Core Flight System Framework

Basecamp

Version 1.18

August 2024

Audience & Prerequisites

- **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.
- **Prerequisites**
  - Course Introductory material provided in the cFS overview slides and video
- **Hands-on cFS Basecamp Exercises**
  - cFS Basecamp, https://github.com/cfs-tools/cfs-basecamp , is a lightweight environment with built-in tutorials for learning the cFS

Blue screens outline hands on exercises

Outline

1. cFS Architecture
2. cFS Framework Services
3. Application Layer Architectural Components
4. cFS Framework Deployment

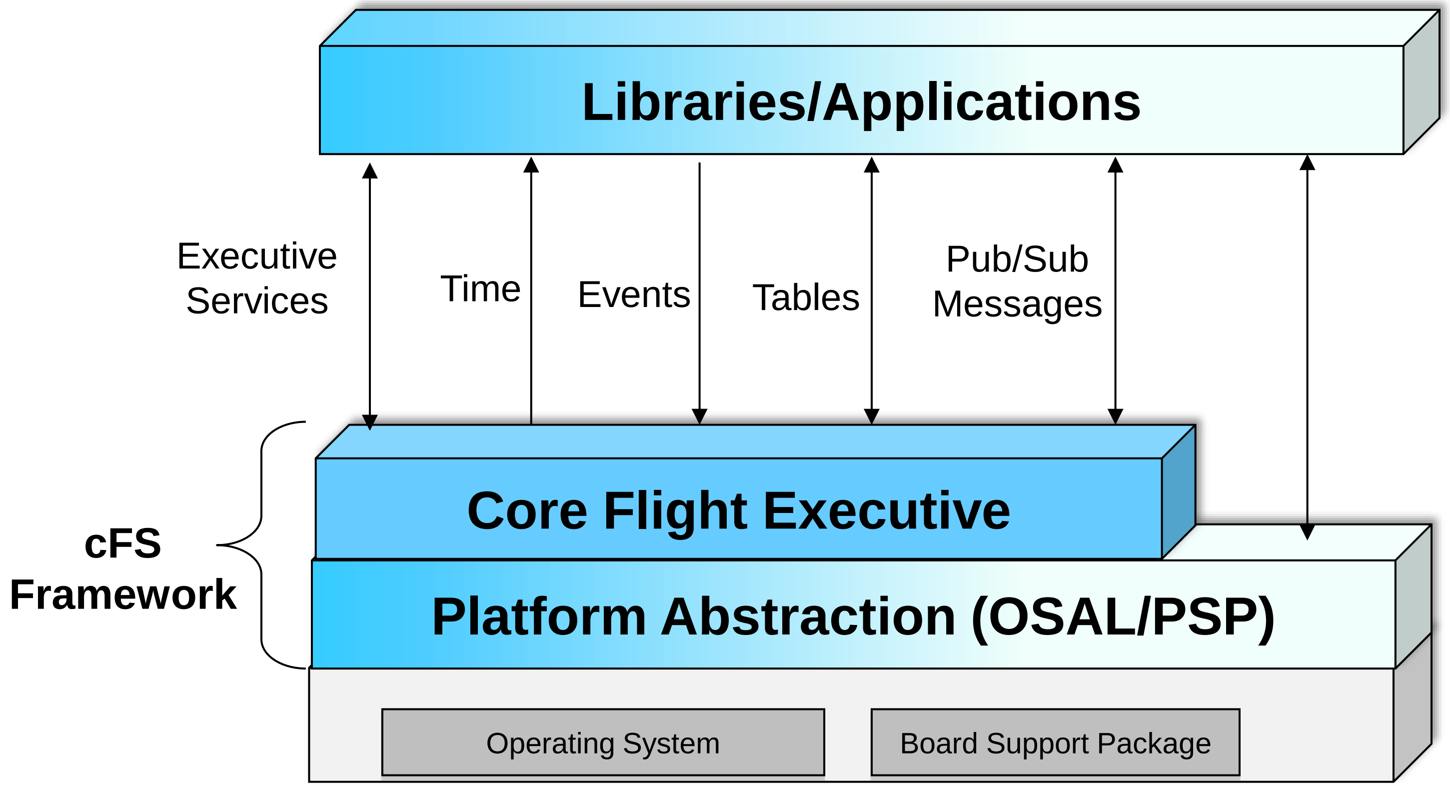
Appendix A: Architectural Design Notation

Appendix B: Supplemental Architectural Material

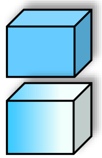
cFS Architecture

Basecamp

cFS Framework Interfaces



OSAL: Operating System Abstraction Layer
PSP: Platform Support Package



= Reusable components
= Reusable components with custom code

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

Platform Abstraction Layer (1 of 2)

Operating System (OS)

- System software that manages computer hardware and software resources, and provides common services for computer programs
- Software services include scheduling different threads of execution, facilitating communication between threads, and managing memory for the entire system

Realtime Operating System (RTOS)

- Supports multi-tasking with preemptive scheduling so time critical tasks execute deterministically

Board Support Package (BSP)

- Contains hardware-specific boot firmware, device drivers and other routines to an operating system to function in a given hardware environment (i.e. a motherboard)

Platform Abstraction Layer (2 of 2)

Operating System Abstraction Layer (OSAL)

- A software library that provides a single Application Program Interface (API) to the core Flight Executive (cFE) regardless of the underlying operating system

Platform Support Package (PSP)

- A software library that provides a single API to the underlying hardware board and BSP
- Library serves as the "glue" between the OS and the cFE
- During system initialization it performs BSP/OS specific setup and then calls cFE’s entry point function

Notes

- The cFS Framework defaults to the Linux and can be run on a personal computer
- The cFS Platform List https://github.com/cfs-tools/cfs-platform-list provides links to cFS ports

Application Layer

- **Applications are architectural components that own and use cFE and operating system resources via the cFE and OSAL APIs**

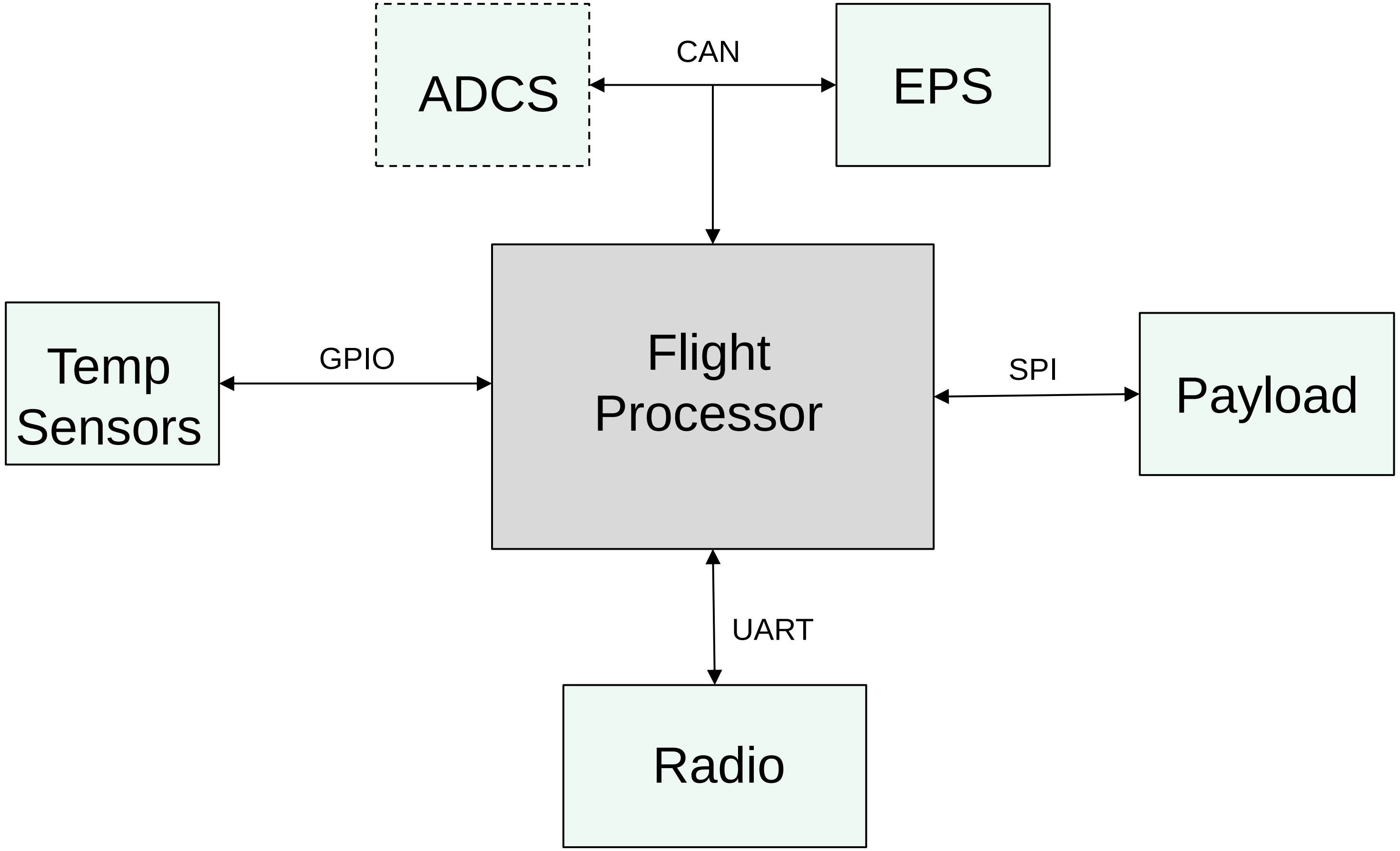
- **cFS Framework Services provide an Application Runtime Environment**
  - Apps register for cFS Framework resources and services
  - Dynamic resource management allows apps to be restarted or replaced without restarting the system

- **Write once run anywhere the cFS Framework has been deployed**
  - Apps are portable across different hardware/operating platforms if they only depend on the cFE and Platform Abstraction layer APIs
  - Allows app development on a desktop deferring embedded software complexities such as cross compilation, deploying to target, etc.
  - Technology projects developed on a desktop have a path to flight projects
  - More powerful model than Smartphone apps that need to be rewritten for each platform

cFS Mission Design Overview

- **The remaining slides in this section introduce a top-down mission app design process**
  - This process provides an application context for the cFS Framework section
- **FSW design begins with a hardware context that bounds the FSW problem space**
- **Initially you may not have a compete hardware context because you are performing trade studies**
  - Build vs buy, component supplier evaluation (influences interfaces), processor selection, etc.
- **In parallel, a mission functional requirements analysis is being performed that identifies the top-level FSW functional requirements**
  - The mission's operational concept plays a significant role in this analysis
  - Performance requirements and design constraints are also elicited which may effect the component selection
  - Simulations may be required to perform the analysis

Example Hardware Context



ADCS
- Attitude Determination and Control System

CAN
- Controller Area Network

EPS
- Electric Power System

GPIO
- General Purpose Input/Output

SPI
- Serial Peripheral Interface

UART
- Universal Asynchronous Receiver/Transmitter..

ADCS Represents external ADCS or ADCS sensors/actuators with controller in FSW

Example Spacecraft Flight Software Top-Level Requirements

FSW

- C&DH
- ADCS-GNC
- Mechanisms
- Thermal
- Power
- Payload

C&DH

- **Operational Environment**
  - App Runtime Env
  - Command Ingest & Distribution
  - Telemetry Collection & Output
  - Inter-processor Comm
  - Time Management
- **Onboard Data Management**
  - File Management
  - File Transfer
  - Recorder Management
- **Autonomy, Failure Detection & Correction**
  - Autonomous Commanding
  - Hardware & Software Monitoring
  - Memory Integrity Support & Checks

ADCS-GNC

- **Attitude Determination and Control**
- **Orbit Determination and Control**
- **Models**
  - Solar
  - Lunar
  - Magnetic Field

Mechanisms

- **Deployments**
- **Solar Array Control**
- **Antenna Control**

Thermal

- **Autonomous Control**
- **Tlm & Cmd I/F**
- **Health & Safety**

Power

- **Autonomous Control**
- **Tlm & Cmd I/F**
- **Health & Safety**

Payload

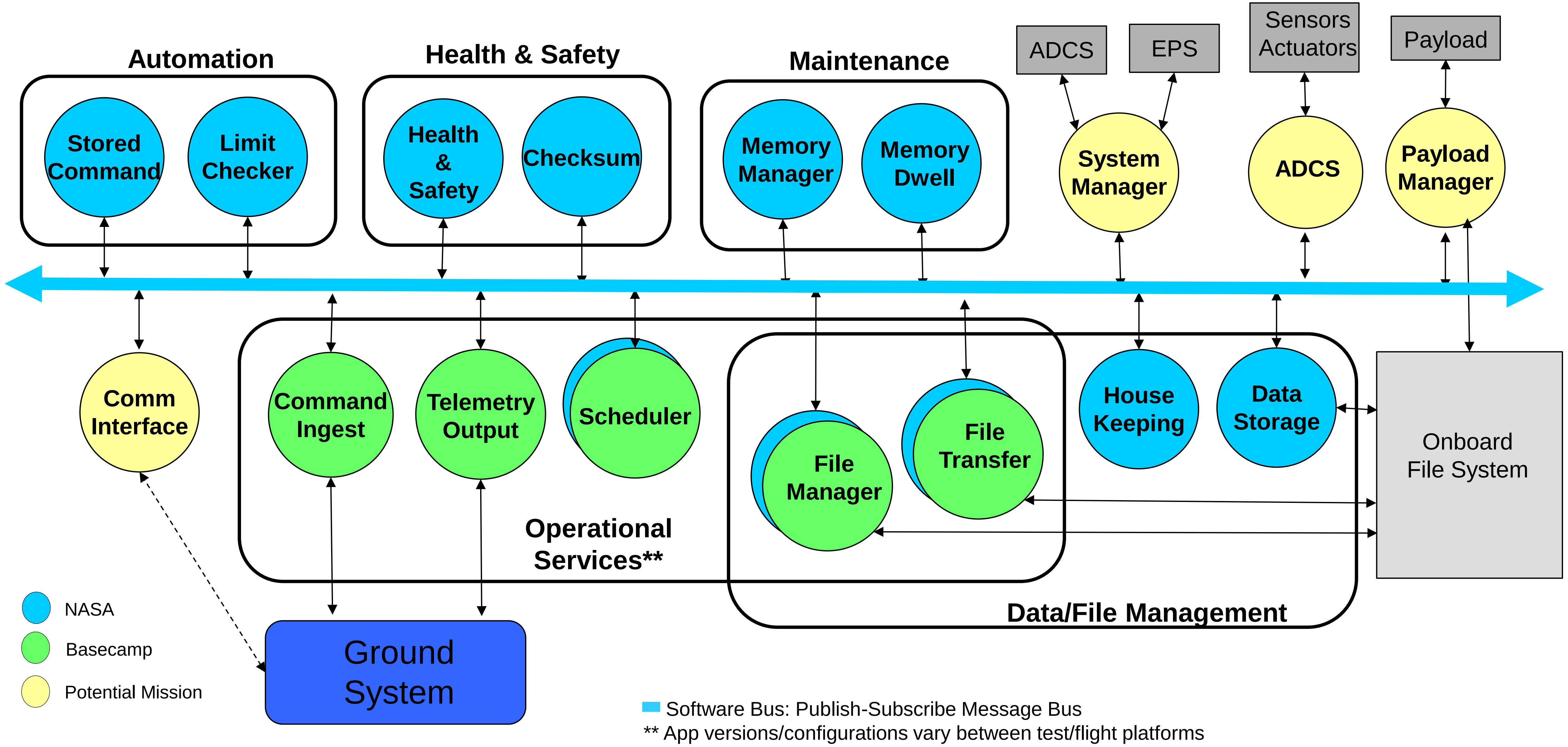
- **Autonomous Control**
- **Data Management**
- **Tlm & Cmd I/F**
- **Health & Safety**

ADCS: Attitude Determination & Control System
C&DH: Command & Data Handling
GNC: Guidance Navigation and Control

App-Centric Design

- **Mission functional requirements are implemented in apps and libraries**
- **A critical cFS mission design mindset is to concurrently think about individual apps and groups of apps implementing functional requirements**
  – For example, one app could determine and publish the sun's position relative to the spacecraft, a second app could subscribe to this data and control gimbaled solar arrays, and a third app could subscribe to both app's data and monitor the data for fault conditions
- **The next slide shows an example mission app suite and highlights the following**
  – Groups of apps provide mission functionality
  – Apps communicate using a publish/subscribe messaging system called the *Software Bus*
  – There are multiple sources of apps
  – Apps are used to manage external interfaces (note libraries and device drivers are not shown)

Example App Context "Lollipop Diagram"



cFS Architecture Wrap Up

- **At this point you do not need to understand each app's functionality**
  - This diagram is being shown to illustrate that it is one of the goals of the FSW mission design process

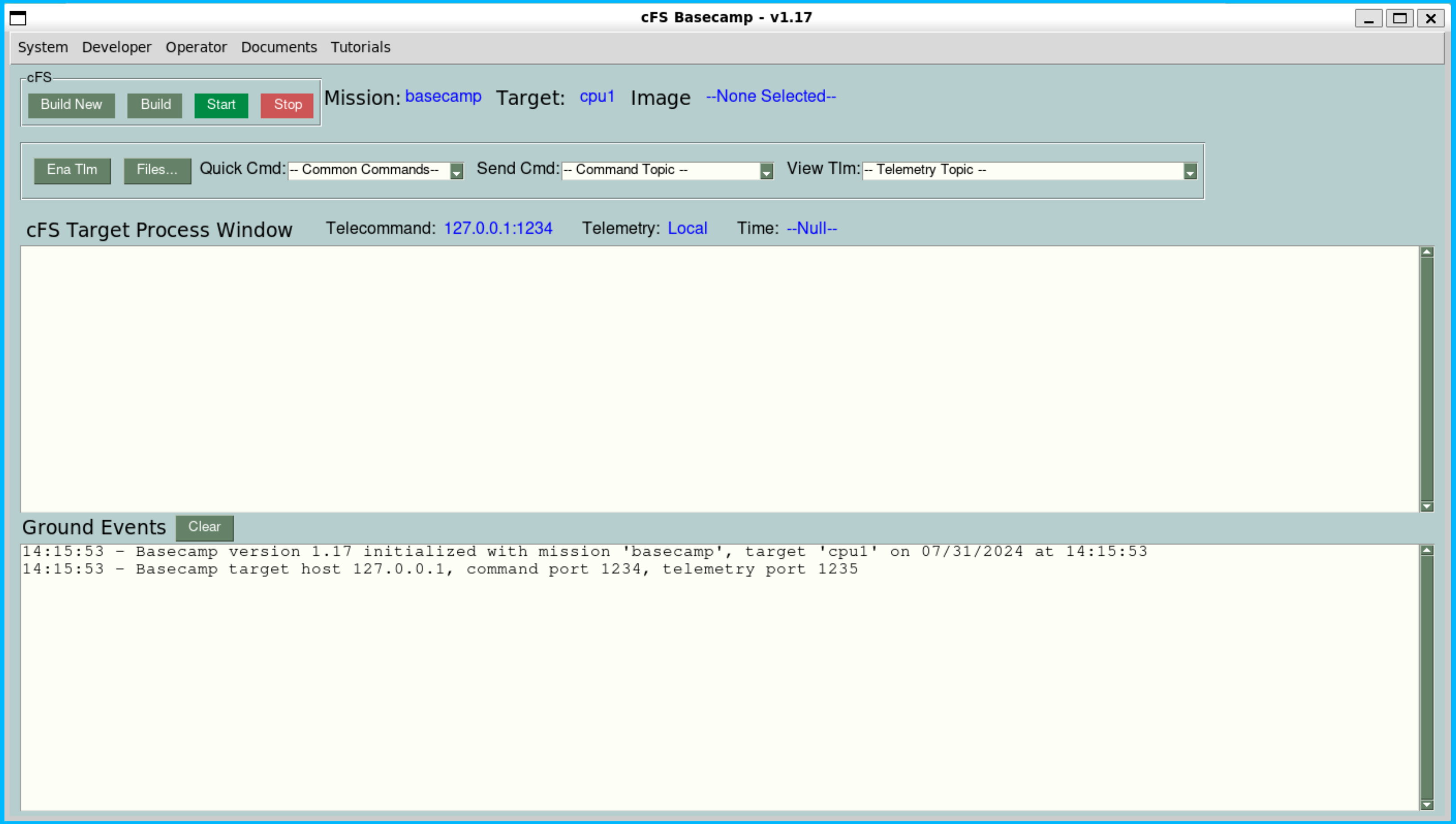
- **Basecamp's *Systems Engineering* documents provide detailed information on the mission design process**

- **cFS's mission app-centric design is the primary motivation for application runtime environment provided by the *cFS Framework Services* described next**

- **Once you have an understanding of the cFS Framework Services, the application layer will be revisited in more detail in the *Application Layer Architectural Components* section**

Install cFS Basecamp

1. Install cFS Basecamp following the instructions at https://github.com/cfs-tools/cfs-basecamp
2. The last step launches Basecamp using the command *"python3 basecamp.py"*



Basecamp is ready for exercises in the next section

cFS Framework Services

Basecamp

Outline

1. Executive Service (ES)
2. Event Service (EVS)
3. Software Bus Service (SB)
4. Table Service (TBL)
5. Time Service (TIME)
6. Operating System Abstraction Layer (OSAL)
7. Platform Support Package (PSP)

- **This section briefly introduces each cFE service's functionality**
- **Basecamp will be used to interact (send commands and observe telemetry responses) with each service**

Executive Services Overview

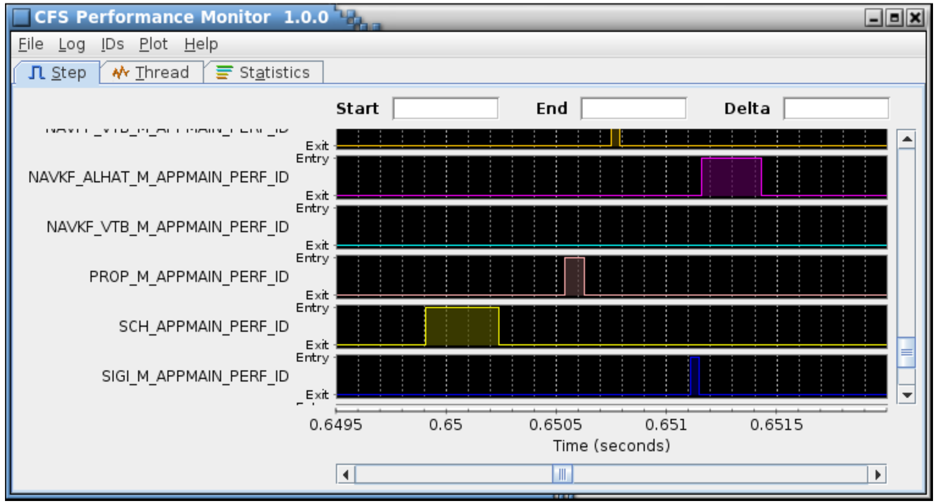
- **Initializes the cFE**
  - Identifies and reports startup/reset type
  - Maintains an exception-reset log across processor resets
- **Creates the application runtime environment**
  - Primary interface to underlying operating system task services & resources
  - Supports starting, stopping, and loading applications during runtime
- **Memory Management**
  - Provides a dynamic memory pool service
  - Provides Critical Data Stores (CDS) which are memory blocks that are preserved across processor resets if the platform supports it
  - Provides a system message log service where messages are stored in memory that can be written to a file on command

