OpenSatKit COSMOS Guide

OSK v3.3

COSMOS 4.x

Outline

1. Introduction
2. OSK COSMOS Organization & Architecture
3. Command & Telemetry Definitions
4. Screen Definitions
5. FSW Table Management
6. Scripting

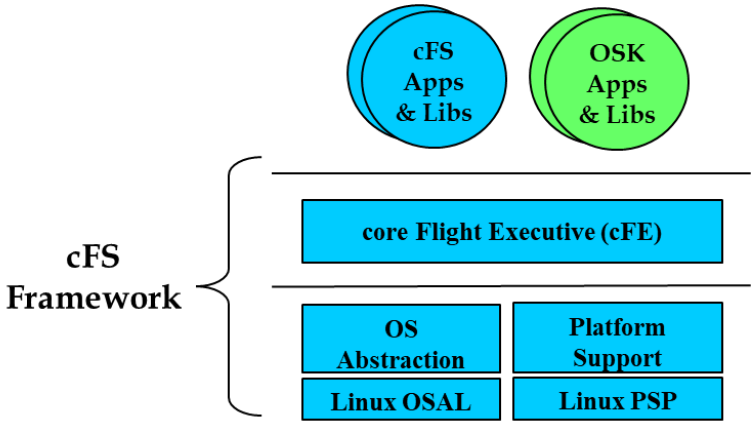
Introduction

- These slides describe how COSMOS has been configured and extended for OpenSatKit (OSK)

- OSK uses COSMOS to send commands to the cFS, receive telemetry from the cFS, and provide a user interface that includes screen, scripts and COSMOS tools.
  - OSK can optionally run with the 42 simulator which is not discussed in this document
- OSK versions 3.* use COSMOS versions 4.* and documentation can be found at https://cosmosc2.com/docs/v4/
  - In 2021 Ball Aerospace released COSMOS 5.0 which is an architectural shift from a local application to a containerized service
- OSK's design evolved as OSK's objectives changed and as COSMOS feature evolved and were better understood
- OSK could benefit from some refactoring but that is unlikely with COSMOS 4.* since it has a limited life due to some of its dependencies on an old version of Ruby and Ruby QT graphics

cFS Terminology

- **<u>cFS Distribution</u> – A cFS development and/or runtime environment that includes the cFE Framework and suite of local and/or remote apps that can be included in building and deploying one or more cFS target**
  - OSK is a cFS distribution
- **<u>cFS Target</u> – The cFE Framework configured and built for a particular platform along with a suite of apps**
  - OSK contains 4 targets:
    - **cfsat:** A cFS educational platform
    - **simsat:** A complete cFS application suite for a reference mission called Simple Sat
    - **pi-sat:** A Raspberry Pi platform for STEM education and hobbyist
    - **sandbox:** A general cFS application Research & Development (R&D) platform



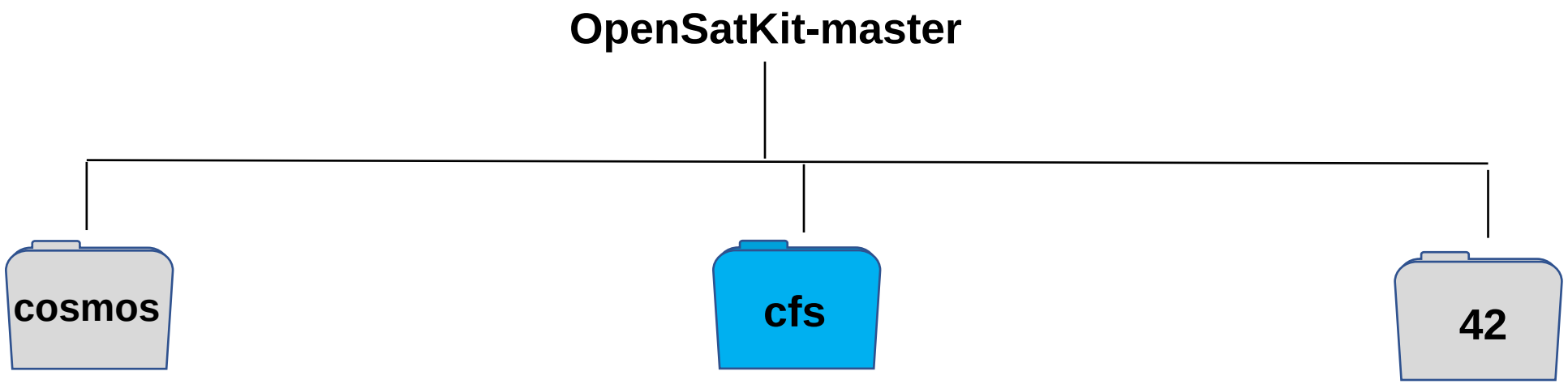
- **<u>cFS Framework</u> – The operating system abstraction and core flight executive (cFE) layers of the cFS architecture**
  - OSK targets use the cFS Linux platform
- **<u>cFS App</u> – An application that is designed to build and run on the cFE Framework**
  - OSK use a combination of open source and custom apps

COSMOS Terminology

- **<u>COSMOS Target</u> – Architectural component that define a destination (local or remote) for commands and/or a source of telemetry**
  - OSK defines one COSMOS target for each cFS apps
- **Targets can also be a mechanism for organizing resources (screens, scripts, etc.) to create a new architectural component**
  - Conceptually these can be though of as virtual targets
  - OSK defines one COSMOS target for each cFS target
  - OSK's *cfs_kit* serves as the application's entry point
- **<u>COSMOS Tool</u> – A standalone COSMOS application with a GUI**
  - Each tool can be started from the COSMOS "Launcher"
  - Tools use configuration files defined in cosmos/config/tools
  - There are interdependencies between some tools. E.g., *Command Sender* must have the *Command and Telemetry Server* running in order for commands to be sent to a target
- **<u>COSMOS Interface</u> – Architectural component used to communicate with a target**
  - Interfaces are managed by the *Command and Telemetry Server*

