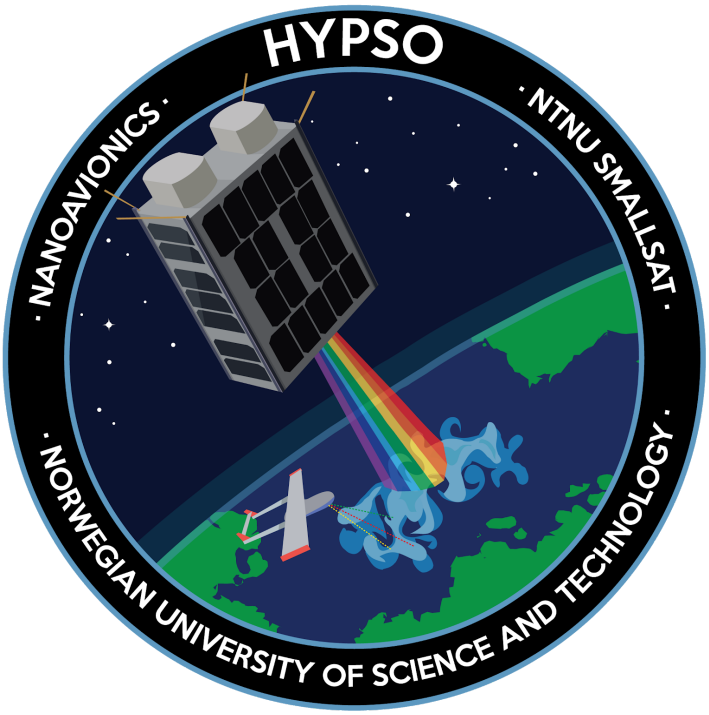
Open MCT Integration

HYPSO-DR-015



Prepared by: HYPSO Project Team
Reference: HYPSO-DR-015
Revision: 2
Date of issue: Date
Status: Preliminary
Document Type: Design Report

Table Of Contents

1 Overview **4**
1.1 Purpose 4
1.2 Scope 4
1.3 Summary 4
1.5 Applicable Documents 5
1.6 Referenced Documents 5

2 System description **6**
2.1 Quick Introduction to OpenMCT 6
2.2 Overview 6
2.3 Software architecture 7
2.3.1 Shared modules 10
2.3.2 Telemetry fetching subsystem 10
2.3.3 Data management and storage subsystem 11
2.3.4 Telemetry serving subsystem 13
2.3.5 OpenMCT client plugins 14
2.4 Code standards 14
2.5 Third-party modules 14

3 System usage **15**
3.1 Setup and configuration 15
3.1.1 Server setup 15
3.1.2 Configuration options 15
3.1.3 Data definitions 15
3.1.4 Data sources 15
3.2 Operation 15
3.2.1 Using OpenMCT 15
3.2.2 Monitoring system status 15
3.3 Expansion options 16
3.3.1 Adding new data sources 16
3.3.2 Telemetry API 16

4 Current status **17**
4.1 Plan 17

5 List of abbreviations **18**

Table 1: Table of Changes

Rev.	Summary of Changes	Author(s)	Effective Date
0.1	*Initial Draft*	*J L Garrett*	
		R Birkeland	
0.2	*Document outline, title change*	Audun V. Nytrø	
0.3	*Documented system architecture and plan ahead*	Audun V. Nytrø	2020-03-27

1 Overview

The HYPSO Mission will primarily be a science-oriented technology demonstrator. It will enable low-cost & high-performance hyperspectral imaging and autonomous onboard processing that fulfill science requirements in ocean color remote sensing and oceanography. NTNU SmallSat is prospected to be the first SmallSat developed at NTNU with launch planned for Q4 2020 followed by a second mission later. Furthermore, vision of a constellation of remote-sensing focused SmallSat will constitute a space-asset platform added to the multi-agent architecture of UAVs, USVs, AUVs and buoys that have similar ocean characterization objectives.

1.1 Purpose

The purpose of the HYPSO-DR-015 OpenMCT Integration Design Report is to provide an overview of the planned system for viewing and browsing historical and live telemetry data using NASA's OpenMCT software package.

1.2 Scope

This document covers the web server that fetches, unpacks and locally stores satellite telemetry data, which also hosts the OpenMCT frontend. It also covers the code required for OpenMCT to use the data provided by this server.

It does not cover any external users of the data used by the telemetry web server, such as the ground station software suite, but does describe how these users can interface with the telemetry web server to get historical and real-time data for use outside OpenMCT.