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

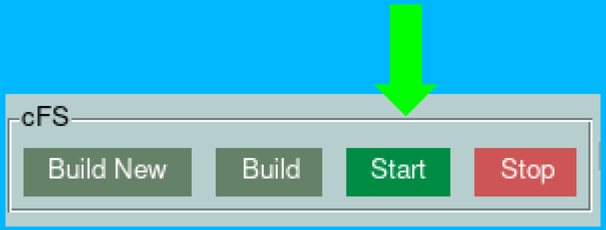
Executive Service Performance

- **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**



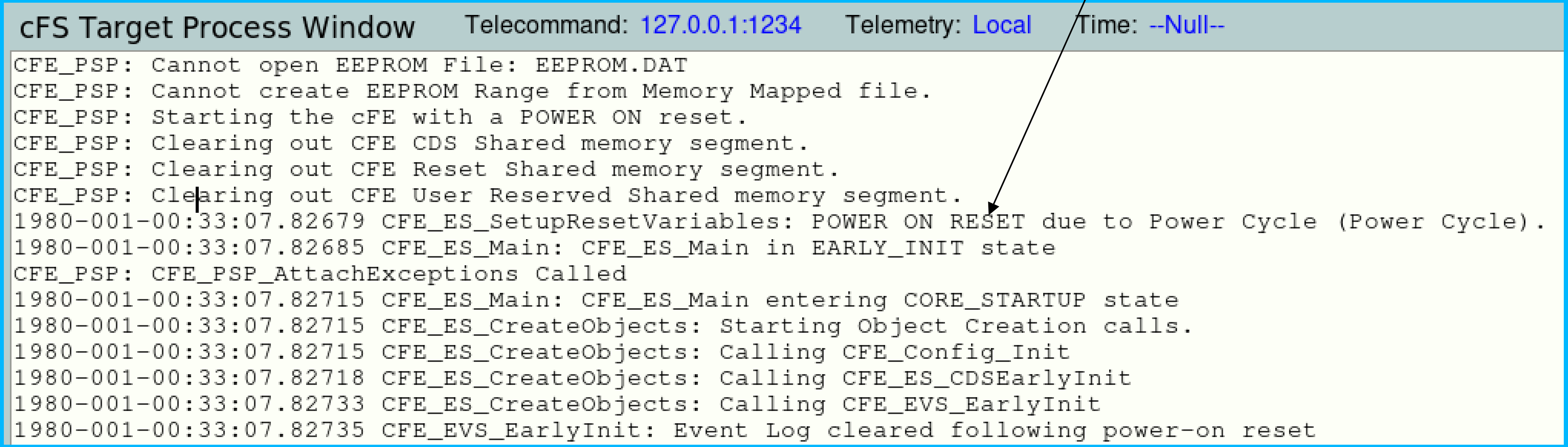
Executive Services Exercises - 1

1. Start the cFS



2. Scroll back in the *cFS Target Process Window* and look for cFE Executive Service (CFE_ES) entries

Power on Reset


cFS Target Process Window Telecommand: 127.0.0.1:1234 Telemetry: Local Time: --Null--

```
CFE_PSP: Cannot open EEPROM File: EEPROM.DAT
CFE_PSP: Cannot create EEPROM Range from Memory Mapped file.
CFE_PSP: Starting the cFE with a POWER ON reset.
CFE_PSP: Clearing out CFE CDS Shared memory segment.
CFE_PSP: Clearing out CFE Reset Shared memory segment.
CFE_PSP: Clearing out CFE User Reserved Shared memory segment.
1980-001-00:33:07.82679 CFE_ES_SetupResetVariables: POWER ON RESET due to Power Cycle (Power Cycle).
1980-001-00:33:07.82685 CFE_ES_Main: CFE_ES_Main in EARLY_INIT state
CFE_PSP: CFE_PSP_AttachExceptions Called
1980-001-00:33:07.82715 CFE_ES_Main: CFE_ES_Main entering CORE_STARTUP state
1980-001-00:33:07.82715 CFE_ES_CreateObjects: Starting Object Creation calls.
1980-001-00:33:07.82715 CFE_ES_CreateObjects: Calling CFE_Config_Init
1980-001-00:33:07.82718 CFE_ES_CreateObjects: Calling CFE_ES_CDSEarlyInit
1980-001-00:33:07.82733 CFE_ES_CreateObjects: Calling CFE_EVS_EarlyInit
1980-001-00:33:07.82735 CFE_EVS_EarlyInit: Event Log cleared following power-on reset
```

Executive Services Exercises - 2

3. Next in the *cFS Target Process Window* find when the libraries and apps are loaded

Startup script

```
1980-012-14:03:20.50282 CFE_ES_Main: CFE_ES_Main entering CORE_READY state
1980-012-14:03:20.50444 CFE_ES_StartApplications: Opened ES App Startup file: /cf/cfe_es_startup.scr
1980-012-14:03:20.50492 CFE_ES_ParseFileEntry: Loading shared library: /cf/cfe_assert.so
1980-012-14:03:20.51536 [BEGIN] CFE FUNCTIONAL TEST
1980-012-14:03:20.51540 [BEGIN] 01 CFE-STARTUP
1980-012-14:03:20.51546 CFE_ES_ParseFileEntry: Loading shared library: /cf/app_c_fw.so
Application C Framework Library Initialized. Version 4.3.0
1980-012-14:03:20.51666 CFE_ES_ParseFileEntry: Loading file: /cf/app_c_demo.so, APP: APP_C_DEMO
1980-012-14:03:20.51801 CFE_ES_ParseFileEntry: Loading file: /cf/ci_lab.so, APP: CI_LAB_APP
1980-012-14:03:20.51919 CFE_ES_ParseFileEntry: Loading file: /cf/kit_to.so, APP: KIT_TO
1980-012-14:03:20.52018 CFE_ES_ParseFileEntry: Loading file: /cf/file_mgr.so, APP: FILE_MGR
1980-012-14:03:20.52125 CFE_ES_ParseFileEntry: Loading file: /cf/file_xfer.so, APP: FILE_XFER
1980-012-14:03:20.52213 CFE_ES_ParseFileEntry: Loading file: /cf/kit_sch.so, APP: KIT_SCH
```

Basecamp
Target
Libs & Apps

Executive Services Exercises - 3

4. Open CFE_ES's Housekeeping (status) telemetry and browse its content

CFE_ES/Application/HK_TLM - Port 9003

App ID: 64 Length: 150 Seq Cnt: 411 Time:

Payload

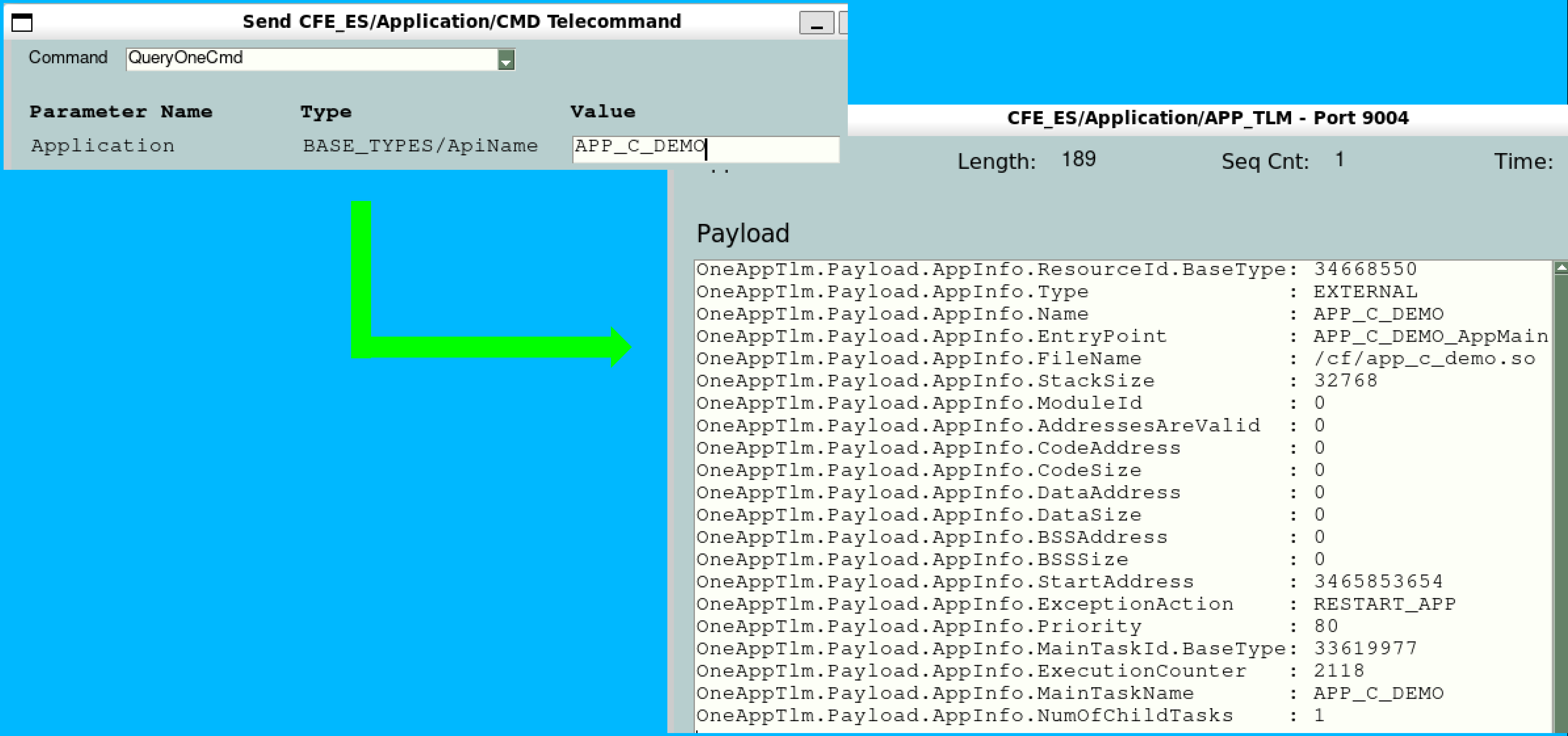
```
HousekeepingTlm.Payload.CommandCounter        : 0
HousekeepingTlm.Payload.CommandErrorCounter   : 0
HousekeepingTlm.Payload.CFECoreChecksum       : 65461
HousekeepingTlm.Payload.CFEMajorVersion       : 6
HousekeepingTlm.Payload.CFEMinorVersion       : 7
HousekeepingTlm.Payload.CFERevision           : 99
HousekeepingTlm.Payload.CFEMissionRevision    : 0
HousekeepingTlm.Payload.OSALMajorVersion      : 5
HousekeepingTlm.Payload.OSALMinorVersion      : 0
HousekeepingTlm.Payload.OSALRevision          : 0
HousekeepingTlm.Payload.OSALMissionRevision   : 255
HousekeepingTlm.Payload.PSPMajorVersion       : 1
HousekeepingTlm.Payload.PSPMinorVersion       : 4
HousekeepingTlm.Payload.PSPRevision           : 0
HousekeepingTlm.Payload.PSPMissionRevision    : 99
HousekeepingTlm.Payload.SysLogBytesUsed       : 3010
HousekeepingTlm.Payload.SysLogSize            : 3072
HousekeepingTlm.Payload.SysLogEntries         : 63
HousekeepingTlm.Payload.SysLogMode            : DISCARD
HousekeepingTlm.Payload.ERLogIndex            : 2
HousekeepingTlm.Payload.ERLogEntries          : 2
HousekeepingTlm.Payload.RegisteredCoreApps    : 5
HousekeepingTlm.Payload.RegisteredExternalApps: 6
HousekeepingTlm.Payload.RegisteredTasks       : 16
HousekeepingTlm.Payload.RegisteredLibs        : 2
HousekeepingTlm.Payload.ResetType             : 1
HousekeepingTlm.Payload.ResetSubtype          : 1
HousekeepingTlm.Payload.ProcessorResets       : 1
HousekeepingTlm.Payload.MaxProcessorResets    : 2
```

cFE apps

Apps loaded via cfe_es_startup.scr

Executive Services Exercises - 4

5. Open CFE_ES APP_TLM and issue CFE_ES *QueryOneCmd* for Basecamp's demo app APP_C_DEMO



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 4^{th} 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

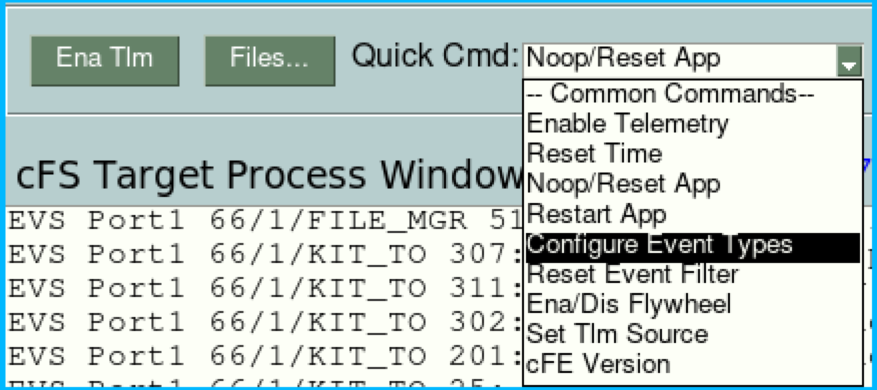
- **Processor scope**
  - Enable/disable event messages based on type
    - Debug, Information, Error, Critical

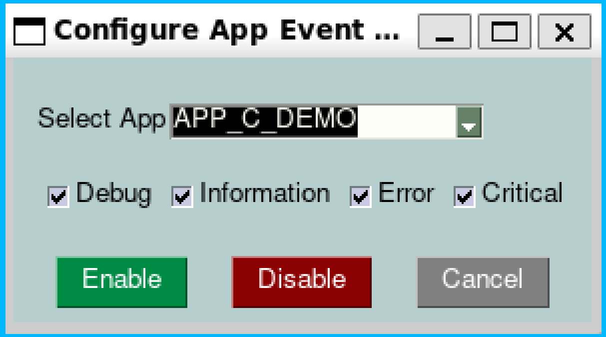
- **Application scope**
  - Enable/disable all events
  - Enable/disable based on type

- **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

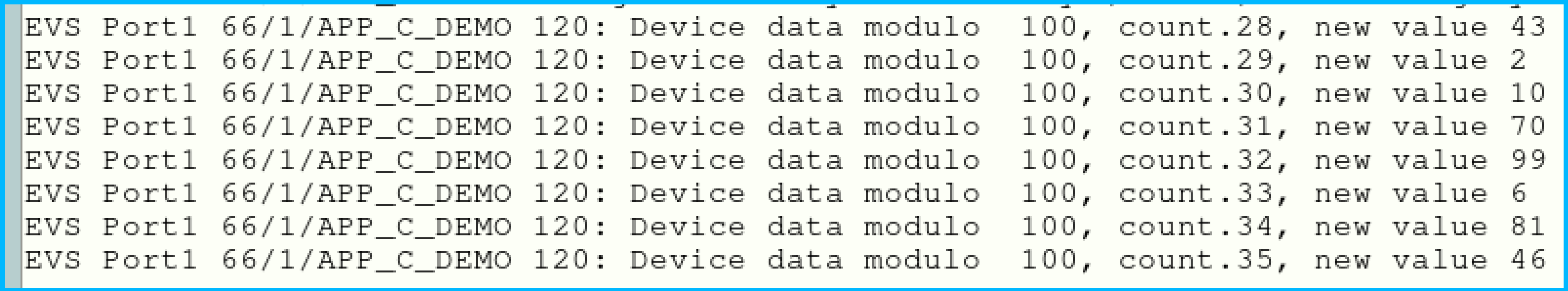
Event Service Exercises - 1

1. Use the *Quick Command* menu to enable debug messages for the demo app, APP_C_DEMO





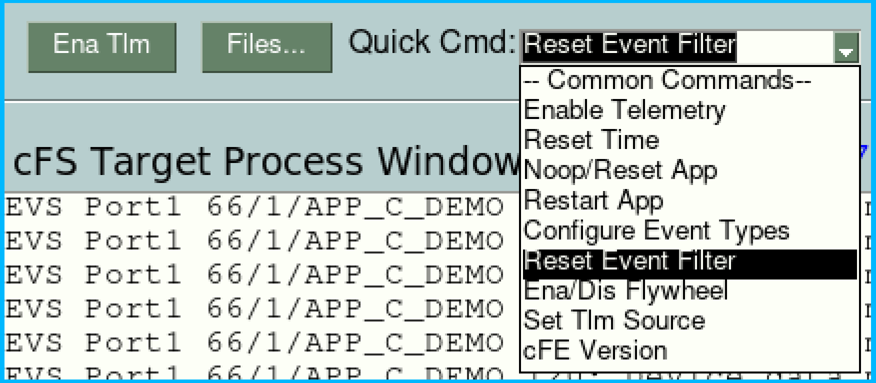
2. In the *cFS Target Process Window* you should see 8 debug event messages because this event is filtered to only send the first 8 occurrences

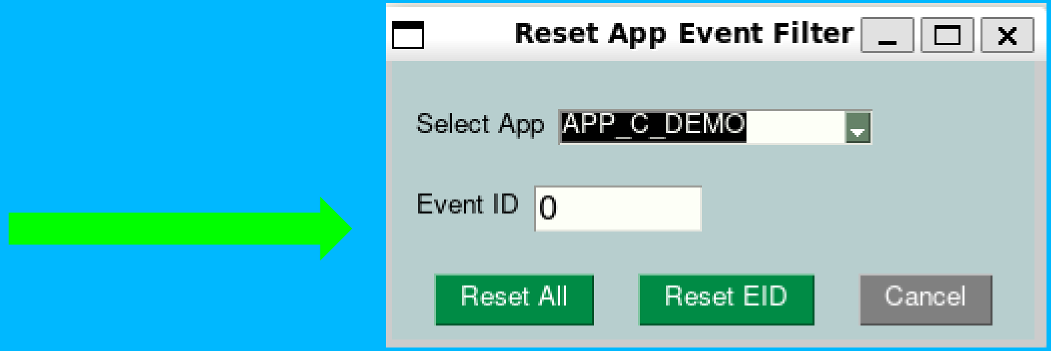


```
EVS Port1 66/1/APP_C_DEMO 120: Device data modulo 100, count.28, new value 43
EVS Port1 66/1/APP_C_DEMO 120: Device data modulo 100, count.29, new value 2
EVS Port1 66/1/APP_C_DEMO 120: Device data modulo 100, count.30, new value 10
EVS Port1 66/1/APP_C_DEMO 120: Device data modulo 100, count.31, new value 70
EVS Port1 66/1/APP_C_DEMO 120: Device data modulo 100, count.32, new value 99
EVS Port1 66/1/APP_C_DEMO 120: Device data modulo 100, count.33, new value 6
EVS Port1 66/1/APP_C_DEMO 120: Device data modulo 100, count.34, new value 81
EVS Port1 66/1/APP_C_DEMO 120: Device data modulo 100, count.35, new value 46
```

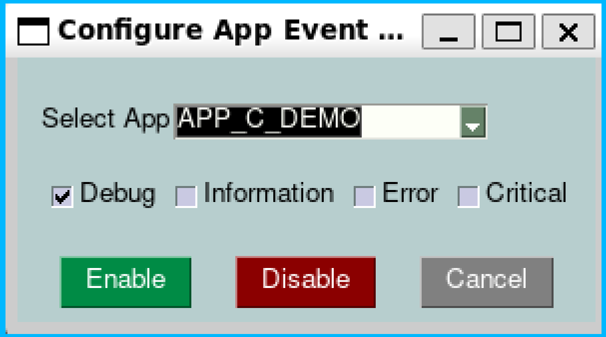
Event Service Exercises - 2

3. Use the *Quick Command* menu to reset all APP_C_DEMO's event filters. The Event ID can be left at 0 because it is not used in the *Reset All* command. Eight more debug event messages should appear in the *cFS Target Process Window.*





4. Use the *Quick Command* menu to disable APP_C_DEMO's debug events. Note you must unselect the other events so they don't get disabled. Repeat exercise 3 and you should not see any debug events.



Software Bus Services Overview

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

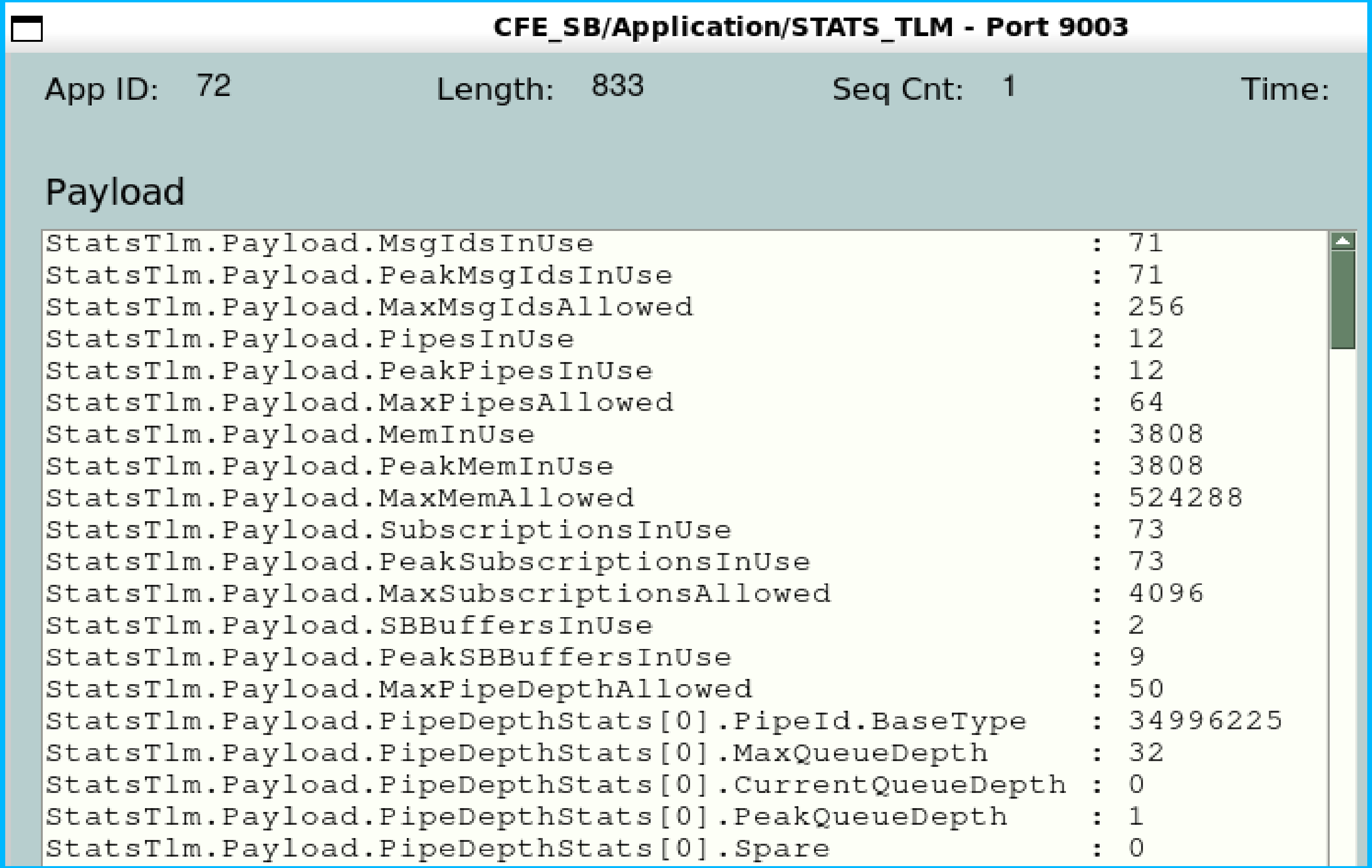
- **Routes messages to all applications that have subscribed to the message (i.e. broadcast model)**
  - Subscriptions are done at application startup
  - Message routing can be added/removed at runtime
  - Sender does not know who subscribes (i.e. connectionless)

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

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

Software Bus Exercises - 1

1. First open CFE_SB's *STATS_TLM* window and then send CFE_SB's *SendSbStatsCmd* command. This will populate the *STATS_TLM* window. APP_D_DEMO's command pipe has a depth of 7 which is unique.