OSK COSMOS
Organization & Architecture

OSK Top-level Directories

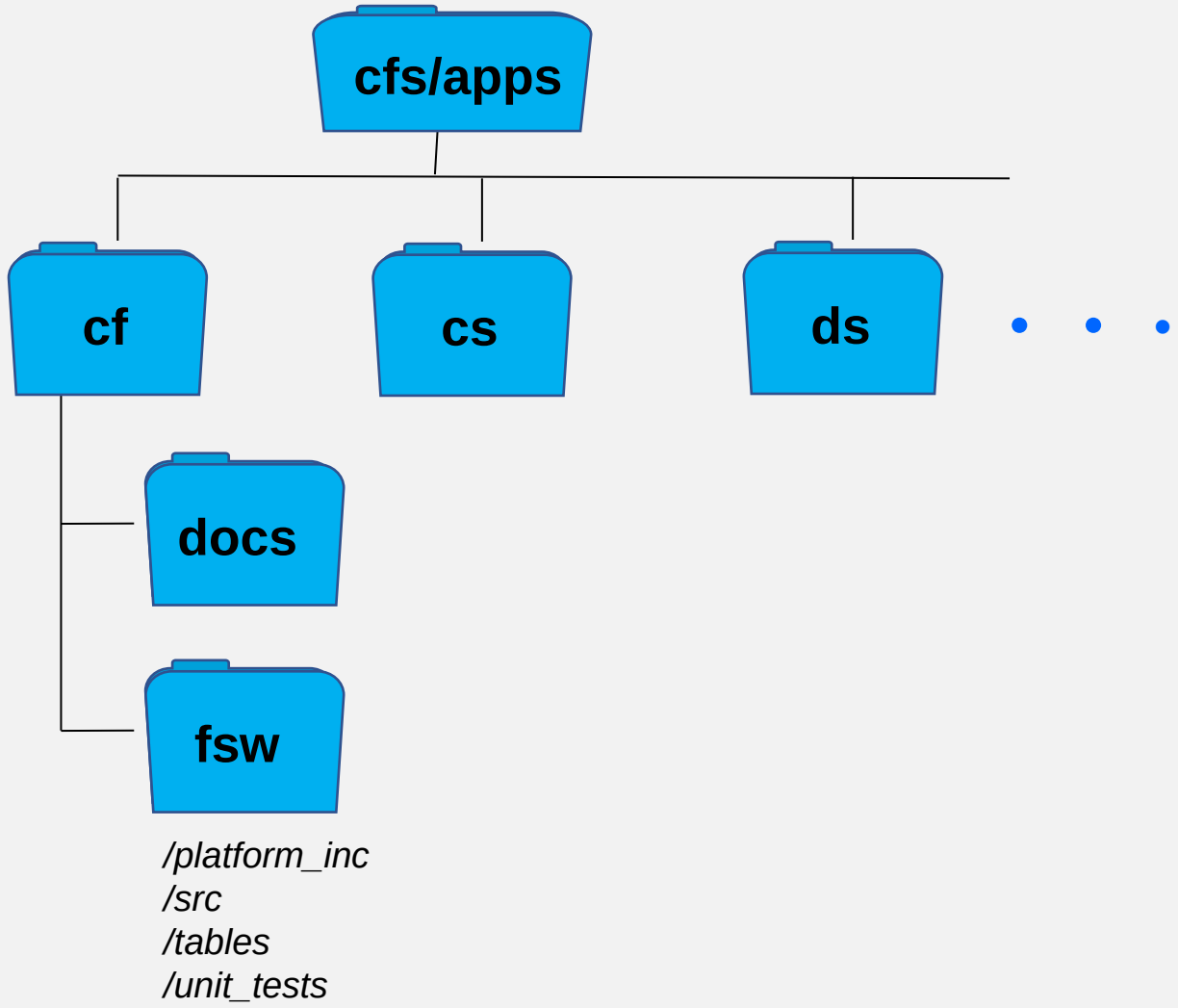


- Original directory structure preserved
- Many customizations and extensions made to make OSK perform more like an application

- Maintains the directory structure used by the *cFS Bundle*
- Additional cFS targets and apps added to the bundle

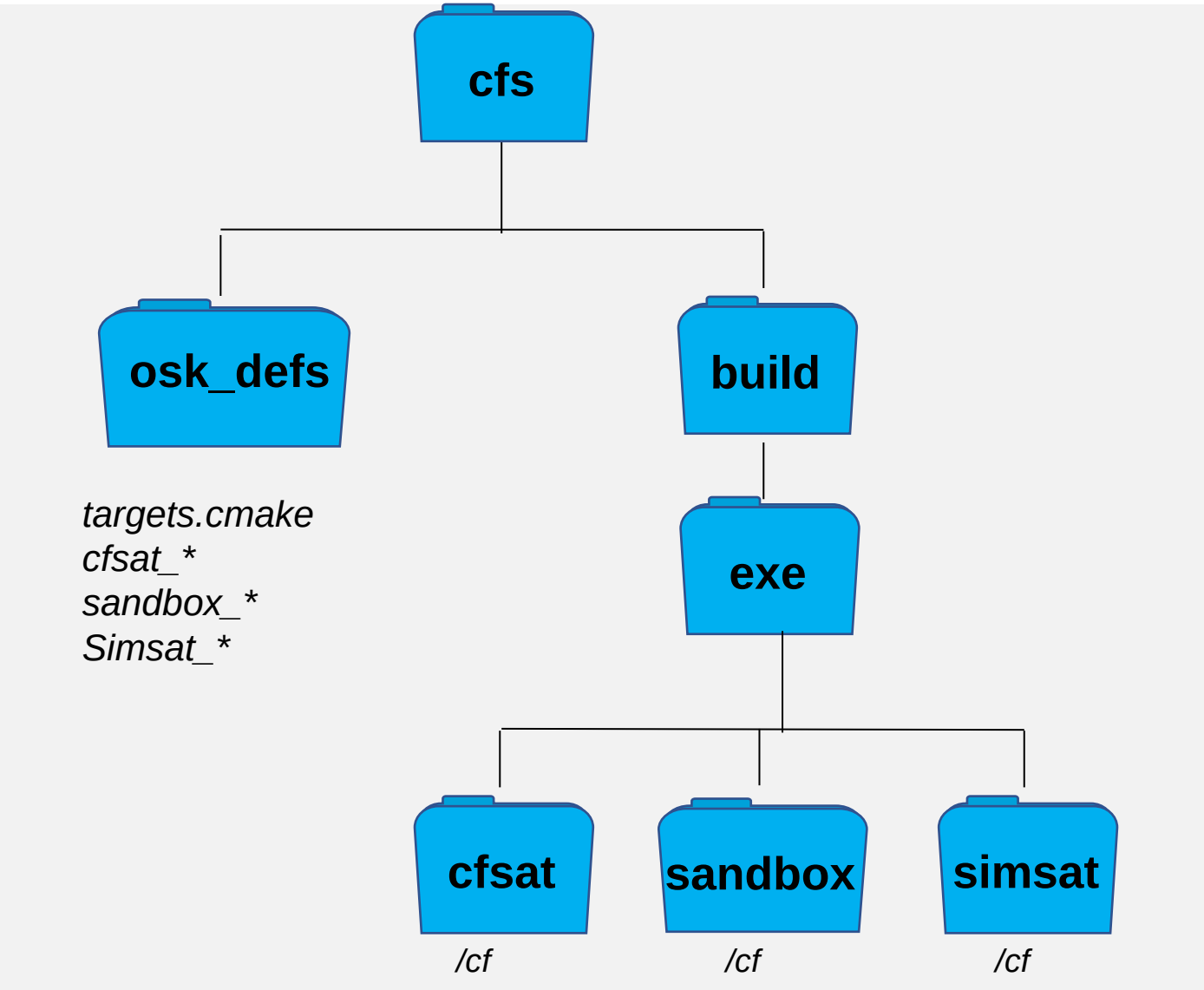
- Added an OSK simulation configuration file used for the closed-loop control example
- This version of the document does not cover the 42 configuration details

