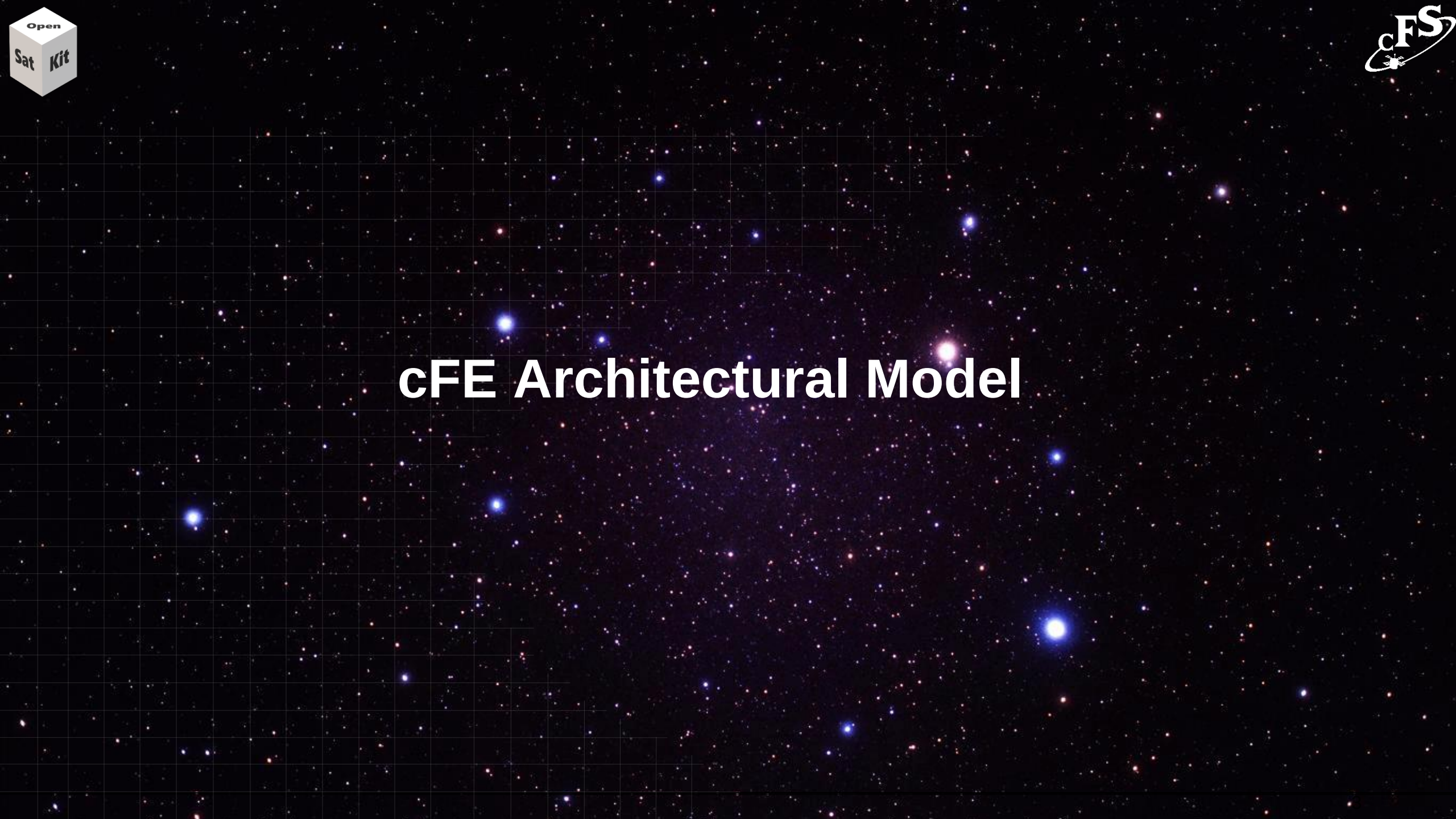
Core Flight Executive Overview

OSK v3.1

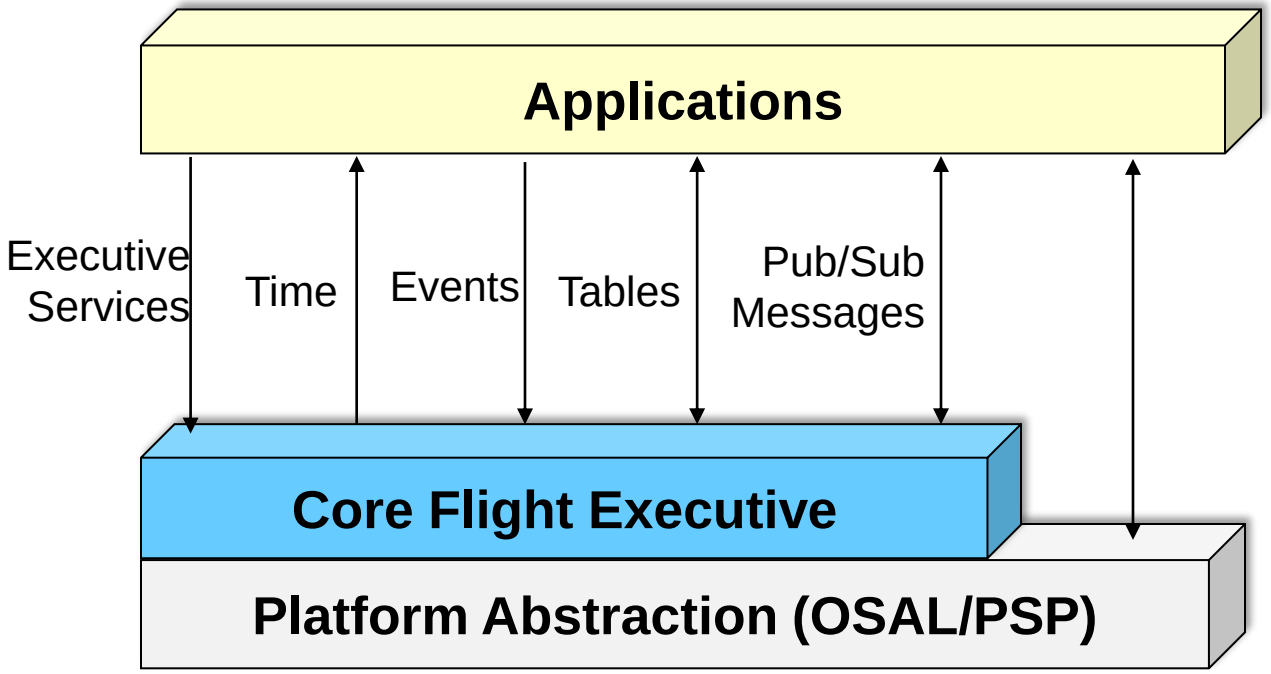
cFE 6.7.1

Introduction

- **Objectives**
  - Describe the core Flight Executive (cFE) from a functional perspective
- **Intended audience**
  - Mostly targeted at software engineers, but systems engineers, non-FSW spacecraft discipline engineers, and technical project managers could also benefit.
- **Perquisites**
  - Introductory material provided in the cFS overview slides and video
- **Slide outline**
  1. cFE architectural model
  2. cFE service functional overview
- **Next steps after this module**
  - Learn details of each individual service
  - Learn to develop applications


