

National Aeronautics and Space Administration

Goddard Space Flight Center

­

**core Flight Executive**

**Software**

**Requirements**

**Specification**

Version 5.1

November 20, 2017

**REVISION HISTORY**

Change Summary

|  |  |  |  |
| --- | --- | --- | --- |
| **Section** | **Requirement** | **Change Date** | **Change Summary** |
| All |  |  | Baseline version of Functional Requirements for Section 3.2, 3.3 and 3.6. Functional Requirements for sections 3.1, 3.4, 3.5,and 3.7 will be baselined in a future release of this document |
| 2.3.4 and 3.4 |  | 1/21/05 | Added baseline of Software Bus Requirements. Also added some items to the Glossary in 1.3 |
|  | EVS and ES | 4/14/05 | Re-baselined the cEVS requirements. Draft version of ES requirements – not baselined! |
| All |  | 10/25/06 | Updated for build 4.0. Updated with IV&V comments 7/20/06 |
| 3.7 | None | 2/6/08 | Added MKS requirements output |
| 3.7 | All functional | 11/20/17 | Removed MKS requirements output, replacing with reference to CCB controlled Functional Requirements (.md format published as .pdf) |

[National Aeronautics and Space Administration i](#_Toc499038608)

[Goddard Space Flight Center i](#_Toc499038609)

[Version 5.1 i](#_Toc499038610)

[REVISION HISTORY ii](#_Toc499038611)

[1 INTRODUCTION 1](#_Toc499038612)

[1.1 Purpose 1](#_Toc499038613)

[1.2 Scope 1](#_Toc499038614)

[1.3 Glossary 1](#_Toc499038615)

[1.4 Applicable Documents 4](#_Toc499038616)

[1.5 Document Overview 4](#_Toc499038617)

[2 cFE Overview 6](#_Toc499038618)

[2.1 cFE Context 6](#_Toc499038619)

[2.1.1 System Context 6](#_Toc499038620)

[2.1.2 Single Flight Processor Context 8](#_Toc499038621)

[2.1.3 Multi-Flight Processor Context 8](#_Toc499038622)

[2.1.4 Instrument Flight Context 9](#_Toc499038623)

[2.1.5 Desktop Context 10](#_Toc499038624)

[2.2 cFE Constraints 10](#_Toc499038625)

[**2.2.1** **Operational Interface** 11](#_Toc499038626)

[**2.2.2** **Software Interfaces** 12](#_Toc499038627)

[**2.2.3** **Hardware Interfaces** 15](#_Toc499038628)

[**2.2.4** **cFE Platform Adaptation** 16](#_Toc499038629)

[2.3 cFE Subsystem Requirements 16](#_Toc499038630)

[**2.3.1** **Executive Services** 16](#_Toc499038631)

[2.3.2 **Time Service** 17](#_Toc499038632)

[**2.3.3** **Event Service** 18](#_Toc499038633)

[**2.3.4** **Software Bus** 19](#_Toc499038634)

[**2.3.5** **Files** 20](#_Toc499038635)

[**2.3.6** **Tables** 20](#_Toc499038636)

[2.4 User Characteristics 21](#_Toc499038637)

[2.5 Development Constraints 22](#_Toc499038638)

[3 Functional Requirements 23](#_Toc499038639)

[3.1 Executive Services 23](#_Toc499038640)

[3.2 Time Services 24](#_Toc499038641)

[3.3 Event Service 25](#_Toc499038642)

[3.4 Software Bus 26](#_Toc499038643)

[3.5 File Service 28](#_Toc499038644)

[3.6 Table Service 28](#_Toc499038645)

[3.7 Functional Requirements 30](#_Toc499038646)

[Implementation Tradeoffs 2](#_Toc499038647)

# INTRODUCTION

## Purpose

This document specifies the functional and performance requirements to be satisfied by the [Core Flight Executive](#Core_Flight_Executive) (cFE) software. The requirements in this document will serve as input to the cFE design and validation efforts.

## Scope

The cFE is project-independent flight software (FSW) that provides a runtime environment and a set of services for hosting FSW [Applications](#Application). The cFE defines an [Application Programming Interface](#Application_Programmers_Interface) (API) that remains constant across different hardware and/or operating system platforms. Applications that comply with the cFE API can be reused for multiple missions. True reuse requires the reuse of test procedures which is beyond the scope of this document. An effort has been made to keep the cFE services to a minimum while achieving the goal of providing a complete operational environment for Applications. Minimizing cFE functions and complexity supports the goals of a small reusable and portable flight executive. Reusable utilities and Applications will be maintained in the Code 582 Library.

The cFE will be used on multiple missions. In order to determine what common core services are used across missions, an analysis of recent in-house Goddard Space Flight Center (GSFC) FSW Command and Data Handling (C&DH) requirements and implementations was performed. The heritage analysis is provided in Appendix B for reference. The analysis provided the basis for the cFE requirements. A formal trace from the cFE requirements to the heritage requirements is not provided. Each requirement has a rationale associated with it and in many instances this rationale is derived from the heritage analysis.

When FSW requirements are written for a single mission they are traced to higher level requirement documents. The heritage analysis only partially fulfills the higher level requirement’s role. Section 2.0 defines the cFE context, constraints imposed by the cFE’s interfaces, and the general functionality provided by the cFE. These definitions also serve as high level requirements for the cFE.

Other factors have influenced the lack of a formal requirements trace effort. First, the cFE provides “low level” functions that have traditionally been driven by design rather than by a requirements definition effort. Therefore heritage requirement documents are non-existent for some cFE features. Second, the cFE is being developed for use on multiple future missions and contains features that have not been considered for previous one-time implementation efforts. A final factor is that the cFE’s context goes beyond the flight context. The cFE can operate in a desktop environment. This supports migrating Applications between non-flight and flight environments and software-only FSW desktop simulators.

These requirements are being written at a time when a Code 582 FSW requirements standard has not been defined. Appendix A provides the context in which the cFE requirements are written. It also provides guidelines for requirements style and content.

## Glossary

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Application (APP) | A set of data and functions that is treated as a single entity by the cFE. cFE resources are allocated on a per-Application basis. |
| Application-Defined-Address-Table | Interface to Table Services where the Application provides the address to reference the table. Tables Services does not alloate any buffer space to handle this type of table. This is a dump only table. |
| Application ID | A processor unique reference to an Application. |
| Application Programmer’s  Interface (API) | A set of [routines](http://www.webopedia.com/TERM/A/routine.html), [protocols](http://www.webopedia.com/TERM/A/protocol.html), and tools for building [software applications](http://www.webopedia.com/TERM/A/application.html) |
| Board Support Package (BSP) | A collection of user-provided facilities that interface an OS and the cFE with a specific hardware platform. The BSP is responsible for hardware initialization. |
| Branch Library | Repository for branch assets |
| Code 582 | The Flight Software Branch in NASA’s Goddard Space Flight Center |
| Command | A SB Message defined by the receiving Application. Commands can originate from other onboard Applications or from the ground. |
| Core Flight Executive (cFE) | A runtime environment and a set of services for hosting FSW [Applications](#Application) |
| Critical Data Store | A collection of data that has an associated integrity check-value such as a Cyclic Redundancy Check (CRC). This value can be used to verify that the data content has not changed between two points in time. |
| Cyclic Redundancy Check | A polynomial based method for checking that a data set has remained unchanged from one time period to another. |
| Event Data | Data describing an Event that is supplied to the cFE Event Service. The cFE Includes this data in an Event Message that is sent on the SB. |
| Double-buffered Table | A type of table for which Table Services allocates a dedicated inactive buffer. (see Single-buffered Table) |
| Event Filter | A numerical value (bit mask) used to determine how frequently to output an application Event Message defined by its Event ID (see definition of Event ID below). |
| Event Format Mode | Defines the Event Message Format downlink option: short or long. The short format is used when there’s limited telemetry bandwidth. The long format is used for logging to a Local Event Log and to an Event Message Port. |
| Event ID | A numeric literal used to uniquely name and possibly describe by event type an application event. This value is supplied to the cFE Event Service by applications upon calls to the Event Service send event function. |
| Event Message | A data item used to notify the user and/or an external Application of a significant event. Event Messages include a time-stamp of when the message was generated, a processor unique identifier, and Event Data. An Event Message can either be real-time or playback from a Local Event Log. |
| Event Message Counter | A count of the number of times a particular Event Message has been generated since a Power-on Reset or since the counter was cleared via a Command. The counter does not rollover so a user cannot lose the knowledge that an event had occurred. |
| Event Message Port | A display device that is used to display Event Messages in a test environment. The communications mechanism between the flight processor and the display device is platform defined. |
| Event Type | A classification of an Event Message such as informational, diagnostic, and critical. See the cFE Application Developer’s Guide for the definitions. |
| Executive Services | Pany system control functions such as startup of the cFE Core, startup of cFE Applications, control of cFE Applications, loading cFE shared libraries, logging of resets, logging of system events, device drivers, interrupt handling, and exception handling. |
| Executive Services Exception and Reset Log | Log which is used to store information about any reset that occurs such as power-on reset, processor reset or Application reset. |
| Executive Services System Log | Log which is used to record significant internal events or errors that may not be able to be sent to the ground as event messages. |
| FIFO | First In First Out - A storage device that implies the first entry in is the first entry out. |
| Flywheel | Time client or time server is said to be “fly wheeling” when the tone and/or the time at the tone data packet is not received for a specified number of updates |
| Hardware Platform | The target hardware that hosts the FSW |
| I/O Data | Any data being written to and read from an I/O port. No structure is placed on the data and no distinction as to the type of I/O device. I/O data is defined separately from memory data because it has a separate API and it’s an optional interface of the cFE. |
| Local Event Log | An optional Critical Data Store containing Event Messages that are generated on the same processor on which it resides. One Local Event Log can be defined for each processor. |
| Log | A collection of data that an application stores that provides information to diagnose and debug FSW problems. |
| Memory Data | Any data being written to and read from memory. No structure is placed on the data and no distinction as to the type of memory is made. |
| Message ID | An identifier that uniquely defines an SB message. |
| MsgId-to-Pipe Limit | The maximum number of messages of a particular Message ID allowed on a Pipe at any time. When a MsgId-to-Pipe Limit is exceeded, it is considered an error and is sometimes referred to as a MsgId-to-Pipe Limit error. |
| Network | A connection between subsystems used for communication purposes. |
| Network Queue | A device that stores messages and controls the flow of SB Messages across a Network. |
| Operating System | TBD – Include resource management, files, and networks in definition. |
| Operational Interface | The cFE [Command](#Command) and [Telemetry](#Telemetry) interface used to managed the cFE. |
| Pipe | A FIFO device that is used by Application’s to receive SB Messages. |
| Pipe Depth | The numbers of SB Messages a Pipe is capable of storing. |
| Pipe Overflow | Occurs when an attempt is made to write to a Pipe that is completely full of SB Messages. The number of SB Messages a Pipe can hold is given by the Pipe Depth. When a Pipe overflows, it is considered an error and is sometimes referred to as a Pipe Overflow error |
| Processor Reset | The processor resets via the execution of its reset instruction, assertion of its reset pin, or a watchdog timeout. |
| Power-on Reset | The processor initializes from a no-power state to a power-on state. |
| Quality of Service (QoS) | Quality of Service has 2 components, Priority and Reliability. |
| Request | The act of an Application invoking a cFE service that resides on the same processor as the Application. Request protocols are defined in the cFE Application Developer’s Guide. Note they may be implemented as either function calls or SB Message exchanges. |
| Routing Information | Any information required to route SB Messages locally or remotely. |
| Single-buffered Table | A type of table for which Table Services uses an inactive buffer that is a shared resource for all cFE Applicatons. (see Double-buffered Table) |
| Software Bus | An inter-Application message-based communications system |
| SB Message | A message that is sent or received on the software bus. |
| Subscribe | The act of requesting future instances of an SB Message to be sent on a particular Pipe. A valid subscription alters the SB Routing Information. |
| Table Validation Function | A function defined by the cFE Application which valids the contents of the table. |
| Telemetry | A SB Message defined by the sending Application that contains information regarding the state of the Application or the state of devices interfaced to the Application. |
| Time Update | Receipt of a valid tone and time at the tone message |
| Unsubscribe | To request that an SB Message no longer be routed to a particular Pipe. Properly unsubscribing to an SB Message alters the SB Routing Information. |
| User | Anyone who interacts with the cFE in its operational state. A user can be a FSW developer, a FSW tester, a spacecraft tester, a spacecraft operator, or a FSW maintainer. |

