Galileo's OSNMA for Mass Market GNSS Receivers

Design, implementation, configuration and usage of a Python Library for Galileo's OSNMA

Prepared by: Marc Cortés-Fargas
Joan Ametller Esquerra

Date: 01-12-2022
Reference: EDU-GAL-OSNMA-0001
Issue: 1.0

Filename:edu-gal-osnma-0001-1.0-osnma for python

Table of Contents

Table of Tables

Table of Figures

Document Changelog

Issue	Date	Location	Changes
1.0	27/11/2022		First issue

List of Acronyms

Table 1-1 Acronym list, extracted from [RD-1] Annex A

Acronym	Meaning
ADKD	Authentication Data & Key Delay
AES	Advanced Encryption Standard
BID	Block ID
CID	Chain ID
CMAC	Cipher-based Message Authentication Code
CPKS	Chain and Public Key Status
CREV	Chain Revoked
DSM	Digital Signature Message
DSM-KROOT	DSM for a KROOT
DSM-PKR	DSM for a PKR
DU	Don't Use
ECDSA	Elliptic Curve Digital Signature Algorithm
EOC	End Of Chain
GSC	European GNSS Service Centre
GST	Galileo System Time
HF	Hash Function
HKROOT	Header and KROOT
HMAC	Hash-based Message Authentication Code
ICD	Interface Control Document
IOD	Issue of Data
ITN	Intermediate Tree Node
KROOT	Root Key
MAC	Message Authentication Code
MACK	MAC and Key
MACLT	MAC Look-up Table
MACSEQ	MAC Sequence
MF	MAC Function
MID	Message ID
MSB	Most Significant Bit
NB	Number of Blocks
NMA	Navigation Message Authentication
NPK	New Public Key
NPKID	New Public Key ID
NPKT	New Public Key Type
OP	Operational
OS	Open Service
PK	Public Key
PKID	Public Key ID
PKR	Public Key Renewal
PKREV	Public Key Revocation
PRN	Pseudo Random Noise

SHA	Secure Hash Algorithm
SIS	Signal In Space
TESLA	Timed Efficient Stream Loss-Tolerant Authentication
TOW	Time of Week
WN	Week Number

1. INTRODUCTION

1.1. OSNMA

OSNMA (**O**pen **S**ervice **N**avigation **M**essage **A**uthentication) is one of Europe's Galileo coolest features: the authentication of GNSS Data. As more and more devices rely on PNT solutions provided by GNSS receivers, the impact of attacks over GNSS signals may have a higher disruptive effect over society. Current dependency on GNSS signals is very high already, being 6-7% of the European GPD dependant on this technology.

Most GNSS systems did not add any mechanism to civil users to protect against these attacks until the addition of OSNMA. OSNMA is a unique Galileo feature that allows any standalone receiver (i.e., not connected to the network) to make sure that navigation data transmitted by satellites has not been altered by an attacker. OSNMA does not protect the user from all possible attacks (e.g., meaconing) but it adds a protection layer against most common spoofing ones, and is expected to be an extra layer of security that can work together with other anti-spoofing/anti-jamming techniques.

The design of and implementation of OSNMA is significantly complex, as it needs to deliver a protection mechanism by using a very small bandwidth and, at the same time, providing a mechanism which can be implemented in mass market chipsets with limited computational power.

Note: the work in this library is not related to ublox as a company, we only use their products as they are well-documented and cost-effective. UBLOX is a trademark of u-blox Holding AG.

OSNMA ADDED VALUE IN A NUTSHELL

Figure 1-1 shows the general GNSS process particularised for Galileo. In the step 1, the Receiver performs the acquisition and tracking of the Signal in Space and also obtains the Navigation Message[1], decoding it in its own particular format, which is explained in each Receiver's Documentation. This is what will be called in this document as "Data Acquisition". The outputs of this step is the navigation data and the *pseudoranges* (i.e. estimated distance between the receiver and each satellite), what will allow the receiver (in step 3) to calculate its position and time. OSNMA provides a mechanism to protect alterations of the former one.

In Step 2, it is shown how the data is transformed from the Receivers format into the Galileo's ICD Format. This is what we call "Data Transformation". As the reader can deduct, Step 1 and Step 2 depend on the receiver's manufacturer.

The Navigation Data includes parameters to compute the user's position (Step 3). Along this document this step will be called as "Data Processing", and will be the core of the library. It does not depend on the receiver's manufacturer.

[1] The Space Segment, besides the navigation message, also sends the carrier and ranging code, but basics on GNSS are outside of this document, see [RD-6] for details.

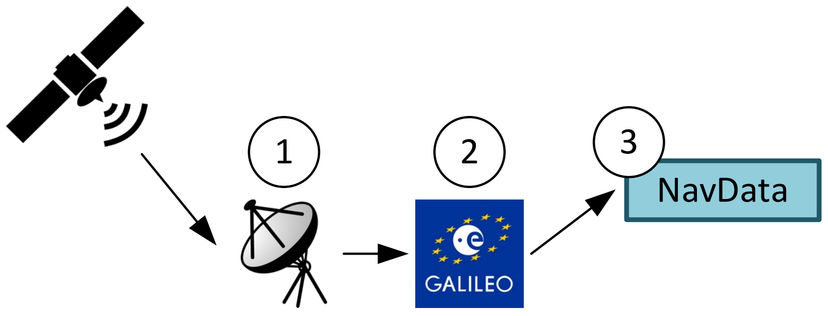


Figure 1-1 Galileo General Schema

One of the main threats to this processing chain is that there are simple devices that can easily "impersonate" the GNSS satellites, sending altered navigation data that the receiver will process as if coming from a real satellite. This is known as spoofing and it can potentially lead receivers to fake positions.This is illustrated in the following figure:

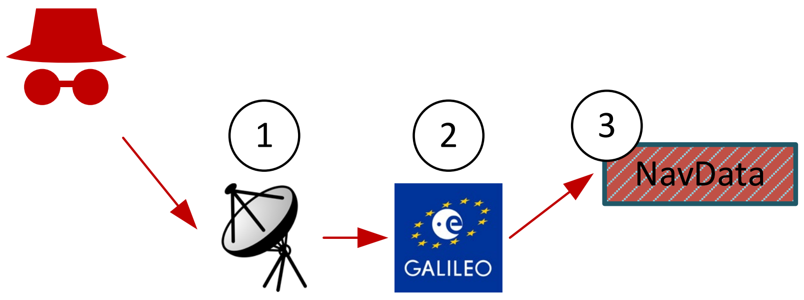


Figure 1-2 Spoofing

This attack is possible mainly because navigation signals are one of the weakest RF signals processed by a receiver what allows an attacker to easily overlap them with a low power transmitter. Moreover, the nature of the navigation data allows easily to be predicted and faked by an attacker. In order to solve that, Galileo Satellites send, besides the Navigation Data, a means for it to be authenticated – the OSNMA. This way, spoofing is much less likely to happen, and add an extra layer of security to any GNSS based application.

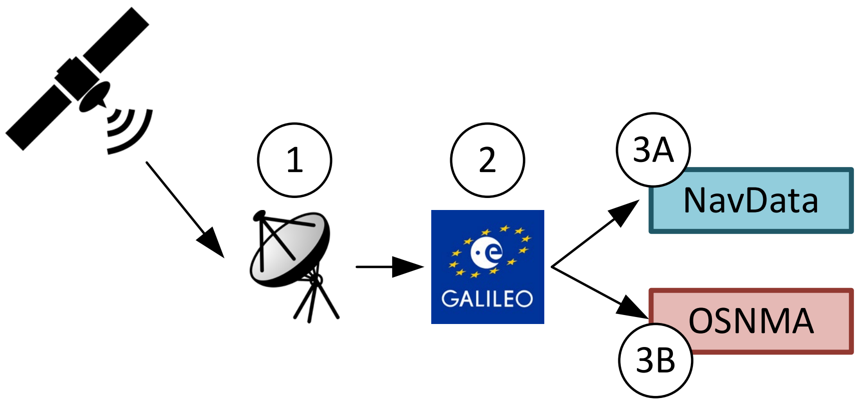


Figure 1-3 Galileo General Scheme with OSNMA

1.2. SCOPE OF THIS DOCUMENT

The main goal of this project is implementing the authentication of the Galileo Open Service navigation message coming from Galileo Satellites receiving them using a mass-market GNSS receiver. The idea is first implementing a prototype based on python to obtain the knowledge and identify the main difficulties of implementing OSNMA. After that, the goal will be doing a final implementation to be used as a library by most receivers with the idea of fostering its use amongst the community.

This document presents the design and implementation SW solution to demonstrate OSNMA capabilities with mass-production receivers.

The library seen in §4 provides any user the means to implement OSNMA in mass market receivers that can be bought for really cheap -e.g., *this NEO-M9N*. Specifically, this library has been developed based on Ublox Receivers, although its core can work for any GNSS receivers that can get Galileo I/NAV Pages.

All the acronyms used in this document can be found in the different reference documents used to create the library provided in §4.

1.3. SCOPE OF THE LIBRARY

The scope of the library includes:

1. Transformation of uBlox Words into Galileo ICD as per [RD-2] from a live receiver;
2. Process Galileo Pages in such fashion a sub-Frame can be formed;
3. Saving the navigation data / OSNMA data in an indexable format (e.g., with the GST associated to the sub-frame) associated to each satellite (also known as Space Vehicle);
4. Transform OSNMA Sub-Frames into DSM and MACK;
5. Verification of KROOT;
6. Verification of Tesla Key (i.e., if Tesla Key belongs to Key Chain);
7. Authentication (i.e., tags verification), including:
   7.1. Self-Authentication; and
   7.2. Cross-Authentication

It is out of the scope of this version of the document:

- **The root key update Over the Air (DSM-PKR).**
- **MACSEQ verification, since currently no flexible tags are expected in MACL table (see Annex C from [RD-1]).**

These issues will be addressed in a future version of the implementation.

1.4. CONTENTS OF THIS DOUMENT

1. Section 1: Introduction, with the scope of the document, the reference documentation and the authors;
2. Section 2: The software design, including a functional tree, the UML structural/behavioural diagrams and brief justifications behind the design decisions. It also includes the interfaces and the methods used by the different Python Classes;
3. Section 3: this section presents the Software Operations Manual, with a set of screenshots and instructions for the End-User to understand the Library provided in the Annex 1, including the configuration of an uBlox receiver via an external library to get the Galileo Navigation Pages;

4. Section 4: provides the future upgrades and updates that are outside the release of this version;
5. Annex 1: this section provides the Source Code, as well as a direct link with the GitHub Repository; and
6. Annex II: this section provides the events generates by the different classes, along with explanations.

1.5. REFERENCE DOCUMENTATION

[RD-1] The European GNSS (Galileo) Open Service Navigation Message Authentication (OSNMA) User Interface Control Document for the Test Phase
Short title: OSNMA User ICD for the Test Phase
Reference: N/A
Issue: 1.0

[RD-2] The European GNSS (Galileo) Open Service Signal-In-Space Interface Control Document
Short title: OS SIS ICD
Reference: N/A
Issue: 1.0

[RD-3] NEO-M9N - Standard precision GNSS module - Integration manuale European GNSS (Galileo) Open Service Signal-In-Space Interface Control Document
Short title: NEO-M9N Integration Manual
Reference: UBX-19014286
Issue: R06

[RD-4] The European GNSS (Galileo) Open Service Navigation Message Authentication (OSNMA) Receiver Guidelines for the Test Phase
Short Title: OSNMA Receiver Guidelines for the Test Phase
Reference: N/A
Issue: 1.0

[RD-5] The European GNSS (Galileo) Open Service Signal-In-Space Interface Control Document
Short title: OS SIS ICD
Reference: N/A
Issue: 2.0

[RD-6] GNSS Data Processing Volume I: Fundamentals and Algorithms
Short title: OS SIS ICD
Reference: ESA TM-23
Available online on https://gssc.esa.int/navipedia/GNSS_Book/ESA_GNSS-Book_TM-23_Vol_I.pdf

[RD-7] GitHub Repository – osnmaPython by @astromarc
Available online on https://github.com/astromarc/osnmaPython

1.6. ABOUT THE AUTHORS

Marc Cortés-Fargas is a Senior Systems Engineer with strong experience analysing, specifying and designing complex aerospace systems. He started his career at AIRBUS where he first dealt with complex aerospace systems, developing technical solutions that ended up being a patent from which he is the main inventor. Since 2019, he works as a Systems Engineer within Galileo Ground Segment at Indra and at Thales Alenia Space. Within this period, he gained a deep knowledge of Galileo architecture. He actively participated in

Galileo's Element design, specification and interfaces definition interacting with consortium members. He also has an important role within the technical team, participating in the Design, Development and AIV including the review of subcontractor deliveries, including a variety of elements within the GMS and GSF segments.

Joan Ametller Esquerra is a Senior Systems Engineer with deep knowledge of Navigation Programs Ground Segments and their software systems since 2006. He started his career in the development of Galileo IOV participating in ULS and GSS development and later on in the GNSS Service Centre (GSC). He was also Technical Manager of PRS activities that Indra started in 2015, due to his background on cryptography and security acquired during his PhD studies. When WP2X contracts started, he took an horizontal role as Software Systems Engineer in all WP2X Indra Contracts (KMFs, GSMC and POCP-S) participating in all the projects analysing specifications, making designs and guiding teams to design and implementation phases. He also participated in EGNOS ground segment (RIMS A G2, NLLP and NLESv3). an experienced GNSS engineer with PhD studies in cryptography and more than 15 years of experience in the Space Industry.

1.7. LEGAL ASPECTS

The software, design and in general all contents in this document comes with absolutely no warranty. The code itself has the license referred in the GitHub repository -see [RD-7].

GALILEO is the Global Navigation Satellite System by the European Union. The library here it is possible only thanks to the implementation of the OS-NMA capability, currently uniquely in all the GNSS systems. This library is then powered by GALILEO.

Ublox is a Swiss company that creates, among other products, GNSS receivers. The work in this document is not related at all to Ublox. An ublox receiver has been chosen as the GNSS Receiver for this library just because is one of the best GNSS receivers for mass market applications.

2. SOFTWARE DESIGN

2.1. DESIGN METHOLOGY

The programming language used for this project is Python. Even though, as a port to C/C++ is planned, the design is aimed to use as few as external libraries or not compatible libraries as possible.

Python has been chosen as it is a multi-platform language with a lot of useful built-in libraries. Moreover, it is a high-level language which allows rapid prototyping.

2.2. FUNCTIONAL TREE

In this section the functional Tree is shown. As no formal requirements have been written, this functional tree is directly derived and inferred from §1.3 and supported by [RD-1], [RD-2] and [RD-4].

As it can be seen, the functional tree is divided in four big blocks:

1. Data Acquisition;
2. Data Transformation;
3. Data Processing; and
4. Data Display.

As shown in Figure 2-28, this library is focused in following a loop, in which for each input data, some values are updated and shown to the user. It is envisaged in this fashion as this library is aimed to be used in live applications in which the Navigation Data is live-authenticated.

After the table, dedicated section explaining the main parts of the Functional Tree can be found.

Please notice that this functional tree will be one of the main inputs when doing the structural design.

Table 2-1 OSNMA Functional Tree

1	**Data Acquisition**
1.1	Real-Time Data Acquisition
1.2	Test Data Acquisition
1.3	Save Real-Time data to Test-Data
2	**Data Transformation**
2.1	Test-Vectors [RD-5] to Galileo ICD
2.2	Test-Data to Galileo ICD
3	**Data Processing**
3.1	Sub-Frame Sequencing
3.2	**OSNMA**
3.2.1	OSNMA Message Sequencing
3.2.2	**Navigation Data Authentication (MACK Processing)**
3.2.2.1	Self-Authentication
3.2.3.2	Cross-Authentication
3.3	**DSM: Digital Signature Message**
3.3.1	DSM Block Sequencing
3.3.2	**DSM-KROOT**

3.3.2.1	Root Key Authentication
3.3.2.2	Tesla Key Chain Authentication
3.3.3	DSM-PKR
4.	**Data Display**

2.2.1. DATA ACQUISITION

This specific function is highly GNSS-Vendor dependant. The solution chosen in this implementation relies on a M9N Ublox Receiver -see §3.1 for more information about the chosen model. In order to get the Galileo Pages, the receiver must be configured as seen in §3.1.1.1.

This module includes:

1. Getting live data (uBlox Words) from the receiver, and save it into a CSV file for further processing; and
2. Getting data from a CSV file containing the Ublox Words (e.g., saved from the previous functionality). This functionality is especially envisaged for testing purposes.

2.2.2. DATA TRANSFORMATION

The data extracted from an uBlox receiver follow this format (once configures to get Galileo Pages):

Table 2-2 uBlox SFRBX (part 1/2)

Field	gnssId	svId	reserved0	freqId	numWords	chn	version	reserved1
Example	Galileo	27	1	0	9	**1**	2	0

Table 2-3 uBlox SFRBX (part 2/2)

Field	**dwrd_01**	dwrd_02	dwrd_03	dwrd_04	dwrd_05	dwrd_06	dwrd_07	dwrd_08	dwrd_09
Example	**9786709**	1431655765	1431655765	1378484224	3006251008	42	2863294065	499073024	1

It is necessary to get the information from the previous tables into the Galileo ICD. For this, we will store the svId, which identifies the satellite, and the “dwrd” (also called Ublox’s Words) from 01 to 08. The matching of this information with the Galileo ICD is as per the following Figure:

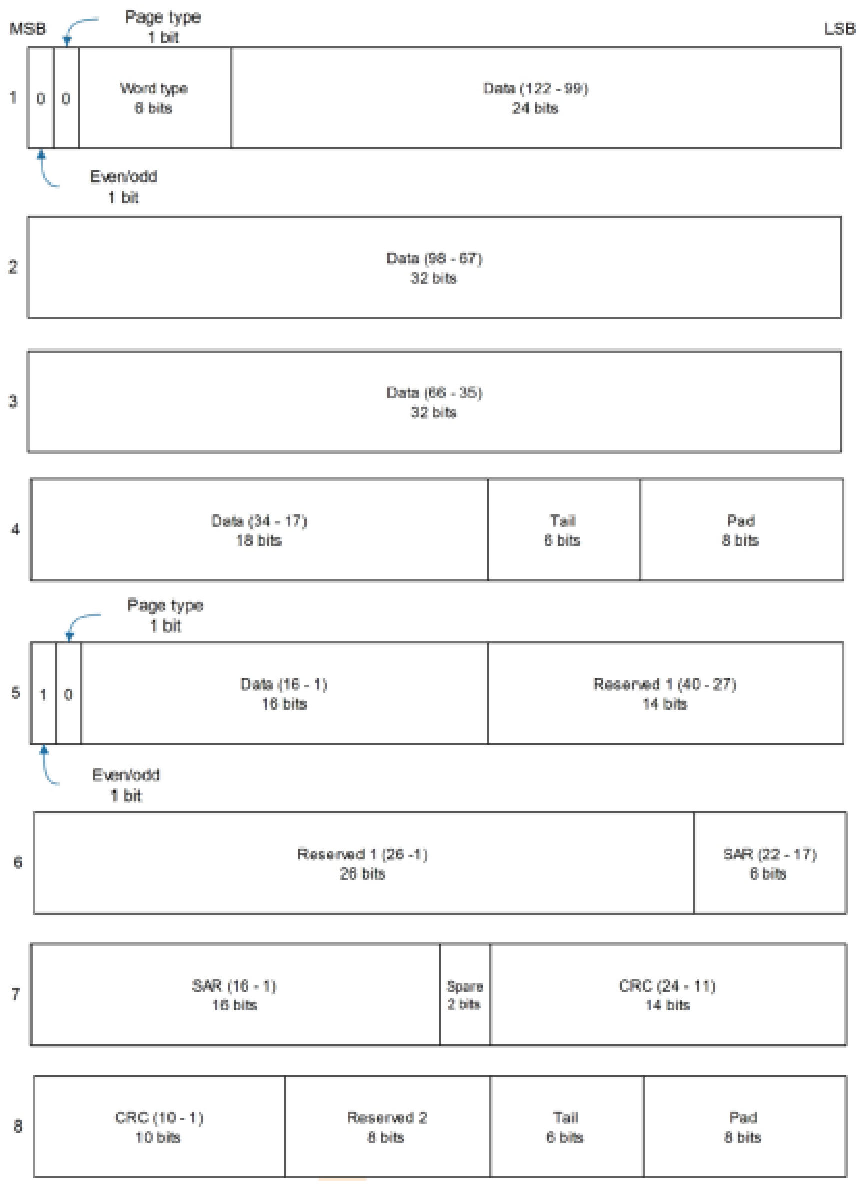
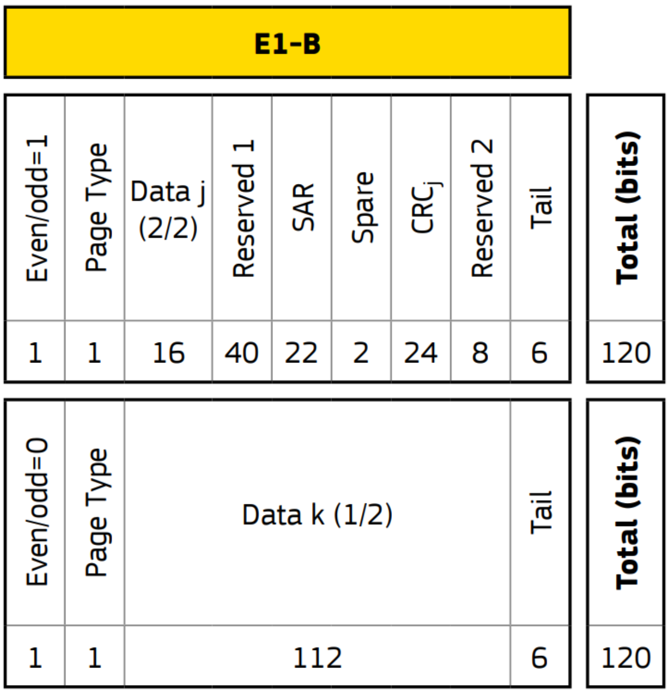


Figure 2-1 uBlox Galileo Pages, extracted from [RD-3]

The figure shows how ublox *dword*'s are assembled together to build Galileo pages as they are specified in the ICD, taking into account that Data J is data in ublox *dword* 5, and Data K is present in ublox *dwords* 1, 2, 3, and 4. Reserved 1, which contains the OSNMA information, is present in ublox *dwords* 5 and 6.

