cFE Core Services:
Document: cFE Application Developers Guide

How I image cFS - Imagine a collection of people standing in a circle all on the same phone call. When someone wants to talk (send a message), they have to ask permission (create the pipe), make sure you're still on the call and get everyone's attention (request data to be put on the pipe), decide who they want to talk to on the call and what to say (Generate Message), they tell that person (or people) what the message is (Send SB Message).

People - cFS Applications
Talking - Sending command or message

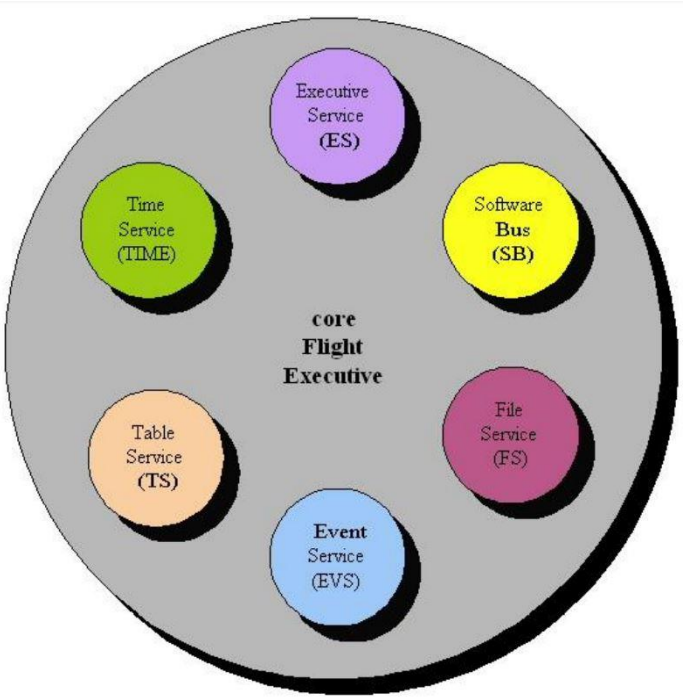
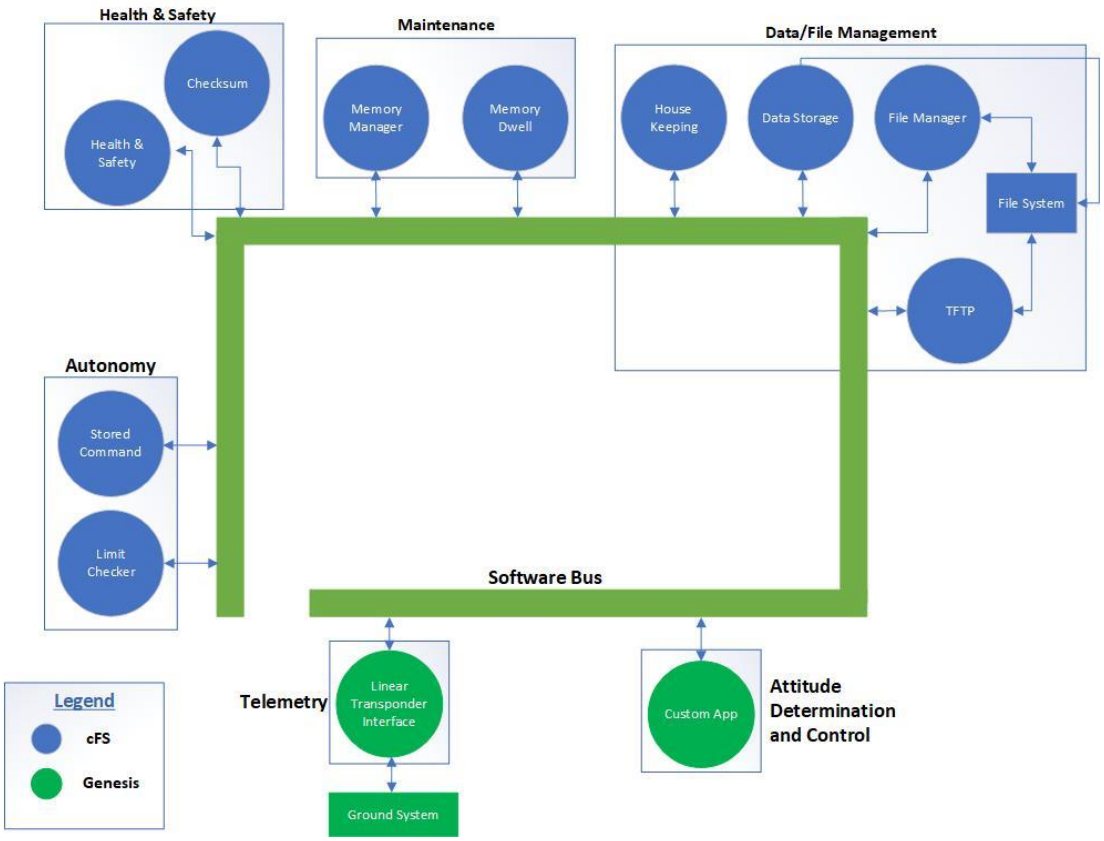