## Applicable Documents

The following documents are relevant to the cFE requirements definition:

1. cFE Development Standards Guide
2. cFE User’s Guide, *Identifier TBD*.
3. cFE Application Developer’s Guide, *Identifier TBD*.
4. IEEE Recommended Practices for Software Requirements Specification, *IEEE Std 830-1998*

## Document Overview

Section 2 provides an overall description of the cFE. It defines the cFE context, interfaces, and high level functions. Section 3 defines the detailed functional requirements. A unique number is assigned to each requirement for requirements traceability. Appendix A defines the cFE requirements context from a process perspective. This includes how the requirements are going to be used and by whom. Appendix A provides cFE requirement guidelines to promote requirements consistency and clarity. Appendix B contains the results of the heritage analysis. A formal trace between the heritage requirements and the cFE has not been not performed. The analysis is provided as an informal narrative and supplied in an Appendix for easy reference. Appendix C provides cFE usage scenarios. These scenarios provide context for the information in Sections 2 and 3.

Structured analysis data flow notation is used for the top-level cFE context and for each requirements subsection detailed analysis in section 3. Since the cFE allows platform-specific deployment options the data flow notation has been extended to include dashed lines that indicate an item is optional.

Additional symbols are needed for architectural context diagrams and interface descriptions. In these diagrams a square represents a hardware system/subsystem and a square with rounded corners represents a software system/subsystem. Hardware and software systems can be nested. In some instances the outermost square is dashed. This is for readability and does not indicate an optional component. Table 2-1 summarizes the symbols used for diagrams.



Table 1-1 Diagram Notation

# cFE Overview

This section describes the factors that affect the cFE as a product and its requirements. Among these factors are context, constraints, and user characteristics. This section provides the background information for the detailed requirements defined in Section 3. This section does not define requirements whose implementation is verified via built tests. Appendix C defines scenarios that have been used to elicit high level requirements. These scenarios complement the information in this section. In the absence of a parent requirement document, this overview is considered a flight C&DH subsystem requirement document from which detailed C&DH FSW requirements in section 3 are derived.

This section contains the following subsections:

2.1 cFE Context

2.2 cFE Constraints

2.3 cFE Functions

2.4 User Characteristics

2.5 Development Constraints

## cFE Context

This subsection defines the cFE’s context. This context defines system boundaries and interfaces. Since the cFE is a multi-mission general purpose software solution that supports a variety of hardware platforms, multiple contextual views are used to provide a comprehensive understanding of the problem space. These views are complimentary and each view emphasizes a different aspect of the context. The following subsections define the cFE context:

2.1.1 System Context

2.1.2 Single Flight Processor Context

2.1.3 Multiple Flight Processors Context

2.1.4 Instrument Context

2.1.5 Desktop Context

.

### System Context

Figure 2-1 illustrates the cFE system context. The cFE interfaces to five external systems: an [Operating System](#Operating_System) (OS), a [Hardware Platform](#Hardware_Platform) (HP), an [Operational Interface](Operational_Interface) (OI), [Applications](#Application) (APP), and other cFE-based systems. The constraints imposed upon the cFE by the interfaces and the constraints imposed upon the interfaces by the cFE must be well understood. These constraints are described in Section 2.2. Section 2.3 summarizes the functions provided by the shaded cFE process.



Figure 2-1 cFE System Context

Figure 2-2 shows a layered view of the interfaces. This view is useful for showing the hierarchical relationships of the interfaces.



Figure 2-2 Layer cFE Context

At the top level, the OI interfaces with the cFE via Commands and Telemetry. As discussed in the next section, the cFE uses SB Messages for Command and Telemetry communications. The OI can interact with the cFE via Applications (the solid line between the OI and Applications layer) or an Application can perform OI activities directly as indicated by the dashed lines between the OI and the Applications layer. For example, a command ingest application can serve as the Command interface for a remote operator and a stored command Application can serve as an Application performing OI duties. The OI and the API are shown as separate data sources/sink in Figure 2-1 because they are logically separate interfaces even though both interfaces can be physically implemented by a single Application. The OI is defined in the cFE User’s Guide (Reference 4).

A primary goal of the cFE is to define a standard project-independent API that provides the essential functions required by flight Applications. This infrastructure enables the development of reusable Applications that can be maintained in the Code 582 Reuse Library. The cFE-to-Application interface is defined in the cFE Application Developer’s Guide (reference 5). Applications do not have a direct interface to the Platform Environment layer. The cFE abstracts these interfaces by providing a platform-independent Executive Services interface. Careful consideration was given as to how much of the OS and Hardware Platform functionality is exposed to Applications while still maintaining a portable cFE.

The cFE-to-Platform Environment (OS and hardware) and cFE-to-cFE interfaces are defined in the cFE Developers Guide (reference 5) which also serves as an ICD to the supported platforms. Analogous to defining a standard API, a programming interface is defined for the Platform Environment. The goal of standardizing this interface is to ease the cFE’s portability. For each new platform, device drivers need to be written and then saved in the Branch Library. From a business perspective this allows the Branch to propose supported hardware platforms to new projects and to provide better costing information when systems engineers perform hardware trades. The details of the interface to the Platform Environment are beyond the scope of the cFE requirements. However, the constraints imposed by the Platform Environment on the cFE or vice versa are very important to the cFE requirements. For example, the cFE may require byte-addressable volatile memory with a minimum address bus width of 16-bits. This constrains what platforms host the cFE. A project can perform a hardware/software trade study if a cFE constraint may be violated.

### Single Flight Processor Context

Figure 2-3 shows the cFE in a single processor flight context using a traditional ground system as the UI. The ground system interfaces with the flight Applications Command Ingest (CI) and Telemetry Output (TO) and not directly with the cFE. It’s anticipated that “essential” Applications such as CI and TO will be entered into the Branch Library and checked out on a per-project need basis. Spacecraft operators send Commands and Files to the cFE and receive Files, Events, and Telemetry from the cFE. Note that this context is relative to the cFE and does not show ground communications with other Applications. For example, a typical spacecraft has a Stored Command (SC) Application that receives stored command loads from the ground and sends stored command dumps to the ground.

**Ground**

**System**

(User Interface)

Flight System

cFE

App

CI

TO

Commands

Events

Telemetry

FT

files

Figure 2-3 cFE Single Flight Processor Context

### Multi-Flight Processor Context

Figure 2-4 shows a multi-flight processor cFE-based architecture. From a UI perspective, it’s similar to the single processor context in that the ground interfaces with flight Applications that perform the CI and TO functions. These Applications do not have to reside on the same processor as shown in the figure. However, the ground does have to be aware of the multi-flight processor environment because the cFE supports redundant processors and redundant Applications on different processors. The cFE Application Developer’s Guide defines the namespace strategy for managing redundancy.

**Ground**

**System**

(User Interface)

Flight Processor 1

cFE

App

TO

Commands

Events

Telemetry



Flight Processor 2

cFE

App

Flight Processor N

cFE

App

Hardware Bus

Flight System

FT

CI

files

Figure 2-4 cFE Multi-Flight Processor Context

In a multi-processor environment, the cFE supports Applications sending and receiving SB Messages across processors. This requires the cFE to have an interface between its platform-specific instantiations as indicated by the double arrows between the cFE components. The details of the cFE’s multi-processor support are analyzed in the Product Constraints section.

### Instrument Flight Context

Figure 2-5 shows a cFE-based platform that does not host the UI Applications. This situation would most likely occur when a cFE-based instrument is developed for an out-of-house spacecraft bus. This context does not drive the cFE requirements because the nature of the spacecraft interface cannot be anticipated and custom software would need to be written to adapt the cFE-based platform to the spacecraft system. Using an industry communications standard as the cFE SB Message protocol may reduce the risk of incompatibility but it cannot be eliminated. The context is shown for completeness.



Figure 2-5 cFE Instrument Processor Context

### Desktop Context

Figure 2-6 shows a desktop cFE-based architecture. From a cFE functional perspective this context is the same as the flight cFE context (the desktop can be multi-processor as well). Recognizing this similarity is important because this cFE is intended to support desktop simulation of flight systems with transparent Application migration from one environment to the other.

The locality of the user is important from an interface constraint perspective. This is address in the Product Constraint section.



Figure 2-6 cFE Desktop Context

## cFE Constraints

The cFE operates within various constraints and places constraints on other systems due to its multi-platform support. These constraints are defined in the following subsections:

2.2.1 Operational Interfaces

2.2.2 Software Interfaces

2.2.3 Hardware Interfaces

2.2.4 Platform Adaptation

Each section provides a description of the interface followed by a table that identifies potential constraints on either the cFE or a platform hosting the cFE. Each constraint has one or more optimization strategies. Optimization is relative to the entire cFE effort and not just one particular interface constraint. The detailed requirements should embody the strategies adopted by the cFE and the constraint being optimized should serve as the requirement’s rationale. For example, providing a short form of an Event Message is a requirement and the rationale is based on the need to satisfy missions with a limited transmission bandwidth. An optimization strategy may result in the omission of a requirement as well. For example, the cFE SB Messages are not required to be defined using an industry standard based on the optimization strategy presented in the Software Interface subsection. Note that it may be helpful to review the scenarios in Appendix C to provide a context for the interfaces and their constraints.

### **Operational Interface**

This section describes the OI between the cFE and its users. and interface optimization characteristics. A user can be a FSW developer, a FSW tester, a spacecraft tester, a spacecraft operator, or a FSW maintainer. A user may be remote as shown in Figure 2-4 or local as shown in Figure 2-6. A remote user presents the most constraining case so it will be analyzed in this section. Figure 2-7 shows a remote user operating a spacecraft.