Flight Software Apps



- One directory for each app

Flight Software Targets



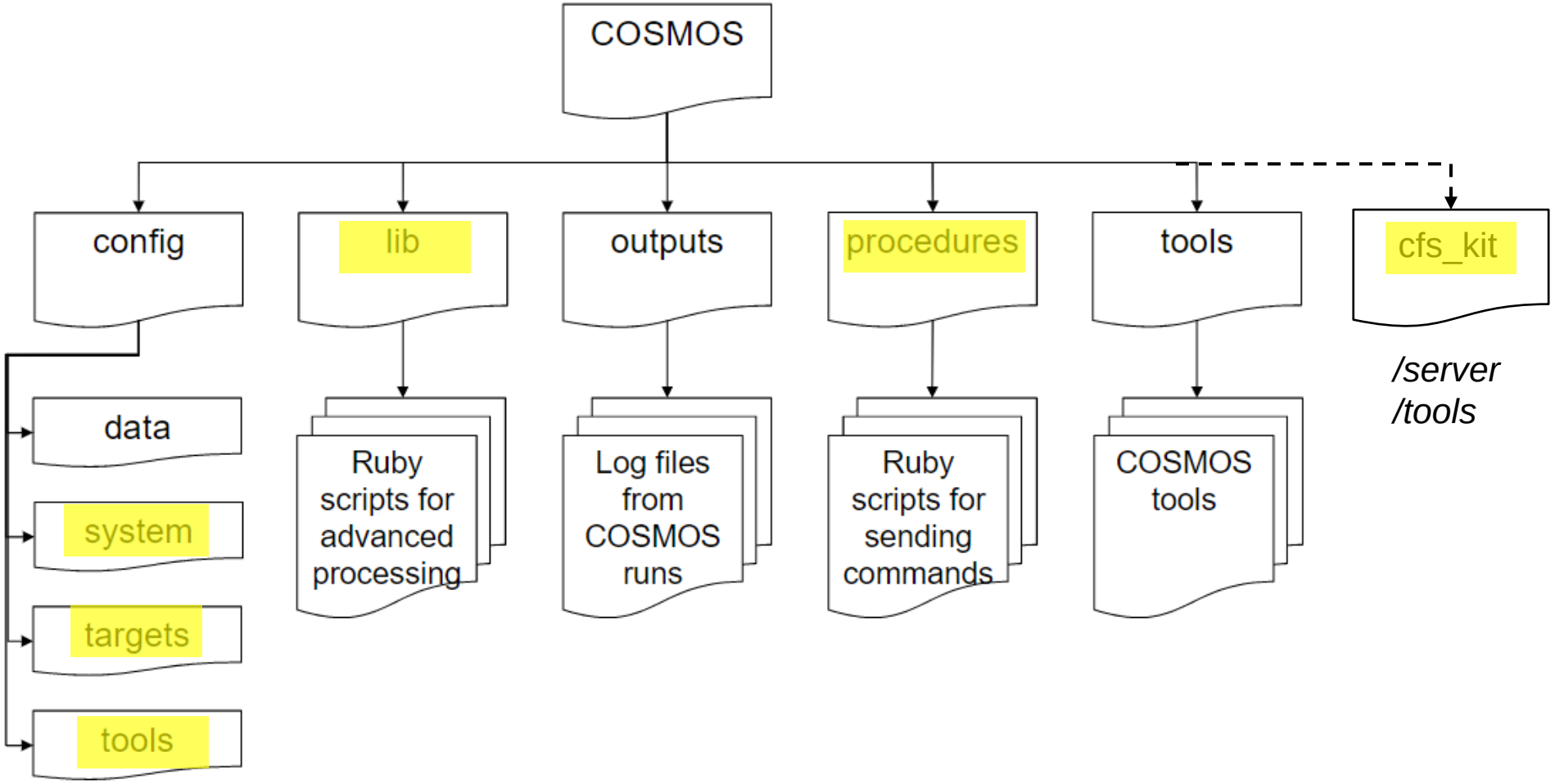
- **The build directory tree is created during the cFS build process**
- **Each target has a *cf* (compact flash) subdirectory**

Targets

- **cfsat**
  - cFS FSW Education
- **pi-sat**
  - Separate repo not shown
  - https://github.com/OpenSatKit/pi-sat
- **sandbox**
  - Research & Development
- **simsat**
  - Mission FSW

COSMOS Directory Structure

```
.
├── Gemfile
├── Launcher
├── Launcher.bat
├── Rakefile
├── config
│ ├── dart (optional)
│ ├── data
│ ├── system
│ ├── targets
│ │ ├── TARGET
│ │ │ ├── cmd_tlm
│ │ │ ├── lib
│ │ │ ├── procedures
│ │ │ ├── screens
│ │ │ ├── sequences
│ │ │ ├── tables
│ │ │ ├── tools
│ │ │ │ ├── table_manager
│ │ │ │ └── ...
│ │ │ ├── cmd_tlm_server.txt
│ │ │ └── target.txt
│ │ └── ...
│ └── tools
│ │ ├── cmd_tlm_server
│ │ └── ...
├── lib
├── outputs
│ ├── dart (optional)
│ ├── handbooks
│ ├── logs
│ ├── saved_config
│ ├── sequences
│ ├── tables
│ └── tmp
├── procedures
├── tools
│ ├── mac
│ └── ...
```



Key directories with OSK customizations

cfs_kit is not part of the COSMOS directory structure

cosmos/config/targets

- **OSK uses COSMOS targets for the following purposes**
  - Each FSW app has its own target
  - Each OSK cFS target (use case) has its own target (CFSAT, SIMSAT, PISAT, SANDBOX)
- **App targets contain some or all the following subdirectories:**
  - cmd_tlm: Command & Telemetry definition files
  - docs: Documents specific
  - lib: App specific ruby scripts defining constants and functions that can be used by screens and test procedures
  - osk[1]: OSK definition files
  - procedures: Test procedures
  - screens: App specific screens. Screens that contain information from more than one app are defined in distribution targets
  - test_files[1]: Files used by test procedures
- **cFS Distribution Targets**
  - cmd_tlm: PISAT defines its remote manager interface. Unused by CFSAT, SIMSAT, and SANDBOX.
  - The remaining directories have similar contents as the app targets except they’re scope is at a cFS systems level

1. OSK specific directories that are not part of the COMOS hierarchy

cosmos/config/system

- Configures COSMOS to auto discover targets
- Defines search paths for OSK extensions

cosmos/config/tools

- **Launcher**
  - Imports osk_system to force OSK ruby libraries to be loaded when the Launcher is started
  - Not sure whether this is the preferred or most elegant method
  - See later “Start up Sequence” slide

- **Table Manager**
  - Contains binary table and binary file format definition files

- **Test Runner**
  - Imports simsat and sandbox functional test suites at startup

cosmos/config/lib

- **Some principles behind the lib extensions**
  - Single definitions for flight and ground software definitions
    - E.g., File Manager's directory listing telemetry packet has a configuration parameter that defines how many files are listed in the packet. This definition impacts the telemetry packet definition and potential screens and ruby scripts that use the telemetry packet.
  - Consistency helps enforce good practices, easier to share code, etc.
    - E.g., Verifying command valid and invalid counters after each command is issued
  - Interface abstraction Dynamic environment and
- **TODO**
  - Describe groups of functions and files
  - Provide some API details to show what functions in which files so user's can go to the file for latest API documentation