It is also worth to mention that, as present in Ublox Documentation, their GNSS receiver only provides data that are CRC verified, hence our Software will not have a CRC verification. This might be included in future versions.



E1-B

Even/odd=1	Page Type	Data j (2/2)	Reserved 1	SAR	Spare	CRC_j	Reserved 2	Tail	**Total (bits)**
1	1	16	40	22	2	24	8	6	120

Even/odd=0	Page Type	Data k (1/2)	Tail	**Total (bits)**
1	1	112	6	120

Figure 2-2 I/NAV Nominal Page with Bits Allocation. Extracted from [RD-2]

It is then necessary an object or a set of objects that parse the uBlox *dwords* contents into the Galileo pages (Data, Reserved 1 (osnma), SAR…) as they are specified in Galileo SiS ICD.

It is also foreseen some sort of page processor, in which the Galileo odd/even pages are introduced and the Value as per Galileo ICD are obtained. This is done this way because the Galileo Test Vectors are usually provided at page level.

2.2.3. DATA PROCESSING

According to [RD-2] §4.3, Galileo messages are organised in Frames, which at the same time are organised in sub-frames that are in turn, organised into Pages.

Each two seconds, the Galileo Satellites transmit a Nominal Page (composed of an Even and Odd Page). Hence, the Galileo Message timeframe is as per the following figure:

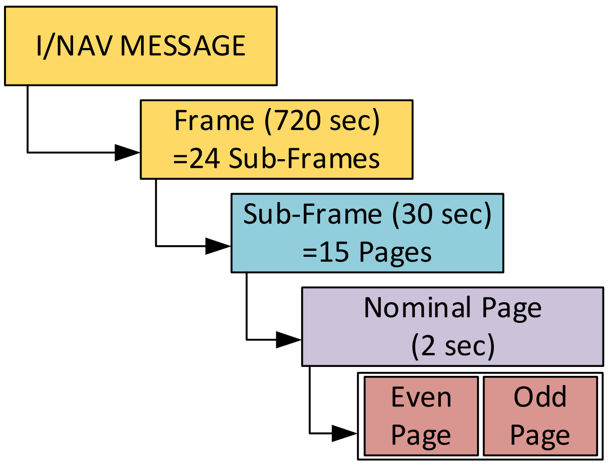


Figure 2-3 Galileo I/NAV Message Timeframe. Adaptation from [RD-2].

As it is explained in [RD-1], the minimum Data Structure necessary to perform OSNMA activities is the Sub-Frame. The following section will explain how to get the Navigation Data and the OSNMA information from the Galileo Sub-Frame.

2.2.3.1. SUB-FRAME SEQUECING

The following Figure depicts the Galileo Sub-frame sequencing:

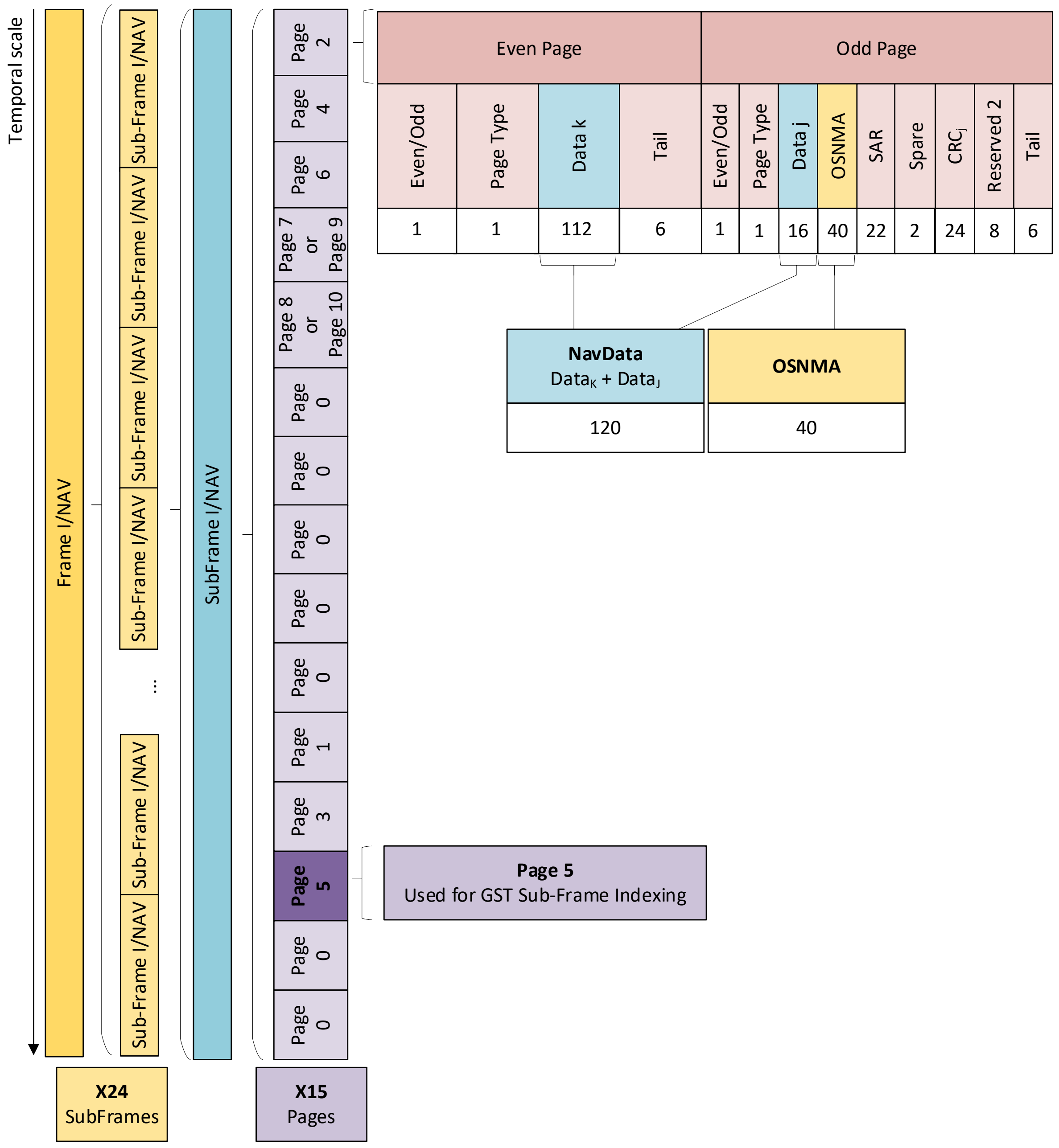


Figure 2-4 Galileo I/NAV Message Structure according to SiS ICD issue 1 [RD-2]

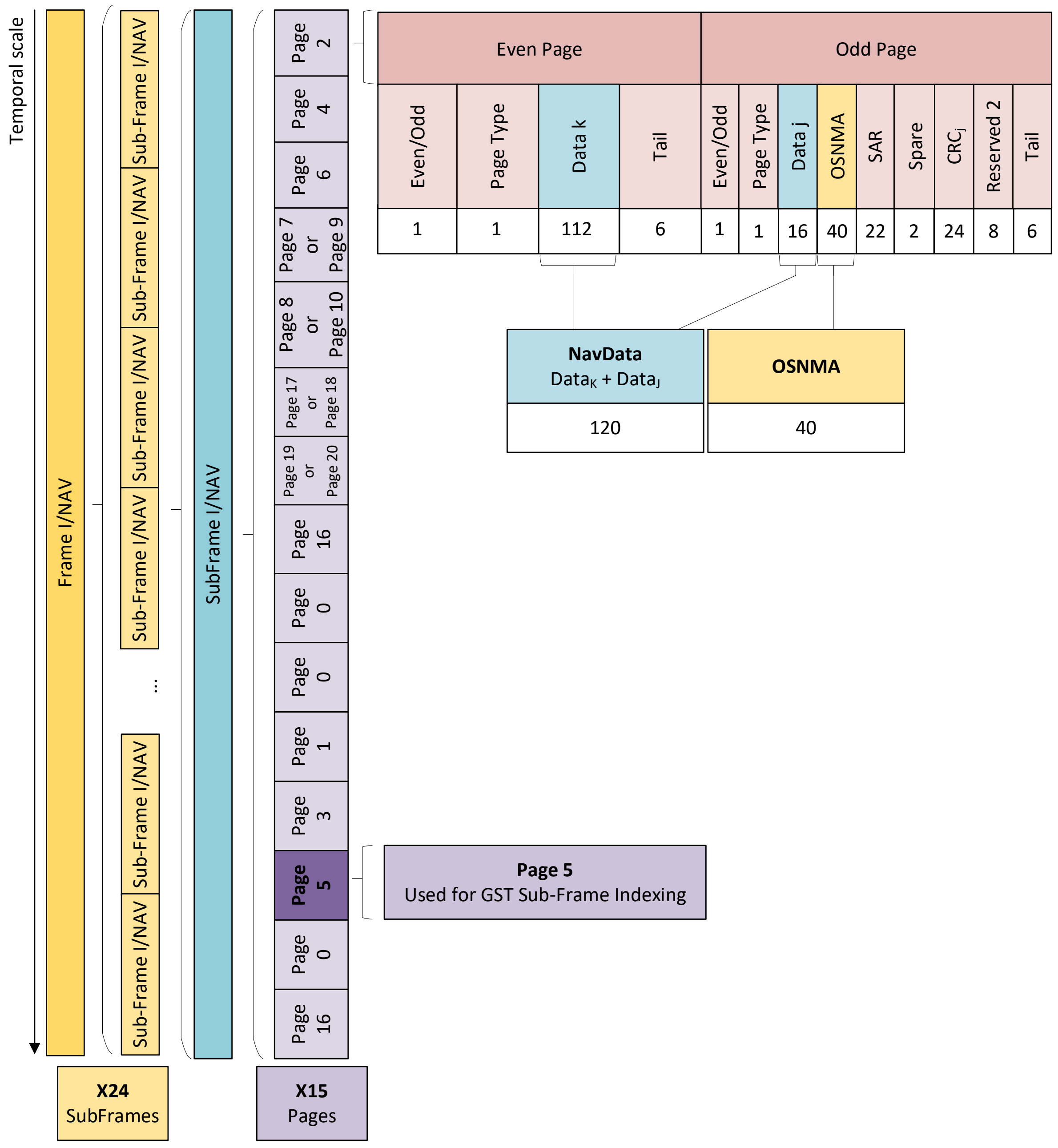


Figure 2-5 Galileo I/NAV Message Structure according to SiS ICD issue 2 [RD-5]

As explained, each Frame is composed of 24 Sub-Frames, and each Sub-Frame is composed of 15 Pages. Each Page is composed of an Even Page and an Odd Page.

As it can be appreciated in the figure, all the OSNMA information is concentrated in a 40 bits section inside each odd page, so the bandwidth that OSNMA can use, for each satellite, it's about 40 bits every two seconds (i.e. 20 bps) what reduces a lot the amount of information that can be added for that purpose.

To get the complete sub-frame, it is necessary to be highlighted that the Pages need to be obtained chronologically following the order depicted in Figure 2-4:[2]

2, 4, 6, 7 or 9, 8 or 10, 0, 0, 0, 0, 0, 1, 3, 5, 0, 0 → Page Sequence for SiS ICD issue 1

or

2, 4, 6, 7 or 9, 8 or 10, 17 or 18, 19 or 20, 16, 0, 0, 1, 3, 5, 0, 16→ Page Sequence for SiS ICD issue 2

Not receiving a page in the said order would mean that we have lost Synchronisation with the Space Vehicle and that we cannot complete the Sub-Frame. Hence, for OSNMA, the partial Sub-Frame obtained must be omitted and the authentication of some subframes of the data stream will not be possible.

As for OSNMA (see §2.2.3.2) the data from previous sub-frames need to be obtained, there is the need to provide some sort of "index" for each sub-frame based on the GST Time. According to [RD-1], this identifier for Sub-Frame has to be understood as the GST obtained at the beginning of the Sub-Frame. However, it needs to be noticed that Page 2 does not have the GST, so other pages should be used (i.e., Page 0 or Page 5, see [RD-2] §4.3.5).

It has to be noticed that according to Figure 2-4, the Page 5 has been chosen for GST Indexing instead of Page 0. The justification behind this design decision relies on the fact that Page 0 may be updated in future ICDs. Indeed, as it can be seen in [RD-5], Pages type 0 after Page 8 or 10 will contain Pages 17 or 18, and the last Page from the Sub-Frame will change from Page 0 to Page 16. Hence, in order for this OSNMA design solution to be consistent with future OS ICD evolutions, the Page to be used to compute the GST for indexing will be considered the page 5.

As GST transmitted in the even Page of Page 5, the formula to get the GST index is:

$$GST_{index} = GST_{Page\ 5} - 25\ seconds$$

2.2.3.2. OSNMA

OSNMA will be divided in different sections, according to the functional tree seen in §2.2.

2.2.3.3. OSNMA MESSAGE SEQUENCING

As per §2.2.3.1, the minimum unit of OSNMA to start to perform Authentication Activities correspond to the OSNMA of the SubFrame, following the same page sequencing explained in the said section.

Anyhow, the data in each page is divided in two parts:

- **HKROOT message** (first 8 bits of the OSNMA field within the page): used to get the OSNMA status, the key root and authentication information (length of key, hash algorithm…). In §2.2.3.5 this information is expanded; and

[2] Page Type and Word Type will be used as synonyms along this document. Anyhow, please notice that [RD-2] has a subtle difference between them: "Word" is considered as only the part of "Data" of the Page.

- **MACK** (last 32 bits of the OSNMA field within the page): used to get the Tesla Key as well as the tags to be compared with the computed HMAC. In the following section we will explain what does it contain.

Accordingly, some sort of object or functionality will need to be implemented to split the OSNMA message into HKROOT and MACK. Details on how to process MACK and HKROOT can be found in the sections below.

2.2.3.4. MACK. NAVIGATION DATA AUTHENTICATION (TAGS VERIFICATION)

The main aim of OSNMA is, as it names indicate, the authentication of the Navigation Data. Even though some pre-conditions are needed (check that Tesla Key belongs to a chain & verification chain's Root Key), this section will start with the authentication itself. Accordingly, for this section, it will be considered that Key Verifications is done in a previous flow (see §2.4.1 for details).

Navigation Data Authentication will be divided into Self-Authentication and Cross-Authentication.

Even though it is divided into those two sections, the solution approach is similar in both cases. Considering a single Space Vehicle and ADKD 0 or 4 (data validated is the previous one sent, no delayed authentication):

- **The first 30 seconds (i.e., in the first Sub-Frame)**, it is received the Navigation Data and it is stored by indexing it with the GST_{index};
- **In the Second Sub-Frame**, the OSNMA data is received and the Tags Information is extracted. This information will contain the HMAC and some extra information to compute the HMAC. The Tag Information is stored by indexing it with the GST_{index}; and
- **In the third subframe**, the OSNMA is received and the Tesla Key is extracted. Then, by using the Tesla Key, the information provided in the Tags in the previous subframe, and the navigation data received in the first subframe, the HMAC can be computed and finally compared with the Tag Received in the second SubFrame.

Please notice that, even though the previous three steps are separated, in all sub-frames all the information is stored. E.g., in the first step the Navigation Data is stored, as well as the OSNMA information that will be used to verify their correspondent Navigation Data.

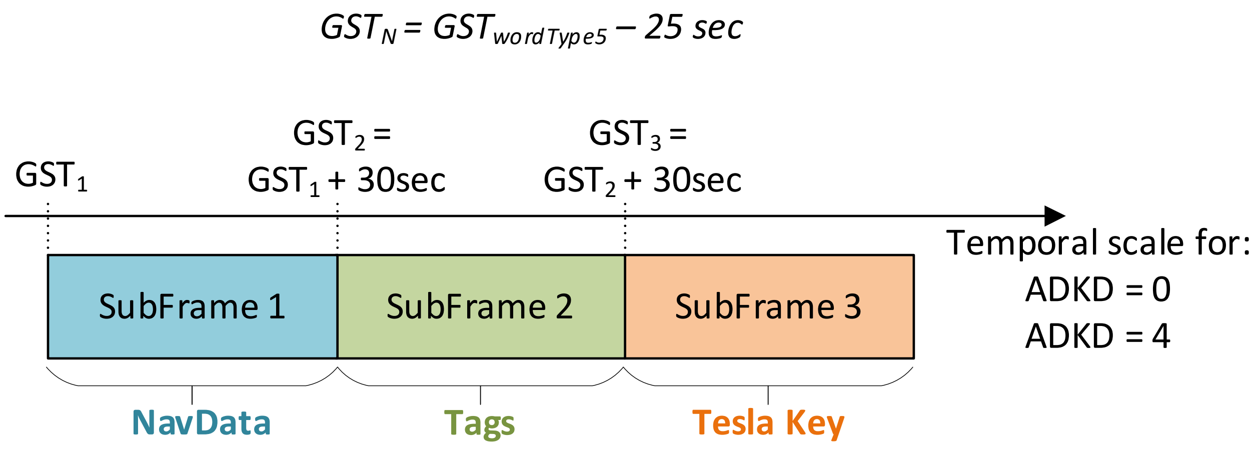


Figure 2-6 Timescale of NavData, Tags and Tesla Key for ADKD 0 and 4

2.2.3.4.1. SELF-AUTHENTICATION

Self-Authentication refer to the capability that a Space Vehicle has to verify the data that itself has previously transmitted. This approach is valid for only the Space Vehicles which are distributing OSNMA Information and, as it is indicated in [RD-1], not all Space Vehicles will be transmitting OSNMA at the same time.

The Self-Authentication is done by using the “Tag0”, and the verification Scheme follows exactly what is seen in Figure 2-6:

1- In the First Sub-Frame, the Navigation Data (and OSNMA Data, even though it is not used) is saved;
2- In the second Sub-Frame, data (navigation & OSNMA) is saved;
3- In the third sub-Frame, the data (navigation & OSNMA) is saved and:
    a. From the Third SubFrame, OSNMA Data, the Tesla Key is Extrated;
    b. From the Second SubFrame, OSNMA Data, the Tag0 is Extracted
    c. From the First SubFrame, the Navigation Data is extracted and arranged; and
    d. By using the Tesla Key and the Navigation Data, the HMAC is computed, and it is compared with the Tag0 from the Second SubFrame.

2.2.3.4.2. CROSS-AUTHENTICATION

Cross-Authentication is an OSNMA feature that can be used to:

- Authenticate Data from those Galileo Space Vehicles which are not transmitting OSNMA; or
- Authenticate Data from other GNSS Constellations (e.g., GPS -even though this is currently part of a future implementation).

In this case, the authentication schema is as follows:

1- During the first 30 seconds, three different SubFrames are received from three different Space Vehicles (SV_A, SV_B and SV_C). We will consider that the only Space Vehicle transmitting OSNMA is SV_A;
2- During the second 30 Seconds, the SubFrame 2A is received. Please notice that in order to perform the cross-authentication of SV_B and SV_C first subframe, no more data is needed from SV_B and SV_C.

Moreover, when processing the Tags from the OSNMA 2A extracted from SubFrame 2A, we find out information of the cross-authenticated satellites. In this example, we will consider that in the Tags there is information about SV_{CB}, but no information of SV_C. Accordingly, Navigation Data from SV_C cannot be authenticated.

3- When the Tesla Key is extracted from OSNMA 3A from SubFrame 3A, this can be used to perform Self-Authentication with data from OSNMA 2B and NavigationData 1A (both from SV_A).
Besides, the Tesla Key from OSNMA 3A can be used to compute HMAC along with the NavigationData from SubFrame1B, and then be compared with the tag corresponding to SV_B found in OSNMA 2A.

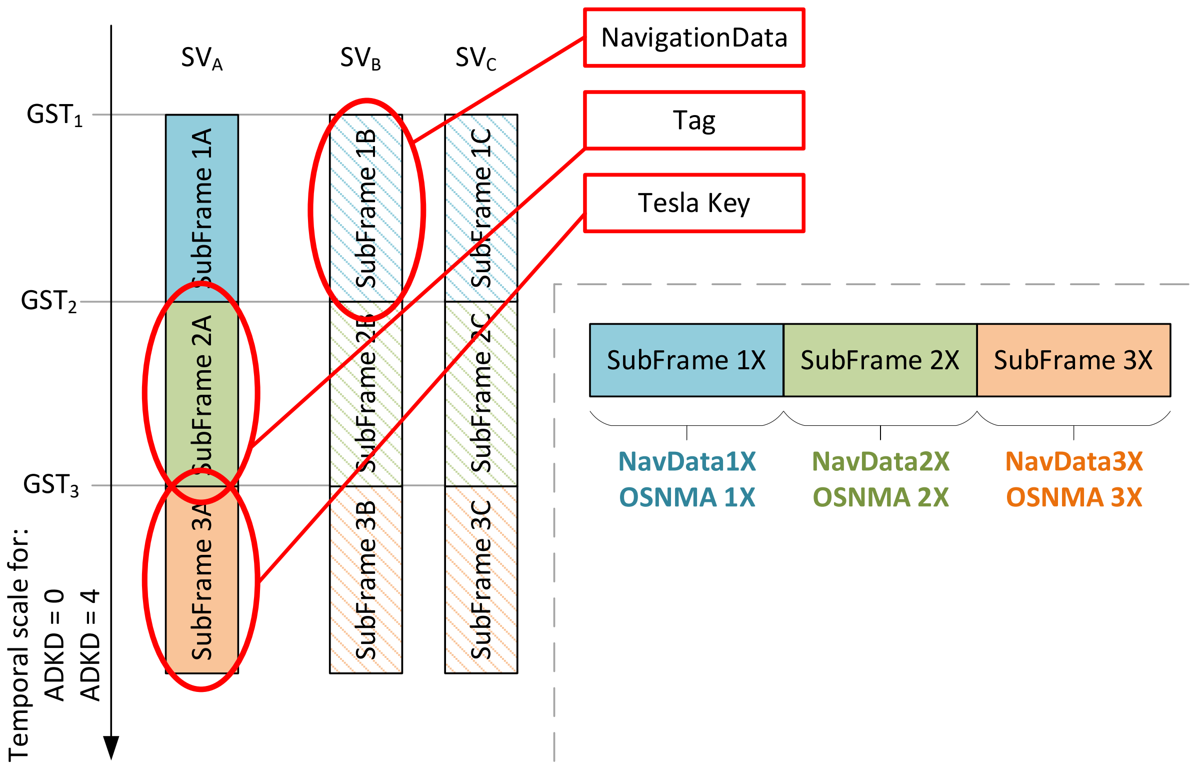


Figure 2-7 Cross-Authentication Example

Please notice that in all the previous examples the ADKD 0 and ADKD 4 is considered. This means that the Navigation Data has a delay of two SubFrames with respect to the Transmitted Tesla Key. The difference between ADKD 0 and ADKD 4 is the parts of the navigation data which are used for computing the HMAC (see Annex B from [RD-1] for details).

