Application Design

August 2025

Objectives & Approach

- **Objectives**
  - Teach the importance of explicitly designing applications versus jumping into coding and working out a design as you go
  - Compare NASA's procedural app design style with Basecamp's object-based design style to highlight their different quality attributes
  - Introduce Basecamp's object-based design principles that can be implemented in either C or C++

- **Approach**
  - The concept of flowing mission functional requirements to the flight software susbsytem and then to applications is used to provide a context for implementing functional requirements
  - The File Manager app is used as a case study to compare NASA's FM procedural design with Basecamp's FILE_MGR object-based design

Audience & Prerequisites

- **Intended audience**
  - Software engineers developing applications for a cFS based system
- **Prerequisites**
  - Familiarity with software engineering is helpful but not necessary
  - Familiarity with the cFS Framework helpful but not necessary
- **Notes**
  - File Manager is used as a case study because a Basecamp app had been created to implement the same functional requirements
  - This module is related to the systems engineering module that shows how an entire mission is decomposed into applications
    - This module addresses application detailed design and the system engineering module covers application roles within the whole system and strategies for application control and data flows
  - Software development metholodolgies is beyond the scope of this module

Outline

1. Design Process Overview
2. Procedural File Manager Design
3. Object-based File Manger Design
4. Conclusion

Appendix A: FM Requirements

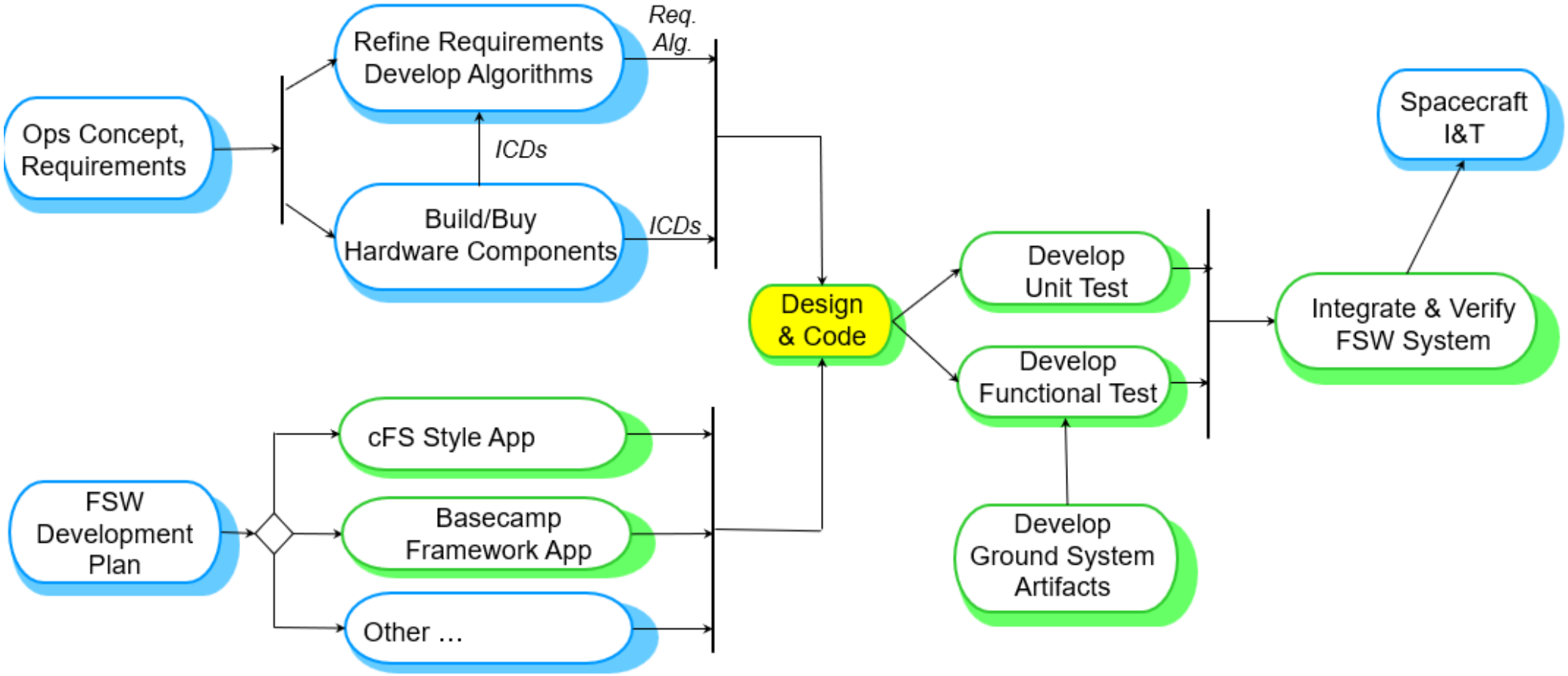
Design Process Overview

Why Design Applications?

- **Mission FSW application design should be an explicit intentional activity that is part of a larger systems engineering effort**
- **Too often developers want to start coding as soon as they have a conceptual idea of what an app should do and work the design out in their head as they code**
- **FSW's role is to aggregate components into a unified system that can be remotely operated which is a systems engineering endeavor**
  - This impacts the allocation of requirements to apps (covered in systems module) and how functional requirements are implemented within an app (covered in this module)
- **I've never heard complaints about time spent designing an app before jumping into coding, even if it's just a few hours at a white board**
  - I have heard complaints about having to formally document a design!
  - I personally have regretted refactoring code that could have been avoided with better upfront designing