1.3 Summary

This document describes the implementation details, usage and status of the system for displaying and analyzing telemetry data from the HYPSO spacecraft in OpenMCT.

An Express-based web server running on Node.js with SQLite is currently being developed for hosting this system, which will allow users to get real-time and historical telemetry data in OpenMCT, or using a separate HTTP or WebSockets client.

The document currently consists of the following:

- Chapter 2: System description, which contains implementation details and explanations for how the overall system is intended to work
- Chapter 3: System usage, which contains usage instructions and recommendations

- Chapter 4: Current status, which explains what work remains and the plan ahead for getting that work done

1.5 Applicable Documents

The following table lists the applicable documents for this document and work.

Table 2: Applicable Documents

ID	Author	Title
[AD01]	ECSS Secretariat	ECSS-M-ST-10C: Space management: Project planning and implementation (tailored)
[AD02]	Norwegian Space Agency	NSC-Payload-PAQA (tailored)
[AD03]		
[AD04]		
[AD05]		

1.6 Referenced Documents

The documents listed in Table 3 have been used as reference in creation of this document.

Table 3: Referenced Documents

ID	Author	Title
[RD01]	Mariusz Grøtte	Telemetry Format Spreadsheet https://docs.google.com/spreadsheets/d/1DCzhNcp5J5GWeUGTtZC0nzXIwu4XpLFEQB6uAN2jq2Y/edit#gid=0
[RD02]		
[RD03]		
[RD04]		
[RD05]		

2 System description

2.1 Quick Introduction to OpenMCT

OpenMCT - short for Open Mission Control Technologies - is a new mission control framework developed and used by NASA for visualization of various types of data inside a web browser, both on mobile and desktop devices.

It is very flexible and extensible, allowing for many different types of data to be integrated and easily accessible on one single website for mission planning and telemetry data analysis. It reduces the need for mission operators to switch between many different applications to view all necessary data.

More information can be found on OpenMCT's official website: https://nasa.github.io/openmct/.

2.2 Overview

At its core, the system consists of a web server that gets data from NanoAvionics' server, stores it locally and hosts the OpenMCT frontend along with code that allows OpenMCT to understand the telemetry data format used by NanoAvionics. A summary and early version of this structure and its connections can be seen in the high-level system block diagram shown in figure 2.1.

The data is stored in a local database to abstract away the connection to NanoAvionics, allowing us to manipulate and access the data as we want without having to depend on their server being available, and to avoid constantly doing requests to it for data we've downloaded before. This reduces our load on their server, and allows us to easily support multiple instances of OpenMCT (or other services that need telemetry data) running and accessing telemetry data simultaneously if required.

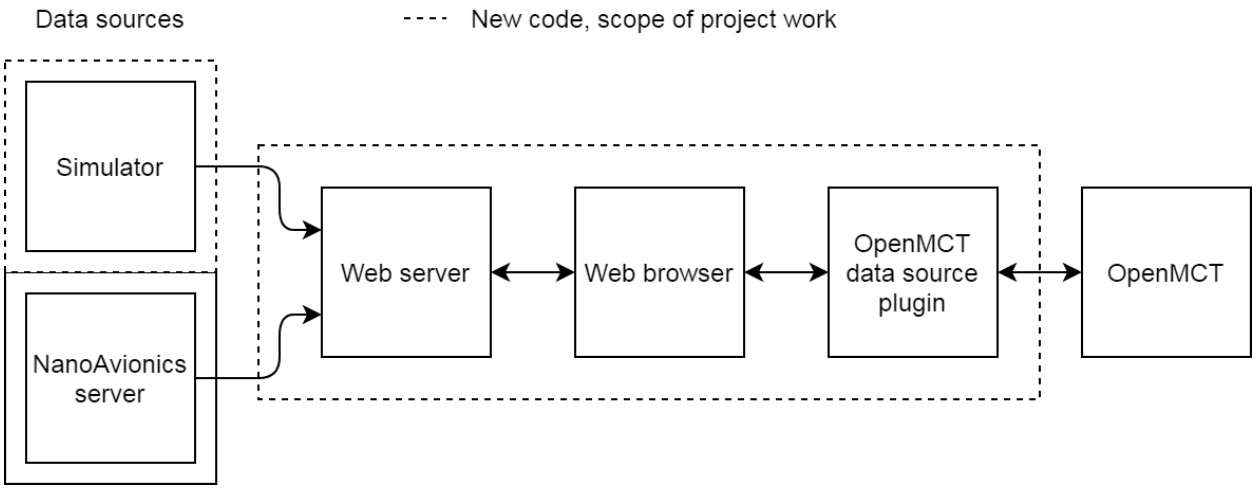


Figure 2.2 High-level system block diagram

2.3 Software architecture

To implement the system outlined above, it was decided to use JavaScript (in the form of ECMAScript 6) for the backend since it's already used in the OpenMCT frontend, to avoid introducing more than one extra programming language for the HYPSO project to support in relation to this system, and to encourage code sharing between the frontend and backend.