Figure 2-7 Remote cFE Operational Interface

Onboard, the cFE exchanges SB messages with ground interface Applications. These flight Applications manage the packaging and transmission of the SB Messages with a ground system. In Figure 2-7 the ground elements are conceptual and don’t represent an individual ground system. The important aspect of the ground system is that a set of cFE tools must be present in order to provide the interface to the cFE. From an ISO’s Open Systems Interconnection (OSI) Model perspective the cFE ground tools implement the Presentation Layer. The ASIST ground system is being enhanced to support the cFE. Together they can be delivered as a product to a mission.

The data flow in Figure 2-7 has many constraints regarding the cFE. Table 2-1 organizes the constraints as a function of a UI Feature.

|  |  |
| --- | --- |
| **Potential Constraint** | **Optimization Strategy** |
| Transmission Bandwidth | 1. Provide a short form of Event Messages for missions with limited bandwidth. 2. Provide minimal continuous cFE health and status telemetry and provide an interface for telemetry on demand. 3. Allow individual Applications to be loaded as opposed to an entire image. |
| Space-Ground  Transport Protocol | 1. Communication delays, frame lengths, and other transport protocol issues do not impact the cFE requirements. The cFE SB Message maximum length of 64K assumes the transport mechanism can accommodate it. |
| Standards | 1. The cFE SB Message do not need to follow a standard. A flight processing penalty may be incurred due to extra formatting by the ground interface Applications. 2. The cFE uses a File System to minimize the need for custom cFE Ground Tools. |
| Operational Interface  Usability | 1. The user must be able to uniquely identify each onboard Application and its cFE supported elements (Commands, Tables, etc). The cFE Tools must support the cFE namespace protocol defined in the cFE User’s Guide. 2. A spacecraft operator must not need to know memory addresses in order to perform an operation. Such a dependency would greatly increase the complexity of the cFE Ground Tools. The cFE uses a File System as the mechanism for loading Applications. |
| Business | 1. Consistent platform for multiple missions simplifies maintenance. 2. Easier to train new employees. 3. Provides a stable platform for introducing new technologies. |

Table 2-1 Operational Interface Constraints

### **Software Interfaces**

This section identifies the cFE interfaces to other software products and discusses the purpose of the interfacing software as it relates to the cFE. Figure 2-8 shows the cFE software interfaces. During execution the cFE interfaces with a Board Support Package, an OS, a File System, a Network, Applications, and other cFEs. The interface to the cFE Ground tools is content only and does not impact the functional requirements. It is included in the diagram for completeness. For each software interface a table defines the constraints imposed by the cFE on the external software or the constraint imposed by the external software on the cFE. These constraints are organized according to features of the interfaces.



Figure 2-8 cFE Software Interfaces

Board Support Package Interface

A board support package (BSP) is a collection of user-provided facilities that interface an OS and the cFE with a specific hardware platform. Table 2-2 lists the cFE/BSP constraints according to BSP features.

|  |  |
| --- | --- |
| **BSP Interface Features** | **cFE/BSP Constraints** |
| Hardware Initialization | 1. TBD – Does anything need to go here? |
| Software Initialization | 1. Provide an indication to the cFE and Applications as to whether the reset is a Power-on Reset or a Processor Reset. 2. TBD – Is it important what piece of software initiates execution? |

Table 2-2 BSP Interface Constraints

Operating System Interface

The purpose of interfacing to the OS is to provide a runtime environment for Applications. This environment includes a multi-tasking environment and controlled access to shared resources. Table 2-3 lists the OS constraints and their cFE consequences.

|  |  |
| --- | --- |
| **OS Interface Features** | **cFE/OS Constraints** |
| Executable Image  Format | 1. Load cFE from a File System as either object modules or executable images. |
| Tasking | 1. The cFE runs as TBD. 2. Applications run as TBD. |
| Message Queues | 1. The OS does not need to support 2. Maximum depth? 3. Number of queues? 4. Dynamic queue management? |
| Semaphores | 1. Maximum number? 2. Dynamic management? |

Table 2-3 BSP Interface Constraints

Application Interface

The purpose of interfacing to Applications is to provide a standard API and robust runtime environment across multiple missions. Robustness includes minimizing the damage incurred by a misbehaving Application (Memory corruption, resource hogging, etc.). Table 2-4 lists the features of the Application interface and the resulting cFE constraints.

|  |  |
| --- | --- |
| **Application Interface Features** | **cFE/Application Constraints** |
| Construction | 1. The cFE is designed to allow new application code to be written, compiled, and linked without rebuilding the system. |
| Initialization | 1. An Application is aware of the type of reset Power-on or Processor Reset. 2. The order of Application initialization on a processor is not mandated by the cFE. 3. Allocate cFE resources during initialization. |
| Platform Abstraction | 1. Hardware abstraction    1. Different categories of Applications need different level of access. General purpose Application versus System Service Application. 2. OS abstraction    1. What tasking model imposed?    2. Different categories of Applications need different level of access. General purpose Application versus System Service Application. 3. Message-based inter-Application communications. Maximum message size is 64K. 4. Time 5. Files 6. Tables 7. The cFE does not make any assumptions as to the role of a processor: main processor, instrument, or support processor. The cFE provides the same API on all processors. |
| Access to  Operational Interface | 1. Commands 2. Telemetry 3. Events |
| Access to  Resources | 1. Event message filters for cFE generated events can be “exhausted” by a misbehaving Application. 2. A misbehaving Application can’t bring down the system. |
| Management | 1. Add application during runtime. Runtime registration 2. Replace an application 3. No need to unload an Application |
| Redundancy | 1. AP IDs must be unique and applications must subscribe to both the primary and redundant messages. The cFE is not impacted if the backup hardware is cold or hot. The ground can use “derived AP IDs” to display identical TLM packets with different AP IDs from redundant hardware. 2. A namespace must be defined to allow the ground to uniquely identify Applications if it’s duplicated on multiple processors. Task IDs only need to be processor unique; not spacecraft unique. |

Table 2-4 Application Interface Constraints

Network Interface

The purpose of interfacing to a Network is to support a multi-processor flight environment. Table 2-4 lists the Application constraints and their cFE consequences. Table 2-5 lists the features of the Application interface and the resulting cFE constraints.

|  |  |
| --- | --- |
| **Network Interface Features** | **cFE/Network Constraints** |
| TBD | 1. TBD – How do different network architectures impact the cFE or what restrictions does the cFE place on a network? 2. Endian issues. 3. Bound latency 4. reliable transfer 5. Allow applications to perform authentication 6. Synchronous and asynchronous networks 7. The SB supports different network Quality of Service (QoS) options such as retries. TBD – Need to define a SB to network/data link interface. What part of Convergence Layer interface needs to be exposed as part of the SB API? Specify the supported communications environments and their impact on the cFE. 8. The software bus builds a subscription table. The router uses this table to build a routing table. This two step process supports statically defining (during system construction rather than runtime) the subscription table and could also be used for routing configuration following a warm start. 9. The routing tables are periodically (TBD – Is there a frequency requirement. If so what drives it?) updated. If a redundant processor is brought on line then the routing table is rebuild. Duplicate routing tables are not used. |

Table 2-5 Network Interface Constraints

File System Interface

The purpose of interfacing to File System is to TBD. Table 2-6 lists the features of the File System interface and the resulting cFE constraints.

|  |  |
| --- | --- |
| **File System Interface Features** | **cFE/File System Constraints** |
| Time Epoch | 1. How do we address cFE to OS-File time Epoch relationships? |

Table 2-6 File System Interface Constraints

cFE Interfaces

The purpose of interfacing to other cFE instantiations is to support a multi-flight architecture and toe support test configurations. Table 2-7 lists the features of the cFE interface and the resulting cFE constraints.

|  |  |
| --- | --- |
| **cFE Interface Features** | **cFE Constraints** |
| TBD | 1. TBD |

Table 2-7 cFE Interface Constraints

### **Hardware Interfaces**

Specify the logical characteristics of each interface between the software product and the hardware components of the system. This includes configuration characteristics. Cover what devices supported, how they are supported, and the protocols.

|  |  |
| --- | --- |
| **H/W Interface Features** | **cFE/Hardware Interface Constraints** |
| Processor | 1. No cFE floating point dependencies. 2. Requires byte memory access. 3. Can be either little or big endian. 4. Minimum 16-bits. |
| Memory Protection | 1. No cFE dependencies on a hardware memory protection. |
| Volatile Memory | 1. TBD |
| Non-Volatile Memory | 1. TBD |
| I/O Ports | 1. TBD |
| Interrupts | 1. TBD |
| Exceptions | 1. TBD |
| Hardware Timers | 1. TBD |
| Serial Port | 1. TBD |
| Multi-processor | 1. The cFE allows random processor initialization sequences. 2. The cFE does not make any assumptions as to the role of a processor: main processor, instrument, or support processor. |
| Redundant Hardware | 1. TBD – Are there any impacts? |

Table 2-7 cFE Interface Constraints

### **cFE Platform Adaptation**

The cFE is specified as a multi-platform product. Mission-specific features and customization requirements which are application for all platforms are tagged with <MISSION\_DEFINED>. Platform-specific features and customizations requirements are tagged with either “<PLATFORM\_DEFINED>” or “<OPTIONAL>.” Additional nomenclature is used along with the tag to specify a cFE default value for the platform-specific feature: “<PLATFORM\_DEFINED, Default\_Value>”. Reference platforms (single processor and multi-processor architectures) are defined to supply the default cFE configuration (see cFE Deployment Guide). These configurations define the “maximum” cFE deployments such that any refined deployment is a subset of a reference platform. The cFE build tests verify the reference platforms.

The cFE is delivered to a Code 582 FSW development effort as a product. There is no need to integrate the cFE requirements with a mission’s C&DH FSW requirements.

## cFE Subsystem Requirements

This section details the cFE subsystem level requirements. Rationale and heritage information is provided for each requirement. The heritage field acknowledges whether the requirement has heritage as opposed to providing a trace to an itemized heritage analysis item. Traceability between the cFE subsystem level requirements and the detailed functional requirements will be included in the requirements traceability matrix.