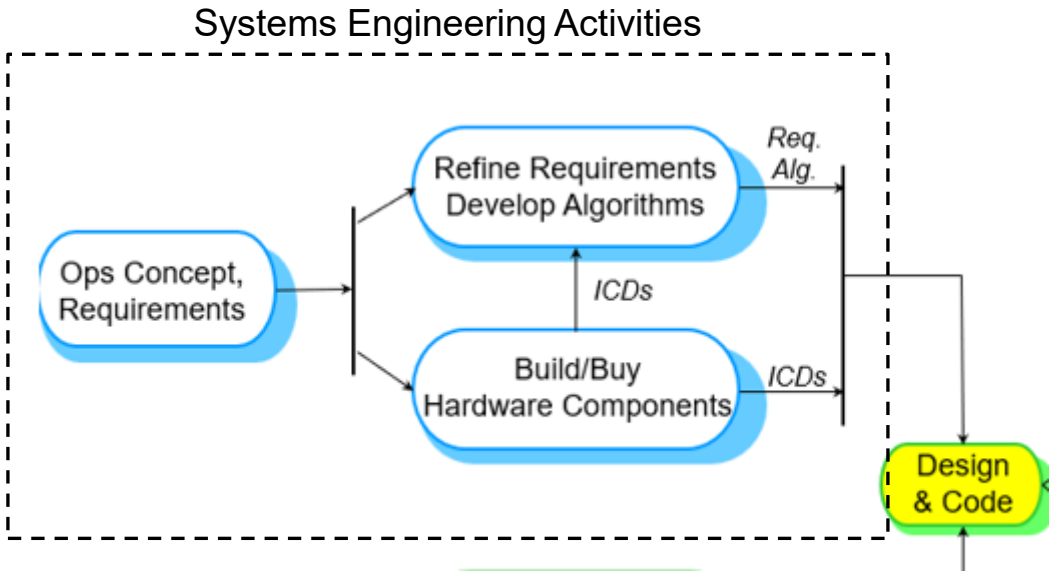
Mission App Development Process (1 of 2)

- **Designing mission applications is an activity that is part of a much more complex process summarized below**



Mission App Development Process (2 of 2)

- **Determining which applications should even be written is the result of systems engineering**



- **This diagram is overly simplified since it flows sequentially**
- **In practice, systems engineering is an iterative process that involves trades, build vs buy decisions, prototyping, etc.****

**Covered in more detail in the Systems Engineering module

Systems Engineering in Practice (1 of 2)

- **File Manager may seem like an "obvious" app but let's consider what may happen in practice for a new mission before a team even decides whether they want the cFS much less a File Manager app!**

- **Initially the spacecraft engineering team may be considering many coupled subsystem requirements and designs in parallel**
  - What processor card should we use?
  - What parts of the power, weight and size budgets can be allocated to the processor card?
  - What components need to interface to the card and which I/O interfaces are supported?
  - What operating systems and development tools are available for the candidate processor cards?
  - Do we want to consider a flight software framework like cFS or F Prime?
  - What resources do these frameworks require?
  - Do we have inhouse expertise for the candidate operating systems and frameworks?
  - Etc.

Systems Engineering in Practice (2 of 2)

- **Systems engineering requires balancing technical, business and user goals****
  - For example, if a company has a business goal to develop multiple spacecraft then a FSW framework may be desirable even if the organization doesn't currently have inhouse expertise
  - In this example longterm business goals influenced the current mission's technical solution

- **Technically systems engineering is a combination of top-down analysis, bottoms-up synthesis and cross-cutting dependencies like power budgets**
  - Each component selection that interfaces with the processor card and requires FSW support levies requirememts on the FSW typically defined in an Interface Document or component user's guide
  - The component's design becomes part of the FSW requirements

**Covered in more detail in the Systems Engineering module

File Manager Requirements (1 of 4)

- **At this point we'll assume that the systems engineering process has determined that the cFS will be used for the mission which means a file system is required**
  - The cFS is a solution that "pushes up" a file system requirement
  - If the cFS were not chosen then having a file system may not be a mission requirement although one may be considered a solution to an onboard data management problem

- **NASA decomposes mission requirements into levels, for example**
  1. Mission
  2. Spacecraft
  3. Subsystem
  4. Component

- **Higher numbered requirement levels are traced to lower numbered requirements**

- **The formality of documenting decomposed requirements is an organizational decision**

File Manager Requirements (2 of 4)

These requirements are paraphrased from an actual mission:

- **Level 1: Mission**
  - Spacecraft X shall implement the CCSDS File Delivery Protocol (CFDP) controlling space-ground file transmissions.
  - Mission ABC operations shall electronically deliver science and housekeeping data files to the Organization ABC as required to satisfy science product latency requirements.

- **Level 2: Spacecraft**
  - Spacecraft X shall provide the capability to perform file and directory management operations by ground command.

File Manager Requirements (3 of 4)

- **Level 3: FSW Subsystem**
  - The flight software shall use the Core Flight System File Manager (FM) application with the configurations defined in the "fm_mission_cfg.h" and "fm_platform_cfg.h" files.

- **Level 4: Component - File Manager**
  - The File Manager app shall perform the following file management and maintenance functions, as specified by ground command:
    - File deletion for a single file, specified by file name
    - Rename a single file from current name to target name
    - Copy a single file, from current location to target location
    - Move a single file, from current location to target location
    - Create directory
    - Delete directory
    - Dump Directory file listing to file

File Manager Requirements (4 of 4)

- **The goal of using actual mission requirements is not to imply they are a gold standard of practice**
  - These requirements illustrate real life gets messy
- **No level 1 requirement states a file system is required, however**
  - The CFDP requirement implies an onboard file system is present
  - The ground requirement to deliver files to Organization ABC could be accomplished by
- **The level 4 requirements are unnecessary because the reuable FM app comes with requirements and a unit test ****
  - What's important to capture at this level is whether any mission requirements can't be accomodated by FM''s current requirements or configuration settings

**Covered in more detail in the Verification and Validation module

Real World Case Study: Devil in the Details

- See notes

App Design Comparison Approach (1 of 2)

- **The next two sections describe the designs of NASA's File Manager (FM) app and Basecamp's File Manager (FILE_MGR) app, respectively**
- **The file *Move* and *Copy* command designs are analyzed in detail**
- **The Conclusion section compares the two design approaches**
- **Comments on the original NASA FM design are not intended to be critical, but instructional**
  - The NASA app design has a long history rooted in extremely constrained flight environments that evolved from procedural programming design practices
  - Once FM was tested there is little incentive for a project to fund "cosmetic" improvements
  - Refactoring a piece of software has the benefit of seeing the complete picture so patterns and optimizations can be discovered regardless of the technology being used