Figure 1: cFE Core Services



Header Files:

- #include “cfe.h”
- Other header files can be found in the “.../cfe-core/inc/” directory

cFE Application Template

- Applications designed to interface with the cFE should follow standard templates.
- cFS application must register their tables with cFE
- Following registration, tables must be initialized, either from a file or from memory

Inside the template

Sending an SB Message to the cFE (HousekeepingCmd)

1.) The contents of the message are stored in a data structure that has been initialized (see TaskInit function)
2.) The mMessage is time stamped with the appropriate mission time stamp
3.) The Message is given to the Software Bus for routing to subscribers pipes

Processing received Command Messages: (NoopCmd function)

1.) Validation of the command
    a.) I.e checking to see if the packet length matches the expected packet length
2.) Performing the action requested by the command

3.) Notification of the operator that the command has succeeded.

Accessing Table Data (RoutineProcessingCmd)

- Requires a combination of GetAddress and RelaseAddress
    - GetAddress - allows the owner of the table (i.e the Application that registered the table) or the Application that is sharing the table to access the table data
- Applications must ReleaseAddress in order for Table Services to be able to manipulate the table

Executive Services Interface

- API calls will start with “CFE_ES_” b/c they are members of the cFE Executive Services API
- Start with “OS_” because they are part of the OS Abstraction Layer
- Applications must register with ES when started
    - CFE_ES_RegisterApp function
- Printf()
    - Int32 CFE_ES_WriteToSysLog(const char *pSpecString, ..);
    - OS_printf(“words that will get printed”);
- Application Registration
    - All Apps MUST register immediately with ES when started
        - CFE_ES_RegisterApp
- Application Names and IDs
    - CFE_ES_GetAppIDByName
        - Return the numeric Application ID when given an App name
    - CFE_ES_GetAppName
        - Return the App name when given the App ID
- Application Start-Up Types
    - CFE_ES_GetAppRestartType will return
        - CFE_ES_APP_POWERON_RESET
        - CFE_ES_APP_COLD_RESET
        - CFE_ES_APP_WARM_RESET

Software Bus Interface

- Message based subscription approach for establishing communication paths
- In order to receive a SB Message
    - App must first create a Pipe on which to receive messages
- One - to - One, One - to - Many, Many - to - One routing Configurations
- Message ID