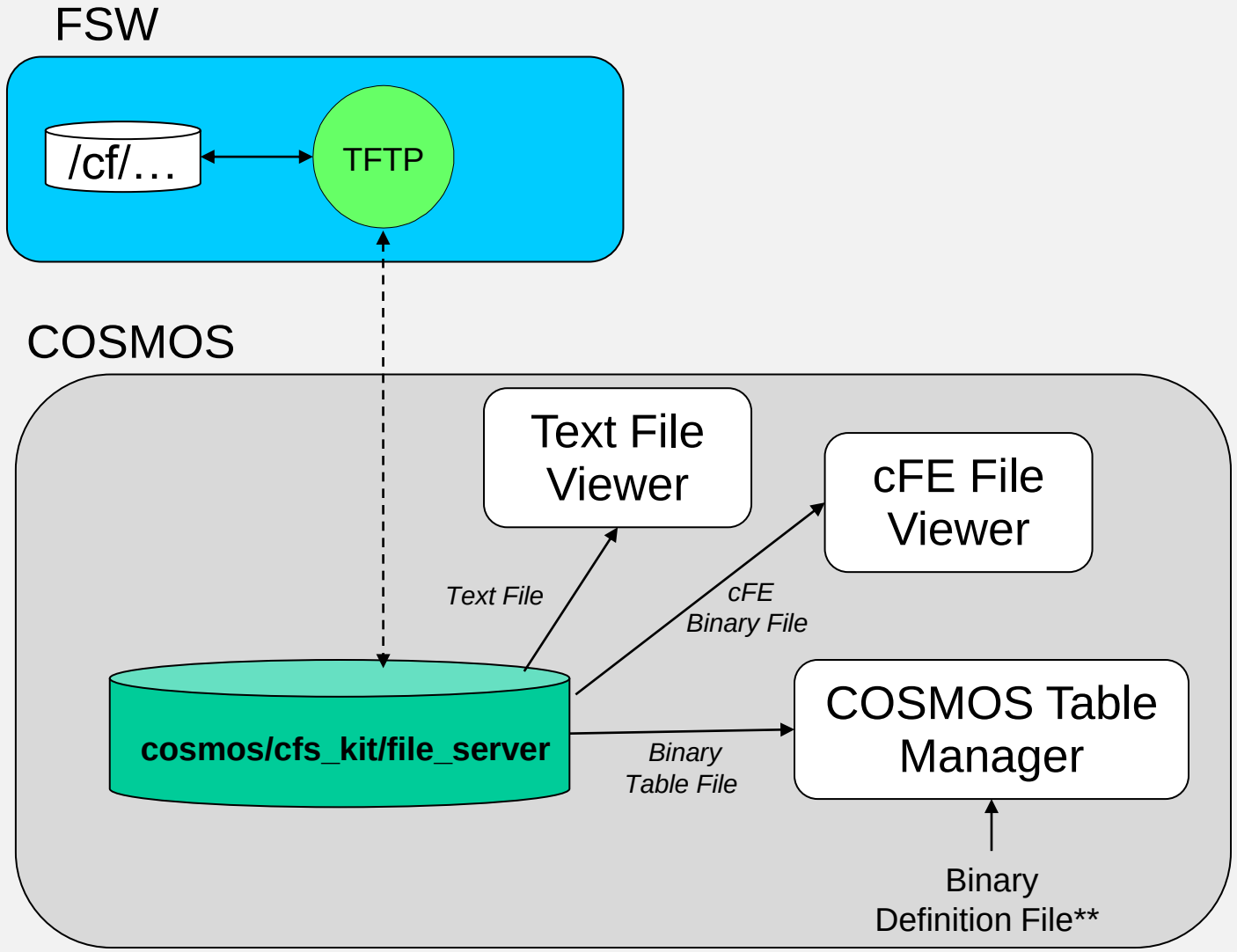
cosmos/config/procedures

- **TODO**

cosmos/config/cfs_kit

- **/tools contains non-ruby tools**
  - Create app is an OSK specific tool that generates apps from templates
  - Performance monitor is a NASA/JSC tool that graphically displays cFE Executive Service performance monitor data files
- **/file_server**
  - Base default directory for files transferred between flight and ground
  - Subdirectories created as needed to help organize files
    - E.g., Simsat and tables

File Management



** Definition files in ~/cosmos/config/tools/table_manager

OSK Message Definition Files

Manual process to keep files in synch

Core Flight System

cfs/osk_defs/

cpu1_osk_sch_msg_tbl.json

cpu1_osk_to_pkt_tbl.json

cfs/apps/xx/fsw/platform_inc/

xx_msgids.h

Long term goal is for a single message definition location and automatic file generation for all dependencies

COSMOS

cosmos/config/targets/XX/cmd_tlm/

cosmos/lib

fsw_msg_id.rb

xx_cmd.txt

xx_tlm.txt

Ruby Modules

- The use of Ruby modules evolved as OSK matured.
- A top-down strategy and guidelines need to be established
- Current Status
  - ‘Osk’ module used to contain OSK extensions
  - ‘Fsw’ module contains definitions that correspond to FSW header definitions.

OSK Startup Sequence

- Describe osk.json files and application environment
-

Background Tasks

- What are they and why do you need them?
- Where and how do you define them
- CFDP and TFTP examples
-

OSK Conventions

- **Most cFE services have commands that can generate a telemetry as part of the response or write information to a file**
  - The verbs ***list*** and ***send*** indicate information is sent in a telemetry packet.
  - ***Write*** is used when information is written to a file
- **The FSW directory /cf (compact flash) is used as the default location for onboard file creation and flight-ground file transfers**
  - This is mapped to ***OpenSatKit/cfs/build/exe/cpu1/cf***
- **OpenSatKit/cosmos/cfs_kit/file_server is used as the default ground file location**
  - Table are located in the ***tables*** subdirectory
- **OSK often uses osk_tmp_bin.dat as a standard temporary binary file name to avoid clutter**
- **OSK does not "cheat" when working with ground and flight tables**
  - **Files are transferred between flight and ground locations and not accessed via shared locations within the VM**

Command & Telemetry Definitions

COSMOS Command & Telemetry Targets

- **COSMOS targets define command and telemetry packets in their *cmd_tlm* directory**
  - ~/cosmos/config/targets/XX/cmd_tlm/

- **Targets use text files to define command and telemetry packets**
  - xx_cmd.txt, xx_tlm.txt where ‘xx’ is the target name
  - Simple COSMOS-specific “language” used to define packet content
  - Allows Embedded Ruby (ERB) defined between ‘<%’ and ‘%>’ delimiters to augment the language

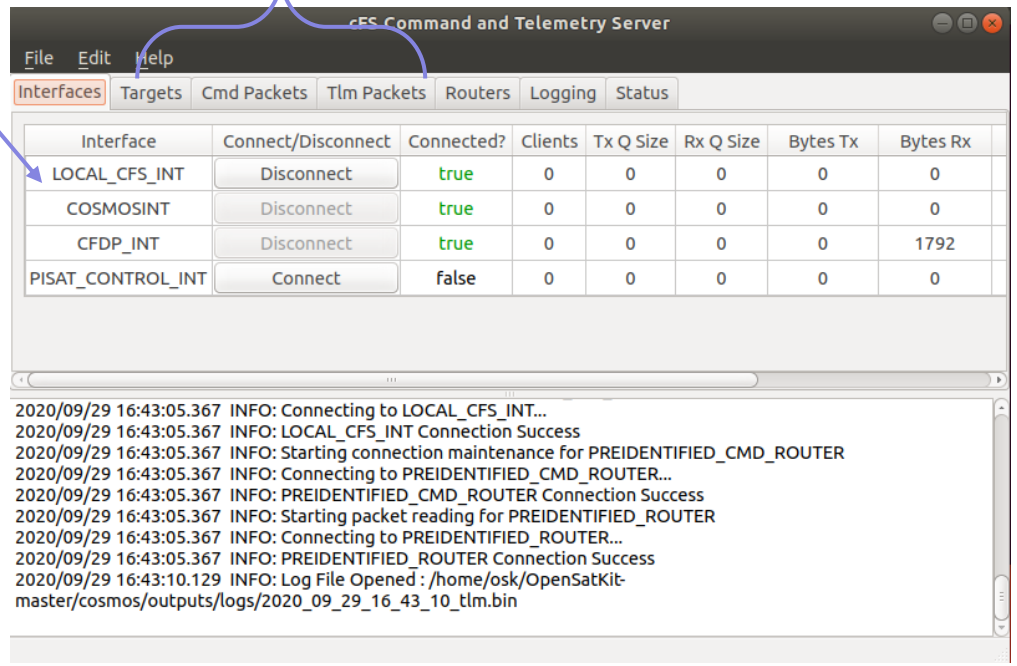
- **COSMOS does not have any predefined packets, so CCSDS command and telemetry packets must be defined by OSK**