App Design Comparison Approach (2 of 2)

- **Basecamp's FILE_MGR is a design refactor of NASA's FM**
  - FILE_MGR implements all FM requirements
  - An additional FILE_MGR requirement for a Send Directory command was added
  - The Send Directory command sends multiple Directory Listing telemetry messages until an entire directory listing has been sent
  - FILE_MGR uses Basecamp's application framework and object-based design

FM

Procedural Design

Introduction

- The original FM design is shown with a brief description of how data and functionality was decomposed and allocated to different files

- FILE_MGR v1.0 was a design refactor of NASA's FM
  - Implemented all of FM's requirements
  - Added the *Send Directory Telemetry* command that sends multiple *Directory Listing* telemetry messages until the entire commanded directory has been sent
  - The design was completely changed to use Basecamp's application framework and object-based design

FM Design

FM Global Data

App Cmd Counters
Child Cmd Counters
Cmd Queue Mgmt Data

- Free Space Telemetry Packet
- File Info Telemetry Packet
- Open Files Telemetry Packet
- Housekeeping Telemetry Packet
- Child Cmd Queue

fm_tbl

AcquirePtr()
ReleasePtr()
Validate()

fm_app

Global Data

fm_child

Task, Process, Loop
CopyFile
MoveFile
RenameFile
DeleteFile
DeleteAllFiles
DecompressFile
ConcatFile
FileInfo
CreateDir
DeleteDir
DirListFile
DirListPkt
SetPermissions
DirListFile
SizeTimeMode
SleepStat

fm_cmd

Noop
ResetCounters
CopyFile
MoveFile
RenameFile
DeleteFile
DeleteAllFiles
DecompressFile
ConcatFiles
GetFileInfo
GetOenFiles
CreateDir
DeleteDir
GetDirListFile
GetDirListPkt
GetFreeSpace
SetTabletate
SetPermissions

fm_cmd_utils

IsValidCmdPktLength
VerifyOverwirte
GetOpenFilesData
GetFilenameState
VerifyNameValid
VerifyFileClosed
VerifyFileExists
VerifyFileNoExist
VerifyFileNotOpen
VerifyDirExists
VerifyDirNoExist
VerifyChildTask
InvokeChildTask
AppendPathSep

fm_msg.h

Cmd Structures
Tlm Structures
File Records

fm_defs.h

Status

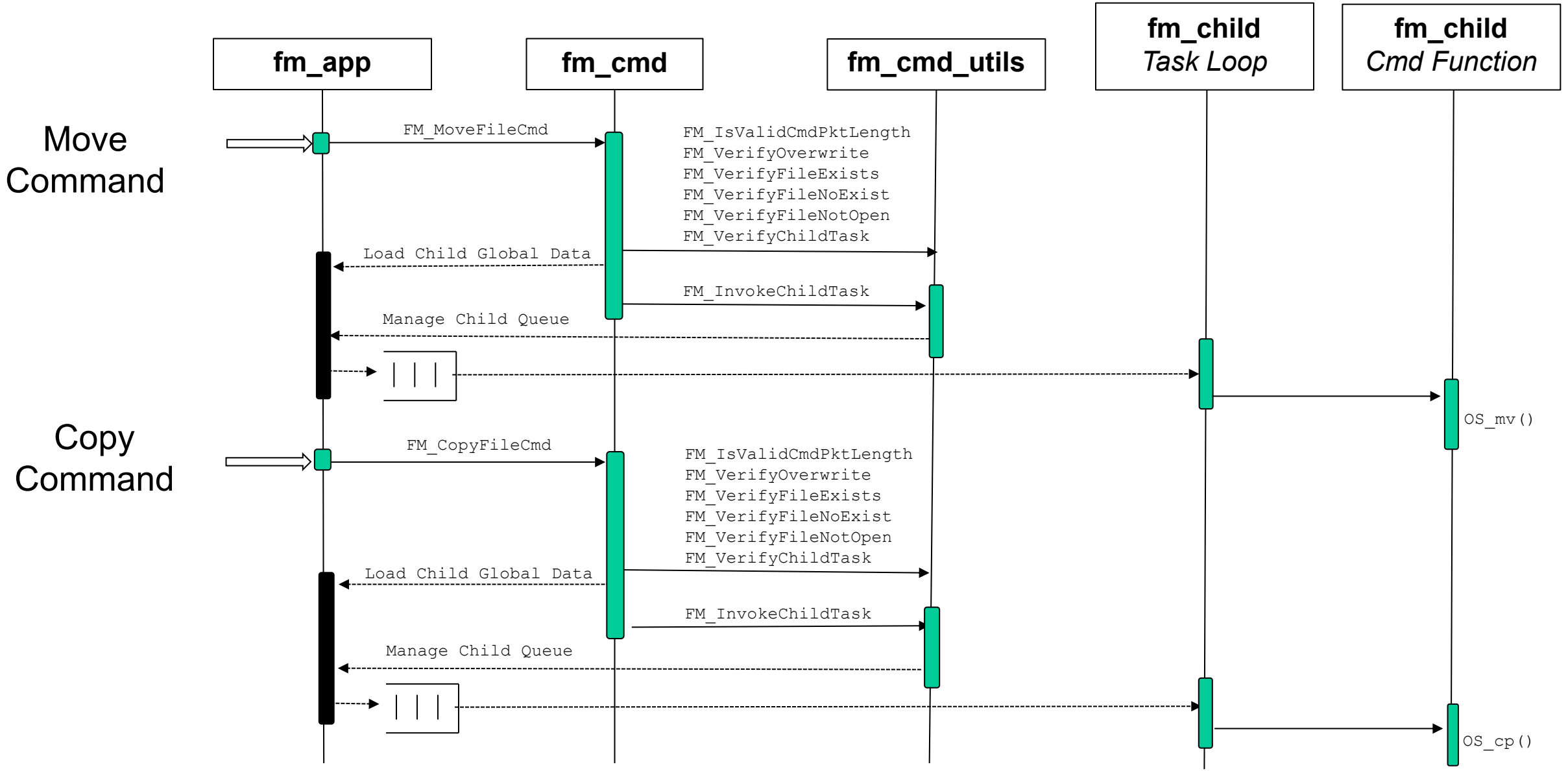
fm_events.h

Event Msg IDs

fm_msgdefs.h

Cmd Function Codes

FM Copy & Move Sequence Diagram



FM Copy & Move Sequence Design Analysis

- **The two functions are nearly identical except for the OS call**
- **What is the sequence (i.e procedure) of activities that must occur to achieve a goal**
- **FM is a rather simplistic app**
  - Service app that responds to commands
- **Verb centric**
- **The grouping of commands into files is based on execution context**
  - All app commands in fm_cmd
  - All child task commands in
  - cohesion

