Command Scheduler

Simon Janum
s194609

Kongens Lyngby 2022

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Abstract

This thesis has been carried out with the goal of implementing a command scheduler for satellite mission, using NASA's core Flight System (cFS), to allow for more functional autonomous operations. The functionalities are explained together with ESA's Packet Utilization Standard, to give an overview of the functionalities a command scheduler can have. As a result a foundation of a command scheduler, that allow for more functional autonomous operations, has been implemented. Some of the functionalities are not fully supported due to various reasons. This thesis will go over the development of such command scheduling for NASA's core Flight System (cFS).

Preface

This thesis was prepared at DTU Compute in fulfilment of the requirements for acquiring an B.Sc. in Software Technology at the department of Applied Mathematics and Computer Science, in the period 13-February to the 19-June. This project was put together with my supervisor Hans Henrik Løvengreen.

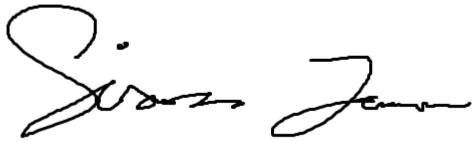
Software design and programming will be present, so it is advantageous to have some knowledge in those fields.

I would like to thank my supervisor for their guidance, support and making the time to have meetings and answering my questions. Without that I would not have been able to finish this project.

I hope you enjoy your reading.

- Simon Janum

Lyngby, 19-June-2022



Simon Janum
s194609

Contents

Chapter 1

Introduction

1.1 motivation

DTUsat at DTU has satellite that's student-built, and has various purposes, but is in need for proper flight software. Different solutions can be provided; one might create a new software, find an already existing software, or in this case, modify existing software to the needs for DTUsat, namely NASA's core Flight System. So what is the NASA's core Flight System?

NASA's core Flight System (cFS) [cFS21] is a free, open-source flight software system developed for space missions. cFS provides most most of the infrastructure needed for such missions in for the form of a number of independent components connected by a software bus.

A deficiency of cFS, however, is the lack of proper functionality for autonomous operation. Especially it is not readily possible to upload commands for later execution. Such a functionality is often called command scheduling. Command scheduling is important, since the ground system sending messages is not always in area of sight of the satellite. The satellite may be on the other side of globe than the ground system, and the ground system can therefore not send the command message until the satellite gets in sight. Scheduling commands on the satellite can prevent this problem.

The goal of this thesis is to give an introduction to OpenSatKit and the tools used in the implementation of the command scheduler, together with showing the implementation of said command scheduler. This includes explaining different functionalities a command scheduler can implement, and how said functionalities have been implemented.

1.2 Structure

The second chapter of this thesis is focused on an introduction to OpenSatKit. A description of its architecture is provided to better understand its layered system. Additionally the second chapter also describes various tools, services and applications that OpenSatKit provides, with focus on those that are important for this project.

The third chapter is a short analysis of what and why is wanted to be implemented. This consist of the main problem wanted to be solved by implementing a command scheduler, as well as an overall understanding to the problem. This part also include implementation goals that is wanted to be achieved.

The fourth chapter of this thesis is focused on how the command scheduler should be designed, from the different aspects presented in chapter 3. A general design will also be presented of how the scheduler should work together with the cFS.

The fifth chapter consist of showing and explaining how and what has been implemented, here code snippets will be provided. This part also includes how testing has been carried out throughout the implementation. Since this is a big project not everything explained in the chapter 3 has been implemented.

Lastly there will be a conclusion about the project.

Chapter 2

Introduction to OpenSatKit

2.1 OpenSatKit architecture

OpenSatKit, also refereed to as 'OSK' is an open-source tool which provide a core Flight System (cFS) and a runtime environment, which then can be used to learn about cFS. (https://github.com/OpenSatKit/OpenSatKit, V3.2).

OSK is composed of 3 components:

- **COSMOS:** User interface for command and control, has a number of tools to simulate the ground system.
- **Core Flight System (cFS):** Flight software that are able to operate in a Linux environment, and can also be exported to other devices.
- **42 Simulator:** Simulator, used to simulate a satellite (not used in this project).

2.2 COSMOS Command and Control

COSMOS is a command and control system, which is open-source, and written in the language Ruby. COSMOS provides different tools intended for testing, such as commanding, scripting and data visualization.

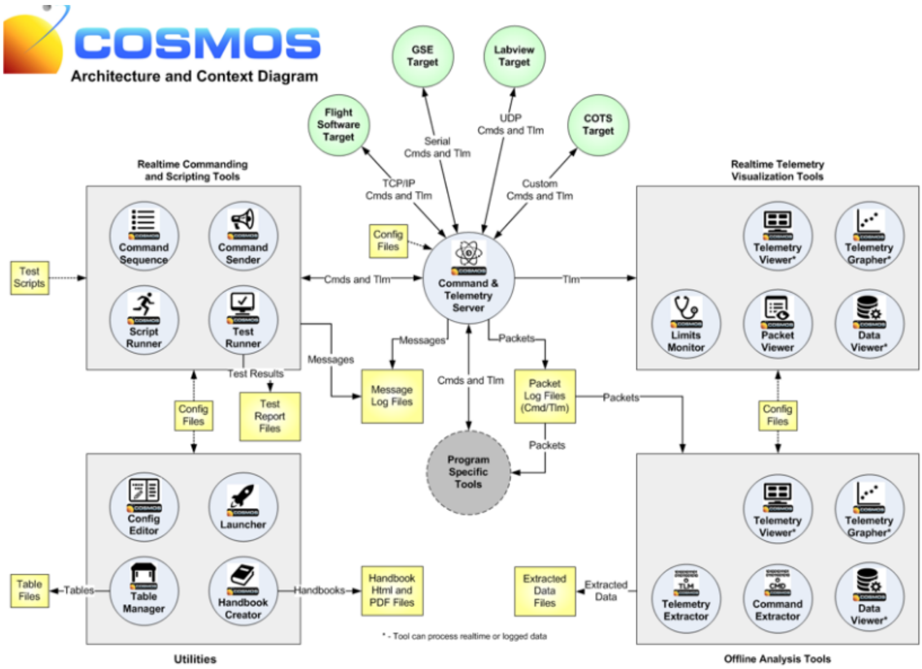


Figure 2.1: COSMOS Architecture [COS]

To achieve its purpose, COSMOS is made up of 15 different applications that provide different functionalities. It is not necessary to use all of of them, as in this project only a handful of them is used. Though being able to access them individually was found to be very helpful.

2.2.1 Launcher

The launcher is the first tool met in COSMOS, it provides an overview of all the applications COSMOS has to offer, as well as launching the applications. The Launcher also serves as a utility for an organized overview of COSMOS.

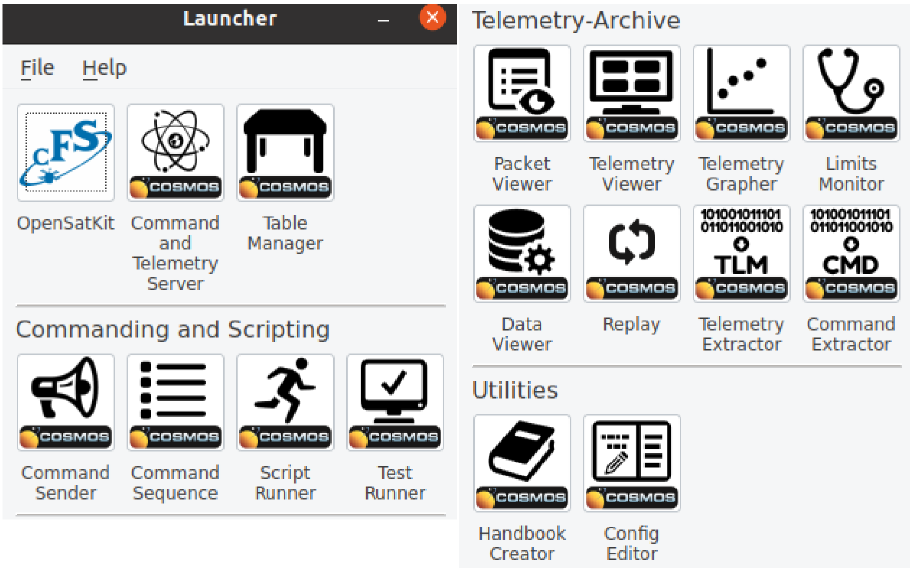


Figure 2.2: The Launcher

2.2.2 Command and Telemetry Server

The Command and Telemetry server provide connection from COSMOS to targets for real-time command and telemetry processing. This means that COSMOS tools, in real-time, communicate with its target through the Command and Telemetry Server application. Which then ensures that all the communication that happens is logged and can be viewed.

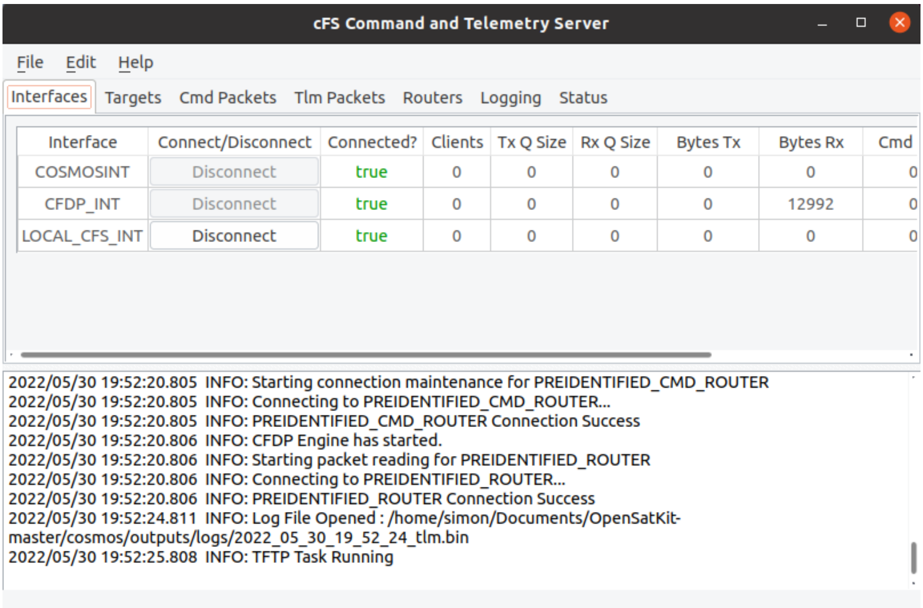


Figure 2.3: Command and Telemtry Server

Figure 2.3 shows the Command and Telemtry Server, from here the user can go into the Command Sender (Cmd Packets tab) and the Packet Viewer (Tlm Packets tab).

2.2.3 Command Sender

Command Sender is a tool at provides an easy method to send single commands to targets. It comes with a GUI that has simple dropdowns and a very easy-to-use selection of the desired commands that can be sent.

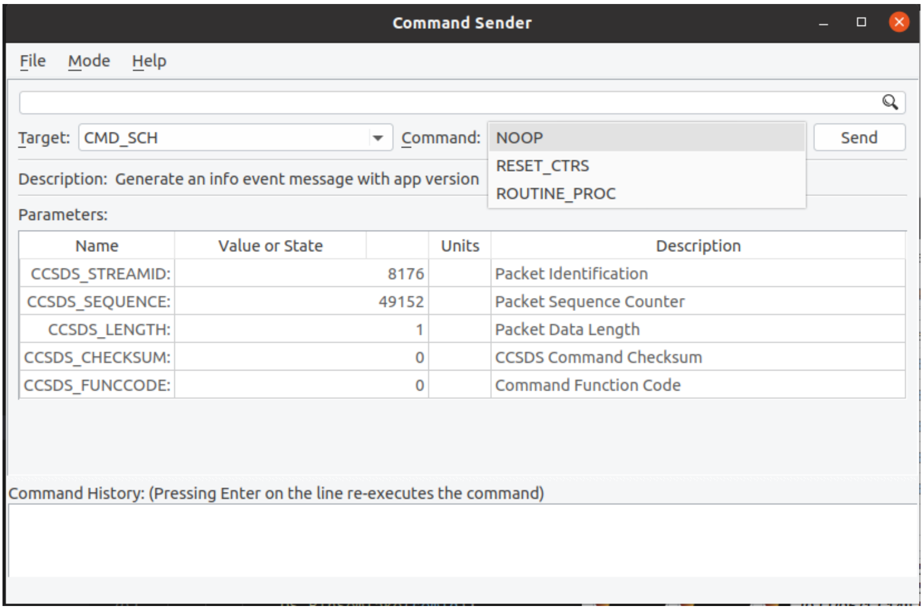


Figure 2.4: The Command Sender

Figure 2.4 shows the Command Sender, here the user can change the target application and command wished to be sent. It is also possible to change the parameters, as well as see the command history at the bottom.

2.2.4 Packet Viewer

Packet Viewer provides a tool that allows the user to view real-time content of any telemetry packet. This can be used together with the Command Sender application, to send and check received packets.

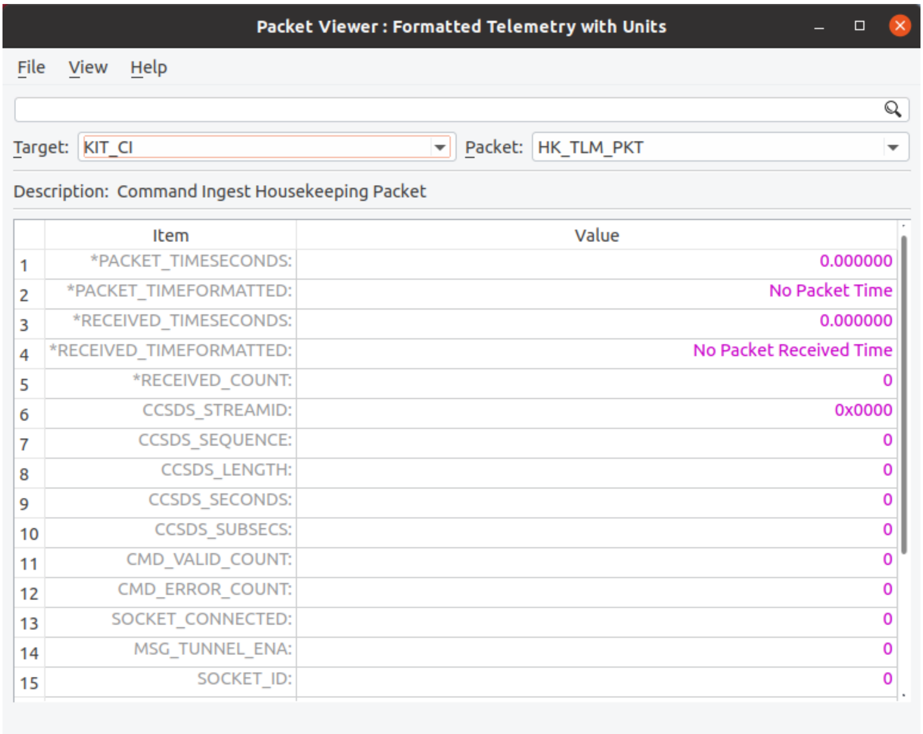


Figure 2.5: The Packet Viewer

2.2.5 cFS Education

OSK also has an interface used for education purposes, where there can be found guides and other helpful tools to get started with OpenSatKit.