In the ADKD 12, the navigation data is as per the following figure (considering that SV_A authenticates SV_B): the key is transmitted with a delay of 10 sub-frames with respect to the Tag Info. That Means that the time of the Tags is delayed 330 seconds from the current GST and the Navigation Data is delayed 360 seconds from current GST.

For the Cross-Authentication, it is also worth to mention that sometimes the PRND (the satellite from to check the Navigation Data) is set to 255. This case will be considered as self-authentication.

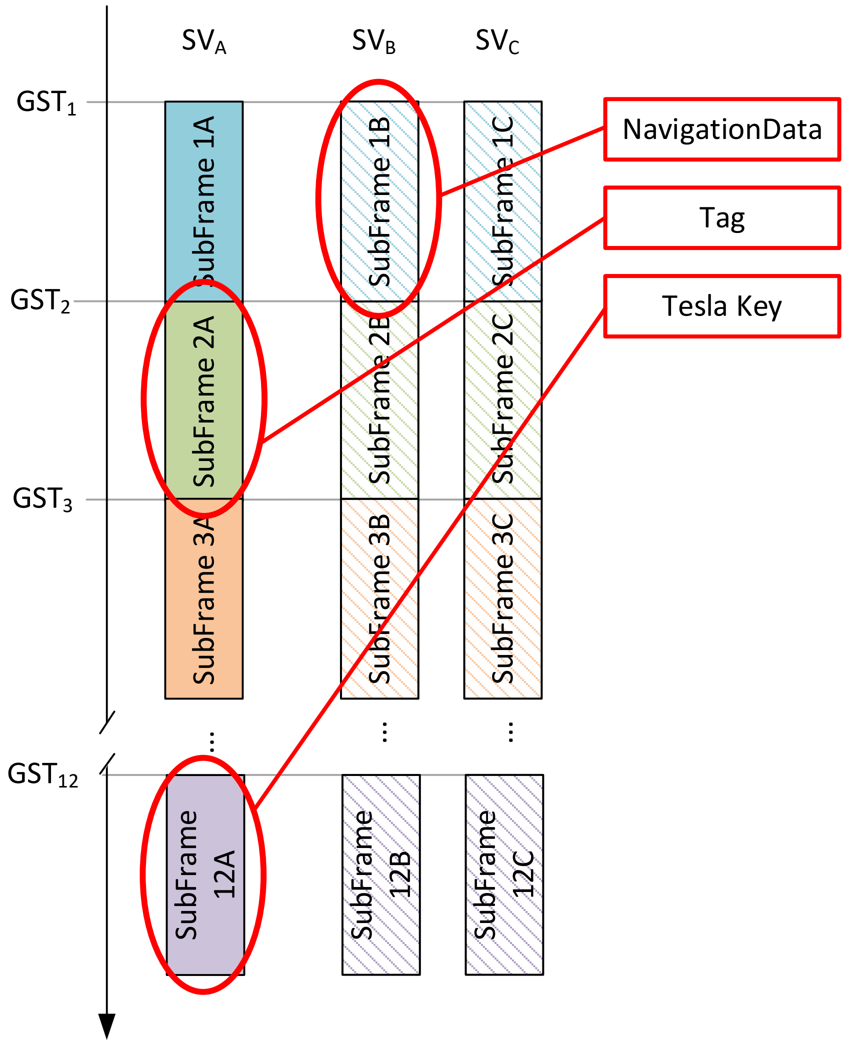


Figure 2-8 Cross-Authentication with ADKD12 (delayed MAC)

2.2.3.5. DSM: DIGITAL SIGNATURE MESSAGE

When all the first 8 bits of the OSNMA part from a Galileo Page corresponding to the same Sub-Frame are arranged, we have the HKROOT.

The HKROOT is divided in two parts:

- NMA Header (providing Navigation Message Authentication Status and more); and
- DSM, which can be used for whether:
    - DSM-PKR:[3] which provides the Public Key Renewal; and
    - DSM-KROOT: Provide the digitally signed KROOT.

2.2.3.5.1. DSM BLOCK SEQUENCING

[3] Outside of the scope of this issue of the document & code

The Space Vehicles transmitting OSNMA also transmits the DSM Blocks within the HKROOT part of the OSNMA. The DSM Blocks, received from different Space Vehicles, are connected all arranged to get the global DSM Message.

It is worth to consider:

- Over the time, different DSM Messages can be transmitted. Accordingly, the Galileo Constellation have a current DSM Message and the historic DSM Messages (if those are saved). This consideration is especially important as the DSM Message ID shall be checked before feeding the DSM Message from the Space Vehicle's SubFrame.
- The Space Vehicles send the DSM Blocks as follows:
  - Each Space Vehicle, send the DSM Block in a sorted way: first the DSM Block with ID0, then with ID1, then with ID2 and so on;
  - The Space Vehicles send different DSM Blocks among them. That means that whilst Space Vehicle 1 send de DSM Block 0, the Space Vehicle 2 may be sending, at the same time, the DSM Block 3. The multiplexing of different blocks by different satellites is done to maximise the reception speed of the whole message by receivers.

The following figure exemplifies an ideal case in which the DSM Message is get within 1 minute 30 seconds:

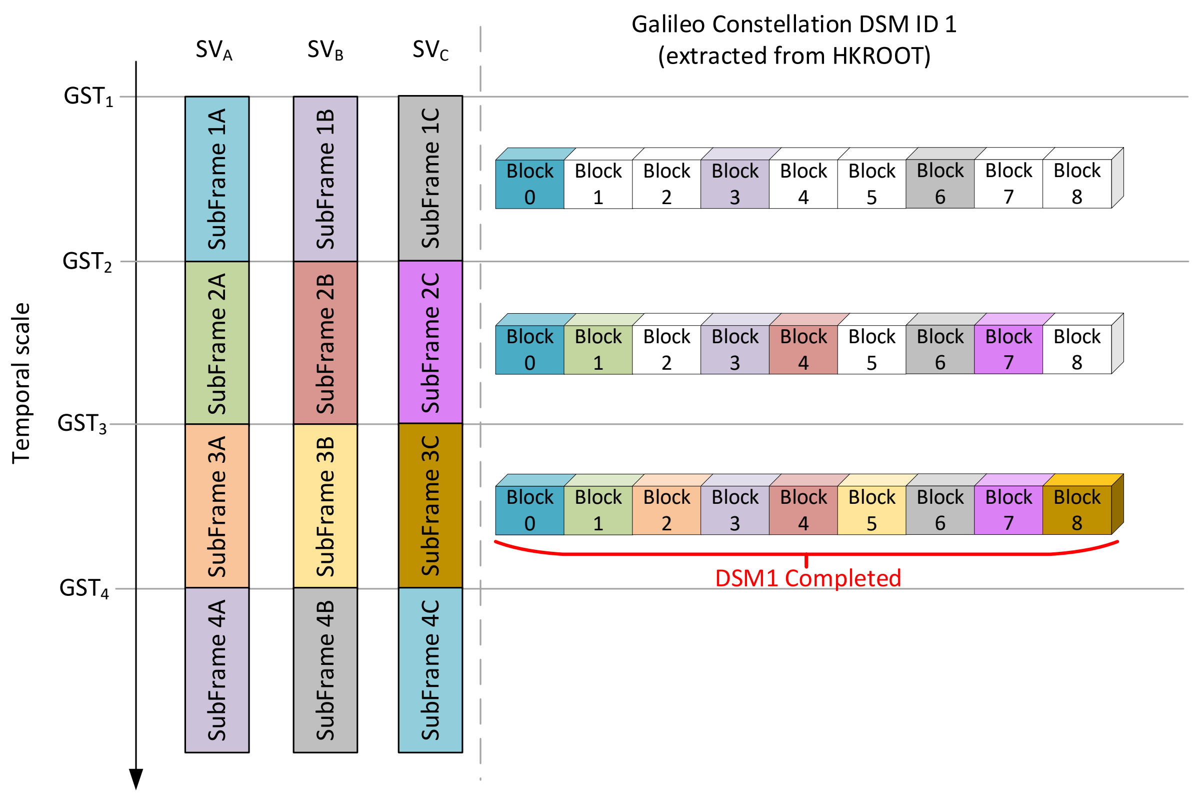


Figure 2-9 DSM Message Sequencing

Figure 2-9 considers:

1. The three Space Vehicles (A, B, C) send OSNMA data;

2. The DSM Message ID is in all the SubFrames in the three Space Vehicles the same (i.e., ID 1 in the figure)
3. The total number of blocks (extracted from HKROOT) is 9.
4. Each space vehicle has an offset of three DSM Blocks within each other. This cannot be the case in real data (e.g., SVA transmits Block 0 and SVB can transmit block 6), but it is shown this way for the sake of clarity.

Besides, also from the previous figure, it is clear that the DSM Message is not an attribute or a propriety of a single Space Vehicle but a property of the whole Galileo Constellation.

2.2.3.5.2. DSM-KROOT

The DSM type KROOT provides the information needed to:

- Perform the authentication seen in §2.2.3.4;
- Get the Tesla Root Key;
- Means to verify that the current Tesla Key belongs to the chain (with the root as Tesla Root key); and
- Means to verify the digital signature of the Tesla Root Key.

Tesla Key authentication with previous Tesla Key
§2.2.3.4 provided an overview of how the Authentication of the Navigation message works, and the key of this authentication is the Tesla Key.

Accordingly, we need some means to verify that this Tesla Key is actually the key that needs to be used and that this key has not been impersonated. In order to do that, let's consider the follow figure:

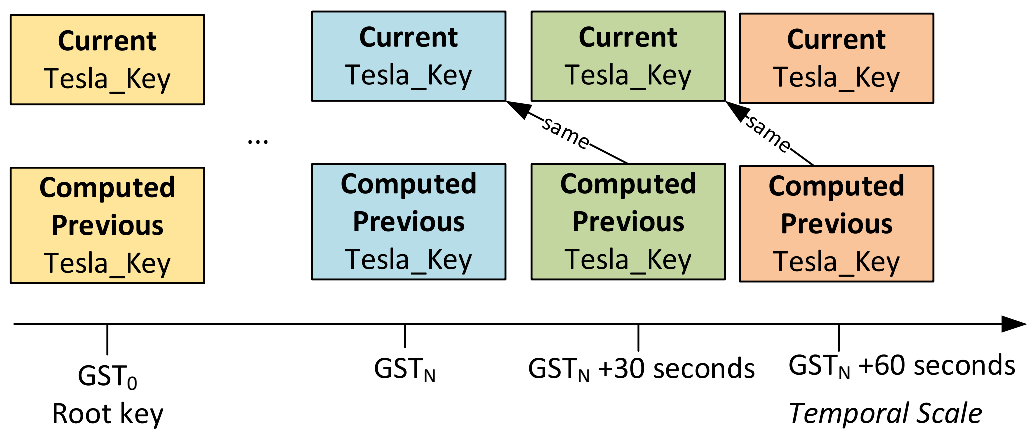


Figure 2-10 Verification of Tesla Key I

Let's consider that we are at a time when we have received a Sub-Frame with OSNMA information, and this time is $\boldsymbol{GST_N + 30\ seconds}$ -i.e., in the green Sub-Frame in the image above.

Among other things within the OSNMA information from the sub-frame (DSM, Tags…) we have the Tesla Key. With this Tesla Key and other information from the DSM-HKRoot, we can compute the previous Tesla Key, i.e., we can compute the Tesla Key at $\boldsymbol{GST_N}$

The computation of this key can be computed by hashing the Current Tesla Key with the previous GST time and a parameter, alpha, from the DSM-HKRoot. It can be computed with the following formula:

$$\boldsymbol{Computed\ Previous\ Tesla\ Key}|_{GST_N+30s} = trunc(l_k, hash(\boldsymbol{Current\_Tesla\_Key}||\boldsymbol{GST_N}||\boldsymbol{\alpha})$$

In which:

- l_k is the length of the key, extracted from the DSM-HKRoot;
- $Hash$ is the Hash function (e.g., SHA256), hash type extracted from DSM-HKRoot; and
- $Alpha$ is a random pattern, extracted DSM-HKroot.

This scheme creates a chain of keys starting in the Tesla Root Key. Since the function to generate the key is unidirectional (property given by the hash function), the chain can only be computed in one direction, that is, one key cab be used to obtain the next one in only one direction, but not the opposite one.

The system generates the chain of keys in the beginning and discloses the last one (Tesla Root Key). In that way, when a key is distributed by the SiS, a receiver is able to compute whether the key belongs to the key chain or not, but is not able to go the other way to be able to compute future keys.

To know if our current Tesla Key belongs to a chain, we just need it to compare with the real Tesla Key provided in the previous Sub-Frame. I.e., if the current Tesla Key belongs to a chain, that means:

$$\boldsymbol{Computed\ Previous\ Tesla\ Key}|_{GST_N+30s} = \boldsymbol{Current\ Tesla\ Key}|_{GST_N}$$

Similarly, if we receive the <u>immediately following</u> subframe, we can know if the key belongs to a chain with:

$$\boldsymbol{Computed\ Previous\ Tesla\ Key}|_{GST_N+60s} = \boldsymbol{Current\ Tesla\ Key}|_{GST_{N+30s}}$$

Tesla Key authentication vs Root Key

In the previous section we have shown how to know if the key belongs to chain by checking the key received in the previous Sub-Frame. However, there are still two considerations important to take into account:

- If a Sub-Frame in the previous corresponding timeslot has not been received, the comparison cannot be done; accordingly it is needed whether to store all the previous keys from the key chain, or;
- To re-compute all the key chain from the current Tesla chain to the Root Tesla Chain, as represented in the following figure:

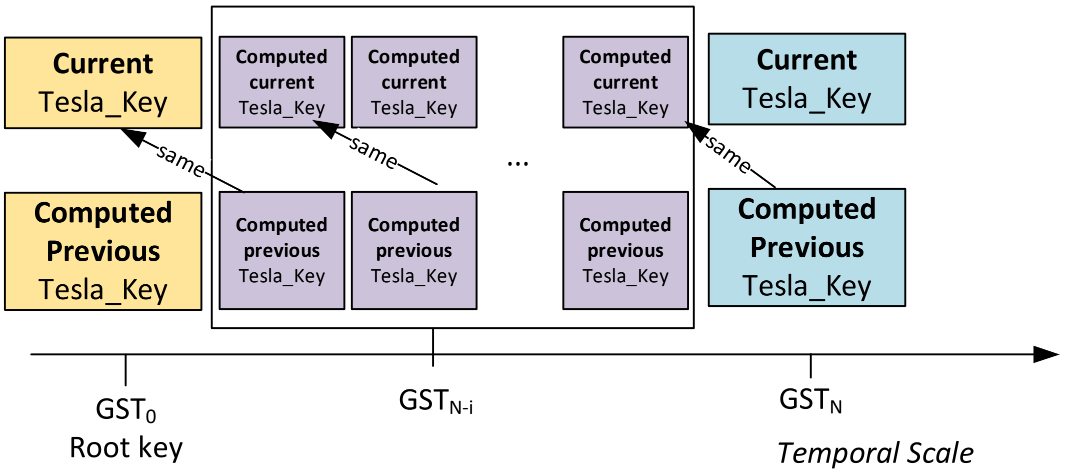

Current Tesla_Key
Computed Previous Tesla_Key
same
Computed current Tesla_Key
Computed previous Tesla_Key
...
Current Tesla_Key
Computed Previous Tesla_Key
GST0 Root key
GSTN-i
GSTN
Temporal Scale

Figure 2-11 Verification of Tesla Key II

The solution chosen in this implementation is to compute, for all received Tesla Keys, the re-computation to the initial Root Tesla Key. According to § 5.5.1 from [RD-1], there will be, at minimum, once per day and at maximum once per hour. Taking into account the most critical case, which is:

- HKROOT once per day, and it has been received at the beginning of the current day; and
- We are at the end of the day.

The maximum number of iterations of previous Tesla Keys is (according to Eq. 19 from [RD-1]), the number of seconds of

$$I = \frac{(GST_N)_{seconds} - (GST_{HKROOT})_{seconds}}{30} + 1$$

$$I_{max} = \frac{Seconds\ in\ a\ day}{30} + 1 = \frac{86400}{30} + 1 = 2881$$

For instance, a famous IoT device such as ESP8266 computes around 1200 hashes per second[4], meaning that in the most critical case a microcontroller will take around 3 seconds to verify if the Tesla Key belongs to the Root Tesla Key Chain. This value is considered acceptable, but for embedded applications a trade-off or a mixed solution can be envisaged.

Tesla Root Key authentication

In the two previous sections we have checked that the Tesla Keys we are receiving through the Signal in Space belong to a Key Chain with a Root Key which is contained in the DSM-KROOT.

However, this Root Key can also be impersonated, and hence, the full Tesla Chain can be compromised. In order to solve this issue, within the DSM-KROOT a Digital Signature is also

[4] According to:
https://hackaday.com/2018/01/03/mine-bitcoin-with-an-esp8266/
https://everythingesp.com/bitcoin-mining-with-esp8266/

provided. Moreover, the message to be verified can be also computed with fields from the DSM-KROOT and other OSNMA data, following §6.3 from [RD-1].

The *Galileo Service Center*[5] will provide a Public Key which can be used to verify the Authenticity of the Digital Signature. Currently a PEM or a XML file can be downloaded from the GSC:

```
-----BEGIN EC PARAMETERS-----

LLgqhkjOPQMBBw==

-----END EC PARAMETERS-----

-----BEGIN PUBLIC KEY-----

JFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE+Q2wvmv3dQg1sQF6OmCEy8skCSiu79
vBnRrKmaPpCJnaMOOvm26Us6ELhebL+q75MAyWAXJjlyRZZwp68gSAHw==

-----END PUBLIC KEY-----
```

Figure 2-12 Example of a PEM file that can be downloaded from the GSC. Characters have been altered.

It is worth to mention that in order to verify a message with a digital signature, only the Public Key part is needed. From the GSC website or from the XML file the Signature and hashing mechanism can be extracted. I.e., ECDSA P-256 or ECDSA P-521.

It also has to be noticed that Galileo's OSNMA also foresees a mechanism for the renewal of the root key which does not need any initial digital signature. This mechanism is the DSM-PKR and will be described in §2.2.3.5.3

2.2.3.5.3. DSM-PRK

The message is devoted to the dissemination of future public keys, since these keys are also updated from time to time. The processing of this message is outside the scope from the current issue of the document.

2.2.4. DATA DISPLAY

It is considered that the user will run the program and see the outputs in the terminal/command prompt. Accordingly, the use of a library that works on Linux/Windows command line will be used.

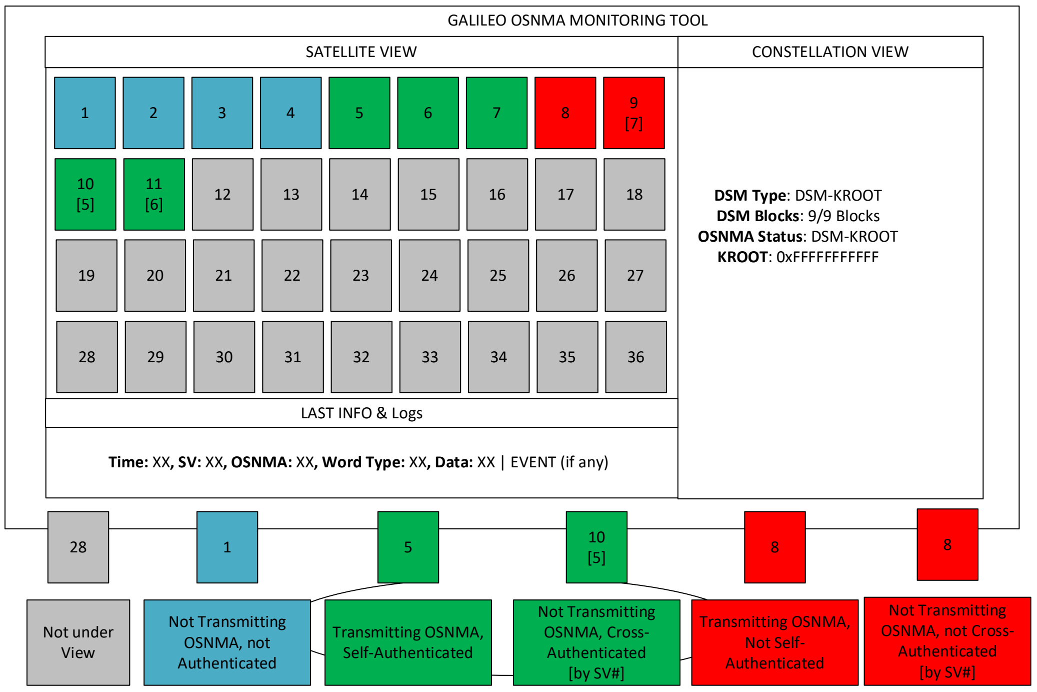
Besides, in this issue, it is considered that the user runs the script and there is nothing more for she/he to execute, just debugging the screen. Hence, no active interaction is expected.

Considering:

- Up to 36 Space Vehicles;
- Some properties are common to Galileo Constellation;
- It is needed to have some visual aid to identify the authentication properties;
- It is needed to visualise the injection data, some basic information & events if they happen (e.g., subframe complete, DSM complete, Data Authenticated…)

[5]Visit https://www.gsc-europa.eu/ and register to get the Public Key (for the Test Phase).

- The following front-end is proposed (even though the implementation may differ):


GALILEO OSNMA MONITORING TOOL
SATELLITE VIEW
CONSTELLATION VIEW
1
2
3
4
5
6
7
8
9 [7]
10 [5]
11 [6]
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
LAST INFO & Logs
Not under View
Not Transmitting OSNMA, not Authenticated
Transmitting OSNMA, Self-Authenticated
Not Transmitting OSNMA, Cross-Authenticated [by SV#]
Transmitting OSNMA, Not Self-Authenticated
Not Transmitting OSNMA, not Cross-Authenticated [by SV#]

DSM Type: DSM-KROOT
DSM Blocks: 9/9 Blocks
OSNMA Status: DSM-KROOT
KROOT: 0xFFFFFFFFFFFF

Time: XX, **SV:** XX, **OSNMA:** XX, **Word Type:** XX, **Data:** XX | EVENT (if any)

Figure 2-13 Front-End Command Line Display Concept

2.3. ARCHITECTURAL DESIGN

This section will be divided into the different “Branches” (in fact, modules) of the Functional Tree seen in §2.2, even though some of the branches will be divided into subsections if needed. It defines the classes used to acquire and process the necessary data to perform and show the Space Vehicles Satellite information. It is worth to mention that some of the classes are supported by functions, defined in the same modules.