Main Task File Copy (1 of 2)

```
boolean FM_CopyFileCmd(CFE_SB_MsgPtr_t MessagePtr)
{

   FM_CopyFileCmd_t *CmdPtr = (FM_CopyFileCmd_t *) MessagePtr;
   FM_ChildQueueEntry_t *CmdArgs;
   char *CmdText = "Copy File";
   boolean CommandResult;

   /* Verify command packet length */
   CommandResult = FM_IsValidCmdPktLength(MessagePtr, sizeof(FM_CopyFileCmd_t), FM_COPY_PKT_ERR_EID, CmdText);

   /* Verify that overwrite argument is valid */
   if (CommandResult == TRUE) {
      CommandResult = FM_VerifyOverwrite(CmdPtr->Overwrite, FM_COPY_OVR_ERR_EID, CmdText);
   }

   /* Verify that source file exists and is not a directory */
   if (CommandResult == TRUE)  {
      CommandResult = FM_VerifyFileExists(CmdPtr->Source, sizeof(CmdPtr->Source), FM_COPY_SRC_ERR_EID, CmdText);
   }

   /* Verify target filename per the overwrite argument */
   if (CommandResult == TRUE) {
      if (CmdPtr->Overwrite == 0) {
          CommandResult = FM_VerifyFileNoExist(CmdPtr->Target, sizeof(CmdPtr->Target), FM_COPY_TGT_ERR_EID, CmdText);
      }
      else {
          CommandResult = FM_VerifyFileNotOpen(CmdPtr->Target, sizeof(CmdPtr->Target),FM_COPY_TGT_ERR_EID, CmdText);
        }
   }
```

Main Task File Copy (2 of 2)

```
   /* Check for lower priority child task availability */
   if (CommandResult == TRUE) {
      CommandResult = FM_VerifyChildTask(FM_COPY_CHILD_ERR_EID, CmdText);
   }

   /* Prepare command for child task execution */
   if (CommandResult == TRUE) {
      CmdArgs = &FM_GlobalData.ChildQueue[FM_GlobalData.ChildWriteIndex];
      /* Set handshake queue command args */
      CmdArgs->CommandCode = FM_COPY_CC;
      strcpy(CmdArgs->Source1, CmdPtr->Source);
      strcpy(CmdArgs->Target,  CmdPtr->Target);
      /* Invoke lower priority child task */
      FM_InvokeChildTask();
   }

   return(CommandResult);

} /* End of FM_CopyFileCmd() */
```

Main Task File Move (1 of 2)

```
boolean FM_MoveFileCmd(CFE_SB_MsgPtr_t MessagePtr)
{
    FM_MoveFileCmd_t  *CmdPtr = (FM_MoveFileCmd_t *) MessagePtr;
    FM_ChildQueueEntry_t *CmdArgs;    char *CmdText = "Move File";
    boolean CommandResult;

    /* Verify command packet length */
    CommandResult = FM_IsValidCmdPktLength(MessagePtr, sizeof(FM_MoveFileCmd_t), FM_MOVE_PKT_ERR_EID, CmdText);

    /* Verify that overwrite argument is valid */
    if (CommandResult == TRUE)
    {
        CommandResult = FM_VerifyOverwrite(CmdPtr->Overwrite,FM_MOVE_OVR_ERR_EID, CmdText);
    }

    /* Verify that source file exists and not a directory */
    if (CommandResult == TRUE)
    {
        CommandResult = FM_VerifyFileExists(CmdPtr->Source, sizeof(CmdPtr->Source), FM_MOVE_SRC_ERR_EID, CmdText);
    }

    /* Verify target filename per the overwrite argument */
    if (CommandResult == TRUE)
    {
        if (CmdPtr->Overwrite == 0)
        {
            CommandResult = FM_VerifyFileNoExist(CmdPtr->Target, sizeof(CmdPtr->Target), FM_MOVE_TGT_ERR_EID, CmdText);
        }
        else
        {
            CommandResult = FM_VerifyFileNotOpen(CmdPtr->Target, sizeof(CmdPtr->Target), FM_MOVE_TGT_ERR_EID, CmdText);
        }
    }
```

Main Task File Move (2 of 2)

```
    /* Check for lower priority child task availability */
    if (CommandResult == TRUE)
    {
        CommandResult = FM_VerifyChildTask(FM_MOVE_CHILD_ERR_EID, CmdText);
    }

    /* Prepare command for child task execution */
    if (CommandResult == TRUE)
    {
        CmdArgs = &FM_GlobalData.ChildQueue[FM_GlobalData.ChildWriteIndex];
        /* Set handshake queue command args */
        CmdArgs->CommandCode = FM_MOVE_CC;
        strcpy(CmdArgs->Source1, CmdPtr->Source);
        strcpy(CmdArgs->Target,  CmdPtr->Target);
        /* Invoke lower priority child task */
        FM_InvokeChildTask();
    }

    return(CommandResult);

} /* End of FM_MoveFileCmd() */
```