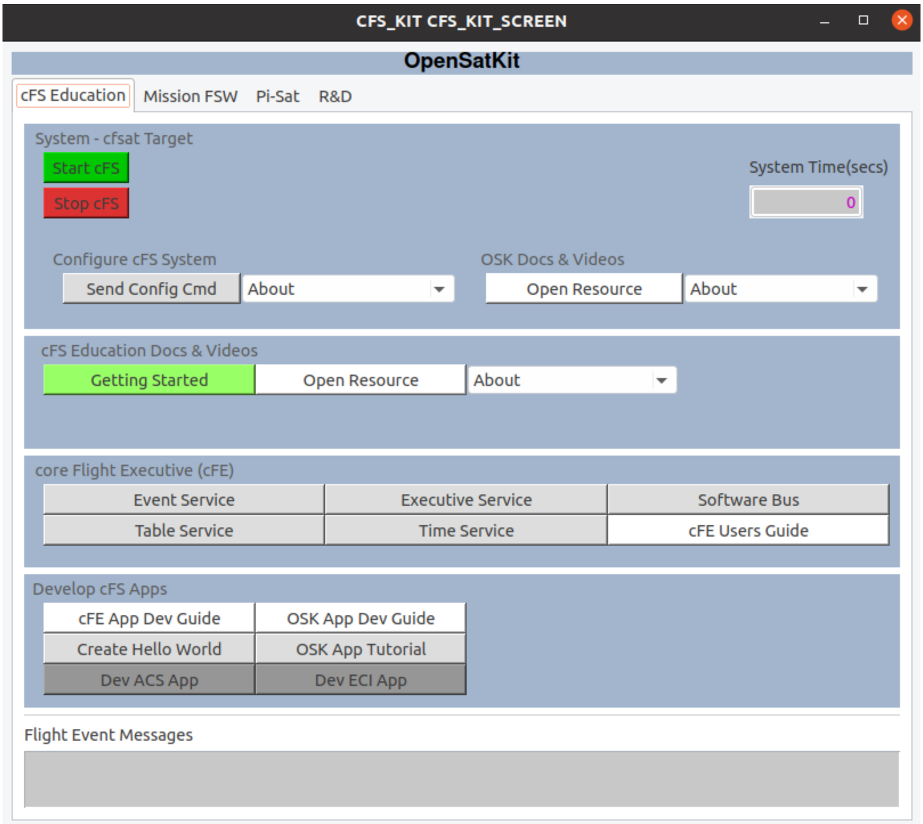


Figure 2.6: cFS Education tab

Figure 2.6 Shows the tab cFS education, here a few guides can be found. These tools are very useful when starting up learning about OSK, for example the "Create Hello World" button will help create a new app, it also helps with compiling and running the cFS.

2.3 Core Flight System (cFS)

This project of this report takes basis of developing an 'extension' to the Core Flight System (cFS) [cFS21]. The cFS is a reusable spacecraft flight software, that also includes a set of reusable software applications. Key aspects of the software is:

- Dynamic run-time environment.
- Layered architecture.
- Component-based design.

The cFS software is not a simulator and is meant to be ported to a real device.

2.3.1 cFS architecture layers

The cFS architecture contains 5 different layers:

RTOS/Boot Layer

This layer Provides the open-source software interface between the processor and flight software.

Platform Abstraction Layer

This layer provides two Application Program Interfaces (API's), which goal is to decouple the higher levels from hardware and operating system implementation details. The two API's are: Operating System Abstraction (OSAL) and Platform Support Package (PSP).

Core Flight Executive (cFE) Layer

This layer contains the cFE which is a portable, platform-independent framework that create an application runtime enviroment. This is done by providing services that were determined to be the most common, see 2.3.2. These services include a Software Bus, Event Messages, Time Management, Executive Services and Table Management.

Application Layer

This layer provide mission functionalities using combinations of cFS community apps and mission specific apps. This is also the layer that where the user can create and modify the apps for their desired purposes.

Development Tools and Ground Systems

These are in the top layer, and are used for testing and running the cFS.

2.3.2 cFE services

Executive Services (ES)

This service manages the software system, aswell as creating the application runtime environment.

Software Bus (SB)

Provides a portable inter-application message service using a publish/subscribe model with CCSDS standards.

By default the Consultative Commitee for Space Data Systems (CCSDS) [CCS03] packets are used for these messages. The packets contains 2 headers, a primary (always big endian) - and a secondary header. The secondary header contains either a command function code or a timestamp, depending on if the packet is a command packet or a telemetry packet. Figure 2.7, 2.8 and 2.9 shows the format of the headers and packet.

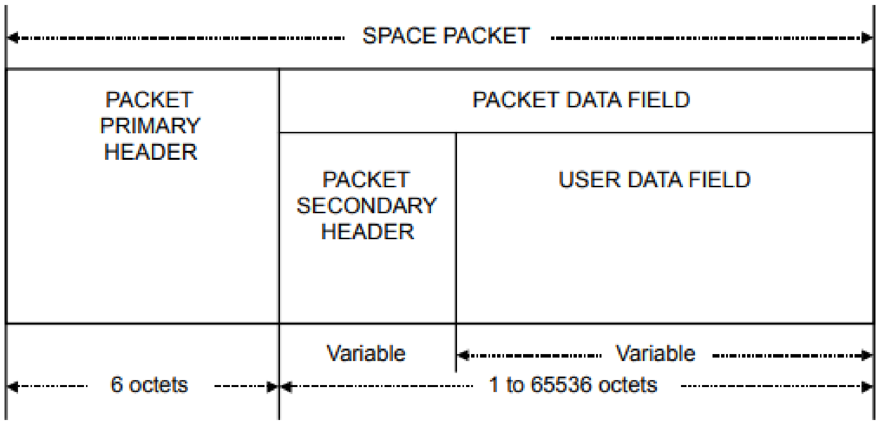


Figure 2.7: CCSDS Packet Format [CCS03]

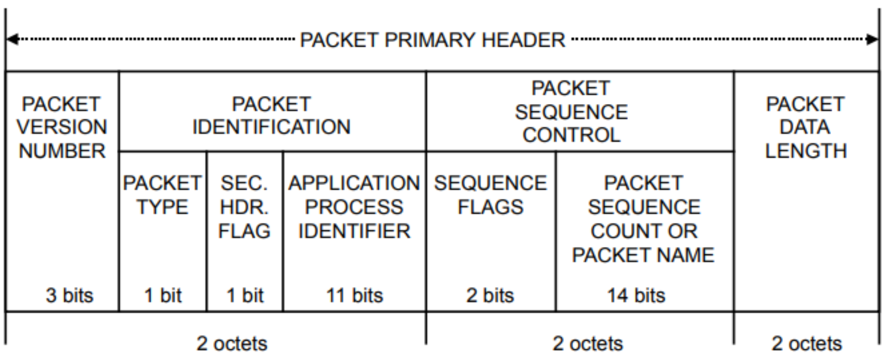


Figure 2.8: CCSDS Primary Header [CCS03]

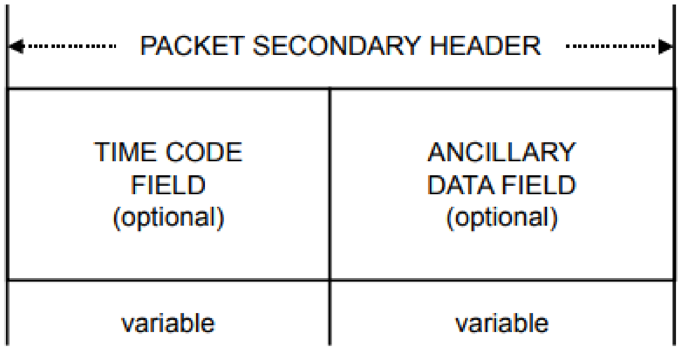


Figure 2.9: CCSDS Secondary Header [CCS03]

An importance for the primary header is that the 16 first bits are referred to as the *message ID* and are used to uniquely identify a message. The 11 last bits of the 16 first bits are referred to as *app ID* and are used as a CCSDS packet identifier.

The secondary header information is depended on the packet type, given in the primary header. If the packet is a telemetry packet, then the secondary header contains a timestamp. If the packet is a command packet, then the secondary header contains the command function code and a checksum.
Each app must create a pipe in order to receive messages, and also subscribe to the individual message ID wished to be received by the app. Apps subscribe and unsubscribe to messages at any time, and the sender does not know who is subscribed, hence its connectionless.

Event Services (EVS)

Provides an interface for sending timestamped text messages on the Software Bus. The event messages cant be of four types: Debug, Informational, Error or Critical. Example of this can be seen in figure 2.10.

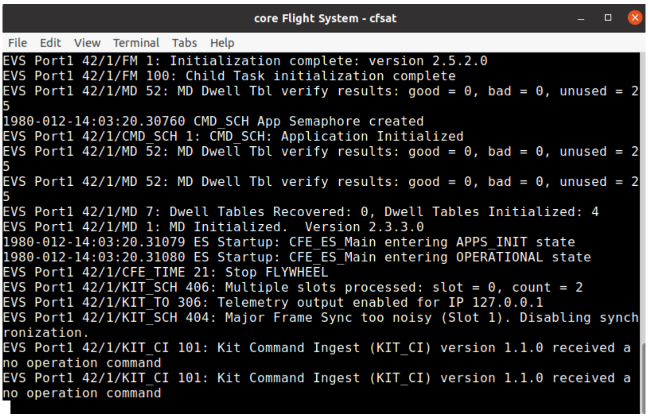


Figure 2.10: Event messages displayed in system terminal

Time Services (TS)

Provides time correlation, distribution and synchronization services. The service supports two time formats, that can be chosen:

- **International Atomic Time (TAI):** The number of seconds and sub-seconds elapsed since the ground epoch. TAI is calculated from the formula: TAI = MET + STCF.
  - Mission Elapsed Counter (MET) time since powering on the hardware containing the counter.
  - Spacecraft Time Correlation Factor (STCF) set by ground operations.
- **Coordinated Universal Time (UTC):** Is a synchronized time with astronomical observations, and is calculated as: UTC = TAI + Leap Seconds. Where leap seconds account for earths rotation.

Table Services (TBL)

Provides ground commands to load a table from a file and dumb tables to a file, where a table is as binary file containing logical group of parameters that are managed as a named entity. Tables can then be used to perform actions on cFS applications. It should be noted that cFE services dont use tables, and therefore can be built without table support.

2.3.3 Important apps

There are many applications on the Software Bus, in this section we will go more in depth on some of the more important apps for the project. Figure 2.11 shows the entire Software Bus.

Kit Command Ingest

Command Ingest (CI) receives commands from an external source, and sends them on the Software Bus. In this case it is COSMOS and commands are CCSDS packets.

Kit Telemetry Output

Telemetry Output (TO) receives telemetry packets from the software bus and send them to an external source (COSMOS).

Kit Scheduler

Scheduler (SCH) sends software bus messages at pre-defined intervals from tables, apps often use these scheduled messages as wake-up signals. But can also be used to do actions in a synchronized manner.

Stored Commands

Stored Commands (SC) executes onboard commands sequences, using tables and files.

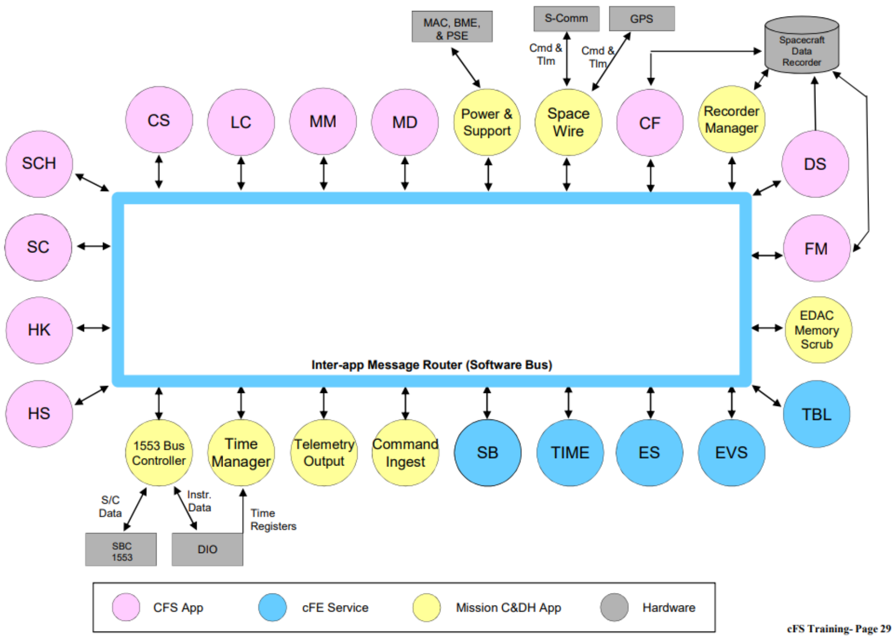


Figure 2.11: The Software Bus [cFS21]

CHAPTER 3

Analysis

3.1 The scheduling service

A scheduling service for flight software can have many functionalities and it need to work with the rest software system. Things such as memory, errors, inconsistencies, functionalities, etc. needs to taken into consideration. Maybe they are not needed, not feasible in the software, or simply left out for various reasons. In this project only some of these aspects are being implemented, where other aspect such as, error handling, memory and optimization are not prioritized. The ESA's Packet Utilization Standard [ESA20] define some of these functionalities and their formats. The functionalities the user can request, explained by the ESA's Packet Utilization Standard [ESA20], are:

- Enable the scheduling of all, or a group of, the packets in the command scheduler.
- Disable the scheduling of app, or a group of, the packets in the command scheduler.
- Add packets to the command scheduler.
- Remove packet(s) from the command scheduler (also used when packets a due for release).

- Time shift packet(s) in the command scheduler. Either a single packet, all packets or a group of packets.
- Report packet(s) in the command scheduler.
- Report the status of the command scheduler.

It is also described that inconsistencies should not exist when using these functions. It is also the user(s) responsible for ensuring that inconsistencies which only can be detected during the scheduling functionality does not occur. Other inconsistencies that would be created when performing a scheduling functionality should be refused by the scheduler.