COSMOS Command & Telemetry Interfaces

- **COSMOS use *interfaces* to communicate with a remote system**
- **Interfaces are defined and managed by the COSMOS Command and Telemetry Server**
  - Interfaces specify which targets are managed by the interface
  - Interfaces are defined in ~/cosmos/config/cmd_tlm_server/cmd_tlm_server.txt

```
INTERFACE LOCAL_CFS_INT udp_cs_interface.rb 127.0.0.1 1234 1235 nil nil 128 nil nil
TARGET CFE_ES
TARGET CFE_EVS
TARGET CFE_SB
TARGET CFE_TBL
TARGET CFE_TIME
TARGET CF
...
```

Access to targets



OSK Command & Telemetry Definition Principles

- **Packet structures used in multiple packets should only be defined once**
  - Ensure consistent packet field names
  - If structure depends on a parameter, then it only needs to change in one place
  - If structure changes during development, then it only changes in one place

- **FSW definitions that impact ground software definitions should only be defined once**
  - CCSDS Message IDs
  - Application configuration parameters

- **The remote target endianness should only be defined once**

- **Hide implementation details from configuration users**
  - Long term goal is to use a toolchain to generate both the flight and ground system artifacts from a single source definition

OSK Command & Telemetry Definition Strategy

- **osk_config.rb**
  - Serves as a single API to access configuration definitions and FSW definitions that are used by ground elements
- **The following definitions are accessed through osk_config.rb**
  - hw_target.rb
    - Processor endian type, CPU address size, Interface target ID
  - ccsds.rb
    - Command header, telemetry header, CFDP PDUs
  - cfe_file.rb
    - cFS binary file headers
  - fsw_config_param.rb
    - Replicates parameters defined in FSW "_cfg.h" files and OSK app JSON files
  - fsw_msg_id.rb
    - Replicates messages IDs defined in FSW "_msgids.h" and OSK app JSON files
- **osk_global.rb**
  - "Standard" command names to ensure consistency

OSK App Target Command File

~/cosmos/config/targets/CFE_EVS/cmd_tlm/evs_cmd.txt

Defined in osk_config.rb

Function Code

Total parameter data bytes

```
<%
  require 'fsw_config_param'
  require 'osk_config'
  require 'osk_global'

  @APP_PREFIX_STR   = "CFE_EVS"
  @CMD_MID_STR      = "CFE_EVS_CMD_MID"

%>

COMMAND <%= @APP_PREFIX_STR %> <%= Osk::CMD_STR_NOOP %> <%= Osk::Cfg.processor_endian %> "
  <%= Osk::Cfg.cmd_hdr(@APP_PREFIX_STR, @CMD_MID_STR, 0, 0) %>

COMMAND <%= @APP_PREFIX_STR %> <%= Osk::CMD_STR_RESET %> <%= Osk::Cfg.processor_endian %> "
  <%= Osk::Cfg.cmd_hdr(@APP_PREFIX_STR, @CMD_MID_STR, 1, 0) %>

COMMAND CFE_EVS ENA_EVENT_TYPE <%= Osk::Cfg.processor_endian %> "This command enables the
  <%= Osk::Cfg.cmd_hdr(@APP_PREFIX_STR, @CMD_MID_STR, 2, 2) %>
  APPEND_PARAMETER BITMASK  8 UINT MIN_UINT8 MAX_UINT8 0 "Event type bitmask (3..0) = (Cri
  APPEND_PARAMETER SPARE    8 UINT MIN_UINT8 MAX_UINT8 0 "Pad to even byte"
```

OSK App Target Telemetry File

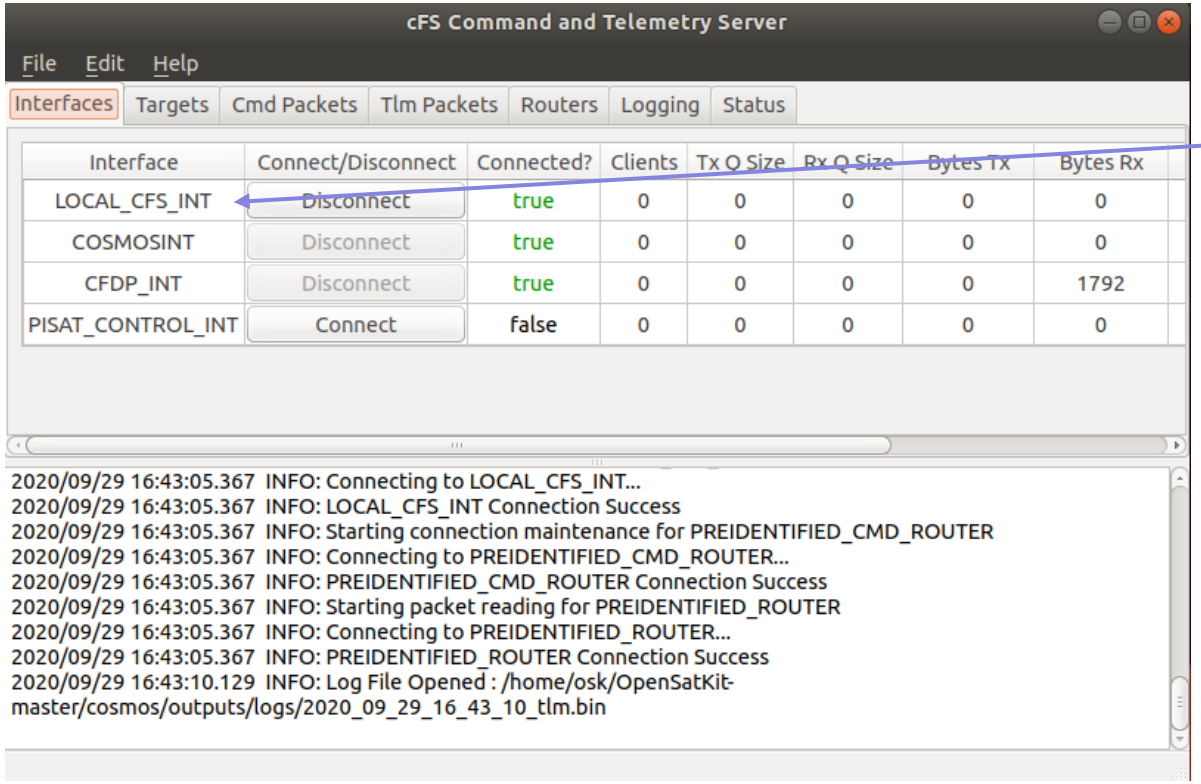
~/cosmos/config/targets/CFE_EVS/cmd_tlm/evs_tlm.txt

Defined in osk_config.rb

```
<%
  require 'osk_config'

  @APP_PREFIX_STR     = "CFE_EVS"
  @HK_TLM_MID_STR     = "CFE_EVS_HK_TLM_MID"
  @EVENT_MSG_MID_STR  = "CFE_EVS_EVENT_MSG_MID"

  #TODO - Add configuration parameters

%>

TELEMETRY CFE_EVS HK_TLM_PKT <%= Osk::Cfg.processor_endian %> "Housekeeping dat
  <%= Osk::Cfg.tlm_hdr(@APP_PREFIX_STR, @HK_TLM_MID_STR) %>
  APPEND_ITEM CMD_VALID_COUNT 8 UINT "EVS Command Counter."
  APPEND_ITEM CMD_ERROR_COUNT 8 UINT "EVS Command Error Counter."
  APPEND_ITEM MSG_FMT_MODE    8 UINT "Event message format mode (short/long)."
    STATE Short 0
    STATE Long  1
```