Child Task File Copy

```
void FM_ChildCopyCmd(const FM_ChildQueueEntry_t *CmdArgs)
{
    const char *CmdText   = "Copy File";
    int32       OS_Status = OS_SUCCESS;

    /* Report current child task activity */
    FM_GlobalData.ChildCurrentCC = CmdArgs->CommandCode;

    /* Note the order of the arguments to OS_cp (src,tgt) */
    OS_Status = OS_cp(CmdArgs->Source1, CmdArgs->Target);
    if (OS_Status != OS_SUCCESS)
    {
        FM_GlobalData.ChildCmdErrCounter++;
        /* Send command failure event (error) */
        CFE_EVS_SendEvent(FM_COPY_OS_ERR_EID, CFE_EVS_EventType_ERROR,
                          "%s error: OS_cp failed: result = %d, src = %s, tgt = %s", CmdText, (int)OS_Status,
                          CmdArgs->Source1, CmdArgs->Target);
    }
    else
    {
        FM_GlobalData.ChildCmdCounter++;
        /* Send command completion event (info) */
        CFE_EVS_SendEvent(FM_COPY_CMD_EID, CFE_EVS_EventType_DEBUG, "%s command: src = %s, tgt = %s", CmdText,
                          CmdArgs->Source1, CmdArgs->Target);
    }

    /* Report previous child task activity */
    FM_GlobalData.ChildPreviousCC = CmdArgs->CommandCode;
    FM_GlobalData.ChildCurrentCC  = 0;
}
```

Child Task File Move

```
void FM_ChildMoveCmd(const FM_ChildQueueEntry_t *CmdArgs)
{
    const char *CmdText   = "Move File";
    int32       OS_Status = OS_SUCCESS;

    /* Report current child task activity */
    FM_GlobalData.ChildCurrentCC = CmdArgs->CommandCode;
    OS_Status = OS_mv(CmdArgs->Source1, CmdArgs->Target);
    if (OS_Status != OS_SUCCESS)
    {
        FM_GlobalData.ChildCmdErrCounter++;
        /* Send command failure event (error) */
        CFE_EVS_SendEvent(FM_MOVE_OS_ERR_EID, CFE_EVS_EventType_ERROR,
                          "%s error: OS_mv failed: result = %d, src = %s, tgt = %s", CmdText, (int)OS_Status,
                          CmdArgs->Source1, CmdArgs->Target);
    }
    else
    {
        FM_GlobalData.ChildCmdCounter++;
        /* Send command completion event (info) */
        CFE_EVS_SendEvent(FM_MOVE_CMD_EID, CFE_EVS_EventType_DEBUG, "%s command: src = %s, tgt = %s", CmdText,
                          CmdArgs->Source1, CmdArgs->Target);
    }

    /* Report previous child task activity */
    FM_GlobalData.ChildPreviousCC = CmdArgs->CommandCode;
    FM_GlobalData.ChildCurrentCC  = 0;
}
```

Code Analysis

- Designs are nearly identical. All FM commands are similar.
- Is command packet shuffling necessary?
- Common “file util” functions could have been moved up like BC’s new FW file_util. This is not a function of procedural vs OO
- Combines child manage functions with file move. The functions are serving two purposes
- The child task functionality could be pulled out of each function and put into the child command dispatching
- Not necessarily a result of procedural programming but the mindset of what sequence of steps needs to occur can foster this
- State information is in app’s global data

FILE_MGR

Object-based Design

File Manager Refactor Overview

- **This section presents the results of refactoring NASA's File Manager (FM) app to use osk_c_fw**
- **Motivations for performing this exercise**
  - The initial effort started when OSK's cFE was updated and the latest NASA FM was not compatible with the latest cFE, so I performed local FM updates. As I performed the updates, I starting seeing how the app could benefit from the osk_c_fw that I had been using for OSK apps.
  - In general, I've been looking at all cFS community apps with an eye for how to make them more amenable to an app store concept. At the time of the refactor, FM had 32 compile-time configuration parameters! Configuration parameters add to an app's ease of adoption, so I wanted to assess what needs to be a configuration parameter and when does it need to be defined, compile-time or runtime?
  - Using an app like FM that has long successful history would help valid the osk_c_fw architecture if the refactoring is successful.
- **app_c_fw may be too much of a 'baby step' for the app store concept**
  - This refactor keeps apps in the C programming language domain which may not be a big enough step forward
  - I hope it is still helpful to the community because it does show benefits of an object-based approach in C

Introduction

- Basecamp's File Manager (FILE_MGR) app is derived from NASA's File Manager (FM) app

- FILE_MGR v1.0 was a design refactor of NASA's FM
  - Implemented all of FM's requirements
  - Added the *Send Directory Telemetry* command that sends multiple *Directory Listing* telemetry messages until the entire commanded directory has been sent
  - The design was completely changed to use Basecamp's application framework and object-based design

File Manager Refactor Approach

- **This section does not document every aspects of the refactor**
  - Keep this section relatively short
  - The source code can be analyzed once the basic design structures are described
- **The file copy and move commands are analyzed in detail to show how the original vs the refactored code implement the functions**
- **Some general observations are made with a summary of results**