cFE Architectural Model

Core Flight Executive Application Services



Executive Services (ES)

- Manage the software system and create an application runtime environment

Time Services (TIME)

- Manage spacecraft time

Event Services (EVS)

- Provide a service for sending, filtering, and logging event messages

Software Bus (SB) Services

- Provide an application publish/subscribe messaging service

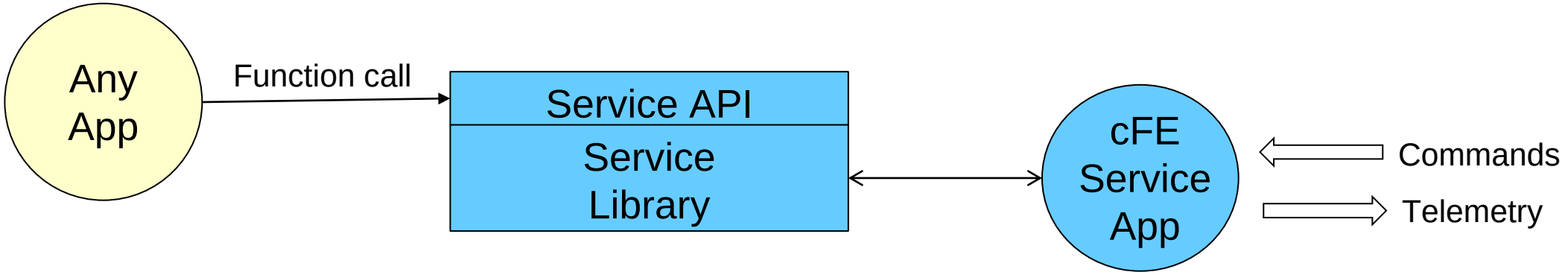
Table Services (TBL)

- Manage application table images

Application-Centric Architecture

- **Applications are an architectural component that owns cFE and operating system resources via the cFE and OSAL Application Programmer Interfaces (APIs)**
- **cFE Services provide an Application Runtime Environment**
- **Resources are acquired during initialization and released when an application terminates**
  - Helps achieve the architectural goal for a loosely coupled system that is scalable, interoperable, testable (each app is unit tested), and maintainable
- **Concurrent execution model**
  - Each app has its own execution thread and apps can spawn child tasks
- **The cFE service and Platform Abstraction APIs provide a portable functional interface**
- **Write once run anywhere the cFS framework has been deployed**
  - Defer embedded software complexities due to cross compilation and target operating systems
  - Smartphone apps need to be rewritten for each platform
  - Provides seamless application transition from technology efforts to flight projects
- **Reload apps during operations without rebooting**

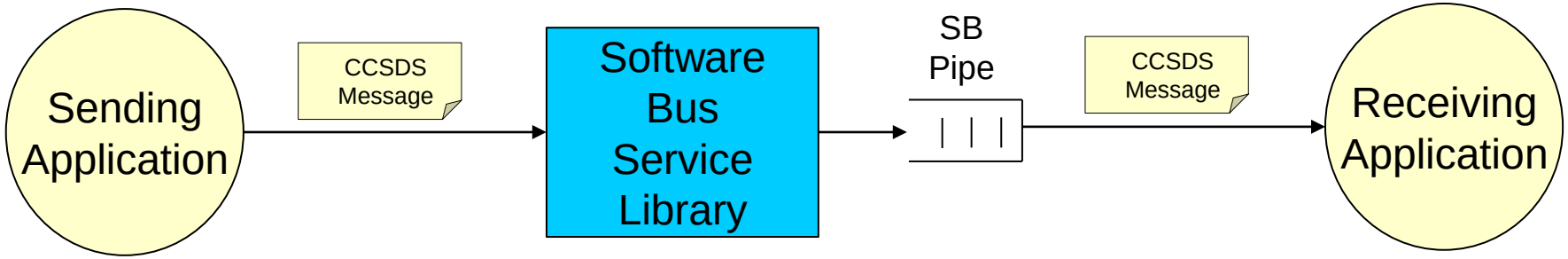
Common cFE Service Design



- **Each cFE service has**
  - A library that is used by applications
  - An application that provides a ground interface for operators to use to manage the service
- **Each cFE Service App periodically sends status telemetry in a "Housekeeping (HK) Packet"**
  - Obtaining additional information beyond HK with commands that
    - Send one-time telemetry packets
    - Write onboard service configuration data to files

⇨ = Software Bus Message

Message-Centric Application Design



- **Applications create *SB Pipe (a FIFO queue)* and subscribe to receive messages**
  - Typically performed during application initialization
- **If needed, apps can subscribe and unsubscribe to messages at any time for runtime reconfiguration**
- ***SB Pipes* used for application data and control flow**
  - Poll and pend for messages

cFE Libraries

- **What is a library?**
  - A collection of utilities available for use by any app
  - Exist at the cFS application layer
- **Libraries are <u>not</u> registered with Executive Services and do not have a thread of execution so limited cFE API usage. For example,**
  - A library can't call CFE_EVS_Register() during initialization
  - The ES API does not provide a function for libraries analogous to CFE_ES_GetAppInfo()
- **Library functions execute within the context of the calling application**
  - CFE_EVS_SendEvent() will identify the calling app
  - Libraries can't register for cFE services during initialization and in general should not attempt to do so
- **No cFE API exists to retrieve library code segment addresses**
  - Prevents apps like Checksum from accessing library code space.
- **Libraries and be statically dynamically linked**
  - Dynamic linking requires support from the underlying operating system
- **Specified in the cfe-es-startup.scr and loaded during cFE initialization**


cFE Services
Functional Overview

Introduction

- **This section briefly introduces each cFE service's functionality**
  - OSK provides detailed material on each service with demos and self-guided tutorials