### **Executive Services**

| **Requirement** | **Heritage** |
| --- | --- |
| SSR1 The cFE shall provide the capability to perform a startup of the cFE on power-on reset, processor reset and by command  *Rationale: Need to be able to startup-up the cFE in order to startup the cFE Applications.* | Yes |
| SSR2 The cFE shall provide the capability to add a new application without recompiling all of the FSW  *Rationale: Key the cFE is the ability to add new cFE Applications to FSW that is already executing on a processor without having to re-compile the load the entire FSW image.*  . | Part |
| SSR3 The cFE shall provide the capability to control cFE Applications including starting, stopping, suspending and resuming the cFE Applications..  *Rationale: Need to be able to control each cFE Application when add new cFE Applications or debugging currently executing ones.* | Part |
| SSR4 The cFE shall provide the capability to use a flight software patch made to volatile memory.  *Rationale: Need to be able to verify that the flight software patches work before committing them to non-volatile memory. Note that this requirement does not cover the ability to load the patch. This requirement only covers the use of the flight software patch. Flight software patches can be made to the cFE as well as the cFE Applications.* | Yes |
| SSR5 The cFE shall provide the capability to use a flight software patch made to non-volatile memory.  *Rationale: Need to be able to use flight software patches that exist in non-volatile memory. Note that this requirement does not cover the ability to load the patch. This requirement only covers the use of the flight software patch. Flight software patches can be made to the cFE as well as the cFE Applications.* | Yes |
| SSR6 The cFE shall provide the capability to |  |
|  |  |
|  |  |

### **Time Service**

| **Requirement** | **Heritage** |
| --- | --- |
| SSR7 The cFE shall provide a <MISSION\_DEFINED> selection that determines whether the cFE is configured to function as a time server or time client.  *Rationale: Every spacecraft needs a time server and the cFE may be configured to perform that function.* | Part |
| SSR8 The cFE shall provide a <MISSION\_DEFINED> selection of which time format (TAI or UTC) is used as the default time representation.  *Rationale: The most common on-board use of time is to time stamp various data. This definition avoids having to modify each individual location in the code that acquires the time when the default time format selection is changed. Applications that request time in a specific format will not be affected by the choice for default time.* | Part |
| SSR9 The cFE shall compute TAI and UTC as follows:  TAI = MET + STCF  UTC = MET + STCF – Leap Seconds  *Rationale: These are the generally accepted spacecraft methods for calculating TAI and UTC.*  *Note: Additional useful information regarding the definition of TAI and UTC and how they differ may be found at various web-sites, including* <http://wwp.greenwichmeantime.com/info/leap-second.htm> | Yes |
| SSR10 The cFE shall preserve the STCF, Leap Seconds and Time Status values in a <MISSION\_DEFINED> Critical Data Store.  *Rationale: Time is a critical system resource and every effort should be made to preserve it. Correct time is required for many applications (attitude control, stored command, etc.) to run correctly.* | Yes |
| SSR11 The cFE shall provide an interface for other cFE components and Applications to query spacecraft time.  *Rationale: It is a common practice for sub-systems to query the current spacecraft time. Sub-systems use time for a variety of purposes, including science calculations, positional computations, time stamping telemetry packets and the performance of various operations at a specific time or timed intervals.* | Yes |

### **Event Service**

| **Requirement** | **Heritage** |
| --- | --- |
| SSR12 The cFE shall provide an interface for other cFE components and Applications to send Event Messages on the Software Bus and optionally to an [Event Message Port](#Event_Message_Port).  *Rationale: Most, if not all, Small Explorer and Mid-Explorer GSFC missions have provided an asynchronous informational/error message to ground operators. These messages have been very useful during development, on-orbit operations, and maintenance. Historically, the messages have at a minimum included a time stamp and a unique event identifier. Many missions have also provided parameterized messages allowing Applications to provide additional data. The cFE provides ASCII messages in either a short format or long format. The short format supports missions with limit telemetry bandwidth.* | Yes |
| SSR13 The cFE shall classify Event Messages according to their Event Type (diagnostic, informational, critical, etc.)  as defined in the cFE Application Developer’s Guide..  *Rationale: Grouping events supports different classes of users. Supports heritage classification of event messages by event type.*  *Note: Testers should not rely on diagnostic Event Messages for cFE verification.* | Partial |
| SSR14 The cFE shall provide an Event Message filtering mechanism with Commands for enabling/disabling filters.  *Rationale: Filtering Event Messages has been inconsistent across heritage missions. Provide this service as part of the cFE provides a consistent OI and relieves Application developers from having to implement a filtering scheme.* | Partial |
| SSR15 The cFE shall provide an Application interface for registering an application to use cFE event services along with registering that applications filtered events and filter masks.  *Rationale: This supports the cFE goal of easy Application integration.* | None |
| SSR16 <OPTIONAL> The cFE shall provide an interface for preserving event messages within a Local Event Log.  *Rationale: This is an optional feature used to support a multi-processor environment. In times when communications to a processor is lost the Event Messages will be preserved.* | None |
| SSR17 <OPTIONAL> The cFE shall provide commands for playing back the Local Event Log, configuring the logging behavior when the log is full (discard newest message or overwrite the oldest message), and clearing the Local Event Log.  *Rationale: In times when communications to a processor is lost preserved Event Messages may be viewed. Supports configuration and reinitialization of the local event log.* | None |

### **Software Bus**

| **Requirement** | **Heritage** |
| --- | --- |
| SSR18 The cFE shall provide a method for an Application to receive any message from any other Application on the same processor.  *Rationale: Many applications need to obtain information from other applications.* | Yes |
| SSR19 The cFE shall provide a method for an Application on one processor to receive any message from any Application on a different processor.  *Rationale: In a multi-processor environment, applications may need to get messages from applications on other processors.* | Yes |
| SSR20 The cFE shall provide the capability for an Application to specify, at run-time, the messages that it wants to receive.  *Rationale: Run-time subscription facilitates ease of integration of applications. No coordination of header files is required.* | No |
| SSR21 The cFE shall make available to the ground, housekeeping information, routing information, and network information.  *Rationale: Ground operations and flight software maintenance personnel need to be able to get enough information to diagnose an SB problem.* | Yes |
| SSR22 The cFE shall provide transfer topologies which allow 1) one-to-one 2) one-to-many and 3) many-to-one.  *Rationale: 1) one-to-one is needed for any message that is being sent from one application to another application 2) one-to-many is used when a message is subscribed to by more than one application. A heritage example of this is that a telemetry packet from an application such as the Checksum application, would be sent to Telemetry Output AND Data Storage (i.e. TO and DS would subscribe to the Checksum message. 3) many-to-one is used when an application wants many messages to arrive on one of the application’s pipes. An example of this is that an application may want all commands, ground and stored commands, to arrive on one pipe.* | Yes |
| SSR23 The cFE shall provide an Application with the ability to 1) poll a pipe for messages 2) pend on a message with a timeout 3) pend on a message indefinitely.  *Rationale: Applications may want to just check for a message and then continue executing. Applications may also want to be data driven, waiting for a message before continuing execution (either indefinitely or for a specific period of time).* | Yes |
| SSR24 The cFE shall provide an Application with the ability to relinquish any messaging resources.  *Rationale: This requirement is intended to cover both an applications ability to remove a pipe and the ability to unsubscribe to a message. This supports the cFE goal of easy Application integration and FSW maintenance. Provides an application the flexibility to change which messages it wants to receive. It also provides the ability to remove unused resources.* | No |
| SSR25 The cFE shall ensure that an Application’s messaging resource limits are not exceeded.  *Rationale: The cFE needs to make sure that applications do not get flooded with messages (more messages than an application defines that it can handle). In addition, the cFE needs to make sure that certain messages that share a pipe with other message, don’t monopolize that pipe.* | Yes |

### **Files**

TBD

### **Tables**

| **Requirement** | **Heritage** |
| --- | --- |
| SSR26 The cFE shall provide a mechanism by which a contiguous collection of parameters, referred to as a Table, can be referenced via a logical identifier, a Table ID, that is independent of the data’s physical address.  *Rationale: Most, if not all, Small Explorer and Mid-Explorer GSFC missions have provided table services through which ground operations was able to modify software parameters without needing to know the exact location of the parameters in memory.* | Yes |
| SSR27 The cFE shall perform table loads synchronously with cFE Applications to ensure data integrity.  *Rationale: If tables are modified by the cFE independently of the cFE Applications, there is a substantial risk of partially updated tables being used by the cFE Applications. This would likely lead to erroneous behavior and possibly software crashes.* | Yes |
| SSR28 The cFE shall provide an OI for performing complete and partial Table loads..  *Rationale: The point of having Tables is to provide a consistent interface for operations personnel to use to modify flight software parameters. Partial table loads are useful because they help ensure that other parameters are not accidentally modified when only a portion of the table needs to change.* | Yes |
| SSR29 The cFE shall provide an OI for dumping Tables to the ground.  *Rationale: Dumping tables allows operations personnel the opportunity to validate or re-discover the values of the current parameters controlling the flight software.* | Yes |
| SSR30 The cFE shall provide an OI for computing a Data Integrity Check value on the contents of a table.  *Rationale: It will be useful, especially with large tables and low telemetry data rates, to validate the contents of a table using a Data Integrity Check algorithm rather than dumping the contents and verifying them on the ground.* | Yes |
| SSR31 The cFE shall provide an Application interface for registering Tables, their types and default values.  *Rationale: This supports the cFE goal of easy Application integration.* | None |
| SSR32 The cFE shall provide an Application interface for unregistering Tables.  *Rationale: This supports the cFE goal of easy Application integration. It allows for the freeing of resources allocated to an Application when the Application exits.* | None |
| SSR33 The cFE shall support the concept of shared Tables between cFE Applications on a single processor.  *Rationale: On rare occasions, some Tables on previous missions have been so large (e.g. a star catalog) that to duplicate them between applications would have complicated operations and been a burden on memory resources.* | Partial |

## User Characteristics

Describe characteristics of the intended users:

FSW developers

Non-FSW Application developers

Ground operations

FSW Maintenance

## Development Constraints

This section describes items that limit the developer’s options.

All code in ANSI C.

# Functional Requirements

Appendix A provides a detailed analysis as to how and why the requirements are worded as they are.

Introductory material for each requirements subsystem that includes a context diagram and expands on the information provided in the functional summary in section 2.3 is presented below in sections 3.1-3.6.

The Functional Requirements are maintained in MKS and presented as a report from MKS in section 3.7.

## Executive Services

The cFE Executive Services provides many system control functions such as startup of the cFE Core, startup of cFE Applications, control of cFE Applications, loading cFE shared libraries, logging of resets, logging of system events, device drivers, interrupt handling, and exception handling.

The cFE Executive Services provides support for power-on and processor restarts as well as cFE Application restarts. The RTOS and the BSP for each processor handles the initialization of the processor hardware. The ES handles the startup of the cFE and the rest of the cFE Applications. The cFE itself is linked and loaded with the RTOS and BSP as a single static executable in non-volatile memory. On system startup the cFE, RTOS and BSP are copied from non-volatile memory into pre-determined addresses in volatile memory. The RTOS and BSP perform the hardware initialization, after which the cFE performs the rest of the cFE and cFE Application startup.

Key to the cFE is the fact that individual cFE applications can be compiled separately from the cFE and other cFE applications. Depending on the platform, the cFE Applications can be statically or dynamically linked. Static linking means the global load image symbols don’t change when the application is bound to the original image, while dynamic linking means the load address of the application is determined at run time. Each cFE Application has a main task and may have child tasks. The child tasks are threads to the main task. When an application is deleted then the main task and all of the child tasks will be deleted as well.

In addition to loading cFE Applications, ES can load cFE Shared Libraries. cFE Shared Libraries are for sets of related functions that need to be shared between cFE Applications.

ES can also load a hardware device driver into the system. A cFE Device Driver consists of a hardware initialization routine, a low level interrupt routine, and a high level device processing routine.

The cFE Executive services maintains information for each cFE application and its tasks in order to provide a way of cleaning up after an application that may have crashed and to provide information to the ground upon request. One of the key concepts of the cFE ES is the ability to start and stop individual applications and tasks without affecting the rest of the applications.