Object Oriented Analysis and Design

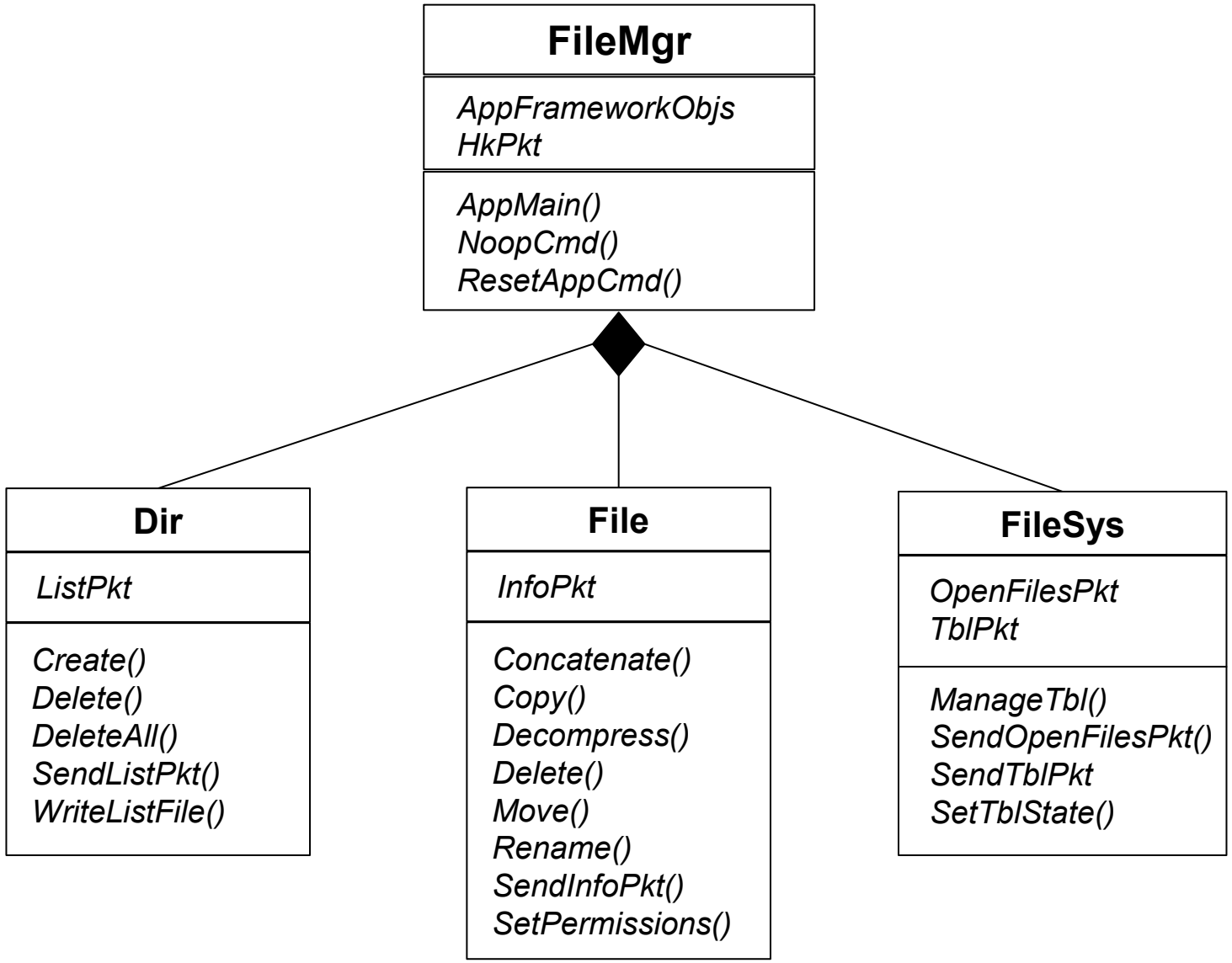
- **Use Class-responsible-collaboration (CRC) analysis**
  - Identified three objects: Directory, File, and File System
    - No state information required but they do have telemetry messages
  - File Utility added to app_c_fw could have been part of file but I wanted it globally accessible. This pokes at another cFS architectural design feature. A file service could be an cFE API and the app would be like other cFE service apps.
- **Release CPU for long process items**

File
- Copy
- Move
- Rename
- Delete
- Delete Internal
- Delete All
- Decompress
- Concatenate
- File Info
- List open files
- Set permissions

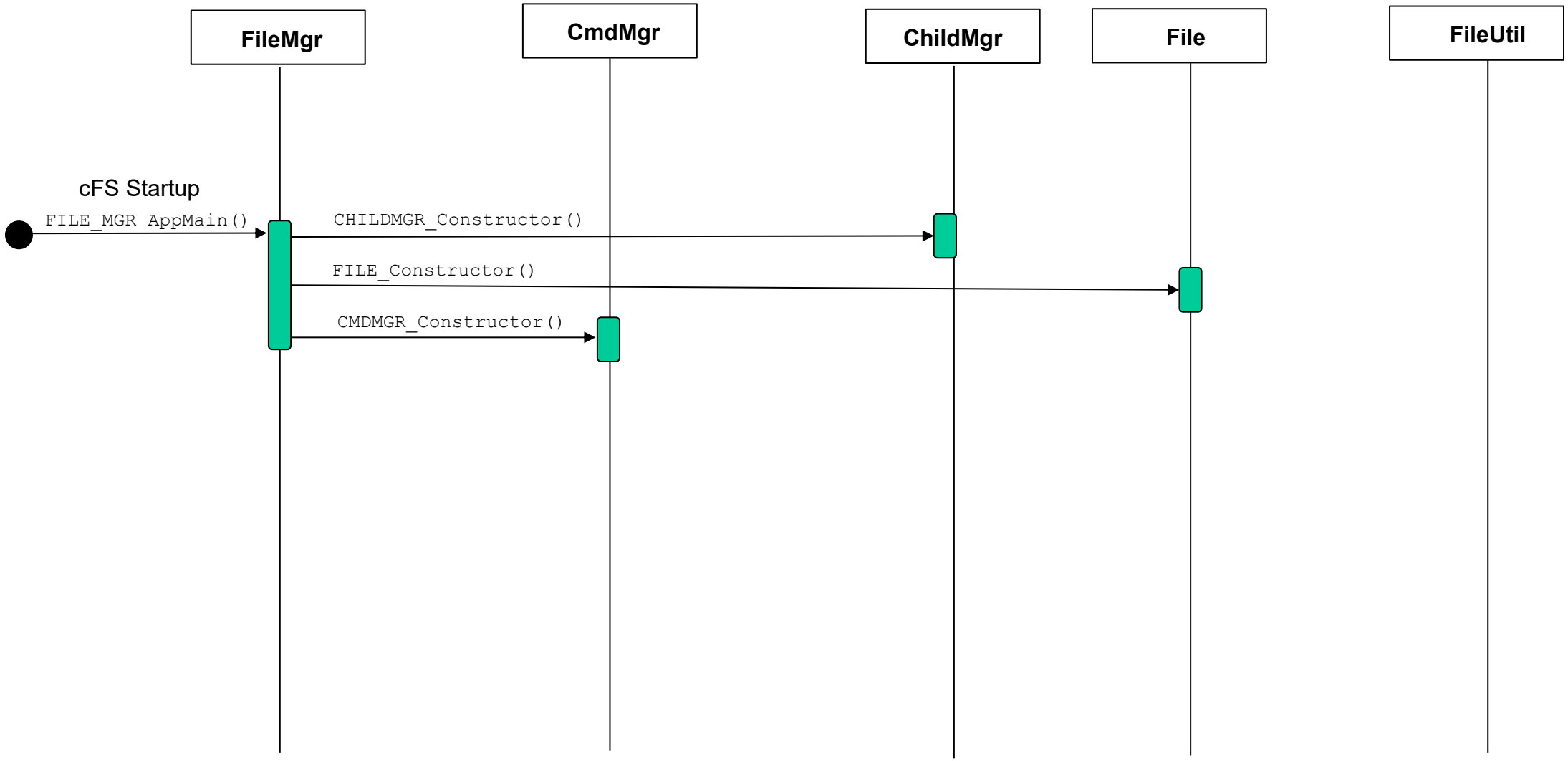
Directory
- Create
- Remove
- Delete
- Send Listing
- Write Listing