APP_C_DEMO's pipe stats

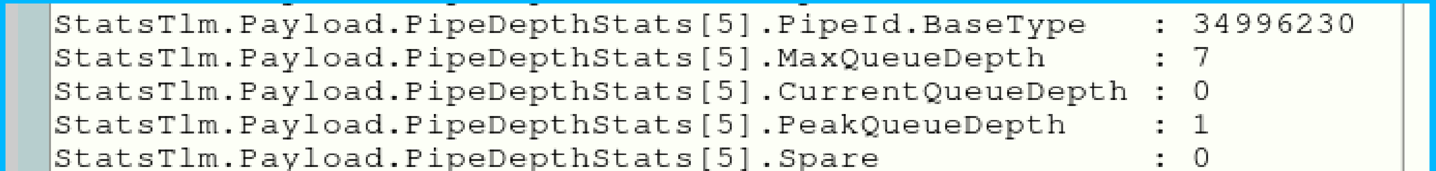
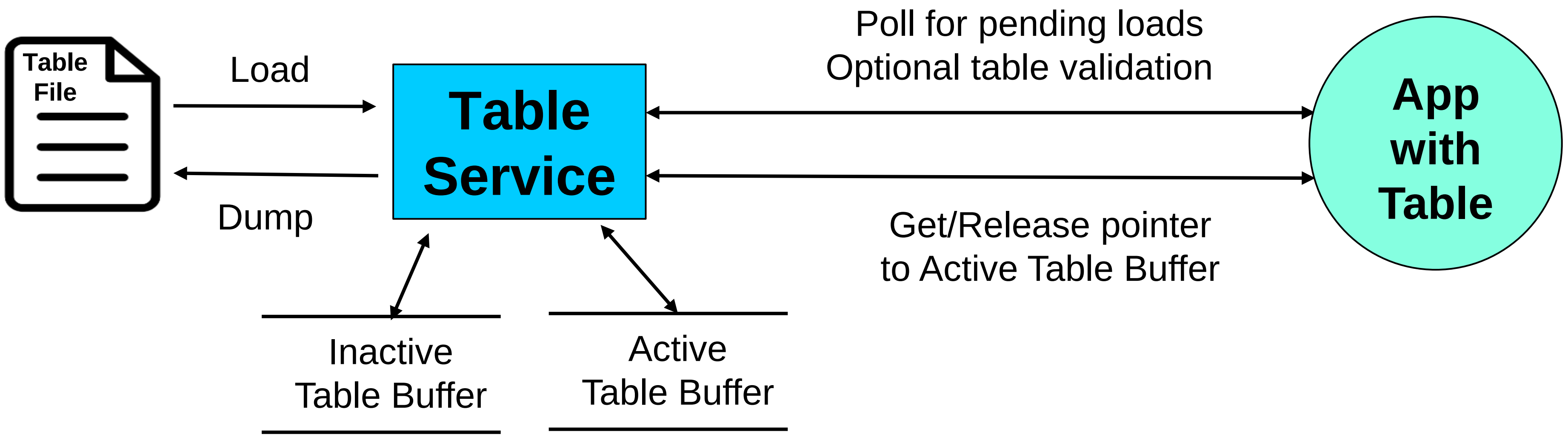


Table Service Overview

- **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**

Table Service Exercises - 1

1. Open CFE_TBL's Housekeeping (status) telemetry and browse its content
   - Basecamp uses app managed JSON table files except for File Manager (FILE_MGR)
   - Basecamp's FILE_MGR app is NASA's FM app refactored to use Basecamps app framework, however the original cFE binary table file was retained

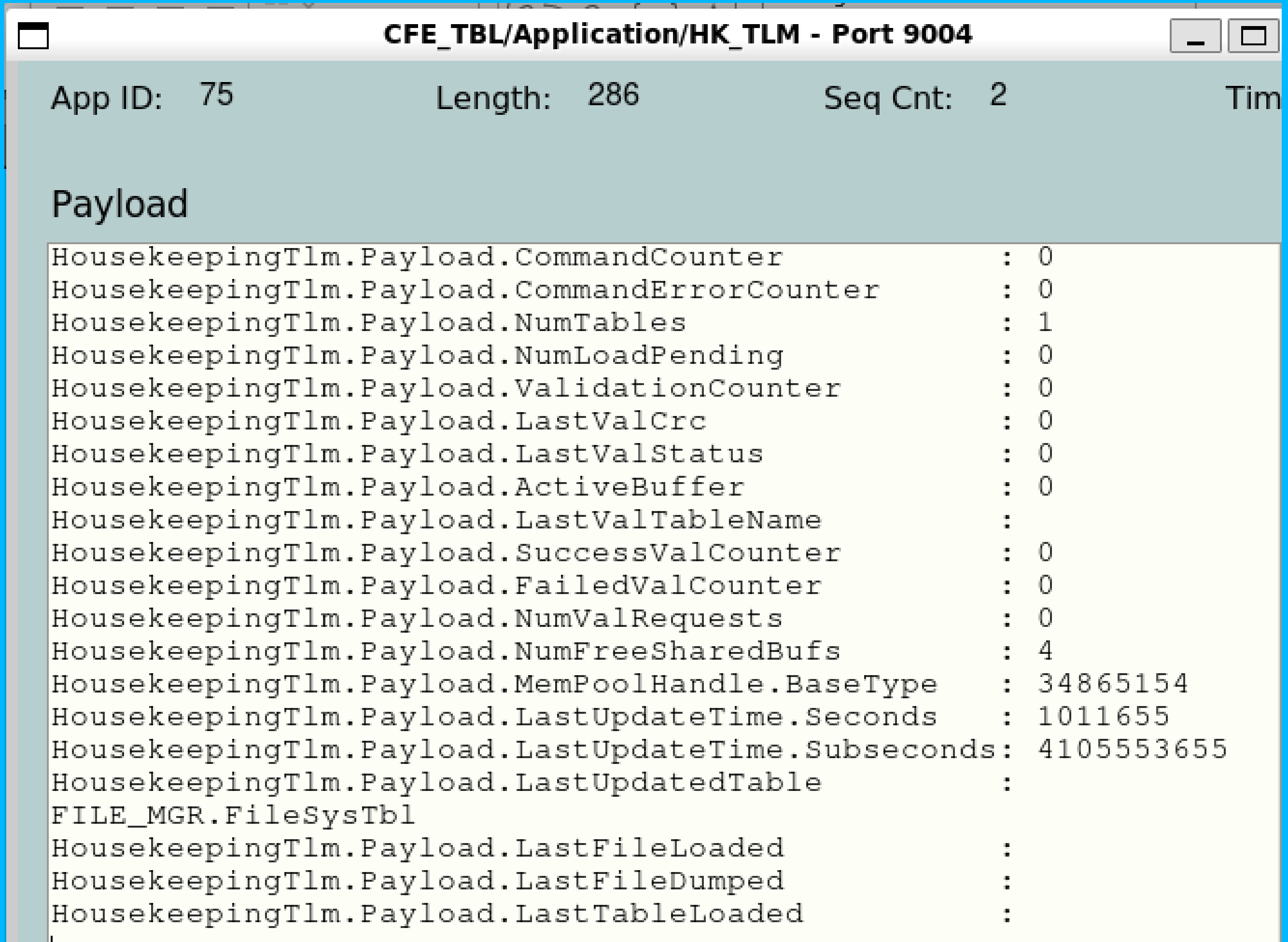
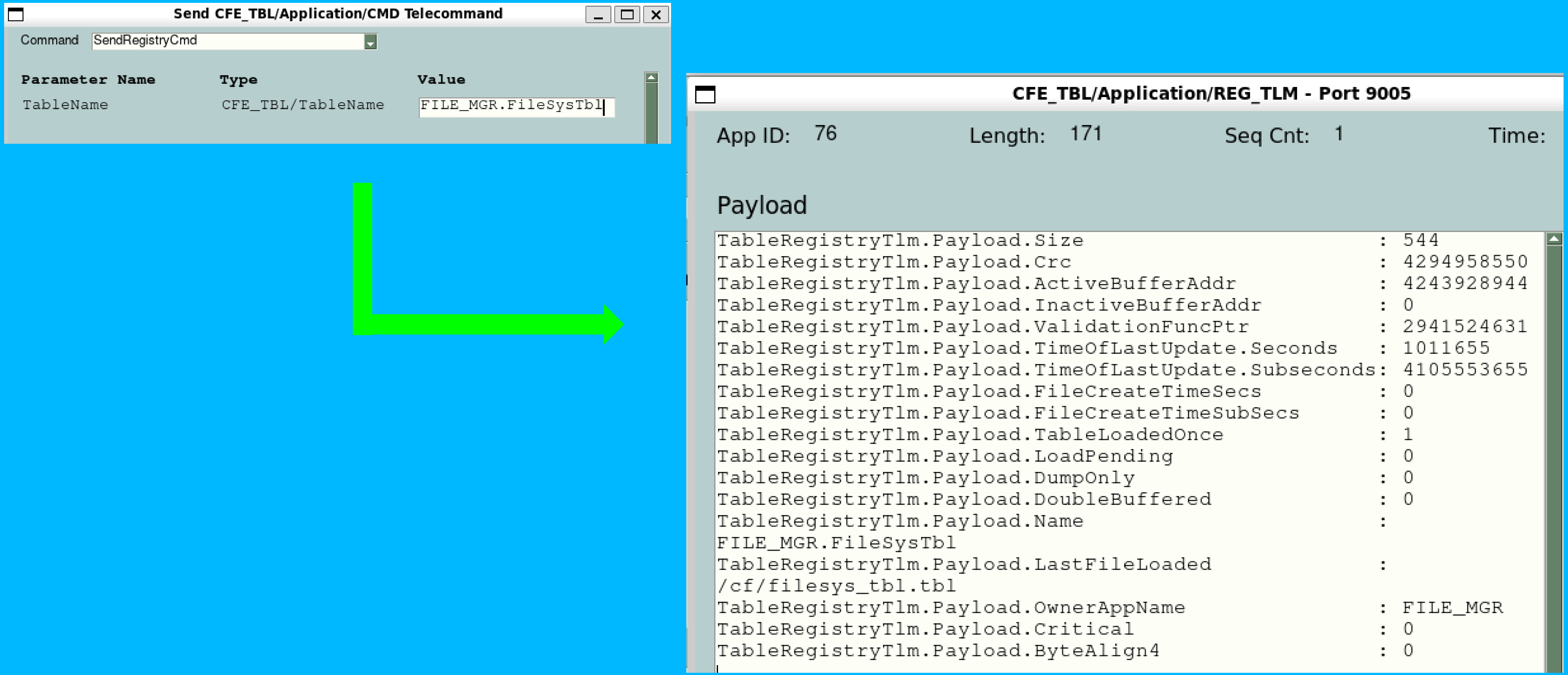


Table Service Exercises - 2

2. First open CFE_TBL's *REG_TLM* window and then send CFE_TBL's *SendRegistryCmd* command for FILE_MGR.FileSysTbl. This will populate the *REG_TLM* window with FILE_MGR.FileSysTbl details.



Time Service Overview

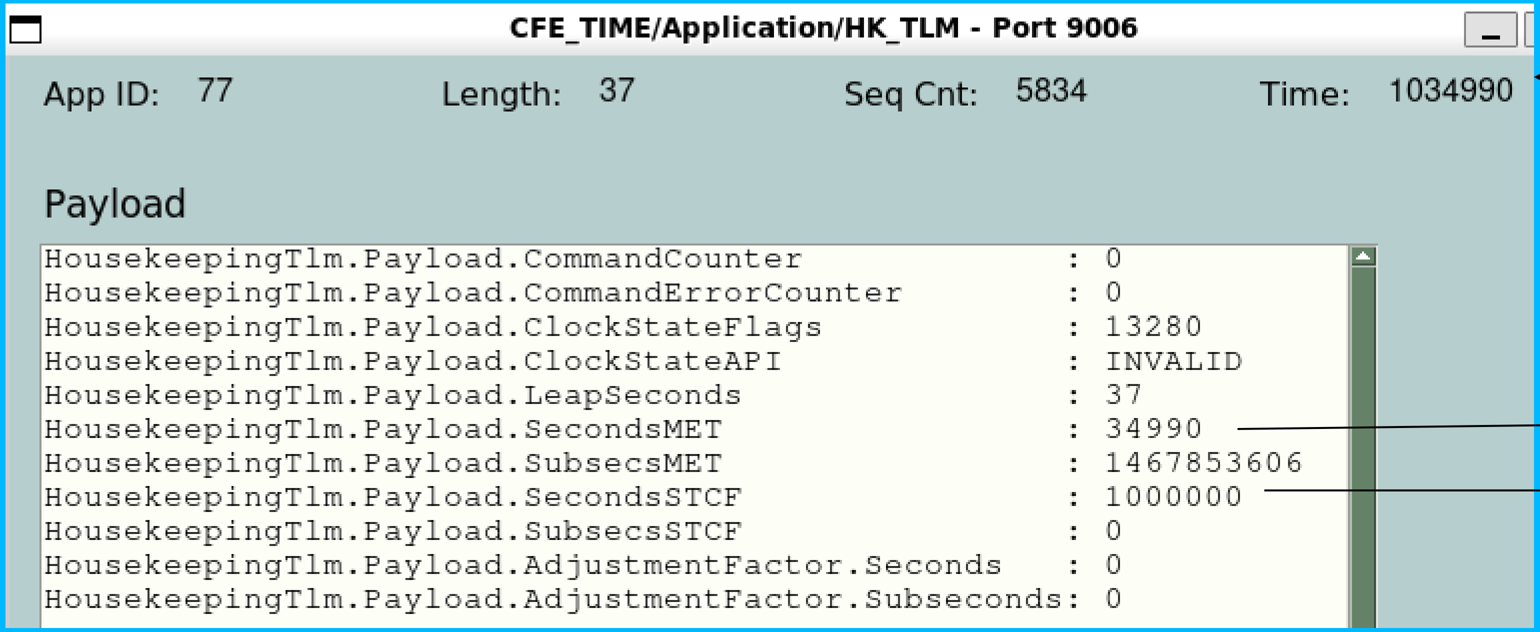
- cFE Time Services provides time correlation, distribution and synchronization services
- Provides a user interface for correlation of Spacecraft Time to the ground reference time (epoch)
- Provides calculation of spacecraft time, derived from mission elapsed time (MET), a spacecraft time correlation 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

Time Service Exercises - 1

1. Open CFE_TIME's Housekeeping telemetry window and observe the following



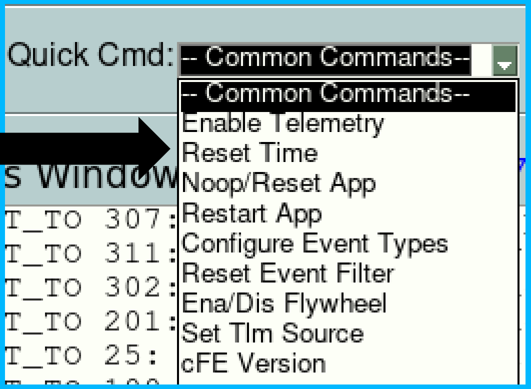
Telemetry message headers time stamped with Spacecraft Time

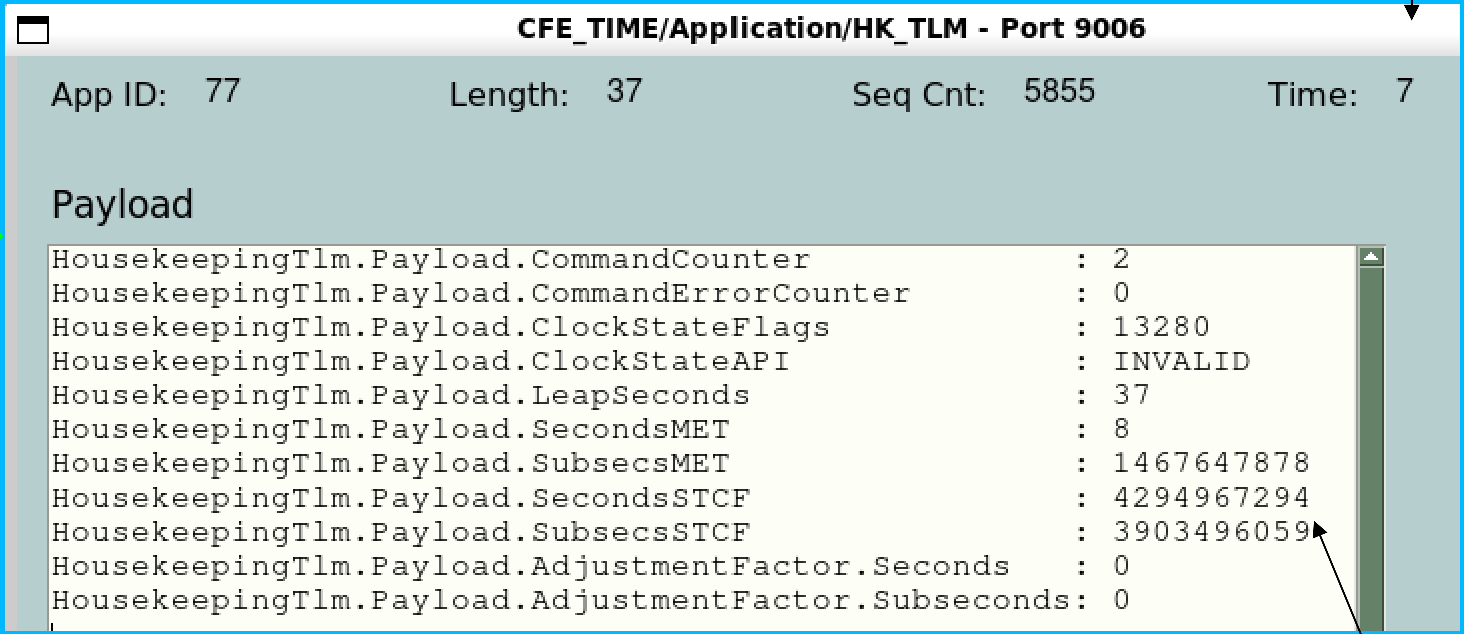
```
    34990 MET
+ 1000000 STCF
--------------
  1034990 Spacecraft Time
```

Time Service Exercises - 2

2. Use the Quick Command menu to Reset Time
   - Sends CFE_TIME's SetMET and SetTime commands with 0 for seconds and sub seconds

Time reset to zero





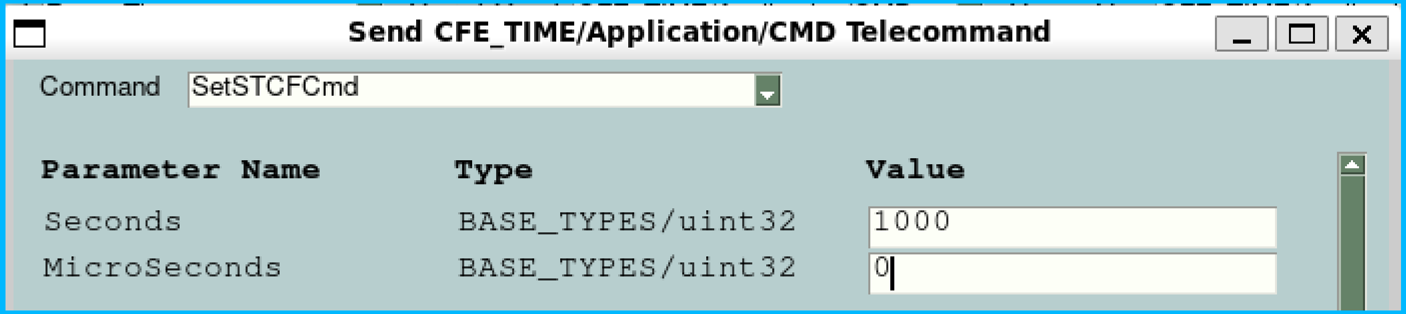
TBD: What's going on with STCF?

cFS Target Process Window displays events for each command

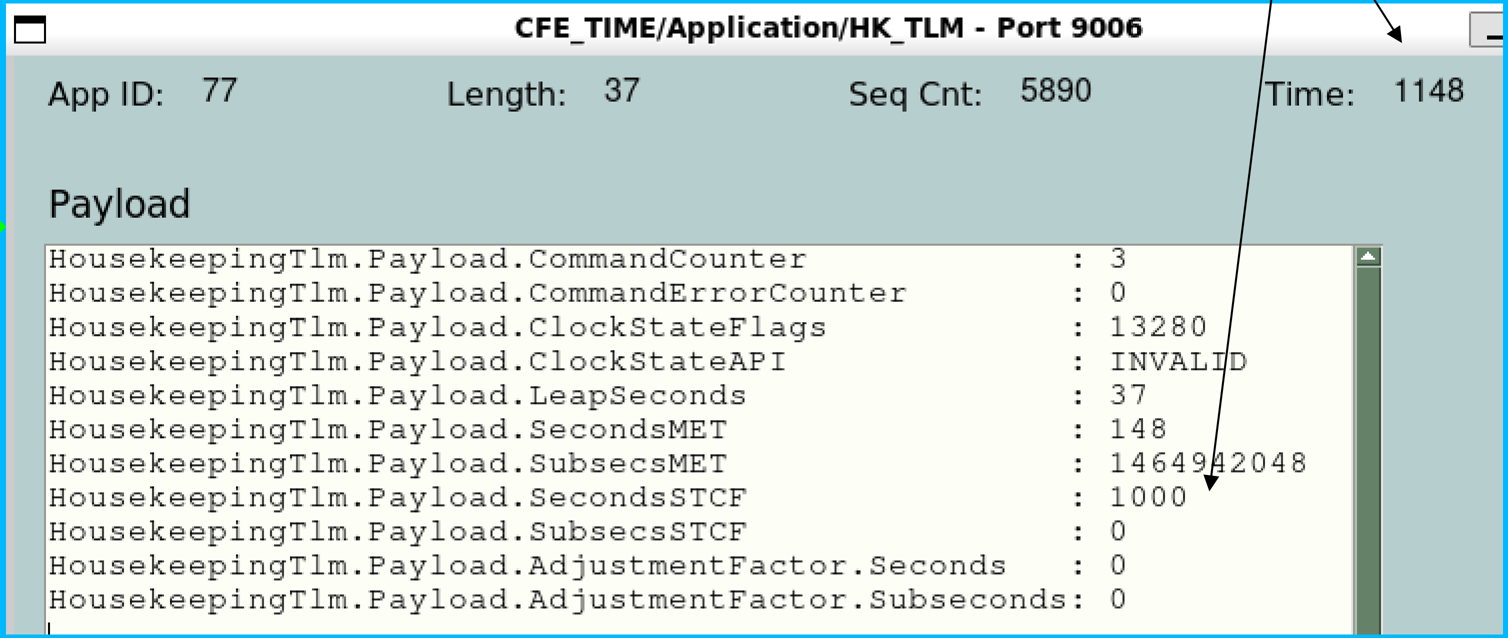
```
EVS Port1 66/1/CFE_TIME 13: Set MET -- secs = 0, usecs = 0, ssecs = 0x0
EVS Port1 66/1/CFE_TIME 12: Set Time -- secs = 0, usecs = 0, ssecs = 0x0
```