- **Section outline**
  1. Executive Service (ES)
  2. Event Service (EVS)
  3. Software Bus Service (SB)
  4. Table Service (TBL)
  5. Time Service (TIME)
  6. cFE directory structure
  7. Configuration parameter design

Executive Services Overview

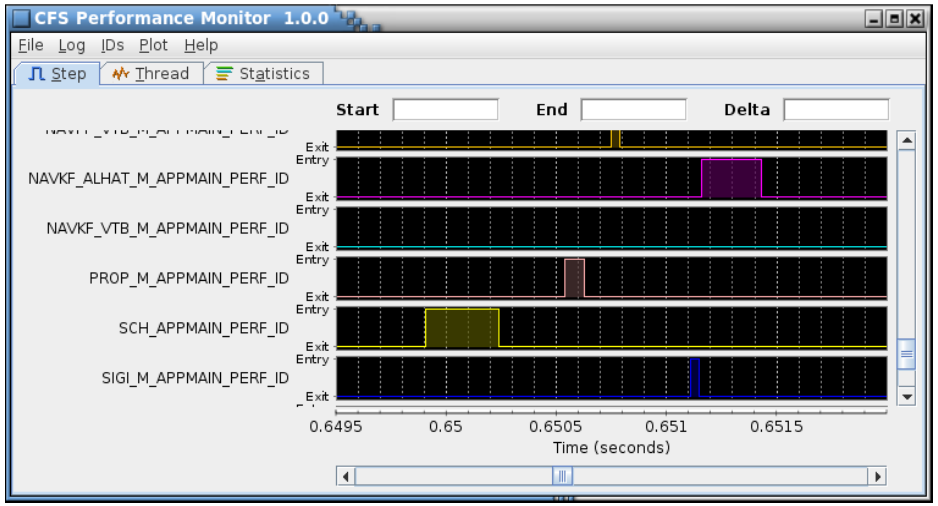
- **Initializes the cFE**
  - Reports reset type
  - Maintains an exception-reset log across processor resets
- **Creates the application runtime environment**
  - Primary interface to underlying operating system task services
  - Manages application resources
  - Supports starting, stopping, and loading applications during runtime
- **Memory Management**
  - Provides a dynamic memory pool service
  - Provides Critical Data Stores (CDS) that are preserved across processor resets

Executive Service Application Support

- **Applications are an architectural component that owns cFE and operating system resources**
- **Each application has a thread of execution in the underlying operating system (i.e. a task)**
- **Applications can create multiple child tasks**
  - Child tasks share the parent task's address space
- **Mission applications are defined in *cfe_es_startup.scr* and loaded after the cFE applications are created**
- **Application Restarts and Reloads**
  - Start, Stop, Restart, Reload commands
  - Data is not preserved; application run through their initialization
  - Can be used in response to
    - Exceptions
    - On-board Failure Detection and Correction response
    - Ground commands

Executive Service Performance Analyzer

- **Provides a method to identify and measure code execution paths**
  - System tuning, troubleshooting, CPU loading
- **Executive Service provides Developer inserts execution markers in FSW**
  - Entry marker indicate when execution resumes
  - Exit marker indicates when execution is suspended
  - CFE_ES_PerfLogExit() => CFE_SB_RcvMsg() => CFE_ES_PerfLogEntry()
- **Operator defines what markers should be captured via filters and defines triggers that determine when the filtered marker are captured**
- **Captured markers are written to a file that is transferred to the ground and displayed using the cFS Performance Monitor (CPM) tool**



Event Service Overview

- **Provides an interface for sending time-stamped text messages on the software bus**
  - Considered asynchronous because they are not part of telemetry periodically generated by an application
  - Processor unique identifier
  - Optionally logged to a local event log
  - Optionally output to a hardware port
- **Four event types defined**
  - Debug, Informational, Error, Critical
- **Event message control**
  - Apps can filter individual messages based on identifier
  - Enable/disable event types at the processor and application scope

Event Message Filtering

- **“Filter Mask”**
  - Bit-wise Boolean AND performed on event ID message counter, if result is zero then the event is sent
  - Mask applied before the sent counter is incremented
  - 0x0000 => Every message sent
  - 0x0003 => Every 4th message sent
  - 0xFFFE => Only first two messages sent
- **Reset filter**
  - Filters can be reset from an application or by command
- **Event filtering example**
  - Software Bus ‘No Subscriber’ event message, Event ID 14
    - See *cfe_platform_cfg.h* CFE_SB_FILTERED_EVENT1
  - Default configuration is to only send the first 4 events
    - Filter Mask = 0xFFFC
- **CFE_EVS_MAX_FILTER_COUNT (cfe_evs_task.h) defines maximum count for a filtered event ID**
  - Once reached event becomes locked
  - Prevents erratic filtering behavior with counter rollover
  - Ground can unlock filter by resetting or deleting the filter

Event Message Control

- **Processor scope**
  - Enable/disable event messages based on type
    - Debug, Information, Error, Critical
- **Application scope**
  - Enable/disable all events
  - Enable/disable based on type
- **Event message scope**
  - During initialization apps can register events for filtering for up to CFE_EVS_MAX_EVENT_FILTERS defined in *cfe_platform_cfg.h*
  - Ops can add/remove events from an app's filter

Software Bus Services Overview

- **Provides an inter-application message service using a publish/subscribe model**

- **Routes messages to all applications that have subscribed to the message (i.e. broadcast model)**
  - Subscriptions are done at application startup
  - Message routing can be added/removed at runtime
  - Sender does not know who subscribes (i.e. connectionless)

