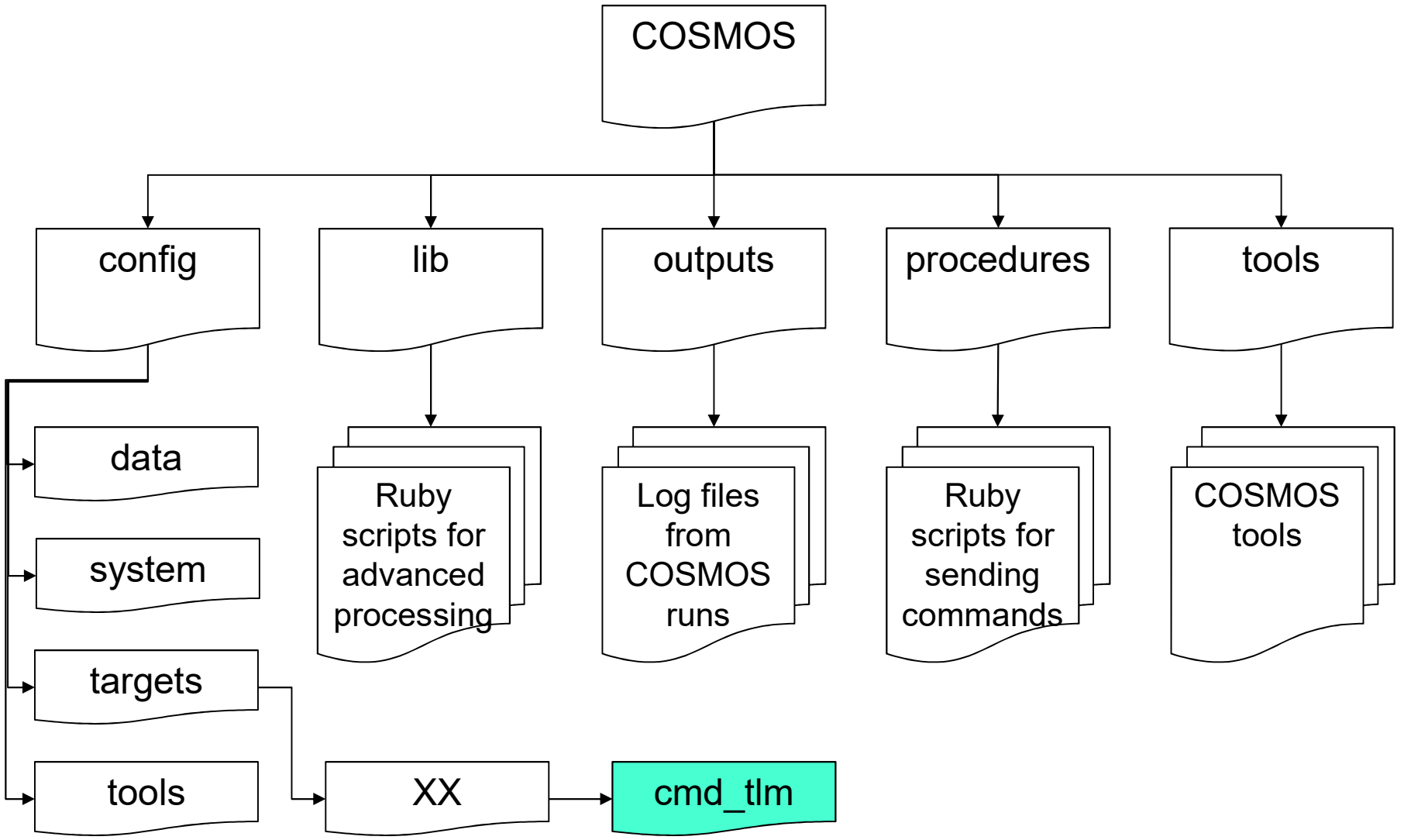



National Aeronautics and Space Administration
Defining Commands

Command Databases

- 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



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**

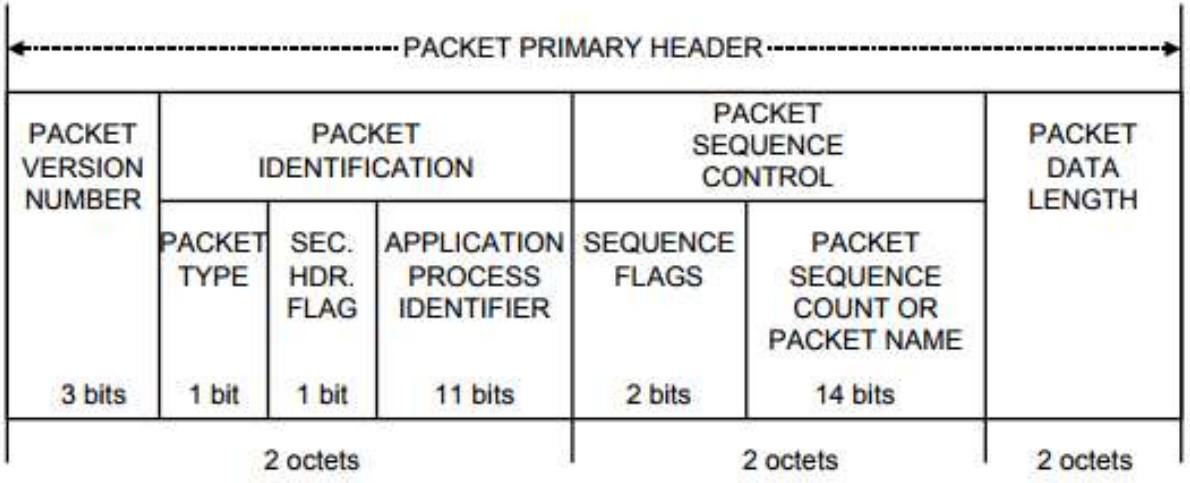
```
COMMAND <Target> <Command Name> <Endianness> <Description>
```

- **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

```
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:**

```
typedef struct
{
   uint8    CmdHeader[CFE_SB_CMD_HDR_SIZE];
} SAMPLE_NoArgsCmd_t;
```

- **Sample_app No-Op Command in COSMOS:**

SAMPLE_APP_CMD_MID
from sample_app_msgids.h