    - Identify what the data is
    - Who would like to receive it
- Creating a Pipe (CFE_SB_CreatePipe()):
    - &QQ_AppData.QQ_Pipe_1 - Variable to hold Pipe ID (struct)
    - QQ_PIPE_1_DEPTH - Depth of Pipe (#define)
        - How many SB Messages that can be queued in the Pipe before overrun
    - QQ_PIPE_1_NAME - Name of Pipe (#define)
- Message Subscription:
    - Request data to be put into that Pipe
    - Define Message IDs in **app_msgid.h**
    - CFE_SB_SubscribeEX()
        - QQ_CMDID_1 - Msg ID to Receive (#define)
        - QQ_AppData.QQ_Pipe_1 - Pipe Msg is to be Rcvd on (struct)
        - CFE_SB_DEFAULT_QOS - Quality of Service
            - Determines the priority and the reliability of Message
            - CFE_SB_DEFAULT_QOD - most common (default)
        - QQ_CMDID_1_LIMIT - Max Number to Queue (#define)
    - Most applications do NOT need to worry about QoS nor Message Limit
        - CFE_SB_Subscribe
- Message Unsubscribing
    - App no longer wishes to receive a Message
    - CFE_SB_Unsubscribe():
        - QQ_CMDID_1 - Msg Id to Not Receive
        - QQ_AppData.QQ_Pipe_1 - Pipe Msg currently Rcvd on
- Creating Software Bus Messages:
    - App shall define the data structure of the SB Message
    - Allocate memory for it (instantiate it)
    - Initialize it with appropriate SB Message Header information
    - Fill the rest of the structure with appropriate data
    - CFE_SB_InitMsg()
        - &QQ_AppData.HkPacket, -Address of SB Message Data Buffer
        - QQ_HK_TLM_MID, -SB Message ID associated with Data
        - sizeof(QQ_HkPacket_t), -Size of Buffer
        - CFE_SB_CLEAR_DATA); -Buffer should be cleared by cFE
    - Defining allocating space for SB Message Header in struct
        - CFE_SB_TLM_HDR_SIZE - Table
        - CFE_SB_CMD_HDR_SIZE - Command
- Sending Software Bus Message

    - Construct the SB Message in memory
    - Set its contents to the appropriate values
    - Then the app calls CFE_SB_SendMsg()

- Large SB Messages
    - Uses the “Zero Copy” Protocol (in Section 6.7 of App Dev Guide)
- Receiving Software Bus Messages:
    - CFE_SB_RcvMsg()
        - &QQ_AppData.MsgPtr,
        - QQ_AppData.CmdPipe,
        - CFE_SB_PEND_FOREVER
    - Process Software Bus Message
        - QQ_AppPipe(QQ_AppData.MsgPtr);
    - CFE_SB_GetUserData
        - Obtain the start address of the SB Message
- Deleting Software Bus Pipes:
    - CFE_SB_DeletePipe(QQ_Pipe_1) -Delete pipe created earlier

Event Service Interface:

- Event Messages
    - Informational text generated by an application in response to commands, software errors, hardware errors, application-initialization, etc.
    - Will ignore all function calls from unregistered applications
    - Visible to the user (printf statements)
    - Max 122 characters
- Event Types
    - CFE_EVS_DEBUG
    - CFE_EVS_INFORMATION
    - CFE_EVS_ERROR
    - CFE_EVS_CRITICAL
- Event Service Registration
    - CFE_EVS_Register
        - Must register with EVS in order to use cFE event services
- Sending an Event Message
    - CFE_EVS_SendEvent()
        - Ex.) CFE_EVS_SendEvent(EventID, EventType, "Unknown stream on cmd pipe: 0x%04X", sid);
    - CFE_EVS_SendTimedEvent()
        - Ex.) CFE_EVS_SendTimedEvent(PktTime, EventID, EventType, "CSS Data Bad: 0x%04X", CssData);

- Event Service Un-registration
    - Unregister from EVS
        - CFE_EVS_Unregister ();