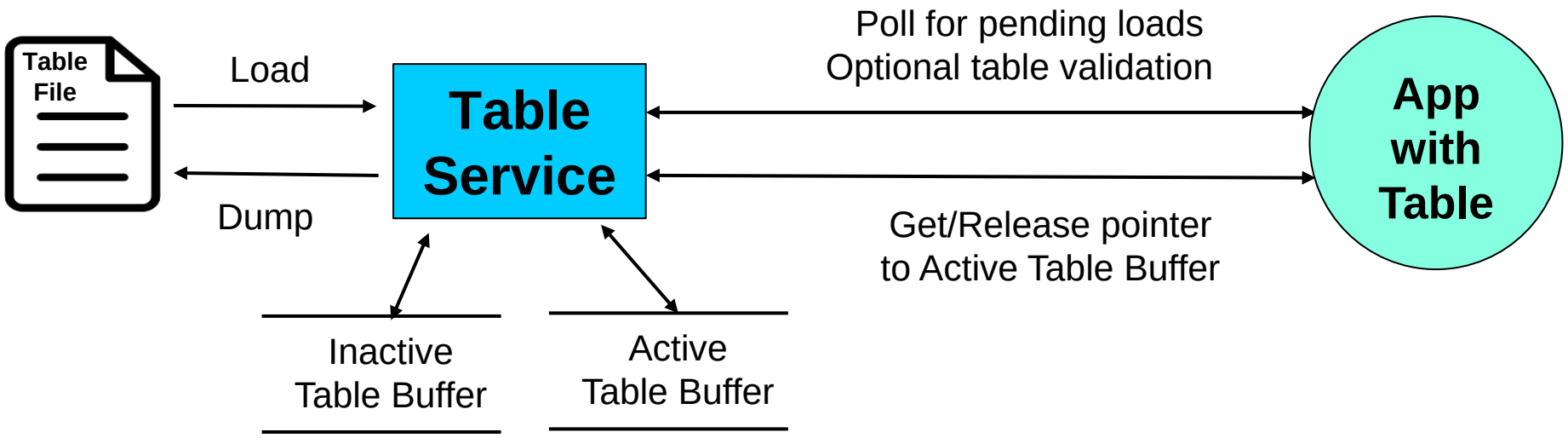
- **Reports errors detected during the transferring of messages**

- **Outputs Statistics Packet and the Routing Information when commanded**

Table Service Overview

- **What is a table?**
  - Tables are logical groups of parameters that are managed as a named entity
- **Parameters typically change the behavior of a FSW algorithm**
  - Examples include controller gains, conversion factors, and filter algorithm parameters
- **Tables service provides ground commands to load a table from a file and dump a table to a file**
  - Table loads are synchronized with applications
- **Tables are binary files**
  - Ground support tools are required to create and display table contents
- **The cFE can be built without table support**
  - Note the cFE applications don't use tables

Table Service Functional Overview



- **Table service contains buffers that hold tables for all applications**
  - Active Table Buffer - Image accessed by app while it executes
  - Inactive Table Buffer - Image manipulated by ops (could be stored commands)
- **"Table Load" is a sequence of activities to transfer data from a file to the Active Table Buffer**
- **"Table Dump" is a sequence of activities to transfer data from a either Table Buffer to a file**
- **Table operations are synchronous with the application that owns the table to ensure table data integrity**

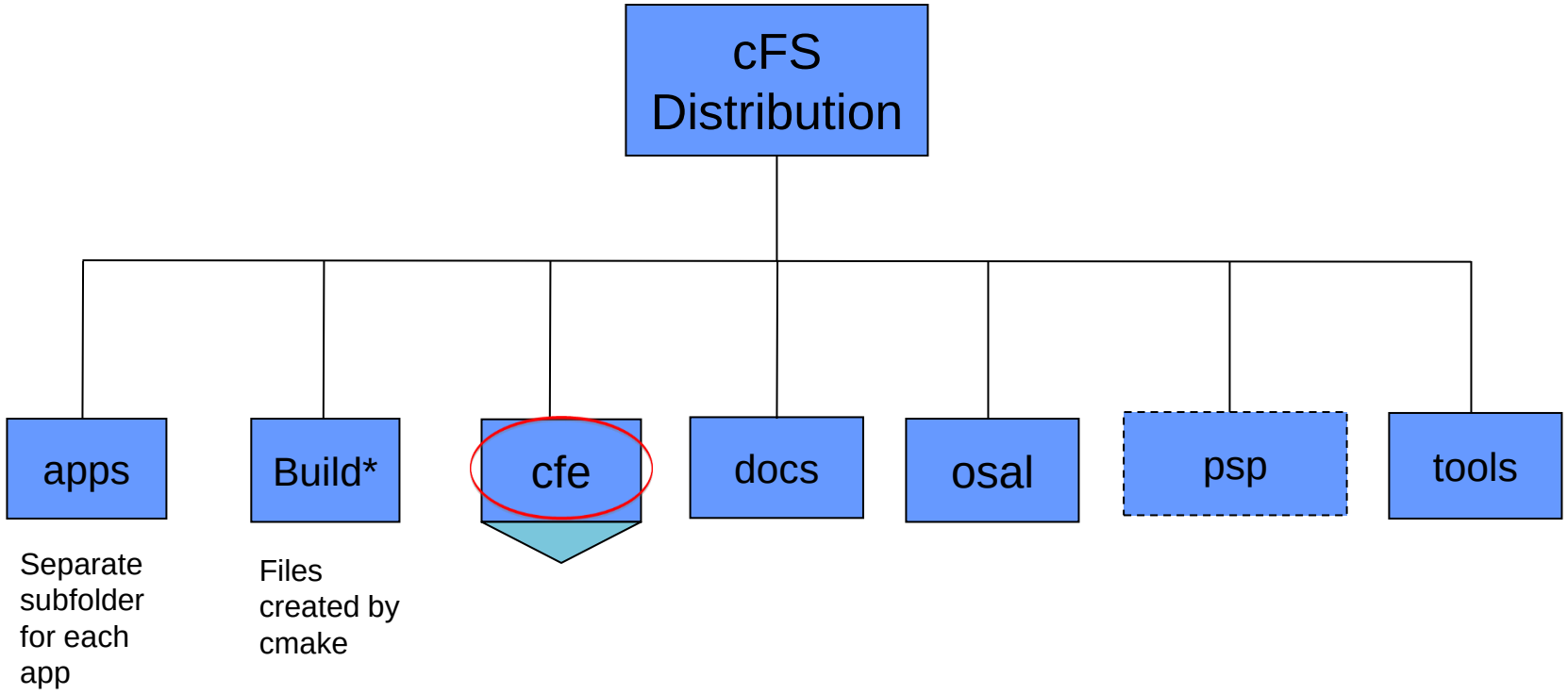
Time Service Overview

- cFE Time Services provides time correlation, distribution and synchronization services
- Provides a user interface for correlation of spacecraft time to the ground reference time (epoch)
- Provides calculation of spacecraft time, derived from mission elapsed time (MET), a spacecraft time correlation factor (STCF), and optionally, leap seconds
- Provides a functional API for cFE applications to query the time
- Distributes a “time at the tone” command packet, containing the correct time at the moment of the 1Hz tone signal
- Distributes a “1Hz wakeup” command packet
- Forwards tone and time-at-the-tone packets
- Designing and configuring time is tightly coupled with the mission avionics design