Seeing as we want the system to be independent of the data source - whether it's directly to NanoAvionics' server, a file-based backup, or another server providing some kind of relevant data, it was natural to split the project into three more or less independent parts: The telemetry fetching system, the data management and storage system, and the telemetry serving system. The data management and storage system is the link between the otherwise independent telemetry fetching and telemetry serving systems.

Seeing as using JavaScript was wanted for the backend, it was decided to use Node.js running an Express-based web app to host and manage each of these subsystems. Express was chosen since it is one of the most used and well-documented web server frameworks for Node.js, and has already been used in multiple other successful implementations of OpenMCT telemetry servers including the current official OpenMCT server tutorial.

Based on the block diagram above a data flow diagram for the full system was created to get an idea of the types and amounts of data that would be going between each subsystem. This can be found in Figure 2.3.1.

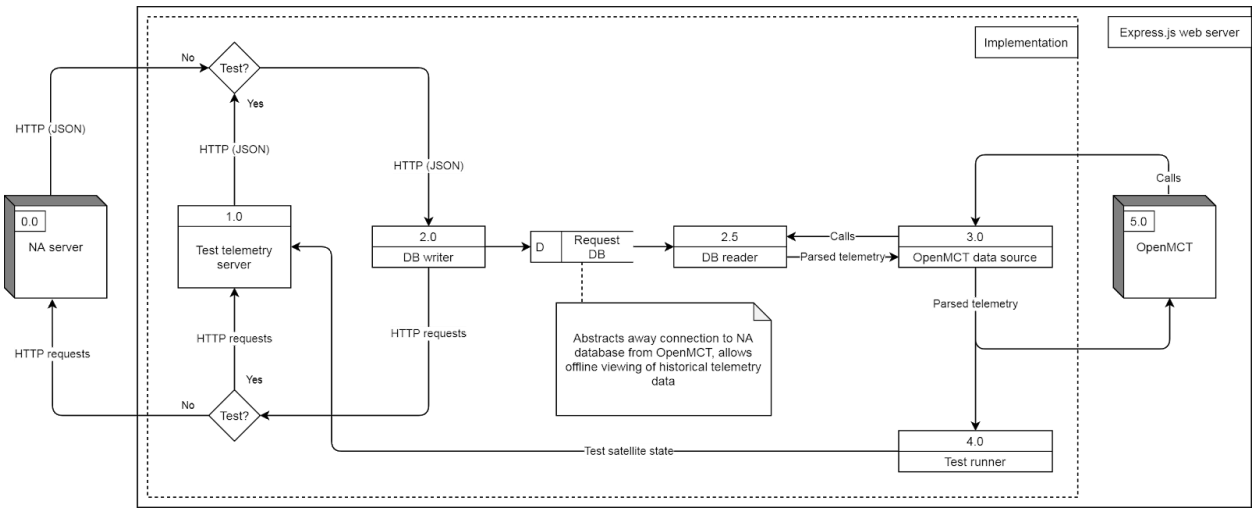


Figure 2.3.1 Level 0 Data Flow Diagram for the full system

After this a full class diagram with the full extent of modules and descriptions of their local variables and external methods was created; this will be referenced actively in the section below outlining the functionality and decisions behind the implementation of each module. This can be found in Figure 2.3.2, with cutouts for each subsystem found in Figures 2.3.3 to 2.3.7.

Attempting to read Figure 2.3.2 directly is not recommended, and it is mostly included as a way to get a quick overview of the connections between the subsystems.