Table Service Interface

- Table
    - Related set of data values that can be loaded and dumped as a single unit by the ground
        - Equivalent to a C structure or array)
    - Used to give ground operators the ability to update constants used by the flight software
    - Dumping infrequently needed status information to the ground on command
    - Active Table - table that an Application can obtain a pointer to and can access the data stored within the Table
    - Inactive Table - is a complete copy of the Active Table that can be operated on either via ground or stored commands.
- Validating a Table
    - Either Active or Inactive
    - Two things happen:
        1.) Table Services calculates the current Data Integrity Value for the table contents.
        2.) The owning task, if it has registered a validation function, is notified that Validation request has been made
- Registering Tables
    - Application must request that a Table Image is created
    - CFE_TBL_Register
        - &MyTableHandle, -Table Handle (to be returned)
        - "MyTableName", - Application specific Table Name
        - sizeof(QQ_MyTable_t), -Size of Table being Registered
        - CFE_TBL_OPT_DEFAULT, -Deflt: Single Buff. and Loadable
        - &QQ_MyTableValidationFunc); -Ptr to table validation function
- Acquiring Table Data
    - Obtain a pointer to the start of the data within the Table
    - Get current Address of MyTable Data
        - CFE_TBL_GetAddress
            - &MyTblePtr - Addr of ptr in which table addr will be ret

            - MyTableHandle -Table Handle from CFE_TBL_Regester call
        - CFE_TBL_GetAddresses
            - Collection of tables
            - Using an array of Table Handles as an input
            - Problem - if there is an error code in one table it will return an error
- Releasing Table Data
    - Once an App is done accessing it Table Data, it must release the pointers
    - CFE_TBL_ReleaseAddress(
        - MyTableHandler);
    - CFE_TBL_ReleaseAddresses
        - For multiple table release

- Managing a Table
    - Allow the operators an opportunity to validate table content and to change the content
    - APIs:
        - CFE_TBL_GetStatus,
        - CFE_TBL_Validate,
        - CFE_TBL_Update
        - CFE_TBL_Manage
- Validating Table Data
    - Validate the table contents prior to activating the table for usage
    - In response to a table validation request the Table Services will
        - Compute a data integrity value
        - Transmit the results to the operator for visual inspection
- Loading/Updating Table Data
    - Applications have control of Table updates within its execution cycle
    - Load MyTable with Data from **Memory**
    - CFE_TBL_Load
        - MyTableHandle, -Table Handle
        - CFE_TBL_SRC_ADDR, -Identify following ptr as memory ptr
        - &MyTblInitData; -Pointer to data to be loaded
    - Load Table with data from a **File**
    - CFE_TBL_Load(
        - MyTableHandle, -Table Handle
        - CFE_TBL_SRC_FILE, -Identify following ptr as string ptr
        - “MyTableInitFile.dat”); -Character string containing filename

    - Update
        - When the Inactive Table Image has been modified by external party (i.e. ground command or stored command processor)
        - CFE_TBL_Update
- Simplifying Table Management
    - Will return status (either Success or Updated)
    - CFE_TBL_Manage
- Creating Table Image Files using elf2cfetble Utility
    - Convert an object file in the ELF format into a cFE Table file format
    - Table Image file has two header
        - cFE Standard File Header
        - Table Image Secondary Header
    - Are required to successfully load an image from a file
    - Ehf2cfetbl utility files:
        - Cfe_tble_filedef.h
        - SampleTbleImg.c
        - ELF_Structures.h (source file)
        - Elf2cfetbl.c (source file)
    - Creating an Executable
        - Only need the two source files
        - Gcc -o elf2cfetble elf2cfetbl.c
- Preparing a Source File for elf2cfetbl
    - CFE_TBL_FILEDEF(
        - ObjName, -Name of the variable previously id
        - TblName, - The FULL name of the table
        - Desc, - 32 character or less description of table image
        - Filename - Default filename expecting to load upon initialization
    - *See SampleTblImg.c for example*
- Elf2cfetbl Utility Command Line Options
    - elf2cfetbl [-tTblName] [-d"Description"] [-h] [-v] [-V] [-s#] [-p#] [-eYYYY:MM:DD:hh:mm:ss] SrcFilename [DestDirectory]
- Converting COFF Object Files into ELF Object Files
    - objcopy -O elf32-little MyObjFilenameInCoffFormat.o MyObjFilenameInElfFormat.o

