Executive Services

Basecamp

August 2025

Training Module Introduction

- **Objectives**
  - Provide a comprehensive description of the cFS Framework Executive Service
- **Intended audience**
  - Software engineers developing with the cFS
- **Prerequisites**
  - cFS Framework Introduction

Blue screens contain hands on cFS Basecamp exercises

- Basecamp is a lightweight environment with built-in tutorials for learning the cFS
- https://github.com/cfs-tools/cfs-basecamp

cFE Service Training Module Topics

The following topics are covered in each cFE service training module. They are listed in this module since Executive Services is typically covered first

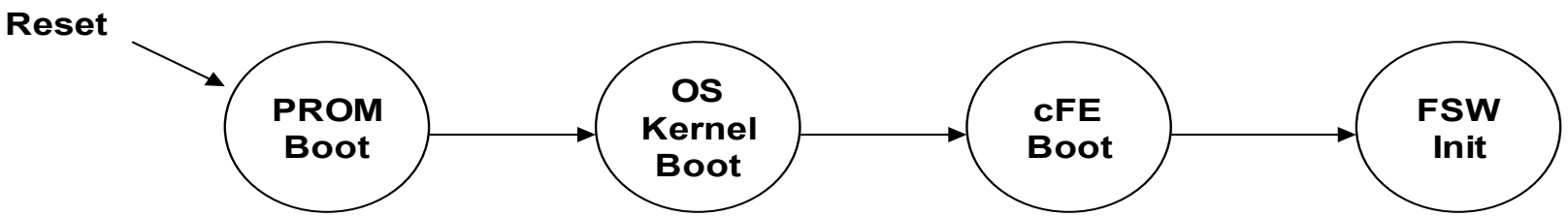
- **Describe Service**
  - What functions are available and how are they used
- **System Initialization**
  - What does the service do during system initialization
  - What can applications do during system initialization
- **Processor Reset Behavior**
  - Identifies and reports startup/reset type
  - Maintains an exception-reset log across processor resets
- **Retrieving Onboard State**
  - What mechanisms are available for operators to get information about the service
- **System Design**
  - Highlight aspects of the service that are important for creating a system of libraries and applications
- **Application Development**
  - Highlight aspects of the service that are important for application developers
- **Function API Summary**
  - Functions that can be used by applications
- **Software Bus Command Summary**
  - List commands that are typically sent from the ground but can also be sent from other onboard apps
- **Configuration Parameter Summary**
  - List mission and platform configurations

Executive Services Overview

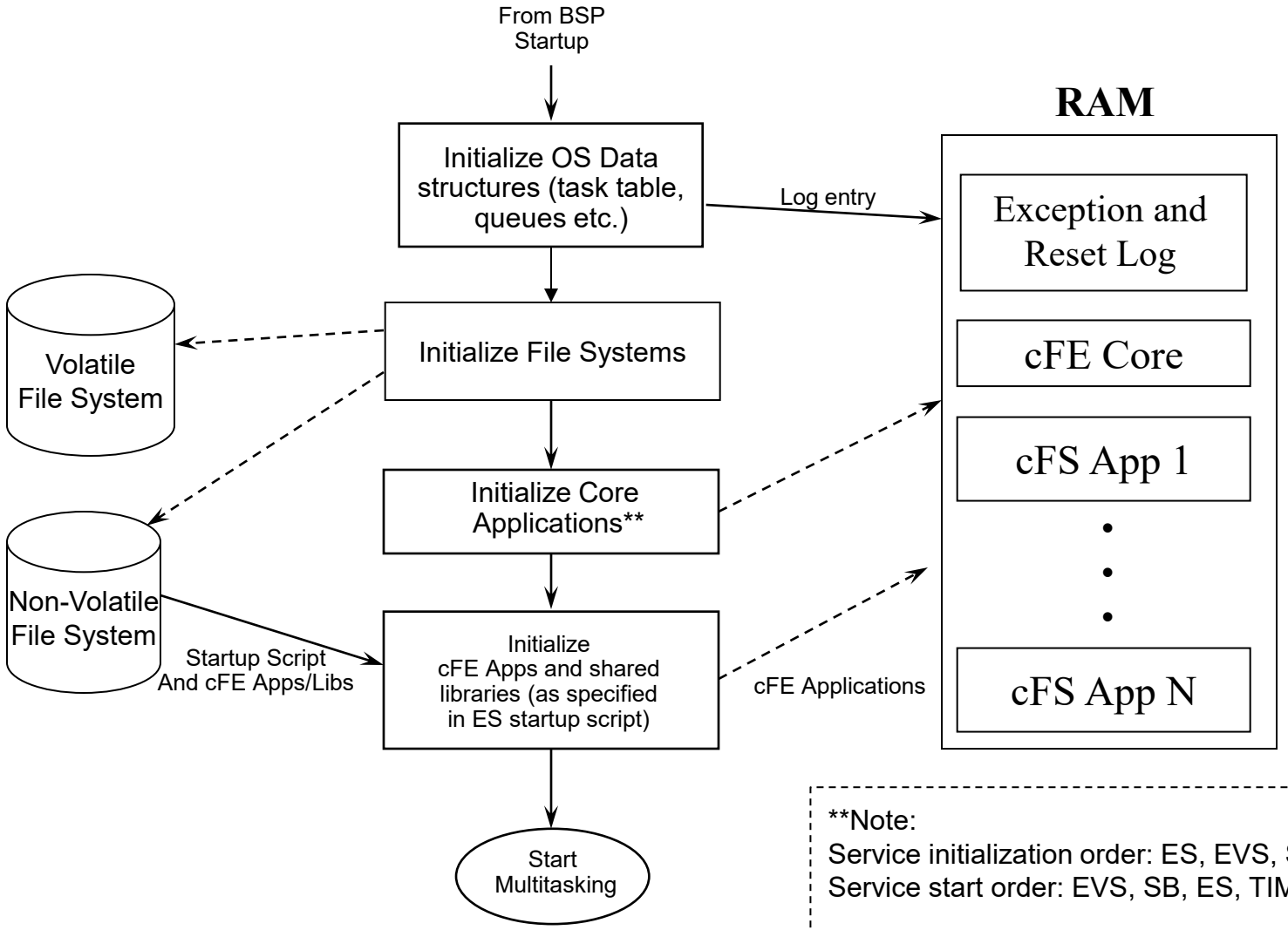
- **Initialize the cFE**
  - Identifies and reports startup/reset type
  - Maintains an exception-reset log across processor resets
- **Creates the application runtime environment**
  - Primary interface to underlying operating system task services & resources
  - Traditionally Real Time Operating Systems (RTOS) have been used for spacecraft, however, many technology missions often in the CubeSat class don't require an RTOS and can use an Operating System (OS) like Linux, therefore the terms RTOS and OS and are used synonymously in this slide deck
  - 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 Services Startup Sequence (1 of 2)

- **Using an image stored in non-volatile memory, the processor boots the Operating System (OS) kernel linked with the Board Support Package (BSP), loader and the non-volatile file system**
  - Accesses simple non-volatile file system
  - Selects primary and secondary images (mission-specific) based on flags and checksum validation
  - Copies operating system image to RAM
- **Operating system startup**
  - Performs self-decompression (optional)
  - Attaches to non-volatile file system
  - Starts the cFE
- **cFE startup sequence**
  - Creates/Attaches to Critical Data Store (CDS)
  - Creates/Attaches to RAM file system
  - Starts cFE service applications (ES, EVS, TBL, SB, & TIME)
  - Loads and starts libraries and applications defined in the ES startup script *cfe_es_startup.scr*



Executive Services Startup Sequence (2 of 2)



The cFE core is started as one unit. The cFE Core is linked with the RTOS and support libraries and loaded into non-volatile memory as a static executable.

Executive Services Startup Script (1 of 2)

- **The startup script *cfe_es_startup.scr* is a text file, written by a developer that contains a list of libraries and applications to be loaded during system startup**
  - Used by the ES application for automating the startup of applications
  - Executive Services supports nonvolatile and volatile memory-based startup scripts
  - A project may provide zero, one or two startup scripts
  - One entry for each library or application

Object Type	`CFE_APP` for an Application, or `CFE_LIB` for a library.
Path/Filename	This is a cFE Virtual filename, not a vxWorks device/pathname
Entry Point	This is the name of the "main" function for App.
CFE Name	The cFE name for the APP or Library
Priority	This is the Priority of the App, not used for a Library
Stack Size	This is the Stack size for the App, not used for a Library
Load Address	This is the Optional Load Address for the App or Library. It is currently not implemented so it should always be 0x0.
Exception Action	This is the Action the cFE should take if the Application has an exception. • 0 = Do a cFE Processor Reset • Non-Zero = Just restart the Application