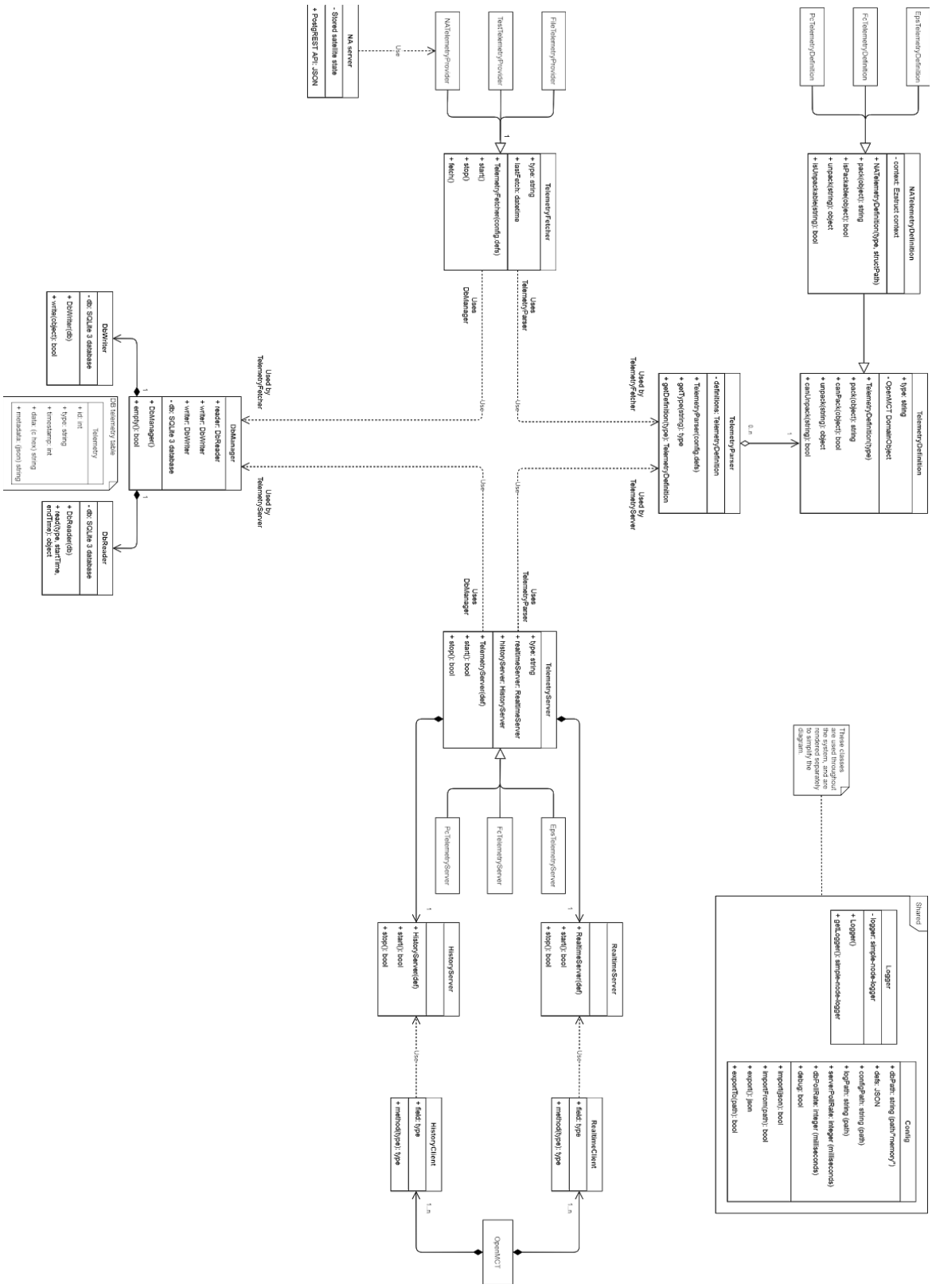


Figure 2.3.2 Class Diagram for full system

2.3.1 Shared modules

These provide simple but important utility functions, such as managing the current configuration of the system and allowing it to be imported/exported to a file, plus providing a system log.

The class diagram for these modules can be found in Figure 2.3.3.