```
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

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

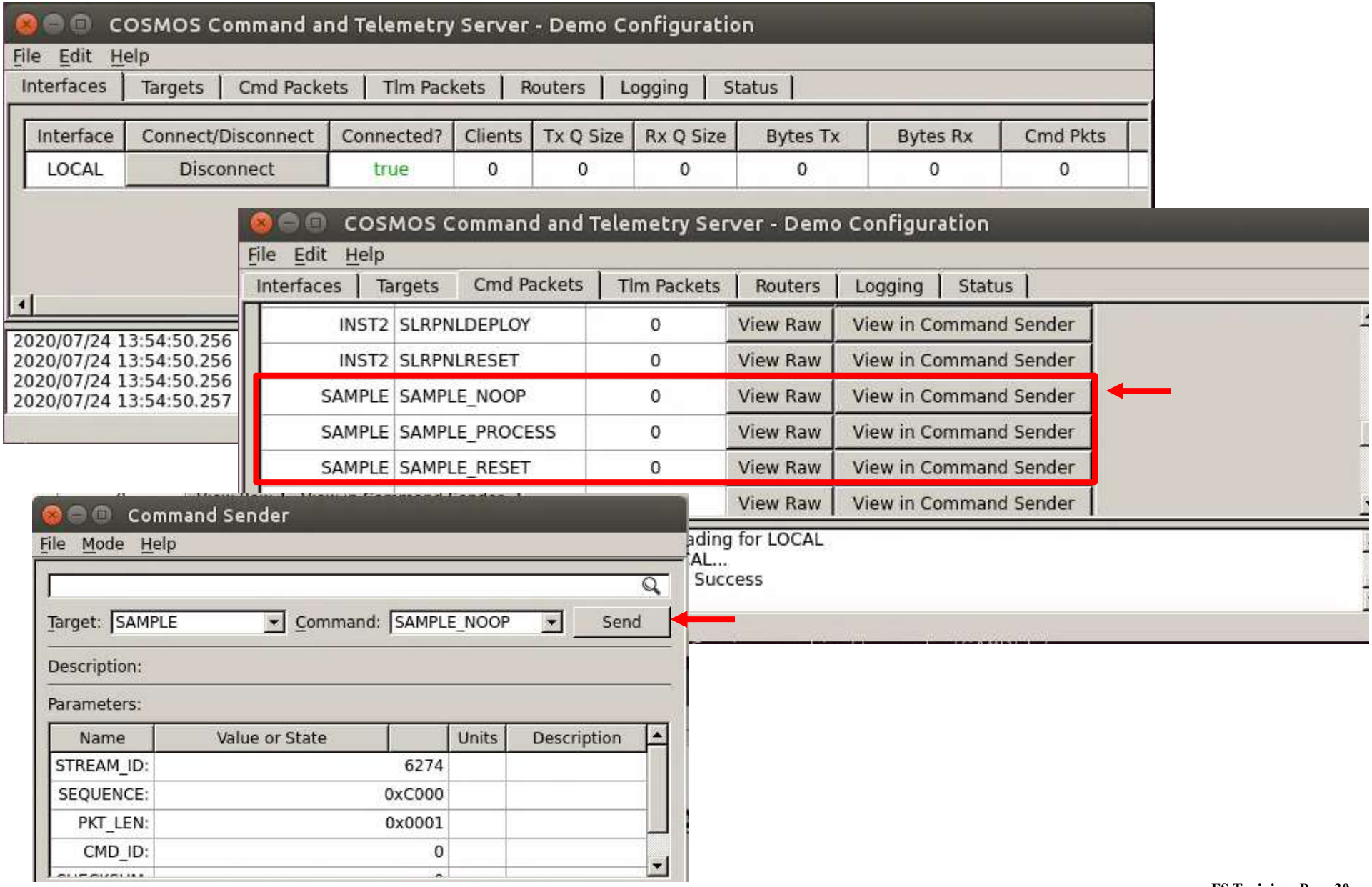
Part 2 – Send commands to sample_app

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

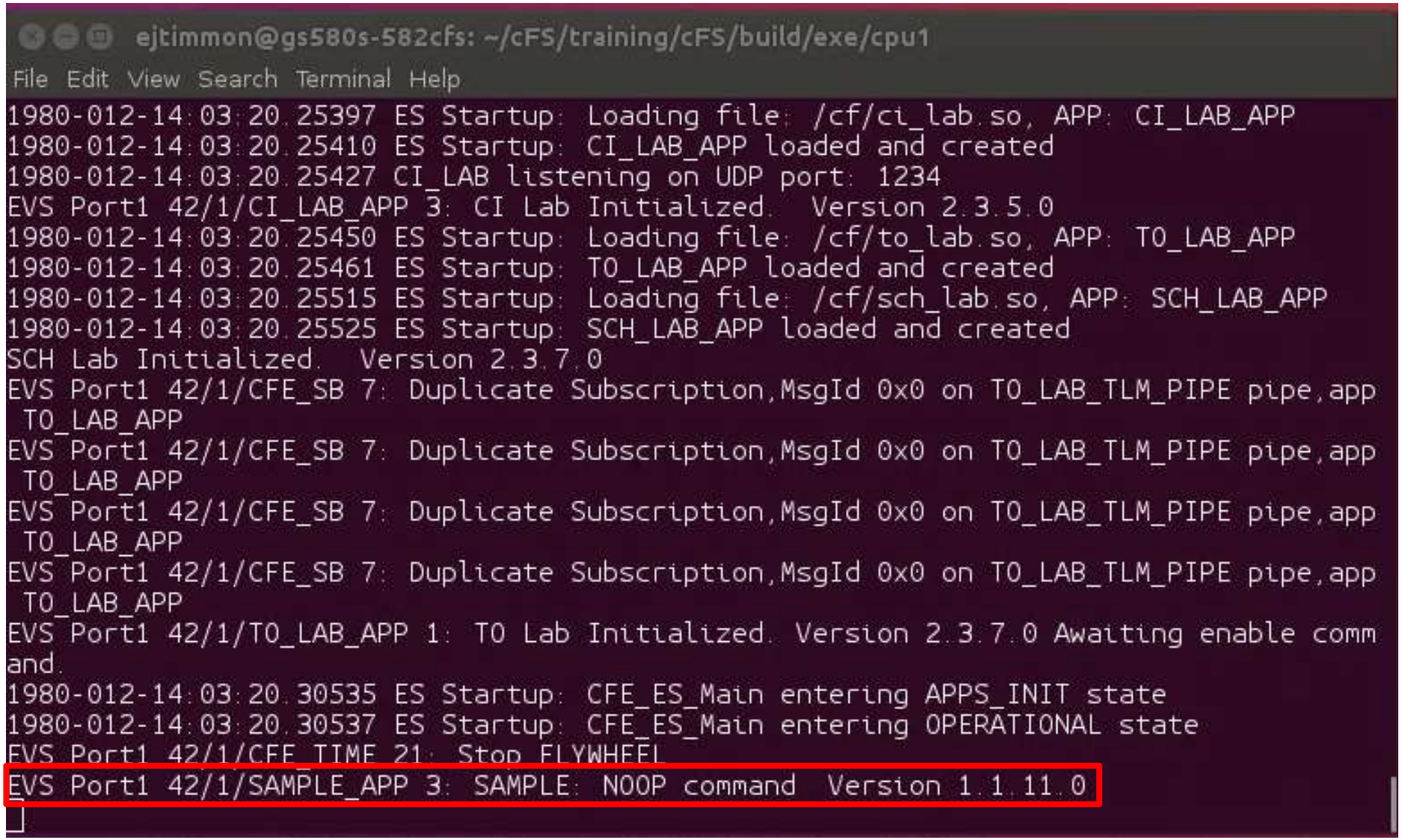
The sample_cmds.txt file:

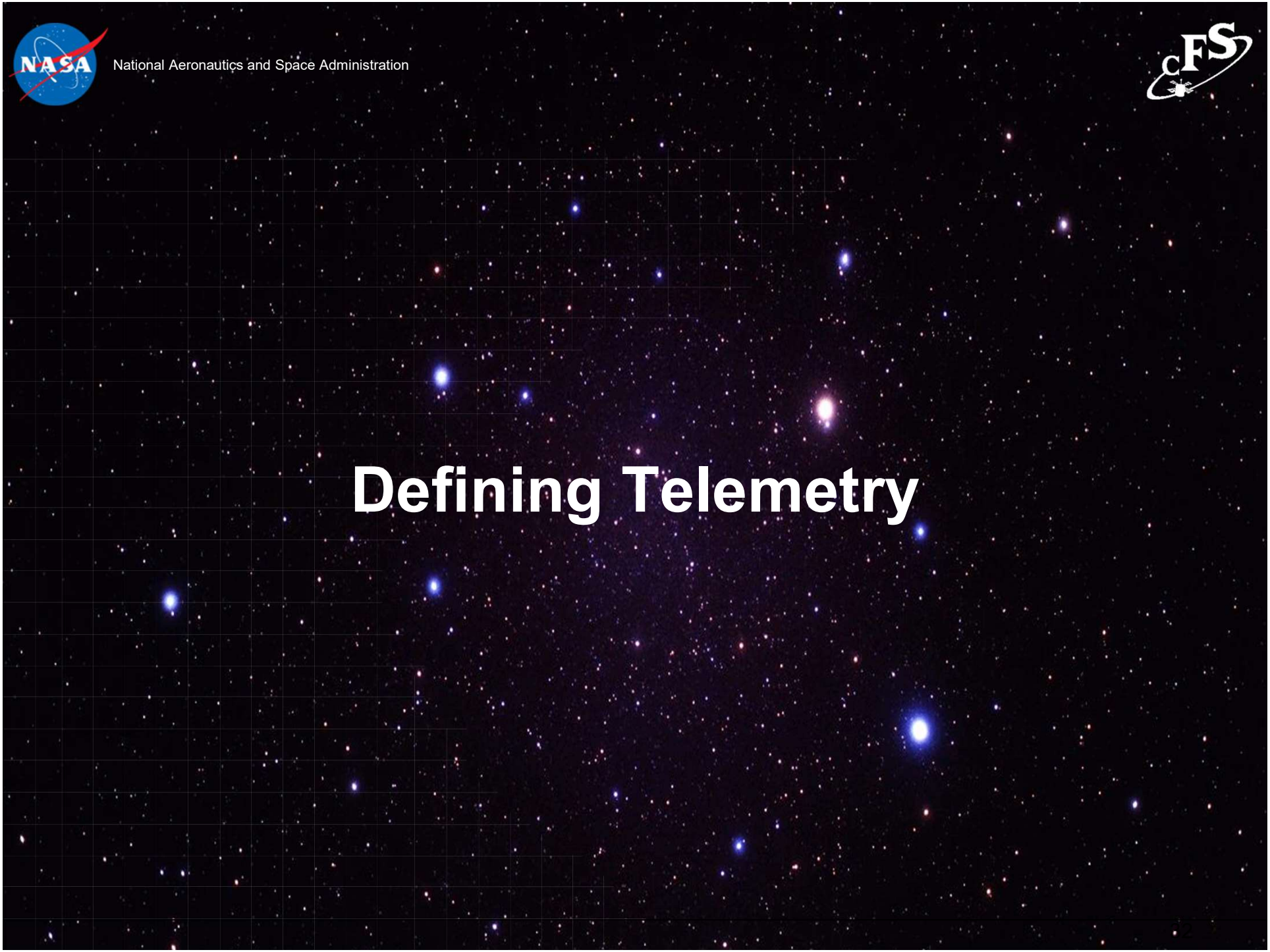
```
COMMAND SAMPLE SAMPLE_NOOP BIG_ENDIAN "Sample_app NOOP Command"
     APPEND_ID_PARAMETER STREAM_ID 16 UINT  0x1882    0x1882     0x1882    ""
     APPEND_PARAMETER    SEQUENCE  16 UINT  0xC000    MAX_UINT16 0xC000    ""
          FORMAT_STRING "0x%04X"
     APPEND_PARAMETER    PKT_LEN   16 UINT  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

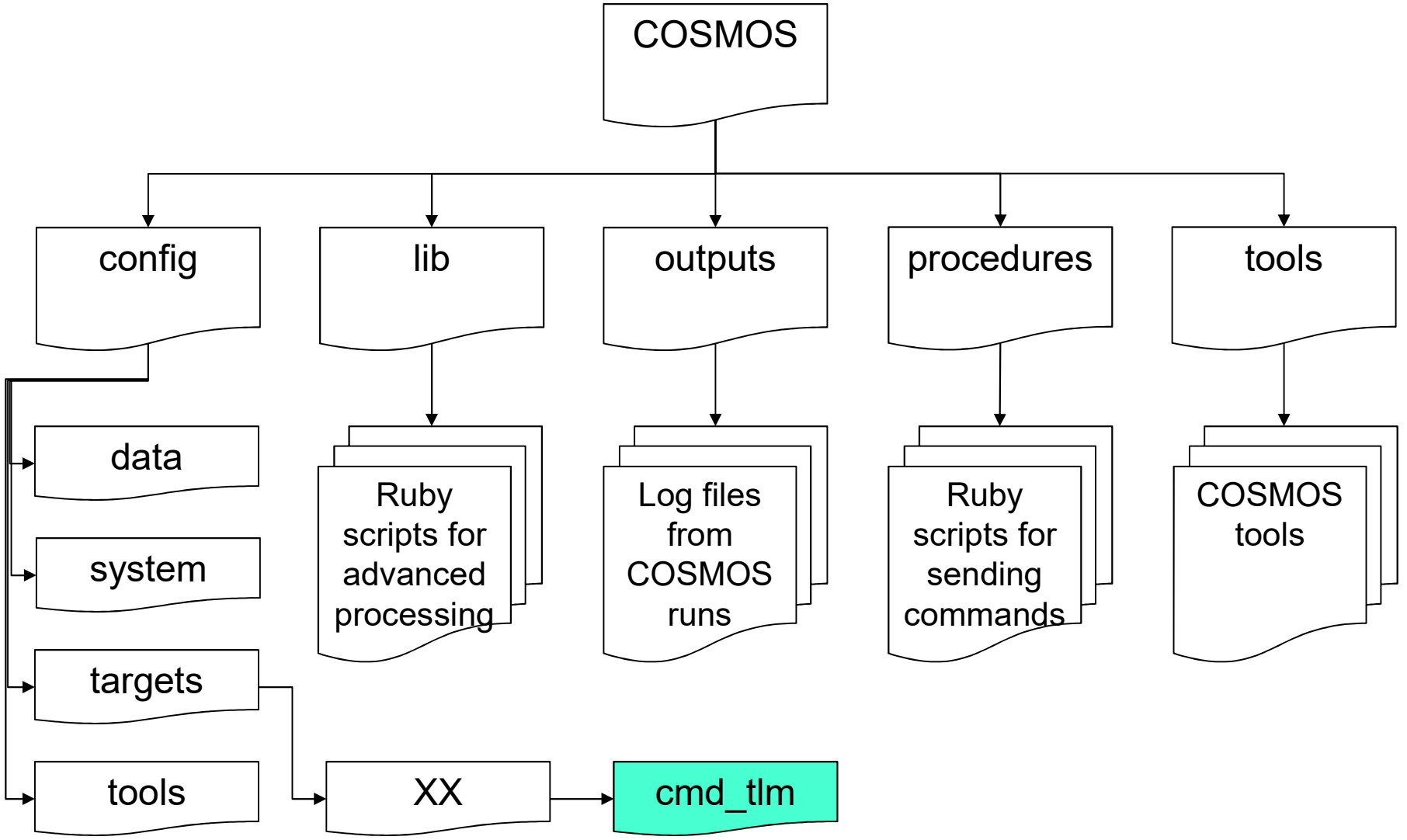



National Aeronautics and Space Administration
NASA
cFS
Defining Telemetry

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

Location of Telemetry Database



Relationship with cFS

- **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:**