The command scheduler application maintains a timeline, that contains the packets together with attributes used for scheduling. The attributes explained by the ESA's Packet Utilization Standard [ESA20] are the following:

- A group (also referred to as a sub-schedule), which is a mechanism for packets that enable them to be controlled together or interlocked to others in that same group (*interlock* meaning the release of the packet is preconditioned on an earlier released packet).
- The number of the interlock to be set by the packet, if any. The success or failure with the packet is associated with that interlock.
- The number of the interlock on which the release of the packet is dependent, if any.
- Whether the release of this Packet is dependent on the success or on the failure of the packet with which it is interlocked.
- Either the absolute on-board time at which the packet is released to its destination application process or a relative time. See section 3.1.3.

The scheduling service should also have access to other information needed for proper execution:

- Get the amount of elements currently in the command scheduler.
- Get the amount of groupings in the command scheduler.
- Get the amount of interlocks in the command scheduler.
- Get a list of the app's the scheduling service can release command packets to.

- Get a list of sources that the scheduling service can receive command packets from that needs to be scheduled.

This information can be used for things such as error detecting and reporting.

3.1.1 Command packet release status

It is the scheduling service that should maintain information to determine if a command packet should be released at its due time.

The release status can be changed by the user when requested to either disable or enable it. This can be a request to either all or a subset of the command packets. Hence a release status can either be *enabled* or *disabled*.

The release status *enabled* is used if the release of the packet is enabled from the command scheduler, or from the sub-schedule (group) in which the command packet belongs to and to the destination app of the packet. Otherwise the release status should be *disabled*, from this a decision table can be made, see figure 3.1.

This functionality should however not be highly prioritised, since the software bus is connectionless, it might be difficult to determine if the destination application process is enabled or disabled. Therefore not supporting the release status, would be the same as only having the status *disabled*.

Schedule	Sub-schedule	Destination application process	Release status
D(isabled)	E(nabled)	E	D
D	D	E	D
D	E	D	D
D	D	D	D
E	E	E	E
E	D	E	D
E	E	D	D
E	D	D	D

Figure 3.1: Decision table for packet release status [CCS03]

3.1.2 Interlock Status

The result when executing a command packed, which sets an interlock, should be used in order to determine the interlock status of other command packets which are due for release. The result of executing a command packet can either be a *success* or a *failure*.

When a command packet has an interlock set and is due for release then 2 things can happen:

- If the release status is *disabled* or the interlock status is *locked*, the command packet is not released and the execution result is set to *failure*.
- Otherwise the command packet is released and the execution result is unknown until the execution of that command packet is complete.

The scheduling service must therefore indicate to the destination app that the released command packet sets an interlock, so when the command packet execution is completed in the destinated app, it can report back to the scheduling service. The report is not received after some maximum execution time, the result is set to *failure*.

When a given command packet is ready for release and the release depends on an interlock, the interlock status is then evaluated as follows:

- If the release of a command packet depends on the *success* of the command packet setting the interlock and the command packets execution result attached to the interlock is *failure*, then the interlock is to be *locked*.
- If the release of a command packet depends on the *failure* of the command packet setting the interlock and the command packets execution result attached to the interlock is a *success* then the interlock is to be *locked*.

3.1.3 Inserting packets

Inserting packets to the scheduler is one of the most important functions, otherwise the whole scheduler wouldn't work properly. Therefore it is important that the packets are inserted correctly in the scheduler.

ESA's Packet Utilization Standard describes one of the ways the packet should be formatted:

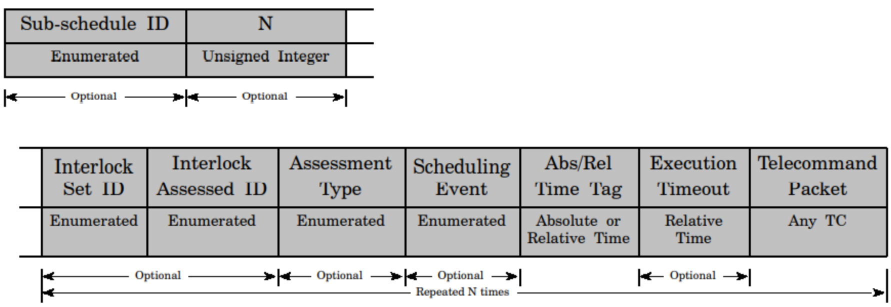


Figure 3.2: Format for inserting packets [CCS03]

Sub-schedule ID:

The identification of the sub-schedule with which the following command packets are associated. This ID cannot take the value 0, and the field should be systematically omitted if the service does not support grouping (sub-scheduling).

N:

N is the amount of command packets to be inserted in the scheduler. The field should be systematically omitted if the service only supports insertion of a single command packet at a time.

Interlock Set ID:

The identification of the interlock to be set by the command packet (0 if no interlock is set). The status of the interlock is to be determined by the *success* or *failure* of the command packets execution. The field should be systematically omitted if the service does not support interlocking.

Interlock Assessed ID:

The identification of the interlock on which the release of the command packet is dependent (0 if no interlock assessed). The field should be systematically omitted if the service does not support interlocking.

Assessment Type:

Determines whether the release of the command packet is dependent on the *success* or the *failure* of the command packet which is interlocked.

Success taking value 1: release if interlocked command packet was successful.

failure taking value 0: release if the interlocked command packet failed at execution.

The parameter should not be present if the packet is not interlock dependent (or the service does not support interlocking).

Scheduling Event

This field determines whether what time should be used, this can be either *absolute* time or *relative* time, see 3.1.3.

If the Scheduling Event has value 0, then absolute time is used.

If the Scheduling Event has value 1, then the command packets release time should be relative to the time which the command scheduler is enabled.

If the Scheduling Event has value 2, then the command packets release time should be relative to the time of notification to the *success* or *failure* of the command packet which sets the interlock.

This field should be systematically omitted if *relative* time is not supported.

Time

A packet can be scheduled in *absolute* time and *relevant* time.

- **Absolute time:** If a packet is scheduled in absolute time, then the packet should be sent to its destination at that time.
- **Relative time:** If a packet is scheduled in relative time, then it should sent to its destination at a time that is the give relative time added to the time of the event indicated from the Scheduling Event.

This provides the user two ways to schedule a packet to a given time. The packet should therefore also have an indication of what time is wanted to be used. Both *absolute* - and *relative* time, should still be represented in such a format, that they can be compared to the time formats explained in 2.3.2.

Execution Timeout

This field is used to tell the latest time that the execution of the command packet is expected to be completed. The field is only used if the an interlock is set. If an execution completion callback is not received within the timeout window, the command packet should be seen as *failed*, for the purpose of handling the interlock.

Possible errors inserting packets

If a command packet is added with *relative* time, with respect to an interlock, it should be *linked* to the latest command packet that was added to the scheduler which sets the interlock. Each command packet that the scheduler receives should be checked for errors, if no errors the packet is added to the scheduler. Errors can occur if:

- The scheduler is full.
- The destination application process of the command packet is invalid.
- The time specification refers to the past.
- The time specification is not supported by the service.
- The command packet is interlock dependent and its release time falls within the execution window of its interlocking command packet.
- The sub-schedule ID or one of the interlock ID's exceeds its max value.
- The Interlock ID's are equal.
- The command packet has an *interlock* relative time and no command packets is added since the last resetting of the scheduler sets the interlock.

3.1.4 Deleting command packets

Deleting packets from the command scheduler, can be very advantageous. Not only can it be used by the user to remove a packet they have scheduled, but it also used when the scheduled packet needs to be released. Deleting packets can also be used to implement other functions, such as changing the time a packet should be delivered.

ESA's Packet Utilization Standard [ESA20] has explained deletion of command packets. The scheduler should refuse to delete an interlocking command packet unless all its interlocked packets has been deleted, unless they are also deleted in that same deletion. If an error should occur nothing should be deleted.

N	Application Process ID	Sequence Count	Number of Telecommands
Unsigned Integer	Enumerated	Unsigned Integer	Unsigned Integer
Optional	Repeated N times		

Figure 3.3: Deleting packets format [CCS03]

N:

This field should be systematically omitted if the service does not support *scatter delete*, meaning deletion of packets that are scattered. (i.e. N = 1).

Sequence Count

The sequence count of the first command packet which should be deleted. The sequence count and APID (application Process ID), is used as a uniquely identifier for the packets. Hence other parameters might as well be used if that also uniquely identify a packet. As explained in 2.3.2, the cFE already has ID's to uniquely identify a message.

Number of Telecommands (command packets):

The number of successive command packets which should be deleted, that satisfy the attributes which uniquely identifies a packet.

3.1.5 Time-shifting command packets

Time-shifting command packets can be time-shifting all packets or a selected amount (including a single time-shifting). The packet should therefore contain a time offset, that is either a positive or negative relative time between the old and the new time, which is used to modify the time in the scheduler by adding the offset. If the time of a command packet in the scheduler that are not yet known, then it shouldn't be time-shifted. This can occur if the time is relative to a scheduling event, which has not yet occurred.

Time-shifting is an important functionality as it lets the user change the scheduled time that they have inserted.

3.2 Scheduling possibilities in cFS

Many of the explained functionalities, from ESA's Packet Utilization Standard, seem to be able to be implemented into the cFS. However functionalities such as *interlocking*, see 3.1.2, might be difficult to implement, since the destinated app needs to report back to the scheduler. Because the messages are being send connectionless it is unknown whether this is possible to implement.
For this to be implemented the method sending the messages on the software bus needs to return some result, or a whole new message needs to be send back, which tells if the operation of the released packet was a *success* or a *failure*.

The parameters to uniquely identify a packet, should also be changed to fit the cFS. Moreover it might be relevant to be able to differentiate the same packet on different times, hence introducing a time attribute when searching for packets that are in the scheduler.

3.3 Prioritized goals

A prioritized list of goals is created for this project, this serves as an overview of whats important to implement first.

The goal is to create a timeline, where the user can put in packets that will then be released at the scheduled time. The timeline should not create any inconsistencies or cause errors in the software. The user should also be allowed to operate the timeline using different functionalities. From this prioritised goals can be listed:

- Modify the cFS to have a timeline where packets can be stored.
- Create a packet format which is used so the user can request scheduling.
- Allow the user to schedule command packets.
- Allow of inserting and deleting packets in the timeline. At this point the user should be able to schedule a packet and release it correctly at its due time.
- Allow for more advance functionalities, such as; grouping, time shifting, etc.
- Optimizations, such as; handling user errors, memory optimization, efficiency, etc.

Chapter 4

Design

4.1 Sending and containing packets

The Software Bus can send and receive messages.. The Software Bus also already have a scheduler that send messages at pre-defined intervals, see 2.3.3. The command scheduler wanted to be implemented in this project, allows for command packets to be send to the software bus and then contained in an scheduling application, that holds them until their specified release time.

To achieve this, the command packets that are requested to be scheduled, need a packet format allow the packet to be send to the scheduler, together with attributes used for scheduling. The packet format can be implemented in different ways:

4.1.1 Secondary Header approach

Since there already exist a secondary header, used partly for command packets, a solution could be to expand the secondary header to contain relevant information for scheduling. This would mean that every packet with a secondary header would have more information that might not be used, if it isn't used for scheduling. The upside to this is that it might be the easier solution since the second header is probably generically coded and can be changed without causing too much trouble.

This also means that the packet wanted to be scheduled needs to be re-routed to the scheduler app, so it can be send out on the bus at the proper time. It will also require less from the user itself, since they only need to select the attributes wanted for scheduling.

4.1.2 Third header approach

This approach similar is similar to expanding the secondary header, instead the creation of a third header might be a good solution. The third header then contains the relevant information that is needed to schedule the packet. This also means that every command packet with a secondary header does not necessary have a third header, therefore some kind of flag is needed to indicate the presence of the third header. Is unknown how difficult it is to introduce a new header.

Like with the secondary header approach the packet needed to be scheduled needs to be re-routed.

4.1.3 Payload approach

In this approach does not modify the headers. Instead the packet is sent to the scheduling app, where the payload is the message wanted to be scheduled, together with needed attributes. This way of having the message wrapped in another message that is being send directly to the scheduling app, may be the better solution since no re-routing is needed. The downside is that it require the user to correctly give the payload, which could be tedious.

This approach also have the option to abstract away from the correctness of the payload and instead have the user be responsible for giving the correct payload.

Storing packets

There are multiple ways of storing packets, in facts CFS already provide *scored commands* that uses files to store commands in, see 2.3.3. The issue with storing the packets in files, is that they don't provide a lot of functionalities, and it can be disadvantageous to reload entire files to update the scheduling. Therefore another solution is needed.

One data structure that would allow for functionalities is a linked list. Compared to other data structures a linked list can shrink and expand at runtime, which results in reduced memory waste.

A downside to using a linked list is that a linked list contains more memory since it has data fields in its construction. But having data fields might prove to be an advantage, since different attributes are needed for certain functionalities.

Buffer

Since a packets payload can have various length it is important to store the whole packet, therefore a buffer should be used to temporary store the packet in the linked list. The requirement for the buffer is that it should be able to contain the largest packet that can be created.

4.2 Design specifications

The development of this implementation will disregard the user interface and work on the foundation of a command scheduler. Therefore the chosen packet format for sending a packet is the *payload approach* see 4.1.3. This means that it is assumed to be the correct playload given to the scheduler.

The scheduler should be created as a separate application in the cFS, that then holds and operate the timeline. The timeline should be represented as a doubly linked list, as it provides the easiest way to create a timeline that allows for most functionalities introduced in chapter 3. The timeline should be designed so the elements in the doubly linked list also contains attributes needed for scheduling.

Operations made on the timeline should be allowed through sending commands to the scheduler, using specific function codes. Therefore packet formats should be created for each individual function, to determine the operation and secure it comes in the correct format. Doing operations should be designed in such, that they do not create inconsistencies.

Checking if packets are due for released should be done at a continuous constant interval, separated from the operations. Hence threading should be introduced, so doing operations and releasing packets for due time are separated, on individual threads. Figure 4.1, shows a representation of how scheduler should work. Depending on how the Software Bus send messages, the packet might need be deleted from the scheduler. after the packet handled at the destinated app.