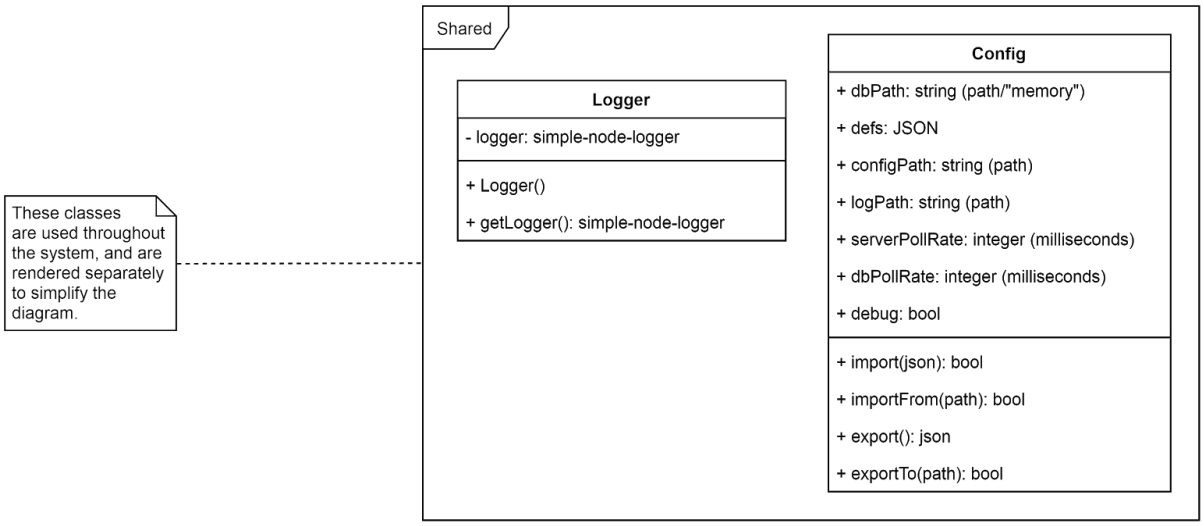


Figure 2.3.3 Class Diagram for shared modules

2.3.2 Telemetry fetching subsystem

This subsystem provides an interface for getting telemetry data from external sources implemented by the module `TelemetryFetcher`, which is realized for various specific external sources as `FileTelemetryProvider`, `TestTelemetryProvider`, and `NATelemetryProvider`. As the names imply, these get telemetry data from a file, a test spacecraft, and NanoAvionics' servers respectively.

The default implementation of all of these are simple polling services that check their sources at a regular interval to see if theres any new data; if so this data is written to the connected `DbManager`. A connection to a `TelemetryDefinition` via `TelemetryParser` may be required if the telemetry type, timestamp or any metadata to be stored cannot be determined directly from the received data without it being unpacked and parsed first. These modules will be introduced in the next section.

The class diagram for these modules can be found in Figure 2.3.4.

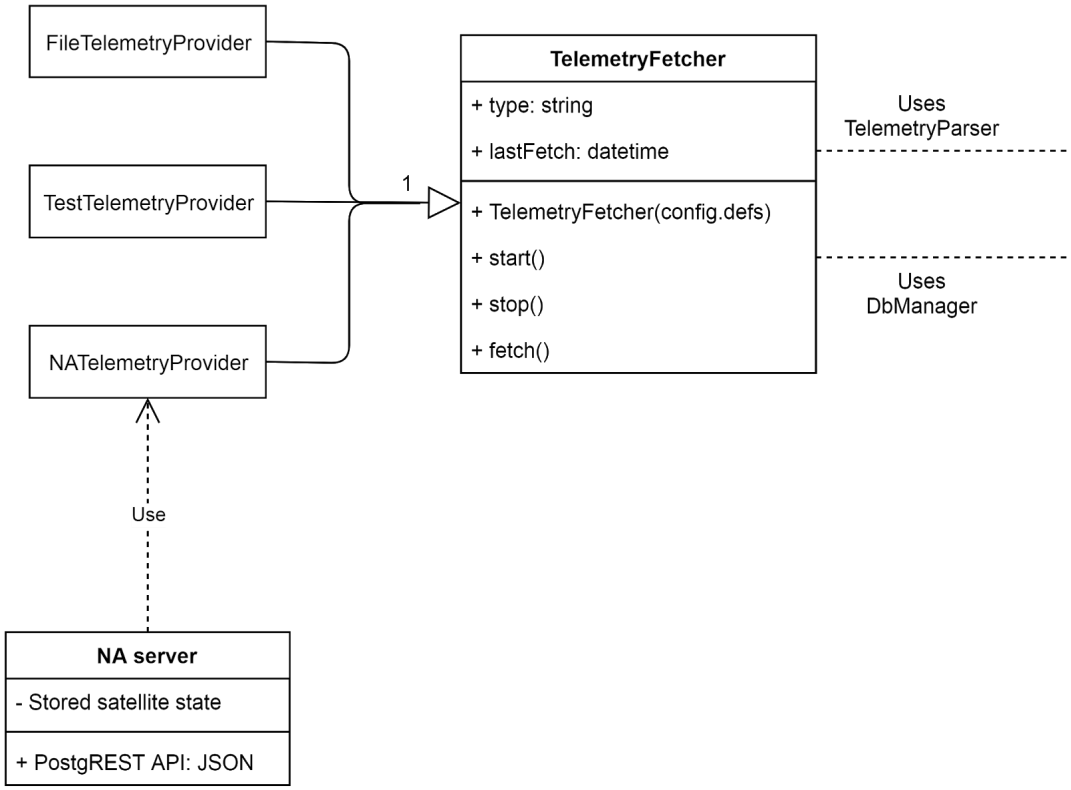


Figure 2.3.4 Class Diagram for telemetry fetching section

2.3.3 Data management and storage subsystem

This subsystem consists of two main modules; the `TelemetryParser` and `DbManager`.