```
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:**

```
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"
```

cFS Primary Header

cFS TLM Secondary Header

SAMPLE_APP_HK_TLM_MID from sample_app_msgids.h

SAMPLE_HkTlm_Payload_t

Exercise 3

- **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

Exercise 3 – Part 2

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.

Same process used to add the "SAMPLE" target in Exercise 1

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 o 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

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

ejtimmon@gs580s-582cfs: ~/cosmosdemo/config

File Edit View Search Terminal Help

```
 1 # Declare Targets that make up the system
 2 # DECLARE_TARGET target_name [substitute_name]
 3
 4 # AUTO_DECLARE_TARGETS
 5 DECLARE_TARGET INST
 6 DECLARE_TARGET INST INST2
 7 DECLARE_TARGET EXAMPLE
 8 DECLARE_TARGET TEMPLATED
 9 DECLARE_TARGET DART
10 DECLARE_TARGET SYSTEM
11 DECLARE_TARGET SAMPLE
12 DECLARE_TARGET TO_LAB
13
```

system.txt

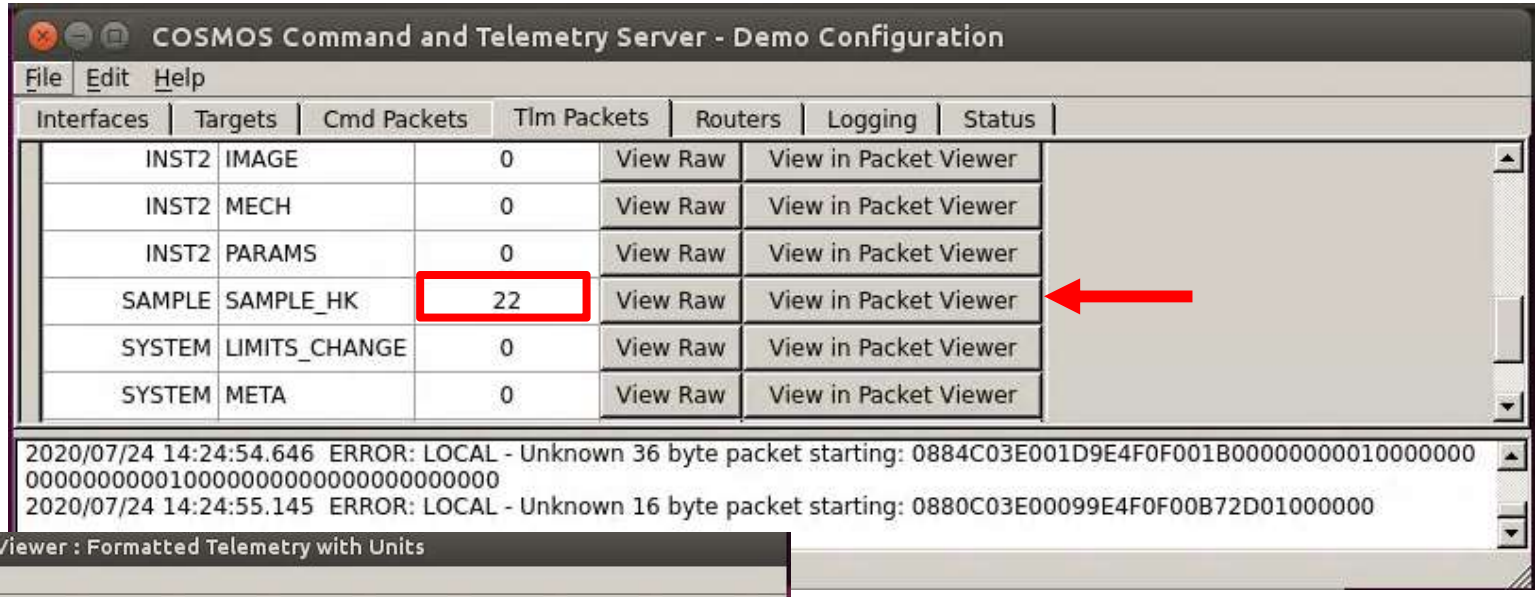
cmd_tlm_server.txt

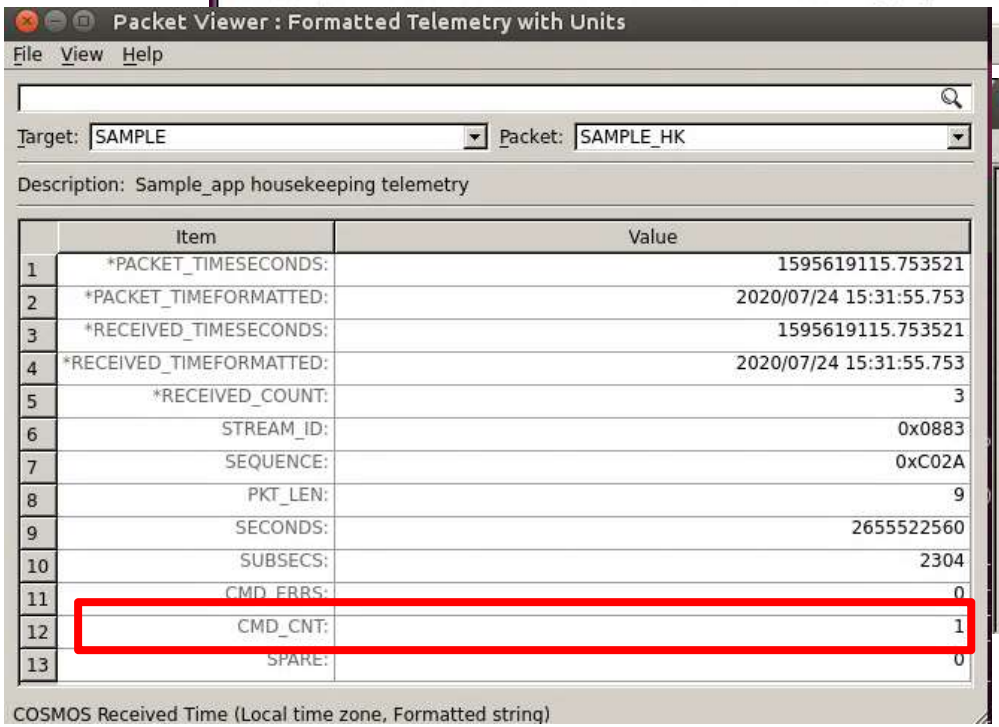
ejtimmon@gs580s-582cfs: ~/cosmosdemo/config