The cFE Executive Services provides logging capabilities. It maintains a log of ES System Services information both nominal and non-nominal information. The ES Systems Services Log could contain items such as the creation of a new application, an error creating an OS queue, and file not found errors. In addition, the cFE Executive Services maintains an Executive Services Exception and Reset Log. This log contains information on any type of exception or reset that happens.

The cFE Executive Services provides exception and interrupt handling capabilities. Most of the interrupt and exception handling details are handled through the OS Abstraction Layer and BSP, but the interface to these components should remain consistent. Through the ES, OS Abstraction Layer, and BSP, applications can connect to interrupt code.

## Time Services

The cFE Time Services (TIME) is responsible for maintaining spacecraft time and providing a standard interface, as defined in the cFE Application Developer’s Guide, for updating and accessing spacecraft time. TIME is not responsible for ground-flight time correlation or inter-processor time synchronization since these functions are mission-specific. However, TIME is structured with these functions in mind and design decisions should be made to ease portability.

A few concepts related to spacecraft time need to be defined in order to provide a context for TIME requirements.

Spacecraft time is measured as some number of seconds (and fraction of a second) from an “epoch”. The choice of the specific epoch for a particular flight system is <MISSION\_DEFINED>. Although spacecraft time is only meaningful in relation to the epoch, it is not necessary that TIME have any direct knowledge of what the epoch might be.

The primary element of spacecraft time is Mission Elapsed Time (MET). The MET is maintained in a hardware register and is a running count of clock ticks. The MET is not true MET in the sense that it is not the elapsed time since launch or separation, but the elapsed time since the hardware register was initialized.

It should be apparent that spacecraft time must also include some form of correlation factor to bind the MET value to the selected epoch. The moment in time “right now” can then be described as the sum of the spacecraft MET plus a correlation factor that is calculated specifically for that MET and a particular epoch. And since “right now” is ticking away at the same rate that MET is incrementing, the correlation factor need only be calculated once after each MET register reset. Note that the value of the correlation factor is not calculated by TIME, but is instead provided from the ground as part of startup system configuration.

Note also that the MET is not always an accurate timer. Because of hardware inaccuracies due to a variety of causes, the correlation factor may need periodic minor adjustments to compensate for the MET. The adjustments may take the form of a single adjustment made infrequently, or in the case of predictable clock drift, the adjustment may be a small constant change applied every second to the correlation factor.

Another feature of TIME is whether it has been configured to function as a time server or time client. When configured as a time server, TIME acquires the MET from the hardware and provides an operational interface to set and adjust the STCF as previously described. As a time server, TIME also generates a “time at the tone” message for distribution to other spacecraft components. The “time at the tone” message contains the MET, STCF and Leap Seconds correct for the moment in time at which the “tone” occurs. The tone may precede or follow the arrival of the message depending on the particular implementation.

When configured as a time client, TIME receives the “time at the tone” message and uses a local hardware clock only to measure the time elapsed from the tone. Even a poor quality timer can measure lengths of time less than a second with reasonable accuracy. As a time client, TIME does not include an operational interface for modifying the STCF or Leap Seconds values as they are included in the message. However, both the server and client configurations provide the same API services. Applications which use the cFE Time Services are unaware whether TIME has been configured to function as a client or a server. A single spacecraft may include multiple cFE components with, at most, one of them configured as a time server.

## Event Service

The cFE Event Service (EVS) provides Applications a common method for sending and filtering Event Messages. The EVS receives Event Data from other cFE components or from Applications. The EVS filters, time-stamps, and formats the Event Data into an Event Message. This process requires applications to register with the EVS. Registration allows applications to use the EVS for sending events and provides a mechanism for applications to input their filtered events and accompanying filters to the EVS. Although the EVS is designed to operate with multiple filtering algorithms, there is currently only one Binary Filtering Algorithm available to the EVS.

The EVS Binary Filtering Algorithm uses an integer counter along with a Binary Filter Mask to determine if an event message is to be sent or not. The algorithm works by logically ANDing the integer counter with the Binary Filter Mask that is assigned to a particular Event ID. When the result of this operation is zero, the event message (identified by Event ID) is sent; otherwise the event message is filtered. Applications must use the EVS registration function to input all Event IDs and accompanying Binary Filter Masks for events requiring filtering. One Binary Filter Mask is needed per registered Event ID. Once registered, application events that require filtering are assigned additional integer counters for each event that is filtered. These integer counters along with the accompanying Binary Filter Mask are used to control the filtering of the event message (identified by Event ID) using the EVS Binary Filtering Algorithm.

A counter is created for each registered application for keeping track of how many times the application has sent an Event Message through the EVS. Along with the application data supplied to the EVS (message string, Event ID, Event Type), the Processor ID and Application ID from the cFE Executive Services and a timestamp from the cFE Time Service are formatted into the EVS Event Message. The Event Message is sent on the Software Bus (SB) as a Short or Long formatted Message and optionally output as a Long formatted message to one or more Event Message Output Ports and/or the Local Event Log. The Local Event Log’s capacity is mission defined meaning each mission will need to choose a reasonable size for the Local Event Log based on the mission specific processor card memory capacities. Figure 3-1 shows the context of the cFE EVS.

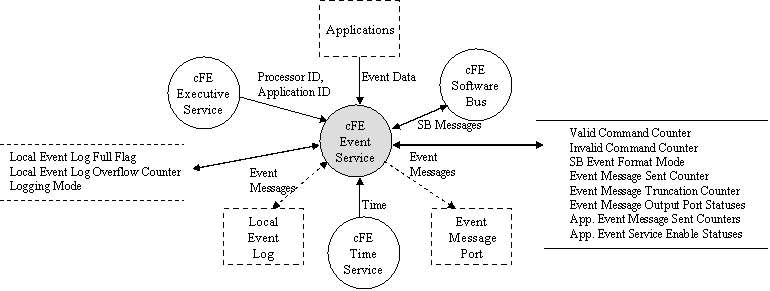


Figure 3-1

Figure 3-2 shows the processing sequence for filtering, formatting, and outputting Event Messages. The intention of this flowchart is to show the order of processing as it applies to functional requirements. The sequence is started when EVS receives Event Data and applies the filtering algorithm for the event message. Filtering includes a check to determine whether the Event Type is enabled or disabled. If the message is filtered the process is terminated. EVS constructs an Event Message for all unfiltered Events. A Long formatted Event Message is sent to all enabled Event Message Output Ports. If the optional Local Event Log is available a Long formatted Event Message is written to the log. Finally, depending on the value of the SB Event Format Mode a Short or Long formatted Event Message is sent on the SB. Note that filtering is applied prior to generating or sending an Event Message to any destination. This prevents a continuous error from flooding any destination and all destinations should have the same sequence of Event Messages.



Figure 3-2

An optional Local Event Log can reside on the same processor as the EVS. This log is used to store and retrieve messages without relying on an external bus. In multi-processor cFE configurations there can be one Local Event Log on each processor. In times when communications to a processor is lost, such as during non-deterministic processor initialization sequences and failure scenarios, Event Messages can be preserved in the Local Event Log. The Local Event Log is not intended to replace the duties of an onboard recorder.

The Local Event Log functional Operator Interface requirements are intentionally simplistic. The ground user can send a command to write all of the Event Messages in the log to a text file for viewing. A selective Event Message playback is not an option. The logging can be configured to behave in one of two ways when the log is full. The event logger can either overwrite the oldest Event Message or preserve the log contents and discard the current Event Message.

## Software Bus

The FSW on all Goddard built spacecraft for at least the past 15 years has used a SB. The SB has proven to be a very effective mechanism for managing Application Command and Telemetry, inter-Application communication, and Application execution. In the past, the SB routing has been statically defined when the system is built and has been limited to a single processor. The cFE is maturing the traditional SB to extend beyond a single processor and to allow dynamic reconfiguration in order to support the cFE goals of easy system integration and runtime reconfigurations.

The Software Bus (SB) is an inter-Application message-based communications system. Figure 3-4.1 shows the context of the software bus with respect to an Application. Each SB Message must have a system unique Message ID. The SB relies on run-time subscriptions to establish communication paths. Any Application can send a SB Message. The SB will route the SB Message to all Application’s that have Subscribed to the SB Message. In a typical scenario, an Application would create a Pipe and Subscribe to messages during it’s initialization process and then call the receive API inside a forever loop to receive the messages.

Processor ID,

Application ID

Time

Event

Data

Routing

Information

Create Pipe

Subscribe to Messages

Send Messages

Receive Messages

cFE

Event

Services

cFE

Time

Service

cFE

Executive

Service

cFE

Software

Bus

Applications

Pipe

Figure 3-4.1 Software Bus Application Context

The SB’s message-based Subscription supports one-to-one, one-to-many, and many-to-one routing configurations. Multiple SB Message ID’s can be routed to a single Pipe. This is commonly done for Applications that need to process ground commands.

The SB also provides different options for an Application to receive messages. An Application can Poll (non-blocking) a Pipe to check if a SB Message is present or it can Pend (blocking) on a Pipe and have its execution suspended until a SB Message arrives. An Application can specify a Pend timeout as well.

The SB also supports inter-processor communication. Figure 3-4.2 shows the context of the SB in a multi-processor configuration. From an Application’s perspective a multi-processor context is transparent. From the SB perspective it must manage routing messages between processors and handle situations such as a processor resets and redundant processors. In addition, many inter-processor communication packages provide services such as prioritized messages and quality of service parameters. In keeping with cFE layered architecture and cFE Application portability goals, the SB must provide access to these services in a platform-independent manner.

Processor A

Network

Queue

Processor B

Network

Queue

Processor C

Network 1

Network 2

Software

Bus

Software

Bus

Software

Bus

Routing

Information

Routing

Information

Routing

Information

Figure 3-4.2 Software Bus Multi-processor Context

In Figure 3-4.2 the double lined Command and Telemetry data flows indicate that an ICD (in this case the CFE Application Developers Guide) governs the interface. This ICD describes how routing information is established and maintained. The Network Queue is used to buffer SB Messages that need to go off the processor board (“off card messages”). SB Message priorities are used to determine the order of transmission.

The SB also supports bridging messages across two distinct networks. In the diagram above, if a packet needs to be routed from Processor A to Processor C, the subscription process sets up the routing on all processors. When the message is sent, Processor B receives the message from Processor A and passes it along to Processor C.

## File Service

TBD

## Table Service

The Table Service (TBL) provides Applications with a mechanism through which an Application’s parameters can be exposed. This exposure allows ground operations and other Applications the capability of modifying parameters within an Application while it is executing. The Table Service creates shared memory resources as Applications register their Tables, responds to externally generated Table commands, modifies Table entries as appropriate and ensures Table data integrity using a data integrity check algorithm. Figure 3-TBD shows the context of the cFE Table Service.

Filenames,

Table Handles,

Data Pointers

Filenames, File Handles, File Content

SB Messages

Event Messages

Time

Processor ID,

Application ID

Applications

Table Registry

* Table name
* Table attributes
* Time of last modification
* Last filename loaded
* Modified since loaded flag
* Etc.