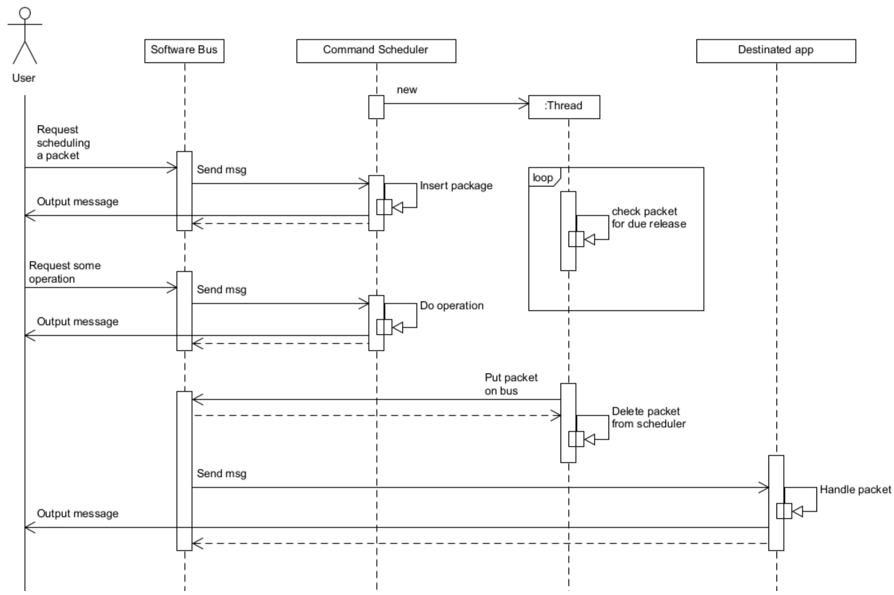


Figure 4.1: UML diagram

Because implementing a command scheduler is a big project, the workflow is be to achieve the listed prioritized goals, see 3.3, starting with the top and moving down.

Chapter 5

Implementation

5.1 Getting started

Creating a new app

OpenSatKit has a helpful tool to create a new app with everything needed to get started. Here the app will be added to the make file and cFE startup. When creating a new app, using the tool, it already has some basic packets that can be send.

Sending packets

Sending packets have been implemented using the approach refereed to as *payload approach*.

All packets are being sent using a *Message pointer* (MsgPtr) which the Software Bus takes as an argument, when sending a new message. The message is defined as a *union* allowing to access the message as different data types.

```
    /**< \brief Generic Software Bus Message Type Definition */
    typedef union {



        CCSDS_PriHdr_t      Hdr;   /**< \brief CCSDS Primary Header # CCSDS_PriHdr_t */
        CCSDS_SpacePacket_t SpacePacket;
        uint32              Dword; /**< \brief Forces minimum of 32-bit alignment for this object */
        uint8               Byte[sizeof(CCSDS_PriHdr_t)];   /**< \brief Allows byte-level access */

    }CFE_SB_Msg_t;
```

Listing 5.1: Generic Software Bus Message

Listing 5.1 shows the Generic Software Bus Message (NOTE: The SpacePacket is also only a primary header), to get the packet, or for example the secondary header, a *struct* is used to format it. This is possible since the whole packet is one sequence in memory, so the application formats the packet to what the headers indicate it to be. The length of the packet is used to validate that the packet is the correct length to be formatted.

In the scheduler a scheduled packet can come in various length, therefore validating the length in the same way is not an option, so for now it is not handled (it is assumed to be correct). Because the length of the scheduled packet can be different from whatever packet might be scheduled, the *struct* used for formatting a packet used for a scheduling request, contains an unfinished array.

```
typedef struct
{
    CCSDS_SpacePacket_t  SpacePacket;   /**< \brief Standard Header on all packets  */
    CCSDS_CmdSecHdr_t    Sec;
    uint8  Time[CCSDS_TIME_SIZE]; //32 bit seconds 16 bit subseconds
    /* payload */
    uint8 playload[];

} CMD_SCH_SchedulePacketCmd_t;
```

Listing 5.2: Schedule packet format

Listing 5.2 shows the packet format used for a inserting a packet. The SpacePacket and Secondary header are whats used to send the message to the scheduling service. The Time attribute is used for indication the time it should be scheduled. The array named *payload* is the packet that wants to be scheduled. It is also in this these structs that implementations for other attributes, explained in the ESA's Packet Utilization Standard [ESA20], should be given, for now it only takes an absolute time and a single packet, see 3.1.3.

The benefit of using this approach is that there is no modification needed in the CCSDS format, since everything is in the payload. The downside is the a user has the responsibility of setting the correct payload, hence a user interface is needed so the user can correctly choose the payload. This however has not yet been implemented.

Similar *structs* are used to format the packets when requesting operations using the different functionalities.

```
typedef struct
{
    CCSDS_SpacePacket_t  SpacePacket;   /**< \brief Standard Header on all packets  */
    CCSDS_CmdSecHdr_t    Sec;
    uint8  Time[CCSDS_TIME_SIZE]; //32 bit seconds 16 bit subseconds
    uint32 MsgId;
    uint32 CmdFunc;

} CMD_SCH_DeletePacketCmd_t;

typedef struct
{
    CCSDS_SpacePacket_t  SpacePacket;   /**< \brief Standard Header on all packets  */
    CCSDS_CmdSecHdr_t    Sec;
} CMD_SCH_ResetSchedulerCmd_t;

typedef struct
{
    CCSDS_SpacePacket_t  SpacePacket;   /**< \brief Standard Header on all packets  */
    CCSDS_CmdSecHdr_t    Sec;
} CMD_SCH_GetSizeCmd_t;

typedef struct
{
    CCSDS_SpacePacket_t  SpacePacket;   /**< \brief Standard Header on all packets  */
    CCSDS_CmdSecHdr_t    Sec;
    uint8  Offset[CCSDS_TIME_SIZE]; //32 bit seconds 16 bit subseconds

} CMD_SCH_ShiftAllPacketsCmd_t;

typedef struct
{
    CCSDS_SpacePacket_t  SpacePacket;   /**< \brief Standard Header on all packets  */
    CCSDS_CmdSecHdr_t    Sec;
    uint8  Time[CCSDS_TIME_SIZE]; //32 bit seconds 16 bit subseconds
    uint32 MsgId;
    uint32 CmdFunc;
    uint8  Offset[CCSDS_TIME_SIZE]; //32 bit seconds 16 bit subseconds

} CMD_SCH_ShiftPacketCmd_t;
```

Listing 5.3: operation packet formats

5.2 The timeline

5.2.1 Elements

The linked list needs to consist of elements with a set of pointers, to make up the timeline, this together with the packet and attributes needed for the scheduler should also be included:

```c
// List element
struct Node
{
    CFE_TIME_SysTime_t time;
    struct Node* next;
    struct Node* prev;
    uint8 BufMsg[];
};
typedef struct Node* NodePtr;
```

Listing 5.4: Elements in the list

The structure of a element, see 5.4, is made with the keywords *struct* and *typedef*. Struct allows for defining a new type with variables, and typdef is used to give that type a new name that can then be referred to. It is important to notice that the *Node* struct has an integer array with a incomplete array, that allow packets to have various sizes when stored.

An element in the linked list also has 2 pointers, making it a so called *doubly linked list*. Using to pointers allows for looking back in the timeline which could be an advantage in later developments. The linked list should also be sorted, and kept sorted. This will be advantageous when checking for packets that are due for release, since all packets don't need to be check if the list is sorted. The scheduler stores the starting element, called *head*, which is used as the starting point for the timeline.

The datatype *CFE_TIME_SysTime_t* contains the time of the system, which is described with seconds and subseconds. Time formats explained in 2.3.2, is formatted to this *CFE_TIME_SysTime_t* depending on if TAI or UTC is selected. The cFE also provides functions to get the current time of the system, and also compare two times. This is useful for checking if a packet is due for release, and also to keep the timeline ordered. The *struct* has time stored in an array, therefore it is formatted into the *CFE_TIME_SysTime_t* when put in the timeline.

To uniquely identify a packet, the message id and function code is used. Hence an element in the scheduler can be checked using those attributes, together with the time of the packets scheduled time. Using the packets scheduled time to identify an element in the timeline, allows the user to distinguish the duplicates of packets at different times.

5.3 Functionalities

The app receives command packets that contain a function code in the secondary header, this is used to determine what functionalities the user has requested. Some of the functionalities, as explained in chapter 3, has been implemented as follows:

Inserting command packets

Inserting commands in the timeline needs to be done is such a way that inconsistencies does not happen, meaning it should keep the timeline sorted and assign pointers correctly. Since the timeline is constructed as sorted doubly linked list, an element in the list might have to be placed in the middle of the list. Therefore it needs to validate that the scheduled release time, for a command packet, is larger than the current time. If it is not a larger time it will reject scheduling the packet. The function *CFE_TIME_Compare* compares two times, which can return 3 results:

- -1: The first time is considered to be before the second time.
- 0: The two times are considered equal.
- 1: The first item is considered to be after the second time.

Insertion is therefore done by traversing through the linked list, inserting the element either when; the next element has a time later that the one trying to be inserted, or an end is reached.

```
1  // insert a packet in the timeline
2  void InsertPackage(CFE_SB_MsgPtr_t MsgPtr, CFE_TIME_SysTime_t time, NodePtr* head){
3      NodePtr node;
4
5      node = (NodePtr) malloc(sizeof(struct Node)+ CFE_SB_GetTotalMsgLength(MsgPtr));
6      node->time = time;
7      node->next = NULL;
8      node->prev = NULL;
9      memcpy(node->BufMsg, MsgPtr, CFE_SB_GetTotalMsgLength(MsgPtr));
10
11     NodePtr current;
12     if (*head == NULL){
13         *head = node;
14     }
15
16     else if (CFE_TIME_Compare(time, (*head)->time) <= 0){
17         node->next = *head;
18         node->next->prev = node;
19         *head =  node;
20     }
21     else {
22         current = *head;
23         while (current->next != NULL && CFE_TIME_Compare(current->next->time, node->time) == -1){
24             current = current->next;
25         }
26         node->next = current->next;
27         if(current->next != NULL){ //if it was inserted in middle
28             node->next->prev = node;
29         }
30         current->next = node;
31         node->prev = current;
32     }
33     CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,"CMD_SCH App: Inserted command at time: %d", time.Seconds);
34     return;
35
36 }
```

Listing 5.5: InsertPackage

Important to notice line 5 to 9 is for correctly allocating a new element (node) and also copying the packet into the element. The element needs to be as large as the structure of the element, plus the length of the packet trying to be scheduled (allocating size of array needed to store the packet). The packet is copied from the scheduled packet into the element, so it is stored in the timeline.

Deleting command packets

Deleting an element in the timeline needs to be done so no inconsistencies occur in the doubly linked list. Moreover it needs to *free* the allocated memory, allocated when the element got inserted, see 5.3.

```
/* Delete a package in the timeline */
void DeletePackage(NodePtr elem, NodePtr* head){
    if (*head == NULL || elem == NULL){ //end elem
        return;
    }
    if (*head == elem){
        *head = elem->next;
    }
    if (elem->next != NULL){
        elem->next->prev = elem->prev;
    }
    if (elem->prev != NULL){
        elem->prev->next = elem->next;
    }
    free(elem);
    elem = NULL;
    return;
}
```

Listing 5.6: Deleting command packet

As it can be seen on listing 5.6, the function *free* is used to free the memory where the package is stored, allowing the memory to be used for something else. Released packets should only removed when the packet is safely on the bus, or the operation is finished. Memory fragmentation could happen if this does not happen correctly.

Time-shifting command packets

Time-shifting a command packet should keep the timeline consistent. The functionality is implemented by extracting packet and deleting the element in the timeline, then shifting the time and then inserting it then correctly. However time-shifting all packets cannot result in the timeline becoming unsorted, since every packet is shifted by the same offset. Therefore deleting and inserting packets are not needed.

```
void ShiftAllTime(CFE_TIME_SysTime_t offset, NodePtr* head){
    NodePtr curr = *head;
    while (curr != NULL){
        curr->time.Seconds += offset.Seconds;
        curr->time.Subseconds += offset.Subseconds;
        curr = curr->next;
    }

}
```

Listing 5.7: Time-shift all

```c
//shift the release time of one element in the timeline
void ShiftTime(CFE_TIME_SysTime_t time, uint16 cmdCode, uint16 id,
     CFE_TIME_SysTime_t offset, NodePtr* head){
    NodePtr elem = FindElem(time, cmdCode, id, head);
    if (elem != NULL){
        uint16 length = CFE_SB_GetTotalMsgLength(elem->BufMsg);
        uint8 msg[length];
        memcpy(msg, elem->BufMsg, length);
        DeletePackage(elem, head);
        offset.Seconds += time.Seconds;
        offset.Subseconds += time.Subseconds;
        InsertPackage(msg, offset, head);

        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                "CMD_SCH: Changed time from: %d, To: %d\n", time.Seconds,
                    offset.Seconds);

    }
    else{
        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                "CMD_SCH: Shift-op, packet not found!");
    }

    return;
}
```

Listing 5.8: Time-shift

As it can be in the listing 5.8, the time-shifting is implemented using deletion and insertion. The stored packet is also needed to be extracted into a local array before it can be inserted again. It also makes use of a helper function *FindElem* which return the node pointer containing the command packet, see 5.9.

```c
// find a packet in the timeline
NodePtr FindElem(CFE_TIME_SysTime_t time, uint16 cmdCode, uint16 id, NodePtr*
      head){

    NodePtr curr = *head;
    while (curr != NULL){
        if (CFE_TIME_Compare(time, curr->time) == 0){
            if(CFE_SB_GetCmdCode(curr->BufMsg) == cmdCode && CFE_SB_GetMsgId(
                 curr->BufMsg) == id){
                CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID,
                     CFE_EVS_INFORMATION,
                    "CMD_SCH: Found elem at time: %d", time.Seconds);
                return curr;
            }
        }
        curr = curr->next;
    }
    return NULL;
}
```

Listing 5.9: Find element

5.4 Threading and Race conditions

Checking when packets should be released, is done with a continuous check in a loop each second, to make this run parallel with the app is it separated on another thread. This means that the scheduling app and checking if packets are due for release are running parallel, therefore it is important that operations does not create race conditions. Example could be inserting a packet while another is getting released, creating inconsistency in the pointers making up the timeline, and in result creating errors (e.g. having packets that no pointer is pointing to, making it unreachable).

5.4.1 Threading

The cFE already has methods that are used to create thread, refereed to as tasks. Their method derive from pthread, one of them is *CreateChildTask()*. CreateChildTask() creates a task under an already existing application, which takes a few arguments. Some of the arguments are a stackpointer and a stacksize, which indicates where the stack is and how large the stack is for the task. Here default settings is used to allocate a stack and stack size.

5.4.2 Race conditions and Mutual Exclusion

Race conditions can occur as a result of two treads operating the same memory (the timeline). To avoid race conditions only one thread must be operating on the timeline, for example releasing and deleting packet must be done at the same time.

The use of semaphores are able to create mutual exclusion, 1 binary semaphore to be exact. Because only 1 operation can be done at a time, whether its checking for packets that needs to be released or doing certain operations on the timeline. Hence before doing any operation that operate or traverse the time-line, needs to acquire the semaphore, releasing it when finished with the operation. Another thread wanting the semaphore that's locked, must therefore wait until it is available .