The first of these, `TelemetryParser`, provides methods for configuring, parsing and unpacking various types of telemetry data, with a general class `TelemetryDefinition` that has a special subclass `NATelemetryDefinition` that provides a quick way to set up the `TelemetryDefinitions` for each type of telemetry we get directly from NanoAvionics' server.

The data that arrives from NanoAvionics' from their PostgREST-based HTTP API is in the form of JSON with some basic metadata and, more importantly, a string that contains a packed C-style struct. In `NATelemetryDefinition` we use a third-party module called EzStruct that allows us to quickly convert this to JSON based directly on the struct header definitions we get in text files from NanoAvionics, speeding up the process of implementing and updating the telemetry definitions greatly. Updating or adding a new `TelemetryDefinition` for parsing telemetry data from NanoAvionics' is usually as simple as downloading the struct header text file and updating the configuration to include it.

The class diagram for this subsystem can be found in Figure 2.3.5.

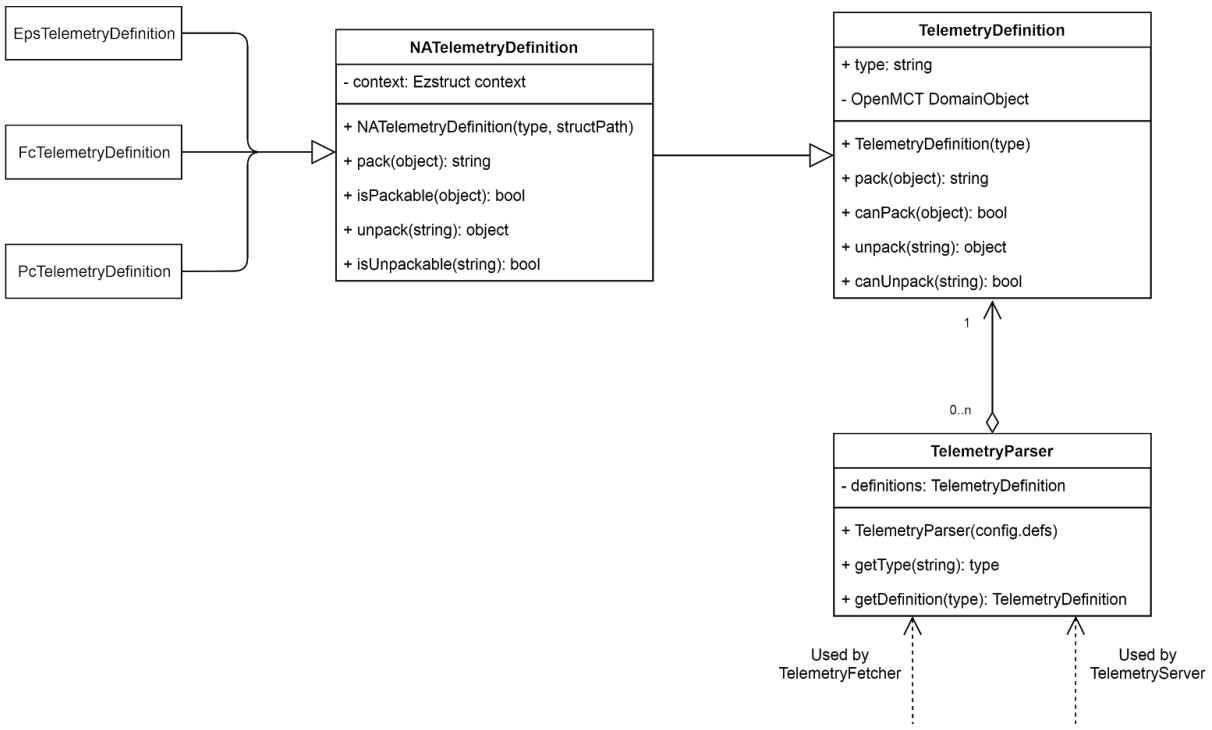


Figure 2.3.5 Class Diagram for telemetry parsing section

The next major part is the `DbManager`, which handles storing data (usually from a `TelemetryFetcher`) and reading data (usually from a `TelemetryServer`). The interface here is fairly simple, as the current implementation just has a single table that can be indexed by timestamp and type (which maps to a `TelemetryDefinition` that can be used to unpack the data stored for that type).