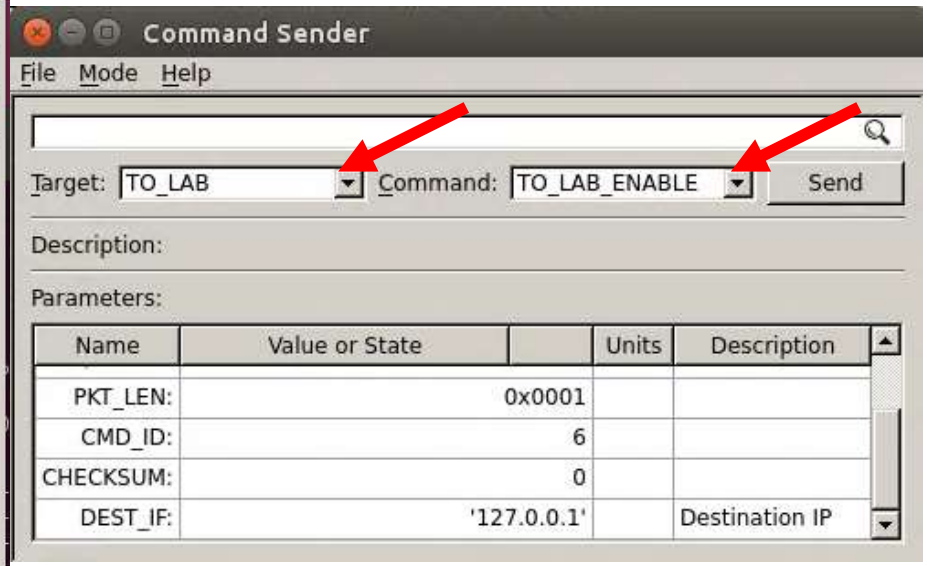
File Edit View Search Terminal Help

```
 1 TITLE 'COSMOS Command and Telemetry Server - Demo Configuration'
 2
 3 # PACKET_LOG_WRITER Parameter Notes
 4 # nil:use default log names
 5 # true: logging enabled
 6 # nil: Don't cycle logs based on time
 7 # 2000000000: Create new log after 2 Billion bytes
 8 # nil: Use the default log directory
 9 # false: Log synchronously - more efficient
10 PACKET_LOG_WRITER DEFAULT packet_log_writer.rb nil true nil 2000000000 nil false
11 # PACKET_LOG_WRITER SYSTEMLOG packet_log_writer.rb system
12
13 INTERFACE LOCAL udp_interface.rb 127.0.0.1 1234 1235 nil nil 128 nil nil
14   TARGET SAMPLE
15   TARGET TO_LAB
16
```