The modules are divided following the functional tree:

- dataAcquisition.py;
- dataTransformation.py;
- data Processing, including:
  - dataProcessingGalileoFrame_Constellation.py
  - Data Processing for OSNMA, including:
    - dataProcessingOsnma_Authenticator.py
    - dataProcessingOsnma_DSM.py
    - dataProcessingOsnma_svKrootOsnmaMack.py
- dataVisualisationSupport.py, a module to support the main code executor with some details of data visualisation.

A general overview (excluding data Visualisation, as it has no OSNMA Business Logic) is shown in the following picture:

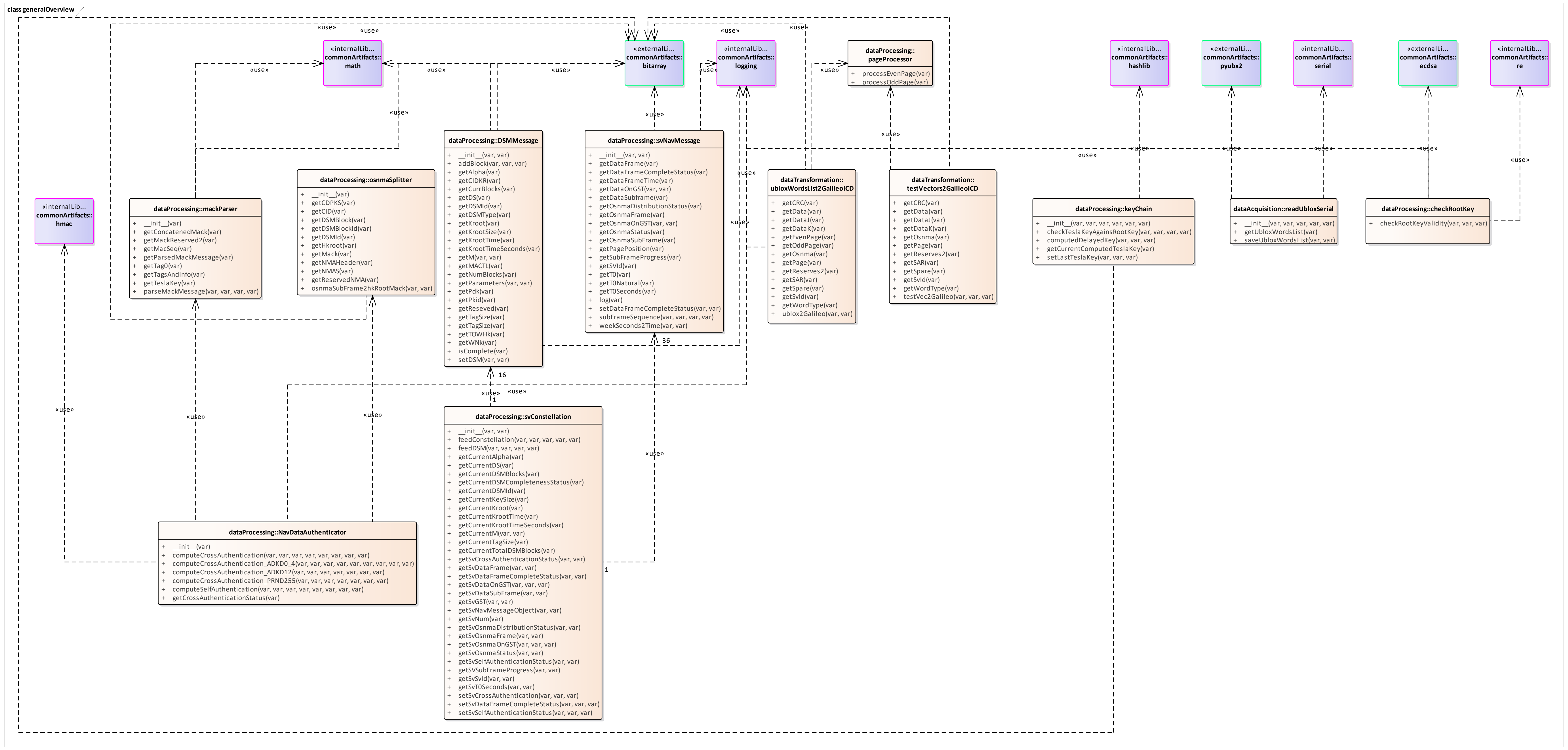

class generalOverview
«use»
«internalLib... commonArtifacts:: math
«externalLi... commonArtifacts:: bitarray
«internalLib... commonArtifacts:: logging
dataProcessing:: pageProcessor
+ processEvenPage(var)
+ processOddPage(var)
«internalLib... commonArtifacts:: hashlib
«externalLi... commonArtifacts:: pyubx2
«internalLib... commonArtifacts:: serial
«externalLi... commonArtifacts:: ecdsa
«internalLib... commonArtifacts:: re
«internalLib... commonArtifacts:: hmac
dataProcessing::mackParser
+ __init__(var)
+ getConcatenedMack(var)
+ getMackReserved2(var)
+ getMacSeq(var)
+ getParsedMackMessage(var)
+ getTag0(var)
+ getTagsAndInfo(var)
+ getTeslaKey(var)
+ parseMackMessage(var, var, var, var)
dataProcessing::osnmaSplitter
+ __init__(var)
+ getCDPKS(var)
+ getCID(var)
+ getDSMBlock(var)
+ getDSMBlockId(var)
+ getDSMId(var)
+ getHkroot(var)
+ getMack(var)
+ getNMAHeader(var)
+ getNMAS(var)
+ getReservedNMA(var)
+ osnmaSubFrame2hkRootMack(var, var)
dataProcessing::DSMMessage
+ __init__(var, var)
+ addBlock(var, var, var)
+ getAlpha(var)
+ getCIDKR(var)
+ getCurrBlocks(var)
+ getDS(var)
+ getDSMId(var)
+ getDSMType(var)
+ getKroot(var)
+ getKrootSize(var)
+ getKrootTime(var)
+ getKrootTimeSeconds(var)
+ getM(var, var)
+ getMACTL(var)
+ getNumBlocks(var)
+ getParameters(var, var)
+ getPdk(var)
+ getPkid(var)
+ getReseved(var)
+ getTagSize(var)
+ getTagSize(var)
+ getTOWHk(var)
+ getWNk(var)
+ isComplete(var)
+ setDSM(var, var)
16
dataProcessing::svNavMessage
+ __init__(var, var)
+ getDataFrame(var)
+ getDataFrameCompleteStatus(var)
+ getDataFrameTime(var)
+ getDataOnGST(var, var)
+ getDataSubframe(var)
+ getOsnmaDistributionStatus(var)
+ getOsnmaFrame(var)
+ getOsnmaOnGST(var, var)
+ getOsnmaStatus(var)
+ getOsnmaSubFrame(var)
+ getPagePosition(var)
+ getSubFrameProgress(var)
+ getSVId(var)
+ getT0(var)
+ getT0Natural(var)
+ getT0Seconds(var)
+ log(var)
+ setDataFrameCompleteStatus(var, var)
+ subFrameSequence(var, var, var, var)
+ weekSeconds2Time(var, var)
36
dataTransformation:: ubloxWordsList2GalileoICD
+ getCRC(var)
+ getData(var)
+ getDataJ(var)
+ getDataK(var)
+ getEvenPage(var)
+ getOddPage(var)
+ getOsnma(var)
+ getPage(var)
+ getReserves2(var)
+ getSAR(var)
+ getSpare(var)
+ getSvId(var)
+ getWordType(var)
+ ublox2Galileo(var, var)
dataTransformation:: testVectors2GalileoICD
+ getCRC(var)
+ getData(var)
+ getDataJ(var)
+ getDataK(var)
+ getOsnma(var)
+ getPage(var)
+ getReserves2(var)
+ getSAR(var)
+ getSpare(var)
+ getSvId(var)
+ getWordType(var)
+ testVec2Galileo(var, var, var)
dataProcessing::keyChain
+ __init__(var, var, var, var, var, var)
+ checkTeslaKeyAgainsRootKey(var, var, var, var)
+ computedDelayedKey(var, var, var)
+ getCurrentComputedTeslaKey(var)
+ setLastTeslaKey(var, var, var)
dataAcquisition::readUbloxSerial
+ __init__(var, var, var, var, var)
+ getUbloxWordsList(var)
+ saveUbloxWordsList(var, var)
dataProcessing::checkRootKey
+ checkRootKeyValidity(var, var, var)
1
dataProcessing::svConstellation
+ __init__(var, var)
+ feedConstellation(var, var, var, var, var)
+ feedDSM(var, var, var, var)
+ getCurrentAlpha(var)
+ getCurrentDS(var)
+ getCurrentDSMBlocks(var)
+ getCurrentDSMCompletenessStatus(var)
+ getCurrentDSMId(var)
+ getCurrentKeySize(var)
+ getCurrentKroot(var)
+ getCurrentKrootTime(var)
+ getCurrentKrootTimeSeconds(var)
+ getCurrentM(var, var)
+ getCurrentTagSize(var)
+ getCurrentTotalDSMBlocks(var)
+ getSvCrossAuthenticationStatus(var, var)
+ getSvDataFrame(var, var)
+ getSvDataFrameCompleteStatus(var, var)
+ getSvDataOnGST(var, var, var)
+ getSvDataSubFrame(var, var)
+ getSvGST(var, var)
+ getSvNavMessageObject(var, var)
+ getSvNum(var)
+ getSvOsnmaDistributionStatus(var, var)
+ getSvOsnmaFrame(var, var)
+ getSvOsnmaOnGST(var, var, var)
+ getSvOsnmaStatus(var, var)
+ getSvSelfAuthenticationStatus(var, var)
+ getSVSubFrameProgress(var, var)
+ getSvSvId(var, var)
+ getSvT0Seconds(var, var)
+ setSvCrossAuthentication(var, var, var)
+ setSvDataFrameCompleteStatus(var, var, var)
+ setSvSelfAuthenticationStatus(var, var, var)
1
dataProcessing::NavDataAuthenticator
+ __init__(var)
+ computeCrossAuthentication(var, var, var, var, var, var, var, var)
+ computeCrossAuthentication_ADKD0_4(var, var, var, var, var, var, var, var, var)
+ computeCrossAuthentication_ADKD12(var, var, var, var, var, var, var)
+ computeCrossAuthentication_PRND255(var, var, var, var, var, var, var)
+ computeSelfAuthentication(var, var, var, var, var, var, var, var)
+ getCrossAuthenticationStatus(var)

Figure 2-14 Architectural Diagram Layout, excluding data Visualisation

2.3.1. DATA ACQUISITION

As explained in the Functional Tree, the SW needs to be created in such fashion that some Test Data can be used as "Test Vectors" to test the end-to-end functionality. As an Ublox Receiver will be used, the Test Data will follow the structure of the raw Ublox Words (see Table 2-2 and Table 2-3). This Test Data will be saved/read in a CSV format, to allow cross-platforms functionalities.

Accordingly, the Data Acquisition Module will be composed by two classes and one interface:

- **Class "readUbloxData":** this class will be used to read a CSV File with the Test Data, along with the different methods needed, including a method to return a list of Ublox Words;
- **Class "readUbloxSerial":** this class will read the information from a Serial Port and will return a list of Ublox Words. It is also worth to mention that this class includes a flag to save the received data in a CSV format;
- **Interface "getUbloxWords":** this interface class is just used to ensure that both "readUbloxData" and "readUbloxSerial" classes utilise the same method to return the Ublox Words.

The following figure shows the classes relationships:

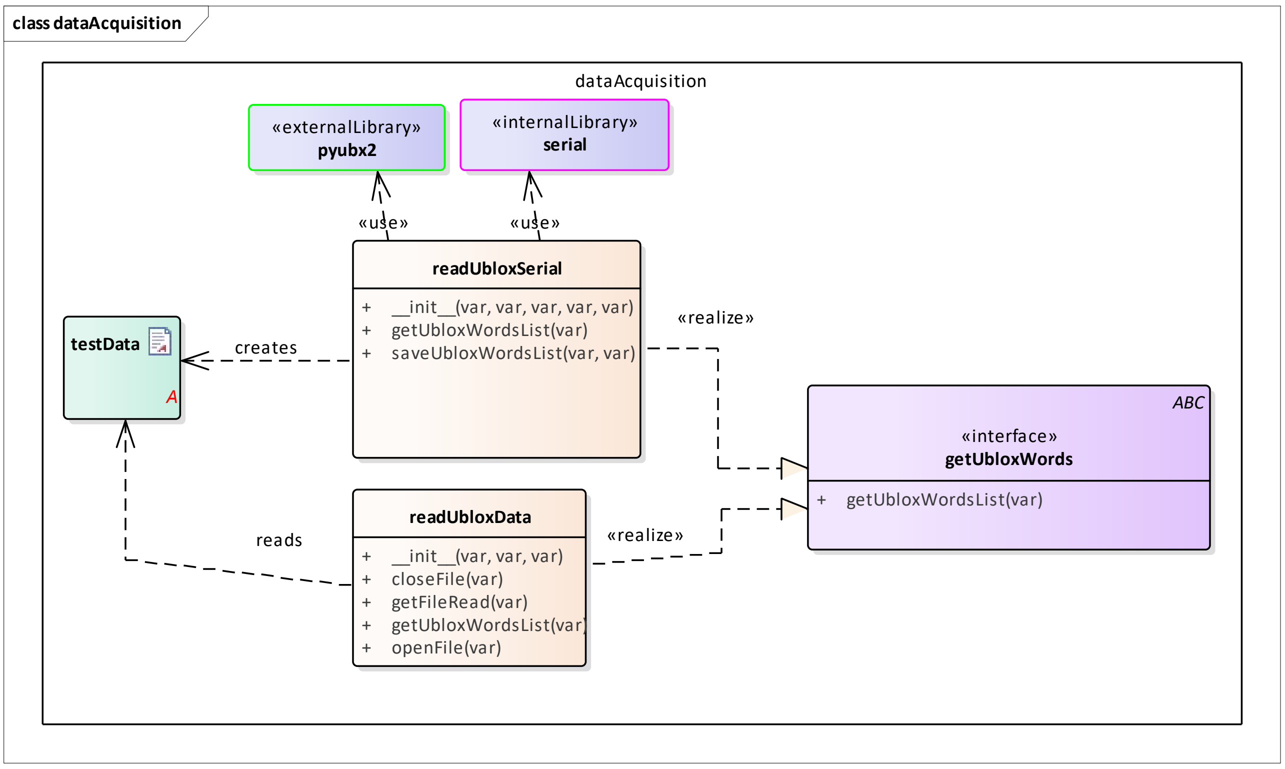


Figure 2-15 dataAcquisition classes and relationships

2.3.1.1. DEPENDANCIES

See Figure 2-15 for internal and external dependencies.

2.3.1.1. LOGGING

Data Acquisition module does not perform any logging.

2.3.2. DATA TRANSFORMATION

Within this section the classes and functions that transform the Ublox format into a format compatible with Galileo's ICD.

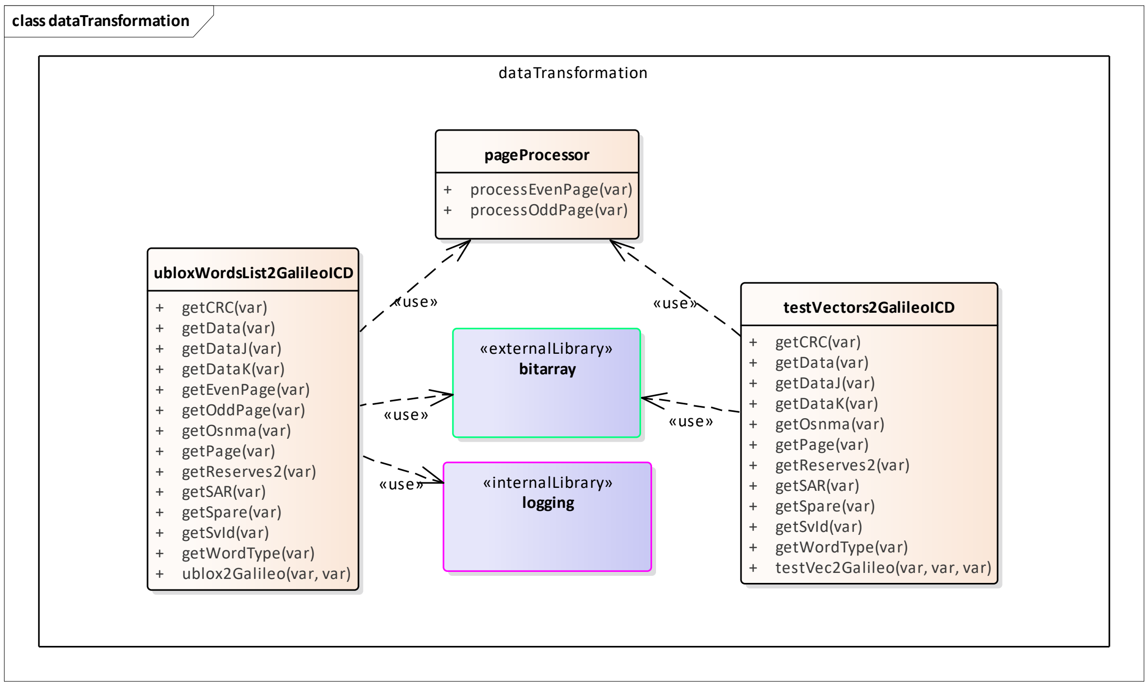


Figure 2-16 dataTransformation classes and relationships

This functional block, composed by the three classes seen in the previous figure, implement the change from ublox words to Galileo ICD:

- **pageProcessor:** Class with the business logic to split the even page and odd page to the different values expected by the Galileo ICD;
- **ubloxWordsList2GalileoICD:** This class transforms a list of ublox words into a Galileo information.
- **testVectors2GalileoICD:** This class is used to transform the input Test Vectors provided to a common input, i.e., Galileo ICD. This class is used only for Testing Purposes.

2.3.2.1. DEPENDANCIES

See Figure 2-16 for internal and external dependencies.

2.3.2.2. LOGGING

Class ubloxWordsList2GalileoICD performs the log Event type 1.

2.3.3. DATA PROCESSING

2.3.3.1. DATA PROCESSING: GALILEO FRAME (SUB-FRAME & FRAME SEQUENCING)

Talking in logical terms, it is known that all the satellites (Space Vehicles) are part of the Galileo System. Precisely, they form the Galileo Constellation. Each Space Vehicle individually contribute to the OSNMA sending the DSM Blocks, which is information that contribute to a the common OSNMA functionality.

Accordingly, there is information that it is only part of the Space Vehicle itself (such as satellite's Ephemerides) but there is also information which is common to all Constellation, mainly the DSM:

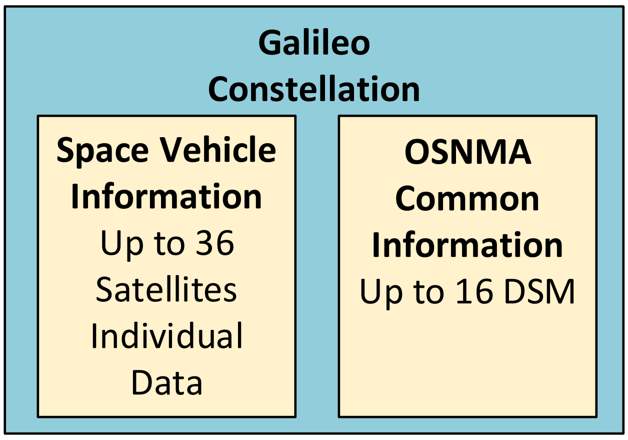

Galileo Constellation
Space Vehicle Information Up to 36 Satellites Individual Data
OSNMA Common Information Up to 16 DSM

Figure 2-17 Galileo Constellation Design Approach

Based on that, it has been designed a class which will encompass the Constellation, that will be associated to, at least, another two classes: one for individual Space Vehicle information and another for the Constellation Common Information.[6]

More precisely, a class called "svConstellation" will encapsulate all the business information. This master class is mainly composed by two lists of two elements each:

- **A first list, with the Space Vehicles Information**, composed by:
  - an **ID** of the Space Vehicle as first element,
  - a second element an object called "**svNavMessage**",
  - a third element with the **Self-Authentication Status**; and
  - a fourth element with the **Cross-Authentication Status**.

  The associated object "svNavMessage" will in fact encapsulate all the functionalities that are common to all Space Vehicles, such as store the Pages, store the subFrames and Frames and store the subFrame's associated OSNMA information. It also has all the business logic to perform all needed operation.
  The total number of Space Vehicles depend on the GNSS Constellation and can be set-up when instantiating the class. E.g., for Galileo this will be up to 36 Space Vehicles.

[6] Note: even though the satellites' almanac can be seen as "Common Information", as it is the same information per Space Vehicle, for the purpose of this library in Common Constellation Information only OSNMA information will be added.

- **A second list with the DSM information,** composed by:
    - an ID of the DSM as first element; and
    - a class that stores all the DSM information as second element. The maximum number of DSM will be 16, as per OSNMA ICD [RD-1], accordingly this is hard-coded as a class attribute.

The following figure shows the explained class relationships.