Time Service Exercises - 3

3. Send a SetSTCFCmd

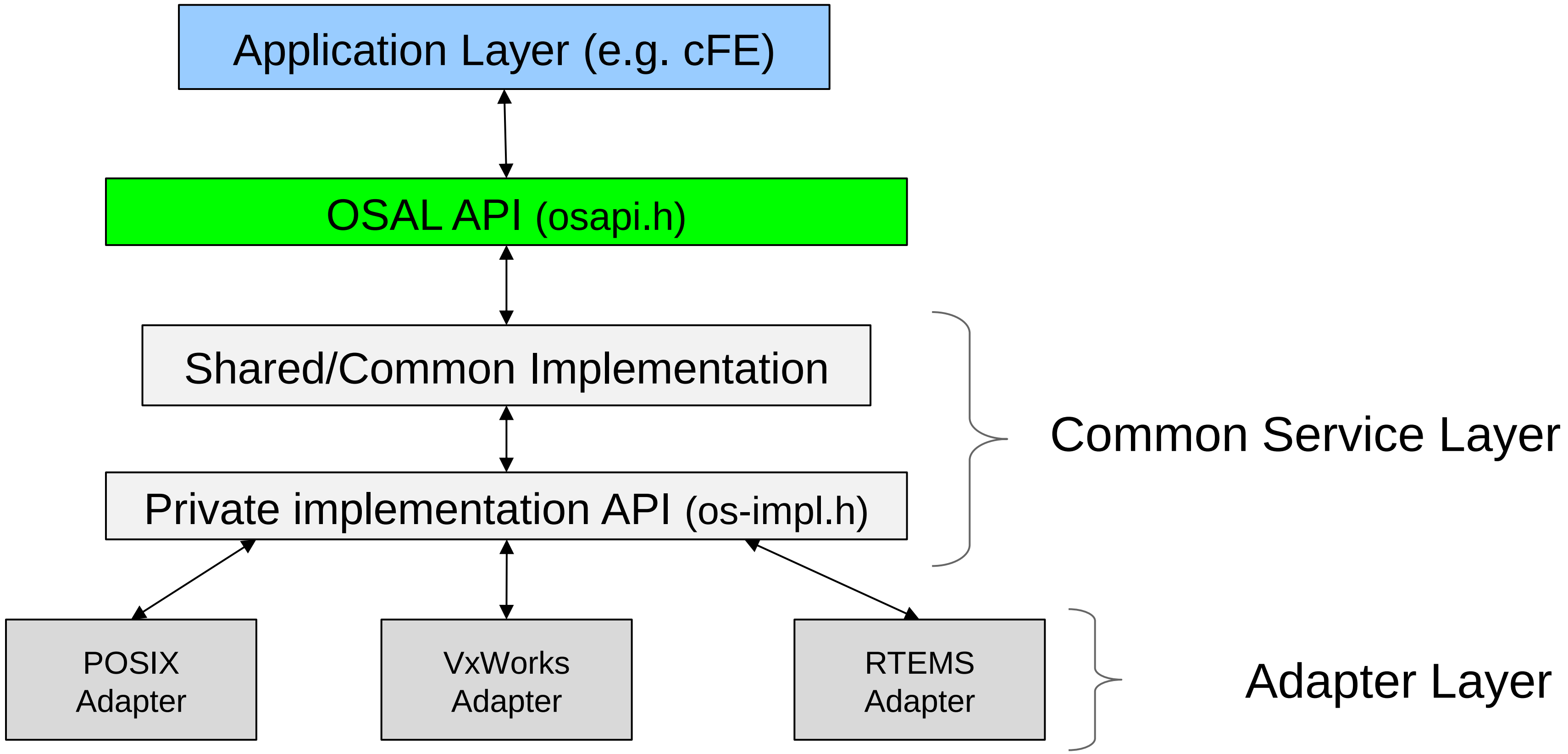


STCF Updated



OSAL Layered Architecture

- **The OSAL has it's own layered architecture with an "application" API that is portable across different operating systems**
  - The OSAL is a product that can be used independent of the cFS
  - The cFE is an OSAL application



OSAL Layered Design Strategy

- **Maximize common service layer functionality and minimize OS adapter code**
  - Avoid duplicate implementations in each OS Adapter
  - Perform validation and error checking in the Common Service Layer so OS adapters can trust data and minimize logic
- **Ensure consistent behavior across operating systems**
  - No chance of one OS checking e.g. if an input is NULL but not the others.
  - If a race condition is found, it would be applied to all OS’s with only one fix to the Common Service Layer. No additional work to copy to other implementations.
- **Use robust design practices in the Common Service Layer**
  - Mutex/Reference count every operation that needs it
  - “Best practices” for Symmetric Multiprocessing and highly multi-threaded systems
- **Reduce the cost of adding new OS Adapters**
  - Only “business logic” that is specific to an OS should be implemented

Common Service Layer

- **Manage common operating resources like tasks, queues, mutexes, etc. as *"Objects"***
- **Each Object has a Type and an Identifier (ID)**
  - A separate number space is used for each ID Type
  - Up to 64K allocations need to occur before and ID is repeated/reused
  - Zero is not a valid ID to catch uninitialized variable errors
- **Use Tables (not cFE tables) to manage Objects**
  - OS independent locking / unlocking semantics
- **Check the validity and state of a passed-in IDs across all Object Types and all OS implementations**
  - If valid, issue new Object ID's and find open Table entries

Platform Support Package (PSP)

- **The PSP functions complete the Platform Abstraction API that is required by the cFE**
  - They serve as the "glue" between the OSAL/RTOS and the cFE Flight Software filling gaps that are not considered part of the OSAL

- **It’s architectural role is equivalent to an OS Adapter with a slightly different scope**
  - A new implementation must be created for each platform

- **CFE_PSP_Main() is the entry point that an RTOS calls to start the cFE**
  - It performs any BSP/RTOS specific setup and then calls the cFE main entry point

- **The cfe_psp.h file defines the entire API**
  - Example functions include getting the processor restart type, flushing a cache, etc.

Application Layer
Architectural Components

Application Layer Introduction

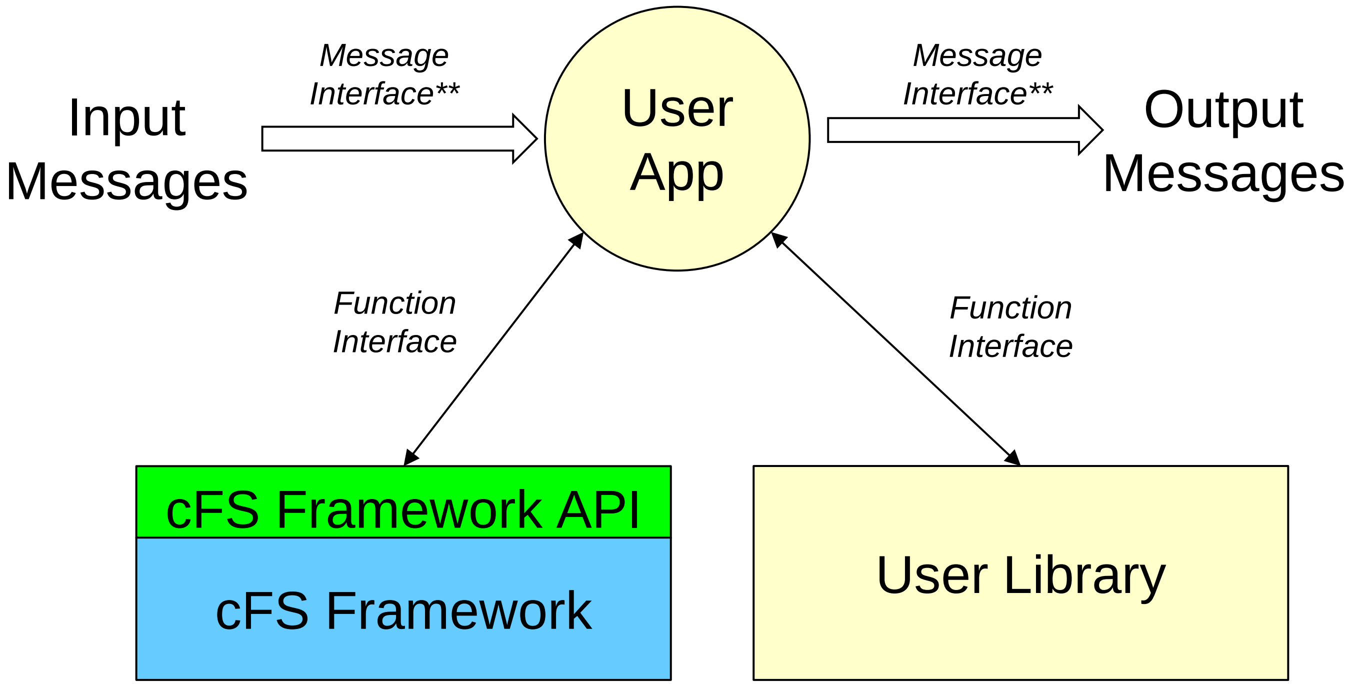
- **The cFS Application Layer contains *Library* and *Application* architectural components**

- **Architectural components have well defined context, interfaces, and relationships with other components**

- **A cFS library/application context is bounded by the following interfaces (see next slide)**
  1. cFE, OSAL, and PSP services and Application Programmer Interfaces (APIs)
  2. The functional interfaces (APIs) defined by libraries
  3. The message interfaces defined by applications
  4. Platform-specific APIs

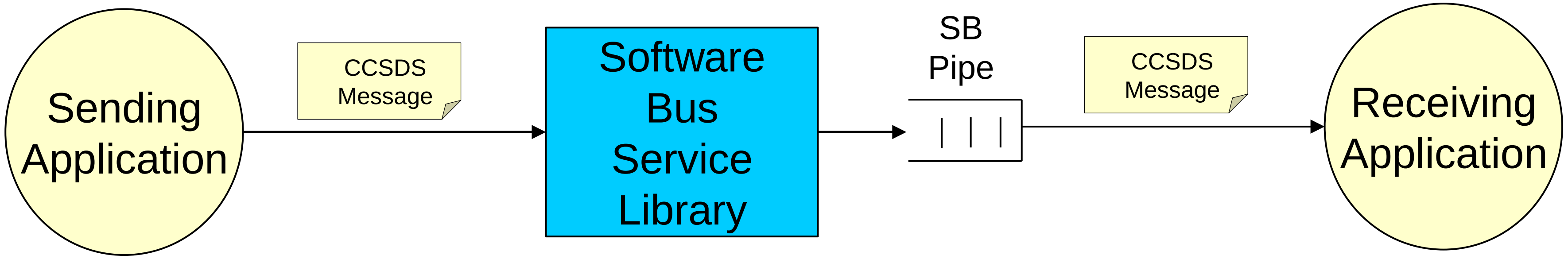
- **This section discusses the first three interfaces**
  – The cFS Systems Engineering document includes design patterns for using platform-specific APIs

Library & Application Context



** Note that if messages are standardized then the concept of service apps could be established

Message-Centric Application Design



- **Inter-app communications using the *Software Bus* is an essential to app designs**
- **The *Software Bus* uses a one-to-many message broadcast model**
  - Applications publish messages without knowledge of destinations
- **To receive messages, applications create an *SB Pipe (a FIFO queue)* and subscribe to 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

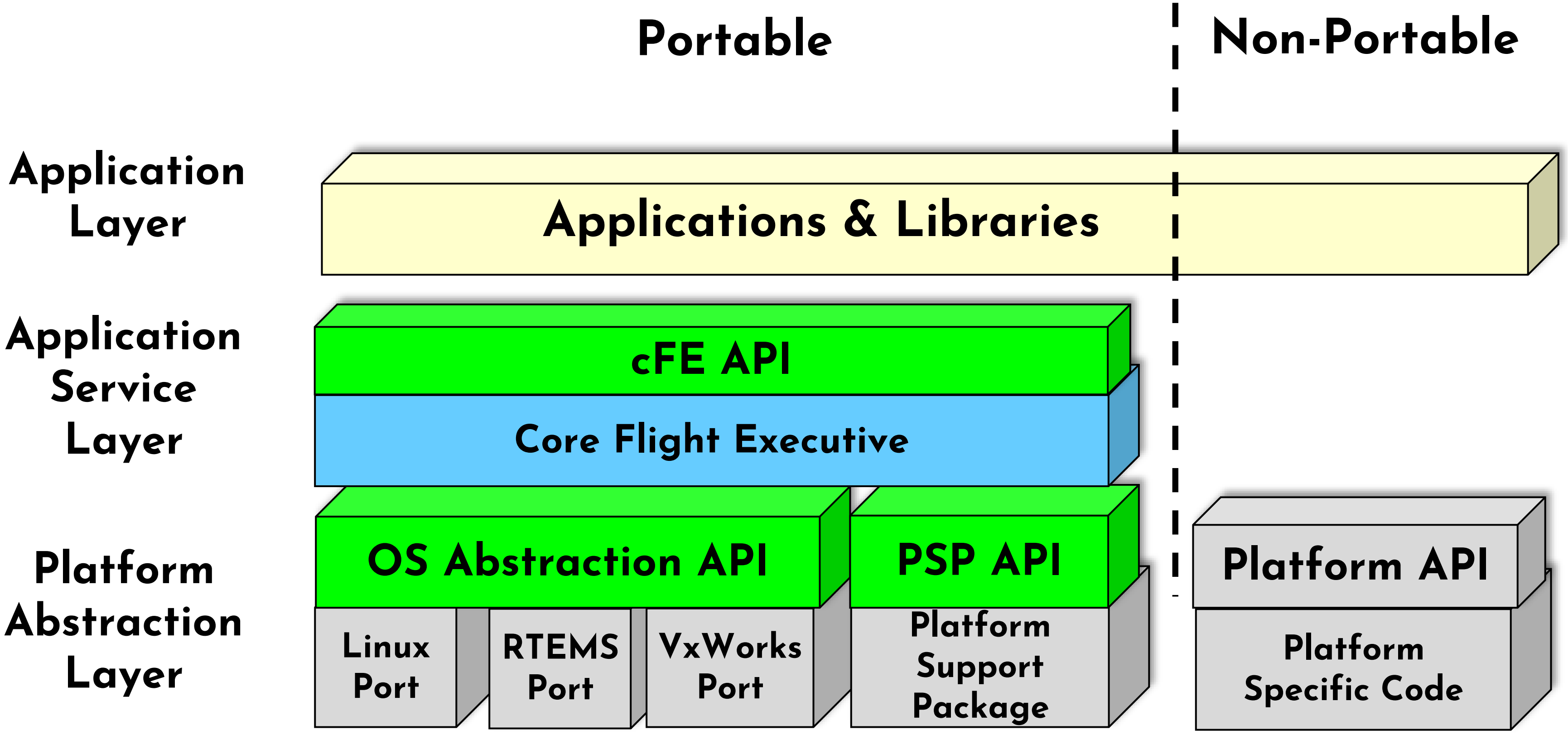
Application Layer Design Concepts

- **Architecture component dynamic deployment model****
  - cFS Framework is built as a single binary image
  - Libraries and applications built as individual object files
  - A startup script defines which lib/app objects are loaded

- **An essential cFS architectural design concept is that libraries and apps can be designed with interdependencies that allow groups of libs/apps to provide mission functionality**
  - Library interfaces are similar to the cFS's layered interfaces
  - App message interfaces are peer-to-peer, however a dependency hierarchy can be created based on the producer-consumer relationships
  - This section introduces the "Operations Service App Suite" that must be included in every cFS target
  - The cFS Systems Engineering document contains more examples functional lib/app groups

** Dynamic (runtime binding) is the default configuration. Static linking is supported.

Library & Application Portability/Reusability



- **Libraries and apps are portable if they only use the cFS APIs shown in green**
- **Many mission specific apps may not be portable to a new mission unless the same platform is used**
  - This does not make them “non-compliant” components, just non-portable
  - Organizational goals, budgets and schedules drive whether to develop reusable component decisions

Applications (1 of 2)

- **What is an Application?**
  - A thread of execution managed by the platform's operating system
  - They acquire and own resources using the Platform Abstraction and Application Service APIs

- **Resources are typically acquired during initialization and released when an application terminates**
  - Helps create a deterministic steady-state system
  - Helps achieve the architectural goal for a loosely coupled system that is scalable, interoperable, testable (each app can be separately unit tested), and maintainable

- **Apps can be reloaded during operations without rebooting**

Applications (2 of 2)

- **Concurrent execution model**
  - Each app has its own priority-based thread of execution and can spawn child tasks with their own priority-based thread of execution
  - Supports complex realtime mission requirements

- **Reusable apps only use the cFS APIs**
  - Write once run anywhere the cFS framework has been deployed
  - Can be written in a desktop environment deferring embedded software development complexities such as cross compilation and target operating systems
  - Provides seamless application transition from technology efforts to flight projects
  - More powerful than the Smartphone situation where different apps are written for each platform

Libraries (1 of 2)

- **What is a library?**
  - A collection of functions and data that are available for use by apps and other libraries
  - Architecturally they exist within the application layer
  - They cannot create tasks and they assume the AppID/TaskID of the caller

- **Libraries are <u>not</u> registered with Executive Services and do not have a thread of execution, so they have limited cFE API usage. For example,**
  - A library can't call CFE_EVS_Register() during initialization
  - The ES API does <u>not</u> 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

Libraries (2 of 2)

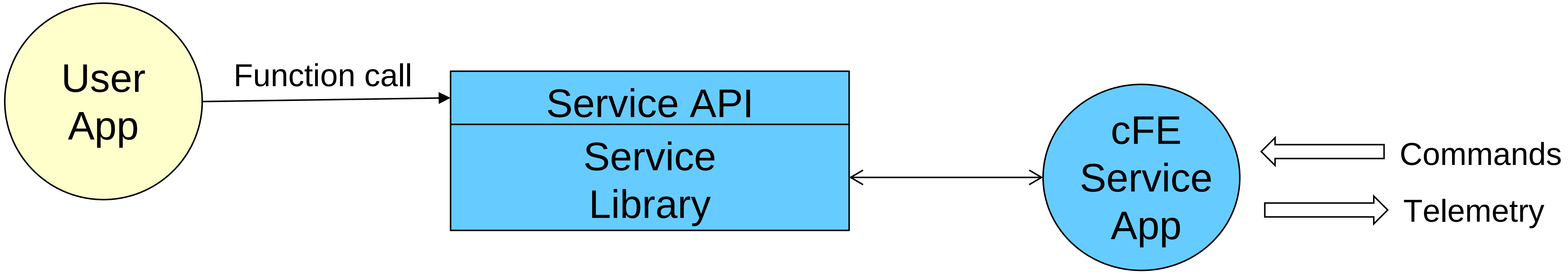
- **Libraries can either be statically or dynamically linked**
  - Dynamic linking requires support from the underlying operating system

- **No cFE API exists to retrieve library code segment addresses**
  - Prevents apps like Checksum from accessing library code space

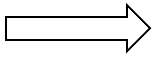
- **Libraries are specified in the cfe_es_startup.scr and loaded during cFE initialization**
  - When using dynamic linking, libraries must be loaded prior to components that use them

- **For libraries that require a ground interface, or some other more complex runtime environment, a helper app is created to provide this support**
  - The cFE's service design uses this approach

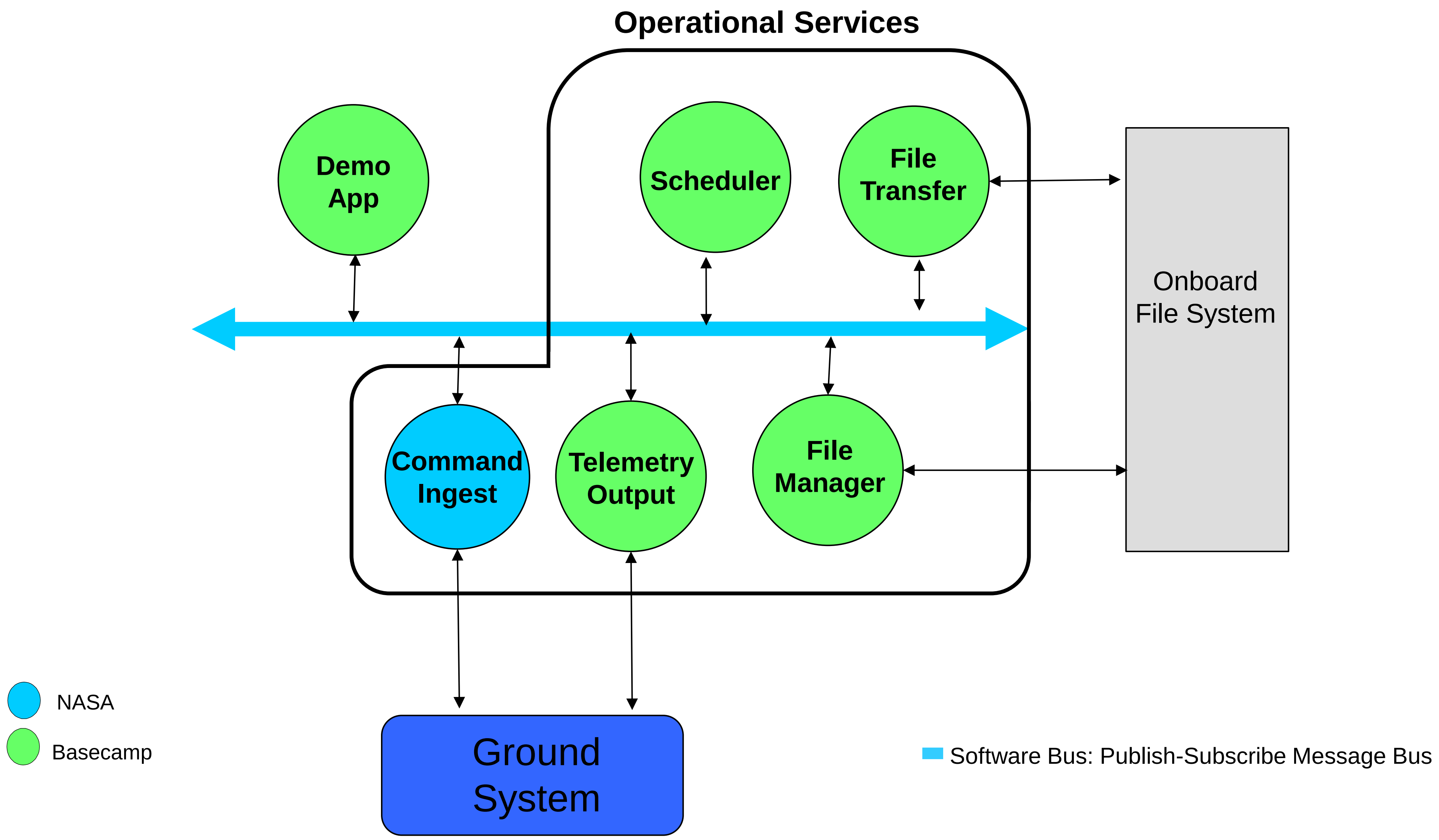
cFE Service Design



- **Each cFE service has**
  - A <u>library</u> that is used by applications
  - An <u>application</u> 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"**
  - Housekeeping is an historical term that means an application's status
- **You can obtain additional service information beyond the HK packet with commands that**
  - Send one-time telemetry packets
  - Write onboard service configuration data to files



Basecamp cFS Target

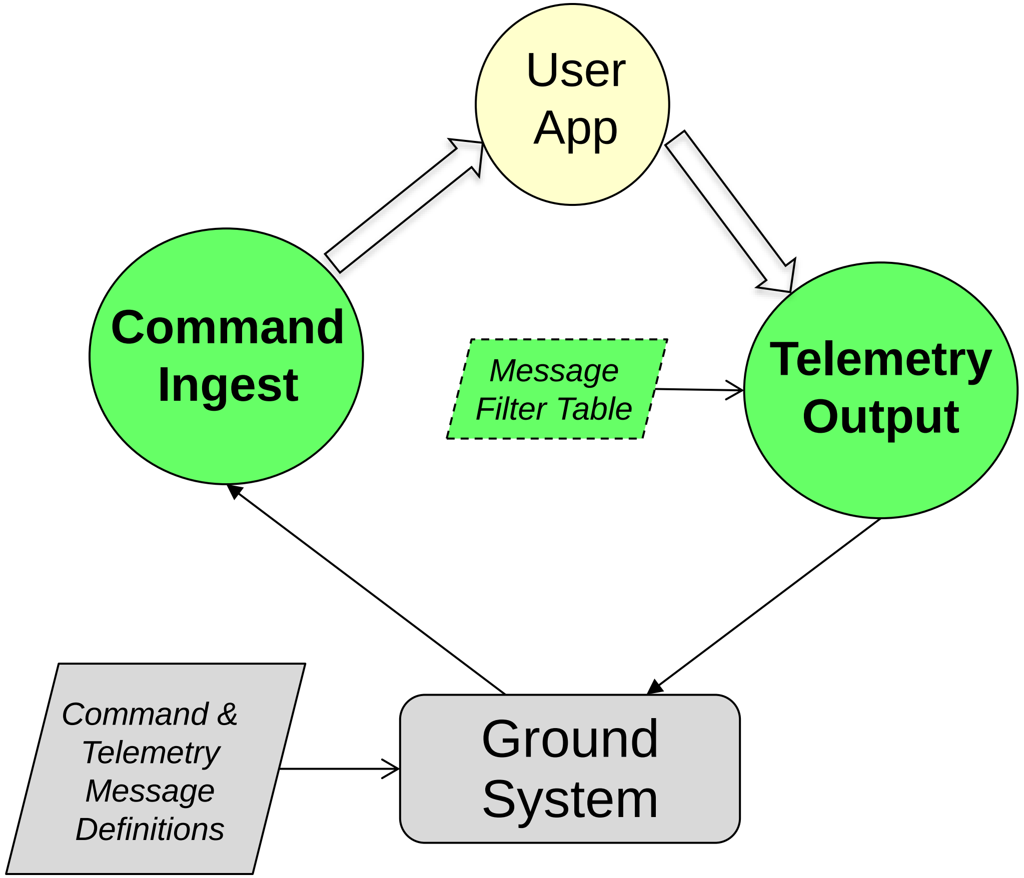


Operational Services App Suite

- **The Operations Service App Suite is a group of apps that provide functionality required by every cFS target in an operational system**
  - A target needs to communicate (receive commands and send telemetry) with at least one external system, typically a ground system
  - The cFS relies on files so a mechanism for transferring files between the target and an external system is needed, as well as remotely managing the target's directories and files
  - The cFS promotes designing synchronous systems so having an app synched with a 1Hz signal that sends periodic scheduling messages helps achieve this goal