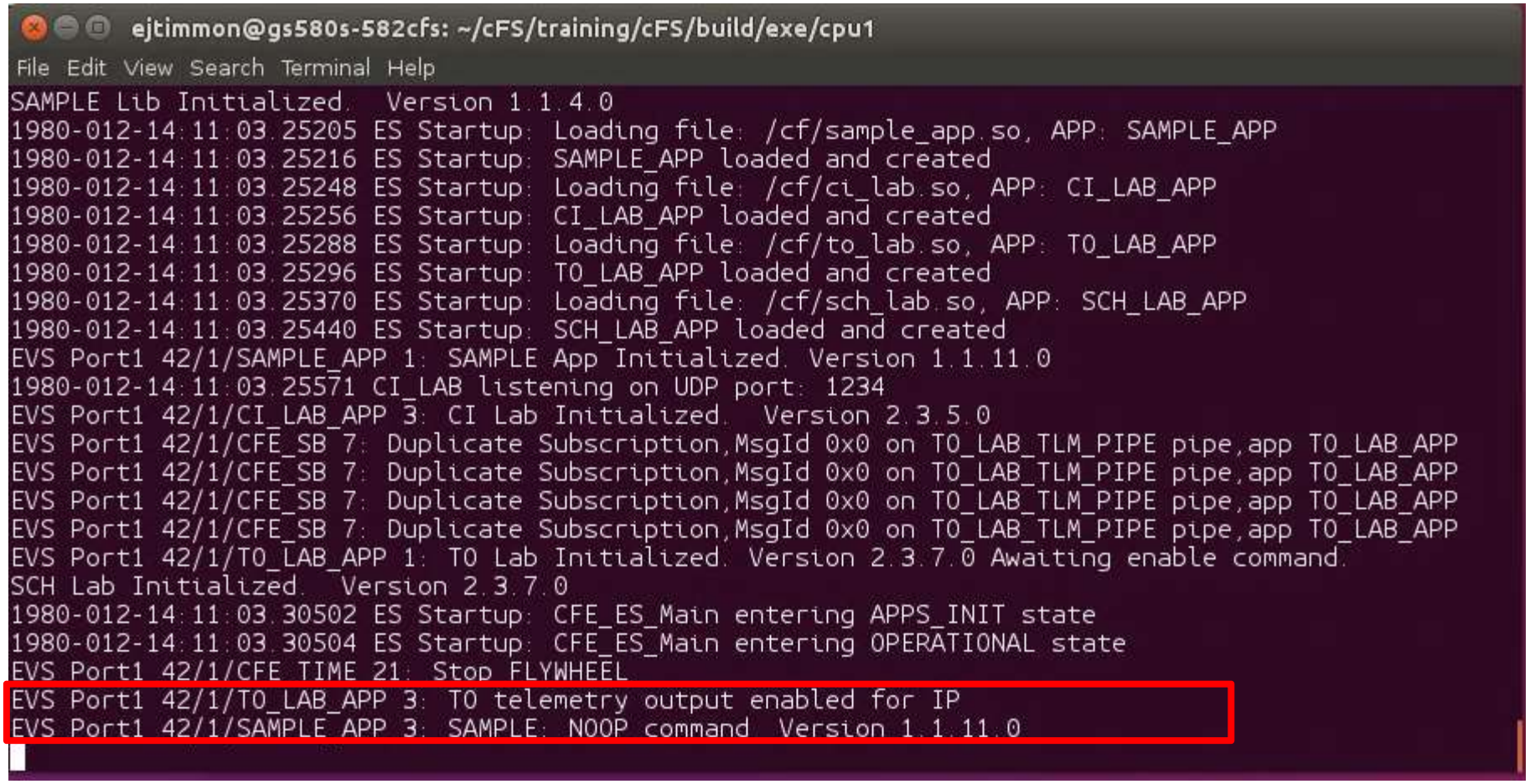
Exercise 3 - Recap

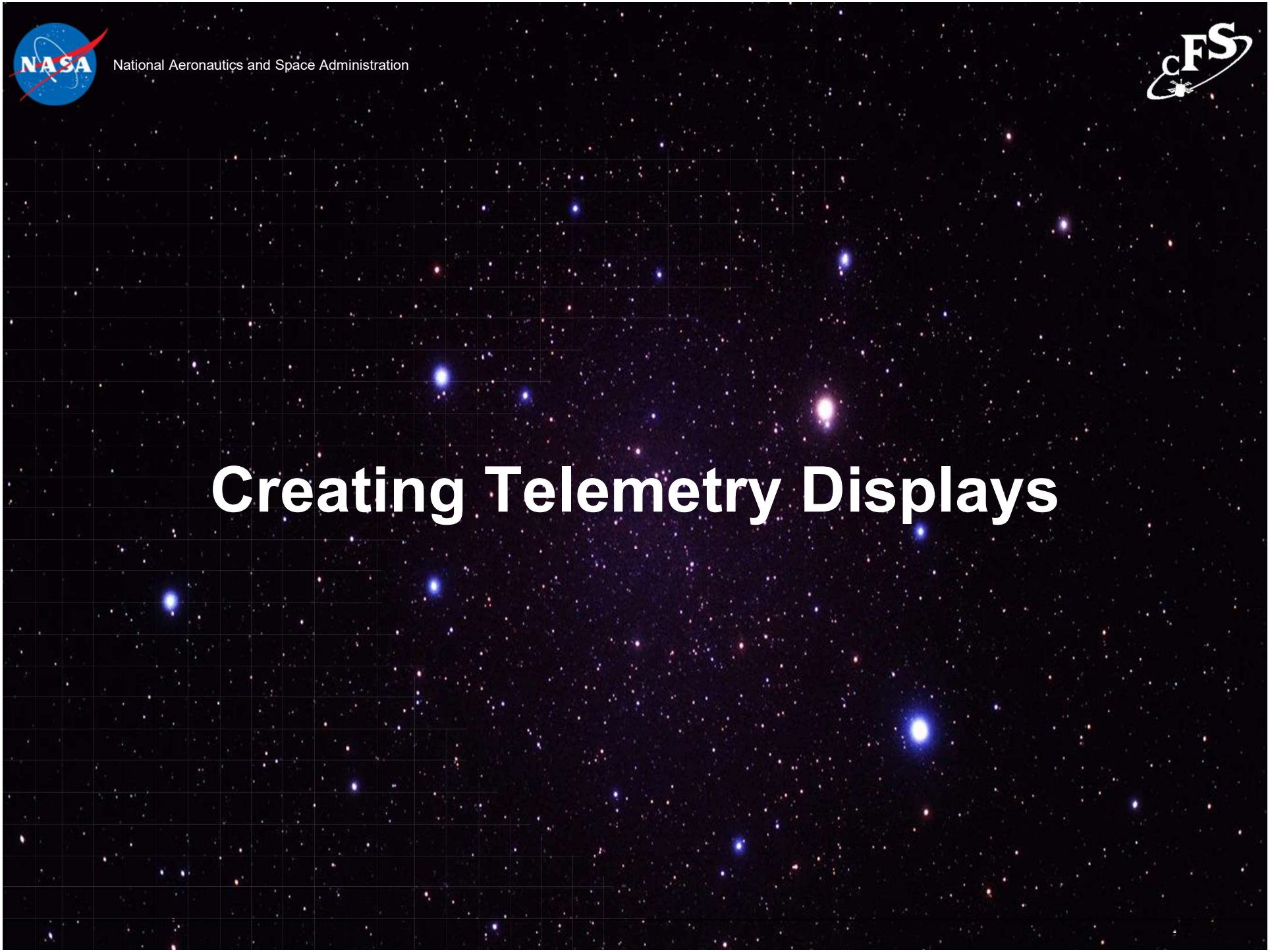






Exercise 3 - Recap



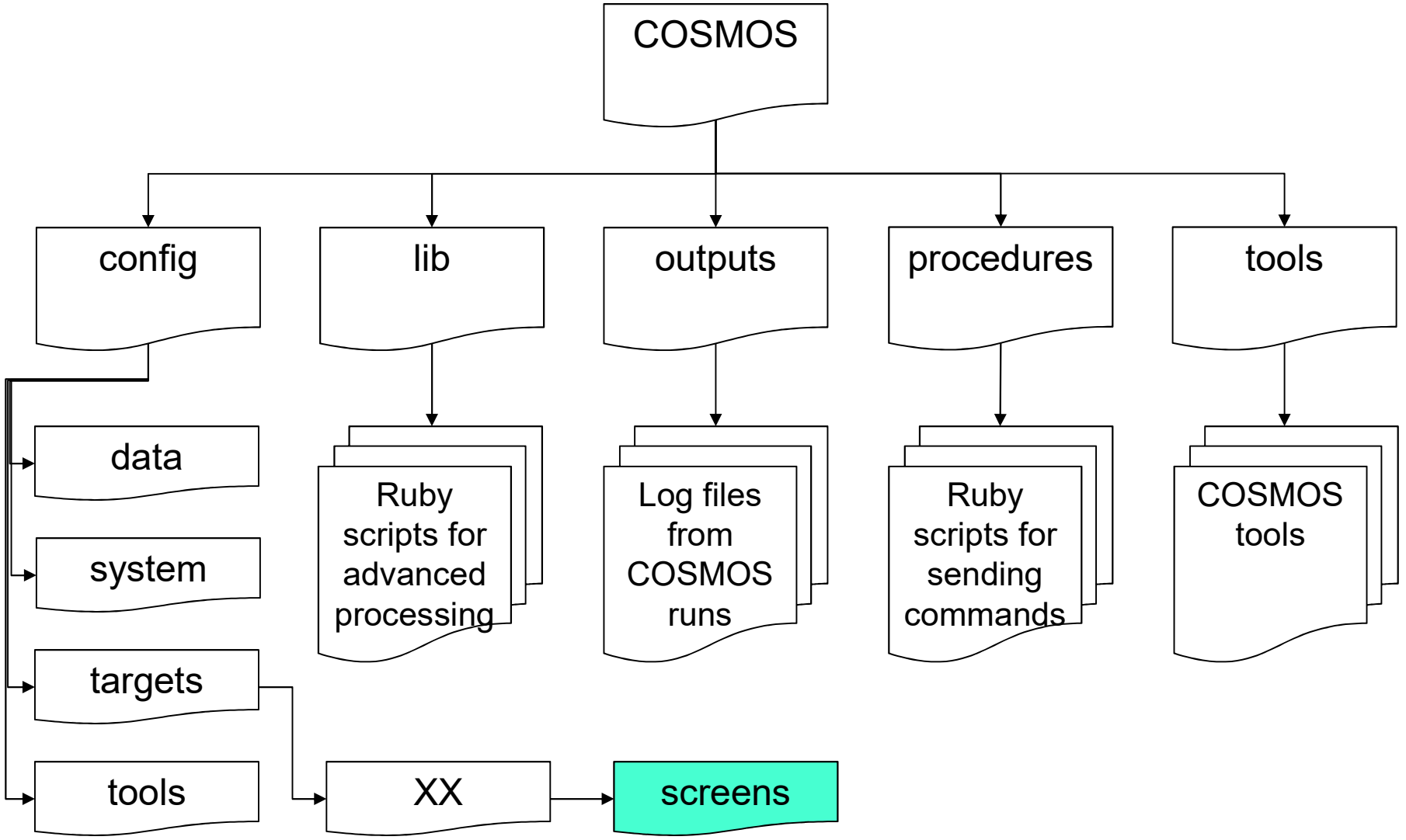

National Aeronautics and Space Administration
NASA
cFS
Creating Telemetry Displays

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



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”

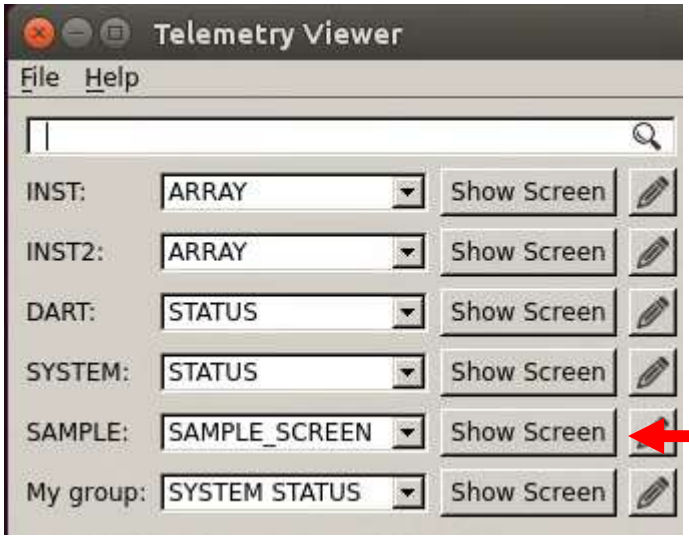
Exercise 4 - Recap

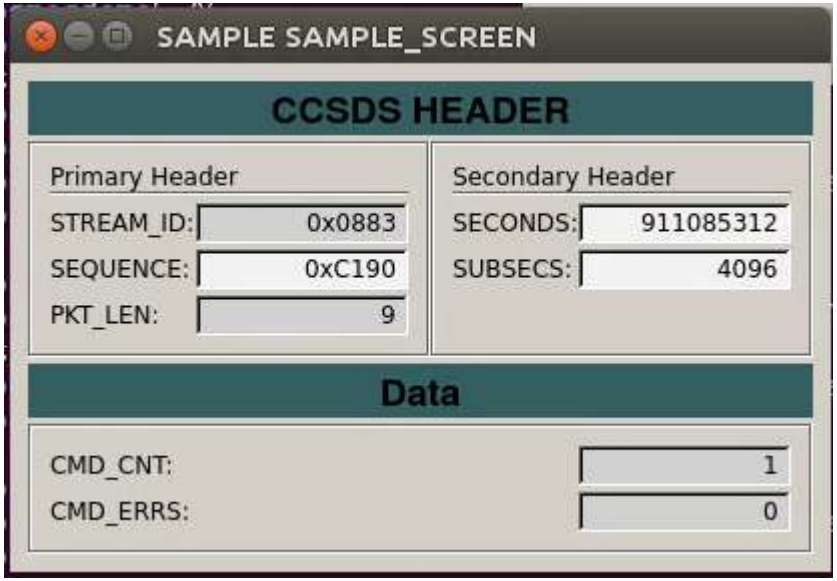
ejtimmon@gs580s-582cfs: ~/cosmosdemo

File Edit View Search Terminal Help

```
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

END