Executive Services Startup Script (2 of 2)

```
CFE_LIB, cfe_assert,  CFE_Assert_LibInit, ASSERT_LIB,    0,   0,     0x0, 0;
CFE_LIB, sample_lib,  SAMPLE_LIB_Init,    SAMPLE_LIB,    0,   0,     0x0, 0;
CFE_APP, sample_app,  SAMPLE_APP_Main,    SAMPLE_APP,   50,   16384, 0x0, 0;
CFE_APP, ci_lab,      CI_Lab_AppMain,     CI_LAB_APP,   60,   16384, 0x0, 0;
CFE_APP, sch_lab,     SCH_Lab_AppMain,     SCH_LAB_APP,   70,   16384, 0x0, 0;
!
! Startup script fields:
! 1. Object Type      -- CFE_APP for an Application, or CFE_LIB for a library.
! 2. Path/Filename    -- This is a cFE Virtual filename, not a vxWorks device/pathname
! 3. Entry Point      -- This is the "main" function for Apps.
! 4. CFE Name         -- The cFE name for the the APP or Library
! 5. Priority         -- This is the Priority of the App, not used for Library
! 6. Stack Size       -- This is the Stack size for the App, not used for the Library
! 7. Load Address     -- This is the Optional Load Address for the App or Library. Currently not implemented
!                        so keep it at 0x0.
! 8. Exception Action -- This is the Action the cFE should take if the App has an exception.
!                        0        = Just restart the Application
!                        Non-Zero = Do a cFE Processor Reset
!
! Other  Notes:
! 1. The software will not try to parse anything after the first '!' character it sees. That
!    is the End of File marker.
! 2. Common Application file extensions:
!    Linux = .so  ( ci.so )
!    OS X  = .bundle  ( ci.bundle )
!    Cygwin = .dll ( ci.dll )
!    vxWorks = .o ( ci.o )
!    RTEMS with S-record Loader = .s3r ( ci.s3r )
!    RTEMS with CEXP Loader = .o ( ci.o )
! 3. The filename field (2) no longer requires a fully-qualified filename; the path and extension
!    may be omitted.  If omitted, the standard virtual path (/cf) and a platform-specific default
!    extension will be used, which is derived from the build system.
```

Executive Services Startup Script Notes

- **If an application relies on a library, then the library must be loaded first**
  - If the cFS target is build to support dynamic loading (default) you will get an unresolved symbol error during initialization if the library is not loaded first

- **Applications can be compressed in non-volatile memory and decompressed by ES when they are loaded**

  Get the details on how this is supported

Executive Services Logs

- **Exception-Reset**

  Logs information related to processor resets and exceptions

- **System Log**

  – cFE apps use this log when errors are encountered during initialization before the Event Services is fully initialized

  – Mission apps can also use it during initialization

    - Recommended that apps should register with event service immediately after registering with ES so app events are captured in the EVS log

  – Implemented as an array of bytes that has variable length strings produced by printf() type statements

Application Management

- **Applications are architectural components that own 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**
- **ES App Management Commands*: Start, Stop, Restart, Reload***
  - Application run through their initialization sequence
  - Application data is not preserved

cFS Tasking Model (1 of 2)

- **The cFS was originally designed to run in the older single memory address space processors and Real Time Operating Systems (RTOS)**
- **Each cFS app consists of one or more RTOS tasks/threads**
  - For example, a cFS app has a main task/thread and each child task is a peer task/thread in the OS
- **On memory protected operating systems such as Linux, VxWorks RTPs, QNX, etc, the entire cFS instance runs in the context of a single process**
  - Similar to an RTOS, the pthreads within that process are all in the same (virtual) address space.

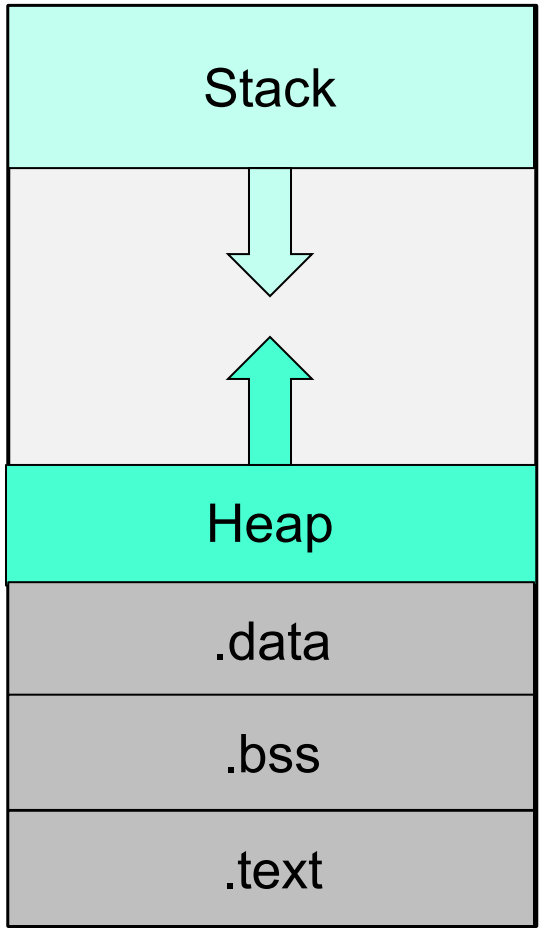
cFS Tasking Model (2 of 2)

- **A disadvantage to this model is that cFS apps can't be separated into separate processes**

- **An advantage to this model is that process based operating systems can support multiple cFS instances**
  - Each cFS instance runs in its own memory protected process
  - If needed, the instances can communicate using the Software Bus Network app

cFS Memory Resources (1 of 3)

Example Processor Memory



Stack

- Local function variables and return addresses
- Grows downward
- Note stack management depends upon the operating system

Heap

- Dynamic memory allocated/deallocated with C functions
- Grows upward

.data

- Initialized global and static variables

.bss

- Uninitialized global and static variables
- “Block Started by Symbol”

.text

- Code segment generated from source code

cFS Memory Resources (2 of 3)

- **In general C's dynamic memory functions are avoided in embedded systems due to memory fragmentation, potential memory leaks, and non-deterministic timing**
- **Organizational embedded system coding standards often prohibit dynamic memory usage**
  - Coding standards tools are available to enforce compliance
- **The cFS does not use C's dynamic memory functions**
  - Users are discouraged from using C's dynamic memory functions but they are not prohibited

cFS Memory Resources (3 of 3)

- **Executive Service's provides a memory pool service for applications that need dynamic memory management**

- **Applications provide the block of memory managed by the memory pool service**
  - The memory is part of the .bss processor memory section
  - Application designers are responsible for ensuring the block of memory is large enough for the application's data needs and the memory pool management data structures

- **Memory is managed in groups of fixed sized blocks**
  - Requested user data is a memory block of equal or greater inn size than the requested size
  - pools are divided is designed for efficiency and deterministic timing over memory block size flexibility

- **Memory pools creation during initialization is encouraged but there aren't any restrictions**

ES Memory Pool Examples (1 of 3)

cFE Software Bus

- cfe\modules\sb\fsw\src\cfe_sb_init.c

cFS Table Services

- cfe\modules\tbl\fsw\src\cfe_tbl_internal.c

Housekeeping App

- hk\fsw\src\hk_app.c

CCSDS File Delivery Protocol (CFDP) App

- cf\fsw\src\cf_app.c

cFS File Systems (1 of N)

- **TODO: Discuss file system relationships between RTOS, OSAL, cFS and File Manager app**

cFS Resource Identifiers

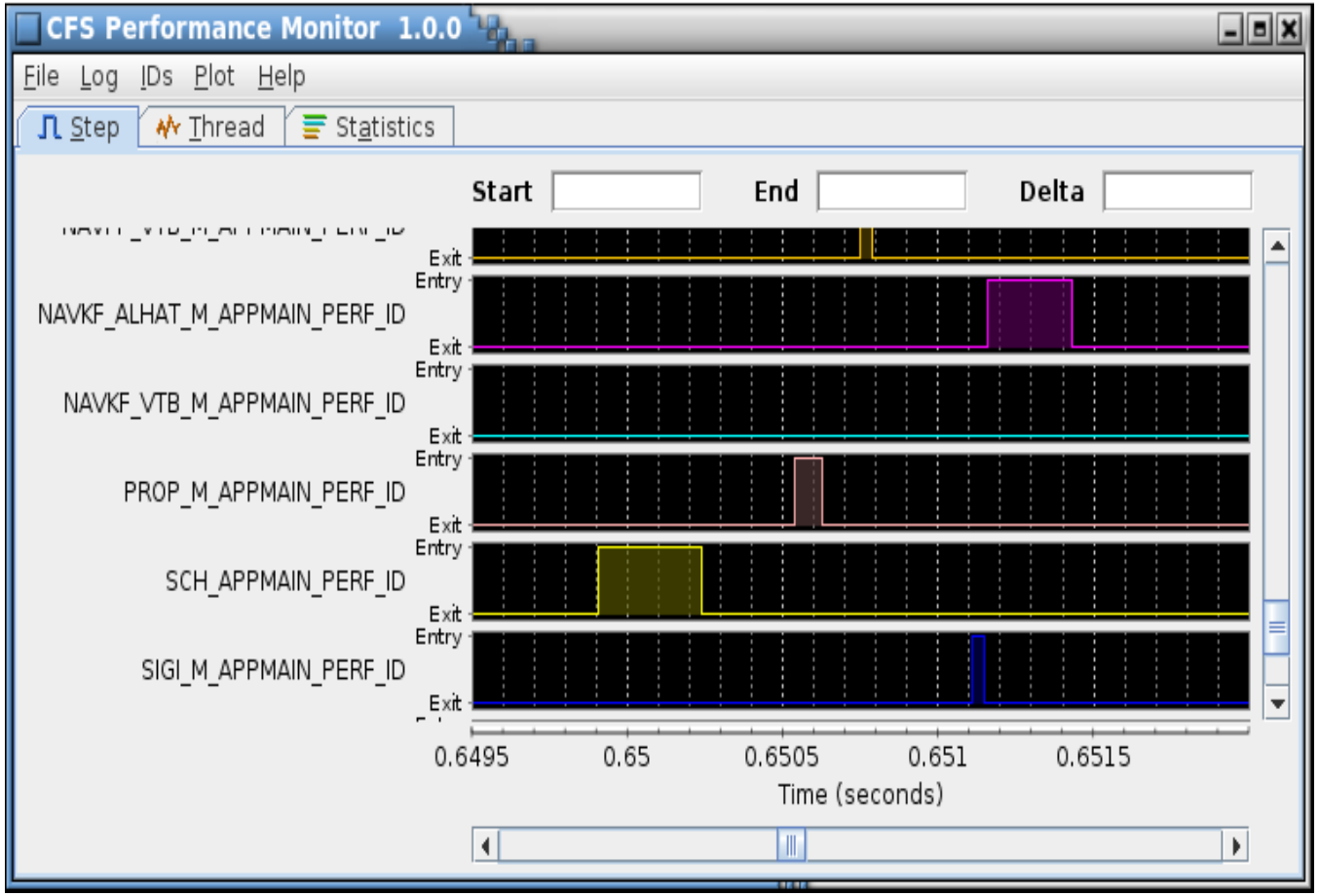
- **The cFS Calelum release in 2021 includes a Resource ID module**
- **Resource IDs are abstract data types used to uniquely identify resources managed by the cFS Framework such as**
  - App IDs, Task IDs, SB Pipe IDs, etc.
- **Resource IDs abstract implementation details and increase API type safety**
  - Even though users do not need to know Resource ID values, note that zero is not a valid value
  - This is done to prevent uninitialized zeroed memory from being treated as a valid ID
- **The Resource ID module provides utility functions to compare IDs and convert between integer types and Resource IDs**