- **An Operations Service App Suite is included in Basecamp's default Target****

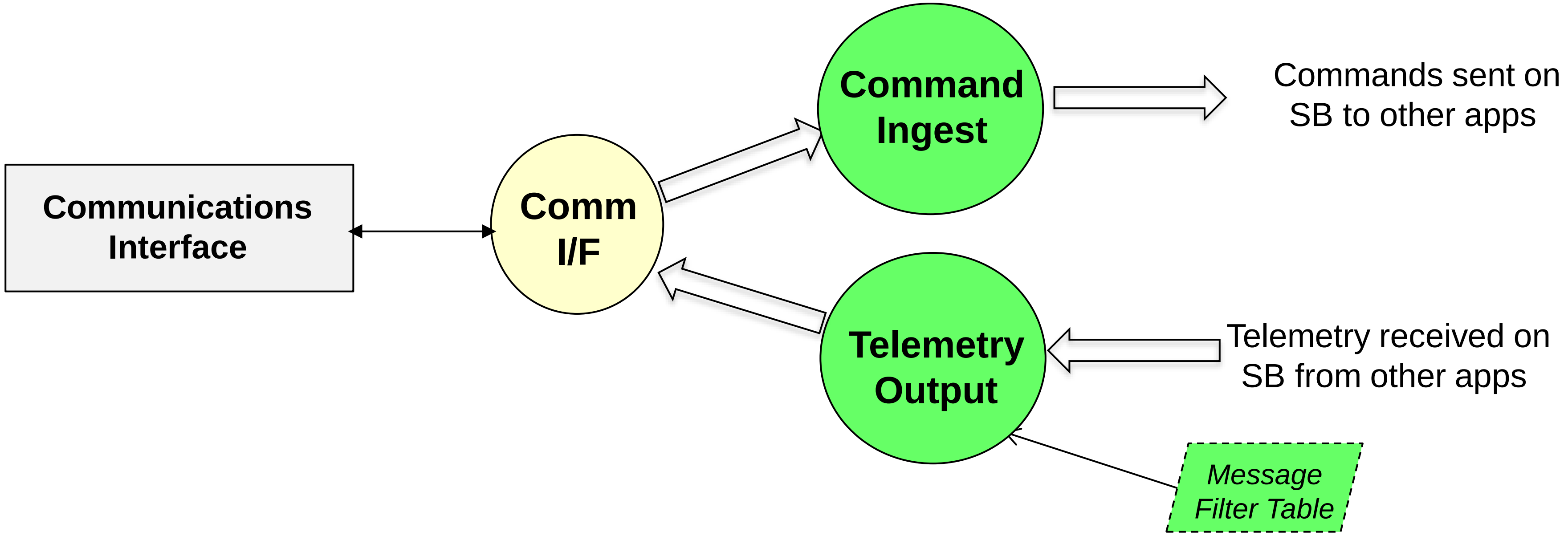
*** A cFS target is an instantiation of the cFE Framework on a platform with a set of library and apps. Not to be confused with a distribution.*



- **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

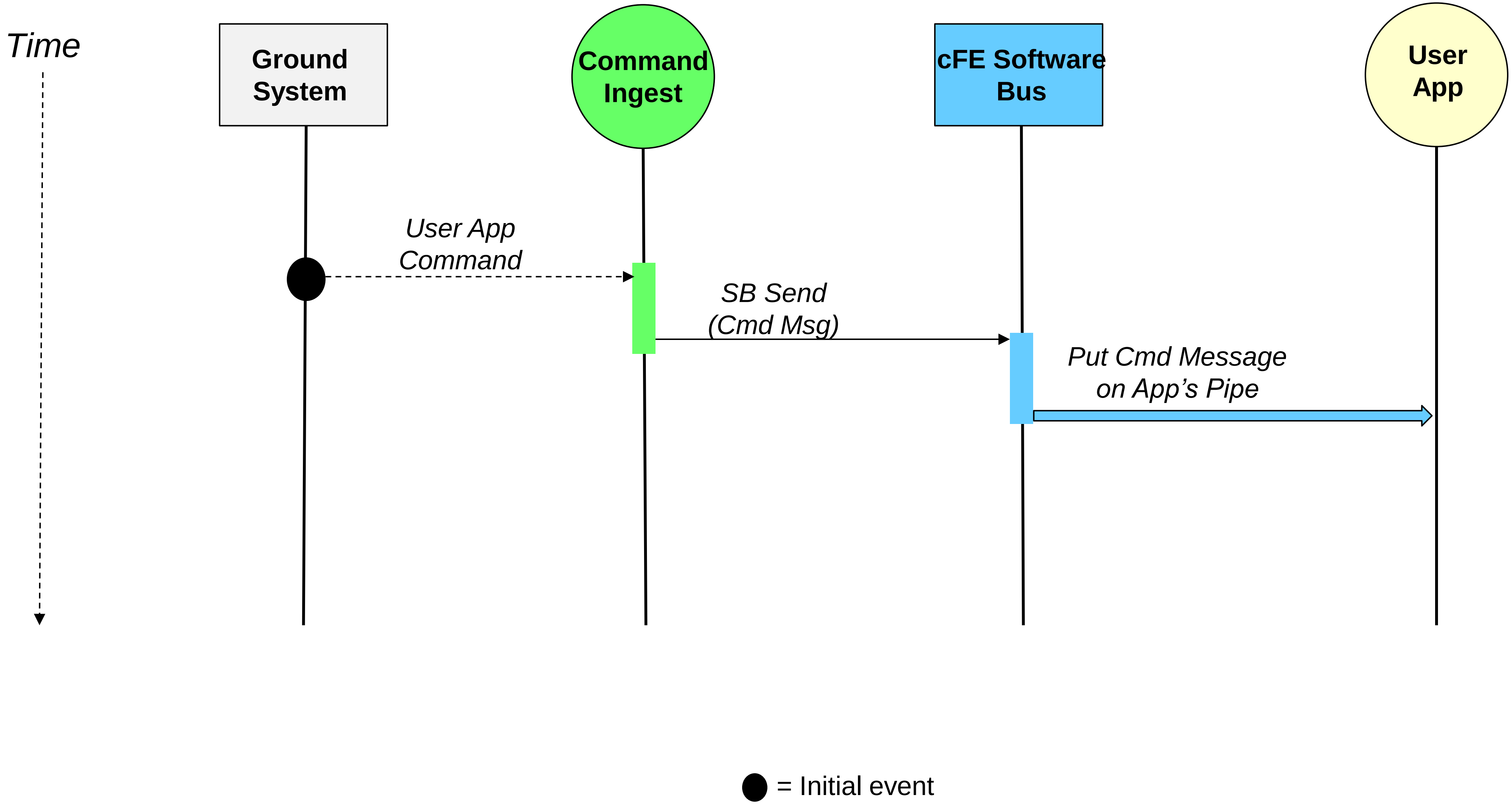
- **The ground and flight messages definitions must match**
  - Basecamp uses Electronic Data Sheets to define messages once and the EDS Toolchain creates ground and flight artifacts
  - In many situations, developers must manually implement separate ground and flight definitions and ensure that they match

Command & Telemetry Context (2 of 2)

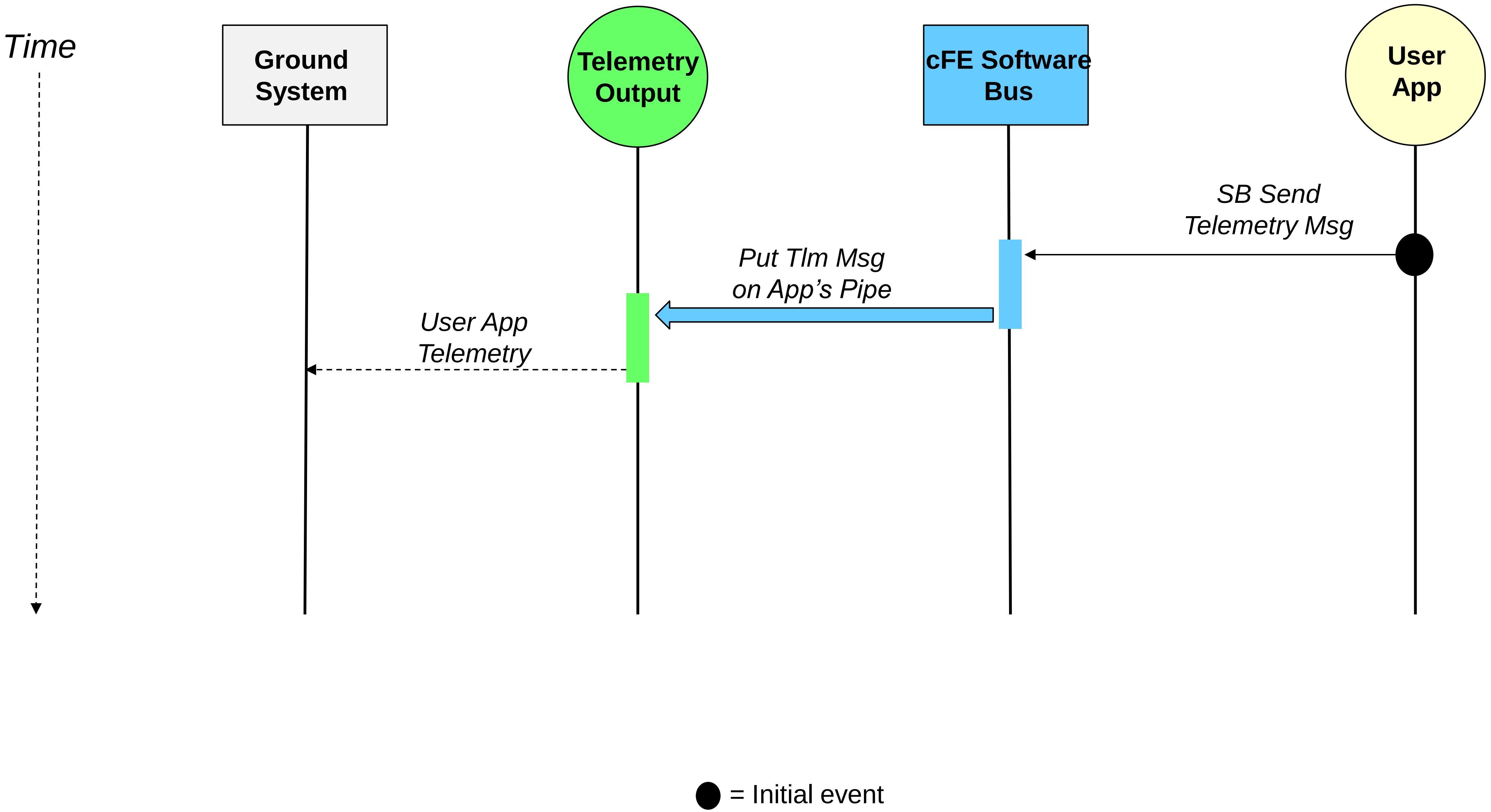


- **Mission external command and telemetry communications is more complicated for embedded systems**
  - An interface app is often used to manage the hardware interface and transferring messages between the Software Bus and the hardware interface
  - The Systems Engineering Document goes into more detail
- **The following versions of CI and TO are available as open source**
  - Basecamp versions use UDP and a JSON-defined filter table
  - cFS Bundle includes ‘lab’ versions that use UDP for the external comm
  - NASA’s Johnson Space Center released versions that use a configurable I/O library for a different external comm links

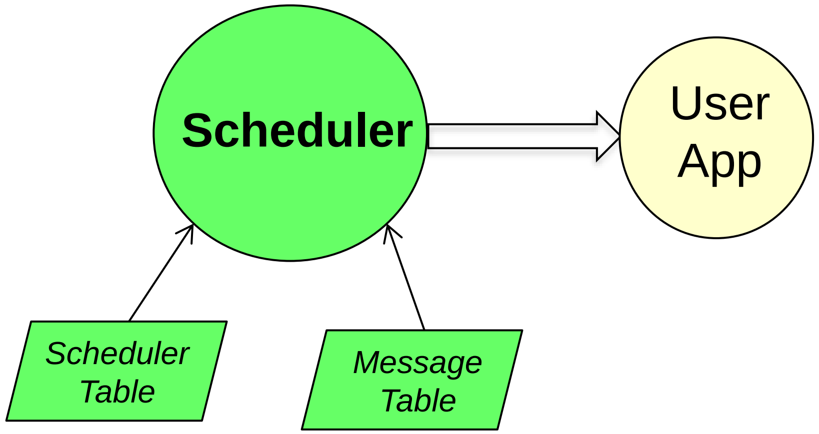
Telecommand from Ground to User App



Telemetry from User App to Ground



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

Suspend execution until a message arrives on app's pipe

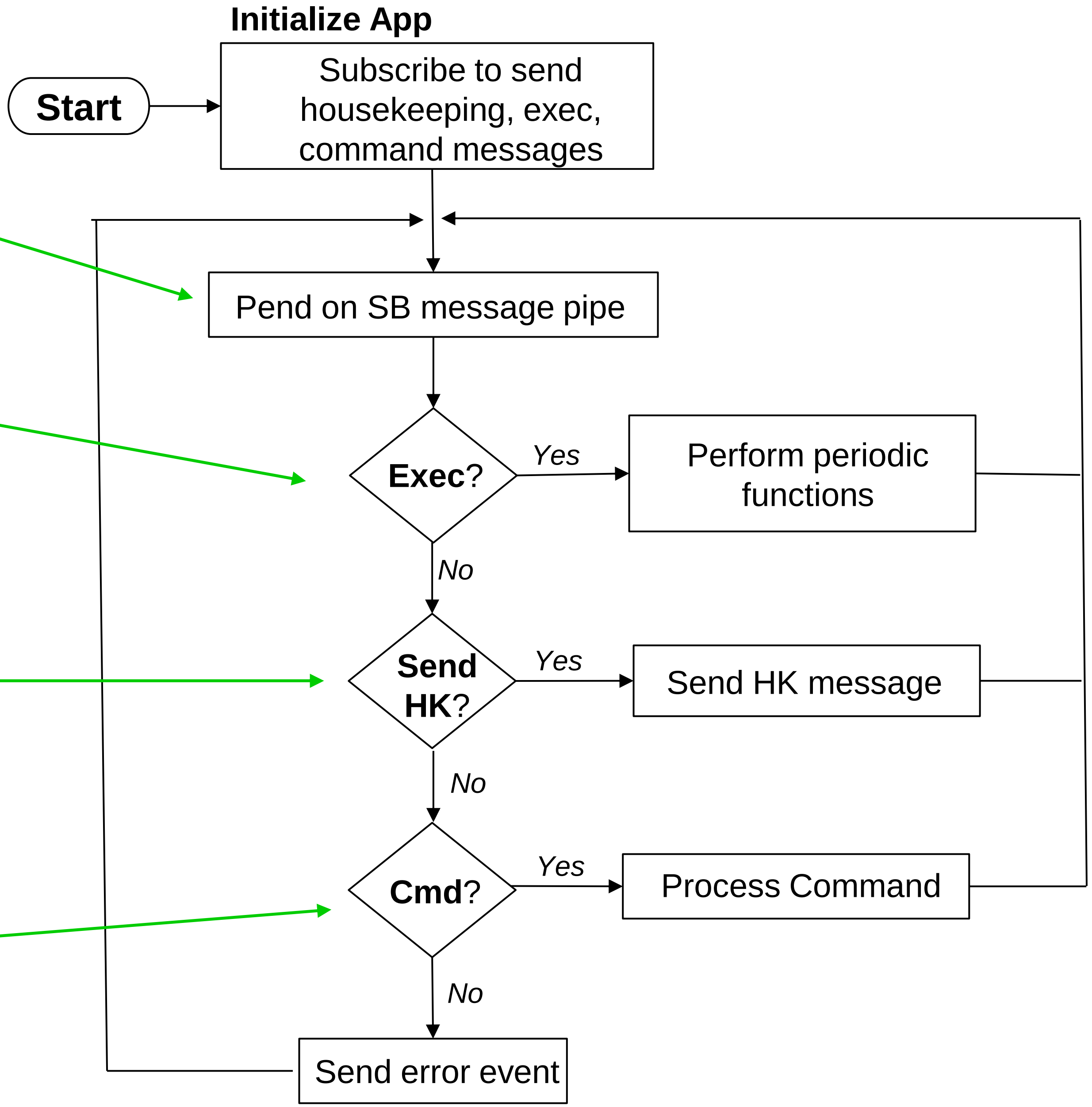
Periodic *execute* message from SCH app

Periodic *request housekeeping* message from SCH app

- Typically, on the order of seconds
- "Housekeeping cycle" convenient time to perform non-critical app functions

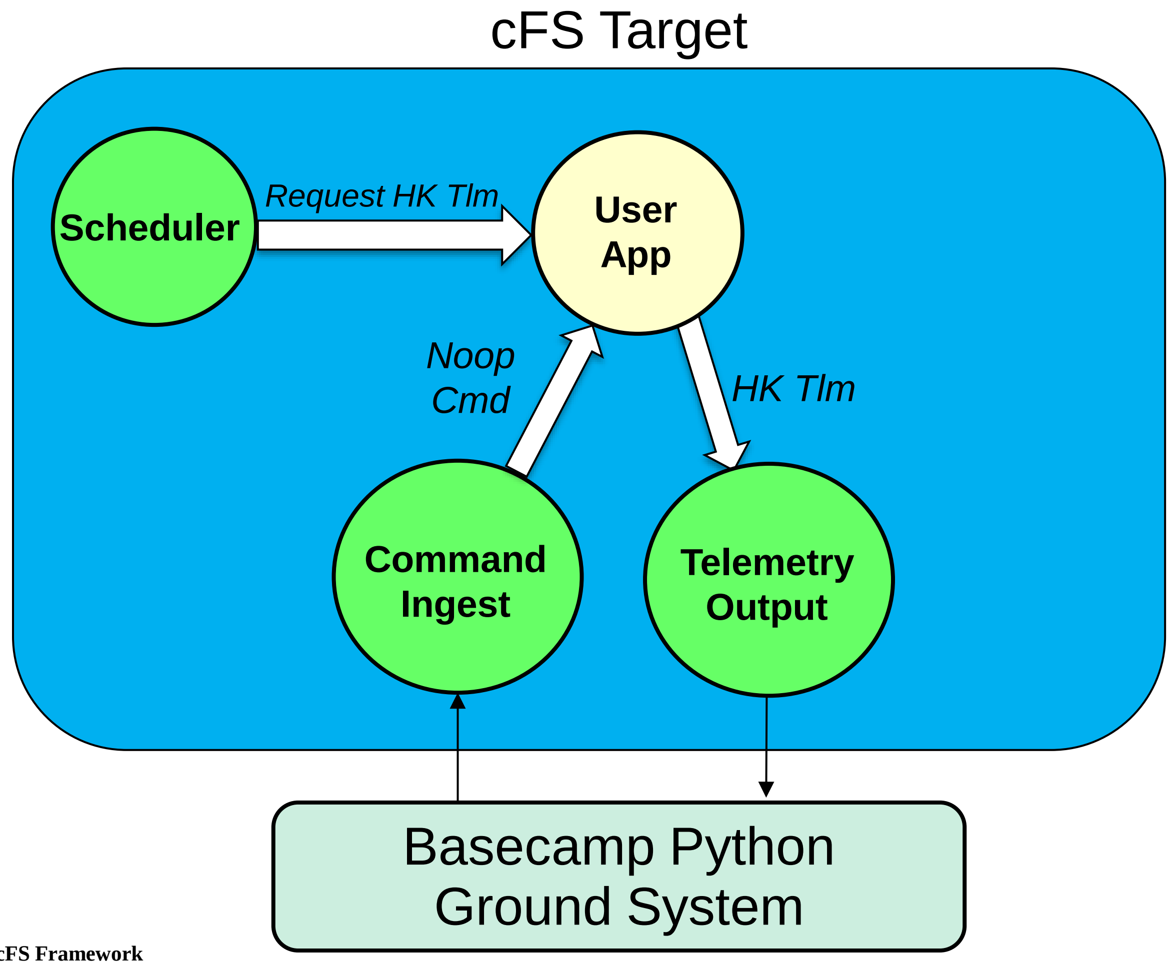
Process commands

- Commands can originate from ground or other onboard apps



Application Runtime Environment Summary

- **By convention every app contains a "No Operation (NOOP)" command**
- **Walking through the NOOP command execution flow is a good way to understand the runtime environment provided by three of operations service apps**

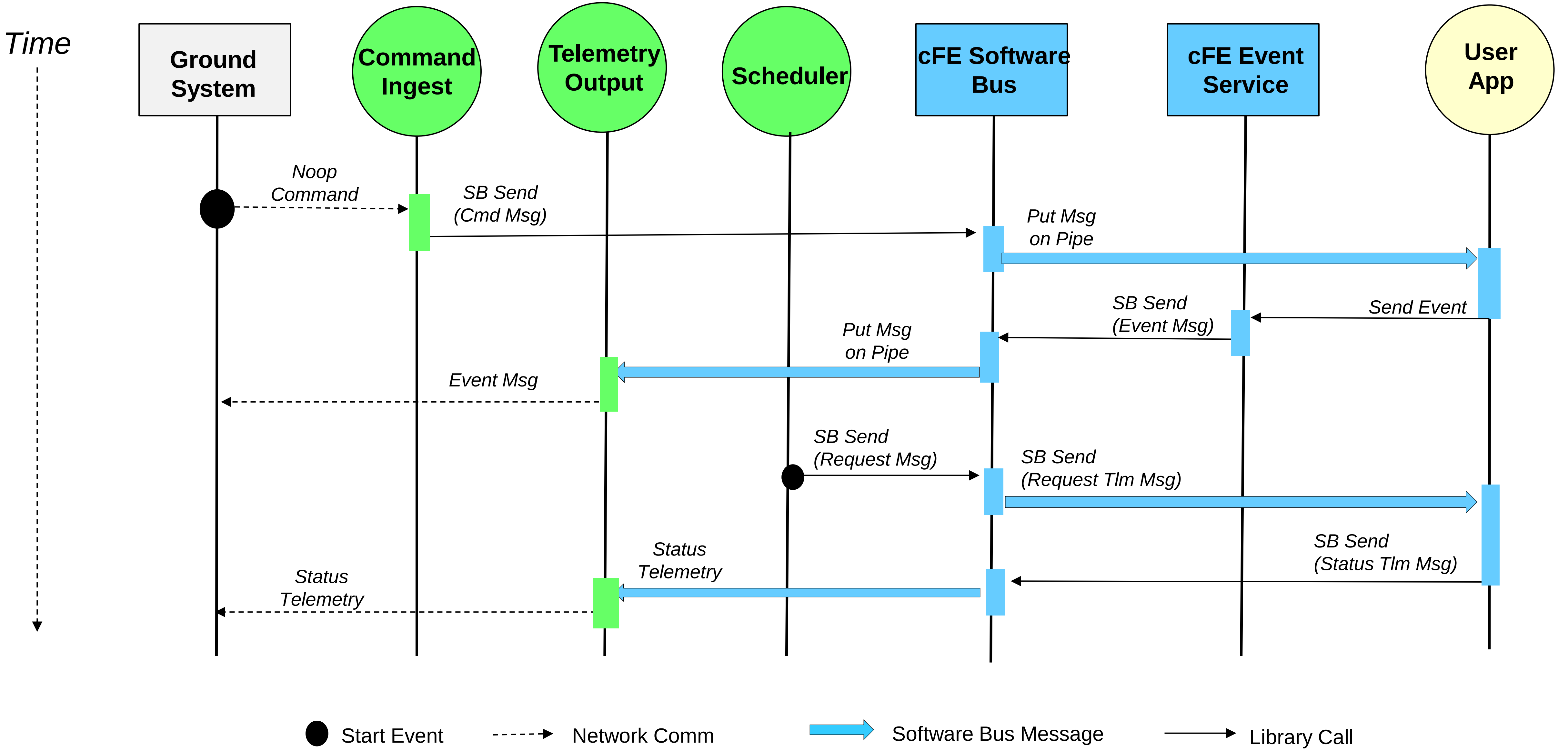


This sequence is illustrated on the next slide

1. When a user sends a NOOP command from the ground system an app responds with
   - An event message that contains the app's version number
   - Increments the command valid counter