```

Exercise 4 - Recap





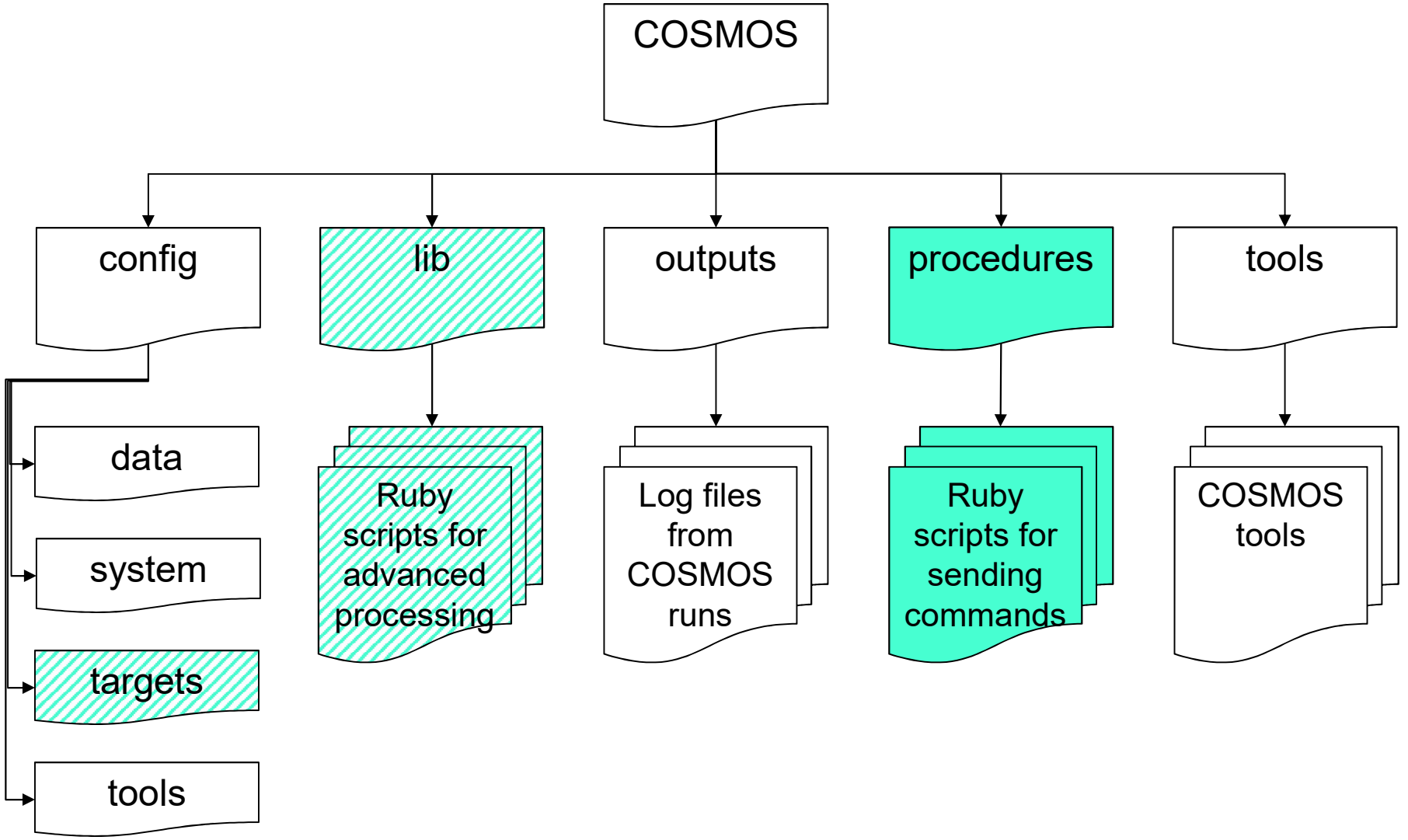

National Aeronautics and Space Administration
Basic Scripting

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/

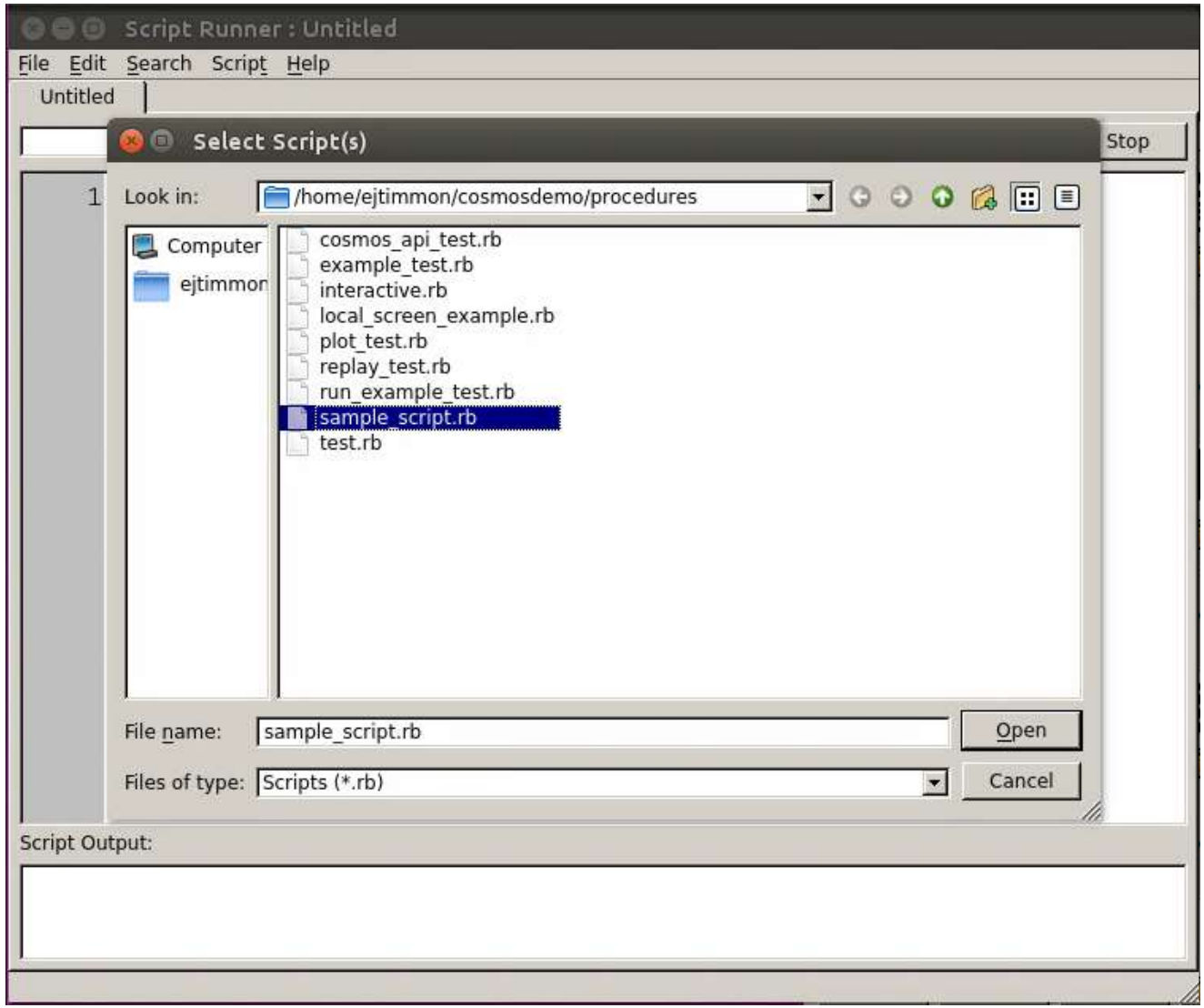
Location of Scripts



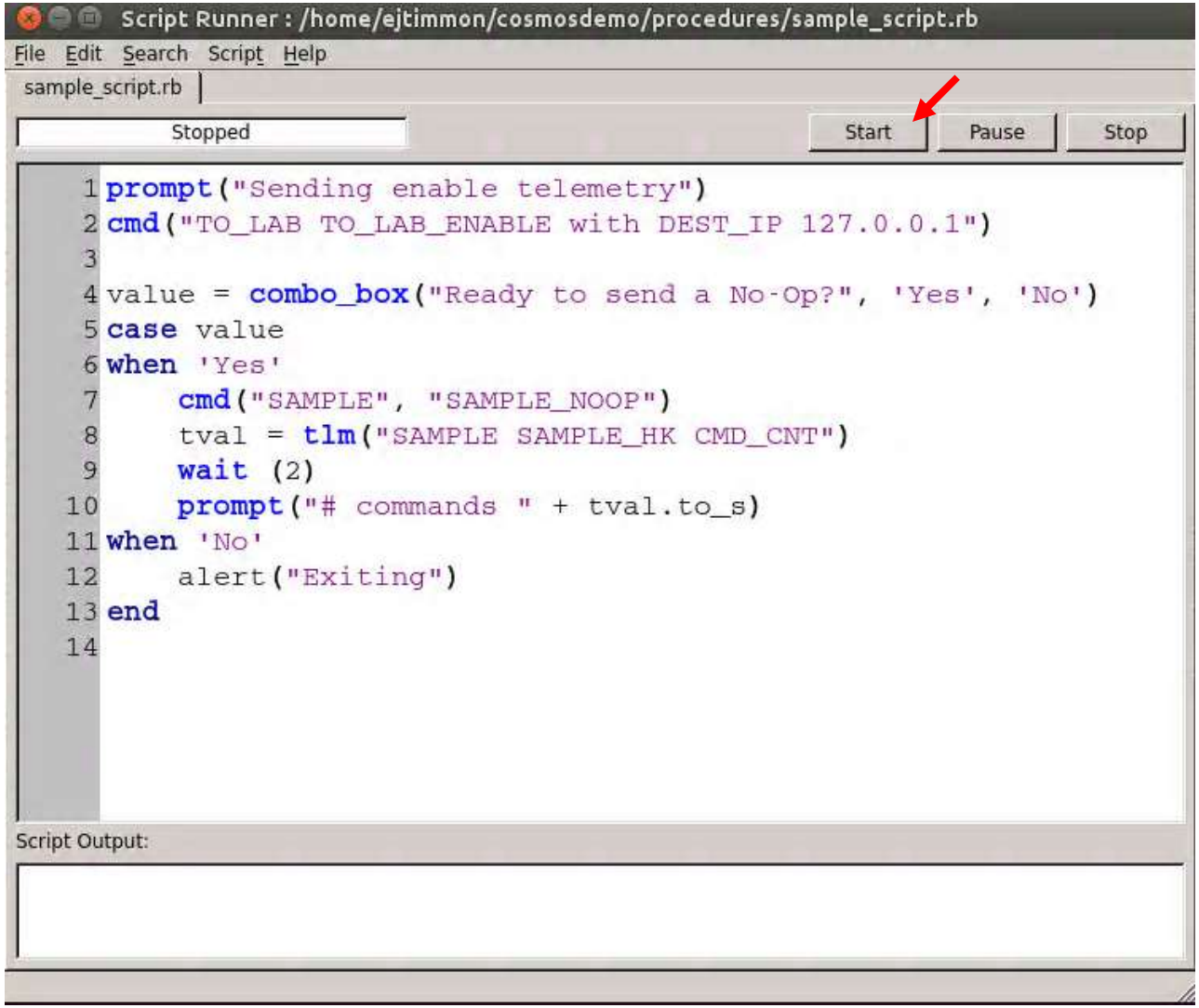
Objective: Write and execute a simple script for the sample_app