File Service Interface

- File Service API is concerned mostly with handling of the cFE File Service standard file header
- File - Collection of data

    - Can be a text document, an executable program, or a collection of data from an instrument.
    - Other attributes - name, location, date, size, owner, and access permissions
- Standard File Header
    - typedef struct
    - {
        - uint32 ContentType; - /* Identifies the content type (magic #=’cFE1’) */
        - uint32 SubType; - /* Type of ContentType, if necessary */
        - uint32 Length; - /* Length of this primary header */
        - uint32 SpacecraftID; - /* Spacecraft that generated the file */
        - uint32 ProcessorID; - /* Processor that generated the file */
        - uint32 ApplicationID; - /* Application that generated the file */
        - uint32 TimeSeconds; - /* File creation timestamp (seconds) */
        - uint32 TimeSubSeconds; -/* File creation timestamp (sub-seconds) */
        - char Description[32]; - /* File description */
    - } CFE_FS_Header_t;
- Accessing and Modifying the Standard File Header
    - CFE_FS_ReadHeader
        - Reads the contents of the header of a specified file and returns it into a given data structure
    - CFE_FS_WriteHeader
        - Populates the structure with the SPacecraftIdD, ProcessorID, ApplicationID, TimeSeconds, and TimeSubsecs
    - CFE_FS_UpdateHeaderTime
        - Update header to output file with current time
    - CFE_FS_SetHeaderTime
        - Update header to output file with time of raw sensor data packet

Time Service Interface

- Allows applications to access, convert, and manipulate the current time
- Time Format
    - typedef struct {
        - uint32 Seconds; /* Number of seconds */
        - uint32 Subseconds; /* Number of 2^(-32) subseconds */
    - } CFE_TIME_SysTime_t;
    -
- Time Values
    - Epoch - mission's time reference to which a derived number of seconds is added
        - An absolute time reference that remains fixed.
        - Should NOT be changed during the life of a mission
    - **Mission Elapsed TIme (MET)**
        - The number of seconds since an arbitrary epoch and is maintained by an on-board oscillator.
        - The raw source of time on the spacecraft.
        - Maintained in a hardware register
        - Running count of clock ticks since the hardware was initialized
        - Based on onboard oscillator
    - **Spacecraft Time Correlation Factor (STCF)**
        - A numeric value used to correlate the MET with the Mission Epoch to obtain the current time.
        - Can be updated with a delta time that is applied once or
        - continuously
    - **International Atomic TIme (TAI)**
        - TAI = MET + STCF
        - Based on highly precise atomic clock
        - **It should be noted that the time referred to by the cFE as TAI is only truly TAI when the chosen epoch is the TAI epoch (00:00:00 January 1, 1958).**
    - Coordinated Universal Time (UTC)
        - UTC = TAI - Leap Seconds
    - Universal Time(UT)
        - Based on the Earth's rotation
    - Universal Time Correlation Factor (UTCF)
        - UTC = Epoch + MET + UTCF
- Time Functions
    - CFE_TIME_GetTime
        - Provides the current spacecraft time

        - Relative to the mission specific epoch time and may be either TAI or UTC
    - CFE_TIME_GetUTC
        - The spacecraft time relative to the Mission Epoch with inclusion of Leap Seconds
    - CFE_TIME_GetTAI
        - Provides spacecraft time since the Mission Epoch and always excludes any Lap Seconds
    - CFE_TIME_Print
        - Print to a string
        - yyyy-ddd-hh:mm:ss.xxxxx\0
- Time Conversion Functions
    - CFE_TIME_Sub2MicroSecs
        - Converts the 32-bit integer subseconds value to an integral number of microseconds range of 0 to 999,999