Time Service Time Formats

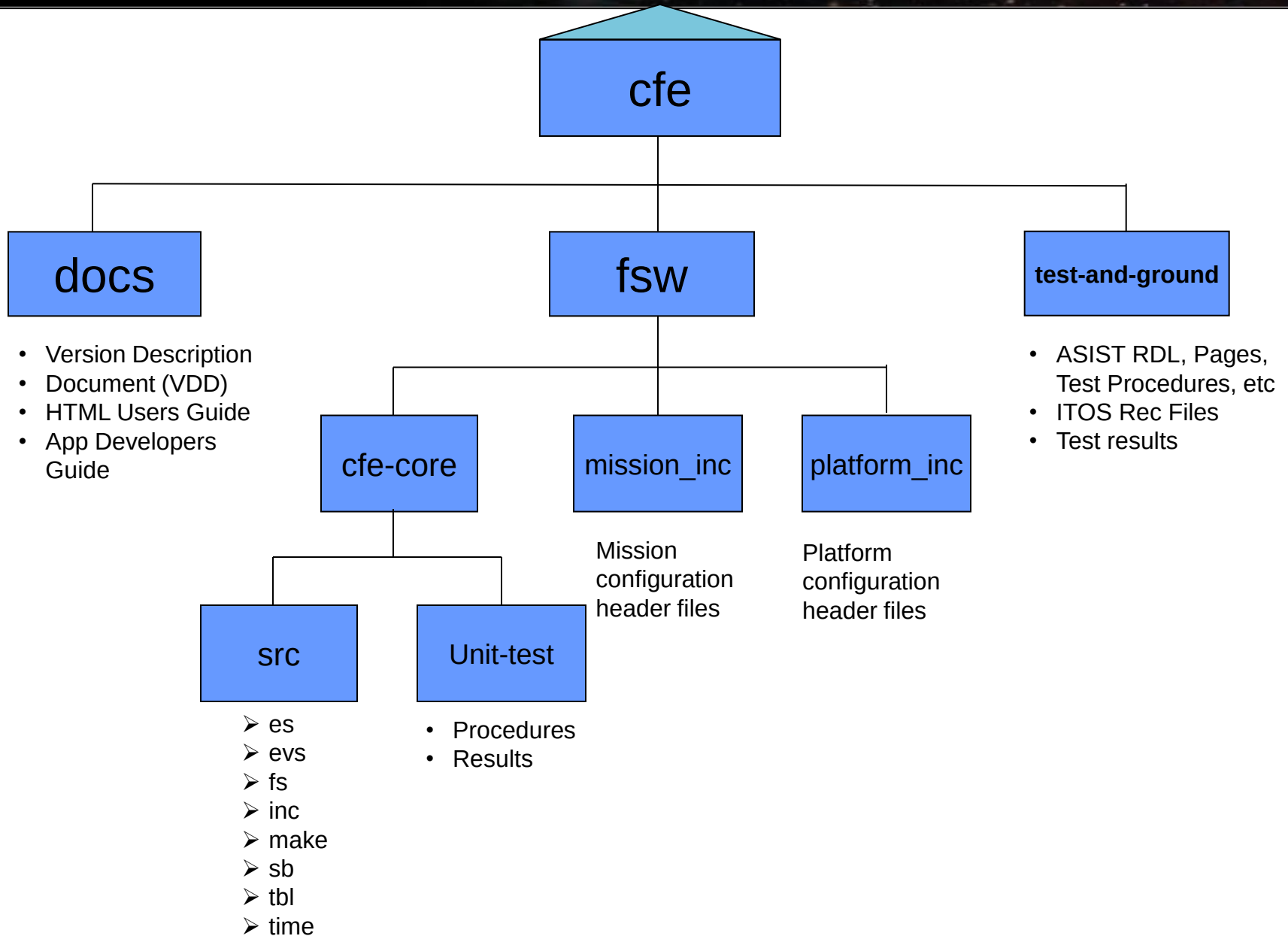
- **Supports two formats**
- **International Atomic Time (TAI)**
  - Number of seconds and sub-seconds elapsed since the ground epoch
  - TAI = MET + STCF
    - Mission Elapsed Counter (MET) time since powering on the hardware containing the counter
    - Spacecraft Time Correlation Factor (STCF) set by ground ops
    - Note STCF can correlate MET to any time epoch so TAI is mandated
- **Coordinated Universal Time (UTC)**
  - Synchronizes time with astronomical observations
  - UTC = TAI – Leap Seconds
  - Leap Seconds account for earth's slowing rotation

cFS Mission Directory Structure



* Files created by cmake

cFE Directory Structure



Configuration Parameter Scope

- **Mission configuration parameters – used for ALL processors in a mission (eg. time epoch, maximum message size, etc)**
  - Default contained in:
    - \cfe\fsw\mission_inc\cfe_mission_cfg.h
    - \apps\xx\fsw\mission_inc\xx_mission_cfg.h. xx_perfids.h

- **Platform Configuration parameters – used for the specific processor (eg. time client/server config, max number of applications, max number of tables, etc)**
  - Defaults contained in:
    - \cfe\fsw\platform_inc\cpuX\cfe_platform_cfg.h, cfe_msgids_cfg.h
    - \apps\xx\fsw\platform_inc\xx_platform_cfg.h, xx_msgids.h
    - \osal\build\inc\osconfig.h

- **Just because something is configurable doesn't mean you want to change it**
  - E.g. CFE_EVS_MAX_MESSAGE_LENGTH