Performance Analyzer Overview (1 of 2)

- **Provides a method to identify and measure code execution paths**
  - System tuning, troubleshooting, CPU loading
- **Executive Services allows developers to insert execution markers in FSW**
  - Entry markers indicate when execution resumes
  - Exit markers indicate when execution is suspended
  - For example, the function calls surrounding a Software Bus pend for message would be
    1. CFE_ES_PerfLogExit()
    2. CFE_SB_RcvMsg() – Pend for message
    3. CFE_ES_PerfLogEntry()
- **Operators define**
  - Which markers should be captured
  - Triggers that determine when the markers are captured

Performance Analyzer Overview (2 of 2)

Captured markers are written to a file that is transferred to the ground and displayed using the cFS Performance Monitor (CPM) tool



Performance Monitor Steps

1 **Apps call ES Performance functions to mark code execution entries/exits**

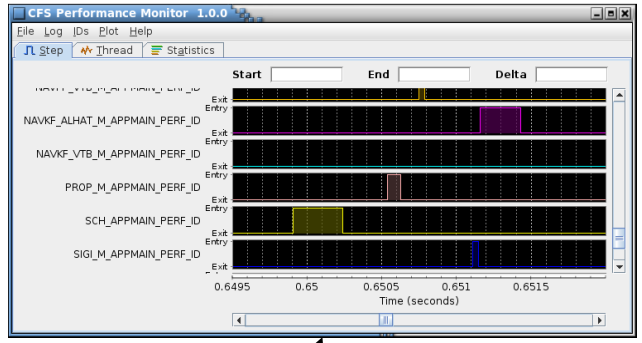
Any App

CFE_ES_PerfLogEntry(XXX_PERF_ID)
CFE_ES_PerfLogExit(XXX_PERF_ID)

Service API
Service Library

cFE Service App

2 **Operator configures ES Performance Monitor and collects data**



3 **Operator transfers the data file to the ground**

Log File

Log File

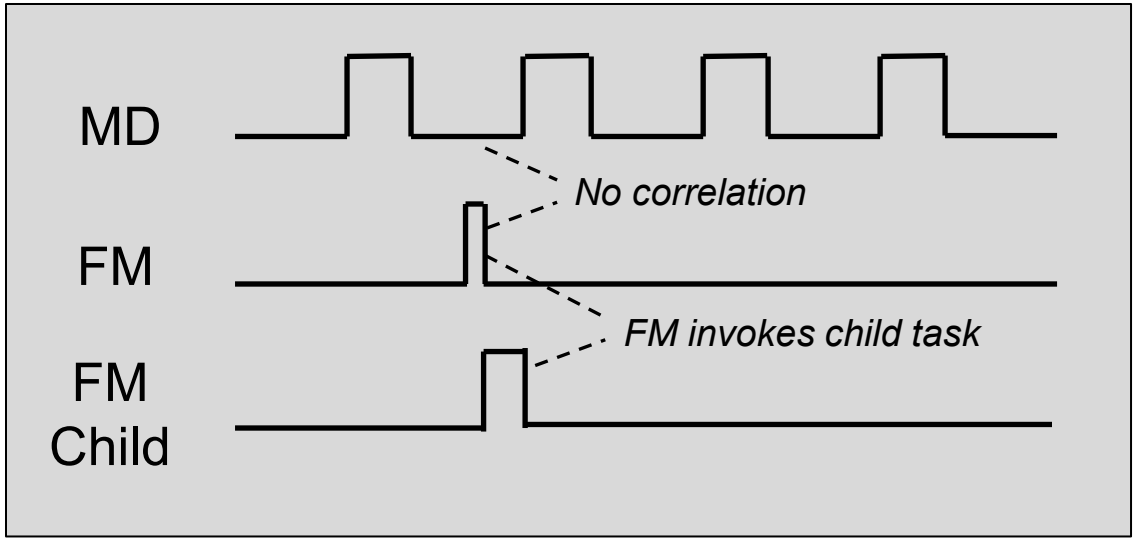
4 **Operator use CPM to display and analysis data**

cFS Performance Monitor

Performance Monitor Scenario

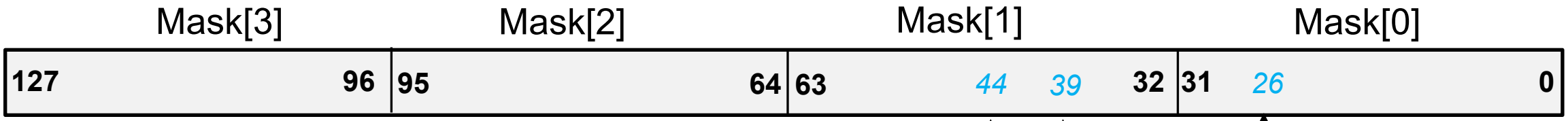
- **Trace execution of two apps**
  - File Manager: Pends indefinitely for a command to perform a file service
    - Uses a child task to perform services used in demo
  - Memory Dwell: Pends on 1Hz packet from Scheduler App
    - No correlation between MD's execution and FM's command processing.
  - "Send Housekeeping Packet Requests" from Scheduler App has been disabled for both apps
- **Data collection scenario**
  1. Start data collection
  2. Wait 4 seconds
  3. Issue FM "Send Directory in Telemetry" command
  4. Wait 4 seconds
  5. Issue FM "Write Directory to File" command
  6. Wait 4 seconds
  7. Issue FM "Send Directory in Telemetry" command
  8. Wait 4 seconds
  9. Stop data collection

Expected FM Command Trace



Performance Monitor Data Collection Configuration

- **Applications define performance IDs that must be unique within a platform**
  - Current convention is to define IDs in cfs/apps/xx/fsw/mission_inc/xx_perfids.h
- **A cFE ES command sets the "Filter Mask" that defines which performance IDs will be logged**
- **A cFE ES command sets the "Trigger Mask" that defines which performance IDs are used to start the data collection**
  - Data collection can also start when the ES collect data command is received
- **A "Masks" is an array of 32-bit words where each bit corresponds to a performance ID** ⚠



44

Perf ID	Definition
44	FM Child Task
39	FM App
26	MD App

Mask[1] = 0x00001000

Mask[1] = 0x00000080

Mask[0] = 0x04000000

⚠ Requires knowledge of binary and hexadecimal numbering systems

Executive Services Reset Behavior (1 of 2)

- **Executive Services uses the OSAL to determine the type of a processor reset**
- **The default cFE configuration recognizes two reset types**
  - Power-on: All volatile memory is initialized
  - Processor Reset: Volatile memory is preserved
- **Processor reset types are platform-specific so verifying available reset types and configuring the OSAL and cFE are part of the cFS porting activities**
- **Power-on Reset**
  - Operating system loaded and started prior to cFE
  - Initializes the file system
  - Critical data stores and logs cleared (initialized by hardware first)
  - ES starts each cFE service and then the mission applications

Executive Services Reset Behavior (2 of 2)

- **Processor Reset Preserves**
  - File system
  - Critical Data Store (CDS)
  - ES System Log
  - ES Exception and Reset (ER) log
  - Performance Analysis data
  - ES Reset info (i.e.reset type, boot source, number of processor resets)
  - Time Data (i.e. MET, SCTF, Leap Seconds)

- **A power-on reset will be performed after a configurable number of processor resets**
  - Ground operations is responsible for managing the processor reset counter

Retrieving Onboard State

- **Telemetry**
  - Housekeeping Status
    - Log file states, App, Resets, Performance Monitor, Heap Stats
- **Telemetry packets generated by command**
  - Single App Information
  - Memory Pool Statistics Packet
  - Shell command output packet, was this removed?
- **Files generated by command**
  - System Log
  - Exception-Reset Log
  - Performance Monitor
  - Critical Data Store Registry
  - All registered apps
  - All registered tasks

System Design: Child Tasks

- **Recommended practice is to create child tasks during app initialization**
- **Relative parent priority depends on the child task's role**
  - Lower priority task to perform lengthy background processing (e.g. File Manager)
  - High priority to service short duration I/O events (e.g. CFE_TIME's 1Hz task)
- **Underlying operating system implementations:**

OS	Call
Posix/Linux	pthread_create()
RTEMS	rtems_task_create()
VxWorks	taskSpawn()