Active and Inactive Table

Memory

Figure 3-TBD

The Table Service accepts Table registration requests from Applications. In response to these requests, it allocates the necessary resources in the Active and Inactive Table memory area to contain the desired Table and any necessary buffers. It also creates an entry in the Table Registry to contain the specified attributes of the Table and any additional Table maintenance data. Active Tables are those portions of memory containing Table data that are actively accessed by the Applications. An Inactive Table is defined as a copied image of the Active Table that can be manipulated via command while the Active Table is being used. Once the Inactive Table image is considered updated, it is either copied (for Single-buffered Tables) into the Active Table image or, for Double-buffered Tables, it becomes the Active Table image. Table content is provided from files. Applications and commands can specify which files to access to obtain the data to be put into a Table. The header information in the file will identify what Table the File is for, how much data is contained in the file and where the data belongs in the Table. The Table Service will then be able to perform a partial or complete Table load based upon the file header information. The Table Service will also be capable of dumping the contents of an Active or Inactive Table image to a file.

## Functional Requirements

The cFE functional requirements are configuration controlled and managed in the NASA Wide Babelfish git repository. These requirements are configuration controlled via a markdown document (“cFE\_FunctionalRequirements.md”) and published as a .pdf (cFE\_FunctionalRequirements.pdf”) for each release. The following list summarizes some key points that will aid the reader when viewing the published functional requirements document:

1. Each requirements subsystem section in the functional requirements is structured as follows (similar to interfaces constraints identified in section 2.2):
   1. Operational Interface – Requirements that pertain to the subsystem’s Operational Interface.
   2. Application Interface – Requirements that pertain to the subsystem’s Application Interface.
   3. Hardware Interfaces – Requirements that pertain to the subsystem’s Hardware Interfaces. This includes initialization.
   4. Platform Adaptation – Requirements that pertain to the subsystem being customized for a platform that have not been captured in previous sections.
2. The phrase “Upon receipt of a Command, the cFE shall …” is used to define a Command that can be issued from the UI. The requirements are worded from the cFE’s perspective in order to elicit cFE functional requirements that are related to the Command. All Commands are defined in the requirements. All commands are tested by test scripts running from a UI system.
3. The phrase “Upon receipt of a Request, the cFE shall …” is used to define a Request that is issued by an Application. The requirements are worded from the cFE’s perspective in order to elicit cFE functional requirements that are related to the Request. All Requests are defined in the requirements. Requests are tested via TBD.
4. Variable requirements are tagged with either <OPTIONAL> or <PLATFORM\_DEFINED>. A reference cFE Platform is defined to provide defaults for variable requirements. These defaults define the limits of the cFE’s capabilities.

Appendix A - Requirement Definition Strategy

This appendix describes the FSW requirements definition process for the cFE. This appendix is being written after several months of developing the cFE requirements. It has become apparent that in order to define requirements for the cFE, a product that will be configurable for multiple missions, we must have a clear understanding as to how and why we define the requirements the way we do.

In parallel to the cFE requirements definition effort, the FSW Branch has been trying to define a FSW requirements review guideline. In order to a have standard FSW requirements review process, again we must understand how and why we define the requirements the way we do. This document is not intended to establish a branch policy, but it should be a good starting point from which a policy can be defined. A consistent requirements definition strategy will aide the branch in project management and process improvement efforts. Project managers can use historical data to estimate FSW development and testing resources for their current project. Branch management can analysis multiple missions to determine evaluate FSW development and testing processes.

This appendix is organized as follows:

A.1 Describes a generic FSW development and validation process. This process provides the context for the FSW SRS and for the ICDs.

A.2 Describes the business aspects of the cFE. This section describes characteristics of the cFE from a business perspective. It also identifies the requirement stakeholders and how they relate to the cFE.

A.3 Describes characteristics of a good SRS.

A.4 Using the information from the previous three sections this section provides specific guidelines for writing the cFE requirements and provides rationale for the decisions.

A.1 FSW Development and validation process

This section shows how the FSW Requirements Specification is used during the FSW development and verification processes for a single mission. These processes define the context for the FSW Requirements Specification. cFE specific issues will be addressed in section A.4. Figure A-1 shows a reference FSW development process. Goddard does not have a standard FSW development process so this may vary between missions. The general process can be viewed in layers. The top-most layer contains spacecraft level activities. The second layer contains spacecraft subsystem activities. Note that the FSW can view a group of subsystems as a FSW system. This is important when the qualification strategies are discussed. The third layer decomposes subsystems into software products and the activities associated with the products. The fourth layer decomposes software products into units. The remainder of this section describes each layer in detail. The downward flow on the left half of the figure is described first followed by the upward flow on the right hand side.



Figure A-1 FSW Development Process

FSW requirements definition begins with the spacecraft system requirements. The system requirements formally flow into subsystem requirements. Not shown is the informal flow of operational scenarios that may impact subsystem requirements. A subsystem can be either hardware, software, or a combination of both. It can also be a remote system from the FSW subsystem. For example, the ground system is a subsystem of the spacecraft system. As the FSW Subsystem Requirements are developed a consistency is maintained between subsystems that interface with the FSW subsystem. This is shown by the horizontal double-arrow between Subsystem X and the FSW Subsystem.

The FSW Requirements Specification is formally derived from the FSW subsystem requirements. At this level ICDs are developed between the FSW subsystem and all of the subsystems that interface with it. From a requirements definition perspective the Level 0 Data flow diagram identifies each of these interfaces. The FSW interfaces to both software and hardware subsystems. For example, a C&DH FSW context diagram would show the GNC FSW as an external data source/sink and vice versa. From a programmatic perspective it is useful to create separate development plans for each FSW subsystem. Each FSW subsystem has its own dependencies that mature at different times (i.e. impacts to schedule). In cases when an external organization provides algorithms for the FSW team to implement an algorithms platform is developed. This platform is shown at this level because results from this platform are used to verify build test results as discussed later.

FSW is implemented at the bottom level. The FSW requirements are specified to a level of detail sufficient for developers to design a system to satisfy the requirements. Implementation details are derived from the requirements, ICDs, and algorithms documents (when applicable). For algorithm-based FSW the algorithms can be written as FSW units on the algorithm platform (and “shared” by FSW), manually translated into FSW, or automatically translated into FSW.

Once a FSW product is created it must be qualified (verified and validated). The qualification processes are shown on the right hand side of Figure A-1 following the upward flow. The goal of each testing level corresponds to a component in the problem definition and solution processes. Tests must be written from a perspective of meeting the goals of a particular level. In terms of software functional coverage overlap will occur at each level. The key to an efficient test program is to only do what is necessary at each level and this is driven by how well the definition and development levels are defined. There are programmatic issues as well such as the fidelity and availability of the test environments that are beyond the scope of this document.

Unit testing verifies the correctness of a unit. Units may be combined to test integrated units. Ideally unit testing is performed on the target platform using the development tools in the same configuration used for the product releases. A simulated target environment can be used as well. The simulation can occur at either a hardware or software level and with different degrees of fidelity. For algorithm-based FSW results should be obtained from the Algorithm developer for verification of the FSW.

The FSW required for performing build testing is integrated into a configuration managed product called a build. The developers enhance the test environment (ground database, page definitions, etc.) as needed and execute tests that verify the interfaces and functionality of the FSW are correct. The testing is done in the build test lab (typically breadboards).

Build testing is performed by a team separate from the development team. The primary goal of build testing is to verify the FSW meets all of its functional and performance requirements as defined by the SRS. A secondary goal is to verify Operational Interface elements introduced during the design and implementation of the FSW. These elements include commands, tables, events, and telemetry. A verification matrix should be maintained for both the requirements and the Operational Interface design elements. A build test team needs both the FSW Requirements Specification and the FSW User’s Guide in order to perform testing. ICDs are not necessarily required because they provide the implementation details.

FSW System Tests validate that the FSW system (a combination of all FSW subsystems) meets the operational requirements. This is a different perspective from build testing, similar to how Build Testing is different from Unit Testing. Unit Tests verify each individual unit and each line of code within a unit. Build Tests verify each functional requirement and each Operational Interface element for a build. System Tests validate operational scenarios and subsystem coordination. System Test are still performed in a test lab as opposed to the on the spacecraft. The fidelity of this lab is mission specific.

A.2 cfe Business process considerations

The section looks at how the cFE will be managed as a product. This is important from a requirements perspective because the cFE is product that will be used on multiple projects. How the cFE will be qualified for a particular project determines the scope and characteristics of the cFE requirements specification. The section looks at the following scenarios:

1. cFE-based products
2. cFE-based products with external organization Applications
3. External Organization cFE-based product

A.2.1 cFE-based Products

This section analyzes the development and qualification of Code 582 products based on the cFE and identifies issues related to the requirements.

Development of the cFE itself

In order to qualify the cFE the multi-mission aspects of the cFE must be understood. Reference missions should be defined to provide concrete instantiations used for testing. A default reference mission should be defined as a starting point for all missions.

Mission Proposals

In this case the proposal itself is the product. The requirements (and other support material) must be clear as to what platforms the cFE supports. Cost estimates for porting options should also be available, but these are beyond the scope of this document.

Non-Space Mission Applications

Missions may also develop cFE compliant Applications independent of a mission. This would typically occur in an advanced technology scenario. The cFE Developers Guide plays the key role in this scenario. Application developers need know how to make their Applications portable. This is based on the requirements but the developer should not be referring directly to the requirements.

The developer needs a test system in order to validate a new Application. The developer should not have to run a cFE test suite. They will be using the cFE as a delivered product.

Space Missions

This scenario includes both spacecraft bus software and instrument software.

The variable cFE requirements must be easily understood. A FSW Systems Engineer needs to evaluate what platforms are supported by the cFE and must be able to perform a cost estimate for non-conforming platform options.

Mission Application developers need to know how to develop mission-specific and potentially reusable cFE-compliant Applications. The cFE Developer’s Guide provides this information, not the requirements.

A cFE flight image and a cFE test suite is instantiated for a mission. The cFE test suite will need to be executed and the results verified. This should be done even for a supported platform.

The cFE requirements numbering should remain independent of the other mission-specific software subsystems, because the cFE itself is a self-contained FSW subsystem.

A.2.2 cFE-based Products with Externally Developed Applications

This scenario would typically occur for a Space Mission when Code 582 is relying on the GNC branch or a scientist to provide Application-specific algorithms. The cFE platform provided to the organization would probably be a desktop version. The cFE would not need to be validated. However, in order for the cFE to be useful a simulation environment would be needed.

An Application developer would also need debugging tools and support utilities, but these are outside the scope of this document.

A.2.3 External Organization cFE-based Product

The level of formal cFE verification depends on what the product is supporting. If the product is for a space mission then the external organization may want to run a cFE test suite even if they’re running on a supported cFE platform. If a cFE Port was required then a build test suite should be run and verified. The level of Code 582 involvement would be project defined.

An example of this scenario is the development of a High Gain Antenna controller that has a 1553 interface. The HGA controller could subscribe to the main processor’s ephemeris and attitude telemetry to get the information it needs to control the antenna.

For technology projects, or something similar, the cFE could be delivered as shrink wrapped product. An Application developer would also need debugging tools and support utilities, but these are outside the scope of this document.

A.3 Characteristics of a good requirements specification

This section defines characteristics of good requirement specifications. Many of these are taken from reference 6.