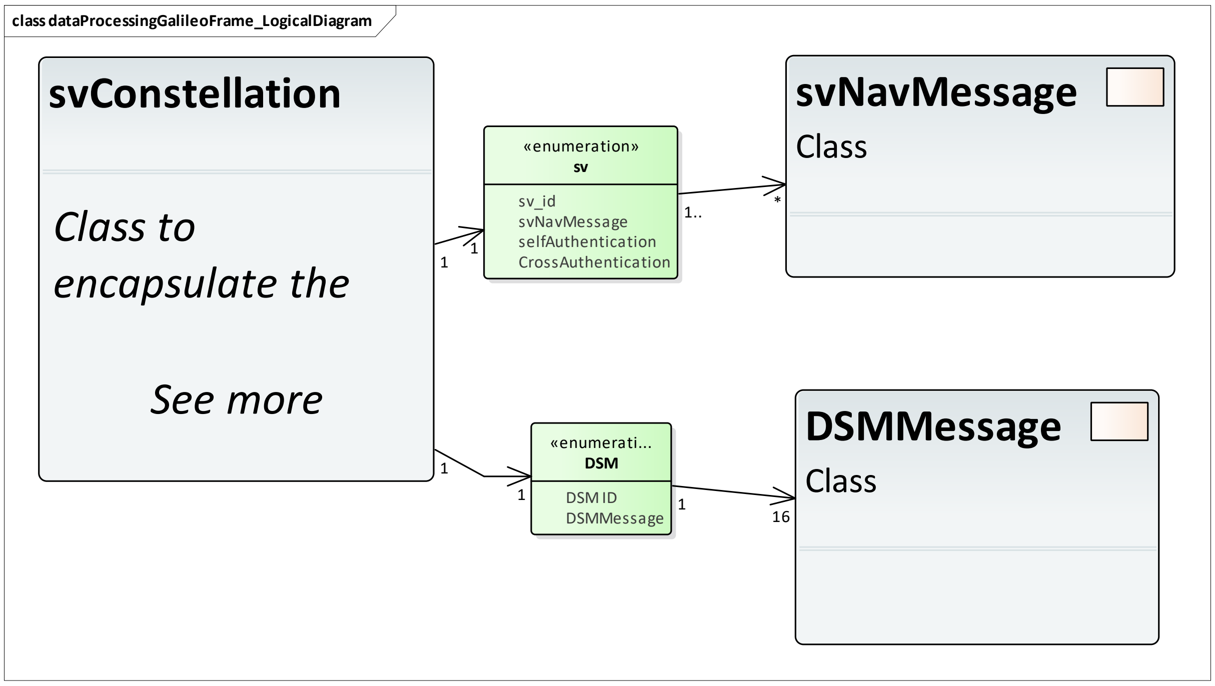


Figure 2-18 Galileo Constellation Logical Diagram

The following figure, instead of having the Logical Diagram, it has the actual class diagram.

Please notice that DSMMessage class is shown with no context nor information as it will be explained in its dedicated section.

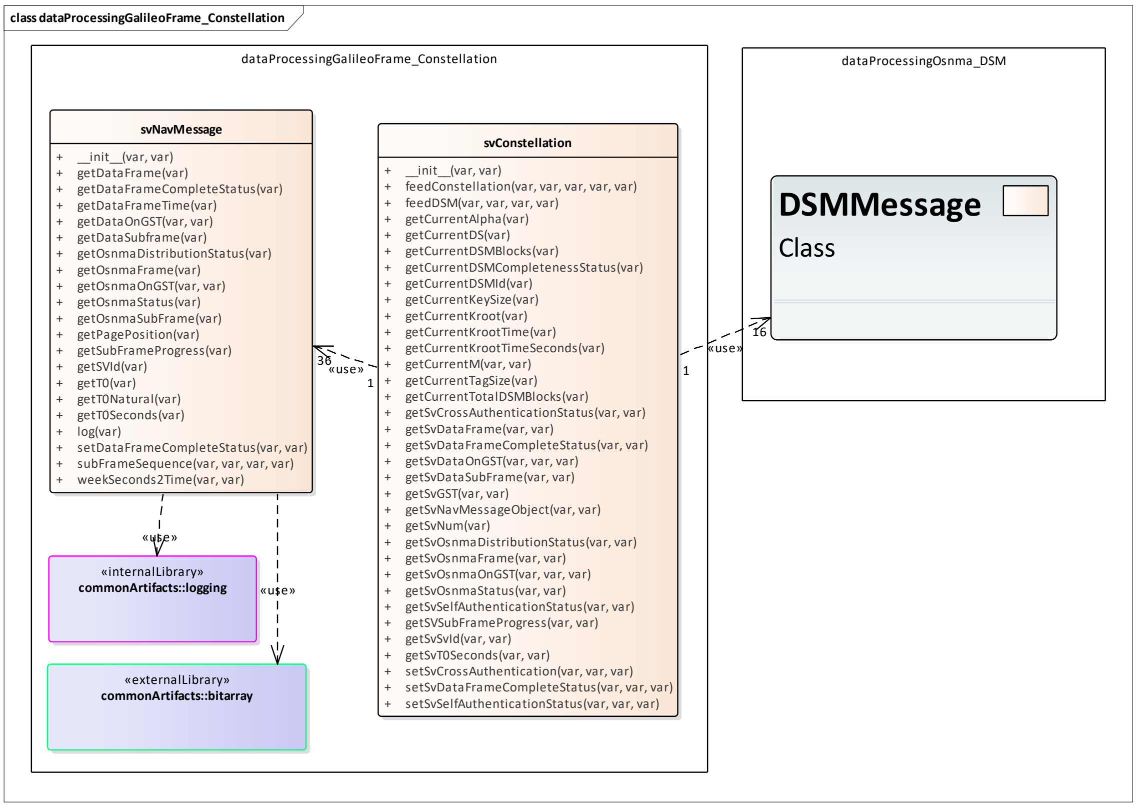

class dataProcessingGalileoFrame_Constellation
dataProcessingGalileoFrame_Constellation
svNavMessage
+ __init__(var, var)
+ getDataFrame(var)
+ getDataFrameCompleteStatus(var)
+ getDataFrameTime(var)
+ getDataOnGST(var, var)
+ getDataSubframe(var)
+ getOsnmaDistributionStatus(var)
+ getOsnmaFrame(var)
+ getOsnmaOnGST(var, var)
+ getOsnmaStatus(var)
+ getOsnmaSubFrame(var)
+ getPagePosition(var)
+ getSubFrameProgress(var)
+ getSVId(var)
+ getT0(var)
+ getT0Natural(var)
+ getT0Seconds(var)
+ log(var)
+ setDataFrameCompleteStatus(var, var)
+ subFrameSequence(var, var, var, var)
+ weekSeconds2Time(var, var)
svConstellation
+ __init__(var, var)
+ feedConstellation(var, var, var, var, var)
+ feedDSM(var, var, var, var)
+ getCurrentAlpha(var)
+ getCurrentDS(var)
+ getCurrentDSMBlocks(var)
+ getCurrentDSMCompletenessStatus(var)
+ getCurrentDSMId(var)
+ getCurrentKeySize(var)
+ getCurrentKroot(var)
+ getCurrentKrootTime(var)
+ getCurrentKrootTimeSeconds(var)
+ getCurrentM(var, var)
+ getCurrentTagSize(var)
+ getCurrentTotalDSMBlocks(var)
+ getSvCrossAuthenticationStatus(var, var)
+ getSvDataFrame(var, var)
+ getSvDataFrameCompleteStatus(var, var)
+ getSvDataOnGST(var, var, var)
+ getSvDataSubFrame(var, var)
+ getSvGST(var, var)
+ getSvNavMessageObject(var, var)
+ getSvNum(var)
+ getSvOsnmaDistributionStatus(var, var)
+ getSvOsnmaFrame(var, var)
+ getSvOsnmaOnGST(var, var, var)
+ getSvOsnmaStatus(var, var)
+ getSvSelfAuthenticationStatus(var, var)
+ getSVSubFrameProgress(var, var)
+ getSvSvId(var, var)
+ getSvT0Seconds(var, var)
+ setSvCrossAuthentication(var, var, var)
+ setSvDataFrameCompleteStatus(var, var, var, var)
+ setSvSelfAuthenticationStatus(var, var, var)
36
«use»
1
«use»
«internalLibrary»
commonArtifacts::logging
«use»
«externalLibrary»
commonArtifacts::bitarray
1
«use»
16
dataProcessingOsnma_DSM
DSMMessage
Class

Figure 2-19 Data Processing Galileo Frame Constellation structural diagram

- **Class svNavMessage:** class that encapsulates the Galileo Sub-Frame and Frame sequencing as well as SubFrame and Frame storing.
- **Class svConstellation:** High level class that stores the information about the Space Vehicles SubFrames (via associated class), DSM Messages (via associated class) and Space Vehicles authentication status (including cross authentication)

2.3.3.1.1. DEPENDANCIES

See Figure 2-19 for internal and external dependencies.

2.3.3.1.2. LOGGING

Class **svNavMessage** performs the logs Event type 2 and Event type 3.

2.3.3.2. DATA PROCESSING: OSNMA

2.3.3.2.1. OSNMA: MESSAGE SEQUENCING

As explained in §2.2.3.3, each 40 bits from each Galileo Page contain the HKROOT Message and the MACK. Accordingly, we need a class to split the information, and to concatenate each part of the data from all pages within a subframe. This way, we will have all the HKROOT part and the MACK part separated. The module in charge of doing this, as well as getting the tags information and tesla key is the **dataProcessingOsnma_svKrootOsnmaMack.**

This module is composed by:

- A class, originally named as **osnmaSplitter** will split the MACK and the HKROOT for a given OSNMA SubFrame. Besides that, it will provide the information needed to take the DSM Block information from it.
- A second class named **mackParser.** This class is used to get the tags and the tesla key for a given mack subframe (which is splitted from the OSNMA SubFrame by the previous class).

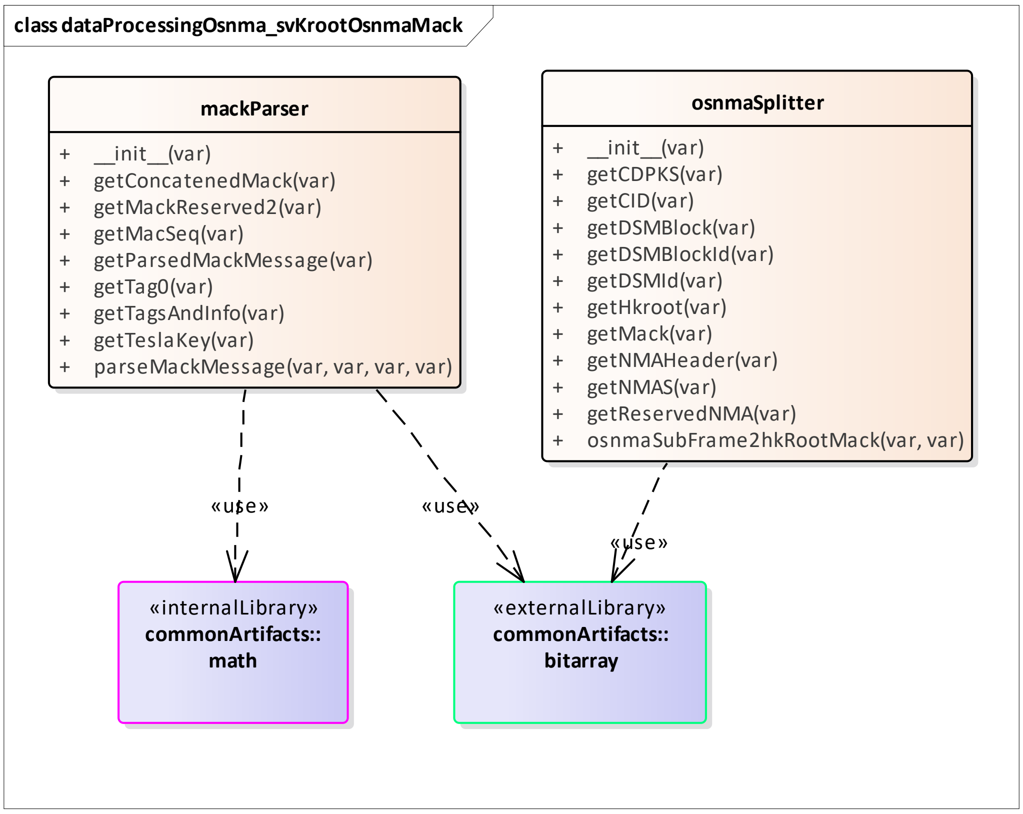


Figure 2-20 Osnma Splitter and Mack Parser for OSNMA Sequencing

2.3.3.2.1.1. *Dependencies*

See Figure 2-20 for internal and external dependencies.

2.3.3.2.1.2. *Logging*

Classes within the module do not implement any logging.

2.3.3.2.1. OSNMA: DSM

This module encapsulates the class that stores and process the Digital Signature Message, as well as all the methods needed for extracting all its related information.

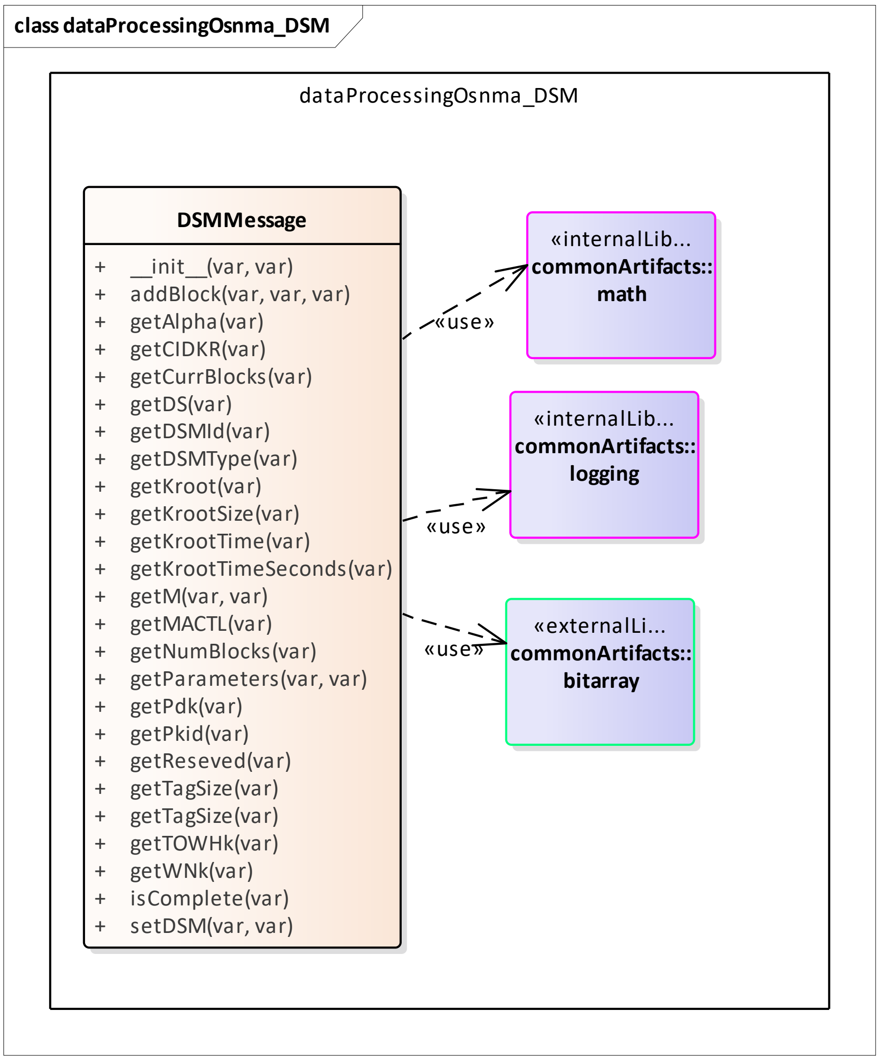


Figure 2-21 Digital Signature Message Class

2.3.3.2.1.1. Dependencies
See Figure 2-21 for internal and external dependencies.

2.3.3.2.1.2. Logging
Class **DSMMessages** logs the events: Event type 4, Event type 5, Event type 6, and Event type 7.

2.3.3.2.2. OSNMA: AUTHENTICATOR

Within the module *dataProcessingOsnma_Authenticator* , the core of the business logic for Navigation Message Authentication is Done.

It is composed by the following classes, which are supported by external functions within the same module:

- Class **NavDataAuthenticator:** This class implements the functionality purely related to the Authentication of the Space Vehicles. As a main method, it has:
  - **getSelfAuthentication**: This method is in charge of computing the Self-Authentication. As an output, it returns:
    - 0: the SV is Self-Authenticated
    - 1: the SV Self-Authentication has failed.
    - 2: the SV cannot be self-authenticated as it is missing the delayed Navigation Data.
  - **computeCrossAuthentication**: this method is in charge of computing the Cross-Authentication. It calls different methods depending on the Cross-Authentication ADKD.
  - **getCrossAuthentication**: it returns a list with the cross-authentication status associated with it's corresponding PRND. It returns a list of items with two elements: the PRND and the Cross-Authentication Status (which follows the same output types as per Self-Authentication)
- Class **checkRootKey:** This class implements the functionality used to check if the Digital Signature provided in the DSM is actually signed by the Galileo Service Centre.
- Class **keyChain:** this class implements the functionality used to check if the Tesla Key received in a Osnma Subframe belongs to the current Tesla Root Key.

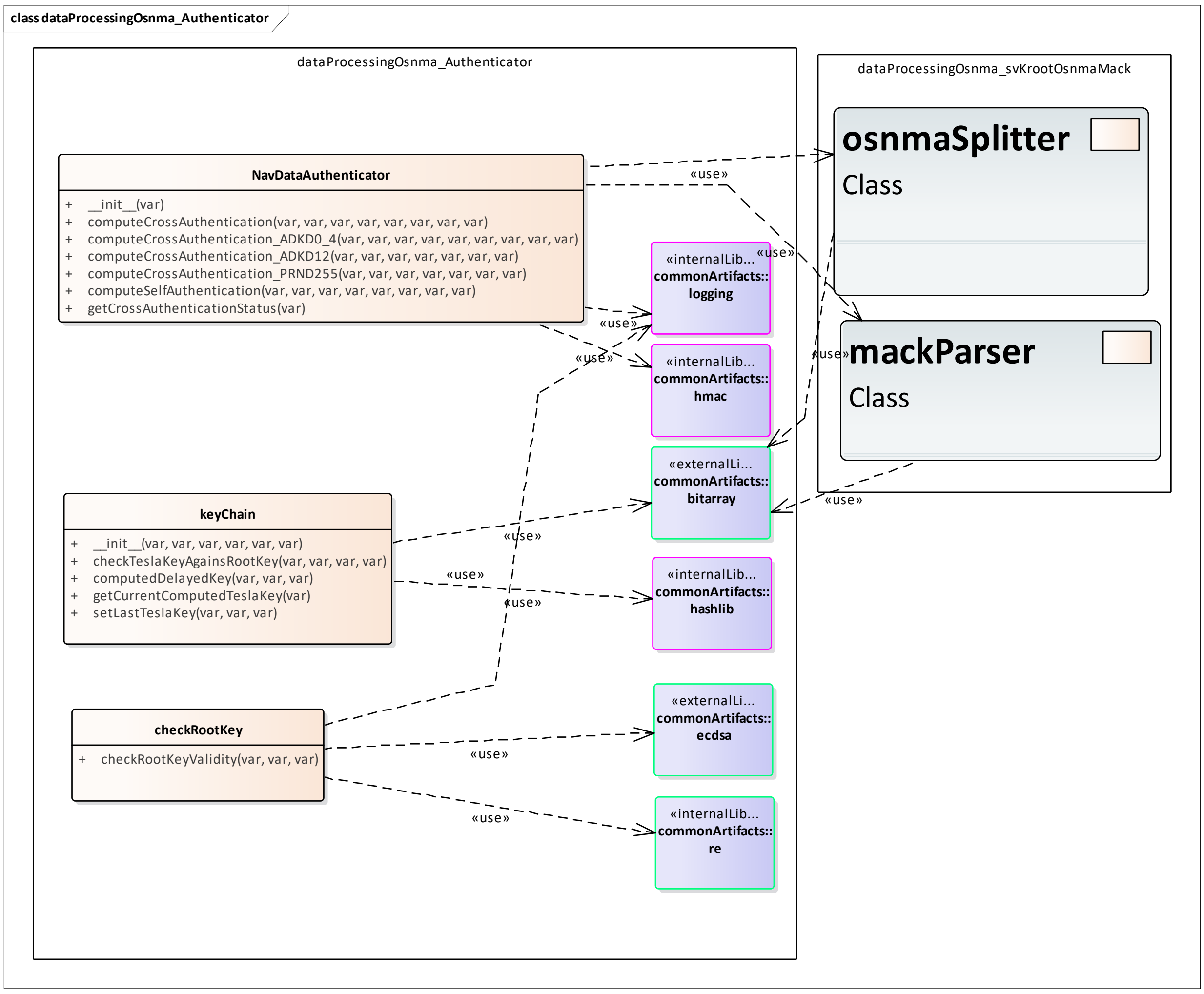

class dataProcessingOsnma_Authenticator
dataProcessingOsnma_Authenticator
dataProcessingOsnma_svKrootOsnmaMack
NavDataAuthenticator
+ __init__(var)
+ computeCrossAuthentication(var, var, var, var, var, var, var, var)
+ computeCrossAuthentication_ADKD0_4(var, var, var, var, var, var, var, var, var)
+ computeCrossAuthentication_ADKD12(var, var, var, var, var, var, var)
+ computeCrossAuthentication_PRND255(var, var, var, var, var, var, var)
+ computeSelfAuthentication(var, var, var, var, var, var, var, var)
+ getCrossAuthenticationStatus(var)
keyChain
+ __init__(var, var, var, var, var, var)
+ checkTeslaKeyAgainsRootKey(var, var, var, var)
+ computedDelayedKey(var, var, var)
+ getCurrentComputedTeslaKey(var)
+ setLastTeslaKey(var, var, var)
checkRootKey
+ checkRootKeyValidity(var, var, var)
osnmaSplitter
Class
mackParser
Class
«internalLib... commonArtifacts:: logging
«internalLib... commonArtifacts:: hmac
«externalLi... commonArtifacts:: bitarray
«internalLib... commonArtifacts:: hashlib
«externalLi... commonArtifacts:: ecdsa
«internalLib... commonArtifacts:: re
«use»

Figure 2-22 Mack authenticator, including Tesla Key Validation and Root Key Authentication

2.3.3.2.2.1. Dependencies

See Figure 2-22 for internal and external interfaces

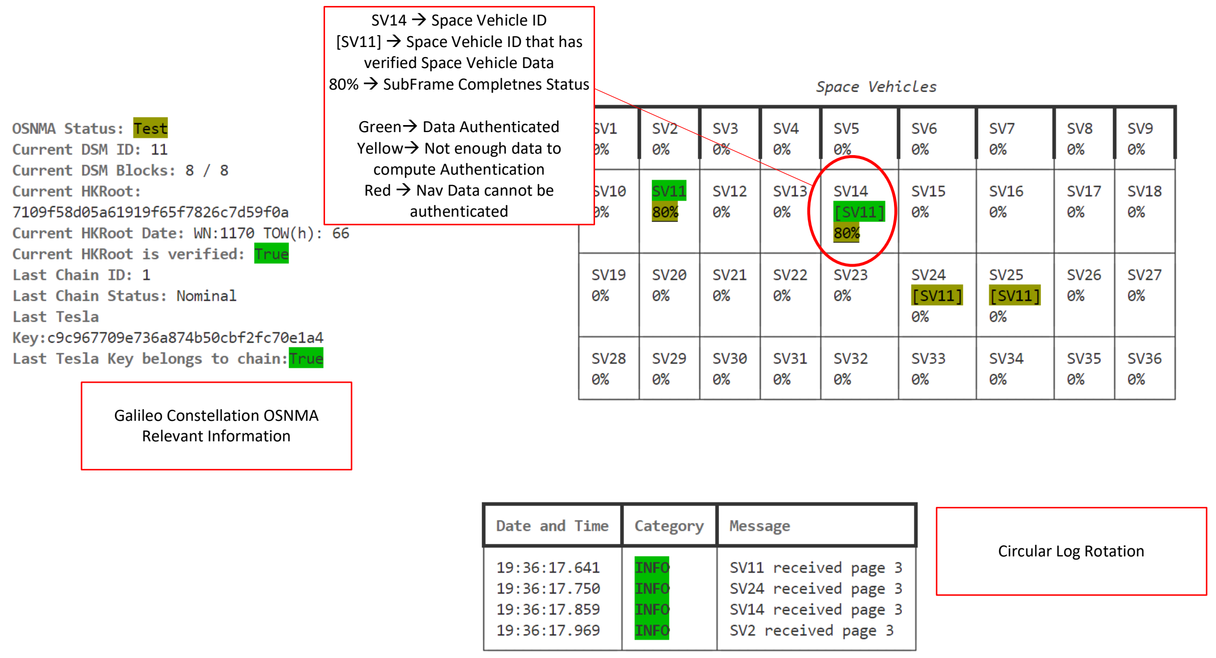
2.3.3.2.2.2. Logging

Within the *dataProcessingOsnma_Authenticator* module:

- Class **checkRootKey** generates the events Event type 8 and Event type 9;
- Class **keyChain** generates the events Event type 10, Event type 11 Event type 12;
- Class **NavDataAuthenticator** generates the events Event type 13, Event type 14, Event type 15, Event type 16, Event type 17, Event type 18, Event type 19, Event type 20, Event type 21, Event type 22, Event type 23, Event type 24, Event type 25 and Event type 26.