The class diagram for this subsystem can be found in Figure 2.3.6.

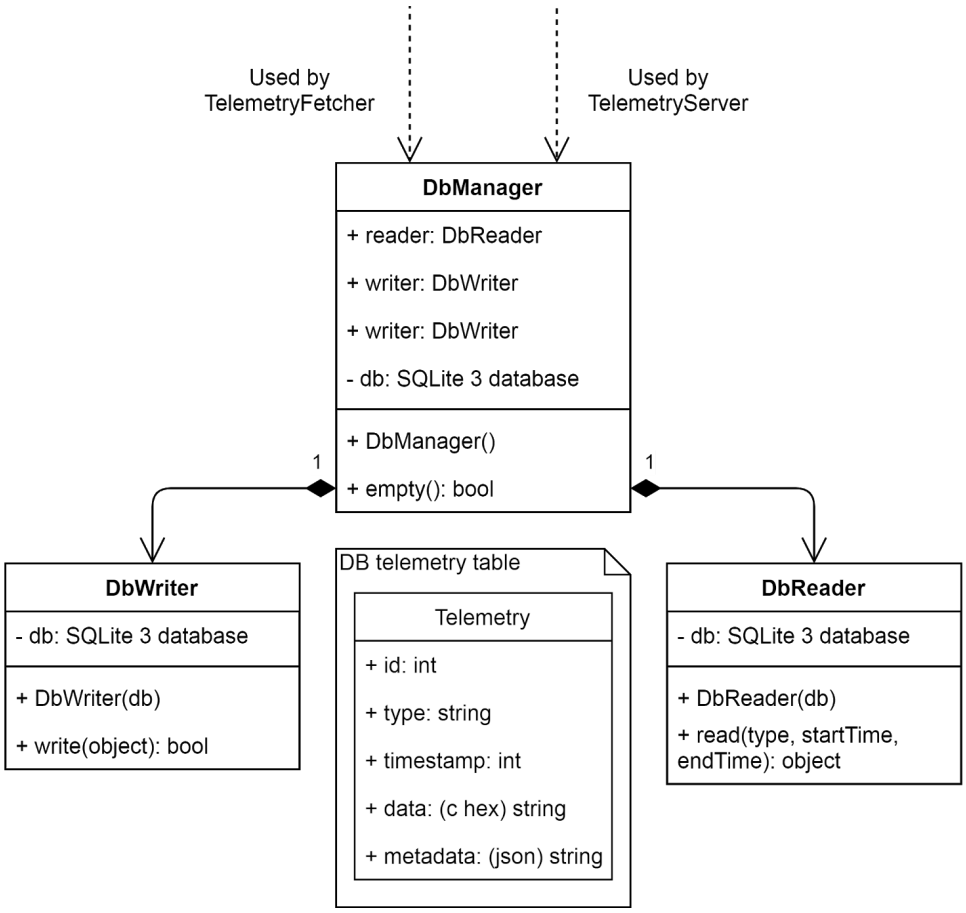


Figure 2.3.6 Class Diagram for database management section

To host the database it was decided to use SQLite; the fairly simple structure of the database used works well with it, and the large benefits it provides in portability (for “offline” work or backups) due to the database being stored in a single file made it a good candidate compared to other more complex database solutions.

2.3.4 Telemetry serving subsystem

This subsystem hosts and manages one or more HTTP/WebSocket servers for getting data for a specified `TelemetryDefinition` from a `DbManager`. The HTTP servers (implemented by `HistoryServer`) are used for allowing a service to request historical data for a period, while the WebSocket servers (implemented by `RealtimeServer`) allow a service to subscribe to real-time data updates without having to continuously poll a HTTP server.