The OSAL have functionalities to create such a semaphore, *OS_ MutSemCreate*, *OS_ MutSemTake* and *OS_ MutSemGive* are used. Example of this can be seen in listing 5.10.

```
// Runs in a different thread
void* CheckSchedule(void* arg){
    while(SchedulerEnabled = true){

        sleep(1);
        OS_MutSemTake(SchedulerSem);
        NodePtr curr = head;
        if(curr != NULL){
            CFE_TIME_SysTime_t currTime = CFE_TIME_GetTime();
            if ( CFE_TIME_Compare(curr->time, currTime) <= 0){

                CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID,
                    CFE_EVS_INFORMATION,"CMD_SCH App: Execute command at
                    time: %u , current time: %u", curr->time.Seconds,
                    currTime.Seconds);
                SendScheduledMsg(curr, &head);

            }
        }
        OS_MutSemGive(SchedulerSem);

    }
    CFE_ES_ExitChildTask();
    return NULL;
}
```

Listing 5.10: CheckSchedule

Listing 5.10 shows the separate thread that checks for packets that are up for release. If the scheduler is enabled it should check for due release packets, since enable and disable has not been implemented, it is set to true on startup. When the disabled the thread should close, using the *ExitChildTask()* function, for now the scheduler is always enabled.

The system is limited to checking at 1 Hz, therefore the thread needs to sleep for 1 second before checking the timeline, this is inefficient and could be optimized. A solution to look into in further developments, would be to use interrupts that signals when a packet is due for release. However in this implementation it will continuously check if a packet is due for release, using *CFE_ Time_ Compare*.

The method sends the message on the software bus using the function *SendScheduledMsg()*, see 5.11. The message is removed from the timeline, when it is successfully sent onto the software bus. Currently if an error occurs it is not handled, e.g the message is too big, and is for now removed as if it was successful. The errors that could occur from sending the message on the software bus, should have avoided by a correctly given packet from the user, but error handling should still be implemented in further developments.

```c
/* "pop" and element */
void SendScheduledMsg(NodePtr elem, NodePtr* head){

    // see if operation was success,
    // removes packet from timeline either way - for now
    if (CFE_SB_SendMsg(elem->BufMsg) == CFE_SUCCESS){
        DeletePackage(elem, head);
    }
    else {
        CFE_EVS_SendEvent(CMD_SCH_NOOP_INF_EID,CFE_EVS_ERROR, "scheduled
            message operation unsuccessful");
        DeletePackage(elem, head);
    }
}
```

Listing 5.11: SendScheduledMsg

5.5 Testing

Testing the command scheduler has been done in two ways:

First way is writing tests, that test functionalities and the timeline. For example inserting and removing elements, can be tested by itself on separate code with written unit testing. These tests don't involve OpenSatKit, but rather the correctness of the functionalities and structure of the command scheduler. See appendix A, for the script used for testing.

The other way is manual testing, where the cFS is being tested, this include packet, formats, sending packet correctly, threading, etc. Because there was not found a good solution to do unit testing inside the OpenSatKit, the test was done manually. This means that in consist of; writing code, build the cFS, run the cFS and then see the outcome, using their output print functions. This process is ineffective, but it can be very flexible in testing different results. Testing for performance and limitations (e.g. the maximum amount of packets the scheduler can hold) has not been done. Figure 5.1, shows the terminal output for a test. The test consist of scheduling a no-op packet, where the time gets shifted with an offset of +10 seconds.

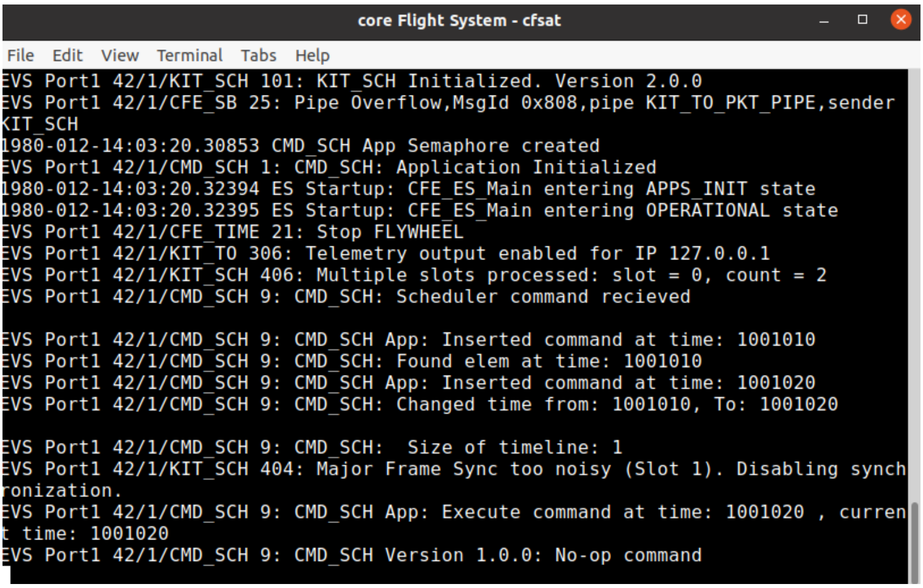


Figure 5.1: Terminal output for scheduling test

Chapter 6

Conclusions

It can be concluded that it is possible to implemented a command scheduler in cFS with more functionalities than what it currently has. Even though the foundation has been implemented for the command scheduler, some functionalities explained are yet not supported. In the future more functionalities can be implemented, together with porting and making a user interface.

The goals that have been achieved:

- Modifying the cFS to have a timeline where packets can be stored.
- Create a packet format which can be used to schedule commands.
- Allow the user to schedule command packets.
- Allow for inserting and deleting scheduled command packets.
- Allow for time-shifting packets, and let the user request to see the amount of element currently in the scheduler.
- Release packets at it due time to the destinated app.
- Avoid race conditions and other inconsistencies.

It is not easy to implement new functionalities in the cFS, as it is a very big and well put together system. There is many part being interconnected together, and normally it would be a team of multiply people designing and implementing such a commands scheduler. There is a big aspect of error detection, efficiency, memory, etc. that is not been thoroughly addressed, as it was deemed out of the scope of this thesis. However these things are very important and should be looked further into in future developments.

Appendix A

Code for testing

```
#include "/home/simon/Documents/OpenSatKit-master/cfs/osal/src/os/inc/
    common_types.h"


#ifndef _cmd_sch_tl_h_
#define _cmd_sch_tl_h_

typedef struct{
    int Seconds;
    int Subseconds;
}myTime;

struct Node
{
    uint8 time;
    struct Node* next;
    struct Node* prev;
};
typedef struct Node* NodePtr;

int GetTlSize_();
bool validate(NodePtr* head, uint8 expectedResult[], uint8 size);
NodePtr FindNode_(uint8 time, NodePtr* head);
void ShiftTime_(NodePtr node, uint8 newTime, NodePtr* head);
void shiftTimeAll_(uint8 offset, NodePtr* head);
void ResetTimeline_(NodePtr* head);
void RemovePackage_(NodePtr del, NodePtr* head);
void InsertPackage_(uint8 time, NodePtr* head);


#endif
```

Listing A.1: Header file for testing

```
/* File: cmd_sch_tl.c
**
** Pupose: Create and manage a timeline consiting of packages,
** that should be send out on the softwarebus later.
*/

#include <string.h>
#include <stdio.h>
#include "cmd_sch_tl.h"
#include <stdlib.h>



int main(int argc, char *argv[]){
    TestSuite();
}
void TestSuite(void){
    test1();
    test2();
    test3();
    test4();
    test5();
    test6();


}
bool validate(NodePtr* head, uint8 expectedResult[], uint8 size){
    NodePtr curr = *head;
    if (GetTlSize_(head) != size){
        return false;
    }
    for (int i=0; i < size; i++) {
        if (curr->time != expectedResult[i]){
            printf("Failed comparing element: %d,%d \n", curr->time,
                 expectedResult[i]);
            return false;
        }
        curr = curr->next;
    }
    return true;
}
void test1(void){
    NodePtr head = NULL;
    uint8 t1 = 1;
    uint8 t2 = 2;
    uint8 t3 = -3;
    uint8 t4 = 4;
    uint8 t5 = 5;

    InsertPackage_(t2, &head);
    InsertPackage_(t1, &head);
    InsertPackage_(t5, &head);
    InsertPackage_(t3, &head);
    InsertPackage_(t4, &head);
    InsertPackage_(t2, &head);
    prettyPrinter(&head);
    uint8 expectedReult[] = {-3,1,2,2,4,5};
    if (validate(&head, expectedReult, 6)){
        printf("test1 passed\n");
    }
    else {
        printf("test1 failed\n");
    }
    ResetTimeline_(&head);
}
```

```c

void test2(void){
    NodePtr head = NULL;
    uint8 t1 = 1;
    uint8 t2 = 2;
    uint8 t4 = 4;

    InsertPackage_(t2, &head);
    InsertPackage_(t1, &head);
    InsertPackage_(t4, &head);
    NodePtr toRemove = FindNode_(t1, &head);
    RemovePackage_(toRemove, &head);
    prettyPrinter(&head);
    uint8 expectedReult[] = {2};
    if (validate(&head, expectedReult, 1)){
        printf("test2 passed\n");
    }
    else {
        printf("test2 failed\n");
    }
    ResetTimeline_(&head);
}

void test3(void){
    NodePtr head = NULL;
    uint8 t1 = 1;
    uint8 t2 = 2;


    InsertPackage_(t2, &head);
    InsertPackage_(t1, &head);
    ResetTimeline_(&head);
    prettyPrinter(&head);
    uint8 expectedReult[] = {};
    if (validate(&head, expectedReult, 0)){
        printf("test3 passed\n");
    }
    else {
        printf("test3 failed\n");
    }
    ResetTimeline_(&head);
}

void test4(void){
    NodePtr head = NULL;
    uint8 t1 = 1;
    uint8 t2 = 2;
    uint8 t5 = 5;


    InsertPackage_(t2, &head);
    InsertPackage_(t1, &head);
    InsertPackage_(t5, &head);
    NodePtr toShift = FindNode_(t1, &head);
    ShiftTime_(toShift, 6, &head);
    prettyPrinter(&head);
    uint8 expectedReult[] = {2,5,6};
    if (validate(&head, expectedReult, 3)){
        printf("test4 passed\n");
    }
    else {
        printf("test4 failed\n");
    }
    ResetTimeline_(&head);
}
```

```

void test5(void){
    NodePtr head = NULL;
    uint8 t1 = 1;
    uint8 t2 = 2;
    uint8 t5 = 5;


    InsertPackage_(t2, &head);
    InsertPackage_(t1, &head);
    InsertPackage_(t5, &head);
    NodePtr toShift = FindNode_(t1, &head);
    ShiftTimeAll_(5, &head);
    prettyPrinter(&head);
    uint8 expectedReult[] = {6,7,10};
    if (validate(&head, expectedReult, 3)){
        printf("test5 passed\n");
    }
    else {
        printf("test5 failed\n");
    }
    ResetTimeline_(&head);
}

void test6(void){
    NodePtr head = NULL;
    uint8 t1 = 1;
    uint8 t2 = 2;


    InsertPackage_(t2, &head);
    InsertPackage_(t1, &head);
    if (GetTlSize_(&head) == 2){
        printf("test6 passed\n");
    }
    else {
        printf("test6 failed\n");
    }
}

void prettyPrinter(NodePtr* head){
    NodePtr curr = *head;
    while (curr != NULL){
        printf("%d ",curr->time);
        curr = curr->next;
    }
    printf("\n");
}
int GetTlSize_(NodePtr* head){
    int count = 0;
    NodePtr curr = *head;

    while (curr != NULL){
        count++;
        curr = curr->next;
    }
    return count;
}

NodePtr FindNode_(uint8 time, NodePtr* head){
        NodePtr curr = *head;
        while (curr != NULL){
            if (curr->time == time){
                return curr;
            }else{
```

```
                curr = curr->next;
            }
        }
        return NULL;
}
void ShiftTime_(NodePtr node, uint8 newTime, NodePtr* head){
    RemovePackage_(node, head);
    InsertPackage_(newTime, head);
}
void ShiftTimeAll_(uint8 offset, NodePtr* head){
    NodePtr curr = *head;
    while (curr != NULL){
        curr->time += offset;
        curr = curr->next;
    }
}
void ResetTimeline_(NodePtr* head){
    while(*head != NULL){

        NodePtr temp = *head;
        *head = (*head)->next;
        free(temp);
        temp = NULL;
    }
}
void RemovePackage_(NodePtr del, NodePtr* head){
    if (*head == NULL || del == NULL){ //empty list or element not present
        return;
    }

    if (*head == del){
        *head = del->next;
    }
    if (del->next != NULL){

        del->next->prev = del->prev;
    }
    if (del->prev != NULL){
        del->prev->next = del->next;
    }
    free(del);
    return;
}

void InsertPackage_(uint8 time, NodePtr* head){

    NodePtr node;
    NodePtr current;
    node = (NodePtr) malloc(sizeof(struct Node));
    node->time = time;
    node->prev = NULL;
    node->next = NULL;


    if (*head == NULL){
        *head = node;
    }
    else if (time <= (*head)->time){
        node->next = *head;
        node->next->prev = node;
        *head =  node;
    }
    else {
        current = *head;
        while (current->next != NULL && current->next->time < node->time){
```

```
            current = current->next;
        }
        node->next = current->next;
        if(current->next != NULL){ //if it was inserted in middle
            node->next->prev = node;
        }
        current->next = node;
        node->prev = current;
    }
    return;
}
```

Listing A.2: C file for testing

Appendix B

Code for scheduler

```
/*
    ******************************************************************************
** File: cmd_sch_app.h
**
** Purpose:
**   This file is main hdr file for the CMD_SCH application.
**
**
******************************************************************************
    */

#ifndef _cmd_sch_app_h_
#define _cmd_sch_app_h_

/*
**   Include Files:
*/

#include "cfe.h"
#include "cmd_sch_tbldefs.h"
#include "cfe_platform_cfg.h"
#include "osk_c_fw.h"

/*
** Event message ID's.
*/
#define CMD_SCH_INIT_INF_EID       1    /* start up message "informational" */

#define CMD_SCH_NOOP_INF_EID       2    /* processed command "informational"
    */
#define CMD_SCH_RESET_INF_EID      3
#define CMD_SCH_PROCESS_INF_EID    4