2.3.4. DATA DISPLAY

Data visualisation will be based on python's rich library. This library provides seamless means to create layouts, tables as well as colorised parameters. On a Laptop, the final implementation of the code with Rich Library looks as follows:


SV14 → Space Vehicle ID
[SV11] → Space Vehicle ID that has verified Space Vehicle Data
80% → SubFrame Completnes Status
Green→ Data Authenticated
Yellow→ Not enough data to compute Authentication
Red → Nav Data cannot be authenticated
Galileo Constellation OSNMA Relevant Information
Circular Log Rotation

```
OSNMA Status: Test
Current DSM ID: 11
Current DSM Blocks: 8 / 8
Current HKRoot:
7109f58d05a61919f65f7826c7d59f0a
Current HKRoot Date: WN:1170 TOW(h): 66
Current HKRoot is verified: True
Last Chain ID: 1
Last Chain Status: Nominal
Last Tesla
Key:c9c967709e736a874b50cbf2fc70e1a4
Last Tesla Key belongs to chain:True
```

Space Vehicles

SV1 0%	SV2 0%	SV3 0%	SV4 0%	SV5 0%	SV6 0%	SV7 0%	SV8 0%	SV9 0%
SV10 0%	SV11 80%	SV12 0%	SV13 0%	SV14 [SV11] 80%	SV15 0%	SV16 0%	SV17 0%	SV18 0%
SV19 0%	SV20 0%	SV21 0%	SV22 0%	SV23 0%	SV24 [SV11] 0%	SV25 [SV11] 0%	SV26 0%	SV27 0%
SV28 0%	SV29 0%	SV30 0%	SV31 0%	SV32 0%	SV33 0%	SV34 0%	SV35 0%	SV36 0%

```
Date and Time   Category   Message
19:36:17.641    INFO       SV11 received page 3
19:36:17.750    INFO       SV24 received page 3
19:36:17.859    INFO       SV14 received page 3
19:36:17.969    INFO       SV2 received page 3
```

Figure 2-23 Front-End Command Line Final Implementation – Laptop

In the following section, the three different layouts will be qualitative explained. See code in §5 for implementation details.

2.3.4.1. CONSTELLATION OSNMA RELEVANT INFORMATION

This layout, on the left of the screen, provides information related to all satellites (i.e., to the constellation.

It is divided into the "Current" information, which is the information of the DSM (which can take hours), and the "Last" Information, which is related to the last received Sub-Frame. Ideally, DSM Related information will change at minimum once per hour. However, information related to Sub-Frame ideally will change every 30 seconds -every time a new sub-frame is processed.

Some values are coloured in order to get better feedback: namely, Green if OK, red KO and dark yellow as partially OK or not computed.

2.3.4.2. CIRCULAR LOG ROTATION

On the bottom part, the Circular Log Rotation can be found. It is worth to mention that this logs are extracted from the ".log" file and then imported as a CSV file: they are not directly put in the Front-End. It has different colouring in the column "Category": Green for INFO and red for WARN.

Moreover, the logs can be also checked in its dedicated file if any post-process or troubleshooting needs to be done.

2.3.4.3. SPACE VEHICLES

In this layout the authentication and sub-Frame completeness status is shown. It is composed by 36 cells, one for each Galileo Satellite, divided along 4 rows. In each cell:

- On the **top** part, it is shown the **Galileo Satellite and its self-Authentication status**. If it has no high screen, it is not self-authenticated and it has not been tried to be self-authenticated. If it highlighted green, it is self-authenticated. If it is red, the self-authentication failed. If it is highlighted yellow, it tried to self-authenticate itself, but it was missing some data.
- On the **middle** part, it can be shown (or not) the **cross-authentication status**. The satellites which is trying to authenticate the data from the satellite is shown between brackets '[SVX]'. Colour coding is as per the previous bullet, but for Cross-Authentication.
- On the **lowest** part, the **Sub-Frame completeness progress** is shown. It is easily recognisable because it has the '%' at the end. It is highlighted in yellow when the completeness status is between 0 and 50%, highlighted in yellow and underscored when it's above 50% and green when it reaches the 100%

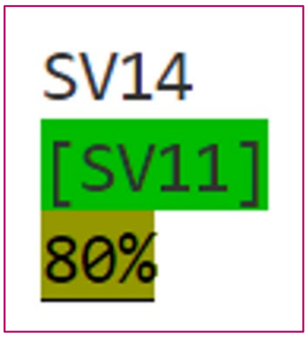
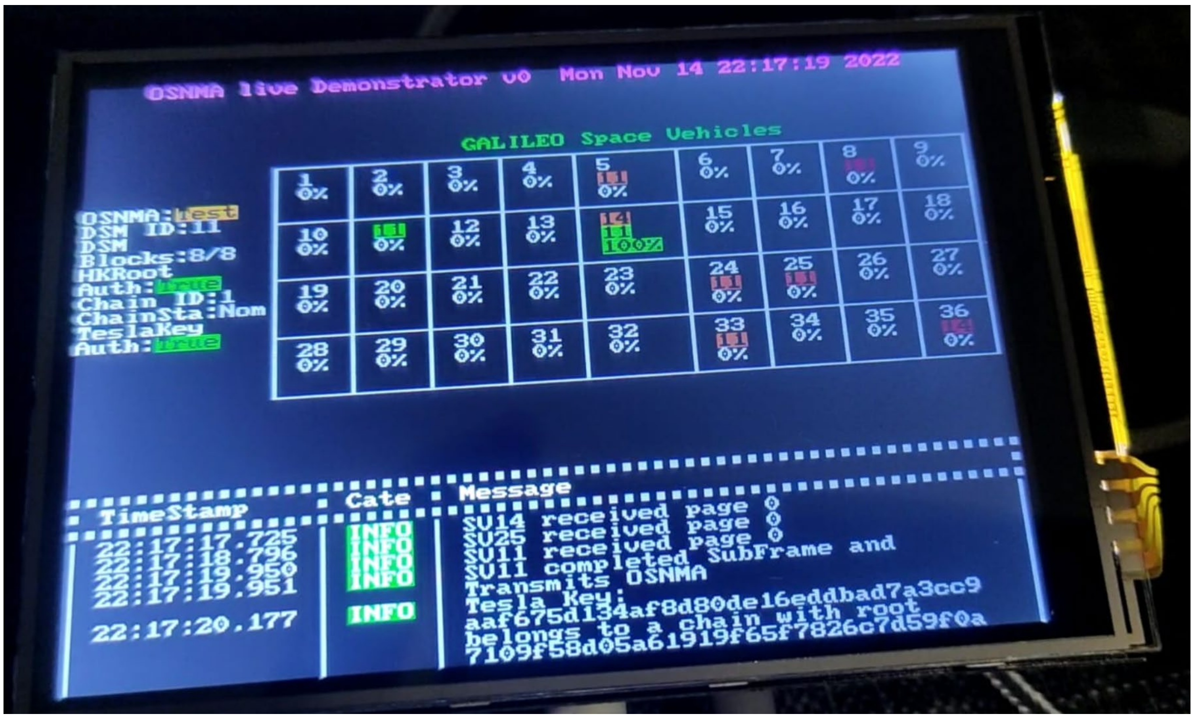


Figure 2-24 Example of Space Vehicle Cell. SV14 has been authenticated by SV11

2.3.4.4. PARTICULARITIES FOR SMALL SCREENS

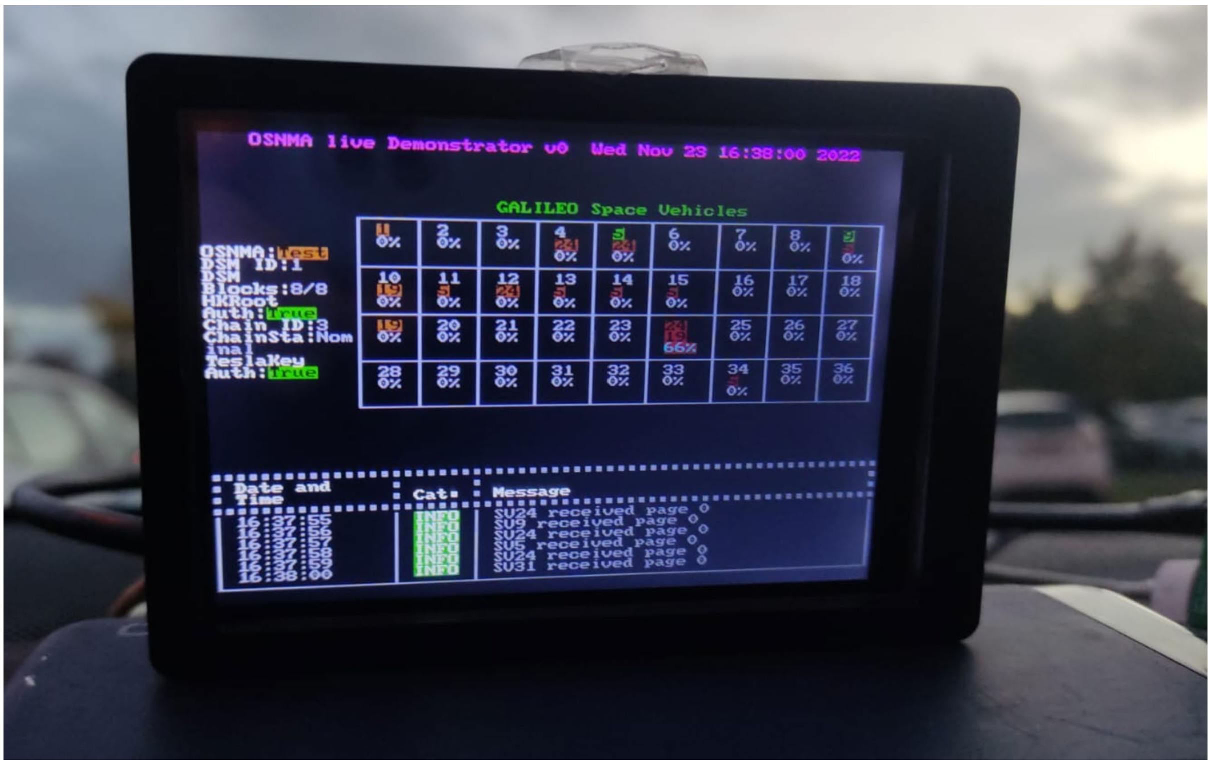
For this project to be portable, it is also implemented in a Raspberry pi microcomputer with a 3.5inch Screen. For this, a limited part of the Front-End Command Line has been implemented, with the following limitations:

- HKRoot is not shown;
- HKRoot sate and time is not shown; and
- Tesla Key is not shown;
- Name "SV" is not shown
- Cross-authentication satellites miss the "[]"



```
OSNMA live Demonstrator v0   Mon Nov 14 22:17:19 2022

GALILEO Space Vehicles

OSNMA:Test
DSM ID:11
DSM
Blocks:8/8
HKRoot
Auth:True
Chain ID:1
ChainSta:Nom
TeslaKey
Auth:True

TimeStamp      Cate   Message
22:17:17.725   INFO   SV14 received page 0
22:17:18.796   INFO   SV25 received page 0
22:17:19.950   INFO   SV11 received page 0
22:17:19.951   INFO   SV11 completed SubFrame and
                      Transmits OSNMA
22:17:20.177   INFO   Tesla Key:
                      aaf675d134af8d80de16eddbad7a3cc9
                      belongs to a chain with root
                      7109f58d05a61919f65f7826c7d59f0a
```

Figure 2-25 Front-End Command Line Final Implementation – Raspberry Pi with 3.5inch screen, I



```
OSNMA live Demonstrator v0   Wed Nov 23 16:38:00 2022

GALILEO Space Vehicles

OSNMA:Test
DSM ID:1
DSM
Blocks:8/8
HKRoot
Auth:True
Chain ID:3
ChainSta:Nom
TeslaKey
Auth:True

Date and Time   Cat   Message
16:37:55        INFO  SV24 received page 0
16:37:56        INFO  SV9 received page 0
16:37:57        INFO  SV24 received page 0
16:37:58        INFO  SV5 received page 0
16:37:59        INFO  SV24 received page 0
16:38:00        INFO  SV31 received page 0
```

Figure 2-26 Front-End Command Line Final Implementation – Raspberry Pi with 3.5inch screen, II

2.4. BEHAVIOURAL MODEL

2.4.1. SPACE VEHICLE OSNMA STATE MACHINE

As seen in Figure 2-13, it is intended that the HMI shows some sort of feedback of the current OSNMA status. This in terms of design translates into the following State Machine:

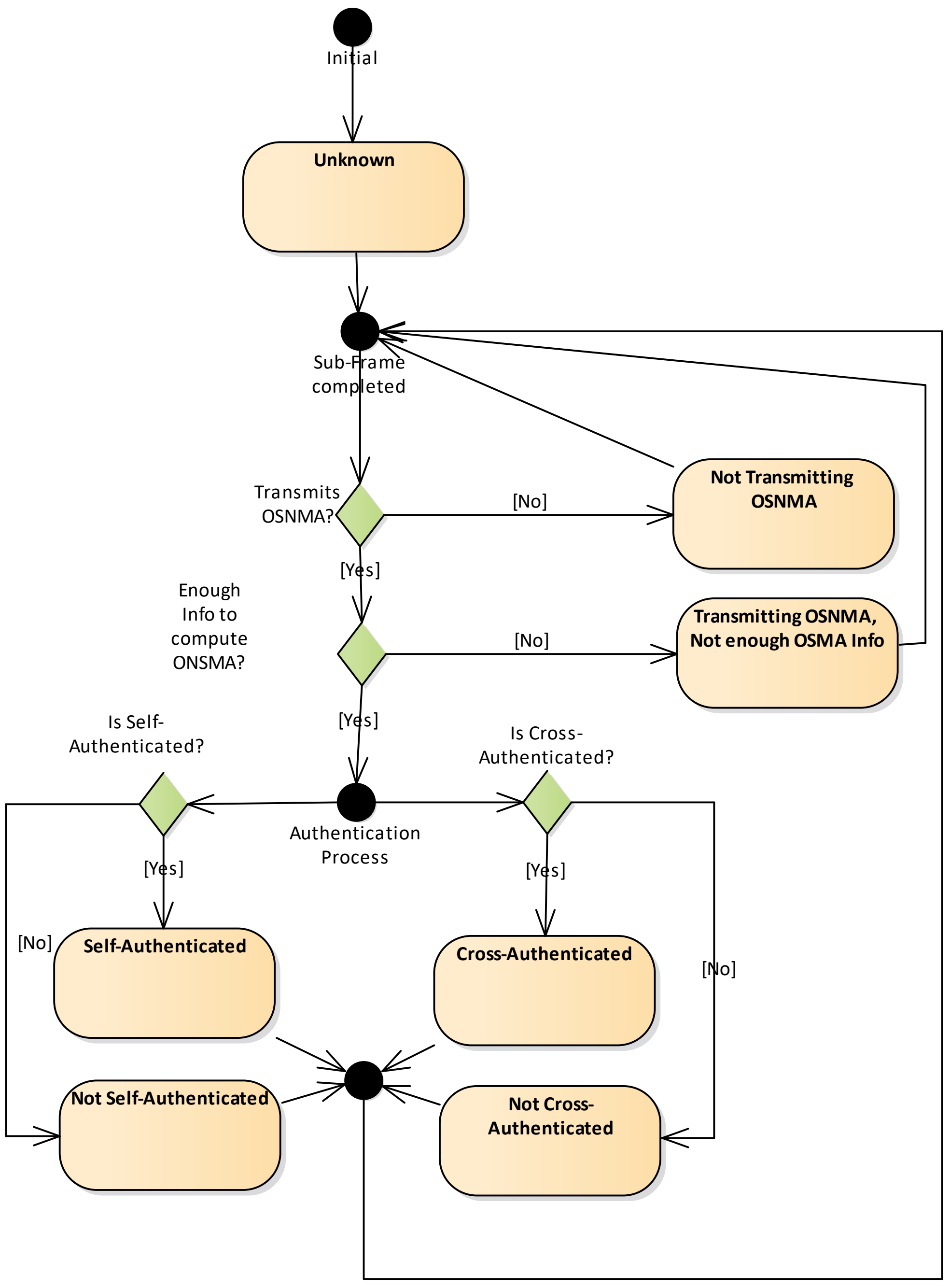


Figure 2-27 SV OSNMA State Machine

Unknown State

This State is reached by default for all Space Vehicles in the Start up.

<u>Not Transmitting OSNMA State</u>

This state is reached if, when completing a SubFrame, there is no information of OSNMA within the said Sub-Frame (i.e., the 40 ONSMA bits are filled by zeros for all pages within the Sub-Frame)

This status will not be shown in the HMI and will not be saved in the SV instance Status. However, specific logs will be saved informing of this.

<u>Transmitting OSNMA, not enough info State</u>

This State is reached if a SV received a Sub-Frame with OSNMA info, but there is not enough information to compute the OSNMA parameters (Verification of Tesla Root chain, etc).

The following states consider that there is enough info to compute the OSNMA information and there is OSMA information:

<u>Self-Authenticated</u>

A SV has successfully Authenticated their Navigation Data

<u>Not Self-Authenticated</u>

The SV has computed and compared the Hashes but they do not coincide

<u>Cross-Authenticated</u>

The SV has been authenticated by another Space Vehicle

<u>Not Cross-Authenticated</u>

The Hashes, one of them provided by another SV, have been compared but they do not coincide

The previous state machine design can be easily challenged, as it has a major drawback: the Space Vehicle will only change the OSNMA Status based on updates on Sub-Frames updates. Potentially, we could consider a Space Vehicle as Self-Authenticated, and then do not received enough Galileo Pages to re-compute the OSNMA, but still consider the Space Vehicle as Self-Authenticated ad eternal.

As explained in §2.2.3.4, the OSNMA part helps to authenticate the Navigation Data that has been received at, the earliest, two sub-frames ago (i.e., 60 seconds). Hence, OSNMA does not provide real-time Authentication.

Accordingly, as further studies on how OSNMA can really translate in real-time applications would need to be performed, it is decided to keep the State Machine as explained. In future versions of this document, a transition back to Unknown State could be done, taking into account, for instance, a time trigger.