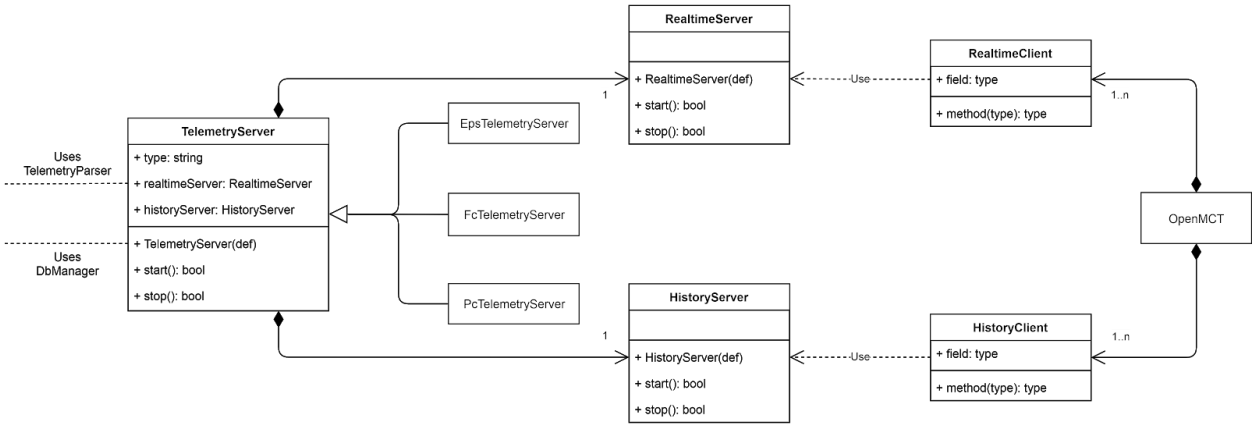


Figure 2.3.7 Class Diagram for telemetry server and OpenMCT connection

2.3.5 OpenMCT client plugins

These allow OpenMCT to get and subscribe to telemetry data using the interface described above in section 2.3.4, and allows mapping this data to various user-defined visualizations.

2.4 Code standards

It was decided to keep the code style and syntax similar to what's already encouraged and used by the OpenMCT project for both the backend and new frontend code required. A somewhat simplified version of OpenMCT's ESLint configuration file is used to enforce this code style across this project; the simplifications are mostly the removal of configuration specific to external dependencies not used in this project.

2.5 Third-party modules

Listing of all external modules, where/how/why they are used and links to their documentation; one section per module.

3 System usage

THIS SECTION IS IN ACTIVE DEVELOPMENT, AND WILL EXPERIENCE MAJOR CHANGES AS THE SYSTEM DESIGN DESCRIBED ABOVE IS STILL IN THE PROCESS OF BEING FINALIZED AND IMPLEMENTED.

3.1 Setup and configuration

3.1.1 Server setup

Ref. repo documentation - installing dependencies, startup, recommended server environment (system requirements?)

3.1.2 Configuration options

Documentation for config options - what can you change, and where?

3.1.3 Data definitions

How are data definitions structured, how to modify existing definitions (specific setup for NA files, uses c structs as template for generating definition - import new text file from NA)

3.1.4 Data sources

Choosing data sources: Read from file, get from web server, get from test server

3.2 Operation

3.2.1 Using OpenMCT

Navigating pre-configured views, creating new views, links to relevant OpenMCT documentation

3.2.2 Monitoring system status

Status indicators and system health - last time data received from local telemetry server, last time data received from external telemetry server, last time data received from spacecraft

3.3 Expansion options

3.3.1 Adding new data sources

Documentation on how to set up new fetcher for data source, setting up definition for parsing and storing the data, setting up OpenMCT plugin to read data

3.3.2 Telemetry API

Allows access to reading unpacked and formatted historical or real-time telemetry data by other services, such as the ground station

4 Current status

See also the issue tracker for this project at https://github.com/AudunVN/hypso-openmct/issues for up-to-date and more detailed descriptions of the status and remaining work required for each subsystem.

Table 4: Module Status Summary

Module	Status
Shared modules	Mostly finished, tests missing
Telemetry fetching	Currently working on loading previously downloaded data file from NA's server, already tested getting data from NA
Data management and storage	Mostly finished, tests missing
Telemetry serving	Work in progress
OpenMCT client	Work in progress, main blocker
Documentation	Work in progress

4.1 Plan

A minimum viable product (MVP) for the system is currently the top priority, and is planned to be finished by early April. This will cover getting basic EPS, FC and PC data from NanoAvionics' server and displaying it in OpenMCT, but will not yet have all the other functionality specified in this document (such as complete unit test coverage or end-to-end testing).

The current main point of uncertainity is the exact implementation of the OpenMCT plugin and data definitions; the rest of the system is fairly well-defined and straightforward to implement.

5 List of abbreviations

Table 5: List of Abbreviations

Abbreviation	Description
DB	Database
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation: Common data-interchange format for web services
NA	NanoAvionics
NASA	National Aeronautics and Space Administration
OpenMCT	Open Mission Control Technologies: https://nasa.github.io/openmct/about-open-mct/