Unique Identifier Configuration Parameters

- **Software Bus Message Identifiers**
  - cfe_msgids.h (message IDs for the cFE should not have to change)
  - app_msgids.h (message IDs for the Applications) are platform configurations
- **Executive Service Performance Identifiers**
  - cFE performance IDs are embedded in the core
  - app_perfids.h (performance IDs for the applications) are mission configuration
- **Task priorities are not configuration parameters but must be managed from a processor perspective**
- **Note cFE strings are case sensitive**

cFS Application Mission and Platform Configuration Files

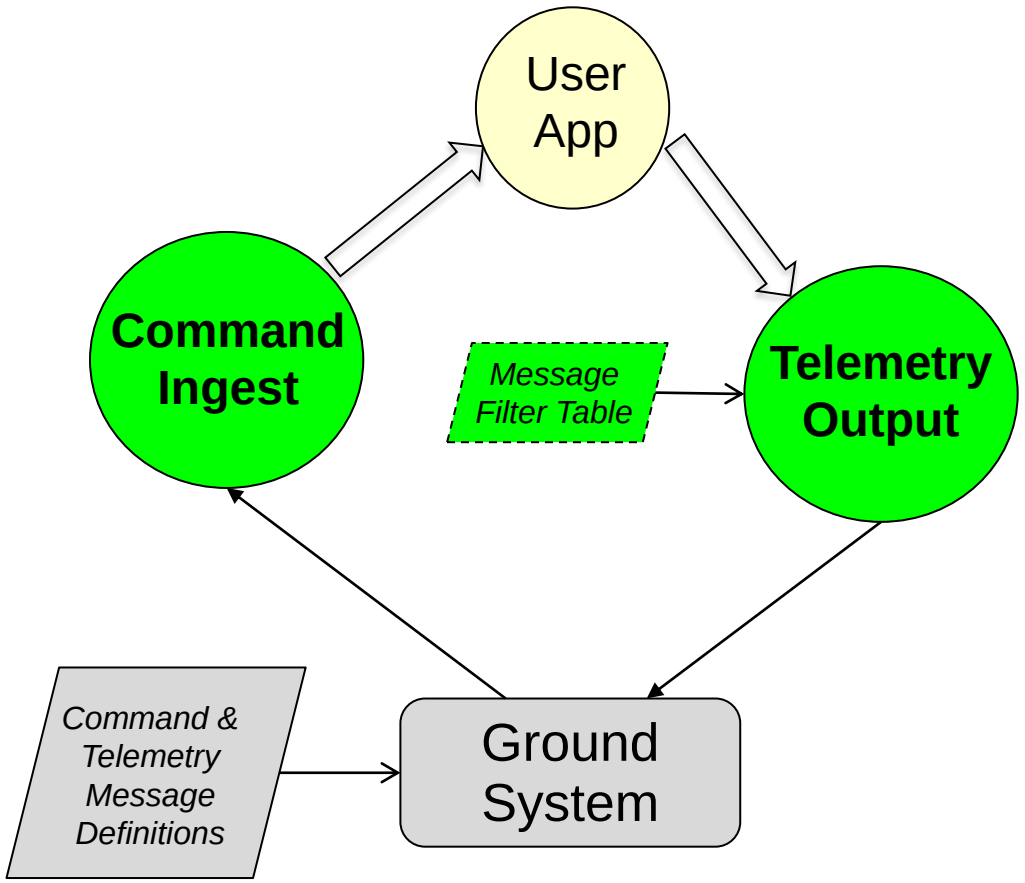
File	Purpose	Scope	Notes
cfe_mission_cfg.h	cFE core mission wide configuration	Mission	
cfe_platform_cfg.h	cFE core platform configuration	Platform	Most cFE parameters are here
cfe_msgids.h	cFE core platform message IDs	Platform	Defines the message IDs the cFE core will use on that Platform(CPU)
osconfig.h	OSAL platform configuration	Platform	
XX_mission_cfg.h	A cFS Application's mission wide configuration	Mission	Allows a single cFS application to be used on multiple CPUs on one mission
XX_platform_cfg.h	Application platform wide configuration	Platform	
XX_msgids.h	Application message IDs	Platform	
XX_perfids.h	Application performance IDs	Platform	


Runtime Application
Context

Introduction

- **This section introduces the concept of an application runtime environment that is created by a common set of apps that are typically present in a cFS distribution**
- **The cFE does not dictate this model but a minimal set of apps is required to make a cFS distribution usable**
  - OSK includes KIT_CI, KIT_TO, and KIT_SCH that perform the necessary functionality
  - NASA maintains a cFS Bundle, https://github.com/nasa/cFS, that include 'lab' versions of these apps
- **OSK's *Mission FSW* provides a SimSat reference mission that describes in detail how groups of apps can collaborate to provide end-user functionality**

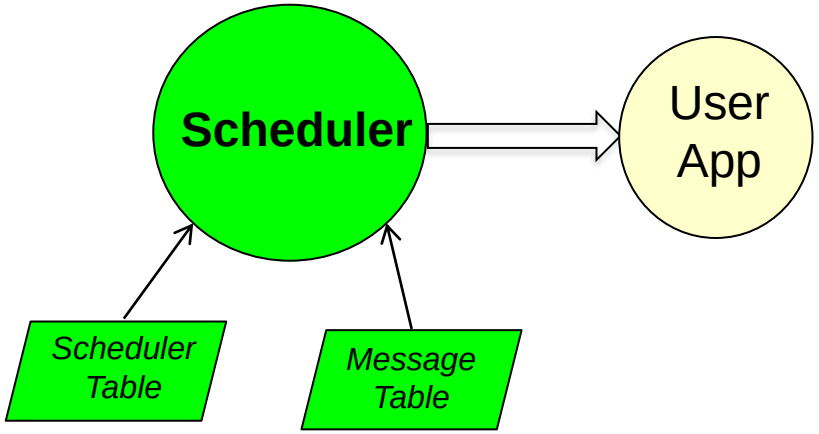
Command & Telemetry Context



- **Command Ingest (CI) App**
  - Receives commands from an external source, typically the ground system, and sends them on the software bus
- **Telemetry Output (TO) App**
  - Receives telemetry packets from a the software bus and sends them to an external source, typically the ground system
  - Optional *Filter Table* that provides parameters to algorithms that select which messages should be output on the external communications link

- **Different versions of CI and TO used on different platforms**
  - cFE delivered with 'lab' versions that use UDP for the external comm
  - JSC released versions that use a configurable I/O library for a different external comm links
  - OSK versions use UDP and a JSON filter table
  - ITAR-restricted flight versions typically used inflight