System Design: Task Priorities

- **Task priorates are defined in multiple places**
  - App main task priorities are defined in *cfe_es_startup.scr* for
  - Child task priorities are typically defined in an app's platform configuration header file
- **This situation makes it hard to understand a platform's complete tasking priority profile**
- **One practice to help this situation is to define all task priorities in a single header file**
  - This file can be documented with rational for priorities
  - It provides a single location to allow someone to review and understand a platform's priority implementation
  - A downside is that it involves double booking keeping with *cfe_es_startup.scr*
- **Another solution would be to create a tool that creates artifacts (*cfe_es_startup.*scr and header files) form a single definition source**

System Design: Memory Resources (1 of 3)

- **ES memory pool configurations and task stack size definitions should be analyzed and verified to ensure they are defined with adequate margin**
- **It's easy to be complacent when the system is running without any apparent issues, however memory related errors can be subtle and difficult to debug**
  - Stack overflows that corrupt an adjacent stack's memory are notoriously deceptive
- **An attempt was made for the cFS default configurations to accommodate a "mid-size" mission**
  - A worthwhile goal, but it can lead to "accept and forget" because the system just works, or appears to
- **Doing a one-time memory utilization analysis may not be adequate if there are many changes since the analysis**
  - Being busy with the changes is precisely when another analysis gets overlooked, a classic situation of "set and forget"

System Design: Memory Resources (2 of 3)

Define and tune task stack sizes

- The main task stack sizes are defined in *cfe_es_startup.scr*
- NASA reusable apps define child task stack sizes in the app's *<app>_platform_cfg.h* file
- No standard exists for where mission-specific apps define their child task stack sizes
- Using ES's default CFE_ES_DEFAULT_STACK_SIZE is not recommended
  - This couples multiple apps to a single definition that would need to be defined to accommodate the largest app stack size
  - NASA's Checksum (CS) uses this configuration parameter

- The operating system's stack memory pool must be sized to accommodate the sum of all of the main task and child task stack sizes

TODO: Identify stack analysis techniques and tools

System Design: Memory Resources (3 of 3)

Define and tune ES memory pools

- **In order to tune ES memory pool configuration parameters such as CFE_PLATFORM_ES_MAX_MEMORY_POOLS you must know all of the memory pool users**
  - The cFE SB and TBL services use memory pools and their needs are dependent upon configuration parameters
  - The NASA CFDP (CF) and Housekeeping (HK) apps use memory pools
- **Refer to the comments in *cpu1_platform_cfg.h* for a comprehensive description on how to set the configuration parameters**
- **Exercise 3 at the end of this module demonstrates how to**

System Design: Reliability Strategies

- **Some Executive Services features can be used as part of a Fault Detection Isolation and Recovery strategy**

- **Resetting the processor is a system level FDIR strategy**
  - This can be achieved by calling CFE_PSP_Restart()
  - NASA's CheckSum app has the ability to withhold servicing the processor watchdog in certain situations
  - *cfe_es_startup.scr* can be configured to reset the processor in the evnt of an application exception

- **Managing individual app executions is a finer grained FDIR response**
  - *cfe_es_startup.scr* can be configured to restart an application if it has an exception
  - An on-board FDIR action (typically implemented in the Limit Checker App) could be to restart an app

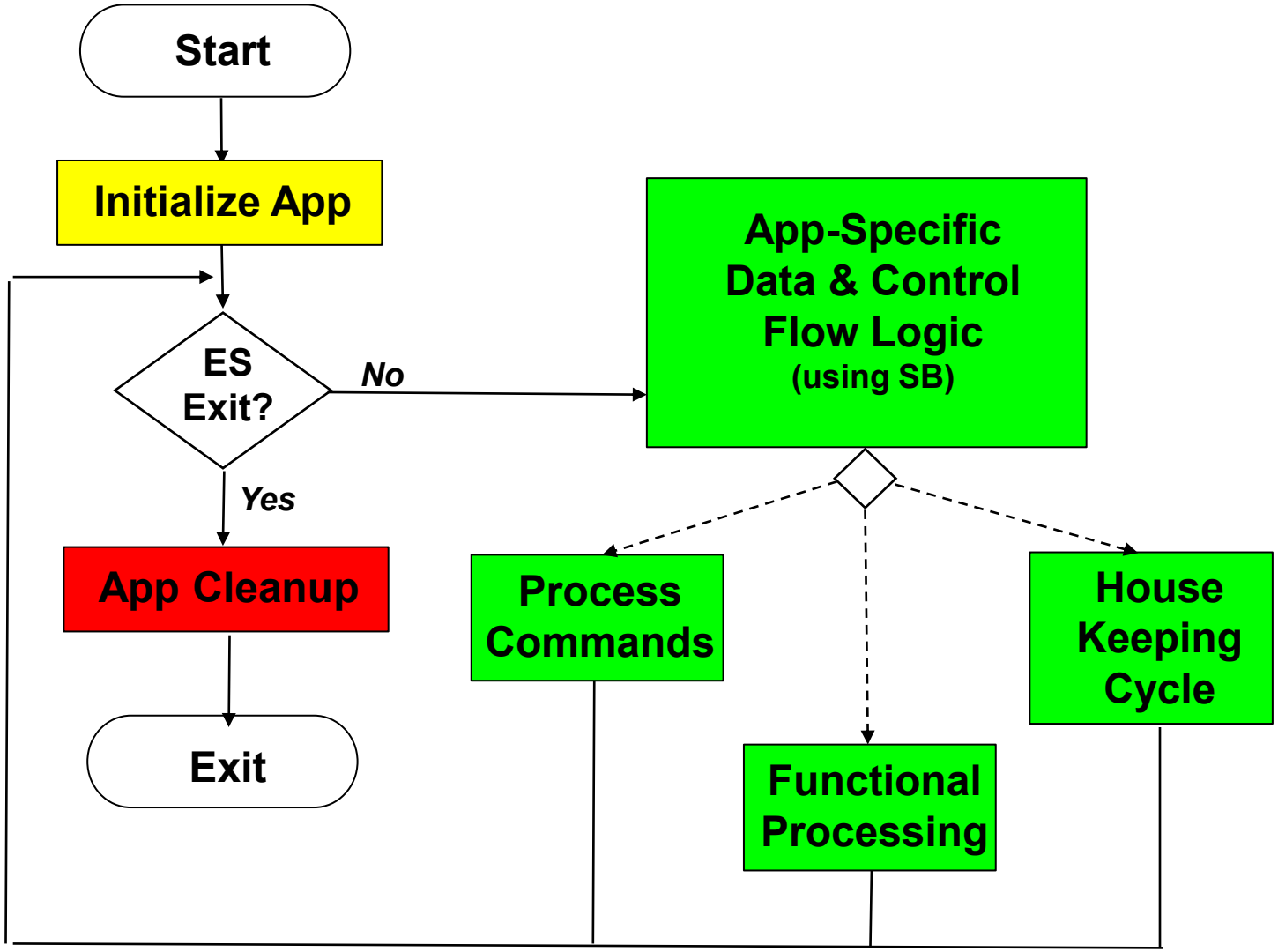
- **Ground test and onboard maintenance operations may use processor resets and individual app management features**

App Development: Initialization

- **Applications rarely if ever need to query ES for a processor startup type**
  - It's usually a binary decision as to whether data is or is not preserved across a processor reset
- **Critical Data Store is available to preserve data across a processor reset**
  - For example, the Data Storage app maintains open file management data in a CDS
- **Here's a CDS code idiom for registering a CDS in an app's initialization function**

```
Result = CFE_ES_RegisterCDS()
if (Result == CFE_SUCCESS)
   Populate CDS
else if (Result == CFE_ES_CDS_ALREADY_EXISTS)
   Restore CDS data
… Continually update CDS as application executes
```

App Development: Example Control Flow



App Development: Example Control Flow ES Calls

Initialize App

CFE_ES_WaitForStartupSync() - Used to delay an app's execution until a specific system state

App-Specific Data & Control Flow Logic (using SB)

CFE_ES_RunLoop() – App main loop check-in, allows ES to terminate app in synch with it's execution

CFE_ES_PerfLogExit() – Mark prior to suspending execution, typically before an SB receive message call

CFE_ES_PerfLogEntry() – Mark after restarting execution, typicall after an SB receive message call

App Cleanup

CFE_ES_ExitApp() – Deallocates resources, terminates any child tasks and the main task

CFE_ES_WriteToSysLog() – Log app termination. Use in case Event Service is inoperable

APIs (1 of 6)

Resource ID APIs	Purpose
CFE_ES_AppID_ToIndex	Calculates a zero-based integer value that may be used for indexing into a local resource table/array.
CFE_ES_LibID_ToIndex	Calculates a zero-based integer value that may be used for indexing into a local resource table/array.
CFE_ES_TaskID_ToIndex	Calculates a zero-based integer value that may be used for indexing into a local resource table/array.
CFE_ES_CounterID_ToIndex	Calculates a zero-based integer value that may be used for indexing into a local resource table/array.

Entry/Exit APIs	Purpose
CFE_ES_Main	This is the entry point into the cFE software.
CFE_ES_ResetCFE	This API causes an immediate reset of the cFE Kernel and all cFE Applications.

Application Control APIs	Purpose
CFE_ES_RestartApp	This API causes a cFE Application to be unloaded and restarted from the same file as the last start.
CFE_ES_ReloadApp	This API causes a cFE Application to be stopped and restarted from the specified file.
CFE_ES_DeleteApp	This API causes a cFE Application to be stopped deleted.