```

```
#define CMD_SCH_MID_ERR_EID        5    /* invalid command packet "error" */
#define CMD_SCH_CC1_ERR_EID        6
#define CMD_SCH_LEN_ERR_EID        7
#define CMD_SCH_PIPE_ERR_EID       8

#define CMD_SCH_SCHEDULER_INF_EID 9    /* scheduler info */

#define CMD_SCH_EVT_COUNT          9    /* count of event message ID's */


/*
** Command packet command codes
*/
#define CMD_SCH_NOOP_CC            0    /* no-op command */
#define CMD_SCH_RESET_CC           1    /* reset counters */
#define CMD_SCH_PROCESS_CC         2    /* Perform Routine Processing */

#define CMD_SCH_INSERT_CC   3            /* insert-op command */

#define CMD_SCH_DELETE_CC   4            /* delete-op command */
#define CMD_SCH_SHIFT_CC   5             /* shift-op command */
#define CMD_SCH_SHIFTALL_CC   6          /* shiftall-op command */
#define CMD_SCH_RESET_SCHEDULER_CC   7  /* reset command */
#define CMD_SCH_SIZE_CC   8              /* size command */

/*
** Table defines
*/
#define CMD_SCH_NUM_TABLES         2    /* Number of Tables */

#define CMD_SCH_FIRST_TBL_FILE  "/cf/cmd_sch_tbl_1.tbl"
#define CMD_SCH_SECOND_TBL_FILE "/cf/cmd_sch_tbl_2.tbl"

#define CMD_SCH_TBL_1_ELEMENT_OUT_OF_RANGE_ERR_CODE    1
#define CMD_SCH_TBL_2_ELEMENT_OUT_OF_RANGE_ERR_CODE   -1

#define CMD_SCH_TBL_ELEMENT_1_MAX  10
#define CMD_SCH_TBL_ELEMENT_3_MAX  30

/*
** Software Bus defines
*/
#define CMD_SCH_PIPE_DEPTH         12   /* Depth of the Command Pipe for
     Application */
#define CMD_SCH_LIMIT_HK           2    /* Limit of HouseKeeping Requests on
     Pipe for Application */
#define CMD_SCH_LIMIT_CMD          4    /* Limit of Commands on pipe for
     Application */

/*
** CMD_SCH Critical Data Store defines
*/
#define CMD_SCH_CDS_NAME            "CMD_SCHCDS"


/*
** Type definition (Critical Data Store data)
*/

typedef struct
{
  uint32  DataPtOne;    /* Values stored in my CDS */
  uint32  DataPtTwo;
  uint32  DataPtThree;
  uint32  DataPtFour;
```

```
  uint32  DataPtFive;

} CMD_SCH_CdsDataType_t;


/************************************************************************/

/*
** Type definition (generic "no arguments" command)
*/

typedef struct
{
  uint8                 CmdHeader[CFE_SB_CMD_HDR_SIZE];
} CMD_SCH_NoArgsCmd_t;

typedef struct
{
  uint8                 CmdHeader[CFE_SB_CMD_HDR_SIZE];
  SchedulerHdr_t  ScheduleHeader;
} CMD_SCH_NoArgsSchCmd_t;

/************************************************************************/

/*
** Type definition (CMD_SCH housekeeping)
*/
typedef struct
{
  uint8                 TlmHeader[CFE_SB_TLM_HDR_SIZE];
  /*
  ** Command interface counters
  */
  uint8                 CmdCounter;
  uint8                 ErrCounter;

} OS_PACK CMD_SCH_HkPacket_t;

typedef struct
{
  uint8                 CmdHeader[CFE_SB_CMD_HDR_SIZE];

  /*
  ** Command interface counters
  */
  uint8                 CmdCounter;
  uint8                 ErrCounter;

} OS_PACK CMD_SCH_SchPacket_t;

/************************************************************************/
// List element
struct Node
{
    CFE_TIME_SysTime_t time;
    struct Node* next;
    struct Node* prev;
    uint8 BufMsg[];
};
typedef struct Node* NodePtr;

typedef struct
{
    CCSDS_SpacePacket_t  SpacePacket;   /**< \brief Standard Header on all
         packets  */
```

```
    CCSDS_CmdSecHdr_t    Sec;
    uint8  Time[CCSDS_TIME_SIZE]; //32 bit seconds 16 bit subseconds
    /* payload */
    uint8 playload[];

} CMD_SCH_SchedulePacketCmd_t;



typedef struct
{
    CCSDS_SpacePacket_t  SpacePacket;   /**< \brief Standard Header on all
        packets  */
    CCSDS_CmdSecHdr_t    Sec;
    uint8  Time[CCSDS_TIME_SIZE]; //32 bit seconds 16 bit subseconds
    uint32 MsgId;
    uint32 CmdFunc;

} CMD_SCH_DeletePacketCmd_t;

typedef struct
{
    CCSDS_SpacePacket_t  SpacePacket;   /**< \brief Standard Header on all
        packets  */
    CCSDS_CmdSecHdr_t    Sec;
} CMD_SCH_ResetSchedulerCmd_t;

typedef struct
{
    CCSDS_SpacePacket_t  SpacePacket;   /**< \brief Standard Header on all
        packets  */
    CCSDS_CmdSecHdr_t    Sec;
} CMD_SCH_GetSizeCmd_t;

typedef struct
{
    CCSDS_SpacePacket_t  SpacePacket;   /**< \brief Standard Header on all
        packets  */
    CCSDS_CmdSecHdr_t    Sec;
    uint8  Offset[CCSDS_TIME_SIZE]; //32 bit seconds 16 bit subseconds

} CMD_SCH_ShiftAllPacketsCmd_t;

typedef struct
{
    CCSDS_SpacePacket_t  SpacePacket;   /**< \brief Standard Header on all
        packets  */
    CCSDS_CmdSecHdr_t    Sec;
    uint8  Time[CCSDS_TIME_SIZE]; //32 bit seconds 16 bit subseconds
    uint32 MsgId;
    uint32 CmdFunc;
    uint8  Offset[CCSDS_TIME_SIZE]; //32 bit seconds 16 bit subseconds

} CMD_SCH_ShiftPacketCmd_t;
/************************************************************************/
/*
** Type definition (CMD_SCH app global data)
*/
typedef struct
{
  /*
  ** Command interface counters.
  */
  uint8                 CmdCounter;
  uint8                 ErrCounter;
```

```

  /*
  ** Housekeeping telemetry packet
  */
  CMD_SCH_HkPacket_t        HkPacket;
  CFE_SB_MsgPtr_t       SchPacket;
  uint8                 SchBuffer[1024];


  /*
  ** Operational data (not reported in housekeeping).
  */
  CFE_SB_MsgPtr_t       MsgPtr;
  CFE_SB_PipeId_t       CmdPipe;


  /*
  ** RunStatus variable used in the main processing loop
  */
  uint32                RunStatus;

  /*
  ** Critical Data store variables
  */
  CMD_SCH_CdsDataType_t      WorkingCriticalData; /* Define a copy of critical data that can be used during Application execution */
  CFE_ES_CDSHandle_t    CDSHandle;            /* Handle to CDS memory block */

  /*
  ** Initialization data (not reported in housekeeping)
  */
  char                  PipeName[16];
  uint16                PipeDepth;

  uint8                 LimitHK;
  uint8                 LimitCmd;

  CFE_EVS_BinFilter_t   EventFilters[CMD_SCH_EVT_COUNT];
  CFE_TBL_Handle_t      TblHandles[CMD_SCH_NUM_TABLES];

} CMD_SCH_AppData_t;



/*************************************************************************/

/*
** Local function prototypes
**
** Note: Except for the entry point (CMD_SCH_AppMain), these
**       functions are not called from any other source module.
*/
void    CMD_SCH_AppMain(void);
int32   CMD_SCH_AppInit(void);
void    CMD_SCH_AppPipe(CFE_SB_MsgPtr_t msg);




void    CMD_SCH_HousekeepingCmd(CFE_SB_MsgPtr_t msg);

void    CMD_SCH_ScheduleCmd(CFE_SB_MsgPtr_t msg);
void    CMD_SCH_NoopCmd(CFE_SB_MsgPtr_t msg);
void    CMD_SCH_ResetCmd(CFE_SB_MsgPtr_t msg);
```

```
void    CMD_SCH_RoutineProcessingCmd(CFE_SB_MsgPtr_t msg);

boolean CMD_SCH_VerifyCmdLength(CFE_SB_MsgPtr_t msg, uint16 ExpectedLength);

int32   CMD_SCH_FirstTblValidationFunc(void *TblData);
int32   CMD_SCH_SecondTblValidationFunc(void *TblData);


void*   CheckSchedule(void* arg);
int GetTlSize(NodePtr* head);
NodePtr FindElem(CFE_TIME_SysTime_t time, uint16 cmdCode, uint16 id, NodePtr*
      head);
void ShiftTime(CFE_TIME_SysTime_t time, uint16 cmdCode, uint16 id,
     CFE_TIME_SysTime_t offset, NodePtr* head);
void shiftAllTime(CFE_TIME_SysTime_t offset,NodePtr* head);
void ResetTimeline(NodePtr* head);
void DeletePackage(NodePtr del,NodePtr* head);
void InsertPackage(CFE_SB_MsgPtr_t MsgPtr, CFE_TIME_SysTime_t time,NodePtr*
     head);

void CMD_SCH_ShiftAllCmd(CFE_SB_MsgPtr_t msg);
void CMD_SCH_ShiftCmd(CFE_SB_MsgPtr_t msg);
void CMD_SCH_DeleteCmd(CFE_SB_MsgPtr_t msg);
void CMD_SCH_ResetScheduler(CFE_SB_MsgPtr_t msg);




#endif /* _cmd_sch_app_h_ */

/************************/
/*  End of File Comment */
/************************/
```