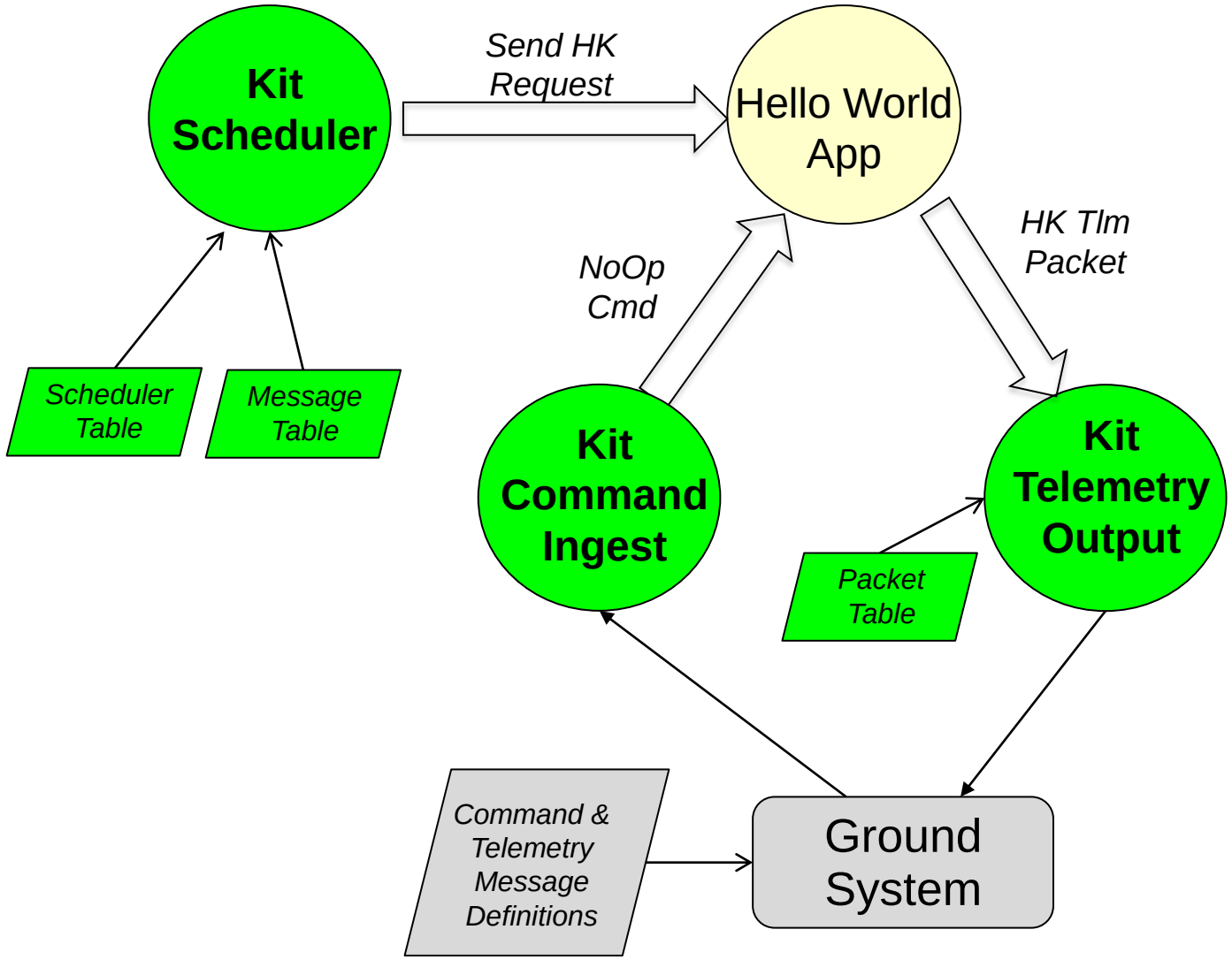
Application Scheduling Context



- **Scheduler (SCH) App**
  - Synchronizes execution with clock's 1Hz signal
  - Sends software bus messages defined in the *Message Table* at time intervals defined in the *Scheduler Table*

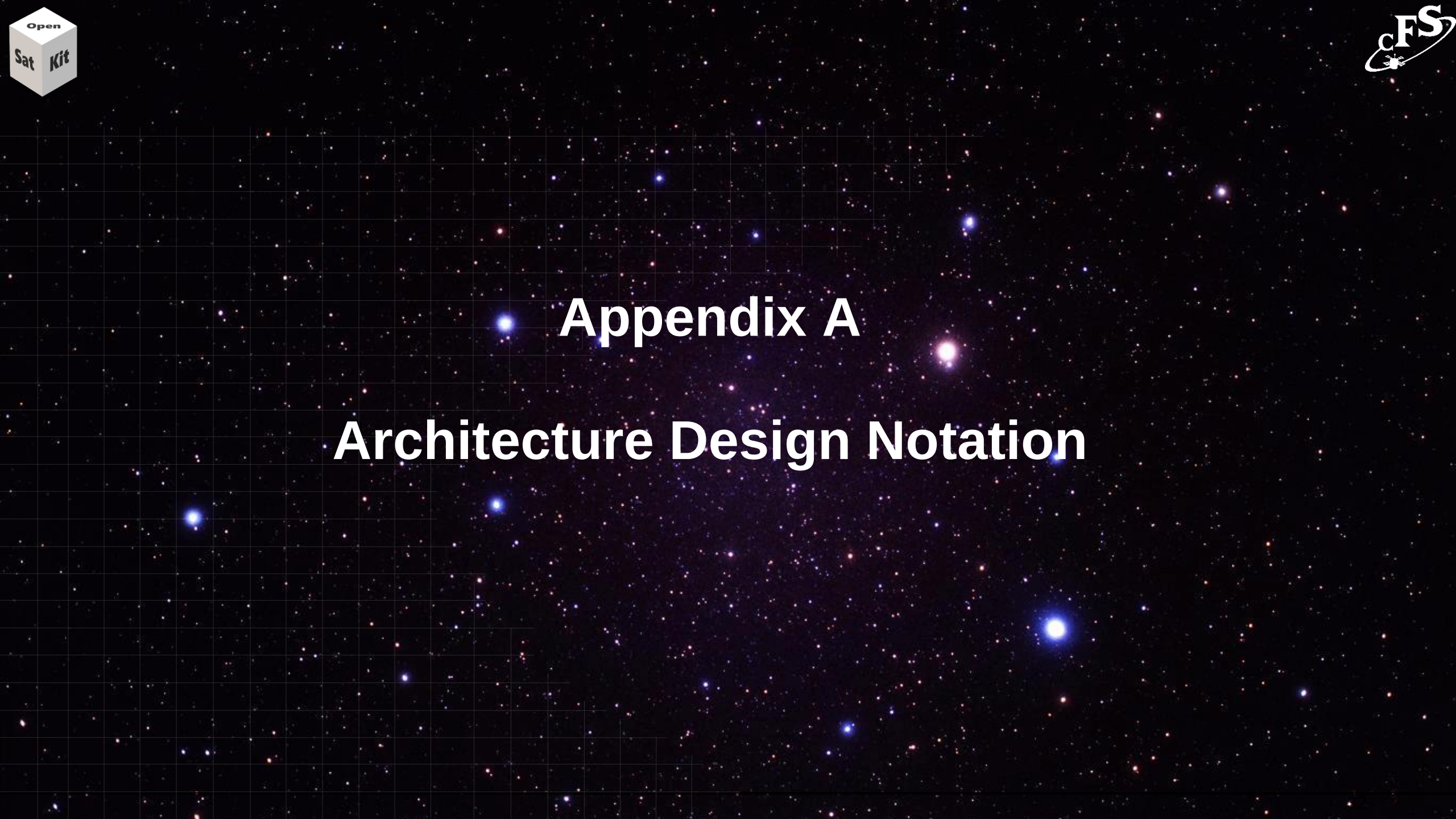
- **Application Control Flow Options**
  - Pend indefinitely on a *SB Pipe* with subscriptions to messages from the Scheduler
    - This is a common way to synchronize the execution of most of the apps on a single processor
    - Many apps send periodic "Housekeeping" status packets in response to a "Housekeeping Request message from Scheduler
  - Pend indefinitely on a message from another app
    - Often used when an application is part of a data processing pipeline
  - Pend with a timeout
    - Used in situation with loose timing requirements and system synchronization Is not required
    - The SB timeout mechanism uses the local oscillator so the wakeup time may drift relative to the 1Hz