A.3.1 Correct

An SRS is correct if, and only if, every requirement sated therein is one that the software shall meet.

*Correctness cannot be automatically verified; it is a human endeavor. Traceability to higher level specification, and customer/user review helps ensure that the SRS correctly reflects the actual needs.*

A.3.2 Unambiguous

An SRS is unambiguous is, and only if, every requirement has only one interpretation.

*This requires that each characteristic of the final product be described using a single unique term. In cases where a term used in a particular context could have multiple meanings, the term should be included in a glossary where its meaning is made more specific.*

*A requirement should be unambiguous to those that create it and to those that use it, developers and testers. This can be difficult because each group may have a different technical background.*

Strategies for mitigating ambiguity include:

1. Focus on the FSW functionality (i.e. what is bounded by the context diagram) not on a system that interfaces with the FSW. For example don’t state, “The ground shall issue a command to …” Instead state “Upon receipt of a command the FSW shall…”
2. Be specific as possible. Natural language is inherently ambiguous and efforts must be made to constrain the ambiguity. Avoid statement such as “the FSW shall support Coarse Sun Sensor data processing.” Instead state “the FSW shall convert valid Coarse Sun Sensor data in engineering units”.
3. Define a single FSW characteristic for each numbered requirement. This also helps traceability.
4. Use brief declarative that represent complete capabilities that are testable.

A.3.3 Complete

An SRS is complete if, and only if, it includes the following elements:

1. All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces. In particular any external requirements imposed by a system specification should be acknowledged and treated.
2. Definition of the response of the software to all realizable classes of input data in all realizable classes of situations. Note that it is important to specify the responses to both valid and invalid input values.
3. Full labels and references to all figures, tables, and diagrams to the SRS and definition of all terms and units of measure.
4. The use of “to be determined” (TBD) can be used to indicate an incomplete SRS, although it should be used sparingly. If it is used it should be accompanied by:
   1. A description of the conditions causing the TBD (e.g., why an answer is not known) so that the situation can be resolved;
   2. A description of what must be done to eliminate the TDB, who is responsible for its elimination, and by when it must be eliminated.

A.3.4 Consistent

An SRS is internally consistent if, and only if, no subset of individual requirements described in it conflict. Three types of likely conflicts in an SRS are as follows:

1. The specified characteristics of real-world objects may conflict. For example, one requirement may state that a status indicator from a remote processor is a flag that toggles, but in another that it’s a counter.
2. The may be a logical or temporal conflict between two specified actions. For example, one requirement may state that “A” must always follow “B” while another may require that “A and B” occur simultaneously.
3. Two or more requirements may describe the same real-world object but use different terms for that object. For example, a storage area may be a called a “buffer” in one requirement and a “log” in another. The use of standard terminology and definition promotes consistency.

In an organization that produces similar products each product’s requirements content should be consistent. Requirement phrasing is indicative of how requirements are elicited. For example, “Upon receipt of a command, the FSW shall…” is stating a requirement relative to events across an interface. Consistently specifying products creates a corporate culture that helps the development, test, and maintenance efforts. The use of an organizational process guide accompanied with requirements guidelines promotes consistency.

A.3.5 Verifiable

An SRS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirements. In general, an ambiguous requirement is not verifiable.

Non-verifiable requirements include statements such as “works well,” “good human interface,” and “shall usually happen.” These requirements cannot be verified because its impossible to define the terms “good,” “well,” or “usually.” The statement “the program shall never enter an infinite loop” is non-verifiable because the testing of this quality is theoretically impossible.

An example of a verifiable statement is

*Output of the program shall be produced within 20 seconds of event X 60% of the time; and shall be produced within 30 seconds of event X 100% of the time.*

This statement can be verified because it uses concrete terms and measurable quantities.

If a method cannot be derived to determine whether the software meets a particular requirement, then that requirement should be removed or revised.

A.3.6 Modifiable

An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirement can be made easily, completely, and consistently while retaining the structure and style. Modifiability generally requires an SRS to

1. Have a coherent and easy-to-use organization with a table of contents, and index, and explicit cross-referencing.
2. Not be redundant (i.e., the same requirement should not appear in more than one place in the SRS).
3. Express each requirement separately, rather than intermixed with other requirements.
4. Replace the text of a deleted requirement with the word “Deleted” and retain the requirement’s number. The requirements management tool maintains a history of requirements changes, so that information on the original requirement, reason for changes, and who made them can be retrieved at any time.
5. The SRS characteristics that aide in traceability (see A.3.7) also help modifiability.

Redundancy itself Is not an error, but it can lead to errors. Redundancy can occasionally help to make an SRS more readable, but a problem can arise when the redundant document is updated. For instance, a requirement may be altered in only one of the places where it appears. The SRS then becomes inconsistent. Whenever redundancy is necessary, the SRS should include explicit cross-references to make it modifiable.

A.3.7 Traceable

An SRS is traceable if the origin of its requirement is clear and it facilitates the referencing of each requirement in future development or enhancement documentation. The following two types of traceability are recommended:

1. *Backward traceability (i.e. to previous stages of development).* This depends upon each requirement explicitly referencing its source in earlier documentation.
2. *Forward traceability (i.*e. to all documents spawned by the SRS). This depends upon each requirement in the SRS having a unique name or reference number.

Prohibiting multiple requirements from being defined in a single requirements reference aides in both types of traceability.

The forward traceability of the SRS is especially important when the software product enters the operation and maintenance phase. As code and design documents are modified, it is essential to be able to ascertain the complete set of requirements that may be affected by those modifications.

A.4 cFE requirement guidelines (STILL BEING WORKED)

This section presents several controversial topics that were raised when originally trying to establish a consistent requirements definition strategy. Theoretically, it seems easy to make simple statements as to what makes a good requirement (see section A.3), but when real life problems are addressed many gray areas surface. This section addresses the gray areas that have typically surfaced during the cFE requirement discussions. There may be multiple resolutions to these topics. The purpose of this section is to define the resolution to the problem for the cFE, provide rationale for the resolution, and identify the potential consequences.

The cFE presents several challenging scenarios with respect to software qualification. First, the cFE is specifying requirements for FSW functions that have traditionally either been poorly specified or left to the designer. These functions include the Executive Services, Event Manager, parts of Time Manager, and parts of the Software Bus. These are “low level” functions that have a limited ground (or any external cFE interface). In order to validate the functions either a cFE test application is required to externally expose the interfaces so the functions can be exercised from a test script or a unit test suite can serve as the validation mechanism. Second, the cFE is tightly coupled with interfaces that are intended to be standardized for multiple missions. In addition to the ground-to-flight interface, these interfaces include the cFE-to-Application interface and the cFE-to-cFE interfaces that exist when the cFE is deployed on a multi-processor platform. The validation challenges are similar to the “low level” functional validation challenges. Finally, the cFE requirements contain <MISSION\_DEFINED> and <OPTIONAL> specifications. This feature presents the challenge of tailoring both the FSW and the test suites for a mission specific deployment.

A goal of the cFE is to allow a new mission to easily specify its mission-unique requirements, build the cFE, deploy the cFE, and validate the mission-unique configuration. The solutions to the previously mentioned challenges should keep this goal in mind.

Refer back to A.1 and show relationship to the process.

**1. General Capabilities**

|  |  |
| --- | --- |
| **Problem** | A requirement is stated as a high level capability. For example, “the cFE shall be capable of easily loading and unloading an Application.” This is not a testable requirement. |
| cFE Strategy | This problem arises from lack of system requirements. Since there isn’t a parent requirements document, system requirements will be stated (along with rationale) in the introductory material. |
| Rationale | This captures the information without the need for another document. |
| Consequences | Formal traceability is not maintained. This is a minor consequence since this product is an internal development effort. |

The wording of the requirements has been carefully chosen. In general cFE functional requirements are elicited via activity across interfaces. This makes the requirements active rather than passive as in statements such as “the cFE shall be capable of …” Two common phrases have important contextual meaning. The phrase “upon receipt of a Command” means the Command is received as a message on the SB and is defined in the cFE- User’s Guide.. The Command can originate from the ground or from another onboard processor either locally or remotely. The phrase “upon receipt of a Request” means the request originated on the same processor as the cFE and the Request is described in the cFE Application Developer’s Guide. From a design perspective a Request can be implemented as either a SB message or a library function call.

**2. Command Specification**

|  |  |
| --- | --- |
| **Problem** | Should all event messages be defined in the FSW Requirements? |
| cFE Strategy | Define all critical events in the requirements.  Build testing validates all event messages. |
| Rationale | Not all events are known until design. Building testing can’t be done without the flight-ground ICD anyway so why not test to the ICD? |
| Consequences | Testers will need to trace to the ICDs. |

|  |  |
| --- | --- |
| **Problem** | Should all event messages be defined in the FSW Requirements? |
| cFE Strategy | Define all critical events in the requirements.  Build testing validates all event messages. |
| Rationale | Not all events are known until design. Building testing can’t be done without the flight-ground ICD anyway so why not test to the ICD? |
| Consequences | Testers will need to trace to the ICDs. |

|  |  |
| --- | --- |
| **Problem** | Should all event messages be defined in the FSW Requirements? |
| cFE Strategy | Define all critical events in the requirements.  Build testing validates all event messages. |
| Rationale | Not all events are known until design. Building testing can’t be done without the flight-ground ICD anyway so why not test to the ICD? |
| Consequences | Testers will need to trace to the ICDs. |

Command versus table load

Should each command have a TLM verification

Is parameter checking an ICD issue or a command issue?

Should all commands be specified?

All MAP ACS commands are except for an init HiFI table from EEPROM.

Many event messages are not defined. For example, rate filter initialized.

**3. Event Messages**

|  |  |
| --- | --- |
| **Problem** | Should all event messages be defined in the FSW Requirements? |
| cFE Strategy | Define all critical events in the requirements.  Build testing validates all event messages. |
| Rationale | Not all events are known until design. Building testing can’t be done without the flight-ground ICD (i.e. User’s Guide) anyway so why not test to the ICD? |
| Consequences | Testers will need to trace to the ICDs. |

|  |  |
| --- | --- |
| Alternate  Strategy A | All events defined in the requirements.  Build testing validates all event messages. |
| Rationale | All events should be the result of a required functional behavior. Build testing validates requirements. |
| Consequences | The requirements document may change frequently.  If a new event message is deemed appropriate is it because the requirement was missing or it really is a product of the design? |

|  |  |
| --- | --- |
| Alternate  Strategy B | All critical events defined in the requirements.  Build testing validates all required event messages. |
| Rationale | Event messages created during the design phase don’t need to be tested by build tests, but only unit tested. |
| Consequences | If an event is never executed by build testing but the unit test driver used doesn’t validate the interface then an erroneous message could be delivered. However, if the cFE provides a standardized unit test environment with automated validation, then this problem could be avoided. |

**4. Telemetry**

|  |  |
| --- | --- |
| **Problem** | Should all event messages be defined in the FSW Requirements? |
| cFE Strategy | Define all critical events in the requirements.  Build testing validates all event messages. |
| Rationale | Not all events are known until design. Building testing can’t be done without the flight-ground ICD (i.e. User’s Guide)anyway so why not test to the ICD? |
| Consequences | Testers will need to trace to the ICDs. |

The requirements should specify critical telemetry points. The design can add additional TLM but the requirements should not need to be updated. For the critical TLM points do the requirements need to state whether they are sent periodically or upon request? Yes I think they do need to state it because it determines what operators need to do.