Listing B.1: Header file for scheduling app

```
/*
     ***************************************************************************
** File: cmd_sch_app.c
**
** Purpose:
**   This file contains the source code for the Cmd_sch App.
**
******************************************************************************
     */

/*
**   Include Files:
*/
#include "cmd_sch_app_msgids.h"
#include "cmd_sch_app_perfids.h"
#include "cmd_sch_app.h"
/*
** System Header files
*/
#include "/home/simon/Documents/OpenSatKit-master/cfs/osal/src/os/shared/os-
     impl.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

/*
** CMD_SCH global data
*/
```

```
CMD_SCH_AppData_t CMD_SCH_AppData;
#define CMDMGR_OBJ (&(CmdSch.CmdMgr))
int32 SchedulerSem = 100;
bool SchedulerEnabled = false;
NodePtr head = NULL;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/* CMD_SCH_AppMain() -- Application entry point and main process loop   */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

void CMD_SCH_AppMain(void)
{
    int32 Status;

    /*
    ** Register the Application with Executive Services
    */
    CFE_ES_RegisterApp();

    /*
    ** Create the first Performance Log entry
    */
    CFE_ES_PerfLogEntry(CMD_SCH_APPMAIN_PERF_ID);

    /*
    ** Perform application specific initialization
    ** If the Intialization fails, set the RunStatus to CFE_ES_APP_ERROR
    ** and the App will not enter the RunLoop.
    */
    Status = CMD_SCH_AppInit();
    if ( Status != CFE_SUCCESS )
    {
        CMD_SCH_AppData.RunStatus = CFE_ES_APP_ERROR;
    }

    /*
    ** Application Main Loop. Call CFE_ES_RunLoop to check for changes
    ** in the Application's status. If there is a request to kill this
    ** App, it will be passed in through the RunLoop call.
    */


   /*

   */
    uint32 threadid = 22;
    int res = CFE_ES_CreateChildTask(&threadid, "CheckSchedule",
         CheckSchedule, NULL, 0, 1, 0);
    if (res != 0){
            CFE_ES_WriteToSysLog("CMD_SCH App: Error creating thread!");
    }
    while ( CFE_ES_RunLoop(&CMD_SCH_AppData.RunStatus) == TRUE )
    {
        SchedulerEnabled = true;
        /*
        ** Performance Log Exit Stamp.
        */
        CFE_ES_PerfLogExit(CMD_SCH_APPMAIN_PERF_ID);

        /*
        ** Pend on the arrival of the next Software Bus message.
        */
        Status = CFE_SB_RcvMsg(&CMD_SCH_AppData.MsgPtr, CMD_SCH_AppData.
```

```
            CmdPipe, CFE_SB_PEND_FOREVER);

        /*
        ** Performance Log Entry Stamp.
        */
        CFE_ES_PerfLogEntry(CMD_SCH_APPMAIN_PERF_ID);

        /*
        ** Check the return status from the Software Bus.
        */
        if (Status == CFE_SUCCESS)
        {
            /*
            ** Process Software Bus message. If there are fatal errors
            ** in command processing the command can alter the global
            ** RunStatus variable to exit the main event loop.
            */
            CMD_SCH_AppPipe(CMD_SCH_AppData.MsgPtr);


            /*
            ** Update Critical Data Store. Because this data is only updated
            ** in one command, this could be moved the command processing function.
            */
            CFE_ES_CopyToCDS(CMD_SCH_AppData.CDSHandle, &CMD_SCH_AppData.WorkingCriticalData);

        }
        else
        {
            /*
            ** This is an example of exiting on an error.
            ** Note that a SB read error is not always going to
            ** result in an app quitting.
            */
            CFE_EVS_SendEvent(CMD_SCH_PIPE_ERR_EID, CFE_EVS_ERROR,
                              "CMD_SCH: SB Pipe Read Error, CMD_SCH App Will Exit.");

            CMD_SCH_AppData.RunStatus = CFE_ES_APP_ERROR;
         }

    } /* end while */
    SchedulerEnabled = false;

    /*
    ** Performance Log Exit Stamp.
    */
    CFE_ES_PerfLogExit(CMD_SCH_APPMAIN_PERF_ID);

    /*
    ** Exit the Application.
    */
    CFE_ES_ExitApp(CMD_SCH_AppData.RunStatus);

} /* End of CMD_SCH_AppMain() */


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/* CMD_SCH_AppInit() -- CMD_SCH initialization */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```

int32 CMD_SCH_AppInit(void)
{
    int32  Status;
    int32  ResetType;
    uint32 ResetSubType;

    /*
    ** Determine Reset Type
    */
    ResetType = CFE_ES_GetResetType(&ResetSubType);

    /*
    ** For a Poweron Reset, initialize the Critical variables.
    ** If it is a Processor Reset, these variables will be restored from
    ** the Critical Data Store later in this function.
    */
    if ( ResetType == CFE_ES_POWERON_RESET )
    {
       CMD_SCH_AppData.WorkingCriticalData.DataPtOne   = 1;
       CMD_SCH_AppData.WorkingCriticalData.DataPtTwo   = 2;
       CMD_SCH_AppData.WorkingCriticalData.DataPtThree = 3;
       CMD_SCH_AppData.WorkingCriticalData.DataPtFour  = 4;
       CMD_SCH_AppData.WorkingCriticalData.DataPtFive  = 5;
    }

    /*
    ** Setup the RunStatus variable
    */
    CMD_SCH_AppData.RunStatus = CFE_ES_APP_RUN;

    /*
    ** Initialize app command execution counters.
    */
    CMD_SCH_AppData.CmdCounter = 0;
    CMD_SCH_AppData.ErrCounter = 0;

    /*
    ** Initialize app configuration data.
    */
    strcpy(CMD_SCH_AppData.PipeName, "CMD_SCH_CMD_PIPE");
    CMD_SCH_AppData.PipeDepth = CMD_SCH_PIPE_DEPTH;

    CMD_SCH_AppData.LimitHK   = CMD_SCH_LIMIT_HK;
    CMD_SCH_AppData.LimitCmd  = CMD_SCH_LIMIT_CMD;

    /*
    ** Initialize event filter table for envents we want to filter.
    */
    CMD_SCH_AppData.EventFilters[0].EventID = CMD_SCH_PROCESS_INF_EID;
    CMD_SCH_AppData.EventFilters[0].Mask    = CFE_EVS_EVERY_FOURTH_ONE;

    CMD_SCH_AppData.EventFilters[1].EventID = CMD_SCH_RESET_INF_EID;
    CMD_SCH_AppData.EventFilters[1].Mask    = CFE_EVS_NO_FILTER;

    CMD_SCH_AppData.EventFilters[2].EventID = CMD_SCH_CC1_ERR_EID;
    CMD_SCH_AppData.EventFilters[2].Mask    = CFE_EVS_EVERY_OTHER_TWO;

    CMD_SCH_AppData.EventFilters[3].EventID = CMD_SCH_LEN_ERR_EID;
    CMD_SCH_AppData.EventFilters[3].Mask    = CFE_EVS_FIRST_8_STOP;


    /*
    ** Register event filter table.
    */
```

```
    Status = CFE_EVS_Register(CMD_SCH_AppData.EventFilters, 4,
                              CFE_EVS_BINARY_FILTER);

    if ( Status != CFE_SUCCESS )
    {
       CFE_ES_WriteToSysLog("CMD_SCH App: Error Registering Events, RC = 0x%08X\n", Status);
       return ( Status );
    }

    /*
    ** Initialize housekeeping packet (clear user data area).
    */
    CFE_SB_InitMsg(&CMD_SCH_AppData.HkPacket, CMD_SCH_HK_TLM_MID, sizeof(CMD_SCH_HkPacket_t), TRUE);
    /*
    ** Create Software Bus message pipe.
    */
    Status = CFE_SB_CreatePipe(&CMD_SCH_AppData.CmdPipe, CMD_SCH_AppData.PipeDepth, CMD_SCH_AppData.PipeName);
    if ( Status != CFE_SUCCESS )
    {
       /*
       ** Could use an event at this point
       */
       CFE_ES_WriteToSysLog("CMD_SCH App: Error Creating SB Pipe, RC = 0x%08X\n", Status);
       return ( Status );
    }

    /*
    ** Subscribe to Housekeeping request commands
    */
    Status = CFE_SB_Subscribe(CMD_SCH_SEND_HK_MID,CMD_SCH_AppData.CmdPipe);
    if ( Status != CFE_SUCCESS )
    {
       CFE_ES_WriteToSysLog("CMD_SCH App: Error Subscribing to HK Request, RC = 0x%08X\n", Status);
       return ( Status );
    }

    /*
    ** Subscribe to CMD_SCH ground command packets
    */
    Status = CFE_SB_Subscribe(CMD_SCH_CMD_MID,CMD_SCH_AppData.CmdPipe);
    if ( Status != CFE_SUCCESS )
    {
       CFE_ES_WriteToSysLog("CMD_SCH App: Error Subscribing to CMD_SCH Command, RC = 0x%08X\n", Status);
       return ( Status );
    }

    /*
    ** Register tables with cFE and load default data
    */
    Status = CFE_TBL_Register(&CMD_SCH_AppData.TblHandles[0], "MyFirstTbl",
                             sizeof(CMD_SCH_Tbl_1_t), CFE_TBL_OPT_DEFAULT,
                             CMD_SCH_FirstTblValidationFunc);
    if ( Status != CFE_SUCCESS )
    {
       CFE_ES_WriteToSysLog("CMD_SCH App: Error Registering Table 1, RC = 0x%08X\n", Status);
       return ( Status );
    }
    else
```

```
    {
       Status = CFE_TBL_Load(CMD_SCH_AppData.TblHandles[0], CFE_TBL_SRC_FILE, CMD_SCH_FIRST_TBL_FILE);
    }

    Status = CFE_TBL_Register(&CMD_SCH_AppData.TblHandles[1], "MySecondTbl",
                              sizeof(CMD_SCH_Tbl_2_t), CFE_TBL_OPT_DEFAULT,
                              CMD_SCH_SecondTblValidationFunc);
    if ( Status != CFE_SUCCESS )
    {
       CFE_ES_WriteToSysLog("CMD_SCH App: Error Registering Table 2, RC = 0x%08X\n", Status);
       return ( Status );
    }
    else
    {
      Status = CFE_TBL_Load(CMD_SCH_AppData.TblHandles[1], CFE_TBL_SRC_FILE, CMD_SCH_SECOND_TBL_FILE);
    }


    /*
    ** Create and manage the Critical Data Store
    */
    Status = CFE_ES_RegisterCDS(&CMD_SCH_AppData.CDSHandle, sizeof(CMD_SCH_CdsDataType_t), CMD_SCH_CDS_NAME);
    if(Status == CFE_SUCCESS)
    {
       /*
       ** Setup Initial contents of Critical Data Store
       */
       CFE_ES_CopyToCDS(CMD_SCH_AppData.CDSHandle, &CMD_SCH_AppData.WorkingCriticalData);

    }
    else if(Status == CFE_ES_CDS_ALREADY_EXISTS)
    {
       /*
       ** Critical Data Store already existed, we need to get a copy
       ** of its current contents to see if we can use it
       */
       Status = CFE_ES_RestoreFromCDS(&CMD_SCH_AppData.WorkingCriticalData, CMD_SCH_AppData.CDSHandle);
       if(Status == CFE_SUCCESS)
       {
          /*
          ** Perform any logical verifications, if necessary, to validate data
          */
          CFE_ES_WriteToSysLog("CMD_SCH App CDS data preserved\n");
       }
       else
       {
          /*
          ** Restore Failied, Perform baseline initialization
          */
          CMD_SCH_AppData.WorkingCriticalData.DataPtOne   = 1;
          CMD_SCH_AppData.WorkingCriticalData.DataPtTwo   = 2;
          CMD_SCH_AppData.WorkingCriticalData.DataPtThree = 3;
          CMD_SCH_AppData.WorkingCriticalData.DataPtFour  = 4;
          CMD_SCH_AppData.WorkingCriticalData.DataPtFive  = 5;
          CFE_ES_WriteToSysLog("Failed to Restore CDS. Re-Initialized CDS Data.\n");
       }
    }
```

```c
    else
    {
        /*
        ** Error creating my critical data store
        */
        CFE_ES_WriteToSysLog("CMD_SCH: Failed to create CDS (Err=0x%08x)",
            Status);
        return(Status);
    }
    Status = OS_MutSemCreate(&SchedulerSem, "sem_schedule",0);
    //Status = OS_BinSemCreate(&SchedulerSem, "sem_schedule", 1, 0); // TODO
        check success
    if(Status == CFE_SUCCESS)
        {
            /*
            ** Perform any logical verifications, if necessary, to validate
                data
            */
            CFE_ES_WriteToSysLog("CMD_SCH App Semaphore created\n");
        }
        else{
             CFE_ES_WriteToSysLog("CMD_SCH: Failed to create semaphore (Err=0x
                 %08x)", Status);
             return(Status);
        }

    /*
    ** Application startup event message.
    */
    CFE_EVS_SendEvent(CMD_SCH_INIT_INF_EID,CFE_EVS_INFORMATION, "CMD_SCH:
        Application Initialized");

    return(CFE_SUCCESS);

} /* End of CMD_SCH_AppInit() */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/* CMD_SCH_AppPipe() -- Process command pipe message                    */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

void CMD_SCH_AppPipe(CFE_SB_MsgPtr_t msg)
{
    CFE_SB_MsgId_t MessageID;
    uint16 CommandCode;

    MessageID = CFE_SB_GetMsgId(msg);
    switch (MessageID)
    {
        /*
        ** Housekeeping telemetry request.
        */
        case CMD_SCH_SEND_HK_MID:
            CMD_SCH_HousekeepingCmd(msg);
            break;

        /*
        ** CMD_SCH ground commands.
        */
        case CMD_SCH_CMD_MID:
            CommandCode = CFE_SB_GetCmdCode(msg);
            switch (CommandCode)
            {
                case CMD_SCH_INSERT_CC:
```

```
                        OS_MutSemTake(SchedulerSem);
                        CMD_SCH_ScheduleCmd(msg);
                        OS_MutSemGive(SchedulerSem);
                        break;
                    case CMD_SCH_DELETE_CC:
                        OS_MutSemTake(SchedulerSem);
                        CMD_SCH_DeleteCmd(msg);
                        OS_MutSemGive(SchedulerSem);
                        break;
                    case CMD_SCH_SHIFT_CC:
                        OS_MutSemTake(SchedulerSem);
                        CMD_SCH_ShiftCmd(msg);
                        OS_MutSemGive(SchedulerSem);
                        break;
                    case CMD_SCH_SHIFTALL_CC:
                        OS_MutSemTake(SchedulerSem);
                        CMD_SCH_ShiftAllCmd(msg);
                        OS_MutSemGive(SchedulerSem);
                        break;
                    case CMD_SCH_RESET_SCHEDULER_CC:
                        OS_MutSemTake(SchedulerSem);
                        CMD_SCH_ResetCmd(msg);
                        OS_MutSemGive(SchedulerSem);
                        break;
                    case CMD_SCH_SIZE_CC:
                        OS_MutSemTake(SchedulerSem);
                        CMD_SCH_GetSizeCmd(msg);
                        OS_MutSemGive(SchedulerSem);
                        break;
                    case CMD_SCH_NOOP_CC:
                        CMD_SCH_NoopCmd(msg);
                        break;
                    case CMD_SCH_RESET_CC:
                        CMD_SCH_ResetCmd(msg);
                        break;

                    case CMD_SCH_PROCESS_CC:
                        CMD_SCH_RoutineProcessingCmd(msg);
                        break;

                    default:
                        CFE_EVS_SendEvent(CMD_SCH_CC1_ERR_EID, CFE_EVS_ERROR,
                         "Invalid ground command code: ID = 0x%X, CC = %d",
                                          MessageID, CommandCode);
                        break;
                }
                break;

            default:
                CFE_EVS_SendEvent(CMD_SCH_MID_ERR_EID, CFE_EVS_ERROR,
                                "Invalid command pipe message ID: 0x%X",
                                 MessageID);
                break;
        }

        return;

} /* End of CMD_SCH_AppPipe() */


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/* CMD_SCH_HousekeepingCmd() -- On-board command (HK request)          */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```

void CMD_SCH_HousekeepingCmd(CFE_SB_MsgPtr_t msg)
{
    uint16 ExpectedLength = sizeof(CMD_SCH_NoArgsCmd_t);
    uint16 i;

    /*
    ** Verify command packet length
    */

    if (CMD_SCH_VerifyCmdLength(msg, ExpectedLength))
    {
        /*
        ** Get command execution counters
        */
        CMD_SCH_AppData.HkPacket.CmdCounter = CMD_SCH_AppData.CmdCounter;
        CMD_SCH_AppData.HkPacket.ErrCounter = CMD_SCH_AppData.ErrCounter;

        /*
        ** Send housekeeping telemetry packet
        */
        CFE_SB_TimeStampMsg((CFE_SB_Msg_t *) &CMD_SCH_AppData.HkPacket);
        CFE_SB_SendMsg((CFE_SB_Msg_t *) &CMD_SCH_AppData.HkPacket);

        /*
        ** Manage any pending table loads, validations, etc.
        */
        for (i=0; i<CMD_SCH_NUM_TABLES; i++)
        {
            CFE_TBL_Manage(CMD_SCH_AppData.TblHandles[i]);
        }

        /*
        ** This command does not affect the command execution counter
        */

    } /* end if */

    return;

} /* End of CMD_SCH_HousekeepingCmd() */


/* Format the time to systime: Only seconds at the moment*/
CFE_TIME_SysTime_t getTime(uint8  Time[6]){
    CFE_TIME_SysTime_t newTime;
    uint32 seconds =  ((uint32) Time[3]<<24) | ((uint32) Time[2]<<16) | ((
        uint32) Time[1]<<8) | ((uint32) Time[0]);
    uint32 subSeconds =  0;
    newTime.Seconds = seconds;
    newTime.Subseconds = subSeconds;



    return newTime;
}

/* Get the size of the timeline */
int GetTlSize(NodePtr* head){
    int count = 0;
    NodePtr curr = *head;

    while (curr != NULL){
        count++;
        curr = curr->next;
```

```
    }
    return count;
}

/* Reset the scheduler (free all elements) */