Hello World App Runtime Environment



Context of "Hello World" app created in the next section

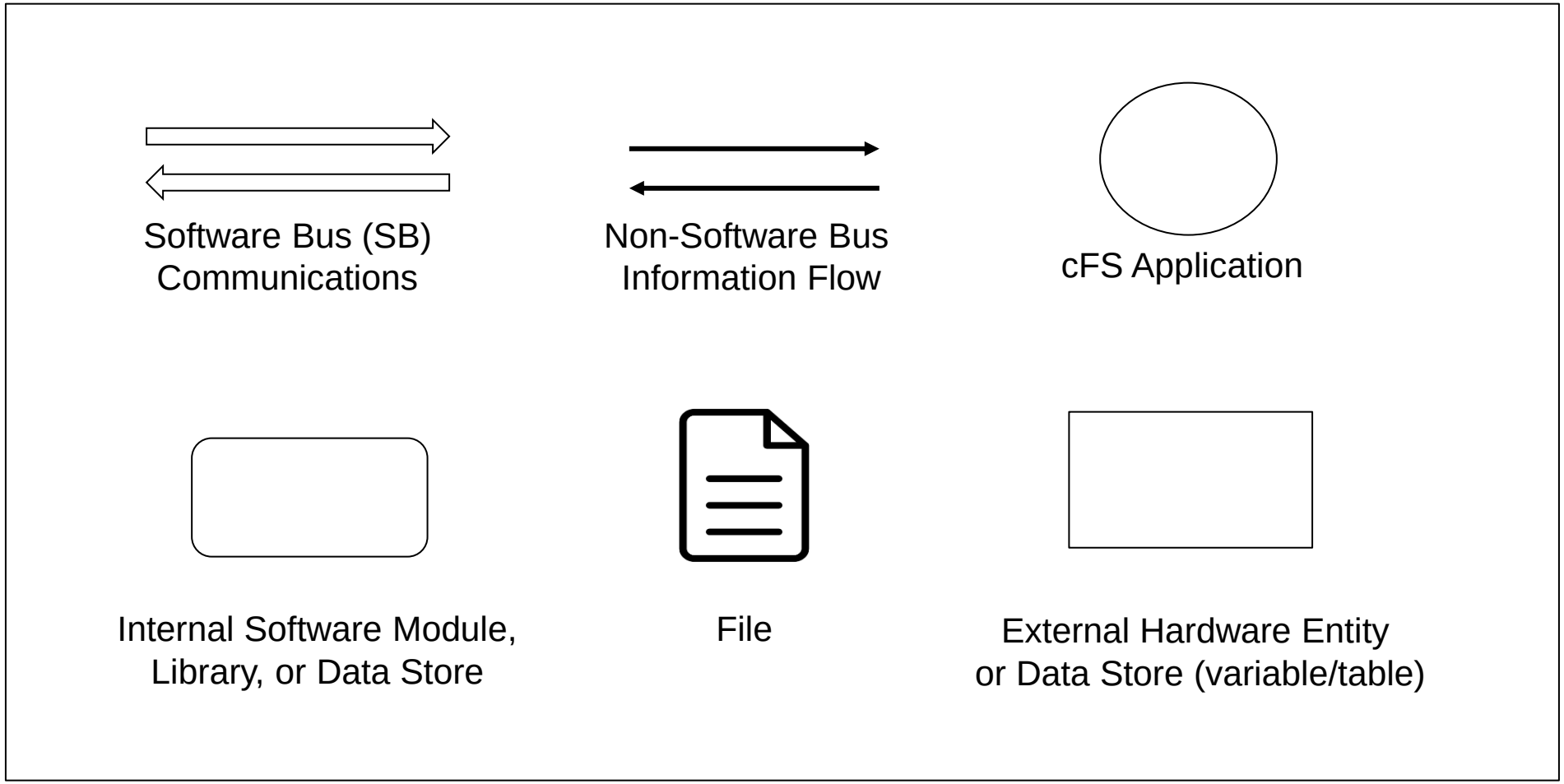
- Every 3 seconds Scheduler sends a "Send Housekeeping Telemetry Request"
  - HK telemetry includes valid and invalid command counters
- When user sends a "No Operation" command from the ground system Hello World responds with
  - An event message that contains the app's version number
  - Increments the command valid counter



Appendix A

Architecture Design Notation

Architecture Design Notation



- Common data flows such as command inputs to an app and telemetry outputs from an app are often omitted from context diagrams unless they are important to the situation

cFE Service Slide Deck Template

The following general outline is used in each of the cFE service documentation slides

- **Describe each service's main features from different perspectives**
  - System functions and operations
    - Feature Overview
    - Initialization and processor reset behavior
    - Onboard state retrieval
  - System integrator and developer
    - Configuration parameter highlights
    - Common practices
- **Student exercises are provided in a separate package**
  - Allows these slides to be maintained independent of the training platform and the training exercises can evolve independent of these slides