2. The Scheduler app periodically sends a "Request Housekeeping Telemetry"
   - HK telemetry includes valid and invalid command counters

No Operation (Noop) Command Sequence



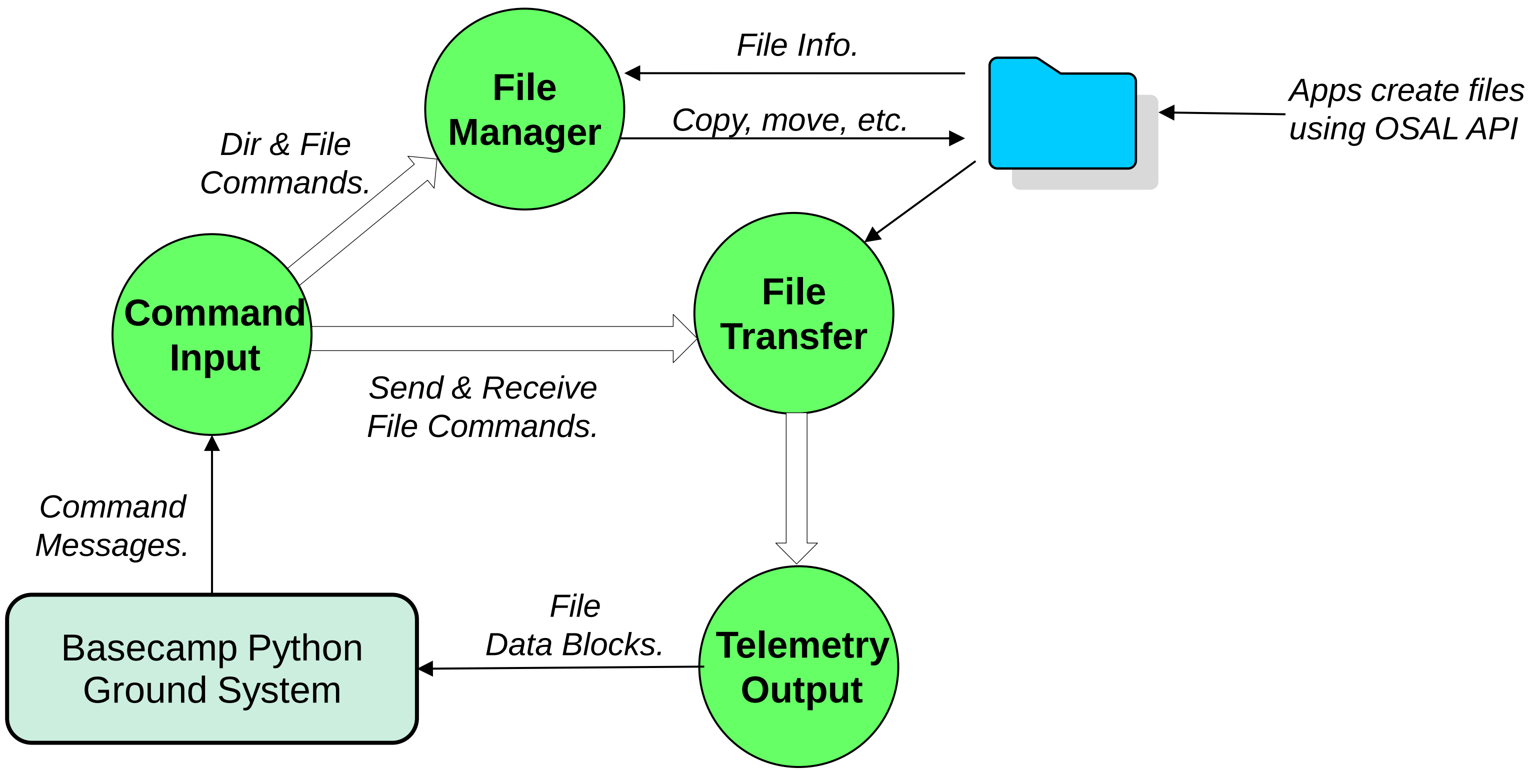
File Management Operations Service Apps

- **The cFS relies on a file system so Basecamp's Operations Service App Suite contains two apps that provide the required functionality**

- **File Manager provides a ground command/telemetry interface for managing onboard directories and files**
  – The NASA File Manager app design was refactored to use an object-based design and Basecamp’s application framework

- **File Transfer transfers files between flight and ground using a custom file transfer protocol implemented in both the flight and ground systems**
  – The protocol is very similar to the Class 1 CCSDS File Delivery Protocol (CFDP)
  – The protocol messages are defined using EDS

File Management Ops Scenario Example



- **Apps create files using the OSAL API so the cFS Framework can manage files resource usage**
- **File Manager is used to manage directories and files**
  - Commands can originate from the ground or other onboard apps
- **File Transfer is used to transfer between the ground and flight**
  - Files are divided into data blocks that are transferred as CSSDS messages
  - The File Transfer to Telemetry Output message interface requires a mechanism to control the data rate

Basecamp Tables

- **Basecamp apps do <u>not</u> use the cFS table service**

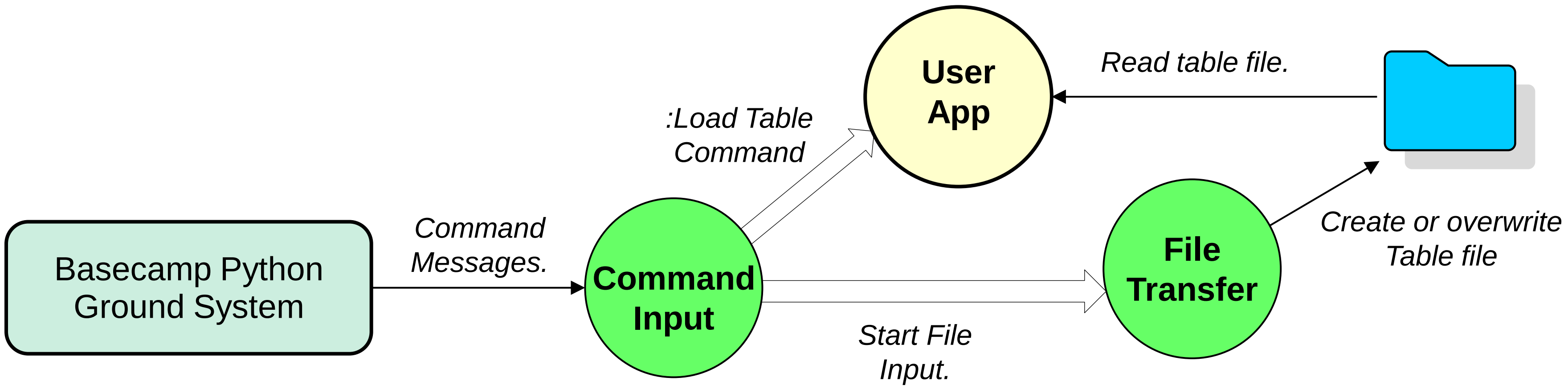
- **JSON files are used to define table parameters and values**
  - The cFS uses binary files
  - Onboard file management apps need to be present to manage table files

- **If a Basecamp app has a table then it provides table load and dump commands**
  - The Basecamp app framework provides table management and JSON parsing services
  - Developers must provide code for loading/dumping table data

- **All Basecamp apps have a JSON initialization file, but it is <u>not</u> a table**

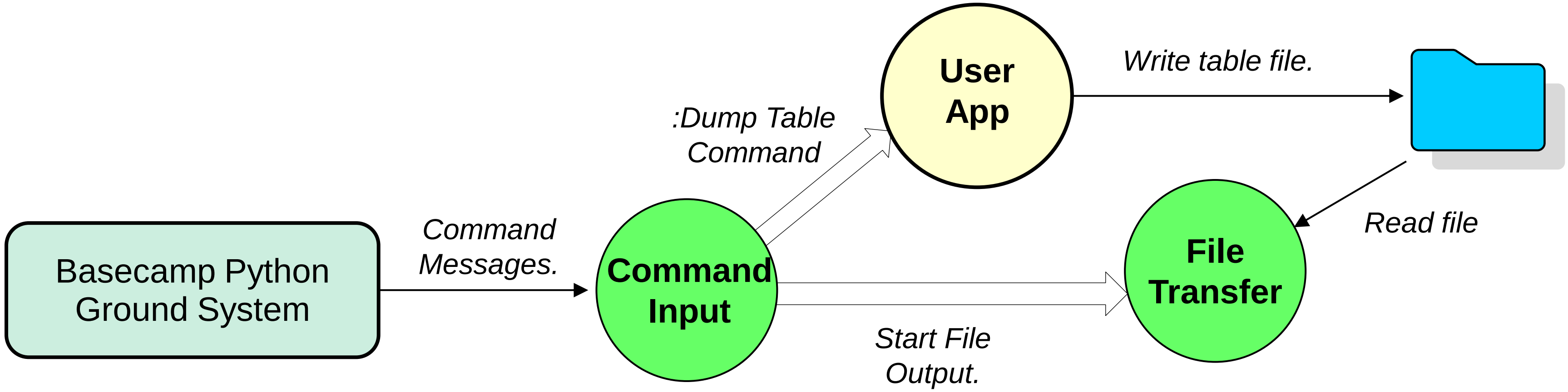
- **The "Hello Table" code tutorial and Basecamp App Developer Guide describe how to create apps with tables**

Load Table



1. **Use File Transfer to transfer the table file from the ground to an onboard directory**
   - The file can be located in any directory
2. **Send a Table Load command to the user app**
   - The command specifies the table filename and directory
   - The user app parses the JSON file using basecamp app framework utilities and optionally validates the contents

Dump Table



1. **Send a Table Dump command to the user app**
   – The command specifies the directory and table filename
2. **Use File Transfer to transfer the table file from an onboard directory to ground**

cFS Message Definitions

- **As previously described, messages are a key component of the cFS architecture**
  - They are used for transferring data and can be used to control an application’s execution
- **The next few slides describe how Electronic Data Sheets are used to define messages**
- **The Deployment Model section describes how the EDS toolchain is used to build a target**

cFS Message Definitions (1 of 2)

- **Messages**
  - Data structures used to transfer data between applications

- **Consultative Committee for Space Data Systems (CCSDS) packet standard used for message format**
  - Simplifies data management since CCSDS standards can used for flight-ground interfaces and onboard messages
  - In theory other formats could be used but has not occurred in practice

- **CCSDS Primary Header (Always big endian)**

<table>
<tr><th colspan="7">PACKET PRIMARY HEADER</th></tr>
<tr><td rowspan="2">PACKET VERSION NUMBER</td><td colspan="3">PACKET IDENTIFICATION</td><td colspan="2">PACKET SEQUENCE CONTROL</td><td rowspan="3">PACKET DATA LENGTH</td></tr>
<tr><td>PACKET TYPE</td><td>SEC. HDR. FLAG</td><td>APPLICATION PROCESS IDENTIFIER</td><td>SEQUENCE FLAGS</td><td>PACKET SEQUENCE COUNT OR PACKET NAME</td></tr>
<tr><td>3 bits</td><td>1 bit</td><td>1 bit</td><td>11 bits</td><td>2 bits</td><td>14 bits</td></tr>
<tr><td colspan="4">2 octets</td><td colspan="2">2 octets</td><td>2 octets</td></tr>
</table>

cFS Message Definitions (2 of 2)

- **“Packet” often used instead of “message” but not quite synonymous**
  - “Message ID” (first 16-bits) used to uniquely identify a message
  - “App ID” (11-bit) CCSDS packet identifier
- **Extended App ID naming domain**
  - cFE 6.6 supports CCSDS’s 2023 Space Packet Protocol updates for complex mission topologies including multiple spacecraft, multiple processors, etc.
- **CCSDS Command Packets**
  - Secondary packet header contains a command function code
  - cFS apps typically define a single command packet and use the function code to dispatch a command processing function
  - Commands can originate from the ground or from onboard applications
- **CCSDS Telemetry Packets**
  - Secondary packet header contains a time stamp of when the data was produced
  - Telemetry is sent on the software bus by apps and can be ingested by other apps, stored onboard and sent to the ground

Electronic Data Sheet Standard

- **An Electronic Data Sheet (EDS) is a formal specification of a device, system, or software interface in a machine readable format**

- **A CCSDS Spacecraft Onboard Interface Services (SOIS) EDS (SEDS) is an EDS defined using the SOIS Dictionary of Terms and the SOIS EDS XML schema**

- **The cFS Framework and EDS toolchain provide definitions that allow users to define command and telemetry messages**
  - *base_types.xml* – Standardize common data types
  - *ccsds_spacepacket.xml* – CCSDS packet headers and fields

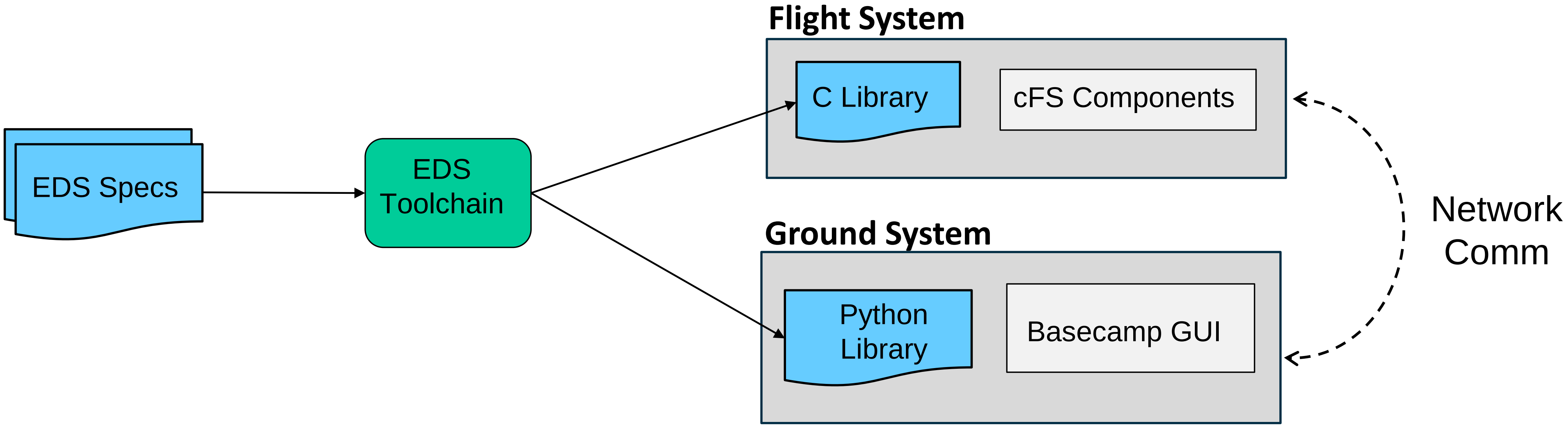
EDS Message Identifiers

- **A *“Topic ID”* uniquely identifies a message within a mission name space**

- **The EDS toolchain maps *Topic IDs* to *Message IDs* that are used on each processor running a cFS target**

- ***cfe-topicids.xml* defines a mission’s Topic IDs**

Single EDS Definition

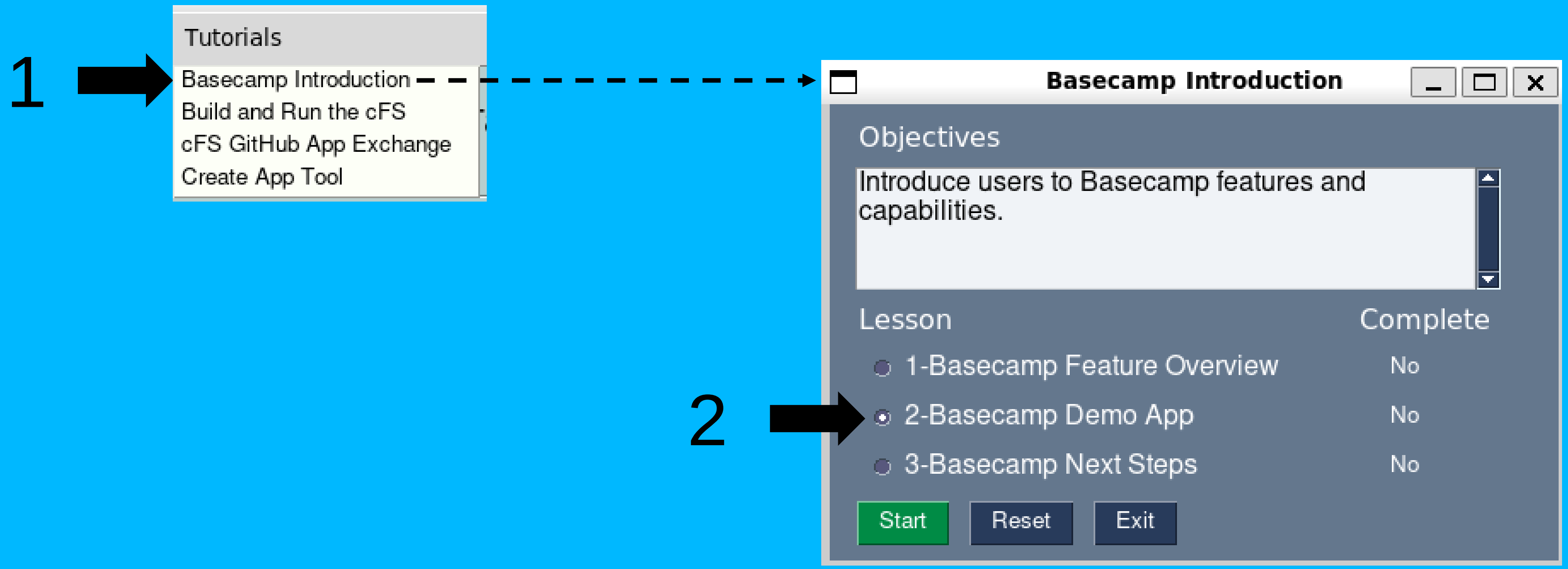


- **Each command and telemetry message is defined once using EDS**

- **The EDS is a network “on the wire” definition and the libraries created by the EDS toolchain manage local processor byte ordering (endianness) conversions**

Demo App Operations

1. From the Tutorial dropdown list select “Basecamp Introduction” and do Lesson 2 “Basecamp Demo App”
    - Interacting with the Demo App illustrates how the Operational Service Apps create a unified system