COSMOS Cmd & Tlm Server

- **The command and telemetry server is automatically started when OSK is started**
  - It reads in the cmd & tlm definitions and stores them in internal COSMOS data structures
- **The cmd_tlm_server.txt file defines which targets are read**



~/cosmos/config/tools/cmd_tlm_server/cmd_tlm_server.txt

```
INTERFACE LOCAL_CFS_INT udp_cs_interface.rb 127.0.0.1 1234 1235 nil nil 128 nil nil
  PROTOCOL READ interfaces/cfdp_test_protocol
  PROTOCOL READ interfaces/cfdp_protocol
  PROTOCOL WRITE interfaces/cfdp_protocol
  PROTOCOL WRITE interfaces/cfdp_test_protocol

  TARGET SYSTEM

  TARGET CFE_ES
  TARGET CFE_EVS
  TARGET CFE_SB
  TARGET CFE_TBL
  TARGET CFE_TIME
```

-
-
-

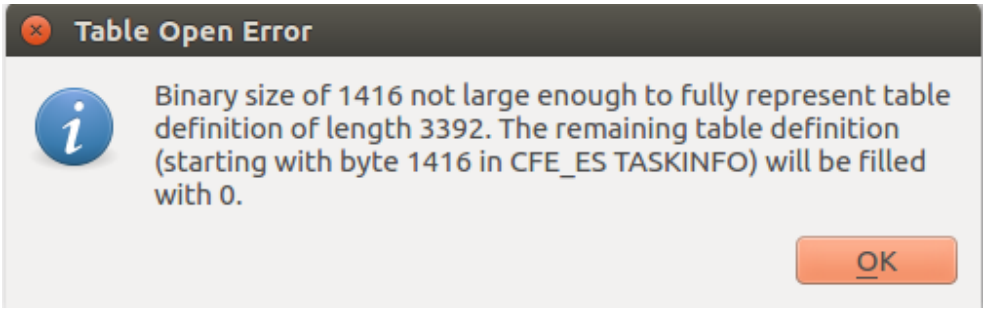
This interface file computes command checksums before a packet is sent

FSW Table Management

- **Summarize table workflows and reference existing examples**

Variable Length Binary Files

- Some cFS binary files are variable length. The Table Manager definition files support fixed length files therefore you may see an error dialog stating the file doesn't contain all of the records. This message is from cFE Executive Service Task Information file.



Screen Definitions

OSK Screen Driven System

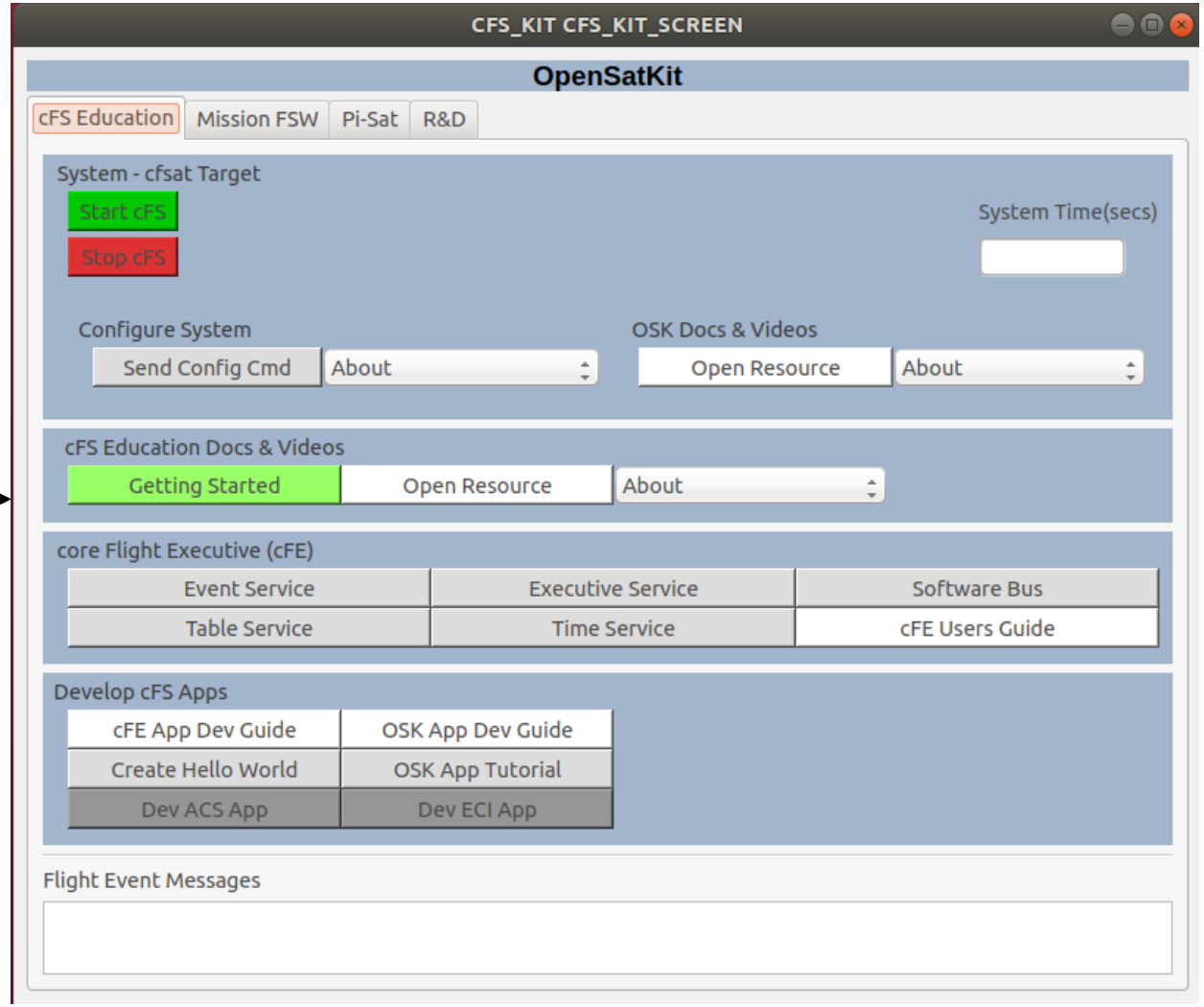
- Four tabs ***cFS Education***, **Mission FSW**, ***PiSat***, and ***R&D*** correspond to four cFS targets/ OSK Use Cases
- Each tab's screen has a similar layout and its own "Getting Started" Guide

User Objective Tab

System/Target Management

Learning Resources

Use Case Specific Content



Viewing Telemetry

Entire CFE_EVS housekeeping packet in PacketViewer

File View Help

Target: CFE_EVS Packet: HK_TLM_PKT

Description: Housekeeping data (general status) autonomously sent

	Item	Value
1	*PACKET_TIMESECONDS:	0.000000
2	*PACKET_TIMEFORMATTED:	No Packet Time
3	*RECEIVED_TIMESECONDS:	0.000000
4	*RECEIVED_TIMEFORMATTED:	No Packet Rec...
5	*RECEIVED_COUNT:	0
6	CCSDS_STREAMID:	0x0000
7	CCSDS_SEQUENCE:	0
8	CCSDS_LENGTH:	0
9	CCSDS_SECONDS:	0
10	CCSDS_SUBSECS:	0
11	CMD_VALID_COUNT:	0
12	CMD_ERROR_COUNT:	0
13	MSG_FMT_MODE:	SHORT
14	MSG_TRUNC_CTR:	0
15	UNREG_APP_CTR:	0
16	OUTPUT_PORT:	0
17	LOG_FULL_FLAG:	FALSE
18	LOG_MODE:	OVERWRITE
19	MSG_SEND_CTR:	0
20	LOG_OVFL_CTR:	0
21	LOG_ENA:	0