APIs (2 of 6)

Basecamp

App Behavior APIs	Purpose
CFE_ES_ExitApp	This API is the "Exit Point" for the cFE application
CFE_ES_RunLoop	This is the API that allows an app to check for exit requests from the system, or request shutdown from the system.
CFE_ES_WaitForSystemState	Allow an Application to Wait for a minimum global system state
CFE_ES_WaitForStartupSync	Allow an Application to Wait for the "OPERATIONAL" global system state
CFE_ES_IncrementTaskCounter	Increments the execution counter for the calling task

Child Task APIs	Purpose
CFE_ES_CreateChildTask	Creates a new task under an existing Application
CFE_ES_GetTaskIDByName	Get a Task ID associated with a specified Task name
CFE_ES_GetTaskName	Get a Task name for a specified Task ID
CFE_ES_DeleteChildTask	Deletes a task under an existing Application
CFE_ES_ExitChildTask	Exits a child task

APIs (3 of 6)

cFE Information APIs	Purpose
CFE_ES_GetResetType	Return the most recent Reset Type
CFE_ES_GetAppID	Get an Application ID for the calling Application
CFE_ES_GetTaskID	Get the task ID of the calling context
CFE_ES_GetAppIDByName	Get an Application ID associated with a specified Application name
CFE_ES_GetLibIDByName	Get a Library ID associated with a specified Library name
CFE_ES_GetAppName	Get an Application name for a specified Application ID
CFE_ES_GetLibName	Get a Library name for a specified Library ID
CFE_ES_GetAppInfo	Get Application Information given a specified App ID
CFE_ES_GetTaskInfo	Get Task Information given a specified Task ID
CFE_ES_GetLibInfo	Get Library Information given a specified Resource ID
CFE_ES_GetModuleInfo	Get Information given a specified Resource ID

APIs (4 of 6)

Miscellaneous APIs	Purpose
CFE_ES_BackgroundWakeup	Wakes up the CFE background task
CFE_ES_WriteToSysLog	Write a string to the cFE System Log
CFE_ES_CalculateCRC	Calculate a CRC on a block of memory
CFE_ES_ProcessAsyncEvent	Notification that an asynchronous event was detected by the underlying OS/PSP

Critical Data Store APIs	Purpose
CFE_ES_RegisterCDS	Reserve space (or re-obtain previously reserved space) in the Critical Data Store (CDS)
CFE_ES_GetCDSBlockIDByName	Get a CDS Block ID associated with a specified CDS Block name
CFE_ES_GetCDSBlockName	Get a Block name for a specified Block ID
CFE_ES_CopyToCDS	Save a block of data in the Critical Data Store (CDS)
CFE_ES_RestoreFromCDS	Recover a block of data from the Critical Data Store (CDS)

APIs (5 of 6)

Memory Manager APIs	Purpose
CFE_ES_PoolCreateNoSem	Initializes a memory pool created by an application without using a semaphore during processing.
CFE_ES_PoolCreate	Initializes a memory pool created by an application while using a semaphore during processing.
CFE_ES_PoolCreateEx	Initializes a memory pool created by an application with application specified block sizes.
CFE_ES_PoolDelete	Deletes a memory pool that was previously created
CFE_ES_GetPoolBuf	Gets a buffer from the memory pool created by #CFE_ES_PoolCreate or #CFE_ES_PoolCreateNoSem
CFE_ES_GetPoolBufInfo	Gets info on a buffer previously allocated via #CFE_ES_GetPoolBuf
CFE_ES_PutPoolBuf	Releases a buffer from the memory pool that was previously allocated via #CFE_ES_GetPoolBuf
CFE_ES_GetMemPoolStats	Extracts the statistics maintained by the memory pool software

Performance Monitor APIs	Purpose
CFE_ES_PerfLogEntry	Entry marker for use with Software Performance Analysis Tool.
CFE_ES_PerfLogExit	Exit marker for use with Software Performance Analysis Tool.
CFE_ES_PerfLogAdd	Adds a new entry to the data buffer

APIs (6 of 6)

Generic Counter APIs	Purpose
CFE_ES_RegisterGenCounter	This routine registers a generic thread-safe counter which can be used for inter-task management.
CFE_ES_DeleteGenCounter	This routine deletes a previously registered generic counter.
CFE_ES_IncrementGenCounter	This routine increments the specified generic counter.
CFE_ES_SetGenCount	This routine sets the specified generic counter to the specified value.
CFE_ES_GetGenCount	This routine gets the value of a generic counter.
CFE_ES_GetGenCounterIDByName	Get the Id associated with a generic counter name
CFE_ES_GetGenCounterName	Get a Counter name for a specified Counter ID

APIs (6 of 6)

Generic Counter APIs	Purpose
CFE_ES_RegisterGenCounter	This routine registers a generic thread-safe counter which can be used for inter-task management.
CFE_ES_DeleteGenCounter	This routine deletes a previously registered generic counter.
CFE_ES_IncrementGenCounter	This routine increments the specified generic counter.
CFE_ES_SetGenCount	This routine sets the specified generic counter to the specified value.
CFE_ES_GetGenCount	This routine gets the value of a generic counter.
CFE_ES_GetGenCounterIDByName	Get the Id associated with a generic counter name
CFE_ES_GetGenCounterName	Get a Counter name for a specified Counter ID

Command List

Parameter	Purpose
CFE_ES_StartPerfDataCmd	Start performance data
CFE_ES_StopPerfDataCmd	Stop performance data
CFE_ES_SetPerfFilterMaskCmd	Set performance filter mask
CFE_ES_SetPerfTriggerMaskCmd	Set performance trigger mask
CFE_ES_HousekeepingCmd	On-board command (HK request)
CFE_ES_NoopCmd	ES task ground command (NO-OP)
CFE_ES_ResetCountersCmd	ES task ground command (reset counters)
CFE_ES_RestartCmd	Restart cFE (may reset processor)
CFE_ES_StartAppCmd	Load (and start) single application
CFE_ES_StopAppCmd	Stop single application
CFE_ES_RestartAppCmd	Restart a single application
CFE_ES_ReloadAppCmd	Reload a single application
CFE_ES_QueryOneCmd	Request tlm packet with single app data
CFE_ES_QueryAllCmd	Write all app data to file
CFE_ES_QueryAllTasksCmd	Write all Task Data to a file
CFE_ES_ClearSyslogCmd	Clear executive services system log
CFE_ES_OverWriteSyslogCmd	Set syslog mode
CFE_ES_WriteSyslogCmd	Process Cmd to write ES System Log to file
CFE_ES_ClearERLogCmd	Clear The exception and reset log
CFE_ES_WriteERLogCmd	Process Cmd to write exception & reset log to a file
CFE_ES_VerifyCmdLength	Verify command packet length
CFE_ES_ResetPRCountCmd	ES task ground command (Processor Reset Count)
CFE_ES_SetMaxPRCountCmd	Set Maximum Processor reset count
CFE_ES_DeleteCDSCmd	Delete Specified Critical Data Store
CFE_ES_SendMemPoolStatsCmd	Telemeter Memory Pool Statistics
CFE_ES_DumpCDSRegistryCmd	Dump CDS Registry to a file

Platform Configuration Parameters

Parameter	Purpose
CFE_PLATFORM_ES_MAX_APPLICATIONS	Max Number of Applications
CFE_PLATFORM_ES_MAX_LIBRARIES	Max Number of Shared libraries
CFE_PLATFORM_ES_ER_LOG_ENTRIES	Max Number of ER (Exception and Reset) log entries
CFE_PLATFORM_ES_ER_LOG_MAX_CONTEXT_SIZE	Maximum size of CPU Context in ES Error Log
CFE_PLATFORM_ES_SYSTEM_LOG_SIZE	Size of the cFE System Log
CFE_PLATFORM_ES_OBJECT_TABLE_SIZE	Number of entries in the ES Object table
CFE_PLATFORM_ES_MAX_GEN_COUNTERS	Max Number of Generic Counters
CFE_PLATFORM_ES_APP_SCAN_RATE	ES Application Control Scan Rate
CFE_PLATFORM_ES_APP_KILL_TIMEOUT	ES Application Kill Timeout
CFE_PLATFORM_ES_RAM_DISK_SECTOR_SIZE	ES Ram Disk Sector Size
CFE_PLATFORM_ES_RAM_DISK_NUM_SECTORS	ES Ram Disk Number of Sectors
CFE_PLATFORM_ES_RAM_DISK_PERCENT_RESERVED	Percentage of Ram Disk Reserved for Decompressing Apps
CFE_PLATFORM_ES_RAM_DISK_MOUNT_STRING	RAM Disk Mount string
CFE_PLATFORM_ES_CDS_SIZE	Critical Data Store Size
CFE_PLATFORM_ES_USER_RESERVED_SIZE	User Reserved Memory Size
CFE_PLATFORM_ES_RESET_AREA_SIZE	ES Reset Area Size
CFE_PLATFORM_ES_NONVOL_STARTUP_FILE	ES Nonvolatile Startup Filename
CFE_PLATFORM_ES_NONVOL_DISK_MOUNT_STRING	Default virtual path for persistent storage
CFE_PLATFORM_ES_VOLATILE_STARTUP_FILE	ES Volatile Startup Filename
CFE_PLATFORM_ES_DEFAULT_APP_LOG_FILE	Default Application Information Filename
CFE_PLATFORM_ES_DEFAULT_TASK_LOG_FILE	Default Application Task Information Filename
CFE_PLATFORM_ES_DEFAULT_SYSLOG_FILE	Default System Log Filename
CFE_PLATFORM_ES_DEFAULT_ER_LOG_FILE	Default Exception and Reset (ER) Log Filename
CFE_PLATFORM_ES_DEFAULT_PERF_DUMP_FILENAME	Default Performance Data Filename
CFE_PLATFORM_ES_DEFAULT_CDS_REG_DUMP_FILE	Default Critical Data Store Registry Filename
CFE_PLATFORM_ES_DEFAULT_POR_SYSLOG_MODE	Default System Log Mode following Power On Reset