2.4.2. ACTIVITY DIAGRAM

The following figure depicts the activity diagram for the whole system:

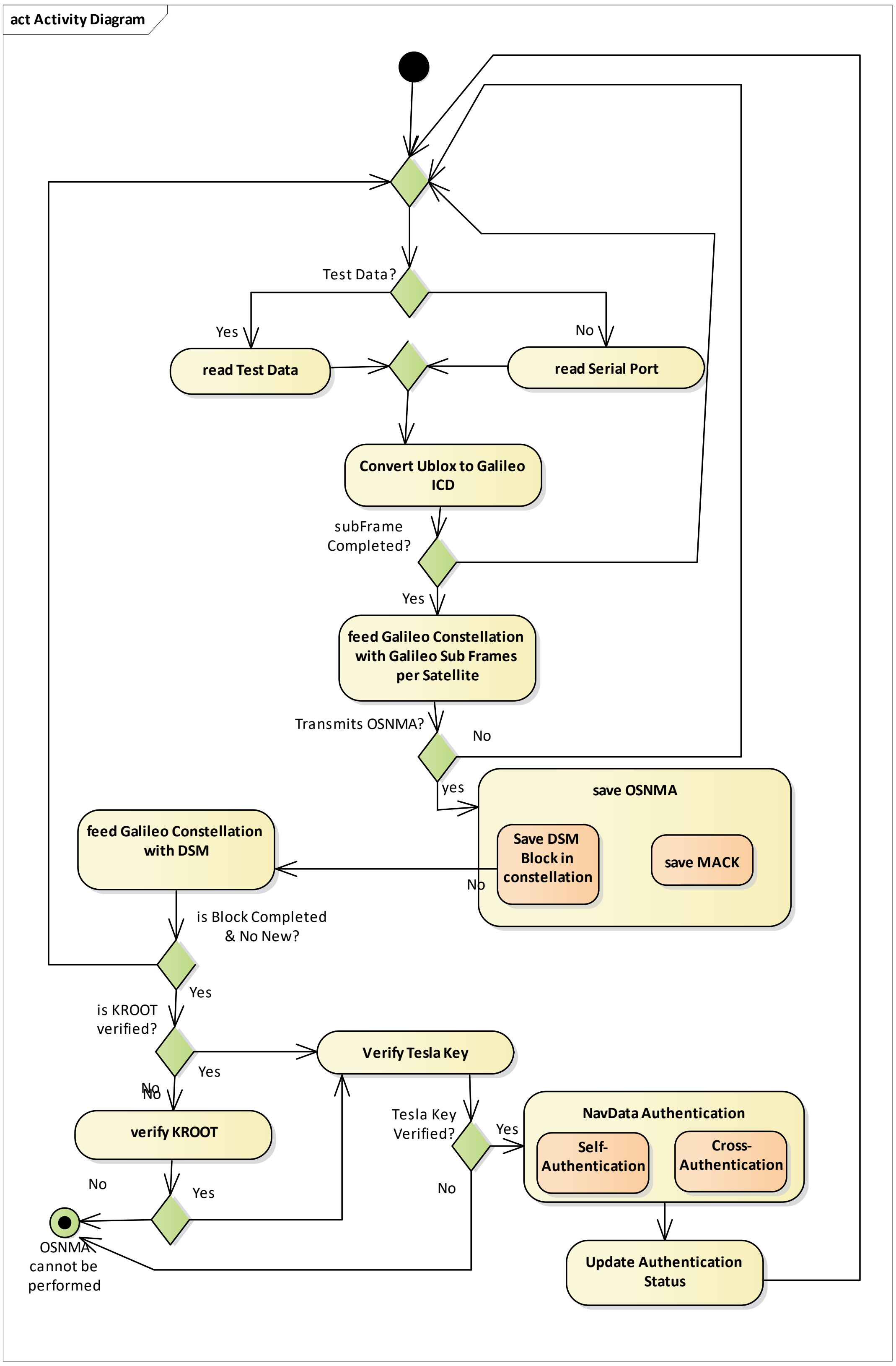


Figure 2-28 Activity Diagram with the high-level activities

3. SOFTWARE OPERATIONS MANUAL

3.1. GETTING THE NAVIGATION DATA FROM A COMMERCIAL GNSS RECEIVER

3.1.1.1. CONFIGURATION OF A UBLOX GNSS RECEIVER

The configuration of an ublox Receiver to get the Navigation Pages is extracted from [RD-3]. Please notice that, at time of writing this document, the most used Ublox models that can give raw Galileo Navigation pages are models family M8 or M9. This has been tested with a receiver from the M9 Family:

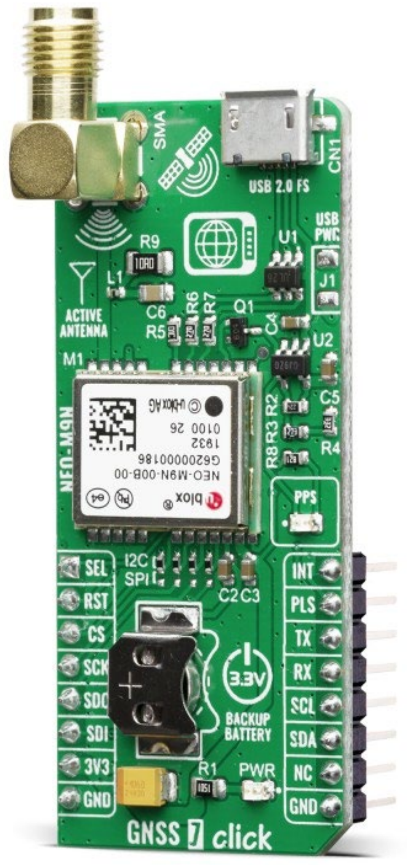


Figure 3-1 GNSS Receiver with ublox M9[7] [8]

Anyhow, it is important to consider the connection conditions, including:

- Boud Rate = 115200
- Frequency: 3Hz
- Set the NMEA protocol 4.1 or higher
- Set to get the Galileo navigation Subframes (SFRBX)

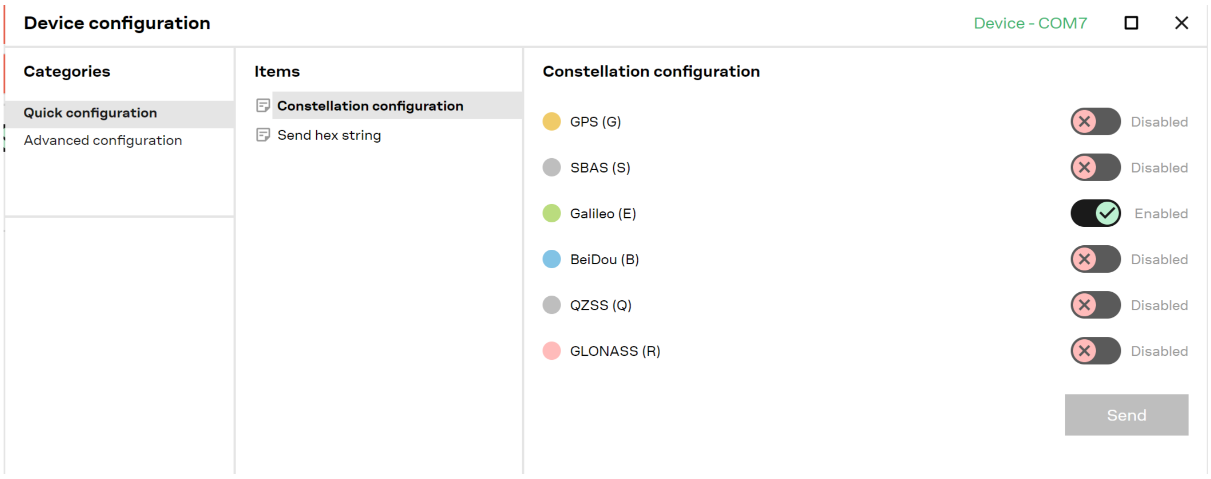
[7] Available in https://www.mikroe.com/gnss-7-click
[8] Please notice that resistor J1 of 0 Ohms has to be added (see under USB PWR) to debug it with the micro USB Port

Ublox provides its own tool to configure their receivers, called “U-Center”, currently with up to version 2. Moreover, there is also a tool called Python tool named PyGPSClient by SemuConsulting.[9] As it is a python tool, it is cross-platform and can be used in multiple OS.

However, for simplifying, the following steps are performed with U-Center 2.[10]

3.1.1.2. SELECT GALILEO AS THE ONLY GNSS PROVIDER

Go to Device Configuration, quick configuration and enable only Galileo:


Device configuration — Device - COM7

Categories: **Quick configuration**, Advanced configuration

Items: **Constellation configuration**, Send hex string

Constellation configuration

GPS (G)	Disabled
SBAS (S)	Disabled
Galileo (E)	Enabled
BeiDou (B)	Disabled
QZSS (Q)	Disabled
GLONASS (R)	Disabled

Figure 3-2 Enabling Galileo

3.1.1.3. SETTING THE BOUD RATE

Search for CFG-UART1 (or the appropriate port) and set the boud rate (in the example 115200)

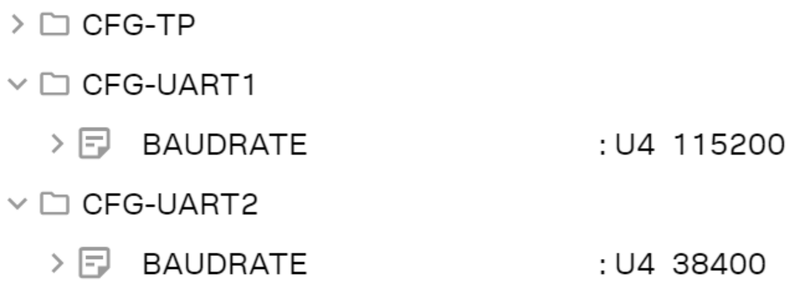
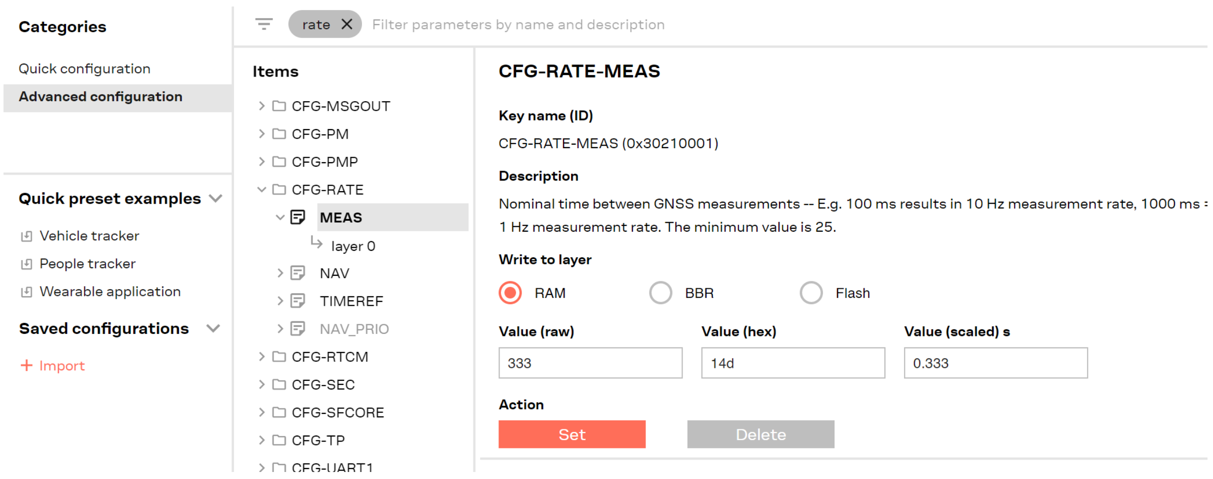

```
> CFG-TP
v CFG-UART1
    > BAUDRATE            : U4  115200
v CFG-UART2
    > BAUDRATE            : U4  38400
```

Figure 3-3 Setting the boudrate

3.1.1.4. SETTING THE FREQUENCY

Set the frequency (in ms). 333ms is roughly the same as 3Hz

[9] Available in their GitHub’s repo under BSD-3-clause license: https://github.com/semuconsulting/PyGPSClient
[10] Check https://www.u-blox.com/en/u-center-2


Categories
Quick configuration
Advanced configuration
Quick preset examples
Vehicle tracker
People tracker
Wearable application
Saved configurations
+ Import

rate ✕ Filter parameters by name and description

Items
CFG-MSGOUT
CFG-PM
CFG-PMP
CFG-RATE
MEAS
layer 0
NAV
TIMEREF
NAV_PRIO
CFG-RTCM
CFG-SEC
CFG-SFCORE
CFG-TP
CFG-UART1

CFG-RATE-MEAS

Key name (ID)
CFG-RATE-MEAS (0x30210001)

Description
Nominal time between GNSS measurements -- E.g. 100 ms results in 10 Hz measurement rate, 1000 ms = 1 Hz measurement rate. The minimum value is 25.

Write to layer
[✓ RAM] BBR Flash

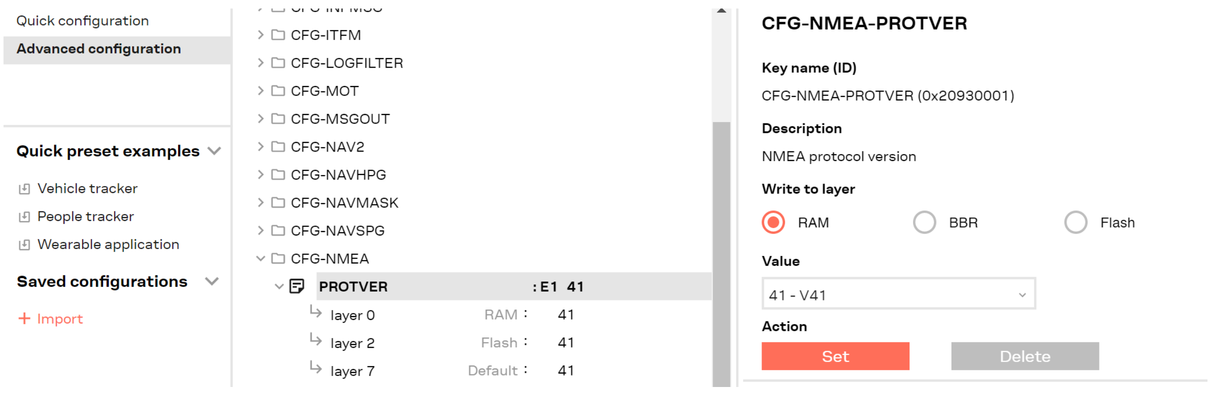
Value (raw): 333
Value (hex): 14d
Value (scaled) s: 0.333

Action
Set Delete

Figure 3-4 Frequency

3.1.1.5. SETTING THE NMEA PROTOCOL

Go to Advanced Configuration, CFG-NMEA and then select value of 41-V41 for 4.1 NMEA version.


Quick configuration
Advanced configuration
Quick preset examples
Vehicle tracker
People tracker
Wearable application
Saved configurations
+ Import

CFG-ITFM
CFG-LOGFILTER
CFG-MOT
CFG-MSGOUT
CFG-NAV2
CFG-NAVHPG
CFG-NAVMASK
CFG-NAVSPG
CFG-NMEA
PROTVER : E1 41
layer 0 RAM : 41
layer 2 Flash : 41
layer 7 Default : 41

CFG-NMEA-PROTVER

Key name (ID)
CFG-NMEA-PROTVER (0x20930001)

Description
NMEA protocol version

Write to layer
[✓ RAM] BBR Flash

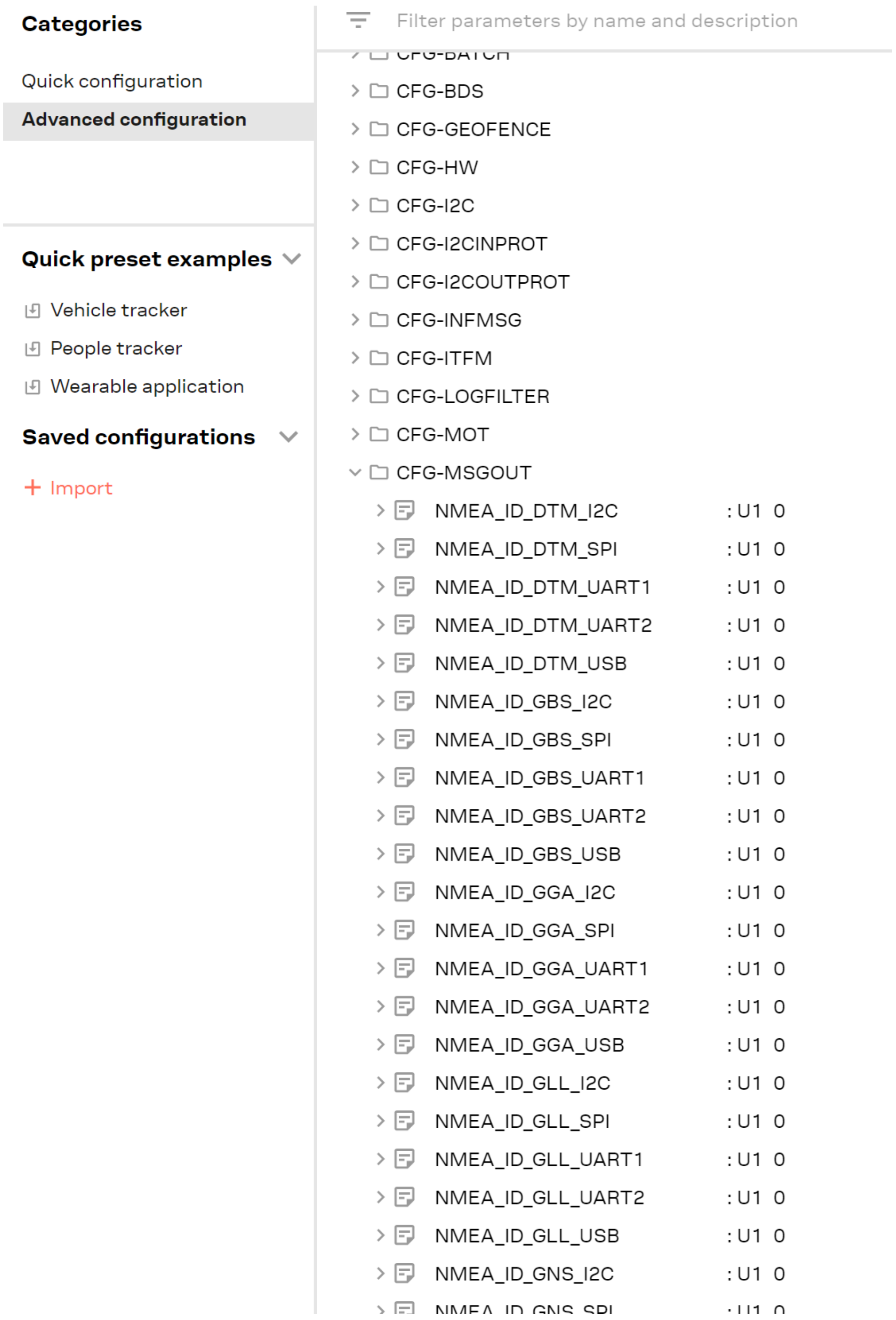
Value
41 - V41

Action
Set Delete

Figure 3-5 NMEA Version

3.1.1.6. GALILEO SFRBX

Go to configuration, Advancec configuration and drop the “CFG-MSGOUT” tab. A lot of fields will open.



Categories

Quick configuration

Advanced configuration

Quick preset examples

Vehicle tracker

People tracker

Wearable application

Saved configurations

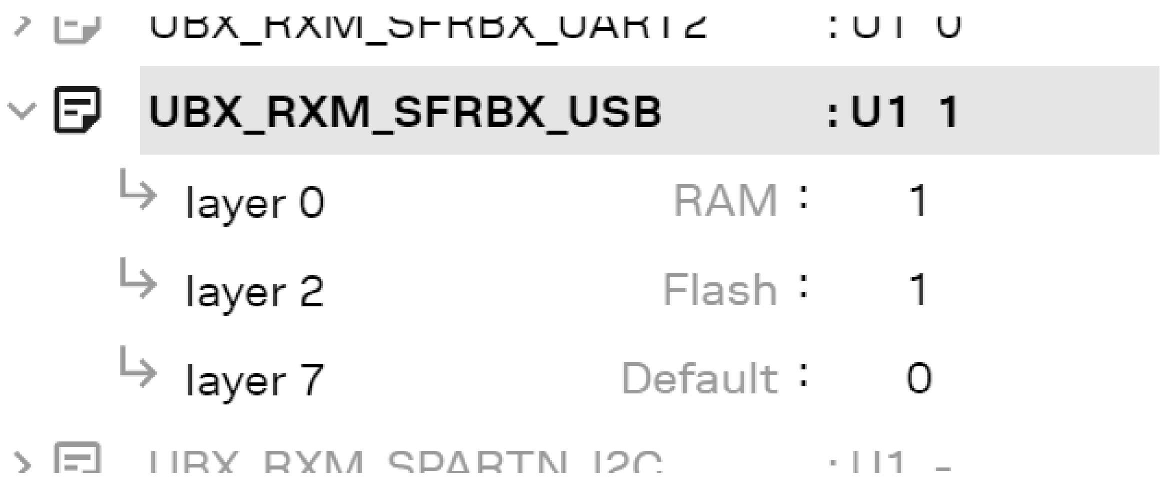
+ Import

Filter parameters by name and description

```
> CFG-BATCH
> CFG-BDS
> CFG-GEOFENCE
> CFG-HW
> CFG-I2C
> CFG-I2CINPROT
> CFG-I2COUTPROT
> CFG-INFMSG
> CFG-ITFM
> CFG-LOGFILTER
> CFG-MOT
v CFG-MSGOUT
    > NMEA_ID_DTM_I2C        :U1 0
    > NMEA_ID_DTM_SPI        :U1 0
    > NMEA_ID_DTM_UART1      :U1 0
    > NMEA_ID_DTM_UART2      :U1 0
    > NMEA_ID_DTM_USB        :U1 0
    > NMEA_ID_GBS_I2C        :U1 0
    > NMEA_ID_GBS_SPI        :U1 0
    > NMEA_ID_GBS_UART1      :U1 0
    > NMEA_ID_GBS_UART2      :U1 0
    > NMEA_ID_GBS_USB        :U1 0
    > NMEA_ID_GGA_I2C        :U1 0
    > NMEA_ID_GGA_SPI        :U1 0
    > NMEA_ID_GGA_UART1      :U1 0
    > NMEA_ID_GGA_UART2      :U1 0
    > NMEA_ID_GGA_USB        :U1 0
    > NMEA_ID_GLL_I2C        :U1 0
    > NMEA_ID_GLL_SPI        :U1 0
    > NMEA_ID_GLL_UART1      :U1 0
    > NMEA_ID_GLL_UART2      :U1 0
    > NMEA_ID_GLL_USB        :U1 0
    > NMEA_ID_GNS_I2C        :U1 0
    > NMEA_ID_GNS_SPI        :U1 0
```