Portions of CFE_EVS housekeeping packet in custom screen displayed by TlmViewer

CFE_EVS CFE_EVS_SCREEN

cFE Event Service

Ground Interface

Send Cmd	NOOP
Display Tlm	EVENT_MSG_PKT
Display File	APP_INFO

Learn

Run Tutorial	EVS_TRAINING_VIDEO

Status

Cmd Valid Cnt	0	Cmd Error Cnt	0
Send Count	0	Msg Format	SHORT
Msg Trunc	0	Port Mask	0
Log Ena	0	Log Mode	OVERWRITE
Log Full	FALSE	Log OvFl	0

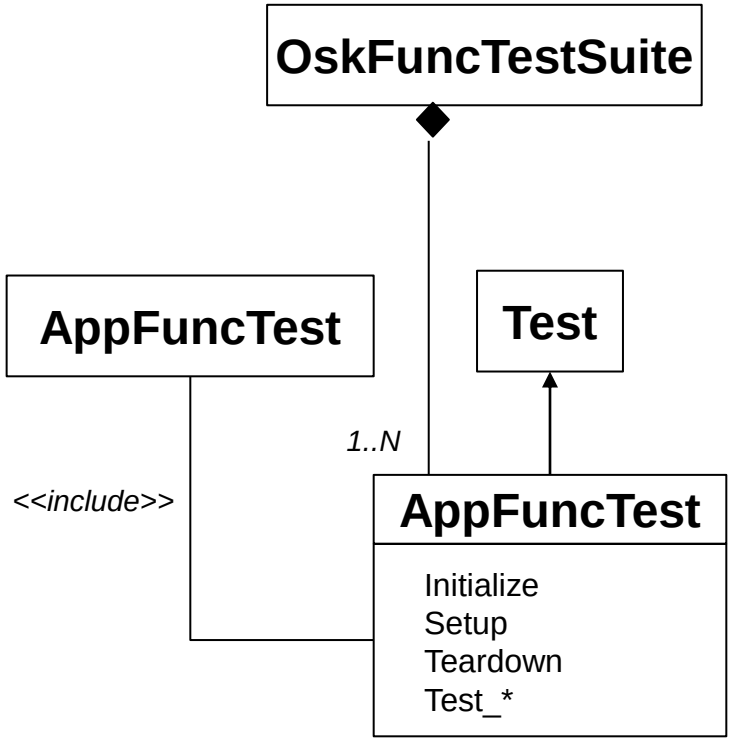
Excerpt from Cosmos/config/targets/CFE_EVS/screens/ cfe_evs_screen.txt

```
TITLE "cFE Event Service"
  SETTING BACKCOLOR 162 181 205
  SETTING TEXTCOLOR black
VERTICALBOX "Ground Interface"
  MATRIXBYCOLUMNS 3 5
    BUTTON 'Send Cmd' 'require
    NAMED_WIDGET cmd COMBOBOX N
    LABEL "    "
    BUTTON 'Display Tlm' 'requi
    NAMED_WIDGET tlm COMBOBOX E
    LABEL "    "
    BUTTON 'Display File' 'requ
    NAMED_WIDGET file COMBOBOX
    LABEL "    "
  END # Matrix
END # Vertical
```

Dynamic Screens

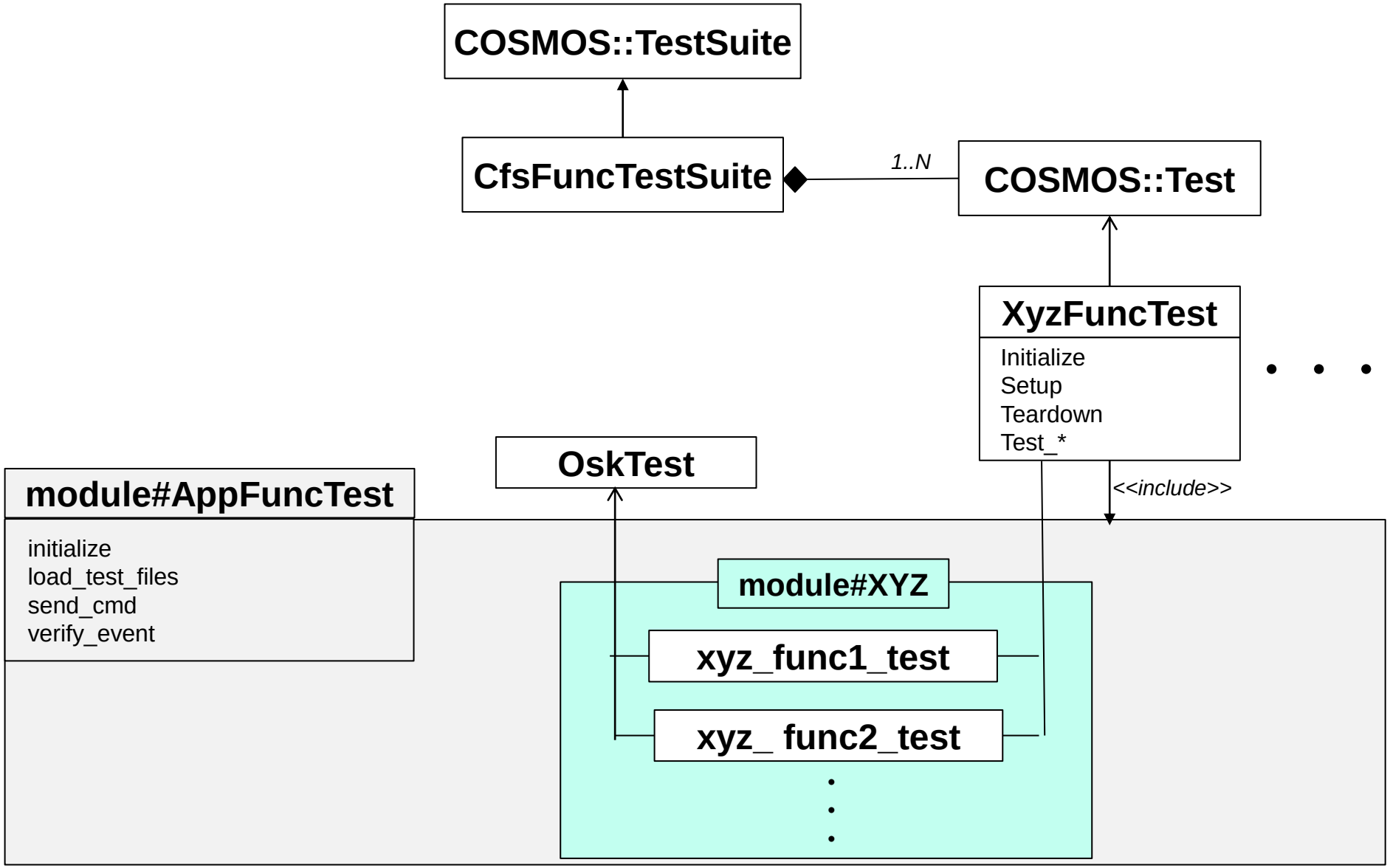
Scripting

Dynamic Screens



- **When should inheritance be used for utilities vs a library with an API or both?**
- **Functional base class could define or do classes and modules provide structure to an API?**