Platform Configuration Parameters

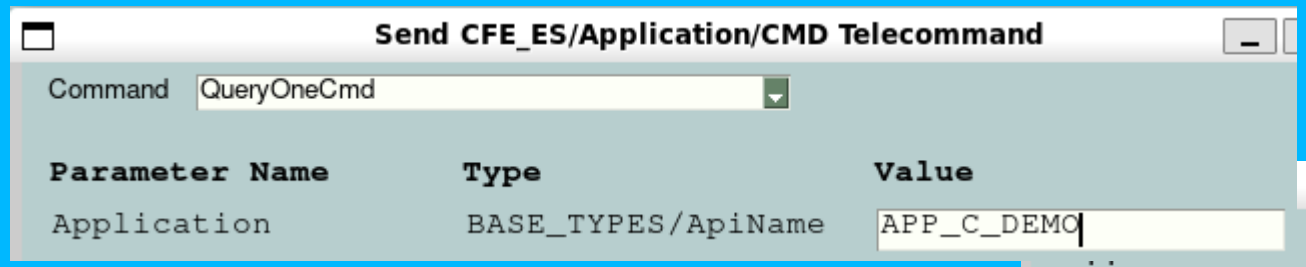
Parameter	Purpose
CFE_PLATFORM_ES_DEFAULT_PR_SYSLOG_MODE	Default System Log Mode following Processor Reset
CFE_PLATFORM_ES_PERF_DATA_BUFFER_SIZE	Max Size of Performance Data Buffer
CFE_PLATFORM_ES_PERF_FILTMASK_NONE	Filter Mask Setting for Disabling All Performance Entries
CFE_PLATFORM_ES_PERF_FILTMASK_ALL	Filter Mask Setting for Enabling All Performance Entries
CFE_PLATFORM_ES_PERF_FILTMASK_INIT	Default Filter Mask Setting for Performance Data Buffer
CFE_PLATFORM_ES_PERF_TRIGMASK_NONE	Default Filter Trigger Setting for Disabling All Performance Entries
CFE_PLATFORM_ES_PERF_TRIGMASK_ALL	Filter Trigger Setting for Enabling All Performance Entries
CFE_PLATFORM_ES_PERF_TRIGMASK_INIT	Default Filter Trigger Setting for Performance Data Buffer
CFE_PLATFORM_ES_PERF_CHILD_PRIORITY	Performance Analyzer Child Task Priority
CFE_PLATFORM_ES_PERF_CHILD_STACK_SIZE	Performance Analyzer Child Task Stack Size
CFE_PLATFORM_ES_PERF_CHILD_MS_DELAY	Performance Analyzer Child Task Delay
CFE_PLATFORM_ES_PERF_ENTRIES_BTWN_DLYS	Performance Analyzer Child Task Number of Entries Between Delay
CFE_PLATFORM_ES_DEFAULT_STACK_SIZE	Default Stack Size for an Application
CFE_PLATFORM_ES_START_TASK_PRIORITY	ES Task Priority
CFE_PLATFORM_ES_START_TASK_STACK_SIZE	ES Task Stack Size
CFE_PLATFORM_ES_CDS_MAX_NUM_ENTRIES	Maximum Number of Registered CDS Blocks
CFE_PLATFORM_ES_MAX_PROCESSOR_RESETS	Number of Processor Resets Before a Power On Reset
CFE_PLATFORM_ES_CDS_MAX_BLOCK_SIZE	ES Critical Data Store Max Memory Pool Block Size
CFE_PLATFORM_ES_MEMPOOL_ALIGN_SIZE_MIN	Define Memory Pool Alignment Size
CFE_PLATFORM_ES_POOL_MAX_BUCKETS	Maximum number of block sizes in pool structures
CFE_PLATFORM_ES_MAX_MEMORY_POOLS	Maximum number of memory pools
CFE_PLATFORM_ES_STARTUP_SYNC_POLL_MSEC	Poll timer for startup sync delay
CFE_PLATFORM_ES_STARTUP_SCRIPT_TIMEOUT_MSEC	Startup script timeout

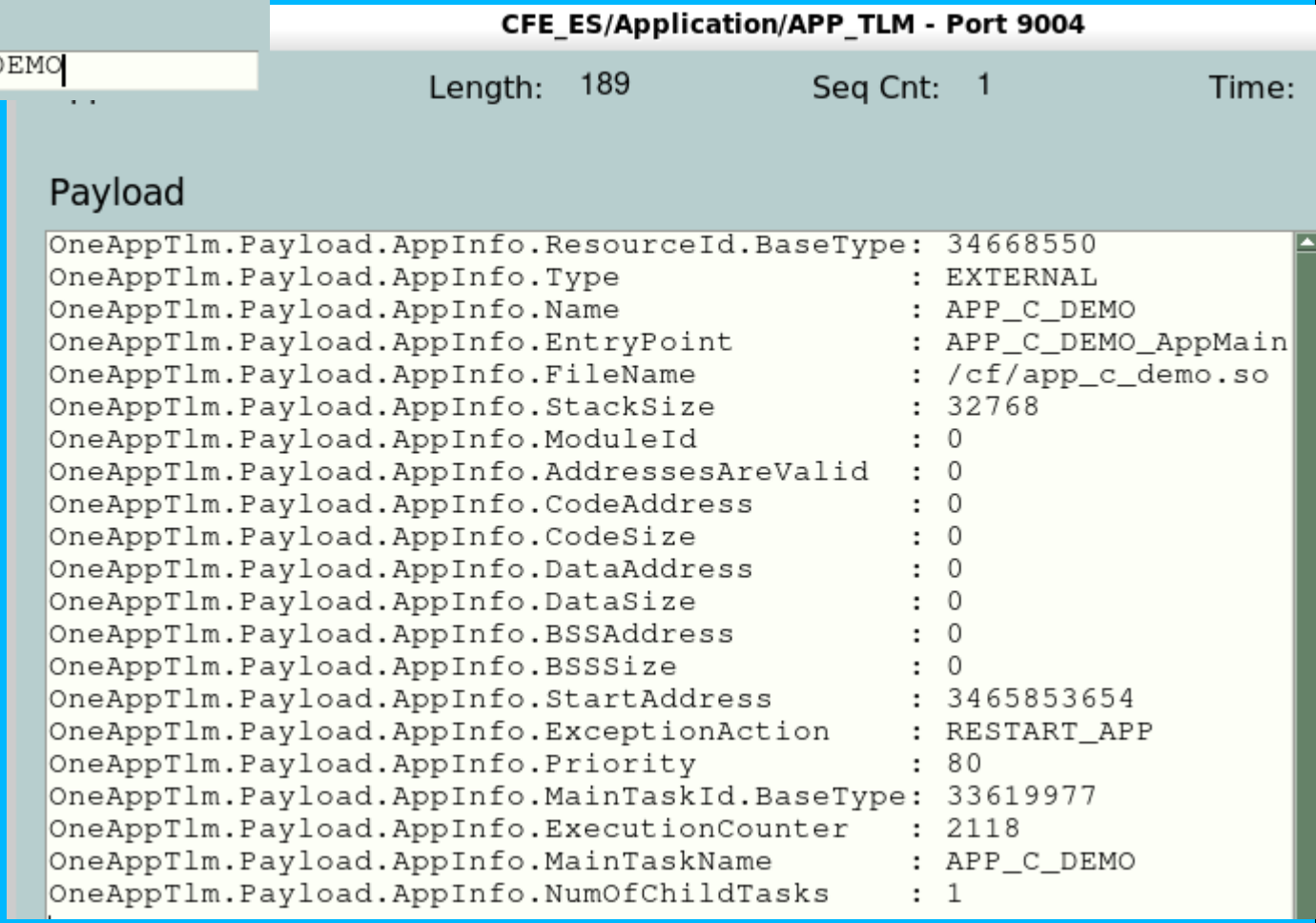
Mission Configuration Parameters

Parameter	Purpose
CFE_MISSION_ES_CDS_MAX_NAME_LENGTH	Maximum Length of CDS Name
CFE_MISSION_ES_DEFAULT_CRC	Mission Default CRC algorithm
CFE_MISSION_ES_MAX_APPLICATIONS	Mission Max Apps in a message
CFE_MISSION_ES_PERF_MAX_IDS	Define Max Number of Performance IDs for messages
CFE_MISSION_ES_POOL_MAX_BUCKETS	Maximum number of block sizes in pool structures
CFE_MISSION_ES_CDS_MAX_FULL_NAME_LEN	Maximum Length of Full CDS Name in messages

Exercise 1 – Query Demo App

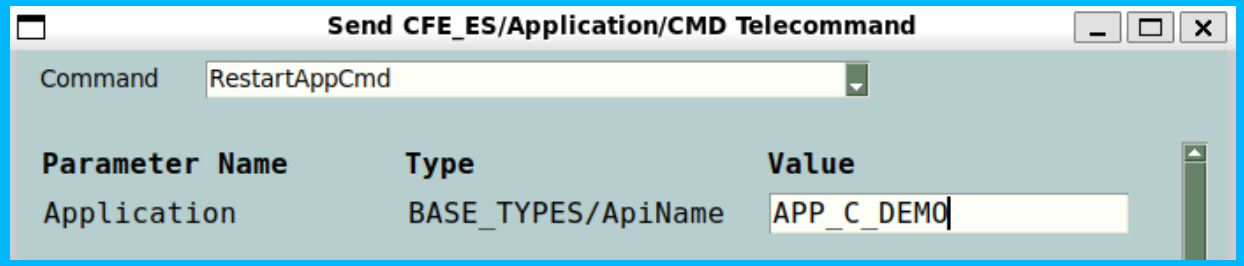
1. Open CFE_ES APP_TLM and issue CFE_ES *QueryOneCmd* for Basecamp's demo app APP_C_DEMO