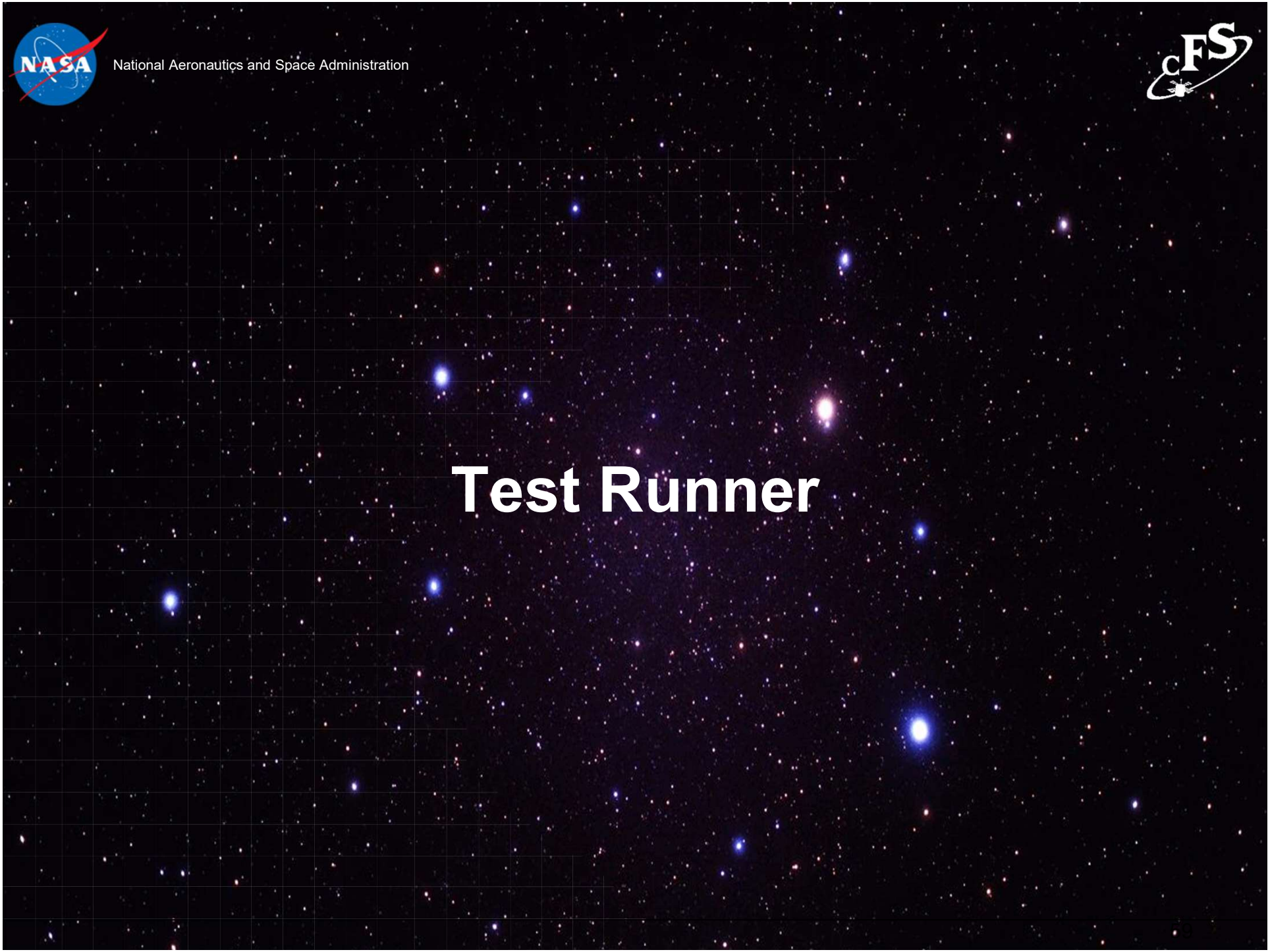
1. Navigate to the cosmos/procedures directory in COSMOS
2. Create a file called sample_script.rb
3. Write a script that sends a sample_app no-op command and receives the housekeeping telemetry
4. Enter the main Cosmos directory and launch COSMOS with “ruby Launcher”
5. On the Launcher window, click 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