Test Runner

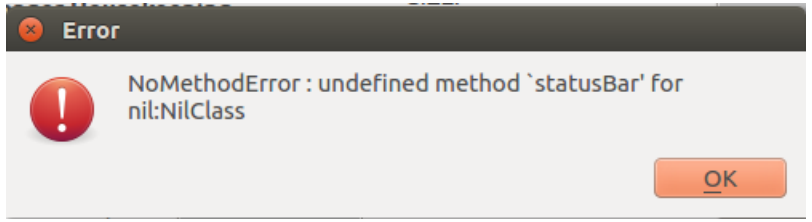

COSMOS::TestSuite
CfsFuncTestSuite
1..N
COSMOS::Test
XyzFuncTest
Initialize
Setup
Teardown
Test_*
OskTest
<<include>>
module#AppFuncTest
initialize
load_test_files
send_cmd
verify_event
module#XYZ
xyz_func1_test
xyz_ func2_test

Appendix A

COSMOS Extras

Occasional COSMOS Error Message

- If you cancel an OSK dialogue you may see the follow COSMOS error dialogue



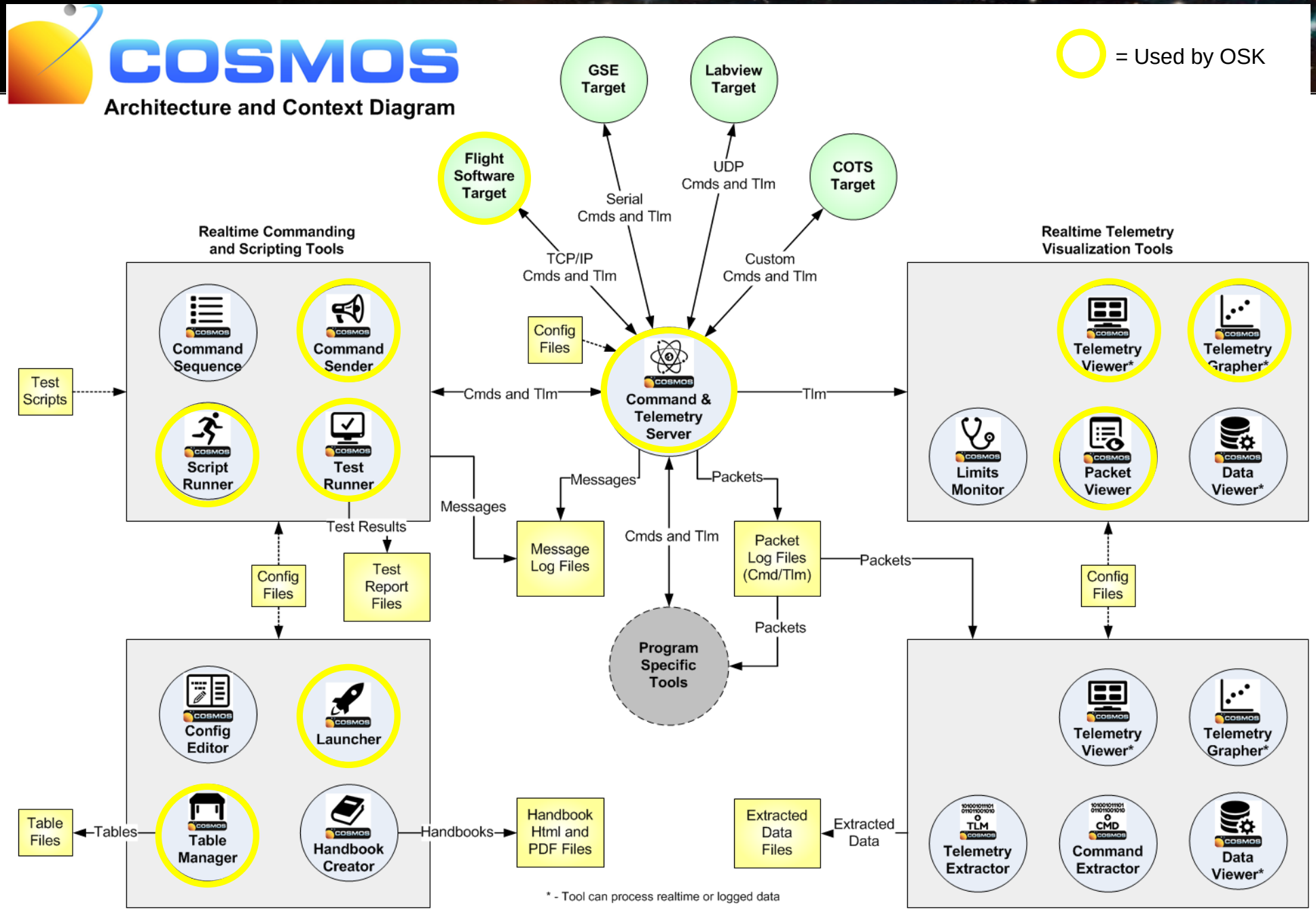
- The issue is very sporadic and could not be resolved with the COSMOS development team

COSMOS Tool Summary (1 of 2)

- **Launcher**
  - Provides a graphical interface for launching each of the tools that make up the COSMOS system
  - *Custom OSK ICON "cFS Starter Kit" launches OSK's main page*
- **Command and Telemetry Server**
  - Connects COSMOS to targets for real-time commanding and telemetry processing.
  - All real-time COSMOS tools communicate with targets through the Command and Telemetry Server ensuring that all communications are logged.
  - Localhost 127.0.0.1 used as cFS connection Targets created
- **Telemetry Viewer**
  - Provides a way to organize telemetry points into custom "screens" that allow for the creation of unique and organized views of telemetry data.
- **Command Sender**
  - Individually send any FSW command using GUI form
  - Raw data files can be used to inject faults
  - *OSK provides custom menus for common cFS commands*
- **Packet Viewer**
  - View any telemetry packet with no extra configuration necessary
  - *OSK provides custom telemetry screens functionally organized*

COSMOS Tool Summary (2 of 2)

- **Telemetry Grapher**
  - Real-time or offline graphing of any FSW telemetry point
  - *OSK provides convenient access through some of its custom screens*
- **Table Manager**
  - Edit and display binary files
  - *OSK provides definitions for most of the cFE binary files and a limited number of cFS application binary files*
- **Script Runner**
  - Develop and execute test procedures using Ruby Scripts and COSMOS APIs
  - *OSK provides additional APIs for functions like file transfer and binary file management*
- **Test Runner**
  - Test framework for organizing, executing, and verifying test scripts
  - *Currently OSK only includes some prototype scripts. The goal is to provide a complete test suite that can be extended by the user.*


COSMOS
Architecture and Context Diagram
= Used by OSK
GSE Target
Labview Target
Flight Software Target
COTS Target
Serial Cmds and Tlm
UDP Cmds and Tlm
TCP/IP Cmds and Tlm
Custom Cmds and Tlm
Realtime Commanding and Scripting Tools
Command Sequence
Command Sender
Script Runner
Test Runner
Test Scripts
Config Files
Command & Telemetry Server
Cmds and Tlm
Tlm
Messages
Packets
Test Results
Test Report Files
Message Log Files
Packet Log Files (Cmd/Tlm)
Program Specific Tools
Realtime Telemetry Visualization Tools
Telemetry Viewer*
Telemetry Grapher*
Limits Monitor
Packet Viewer
Data Viewer*
Config Editor
Launcher
Table Manager
Handbook Creator
Table Files
Tables
Handbooks
Handbook Html and PDF Files
Extracted Data Files
Extracted Data
Telemetry Extractor
Command Extractor
* - Tool can process realtime or logged data
Utilities
Offline Analysis Tools