Freespace Table
- Get Free Space
- Set Entry state

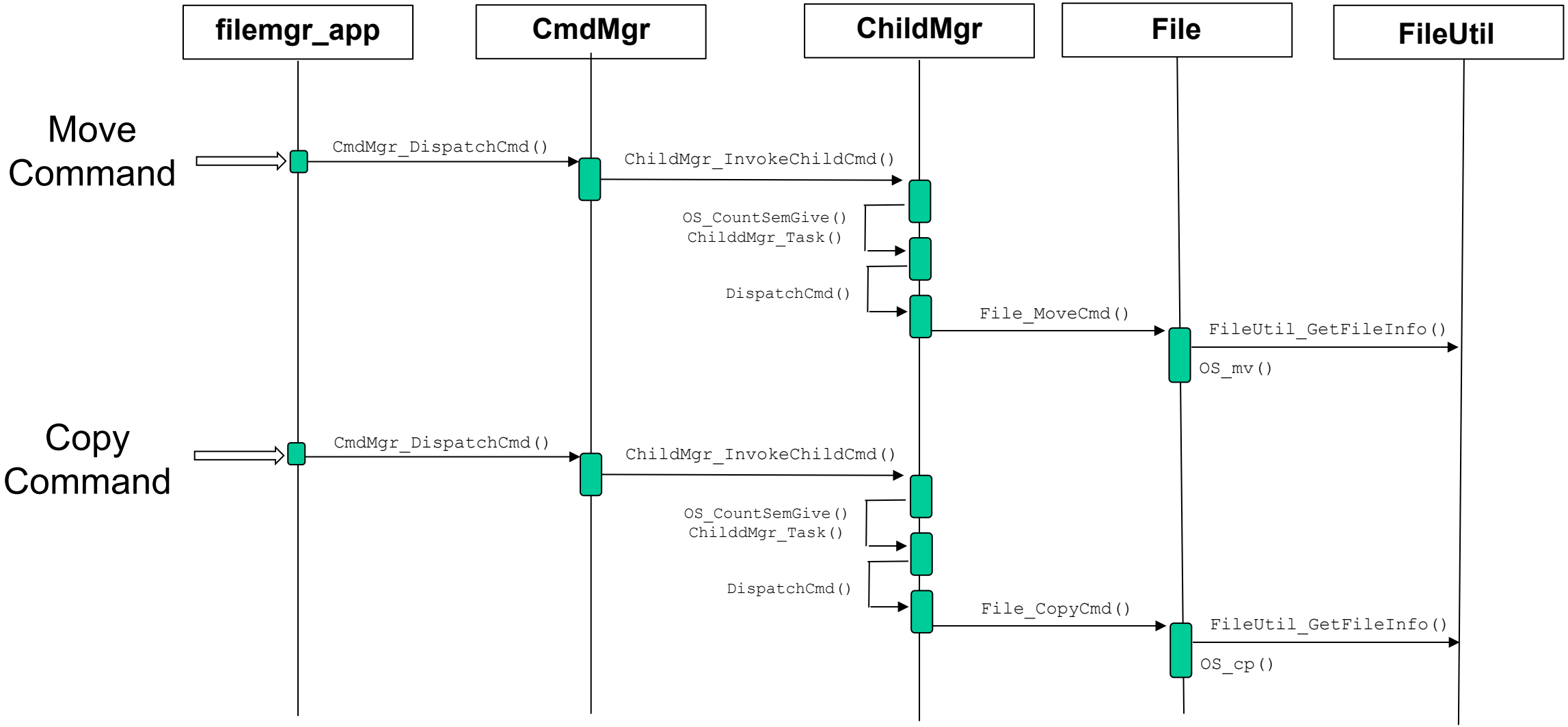
Refactored File Manager Design



Refactored File Manager Sequence Diagram (1 of 2)



Refactored File Manager Sequence Diagram (2 of 2)



Child Task Data Flow

Conclusion

Refactor Observations and Results

- **Moved functionality to framework and need to count those lines of code**
- **Objects suitable for multiple apps migrate to the framework or to a shared library**
  - FileUtils
- **OO 'smells' & design principles**

App	C Src Files	LOC	Platform Compile- Cfg	Init Runtime-Cfg	Notes
FM	20	3038	32	n/a	
FileMgr	12	1681	6	25	App name and table name repeated because binary table requires them during compilation
Framework					

TBD

- **Telemetry Design**
  - HK vs
- **Ops versus design nomenclature**
  - Design command names vs EDS operational names

- **Refactoring utilities is independent of procesure vs OO**
  - Although framework based architecture makes sharing among apps easy
- **Unit testing**
  - Code Complete reference
- **Is number of commands (queue size) a configuration parameter?**
- **Look at parameters in general**
- **FILE_MGR is harder to learn but once you know the framwork should be easy to extend and a maintain. Quality attribute trade offs.**

Appendix A

FM Requirements

FM Requirements (1 of 3)