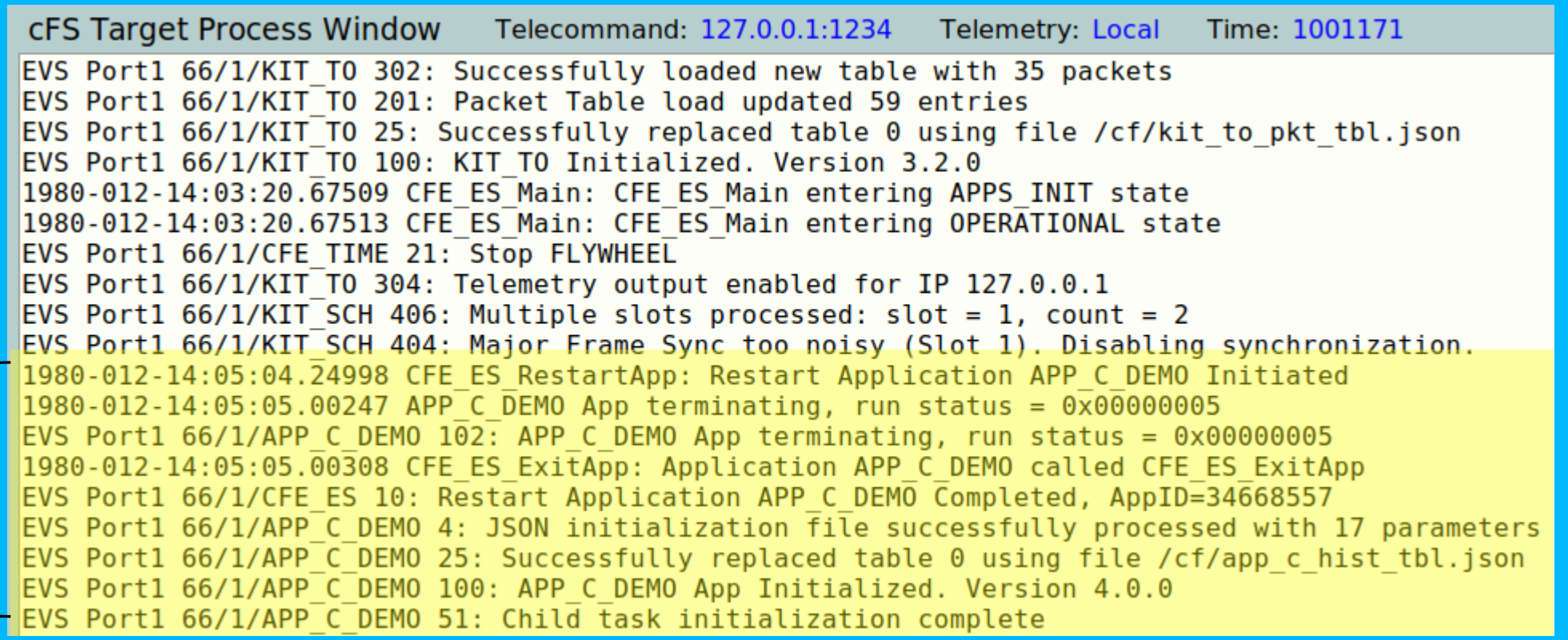




Exercise 2 – Reset Demo App

1. Issue CFE_ES *RestartAppCmd* for Basecamp's demo app APP_C_DEMO





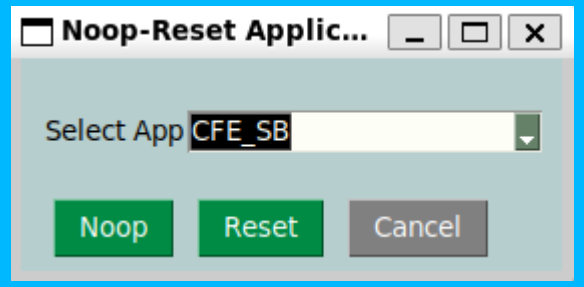
Events Associated With Restart

Exercise 3 – Memory Pool Statistics (1 of 5)

- In this exercise we'll use ES's SendMemPoolStatsCmd to access details statistics about SB's use of ES's memory pool service. The SB module includes an exercise that looks into the details of the statistics and how to interpret them and use them for tuning the system.

- ES's SendMemPoolStatsCmd requires the memory pool's handle which is a problem because it is not diretlyu available to ground operators. A trouble ticket has been submitted. Basecamp's SB's Noop command has been modified to send SB's memory pool handle. Slide 3 of this exercise shows another method to find the handle address but it's not perfect.

1. Using the Quick Cmd dropdown send a CFE_SB Noop command.

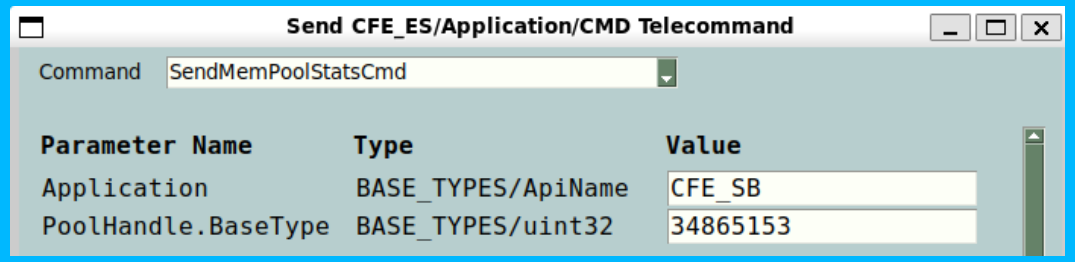


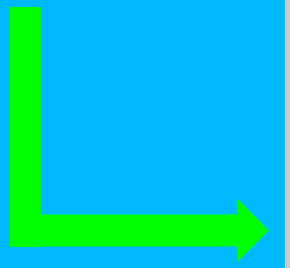
This is handle value will be used in the SendMemPoolStatsCmd

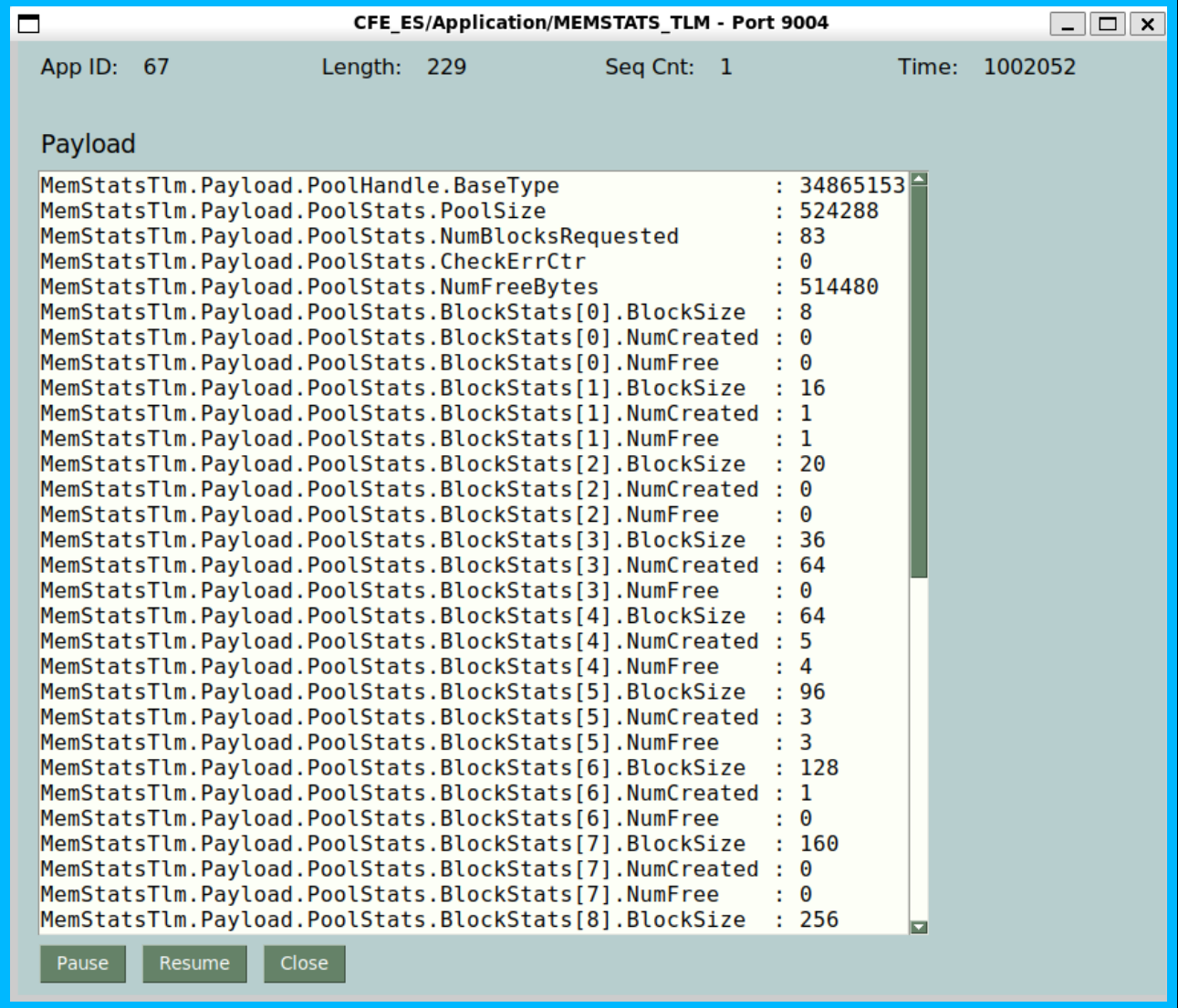
```
EVS Port1 66/1/CFE_SB 28: No-op Cmd Rcvd:  cFE DEVELOPMENT BUILD v6.8.0-rc1+dev1024 (Codename: Bootes), Last Official Release: cfe
v6.7.0
EVS Port1 66/1/CFE_SB 28: CFE_SB_Global.Mem.PoolHdl = 34865153
```