Figure 3-6 Setting UBlox to output SFRBX

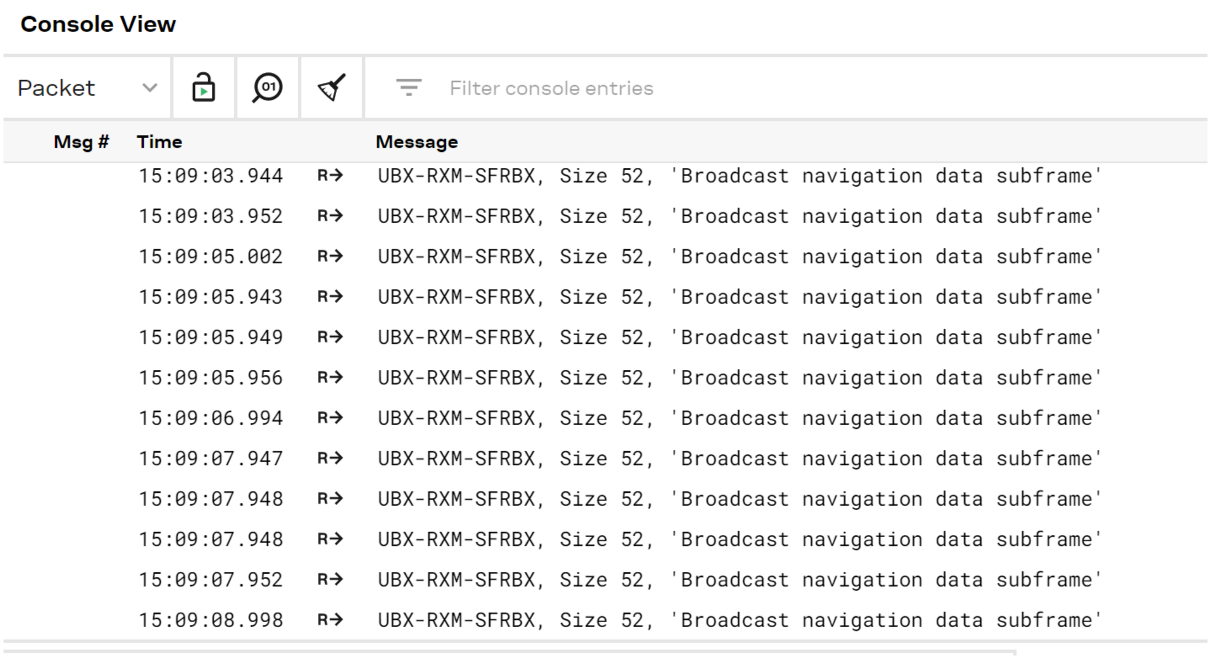
Then, make sure to deactivate all NMEA, by setting them to zero, and set to 1 the UBX_RXM_SFBRX_USB, selecting the layer accordingly



```
UBX_RXM_SFRBX_UART2        :U1  0
UBX_RXM_SFRBX_USB          :U1  1
  layer 0           RAM :   1
  layer 2         Flash :   1
  layer 7       Default :   0
UBX_RXM_SPARTN_I2C         :U1  -
```

Figure 3-7 Detail of SFRBX output

After all the configurations, U-Center console should output something like this:



Console View

```
Msg #  Time           Message
       15:09:03.944 R→ UBX-RXM-SFRBX, Size 52, 'Broadcast navigation data subframe'
       15:09:03.952 R→ UBX-RXM-SFRBX, Size 52, 'Broadcast navigation data subframe'
       15:09:05.002 R→ UBX-RXM-SFRBX, Size 52, 'Broadcast navigation data subframe'
       15:09:05.943 R→ UBX-RXM-SFRBX, Size 52, 'Broadcast navigation data subframe'
       15:09:05.949 R→ UBX-RXM-SFRBX, Size 52, 'Broadcast navigation data subframe'
       15:09:05.956 R→ UBX-RXM-SFRBX, Size 52, 'Broadcast navigation data subframe'
       15:09:06.994 R→ UBX-RXM-SFRBX, Size 52, 'Broadcast navigation data subframe'
       15:09:07.947 R→ UBX-RXM-SFRBX, Size 52, 'Broadcast navigation data subframe'
       15:09:07.948 R→ UBX-RXM-SFRBX, Size 52, 'Broadcast navigation data subframe'
       15:09:07.948 R→ UBX-RXM-SFRBX, Size 52, 'Broadcast navigation data subframe'
       15:09:07.952 R→ UBX-RXM-SFRBX, Size 52, 'Broadcast navigation data subframe'
       15:09:08.998 R→ UBX-RXM-SFRBX, Size 52, 'Broadcast navigation data subframe'
```

Figure 3-8 Final Output

3.2. OSNMAPYTHON LIBRARY

To download the code, simply perform:

```
git clone https://github.com/astromarc/osnmaPython.git
```

This command will download all test data and documentation as well. If you only need the codes, please just download:

```
https://minhaskamal.github.io/DownGit/#/home?url=https://github.com/
astromarc/osnmaPython/tree/master/code
```

Once the Repository is downloaded,

- **Docs**: This folder includes all the documentation data, including this document
- **Test Data**: This folder includes different Test Data. It includes a couple of test data files in the main folder. It also includes a folder with "examples" with real logs and real Test Data directly extracted.

  For test purposes, the file *2022_01_25.csv* is recommended as it has all the functionalities and it was downloaded before the test in the Signal in Space Issue 2 in Galileo Satellites.

- **Code**:
  - getSerialData.py, a standalone module not related with the rest of the project, just to save the raw Ublox Words. This module is not used by the main program: it can be used just to save test input data;
  - logger.py, a centralise of logging configuration
  - dataAcquisition.py;
  - dataTransformation.py;
  - dataProcessingGalileoFrame_Constellation.py
  - dataProcessingOsnma_Authenticator.py
  - dataProcessingOsnma_DSM.py
  - dataProcessingOsnma_svKrootOsnmaMack.py
  - dataVisualisationSupport.py,
  - dataVisualisationSupport_raspberry.py,
  - **mainHMI**.py; the main program

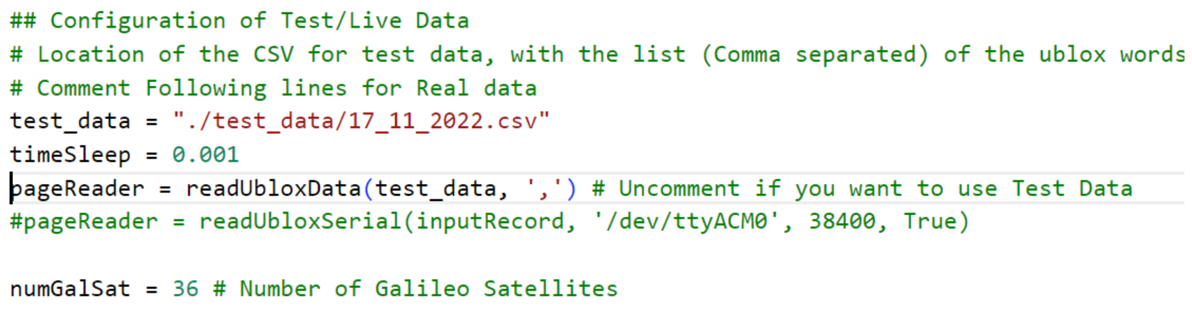
To execute the program, just run mainHMI.py with:

```
python.exe .\code\mainHMI_Laptop.py
```

The previous command assumes Windows and current location as root of the library.

3.2.1. USAGE OF LIBRARY WITH TEST DATA

If you want to use test data, put the path of the data under the test_data variable and use pageReader as the readUblox Data. Time Sleep is the time of reading between rows.

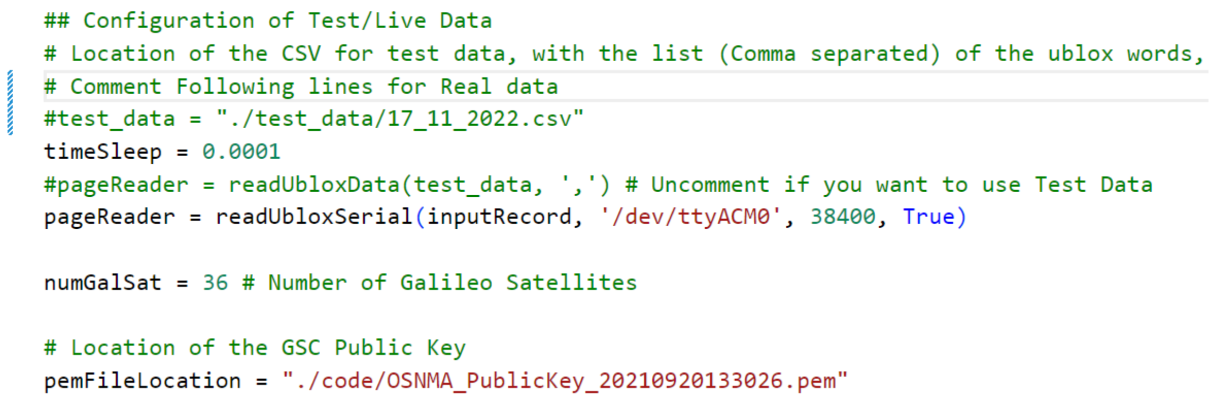


```
## Configuration of Test/Live Data
# Location of the CSV for test data, with the list (Comma separated) of the ublox words
# Comment Following lines for Real data
test_data = "./test_data/17_11_2022.csv"
timeSleep = 0.001
pageReader = readUbloxData(test_data, ',') # Uncomment if you want to use Test Data
#pageReader = readUbloxSerial(inputRecord, '/dev/ttyACM0', 38400, True)

numGalSat = 36 # Number of Galileo Satellites
```

Figure 3-9 Library with Test Data

3.2.2. USAGE OF LIBRARY WITH LIVE DATA

If, instead of using test data, you want to use real data, use readUbloxSerial and put a timeSleep variable low (i.e., 0.0000001) or eliminate any trace of it in the body of the program.

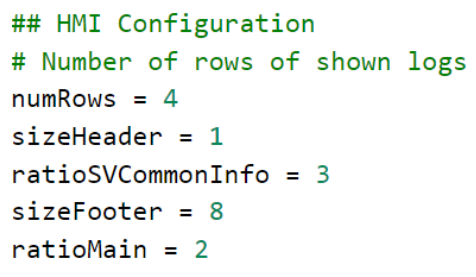


```
## Configuration of Test/Live Data
# Location of the CSV for test data, with the list (Comma separated) of the ublox words,
# Comment Following lines for Real data
#test_data = "./test_data/17_11_2022.csv"
timeSleep = 0.0001
#pageReader = readUbloxData(test_data, ',') # Uncomment if you want to use Test Data
pageReader = readUbloxSerial(inputRecord, '/dev/ttyACM0', 38400, True)

numGalSat = 36 # Number of Galileo Satellites

# Location of the GSC Public Key
pemFileLocation = "./code/OSNMA_PublicKey_20210920133026.pem"
```

Figure 3-10 Library with Real Data

3.2.1. FRONT-END TUNNING

The following variables show the different easy tunings that can be done in the Front-End, in order to adjust it to your screen:

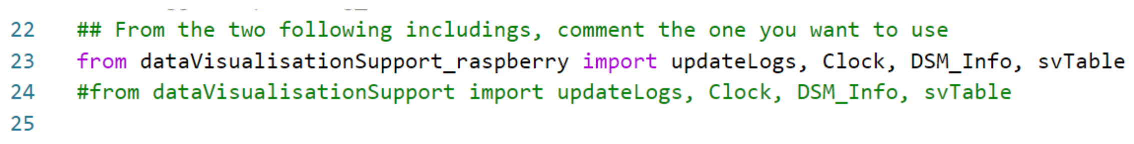


```
## HMI Configuration
# Number of rows of shown logs
numRows = 4
sizeHeader = 1
ratioSVCommonInfo = 3
sizeFooter = 8
ratioMain = 2
```

Figure 3-11 Front-End Configuration

3.2.2. MICROCOMPUTERS SCREEN ADAPTATION

If you are using a Raspberry pi with a 3.5 Inches screen, you may want to use a reduced HMI as the default one does not perfectly fit small screen. For this, just comment/uncomment the module you want to use, as per the following image:



```
## From the two following includings, comment the one you want to use
from dataVisualisationSupport_raspberry import updateLogs, Clock, DSM_Info, svTable
#from dataVisualisationSupport import updateLogs, Clock, DSM_Info, svTable

```

Figure 3-12 Raspberry Pi or Laptop visualisation configuration

3.2.3. OTHER VARIABLES

3.2.3.1. PEM FILE

The public key from GSC (that shall be download from it's webpage, as it can be updated) is loaded with the "pemFileLocation" variable. See Figure 3-10 for place location, and see Annex D from [RD-1] to download the GSC OSNMA Public Key.

3.2.3.2. LOG FILE

By default, log file will be saved following the format YYYY_MM_DD-hh_mm.log

It can be updated with tuning of the "logFile" variable. Further log tunning can be done with the `log_centralise` variable

3.2.3.3. INPUT DATA RECORD NAMING

If you decide to use real data, you can save the raw input in order to further troubleshooting.

By default, input record file will be saved following the format YYYY_MM_DD-hh_mm.csv

It can be updated with tuning of the "`inputRecord`" variable.

4. FUTURE WORK

For the following issues, points below are expected to be added -even though no priority has been set at the moment of writing this:

1. Add the functionality related to DSM-PKR;
2. Add Unit Test;
3. Currently, there is a problem with the table of "Space Vehicles", that it blinks. As it only appears for the raspberry (model zero);
4. Code Re-factoring and class methods cleaning;
5. Add a "expiry" time for Authentication SV Status.
6. Add support for NMEA sentences;
7. Add PVT in the screen;
8. Add a LED light as a visual indicator in the Raspberry Pi prototype
9. Add Sequence Diagrams in the behavioural models;
10. Add further explanation on the Activity Diagram;
11. This python library is a demonstrator prototype for the OSNMA implementation. A port to a C library is expected in the future, to bring this implementation in embedded environments for mass market receivers.

5. ANNEX I: SOURCE CODE

See [RD-7] GitHub Repository – osnmaPython by @astromarc

6. ANNEX II: EVENTS

Event type 1

Event Title: Reception of Navigation Page

Criticality: Info

Definition: The system has received a page from a Space Vehicle. According to the Page Number, the Satellite would be in SiS ICD v1 or SiS ICD v2.

Structure: SV{svId} received page {page number}

Example: `SV18 received page 7`

Event type 2

Event Title: Completeness of SubFrame with OSNMA Information

Criticality: Info

Definition: The system has received a complete Sub-Frame with OSNMA information

Structure: SV{svId} completed SubFrame and Transmits OSNMA

Example: `SV18 completed SubFrame and Transmits OSNMA`

Event type 3

Event Title: Completeness of SubFrame with no OSNMA Information

Criticality: Info

Definition: The system has received a complete Sub-Frame with no OSNMA information

Structure: SV{svId} completed SubFrame and Does Not Transmits OSNMA

Example: `SV2 completed SubFrame and Does Not Transmits OSNMA`

Event type 4

Event Title: DSM Reception of Block

Criticality: Info

Definition: A DSM has received a DSM Block

Structure: DSM{dsmId} Received Block #{numBlock}

Example: `DSM#6 Received Block# 5`

Event type 5

Event Title: DSM Reception of Block not successful

Criticality: Info

Definition: A DSM has received a DSM Block but it is a reserved block

Structure: DSM{dsmId} cannot be added as it is a Reserved block

Example: `DSM#6 cannot be added as it is a Reserved block`

Event type 6

Event Title: DSM is completed

Criticality: Info

Definition: A DSM has received all blocks, and then can be used.

Structure: DSM{dsmId} is complete

Example: `DSM#7 is complete`

Event type 7

Event Title: DSM number total of blocks

Criticality: Info

Definition: A DSM has received a blocks that contain the information of the total number of blocks

Structure: DSM{dsmId} has a total of {numBlocks} blocks

Example: `DSM#6 has a total of 8 blocks`

Event type 8

Event Title: Digital Signature Verification Fail

Criticality: Warning

Definition: a Digital Signature from a DSM has been tried to be authenticated with the GSC Key but the verification failed

Structure: Digital Signature {DS} cannot be verified agains GSC Public Key {GSCPublicKey}

Example:
```
Digital Signature
d835cb30405864ef2e69740e96191dcf651e76cae8cbd0240a5c82bc2321832dcafe400c3434fb
22262e1c832dcafe400c9d216214b1a6e6d809e90c16ed9951 cannot be verified against
GSC Public Key
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAErZl4QOS6BOJl6zeHCTnwGpmgYHEbgezdrKuYu/ghBq
HcKerOpF1eEDAU1azJ0vGwe4cYiwzYm2IiC30L1EjlVQ==
```

Event type 9

Event Title: Digital Signature Verification Success

Criticality: Info

Definition: A Digital Signature is verified against Galileo Service Center Public Key

Structure: Digital Signature {DS} verified against GSC Public Key{GSCPublicKey}

Example: `Digital Signature d1a49167f3deed181e46c277f8553974c5a94f7331c0eee71d50df130df26a807993aad45e0cb01b01f8303a884309390fed112ca64d4d38b21953a3c0b98f9d verified against GSC Public Key MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAErZl4QOS6BOJl6zeHCTnwGpmgYHEbgezdrKuYu/ghBqHcKerOpF1eEDAU1azJ0vGwe4cYiwzYm2IiC30L1EjlVQ==`

Event type 10

Event Title: HKRoot decoded

Criticality: Info

Definition: A log with the trace of an HKRoot from a DSM

Structure: HKroot received {hkRoot}

Example: `HKROOT decoded: 457578747a39404a1fefee5bd408ce93`

Event type 11

Event Title: Tesla Key verification: success

Criticality: Info

Definition: A Tesla Key has been correctly verified against its HKRoot Key

Structure: Tesla Key {teslaKey} belongs to a chain with root {hkRoot}

Example: `Tesla Key: ebb59e689f9cea6ee5f03eb387c603cf belongs to a chain with root 457578747a39404a1fefee5bd408ce93`

Event type 12

Event Title: Tesla Key verification: failure

Criticality: Warning

Definition: A Tesla Key has not been correctly verified against its HKRoot Key

Structure: Tesla Key {teslaKey} cannot be verified against root key {hkRoot}

Example: `Tesla Key: ebb59e689f9cea6ee5f03eb387c603cf cannot be verified against root key 457578747a39404a1fefee5bd408ce93`

Event type 13

Event Title: Self-Authentication: Success

Criticality: Info

Definition: A Space Vehicle has successfully been Self-Authenticated

Structure: SV {svId} is Self-Authenticated

Example: `SV2 is Self-Authenticated`

Event type 14

Event Title: Self-Authentication: Failure

Criticality: Warning

Definition: A Space Vehicle has failed the Self-Authentication

Structure: SV {svId} is not Self-Authenticated. Computed tag {compTag} ; expected tag {expTag}

Example: `SV2 is not Self-Authenticated. Computed tag x/bbta expected tag ‘obbb/xxa`

Event type 15

Event Title: Self-Authentication: Missing Navigation Data

Criticality: Info

Definition: A Space Vehicle Cannot Self-Authenticate due to the lack of Navigation Data

Structure: SV {svId} cannot perform Self-Authentication as it is missing the delayed Navigation Data

Example: `SV2 cannot perform Self-Authentication as it is missing the delayed Navigation Data`

Event type 16

Event Title: Self-Authentication: Missing delayed Tag0

Criticality: Info

Definition: A Space Vehicle Cannot Self-Authenticate due to the lack of delayed Tag 0

Structure: SV {svId} cannot perform Self-Authentication as it is missing the delayed Tag0

Example: `SV2 cannot perform Self-Authentication as it is missing the delayed Tag0`

Event type 17

Event Title: Self-Authentication, PRN255: Success

Criticality: Info

Definition: A Space Vehicle is Self-Authenticated with PRN255

Structure: SV {svId} is Self-Autenticated with PRN 255

Example: `SV2 is Self-Autenticated with PRN 255`

Event type 18

Event Title: Self-Authentication, PRN255: Failure

Criticality: Warning

Definition: A Space Vehicle is Self-Authenticated with PRN255

Structure: SV {svId} failed when Authenticating with PRN 255

Example: `SV2 failed when Authenticating with PRN 255`

Event type 19

Event Title: Self-Authentication, PRN255: Missing Navigation Data

Criticality: Info

Definition: A Space Vehicle cannot self-authenticate itself due to lack of navigation Data

Structure: SV {svId} cannot be Self-Autenticated with PRN255 (ADKD4) due to lack of Navigation Data

Example: `SV2 cannot be Self-Autenticated with PRN255 (ADKD4) due to lack of Navigation Data`

Event type 20

Event Title: Cross-Authentication, ADKD12: Success

Criticality: Info

Definition: A Space Vehicle has Cross-Authenticate ADKD12 (delayed MAC) navData from another SV (can be itself).

Structure: SV {svIdA} has successfully Cross-Authenticated ADKD12 NavData from SV {svIdD}

Example: `SV2 has successfully Cross-Authenticated ADKD12 NavData from SV 2`

Event type 21

Event Title: Cross-Authentication, ADKD12: Failure

Criticality: Warning

Definition: A Space Vehicle has failed when Cross-Authenticate ADKD12 (delayed MAC) navData from another SV (can be itself).

Structure: SV {svIdA} failed when Cross-Authenticating ADKD12 NavData from SV{svIdD}

Example: `SV2 failed when Cross-Authenticating ADKD12 NavData from SV 2`

Event type 22

Event Title: Cross-Authentication, ADKD12: Missing Navigation Data

Criticality: Info

Definition: A Space Vehicle Cannot cross-authenticate Navigation Data from another SV due to the lack of it

Structure: SV {svIdA} cannot Cross-Authenticate ADKD12 NavData from SV{svIdD} due to lack of Navigation data

Example: `SV2 cannot Cross-Authenticate ADKD12 NavData from SV2 due to lack of Navigation Data`

Event type 23

Event Title: Cross-Authentication, ADKD12: Missing Tags

Criticality: Info

Definition: A Space Vehicle Cannot cross-authenticate Navigation Data from another SV due to the lack of Tags

Structure: SV {svIdA} cannot authenticate ADKD12 NavData due to lack of Tags

Example: `SV2 cannot authenticate ADKD12 NavData due to lack of Tags`

Event type 24

Event Title: Cross-Authentication, ADKD 0/4: Success

Criticality: Info

Definition: A Space Vehicle Cannot has cross authenticated the Navigation Data from another Space Vehicle

Structure: SV {svIdA} h `has successfully Cross-Authenticated ADKD{ADKDType} NavData from {svIdD}`

Example: `SV2 has successfully Cross-Authenticated ADKD0 NavData from SV 12`

Event type 25

Event Title: Cross-Authentication, ADKD 0/4: Failure

Criticality: Warning

Definition: A Space Vehicle Cannot has failed when Cross-Authenticating the Navigation Data from another Space Vehicle

Structure: SV {svIdA} `failed when Cross-Authenticating ADKD{ADKDType} NavData from {svIdD}`

Example: `SV2 failed when Cross-Authenticating ADKD0 NavData from SV 12`

Event type 26

Event Title: Cross-Authentication, ADKD 0/4: Missing Navigation Data

Criticality: Info

Definition: A Space Vehicle Cannot Cross-Authenticate the Navigation Data from another Space Vehicle due to lack of navigation Data

Structure: SV {svIdA} cannot authenticate ADKD{ADKDType} NavData from {svIdD} due to lack of navigation data

Example: `SV2 cannot authenticate ADKD0 NavData from SV 12 due to lack of Navigation Data`

End of Document