cFS Framework Deployment

Framework Deployment Agenda

1. cFS Framework Modules

2. cFS Framework Configuration Parameters

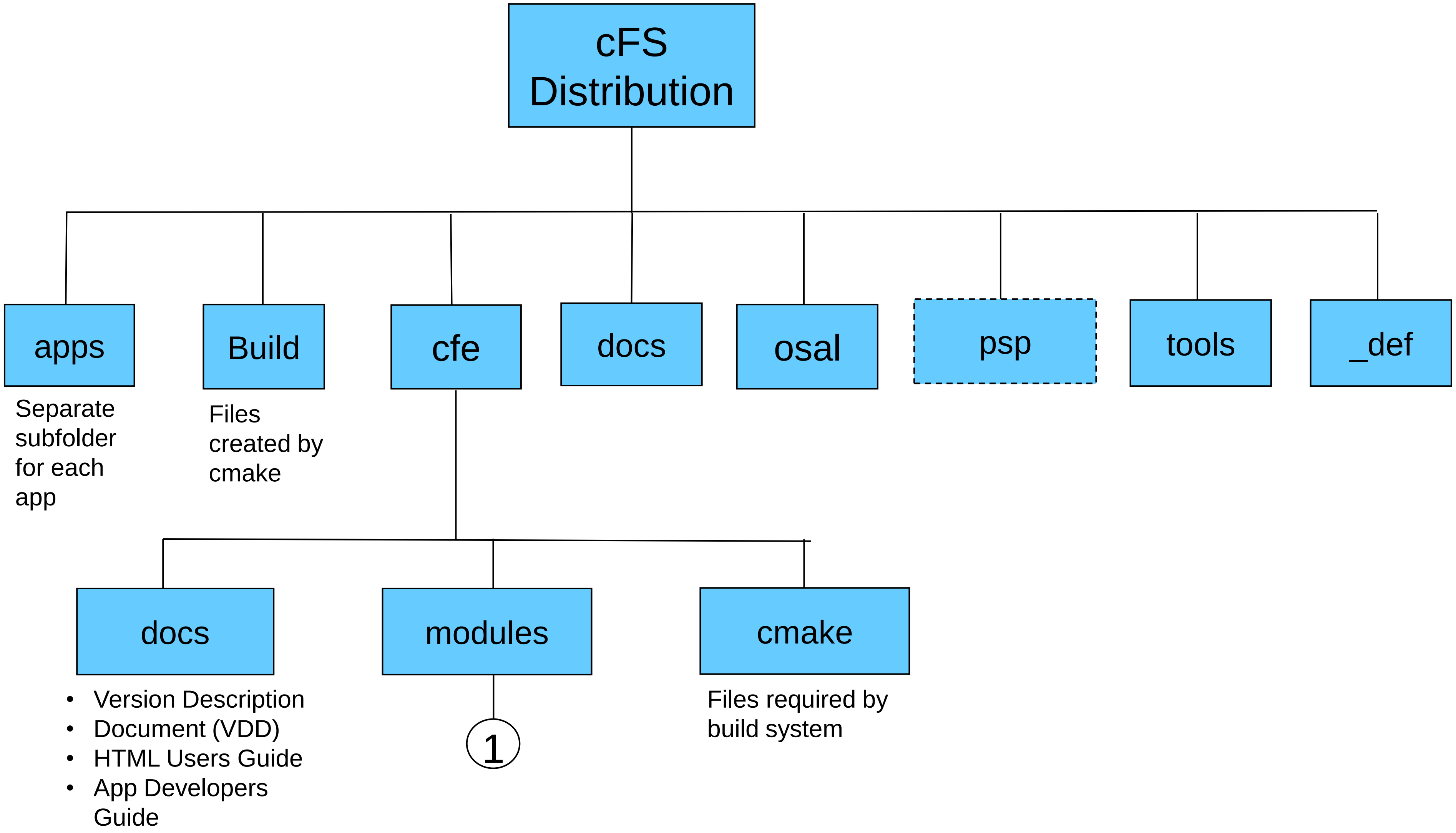
3. Electronic Data Sheet Toolchain

4. Complete Basecamp’s ‘Build and Run cFS’ tutorials

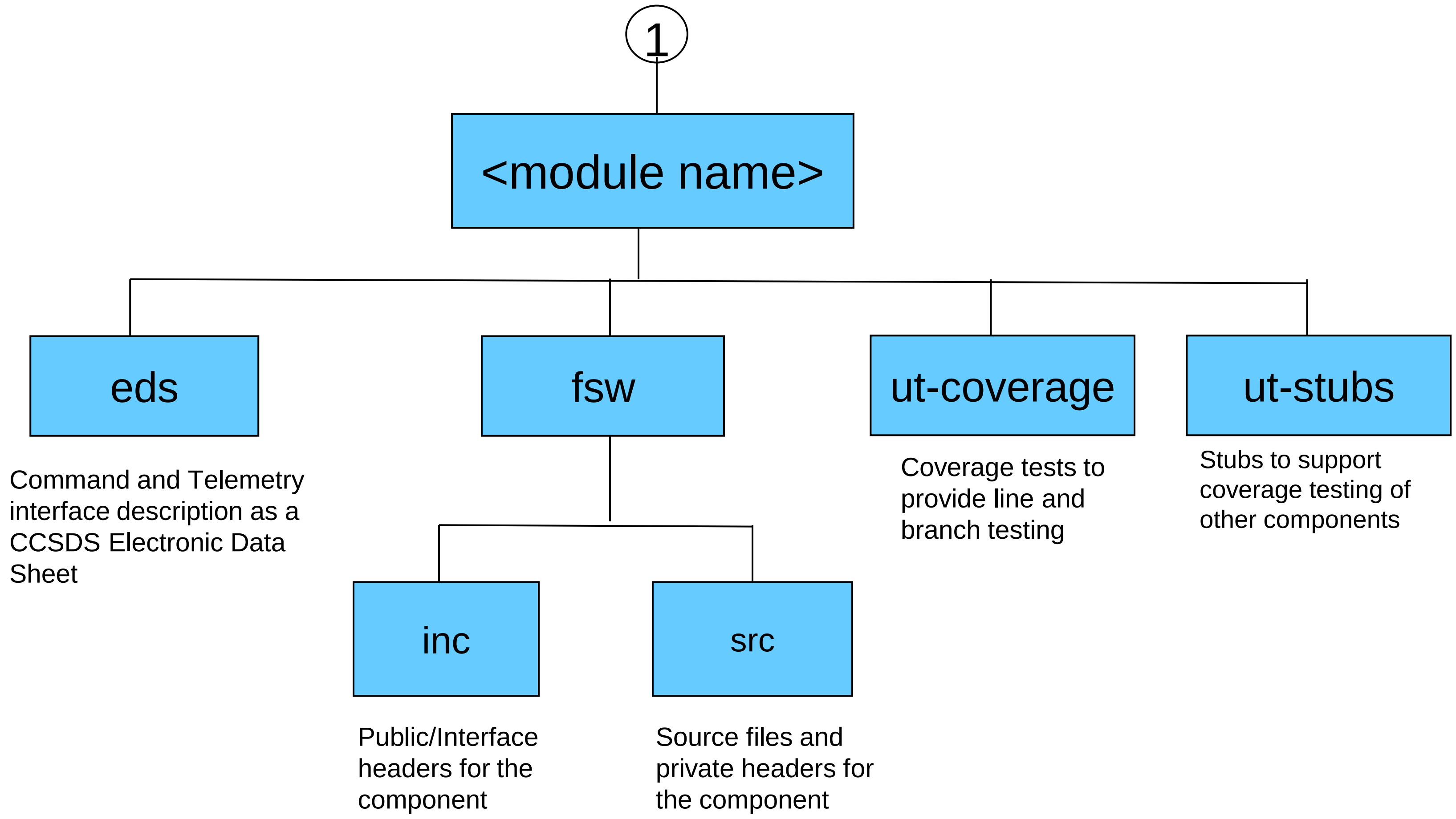
Module Overview

- The cFS framework is compiled and linked as a single binary
- cFS Caelum released in 2021 decomposes the cFS framework into Modules
- The Module structure allows advanced users to add, remove, or override entire core services as necessary to support their particular mission requirements
- cFE “out of the box” provides reference implementations that meet the needs of most missions

cFS Mission Directory Structure



Module Directory Structure



Module Summary

Module	Purpose/Content
cfe_assert	A CFE-compatible library wrapping the basic UT assert library.
cfe_testcase	A CFE-compatible library implementing test cases for CFE core apps.
core_api	Contains the public interface definition of the complete CFE core - public API/headers only, no implementation.
core_private	Contains the inter-module interface definition of the CFE core - internal API/headers only, no implementation.
es	Implementation of the Executive Services (ES) core module.
evs	Implementation of the Event Services (EVS) core module.
fs	Implementation of the File Services (FS) core module.
msg	Implementation of the Message (MSG) core module.
resourceid	Implementation of the Resource ID core module.
sb	Implementation of the Software Bus (SB) core module.
sbr	Implementation of the Software Bus (SB) Routing module.
tbl	Implementation of the Table Services (TBL) core module.
time	Implementation of the Time Services (TIME) core module.

Configuration Parameter Scope

- **Mission configuration parameters – used for ALL processors in a mission (e.g. 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 (e.g. 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

TODO: EDS config.xml

Configuration Parameters Notes

- **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**

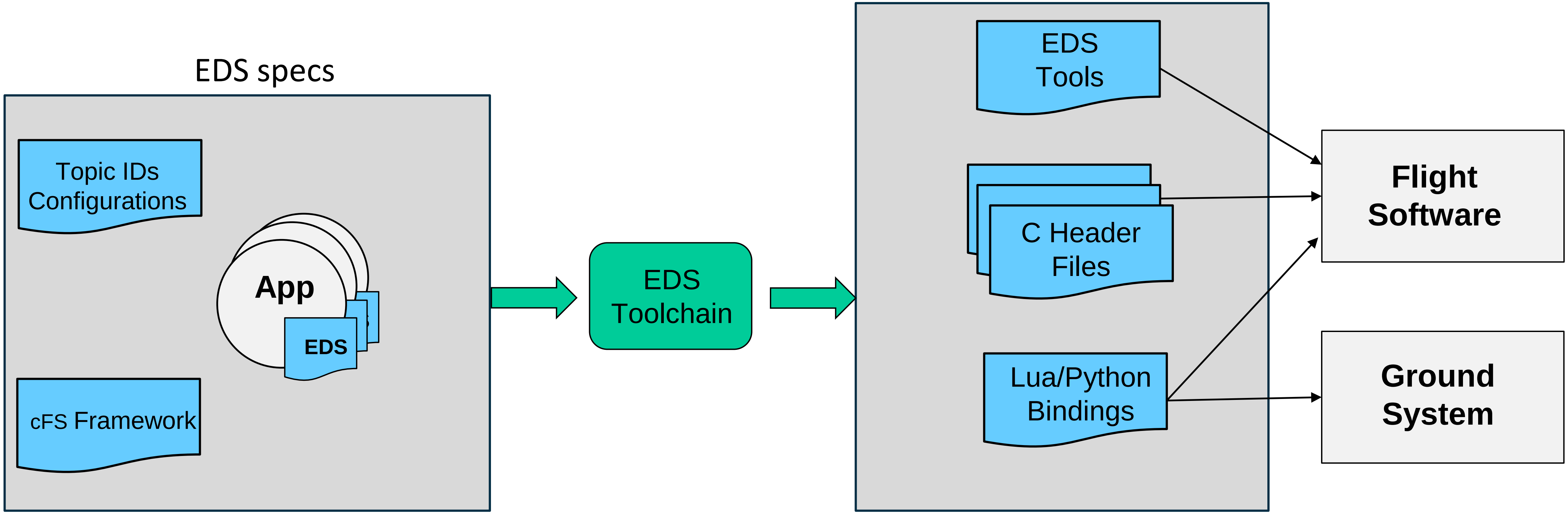
Configuration File Summary

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	

TODO: EDS config.xml

Electronic Data Sheet Toolchain

The EDS toolchain processes EDS files producing ground and flight artifacts



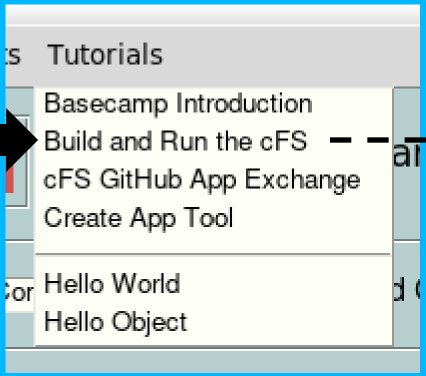
Basecamp Topic ID Tool

- **Basecamp augments the cFS toolchain with a new “make target” called “topicids”**
  - Invoked at the command line with *“make topicids”*
  - It runs *cfe-eds-framework/basecamp_defs/cfe_topicids.py* prior to running the cFS *“make install”* target

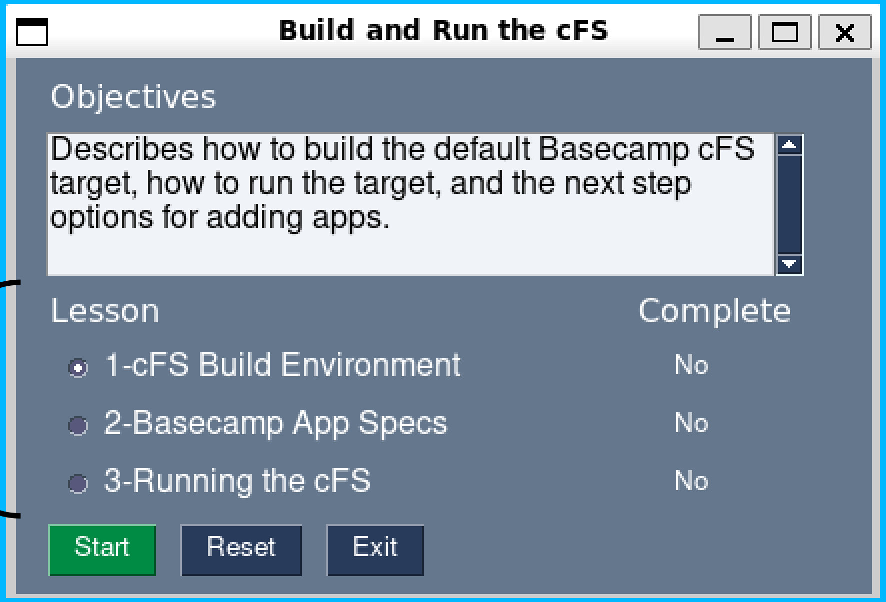
- **Functions performed by *cfe_topicids.py***
  1. Read mission Topic IDs defined in *cfe-topicids.xml*
  2. Update Topic ID values in library and application initialization parameter JSON files
  3. Update Topic ID values in application table parameter JSON files

1. From the Tutorial dropdown list select “Build and Run the cFS” and complete all lessons

1



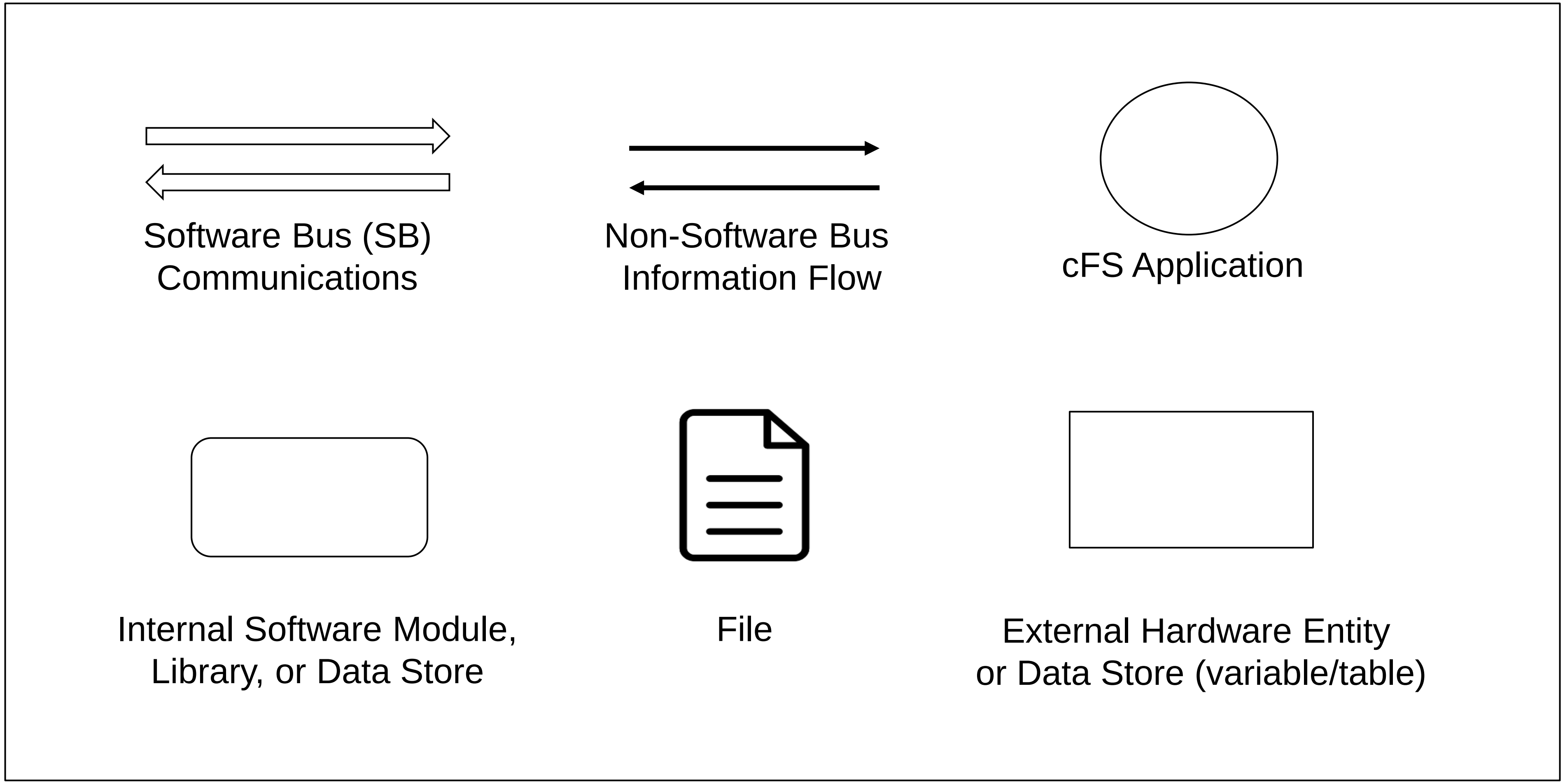
2



Appendix A

Architecture Design Notation

Architecture Design Notation



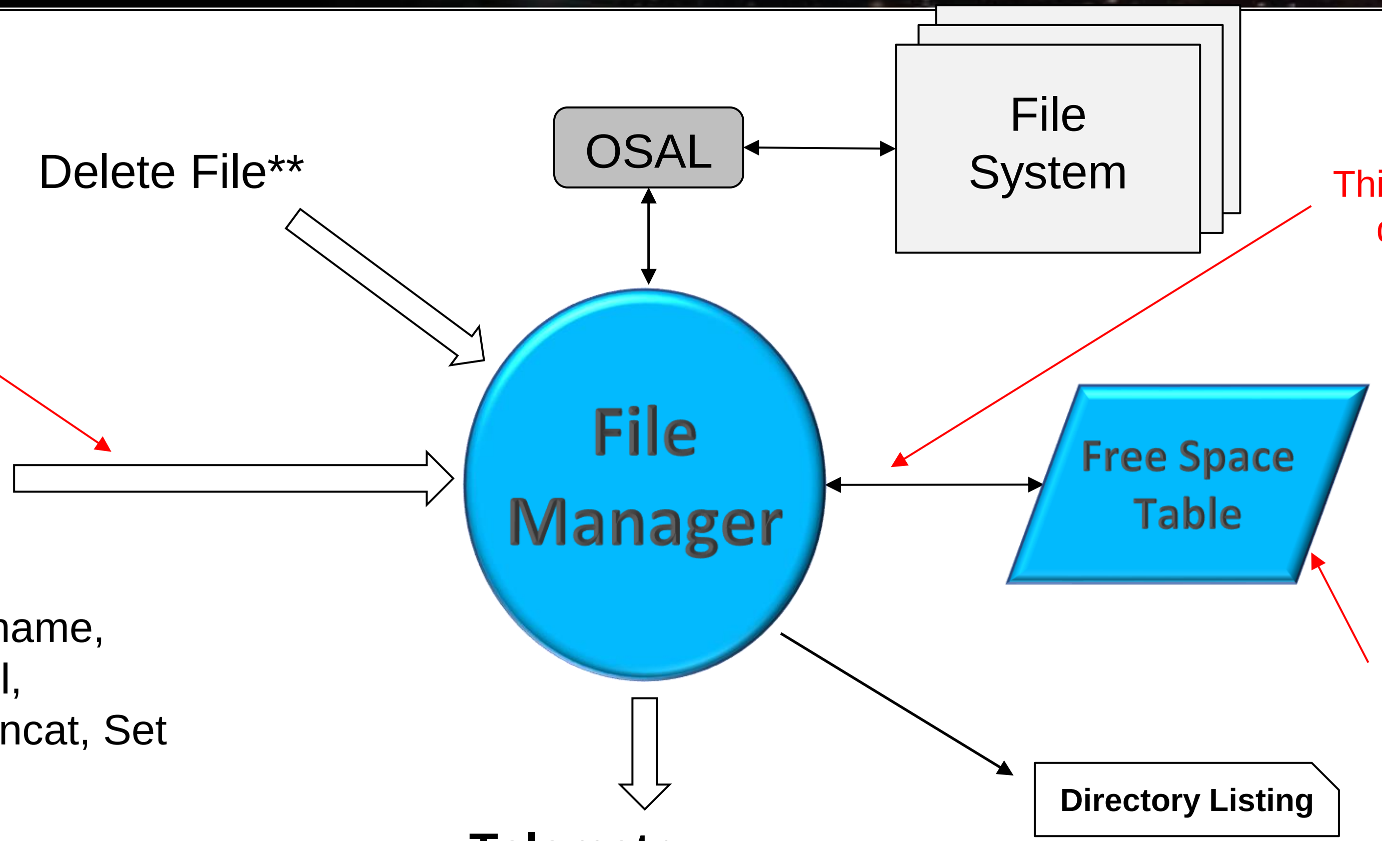
- Common data flows such as command inputs to an app and telemetry outputs from an app are often omitted from context diagrams unless they are important to the situation

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

Context Diagram Example 1



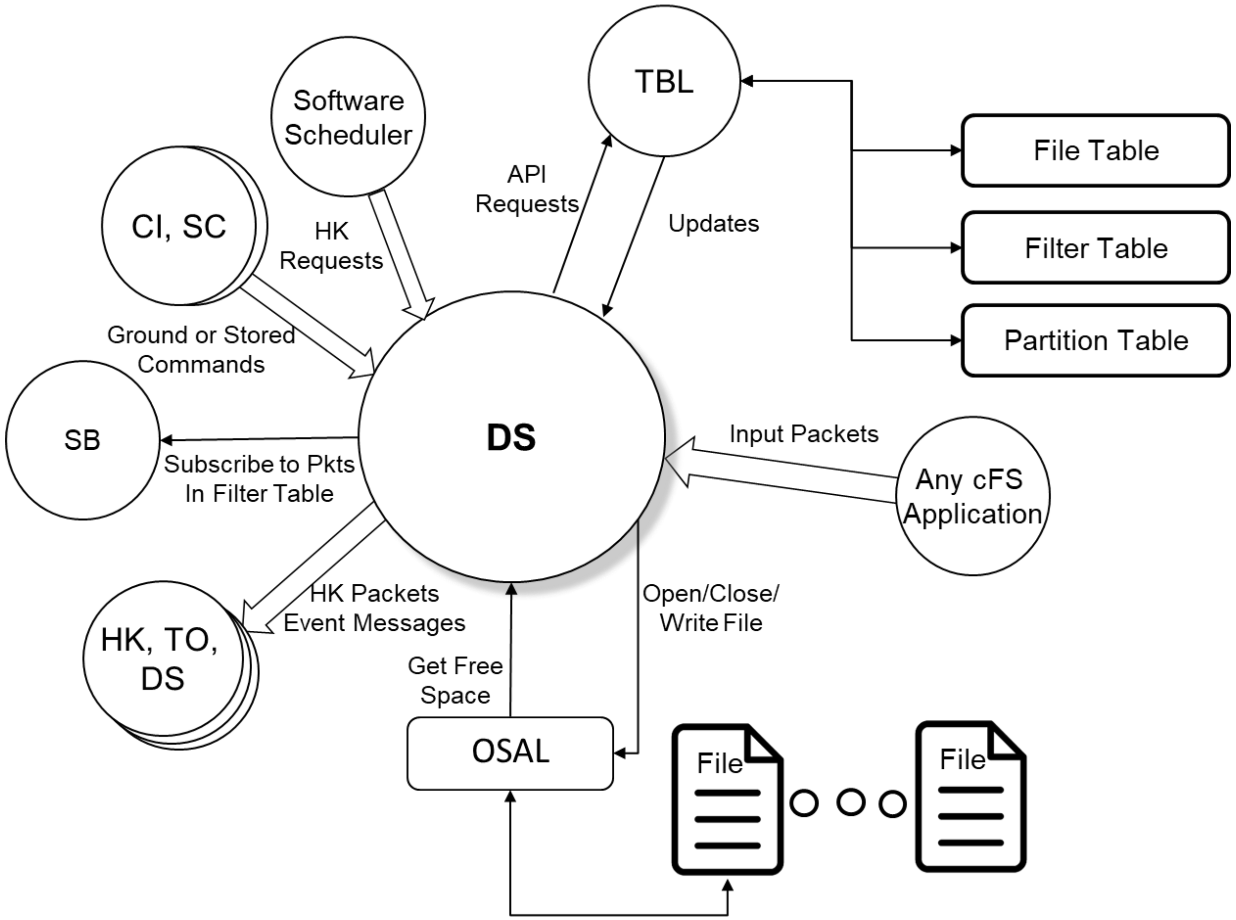
Commands
- Directories
  - Create, Delete
- Files
  - Copy, Move, Rename, Delete, Delete All, Decompress, Concat, Set Permissions
- Status
  - Send File Info Tlm Pkt
  - Send Open File Tlm Pkt
  - Write Dir to File
  - Send Dir Tlm Pkt
  - Send Free Space Tlm Pkt

Telemetry
- Housekeeping
- File Info
- Open Files
- Directory Listing
- File Sys Free Space

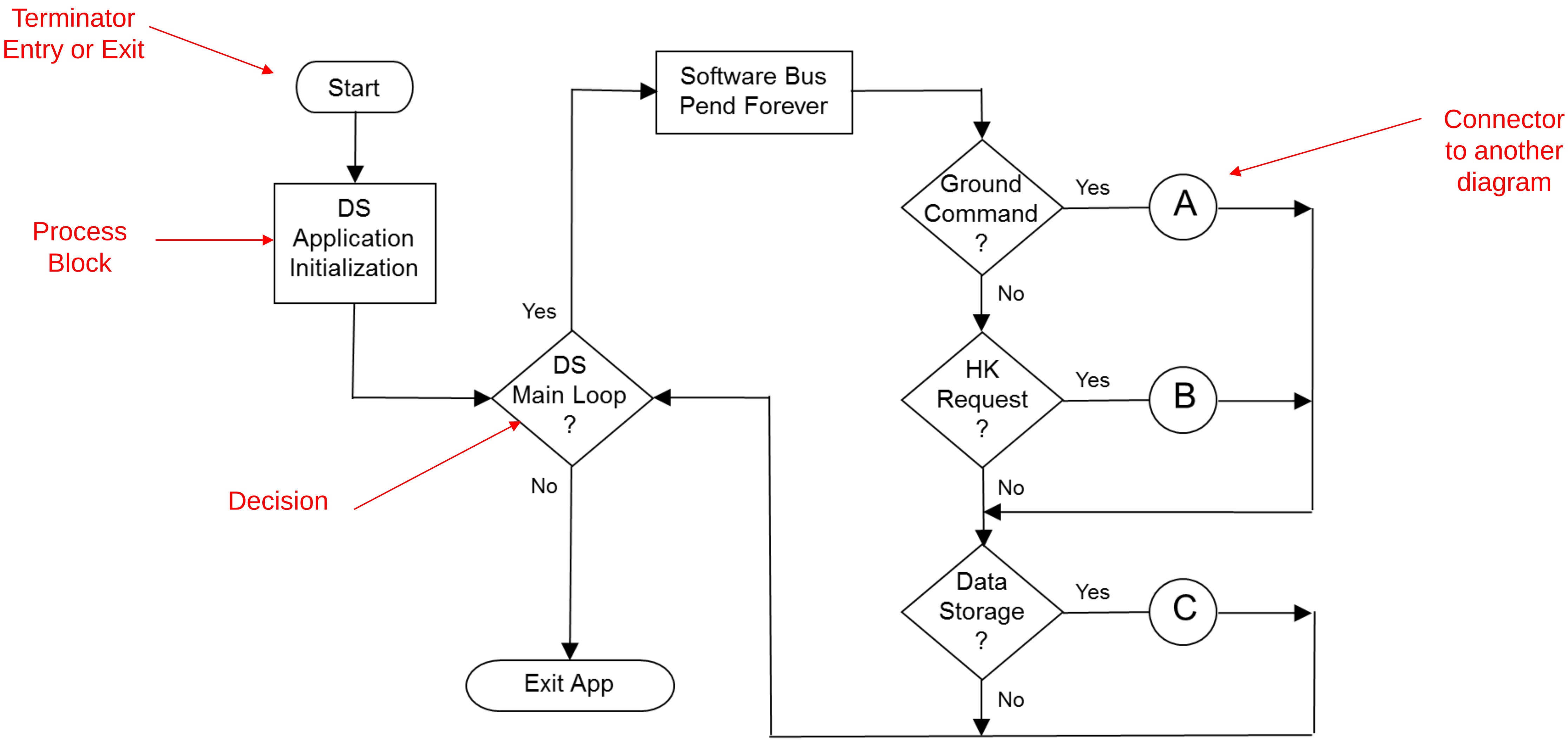
** Onboard command that doesn’t affect ground command counters

Context Diagram Example 2

This is a more complete context diagram which is technically correct, but it can obscure the application-specific information that is most important. For example, HK request from the scheduler and outputting HK packet & event messages on the software bus are common design practice that may be omitted if people are comfortable with some assumptions. The important part of the diagram is showing interface boundaries to understand where control and data flow. Too much information is harder to maintain.