**5. Algorithms: Requirement vs. Design vs. ICD**

|  |  |
| --- | --- |
| **Problem** | Should all event messages be defined in the FSW Requirements? |
| cFE Strategy | Define all critical events in the requirements.  Build testing validates all event messages. |
| Rationale | Not all events are known until design. Building testing can’t be done without the flight-ground (i.e. User’s Guide) anyway so why not test to the ICD? |
| Consequences | Testers will need to trace to the ICDs. |

Time and equations

Event Message filtering scheme: should it be in the requirements or an ICD? There are advantages to deferring the discussion in the development lifecycle.

**6. Requirement vs. Design**

|  |  |
| --- | --- |
| **Problem** | Should all event messages be defined in the FSW Requirements? |
| cFE Strategy | Define all critical events in the requirements.  Build testing validates all event messages. |
| Rationale | Not all events are known until design. Building testing can’t be done without the flight-ground ICD (i.e. User’s Guide) anyway so why not test to the ICD? |
| Consequences | Testers will need to trace to the ICDs. |

Partial table loads and what should be checked. If specify an offset and a length then it implies design.

**6. Executive Services**

|  |  |
| --- | --- |
| **Problem** | Should all event messages be defined in the FSW Requirements? |
| cFE Strategy | Define all critical events in the requirements.  Build testing validates all event messages. |
| Rationale | Not all events are known until design. Building testing can’t be done without the flight-ground ICD (i.e. User’s Guide) anyway so why not test to the ICD? |
| Consequences | Testers will need to trace to the ICDs. |

Explicitly state that they return errors.

**7. Requirement vs. ICD**

|  |  |
| --- | --- |
| **Problem** | Should all event messages be defined in the FSW Requirements? |
| cFE Strategy | Define all critical events in the requirements.  Build testing validates all event messages. |
| Rationale | Not all events are known until design. Building testing can’t be done without the flight-ground ICD (i.e. User’s Guide) anyway so why not test to the ICD? |
| Consequences | Testers will need to trace to the ICDs. |

How does it affect V&V? May need a bench level test effort.

ICD versus Requirement Issues

Software Bus message validation

What should be specified for packet validation? Don’t want the requirements to have knowledge of the message structure. Think about what you’re trying to achieve in order to decide what should be specified. Do you want reliable transfer? However is checking the packet length a “how” not a what?

Should command vs. Table be specified? For example ephemeris updates.

**8. Error Conditions**

|  |  |
| --- | --- |
| **Problem** | Should all event messages be defined in the FSW Requirements? |
| cFE Strategy | Define all critical events in the requirements.  Build testing validates all event messages. |
| Rationale | Not all events are known until design. Building testing can’t be done without the flight-ground ICD anyway so why not test to the ICD? |
| Consequences | Testers will need to trace to the ICDs. |

Is it best to say for all error conditions specified in the ICD the cFE will issue and event message? Otherwise need to know the format in order to itemize what needs to be issued.

When a command is specified should all error paths be specified? This is similar to knowing the structure.

**9. Capacity**

|  |  |
| --- | --- |
| **Problem** | Should all event messages be defined in the FSW Requirements? |
| cFE Strategy | Define all critical events in the requirements.  Build testing validates all event messages. |
| Rationale | Not all events are known until design. Building testing can’t be done without the flight-ground ICD anyway so why not test to the ICD? |
| Consequences | Testers will need to trace to the ICDs. |

The cFE shall support a maximum of 16 interrupts.

**9. Variable Requirements**

|  |  |
| --- | --- |
| **Problem** | Should all event messages be defined in the FSW Requirements? |
| cFE Strategy | Define all critical events in the requirements.  Build testing validates all event messages. |
| Rationale | Not all events are known until design. Building testing can’t be done without the flight-ground ICD anyway so why not test to the ICD? |
| Consequences | Testers will need to trace to the ICDs. |

The cFE shall support a maximum of 16 interrupts.

Variable requirements are tagged with either <OPTIONAL> or <PLATFORM\_DEFINED>. Quantifiable requirements that can be tailored by a mission are structured as a high level requirement stating the cFE quantity and a sub-requirement defining the mission specific quantity. For example, a high level requirement (x.1) would specify the maximum number of cFE Applications and a sub-requirement (x.1.1) would specify the <PLATFORM\_DEFINED> number of Applications. The following table further defines tagged requirements. Note that the FSW and test designs should have a mechanism for addressing each type of variability.

|  |  |
| --- | --- |
| **Keyword** | **Description/Notes** |
| PLATFORM\_DEFINED | An aspect of the cFE that can be configured during the instantiation of the cFE. A reference platform defines the cFE PLATFORM\_DEFINED bounds. The cFE build test suite validates the reference platform. |
| OPTIONAL | A requirement that can be included or excluded in its entirety. |

*Provide rationale for the following statement:*

An earlier version of the requirements used the phrases “Upon receipt of a Command to … as specified in the cFE COMMAND AND TELEMETRY ICD, the cFE shall,…” and “Upon receipt of a Request to … as specified in the cFE Application Developer’s Guide, the cFE shall…”. Due to the convention described in the previous paragraph the “as specified in the ICD” phrasing has been removed to make the requirements more concise.

*Move to examples section in pervious section:*

For each Command and Request the successful case is presented as the top-level requirement. Error cases are identified as sub-requirements. Error cases are enumerated as individual requirements if the error condition requires a specific functional behavior. Otherwise it is assumed that the ICD corresponding to the Command or Request enumerates the error conditions.

Appendix B - Heritage Analysis

This appendix contains the results of a heritage analysis effort that was performed prior to the definition of the cFE requirements.

B.1 Executive Services

TBD

B.2 Time Manager

TBD

B.3 Event Manager

Most (if not all) SMEX, MIDEX missions at Goddard have provided an asynchronous error message interface. These messages have been very useful during development and for on orbit operations.

Flight operators have stated that event messages provide them with a single display window that “gets their attention” with important spacecraft information. This information has generally included a time stamp and real-time telemetry that indicated the exact nature of the event. This data could then catch transients that would not show up in the periodic telemetry that is normally transmitted to the ground.

Some missions have chosen to implement messages as strings, complete with printf-formatted output. While more resource (both bandwidth and on board memory) constrained mission have just used small integer message ids with data and let the ground system output an appropriate message string.

To prevent a single application from flooding the system with messages, mission engineers have used some sort of event filtering. The filtering has included, x per unit time, one of x, first x, and simple on/off.

**JWST**

JWST used C++ to implement an event class where the application would concatenate each piece of the message string together with the data before calling the send method. Currently, only integer parameters are part of the class. Envisioned, but not yet implemented, is operator overloading for different parameter data types.

There is no filtering or anti “flood” mechanism built in, it would be up to the application to implement.

**Triana**

Triana used C++ where each application inherited the eventFactory class where they would add application specific aspects and call the event method. Each application handled the interface ad hoc with some designs implementing overloaded operators to deal with combinations of different parameter types (byte, float, double, etc).

Strings were not implemented for telemetry, but were implemented for debug port output, ad hoc per application.

**Triana ACS,**

Developers wrote their own event handler that called the C&DH one. They used simple mask filters similar to the proposed scheme. See code below.

**GLAS,**

GLAS used an implementation in C similar to the standard printf interface. This method provided flexibility with a variable number of parameters though a familiar API. Filters were implemented ad hoc with an “if counter < x” statement and were typically done per task. Some applications included filtering based on sending only a certain number of events per time period. (Not sure how to address this, or weather it’s ready needed when you have counters for each message)

**FAST**

FAST was a resource constrained mission that implemented a configurable per message filter design. Each message type had an associated unique small integer number, which was used to index a table of error counters and a table of message filter values (loadable and individually commandable). Each time the error handler was called it would “and” the counter with the corresponding filter value, when the result was 0, the message was sent. The counter was then incremented. This method, allowed some flexibility in the filtering. For example, a filter of 0xfff0 would send the first 16 errors and stop. A value of 0x0001 would send every other message. This simple mechanism satisfied the needs of the spacecraft operators with relative ease and did not tax the 8 bit uProcessor too much.

An advantage of incrementing the counter after the “and”, meant that no message could be completely turned off inadvertently, a disadvantage was no message could be completely turned off on purpose. I should note, that counters did not rollover.

Bandwidth constraints precluded the use of message strings.

# Implementation Tradeoffs

The proposed API allows for the range of a no strings to a full printf formatted strings implementation, without changing the interface.

The software systems engineer can decide whether to send strings or not as bandwidth allows.

Same tradeoff for filtering, if the software systems engineer decides not to use individual message filtering, then the register filter call may be de-scoped to a single application global filter only.

**Proposed implementation notes.**

API should implement multiple output ports; such that messages may be configured via command to go out via TLM, debug port, console, etc. **Triana** started down this path but did not complete it for unknown reasons.

Applications use strings where bandwidth allows. Strings reduce the impact to ground data systems in that there is no database entries to create and maintain. Note that this does not affect the ground system requirement to catalog and provide operator instructions for each event. Most of this information is already contained in the software user’s guide.

Tables are used in the API, but we should also allow an interface without tables.

When no real-time filter table is registered, the module shall provide a default of 1 filter for all the calling tasks messages.

B.4 Software Bus

Notes regarding Copy/Pointer Transfer Modes:

The heritage SB (on SDO,MAP,GLAS type missions) allows the Send to be copy or pointer mode and also allows the Receive to be copy or ptr mode. This gives 4 possible transfer modes.

1.When send and rcv are copy mode – pkt is copied twice from source to destination

2.When send is copy and rcv is ptr – pkt is copied once from source to destination

3.When send is ptr and rcv is copy – pkt is copied once from source to destination

4.When send and rcv are ptr mode – pkt is not copied at all,from source to destination

On Swift: only 1 mode was offered (#2 above) – the send is copy mode, the rcv is ptr mode.

Our plan is to offer the Swift version(#2 above) which is a single-copy transfer mode and the zero copy mode (#4 above). These two modes do not need the receive in copy mode, therefore there is no current plan to implement the copy-mode receive.

B.5 File Manager

TBD

B.6 Table Manager

TBD

Appendix C - Scenario Analysis

This appendix provides a set of usage scenarios that were used to elicit requirements. Rationale is provided for each scenario that is supported by the cFE.

C.1 Initialization

Power-on Reset

Processor Reset

C.2 Hardware Access

Reading/writing I/O ports

Interrupts

Exceptions

C.3 Application management

Defining an Application

Load a new Application

Unload an Application

Modify an existing Application

Reset a single application

C.4 Application services

Semaphores

Message Queues

Shared Memory

C.5 Application Communications

Initialization

Between two local Applications

Between two non-local Applications

Redundant Applications

C.6 Times scenarios

Ground management of time

Application time service needs

C.7 Event scenarios

C.8 File scenarios

Load

Dump

Directory

C.8 Table scenarios

Load

Dump

Copy