void ResetTimeline(NodePtr* head){
    while(*head != NULL){
        NodePtr temp = *head;
        *head = (*head)->next;
        free(temp);
        temp = NULL;
    }
}

/* Delete a package in the timeline */
void DeletePackage(NodePtr elem, NodePtr* head){
    if (*head == NULL || elem == NULL){ //end elem
        return;
    }
    if (*head == elem){
        *head = elem->next;
    }
    if (elem->next != NULL){
        elem->next->prev = elem->prev;
    }
    if (elem->prev != NULL){
        elem->prev->next = elem->next;
    }
    free(elem);
    elem = NULL;
    return;
}

/* "pop" and element */
void SendScheduledMsg(NodePtr elem, NodePtr* head){

    // see if operation was success,
    // removes packet from timeline either way - for now
    if (CFE_SB_SendMsg(elem->BufMsg) == CFE_SUCCESS){
        DeletePackage(elem, head);
    }
    else {
        CFE_EVS_SendEvent(CMD_SCH_NOOP_INF_EID,CFE_EVS_ERROR, "scheduled
            message operation unsuccessful");
        DeletePackage(elem, head);
    }
}


// Checks the schedule for packets
// Runs in a different thread
void* CheckSchedule(void* arg){
    while(SchedulerEnabled = true){

        sleep(1);
        OS_MutSemTake(SchedulerSem);
        NodePtr curr = head;
        if(curr != NULL){
            CFE_TIME_SysTime_t currTime = CFE_TIME_GetTime();
            if ( CFE_TIME_Compare(curr->time, currTime) <= 0){

                CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID,
                    CFE_EVS_INFORMATION,"CMD_SCH App: Execute command at
                    time: %u , current time: %u", curr->time.Seconds,
                    currTime.Seconds);
```

```
                SendScheduledMsg(curr, &head);

            }
        }
        OS_MutSemGive(SchedulerSem);

    }
    CFE_ES_ExitChildTask();
    return NULL;
}


// insert a packet in the timeline
void InsertPackage(CFE_SB_MsgPtr_t MsgPtr, CFE_TIME_SysTime_t time, NodePtr*
     head){
    NodePtr node;

    node = (NodePtr) malloc(sizeof(struct Node)+ CFE_SB_GetTotalMsgLength(
         MsgPtr));
    node->time = time;
    node->next = NULL;
    node->prev = NULL;
    memcpy(node->BufMsg, MsgPtr, CFE_SB_GetTotalMsgLength(MsgPtr));

    NodePtr current;
    if (*head == NULL){
        *head = node;
    }

    else if (CFE_TIME_Compare(time, (*head)->time) <= 0){
        node->next = *head;
        node->next->prev = node;
        *head =  node;
    }
    else {
        current = *head;
        while (current->next != NULL && CFE_TIME_Compare(current->next->time,
             node->time) == -1){
            current = current->next;
        }
        node->next = current->next;
        if(current->next != NULL){ //if it was inserted in middle
            node->next->prev = node;
        }
        current->next = node;
        node->prev = current;
    }
    CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,"CMD_SCH
         App: Inserted command at time: %d", time.Seconds);
    return;

}

// find a packet in the timeline
NodePtr FindElem(CFE_TIME_SysTime_t time, uint16 cmdCode, uint16 id, NodePtr*
      head){

    NodePtr curr = *head;
    while (curr != NULL){
        if (CFE_TIME_Compare(time, curr->time) == 0){
            if(CFE_SB_GetCmdCode(curr->BufMsg) == cmdCode && CFE_SB_GetMsgId(
                 curr->BufMsg) == id){
                CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID,
                     CFE_EVS_INFORMATION,
                    "CMD_SCH: Found elem at time: %d", time.Seconds);
```

```
                return curr;
            }
        }
        curr = curr->next;
    }
    return NULL;
}

//shift the release time all elements
void ShiftAllTime(CFE_TIME_SysTime_t offset, NodePtr* head){
    NodePtr curr = *head;
    while (curr != NULL){
        curr->time.Seconds += offset.Seconds;
        curr->time.Subseconds += offset.Subseconds;
        curr = curr->next;
    }

}

//shift the release time of one element in the timeline
void ShiftTime(CFE_TIME_SysTime_t time, uint16 cmdCode, uint16 id,
     CFE_TIME_SysTime_t offset, NodePtr* head){
    NodePtr elem = FindElem(time, cmdCode, id, head);
    if (elem != NULL){
        uint16 length = CFE_SB_GetTotalMsgLength(elem->BufMsg);
        uint8 msg[length];
        memcpy(msg, elem->BufMsg, length);
        DeletePackage(elem, head);
        offset.Seconds += time.Seconds;
        offset.Subseconds += time.Subseconds;
        InsertPackage(msg, offset, head);

        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                "CMD_SCH: Changed time from: %d, To: %d\n", time.Seconds,
                    offset.Seconds);

    }
    else{
        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                "CMD_SCH: Shift-op, packet not found!");
    }

    return;
}



void CMD_SCH_ResetScheduler(CFE_SB_MsgPtr_t msg)
{
    uint16 ExpectedLength = sizeof(CMD_SCH_ResetSchedulerCmd_t);

    /*
    ** Verify command packet length...
    */
    if (CMD_SCH_VerifyCmdLength(msg, ExpectedLength))
    {
        CMD_SCH_AppData.CmdCounter++;

        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                    "CMD_SCH: Reset scheduler command");

        ResetTimeline(&head);
    }
    else{
        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
```

```
                            "CMD_SCH: Reset-op , but wrong length!");
    }
    return;

}
void CMD_SCH_DeleteCmd(CFE_SB_MsgPtr_t msg)
{
    uint16 ExpectedLength = sizeof(CMD_SCH_DeletePacketCmd_t);

    /*
    ** Verify command packet length...
    */
    if (CMD_SCH_VerifyCmdLength(msg, ExpectedLength))
    {
        CMD_SCH_AppData.CmdCounter++;

        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                          "CMD_SCH: Delete packet command");
        CMD_SCH_DeletePacketCmd_t* packet = (CMD_SCH_DeletePacketCmd_t*) msg;
             //format message
        NodePtr elem = FindElem(getTime(packet->Time), packet->CmdFunc,
            packet->MsgId, &head);
        DeletePackage(elem, &head);
    }
    else{
        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                        "CMD_SCH: Delete-op, but wrong length!");
    }
    return;

}

void CMD_SCH_ShiftCmd(CFE_SB_MsgPtr_t msg)
{
    uint16 ExpectedLength = sizeof(CMD_SCH_ShiftPacketCmd_t);

    /*
    ** Verify command packet length...
    */
    if (CMD_SCH_VerifyCmdLength(msg, ExpectedLength))
    {
        CMD_SCH_AppData.CmdCounter++;

        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                          "CMD_SCH: Shift packet command");
        CMD_SCH_ShiftPacketCmd_t* packet = (CMD_SCH_ShiftPacketCmd_t*) msg;
            //format message

        ShiftTime(getTime(packet->Time), packet->CmdFunc, packet->MsgId,
            getTime(packet->Offset), &head);

    }
    else{
        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                        "CMD_SCH: Shift-op, but wrong length!");
    }
    return;

}

void CMD_SCH_ShiftAllCmd(CFE_SB_MsgPtr_t msg)
{
    uint16 ExpectedLength = sizeof(CMD_SCH_ShiftAllPacketsCmd_t);

    /*
```

```
    ** Verify command packet length...
    */
    if (CMD_SCH_VerifyCmdLength(msg, ExpectedLength))
    {
        CMD_SCH_AppData.CmdCounter++;

        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                          "CMD_SCH: Shift packet command");
        CMD_SCH_ShiftAllPacketsCmd_t* packet = (CMD_SCH_ShiftAllPacketsCmd_t *) msg; //format message

        ShiftAllTime(getTime(packet->Offset), &head);

    }
    else{
        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                        "CMD_SCH: ShiftAll-op, but wrong length!");
    }
    return;

}

void CMD_SCH_GetSizeCmd(CFE_SB_MsgPtr_t msg)
{
    uint16 ExpectedLength = sizeof(CMD_SCH_GetSizeCmd_t);

    /*
    ** Verify command packet length...
    */
    if (CMD_SCH_VerifyCmdLength(msg, ExpectedLength))
    {
        CMD_SCH_AppData.CmdCounter++;
        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                          "CMD_SCH:  Size of timeline: %d", GetTlSize(&head));
    }
    else{
        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                        "CMD_SCH: GetSize-op, but wrong length!");
    }
    return;

}

void CMD_SCH_ScheduleCmd(CFE_SB_MsgPtr_t msg){

        CMD_SCH_AppData.CmdCounter++;


        CMD_SCH_SchedulePacketCmd_t *CmdHdrPtr = (CMD_SCH_SchedulePacketCmd_t *) msg;
        CFE_SB_MsgPtr_t MsgTest = &CmdHdrPtr->playload;
        CFE_TIME_SysTime_t newTime = getTime(CmdHdrPtr->Time);
        newTime.Seconds+=2; //testing
        if (CFE_TIME_Compare(newTime, CFE_TIME_GetTime()) == 1){

            CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
            "CMD_SCH: Scheduler command recieved\n");
            InsertPackage(MsgTest, newTime, &head);
            /* some test */
            CFE_TIME_SysTime_t offset; //testing
            offset.Seconds = 10; //testing
            offset.Subseconds = 0; //testing
            ShiftTime(newTime, CFE_SB_GetCmdCode(MsgTest),CFE_SB_GetMsgId(MsgTest),offset, &head); //testing
```

```
            CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                              "CMD_SCH:  Size of timeline: %d", GetTlSize(&head))
                              ;
            /*          */
        }
        else{
            CFE_EVS_SendEvent(CMD_SCH_NOOP_INF_EID, CFE_EVS_INFORMATION,
                             "CMD_SCH Version 1.0.0: Schedule command but invalid
                              time!");
        }


    return;

}



/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/* CMD_SCH_NoopCmd() -- CMD_SCH ground command (NO-OP)
    */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void CMD_SCH_NoopCmd(CFE_SB_MsgPtr_t msg)
{
    uint16 ExpectedLength = sizeof(CMD_SCH_NoArgsCmd_t);
    /*
    ** Verify command packet length...
    */
    if (CMD_SCH_VerifyCmdLength(msg, ExpectedLength))
    {
        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                         "CMD_SCH Version 1.0.0: No-op command");


    }
    else{
        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                          "CMD_SCH Version 1.0.0: No-op command, but wrong
                           length!");
    }

    return;

} /* End of CMD_SCH_NoopCmd() */


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/* CMD_SCH_ResetCmd() -- CMD_SCH ground command (reset counters)
    */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

void CMD_SCH_ResetCmd(CFE_SB_MsgPtr_t msg)
{
    uint16 ExpectedLength = sizeof(CMD_SCH_NoArgsCmd_t);

    /*
    ** Verify command packet length...
    */
    if (CMD_SCH_VerifyCmdLength(msg, ExpectedLength))
    {
        CMD_SCH_AppData.CmdCounter = 0;
```

```
        CMD_SCH_AppData.ErrCounter = 0;

        CFE_EVS_SendEvent(CMD_SCH_SCHEDULER_INF_EID, CFE_EVS_INFORMATION,
                          "CMD_SCH: Reset Counters command");
    }

    return;

} /* End of CMD_SCH_ResetCmd() */


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/*  CMD_SCH_RoutineProcessingCmd() - Common Processing for each cmd.    */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

void CMD_SCH_RoutineProcessingCmd(CFE_SB_MsgPtr_t msg)
{
    uint16 ExpectedLength = sizeof(CMD_SCH_NoArgsCmd_t);
    CMD_SCH_Tbl_1_t   *MyFirstTblPtr;
    CMD_SCH_Tbl_2_t   *MySecondTblPtr;

    /*
    ** Verify command packet length
    */
    if (CMD_SCH_VerifyCmdLength(msg, ExpectedLength))
    {
        /* Obtain access to table data addresses */
        CFE_TBL_GetAddress((void *)&MyFirstTblPtr, CMD_SCH_AppData.TblHandles
             [0]);
        CFE_TBL_GetAddress((void *)&MySecondTblPtr, CMD_SCH_AppData.
             TblHandles[1]);

        /* Perform routine processing accessing table data via pointers */
        /*                            .                                 */
        /*                            .                                 */
        /*                            .                                 */

        /* Once completed with using tables, release addresses          */
        CFE_TBL_ReleaseAddress(CMD_SCH_AppData.TblHandles[0]);
        CFE_TBL_ReleaseAddress(CMD_SCH_AppData.TblHandles[1]);

        /*
        ** Update Critical variables. These variables will be saved
        ** in the Critical Data Store and preserved on a processor reset.
        */
        CMD_SCH_AppData.WorkingCriticalData.DataPtOne++;
        CMD_SCH_AppData.WorkingCriticalData.DataPtTwo++;
        CMD_SCH_AppData.WorkingCriticalData.DataPtThree++;
        CMD_SCH_AppData.WorkingCriticalData.DataPtFour++;
        CMD_SCH_AppData.WorkingCriticalData.DataPtFive++;

        CFE_EVS_SendEvent(CMD_SCH_PROCESS_INF_EID,CFE_EVS_INFORMATION, "
             CMD_SCH: Routine Processing Command");

        CMD_SCH_AppData.CmdCounter++;
    }

    return;

} /* End of CMD_SCH_RoutineProcessingCmd() */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
```

```
/* CMD_SCH_VerifyCmdLength() -- Verify command packet length            */
/*                                                                       */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

boolean CMD_SCH_VerifyCmdLength(CFE_SB_MsgPtr_t msg, uint16 ExpectedLength)
{
    boolean result = TRUE;
    uint16 ActualLength = CFE_SB_GetTotalMsgLength(msg);

    /*
    ** Verify the command packet length...
    */
    if (ExpectedLength != ActualLength)
    {
        CFE_SB_MsgId_t MessageID = CFE_SB_GetMsgId(msg);
        uint16 CommandCode = CFE_SB_GetCmdCode(msg);

        CFE_EVS_SendEvent(CMD_SCH_LEN_ERR_EID, CFE_EVS_ERROR,
           "CMD_SCH: Invalid cmd pkt: ID = 0x%X,  CC = %d, Len = %d, ExLen = %d",
                          MessageID, CommandCode, ActualLength, ExpectedLength);
        result = FALSE;
        CMD_SCH_AppData.ErrCounter++;
    }

    return(result);

} /* End of CMD_SCH_VerifyCmdLength() */


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                       */
/* CMD_SCH_FirstTblValidationFunc() -- Verify contents of First Table   */
/*                                  buffer contents                      */
/*                                                                       */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

int32 CMD_SCH_FirstTblValidationFunc(void *TblData)
{
    int32              ReturnCode = CFE_SUCCESS;
    CMD_SCH_Tbl_1_t *TblDataPtr = (CMD_SCH_Tbl_1_t *)TblData;

    if (TblDataPtr->TblElement1 > CMD_SCH_TBL_ELEMENT_1_MAX)
    {
        /* First element is out of range, return an appropriate error code */
        ReturnCode = CMD_SCH_TBL_1_ELEMENT_OUT_OF_RANGE_ERR_CODE;
    }

    return ReturnCode;
}


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                       */
/* CMD_SCH_SecondTblValidationFunc() -- Verify contents of Second Table */
/*                                   buffer contents                     */
/*                                                                       */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

int32 CMD_SCH_SecondTblValidationFunc(void *TblData)
{
    int32              ReturnCode = CFE_SUCCESS;
    CMD_SCH_Tbl_2_t *TblDataPtr = (CMD_SCH_Tbl_2_t *)TblData;

```

```
    if (TblDataPtr->TblElement3 > CMD_SCH_TBL_ELEMENT_3_MAX)
    {
        /* Third element is out of range, return an appropriate error code */
        ReturnCode = CMD_SCH_TBL_2_ELEMENT_OUT_OF_RANGE_ERR_CODE;
    }


    return ReturnCode;
}

/***********************/
/*  End of File Comment */
/***********************/
```

Listing B.2: C file for scheduling app

Bibliography

[CCS03] CCDS space packet protocol. https://public.ccsds.org/Pubs/133x0b2e1.pdf, 2003.

[cFS21] NASA STI Program core flight system (cfs) training. https://ntrs.nasa.gov/api/citations/20205011588/downloads/TM%2020205000691%20REV%201.pdf, 2021.

[COS] COSMOS the user interface for command and control of embedded systems. `https://cosmosc2.com/`.

[ESA20] Space engineering: Ground systems and operations — telemetry and telecommand packet utilization. https://cwe.ccsds.org, 2020.