1	Summary	Custom field (Requirement ID)	Description
2	FM1000	FM1000	Upon receipt of a No-Op command, FM shall increment the Valid Command Counter and generate an event message.
3	FM1001	FM1001	Upon receipt of a Reset Counters command, FM shall reset the following housekeeping variables to a value of zero: a) Valid Command Counter(s) b) Command Rejected Counter(s)
4	FM1002	FM1002	If the computed length of any FM command is not equal to the length contained in the message header, FM shall reject the command
5	FM1003	FM1003	If FM accepts any command as valid, FM shall execute the command, increment the FM Valid Command Counter and issue an event message
6	FM1004	FM1004	If FM rejects any command, FM shall abort the command execution, increment the FM Command Rejected Counter and issue an error event message
7	FM1005	FM1005	If the filename specified in any FM command is not valid, FM shall reject the command
8	FM1006	FM1006	If the directory specified in any FM command is not valid, FM shall reject the command
9	FM1008	FM1008	The CFS FM FSW shall utilize full path specifications having a maximum length of <PLATFORM_DEFINED> characters for all command input arguments requiring a file or pathname.
10	FM1009	FM1009	Upon receipt of the Set Table Entry State Command, FM will set the enable/disable state for the specified entry in the File System Free Space Table.
11	FM1009.1	FM1009.1	If the File System Free Space table has not been loaded FM will reject the command and send an event message
12	FM1009.2	FM1009.2	If the command-specified entry in the File System Free Space Table is invalid FM will reject the command and send an event message
13	FM1009.3	FM1009.3	If the command-specified state is invalid FM will reject the command and send an event message
14	FM1009.4	FM1009.4	If the command-specified entry in the File System Free Space Table is unused FM will reject the command and send an event message
15	FM2002	FM2002	Upon receipt of a File Copy command, FM shall copy the command-specified file to the command-specified destination file
16	FM2002.1	FM2002.1	If the command-specified destination file exists and the command-specified overwrite flag is set to FALSE, FM shall reject the command

FM Requirements (2 of 3)

	Summary	Custom field (Requirement ID)	Description
17	FM2004	FM2004	Upon receipt of a Move command, FM shall move the command-specified file to the command-specified destination file.
18	FM2004.1	FM2004.1	If the command-specified destination file exists and the command-specified overwrite flag is set to FALSE, FM shall reject the command
19	FM2005	FM2005	Upon receipt of a Rename command, FM shall rename the command-specified file to the command-specified destination file.
20	FM2005.1	FM2005.1	If the command-specified destination file exists, FM shall reject the command.
21	FM2007	FM2007	Upon receipt of a Delete All Files command, FM shall delete all the files in the command-specified directory
22	FM2007.1	FM2007.1	If the command-specified directory contains an open file, FM shall not delete the open file.
23	FM2007.1.1	FM2007.1.1	For any open files that are not deleted, FM shall issue one event message.
24	FM2007.2	FM2007.2	If the command-specified directory contains a subdirectory, FM shall not delete the subdirectory
25	FM2007.2.1	FM2007.2.1	For any subdirectories are not deleted, FM shall issue one event message.
26	FM2008	FM2008	Upon receipt of a Delete command, FM shall delete the command-specified file.
27	FM2008.1	FM2008.1	If the command-specified file is open, FM shall reject the command to delete the file.
28	FM2008.2	FM2008.2	If the command-specified file is a directory, FM shall reject the command
29	FM2009	FM2009	Upon receipt of a Decompress command, FM shall decompress the command-specified file to the command-specified file.
30	FM2009.1	FM2009.1	If the command-specified destination file exists, FM shall reject the command
31	FM2010	FM2010	Upon receipt of a Concatenate command, FM shall concatenate the command-specified file with the second command-specified file, copying the result to the command-specified destination file.
32	FM2010.1	FM2010.1	If the command-specified destination file exists, FM shall reject the command

FM Requirements (3 of 3)

	Summary	Custom field (Requirement ID)	Description
33	FM2011	FM2011	Upon receipt of a File Info command, FM shall generate a message containing the following for the command-specified "file:" a) the file size, b) last modification time, c) file status (Open, Closed, or
34	FM2012	FM2012	Upon receipt of a List Open Files command, FM shall generate a message containing: a) the number of open files b) Up to <PLATFORM_DEFINED> names/paths (of each open file) and the applicatic
35	FM2013	FM2013	Upon receipt of a Set File Permissions command, FM shall set the command-specified file's permissions to the command-specified mode.
36	FM2014	FM2014	Upon receipt of a Verify File Signature command, FM shall verify the signature of the command-specified file.
37	FM3000	FM3000	Upon receipt of a Create Directory command, FM shall create the command-specified directory on the command-specified filesystem
38	FM3001	FM3001	Upon receipt of a Delete command, FM shall remove the command-specified directory from the command-specified filesystem.
39	FM3001.1	FM3001.1	If the specified directory contains at least one file or subdirectory, the command shall be rejected
40	FM3002	FM3002	Upon receipt of a Directory Listing To File command, FM shall write the contents of the command-specified directory on any of the on-board file systems to the command-specified file. The followin
41	FM3002.1	FM3002.1	FM shall issue an event message that reports: a) The number of filenames written to the specified file b) The total number of files in the directory c) The command-specified file's filename
42	FM3002.2	FM3002.2	FM shall use the <PLATFORM_DEFINED> default filename if a file is not specified
43	FM3002.3	FM3002.3	If the command-specified GetSizeTimeMode flag is set to FALSE, FM shall initialize the following values to Zero: b) file size in bytes of each file c) last modification time of each file e) Mode (permissi
44	FM3003	FM3003	Upon receipt of a Directory Listing command, FM shall generate a message containing the following for up to <PLATFORM_DEFINED> consecutive files starting at the command specified offset: a) D
45	FM3003.1	FM3003.1	If the command-specified GetSizeTimeMode flag is set to FALSE, FM shall initialize the following values to Zero: b) file size in bytes of each file c) last modification time of each file e) Mode (permissi
46	FM4000	FM4000	FM shall generate a housekeeping message containing the following: a) Valid Command Counter(s) b) Command Rejected Counter(s) c) Total number of open files
47	FM4001	FM4001	Upon receipt of a Report Device Free Space command, FM shall generate a message containing for each enabled device in the FM device table the amount of available free space.
48	FM5000	FM5000	Upon initialization of the FM Application, FM shall initialize the following data to Zero a) Valid Command Counter b) Command Rejected Counter