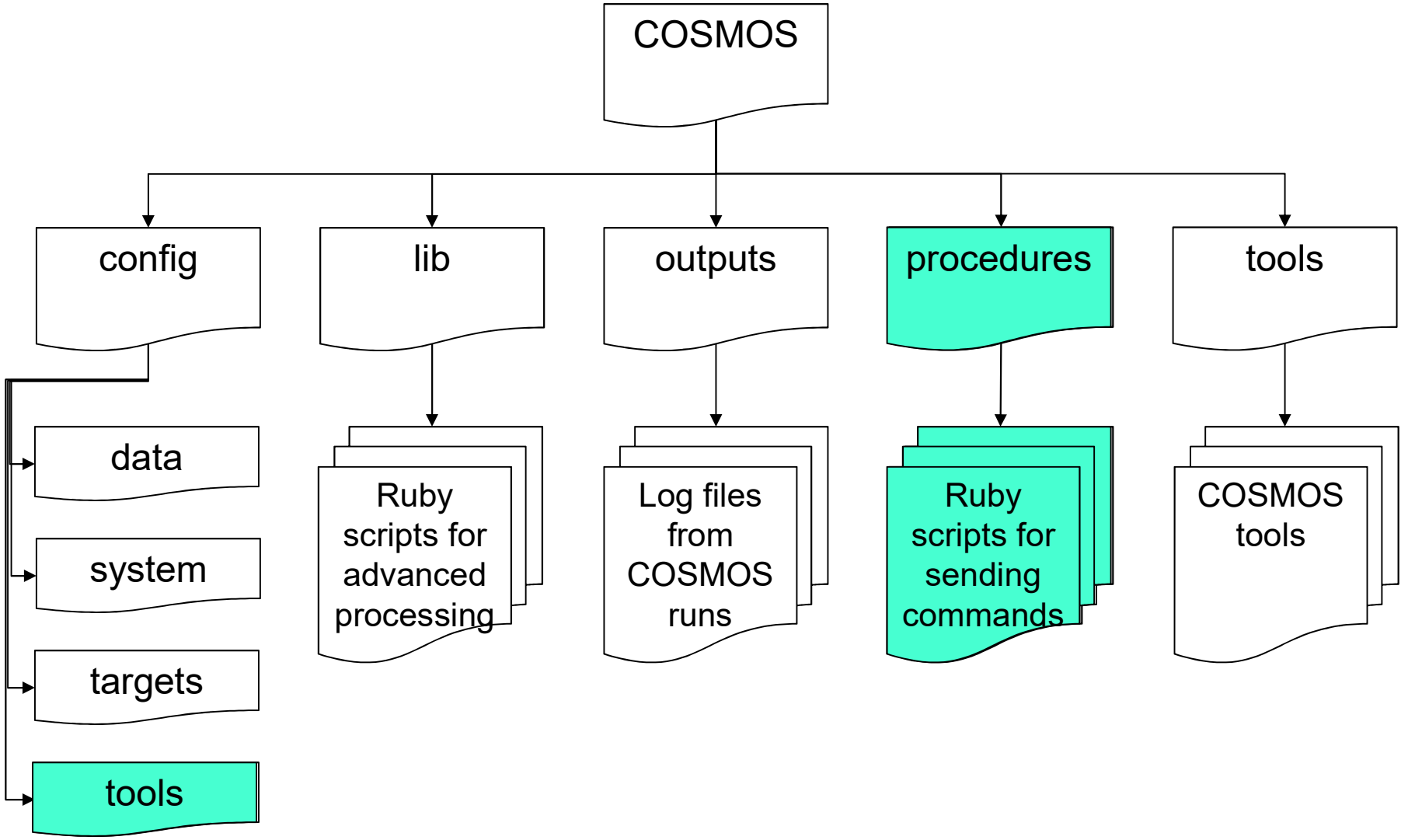



National Aeronautics and Space Administration
Test Runner

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



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

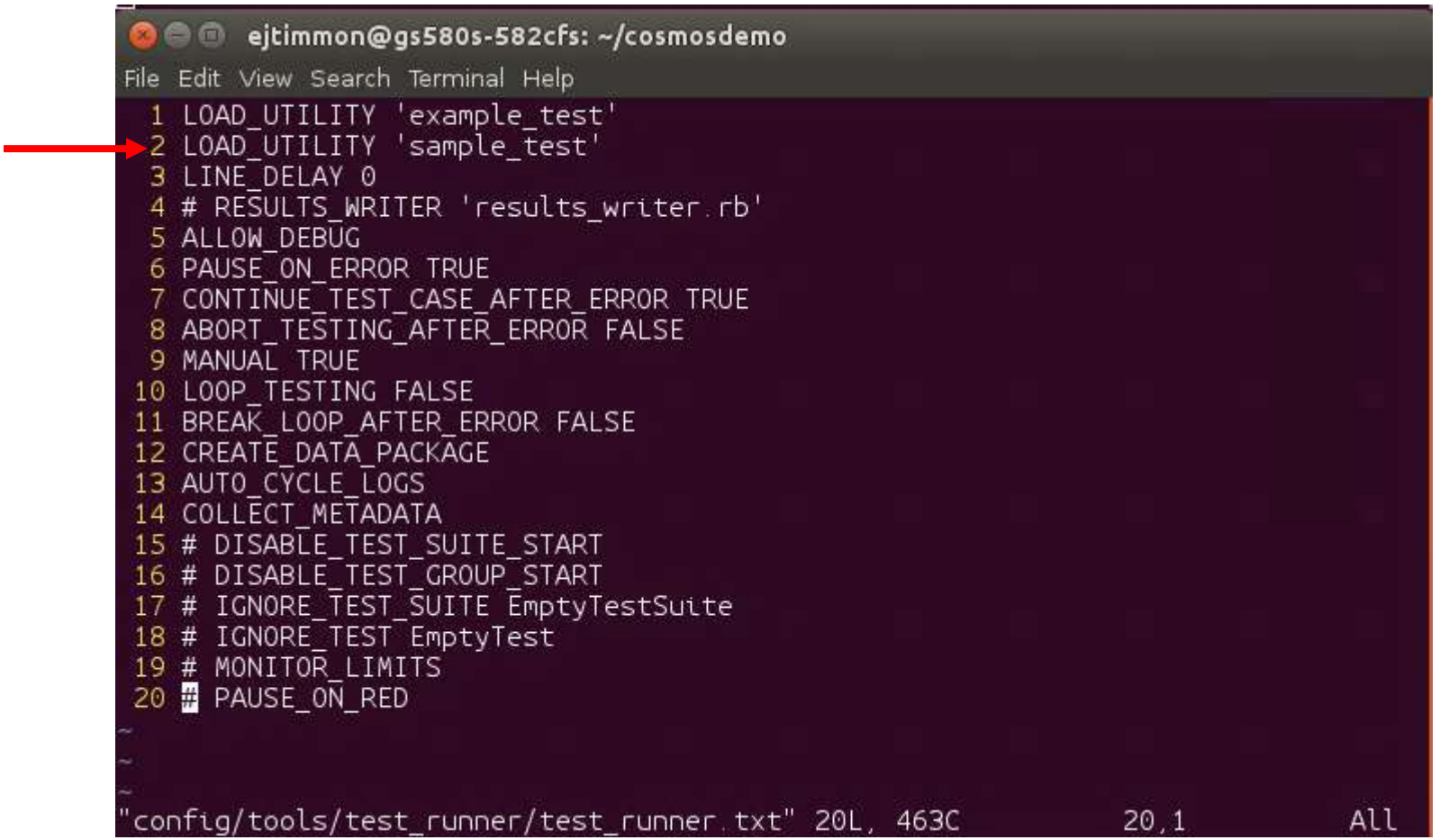
1. Navigate to the cosmos/procedures directory in COSMOS
2. Create a file called sample_test.rb
3. Write a test that sends a sample_app no-op command, receives the housekeeping telemetry, and verifies that the sample_app command counter incremented by 1
   - This script should have a test suite and a 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”

Exercise 6 Continued

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



Exercise 6 - Recap

sample_test.txt

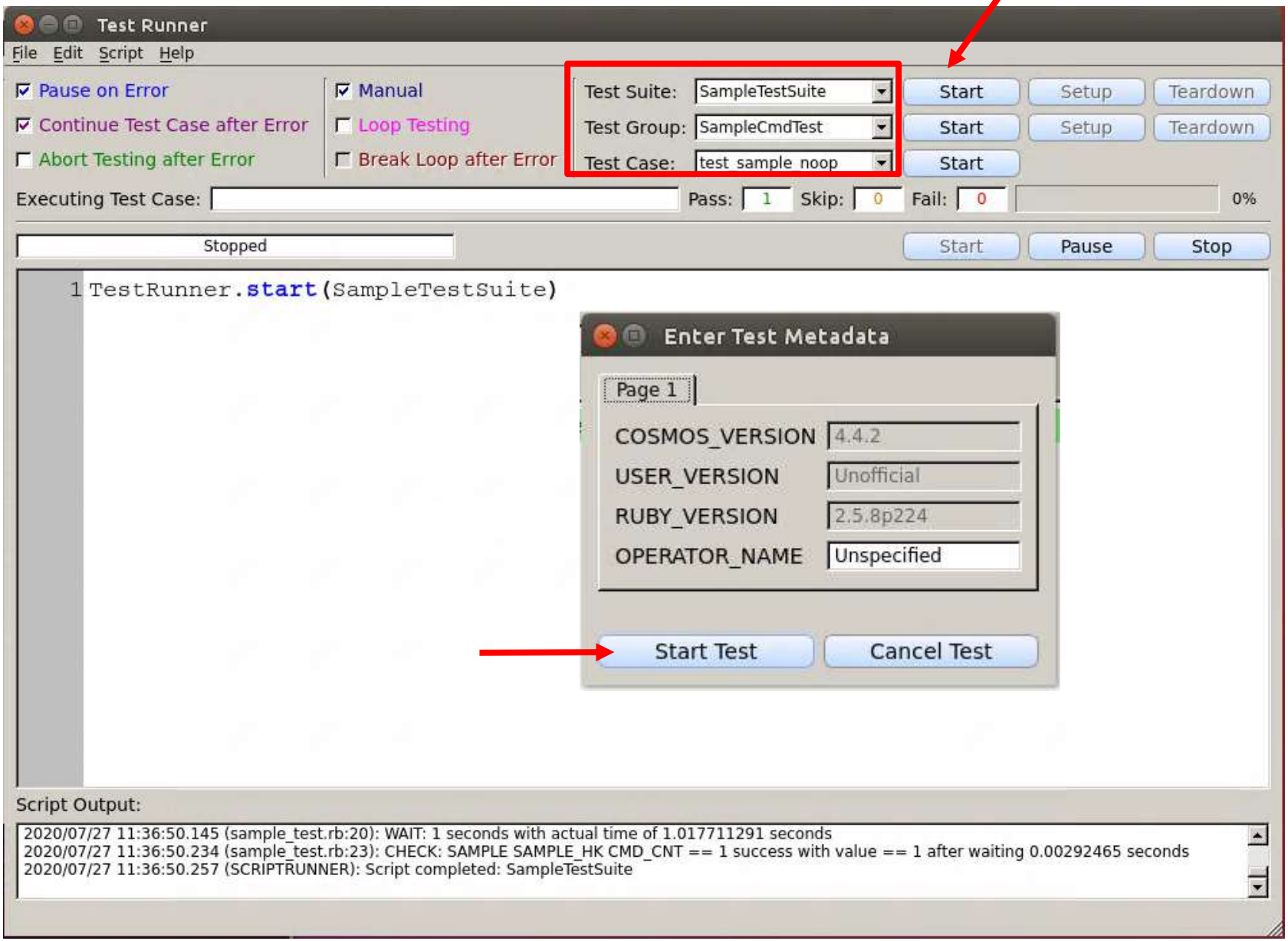
Creates a Test Group to which individual test cases can be added

Creates a single Test Case.
Note that test cases must start with "test"

Creates a Test Suite and adds one Test Group to it

```
ejtimmon@gs580s-582cfs: ~/cosmosdemo
File  Edit  View  Search  Terminal  Help
 1 load 'cosmos/tools/test_runner/test.rb'
 2
 3 class SampleCmdTest < Cosmos::Test
 4
 5     def initialize
 6         super()
 7     end
 8
 9     def test_sample_noop
10         puts "Running #{Cosmos::Test.current_test_suite} #{Cosmos::Test.current_test} #{Cosmos::Test.cu
   rrent_test_case}"
11         cmd("TO_LAB TO_LAB_ENABLE with DEST_IP 127.0.0.1")
12
13         puts "Getting intial command count"
14         wait(2)
15         initcnt = tlm("SAMPLE SAMPLE_HK CMD_CNT")
16         wait(1)
17
18         puts "Sending no-op command"
19         cmd("SAMPLE", "SAMPLE_NOOP")
20         wait(1)
21
22         expcnt = initcnt + 1
23         wait_check("SAMPLE SAMPLE_HK CMD_CNT == #{expcnt}", 5)
24
25
26     end
27 end
28
29 class SampleTestSuite < Cosmos::TestSuite
30     def initialize
31         super()
32         add_test('SampleCmdTest')
33     end
34 end
~
~
~
~
                                                                    34,4          All
```

Exercise 6 – Recap



Exercise 6 - Recap