Flowchart Example



App-Level Sequence Diagram Example

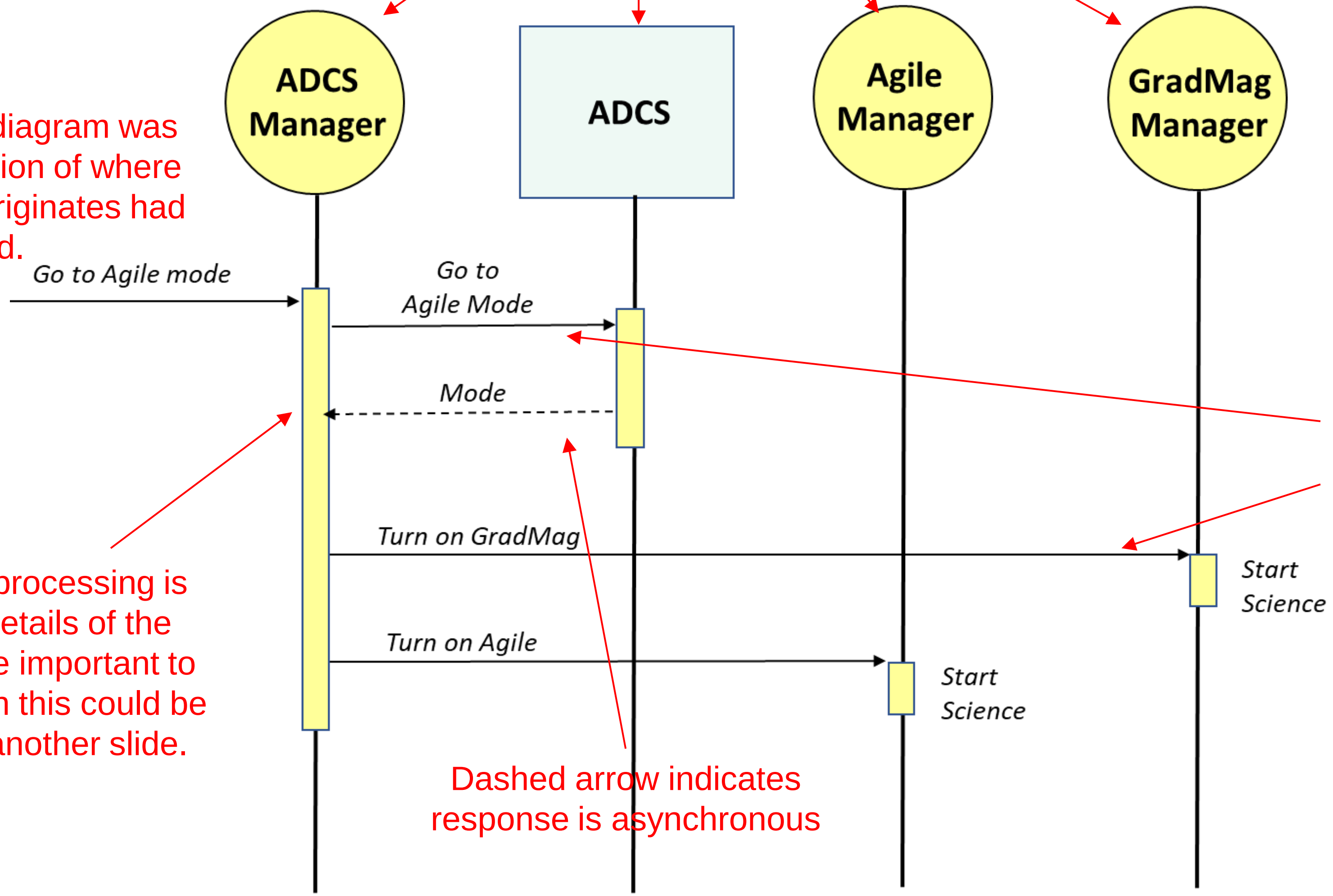
Often not show but if audience is unfamiliar with sequence diagram it can be helpful

Time

Applications, libraries and hardware devices shown across the top

Level of detail shown is based on the goal of what is trying to be communicated. This diagram is showing a mode transition sequence, so it is not important to show that an ADCS request may go through a library or device driver to command the ADCS device.

At the time this diagram was written the decision of where this command originates had not been decided.



The thin line represents communication and how the communication occurs depends upon the item at the top of the subsystem line. App-to-app communication occurs via software bus messages. An app-to-device communication typically occurs through a device driver.

Blocks show processing is required. If details of the processing are important to document, then this could be contained in another slide.

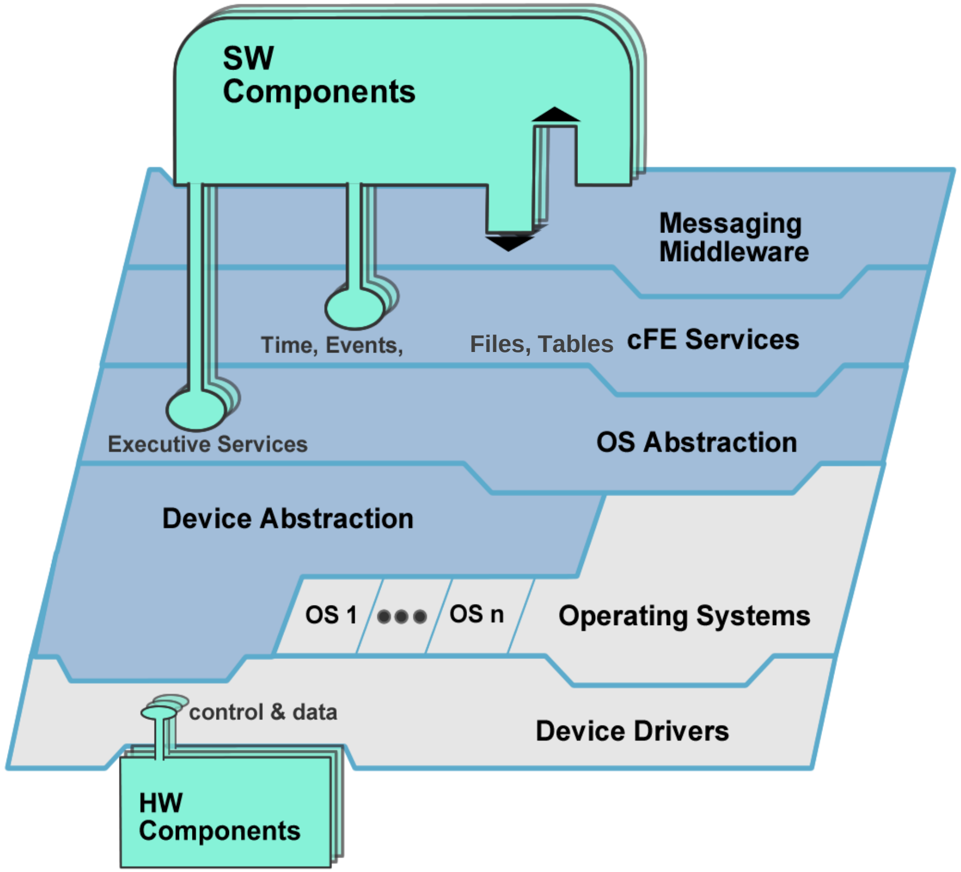
Dashed arrow indicates response is asynchronous

Appendix B

Supplemental Architectural Material

Layered Service Architecture

- Each layer and service has a standard API
- Each layer "hides" its implementation and technology details.
- Internals of a layer can be changed -- without affecting other layers' internals and components.
- Provides Middleware, OS and HW platform-independence.

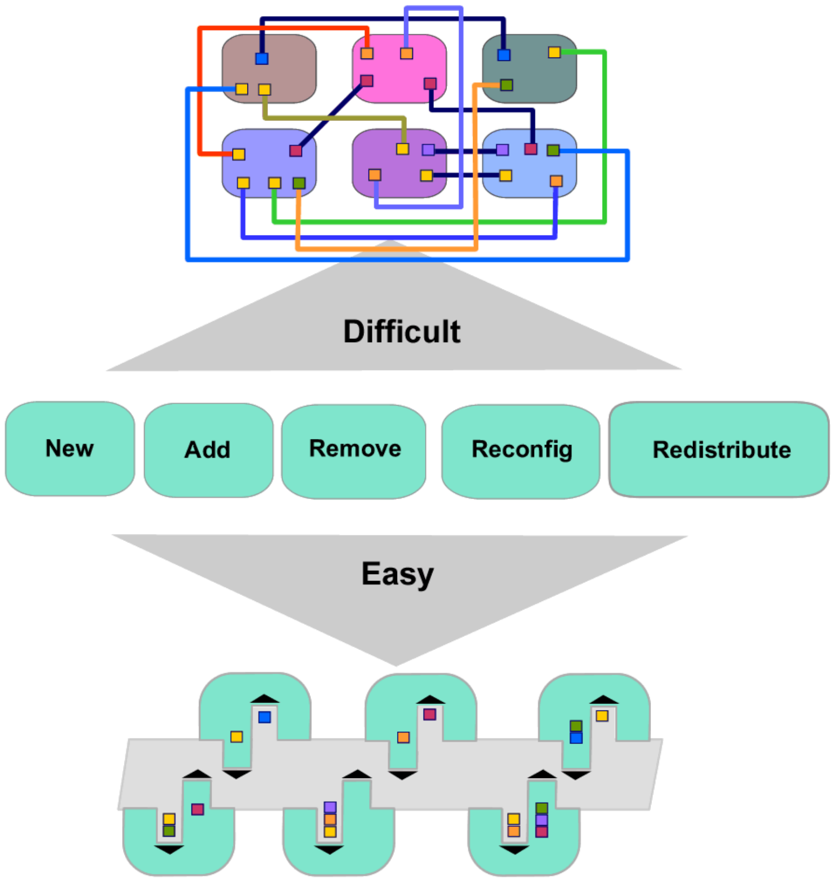


Plug and Play

- cFE API’s support add and remove functions
- SW components can be switched in and out at runtime, without rebooting or rebuilding the system SW.
- Qualified Hardware and cFS-compatible software both “plug and play.”

Impact:

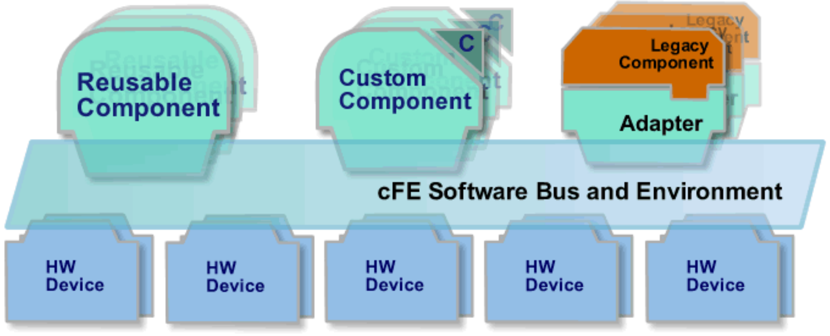
- Changes can be made dynamically during development, test and on-orbit even as part of contingency management
- Technology evolution/change can be taken advantage of later in the development cycle.
- Testing flexibility (test apps, simulators)



This powerful paradigm allows SW components to be switched in and out at runtime, without rebooting or rebuilding the system SW.

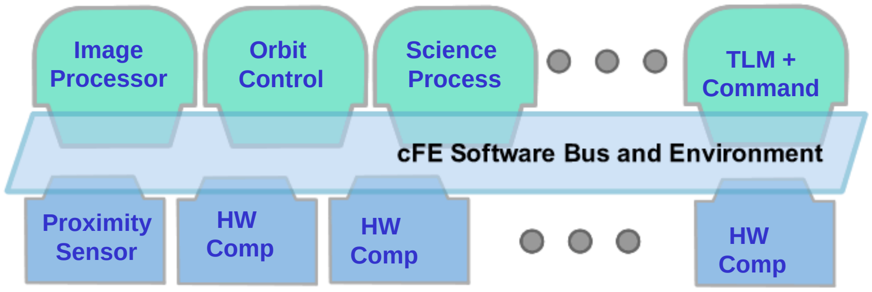
Reusable Components

- **Common FSW functionality has been abstracted into a library of reusable components and services.**
- **Tested, Certified, Documented**
- **A system is built from:**
  - Core services
  - Reusable components
  - Custom mission specific components
  - Adapted legacy components



Impact:

- **Reuse of tested, certified components supplies savings in each phase of the software development cycle**
- **Reduces risk**
- **Teams focus on the custom aspects of their project and don't "reinvent the wheel."**



cFS Metrics

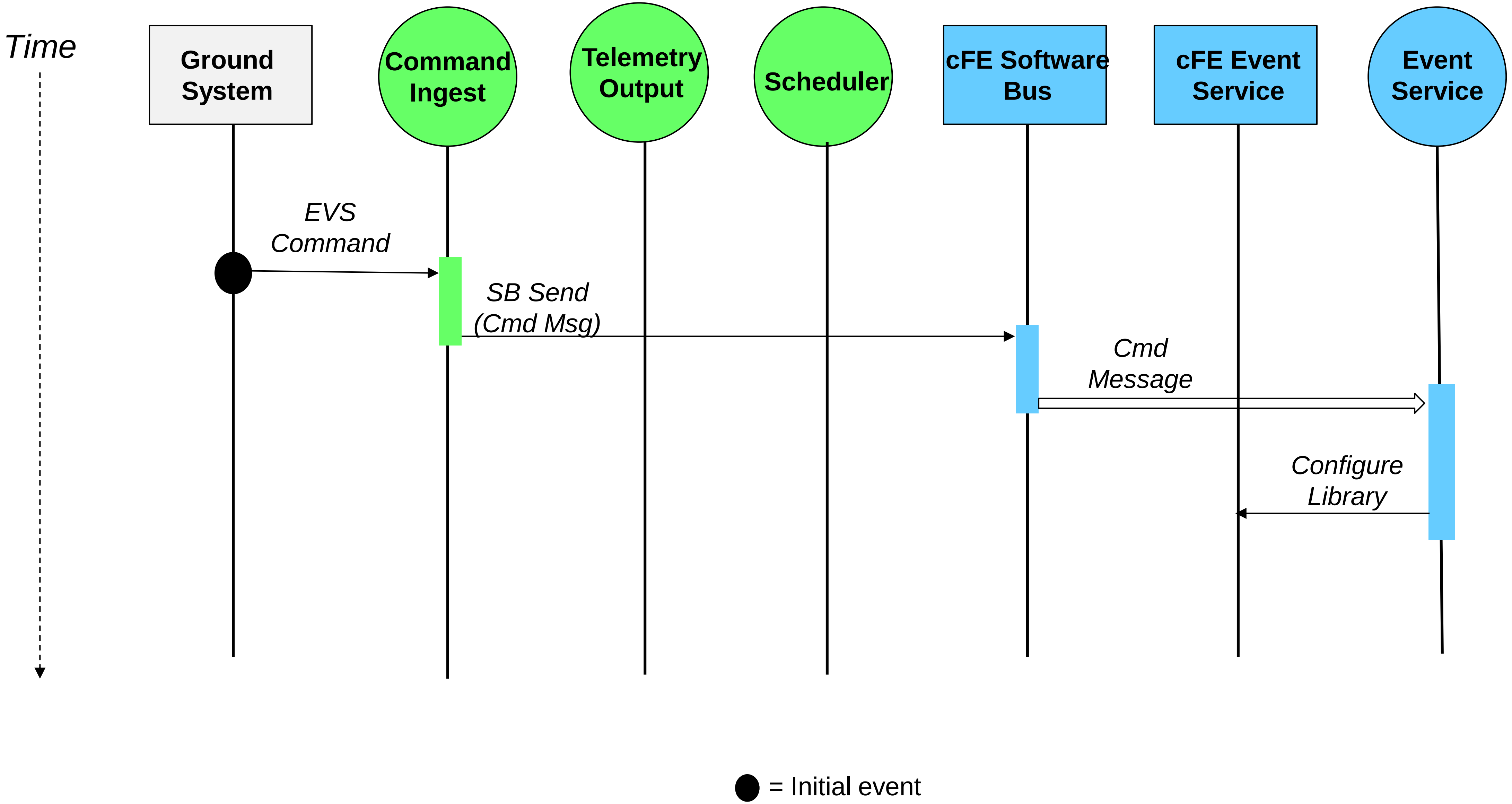
cFE/ App	Logical Lines of Code (non-table)	Config. Parameters	EEPROM (bytes)
cFE 6.4.0	12,930	General: 17 Executive Service: 46 Event Service: 5 Software Bus: 29 Table Service: 10 Time Service: 32	341,561
CFDP	8,559	33	85,812
Checksum	2,873	15	35,242
Data Storage	2,429	27	40,523
File Manager	1,853	22	16,272
Health & Safety	1,531	45	15071
House-Keeping	575	8	8.059
Limit Checker	2,074	13	31,026
Memory Dwell	1,035	8	8,617
Memory Manager	1,958	25	15,840
Scheduler	1,164	19	35,809
Stored Command (124 command sequences)	2,314	26	104,960

Example Mission Code Metrics
Goddard Class B Mission

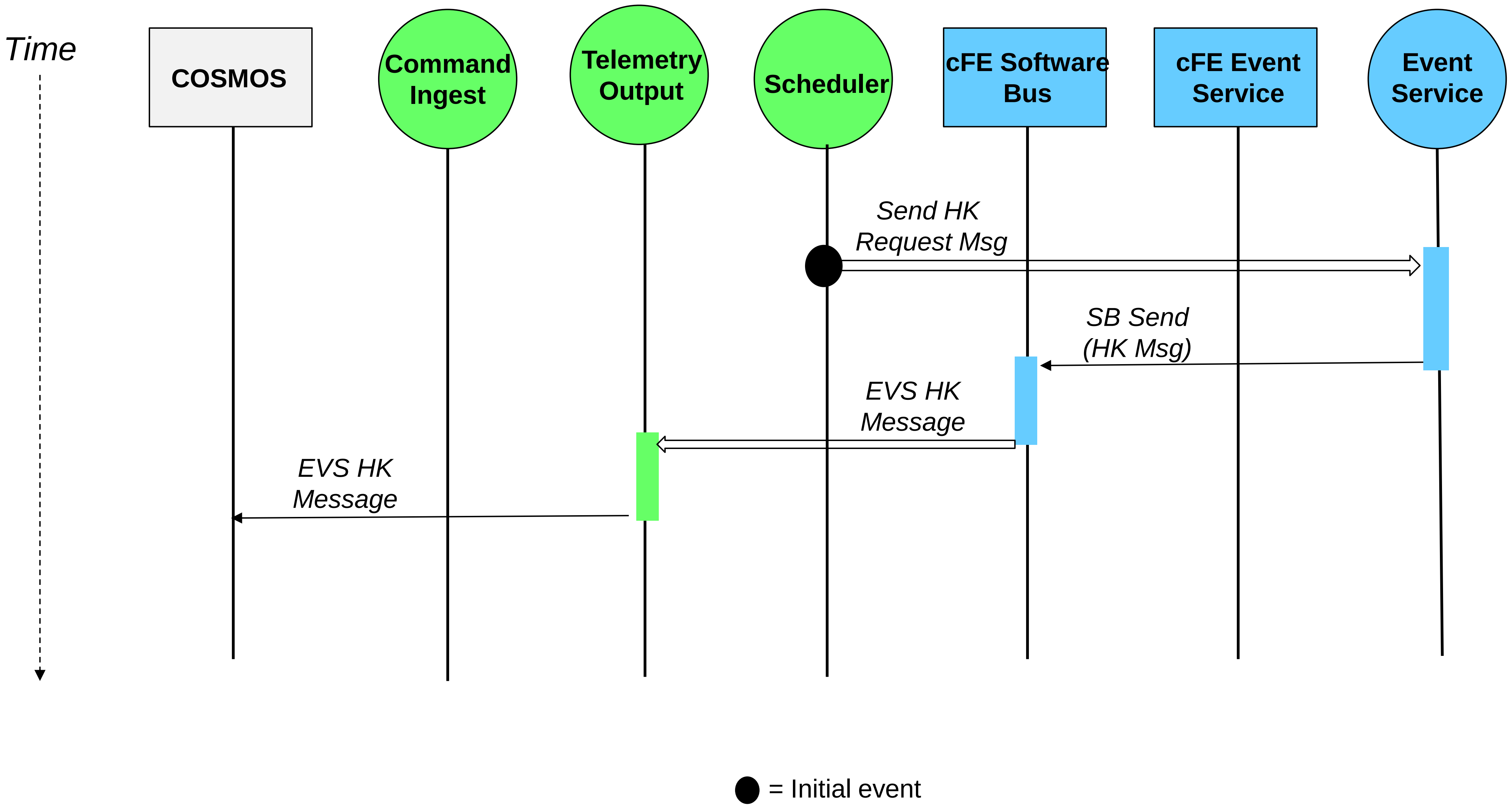
- **Noteworthy items**
  - + cFE was very reliable and stable
  - + Easy rapid prototyping with heritage code that was cFE compliant
  - + Layered architecture has allowed COTS lab to be maintained through all builds
  - – Addition of PSP changed build infrastructure midstream
- **Lines of Code Percentages:**

Source	Percentage
BAE	0.3
EEFS	1.7
OSAL	2.1
PSP	1.0
cFE	12.4
GNC Library	1.6
CFS Applications	23.5
Heritage Clone & Own	38.9
New Source	18.5

Ops Sends EVS Configuration Command



Event Service App Sends Housekeeping Telemetry



References

- **cFE EDS Framework Toolchain, https://github.com/jphickey/cfe-eds-framework**
- **TODO**
  - CCSDS
  - Free RTOS JSON parser
  - cFS
  - Space Steps