Exercise 3 – Memory Pool Statistics (2 of 5)

2. Send ES's SendMemPoolStatsCmd using the memory handle value obtained from the Noop command







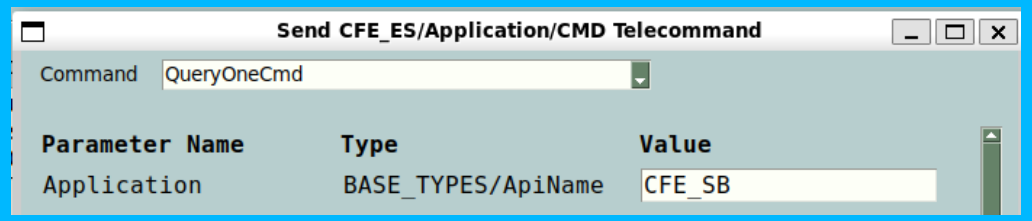
Each BlockStat array element contains:

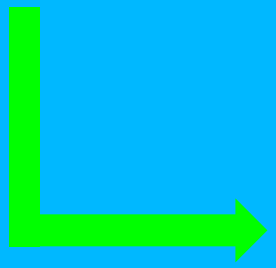
- BlockSize
  - TODO
- NumCreated
  - TODO
- NumFree
  - TODO

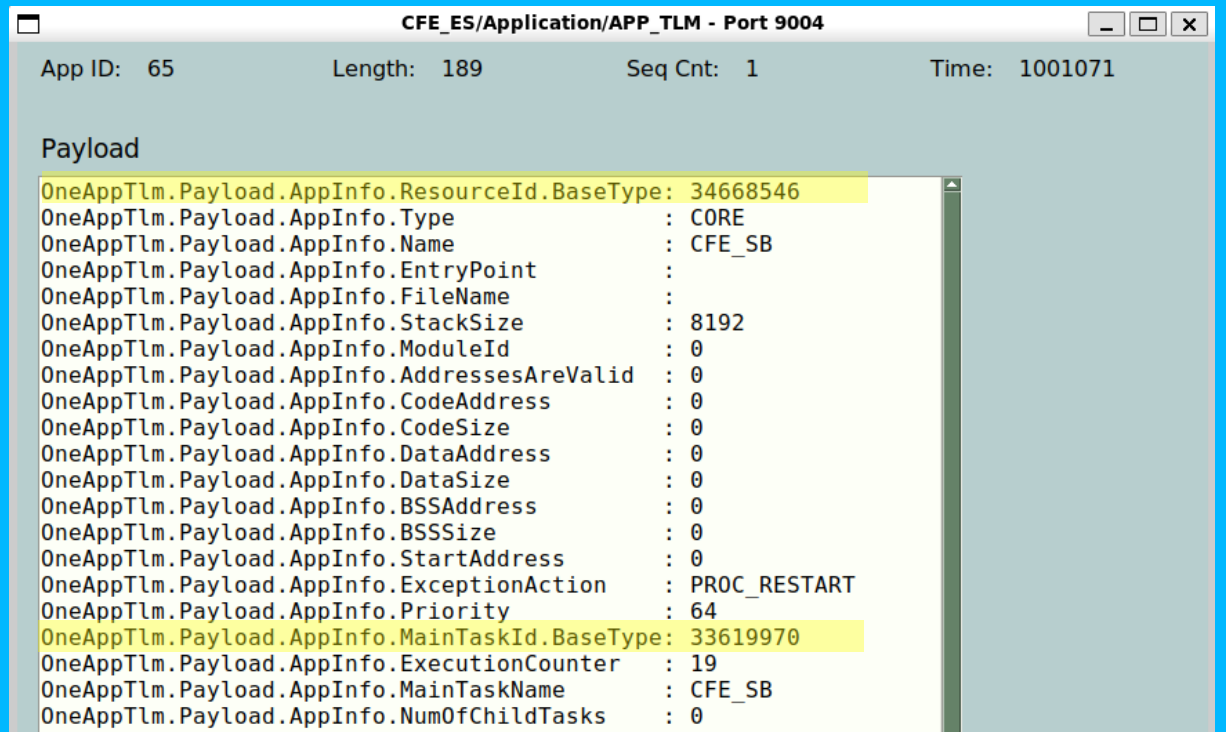
Exercise 3 – Memory Pool Statistics (3 of 5)

- This slide and the next one show a method for how to obtain an app's memory pool handle although it's not perfect

1. Launch the CFE_ES APP_TLM page and issue a CFE_ES QueryOneCmd to







Exercise 3 – Memory Pool Statistics (4 of 5)

2. Osapi-idmap.h

```
#define OS_OBJECT_TYPE_UNDEFINED   0x00 /**< @brief Object type undefined */
#define OS_OBJECT_TYPE_OS_TASK     0x01 /**< @brief Object task type */
#define OS_OBJECT_TYPE_OS_QUEUE    0x02 /**< @brief Object queue type */
#define OS_OBJECT_TYPE_OS_COUNTSEM 0x03 /**< @brief Object counting semaphore type */
#define OS_OBJECT_TYPE_OS_BINSEM   0x04 /**< @brief Object binary semaphore type */
#define OS_OBJECT_TYPE_OS_MUTEX    0x05 /**< @brief Object mutex type */
#define OS_OBJECT_TYPE_OS_STREAM   0x06 /**< @brief Object stream type */
#define OS_OBJECT_TYPE_OS_DIR      0x07 /**< @brief Object directory type */
#define OS_OBJECT_TYPE_OS_TIMEBASE 0x08 /**< @brief Object timebase type */
#define OS_OBJECT_TYPE_OS_TIMECB   0x09 /**< @brief Object timer callback type */
#define OS_OBJECT_TYPE_OS_MODULE   0x0A /**< @brief Object module type */
#define OS_OBJECT_TYPE_OS_FILESYS  0x0B /**< @brief Object file system type */
#define OS_OBJECT_TYPE_OS_CONSOLE  0x0C /**< @brief Object console type */
#define OS_OBJECT_TYPE_USER        0x10 /**< @brief Object user type */
```

3. Cfe_core_resourceid_basevalues.h

```
     */
    CFE_RESOURCEID_ES_TASKID_BASE_OFFSET = OS_OBJECT_TYPE_OS_TASK,

    /* Other ES managed resources */
    CFE_RESOURCEID_ES_APPID_BASE_OFFSET      = OS_OBJECT_TYPE_USER + 1,
    CFE_RESOURCEID_ES_LIBID_BASE_OFFSET      = OS_OBJECT_TYPE_USER + 2,
    CFE_RESOURCEID_ES_COUNTID_BASE_OFFSET    = OS_OBJECT_TYPE_USER + 3,
    CFE_RESOURCEID_ES_POOLID_BASE_OFFSET     = OS_OBJECT_TYPE_USER + 4,
    CFE_RESOURCEID_ES_CDSBLOCKID_BASE_OFFSET = OS_OBJECT_TYPE_USER + 5,

    /* SB managed resources */
    CFE_RESOURCEID_SB_PIPEID_RESOURCE_BASE_OFFSET = OS_OBJECT_TYPE_USER + 6,

    /* configuration registry */
    CFE_RESOURCEID_CONFIGID_BASE_OFFSET = OS_OBJECT_TYPE_USER + 7,
```

Consider moving to resource ID section

Exercise 3 – Memory Pool Statistics (5 of 5)

2. Resource ID Mapping
   - Resource_ID User_ID
   - Compile Time Runtime: ID computed based on order of resource acquisition
   - 0x0210 Task
   - 0x0211 App
   - 0x0212 Library
   - 0x0213 Count
   - 0x0214 Pool
   - 0x0215 CDS
   - 0x0216 Pipe
   - 0x0217 Config

3. CFE_SB Resources
   - 34865153 => 0x0214 0001 (1 is first ID, 0 is not a valid ID to avoid non-initialized valid ID)

4. CFE_SB Query
   - 33,619,970 => 0x0201 0002 ## OSAL Task ID and SB is the second CFE app instantiated. Doesn't agree with earlier slide or is there non-determinism in task ID creation?
   - 34668546 => 0x0211 0002 ## App ID

5. CFE_TBL Pool ID computation
   - Knowing CFE_TBL is instantiated after CFE_SB adding 1 to SB's Pool ID works
   - 34865154 => 0x0214 0002

Consider moving to resource ID section

Exercise 4 –Performance Monitor

Follow the instructions in cfs-basecamp/gnd-sys/cmd-sender/cfe_es_perf_mon_exercise.txt