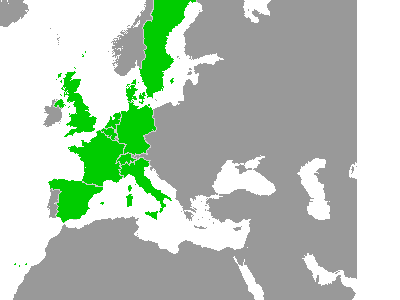
Frame to be used to indicate a customer reference number.

|  |  |  |  |
| --- | --- | --- | --- |
| Client : |  | C/Ref. : |  |

Work-Package 2 : “Requirements”

API Requirements for OpenETCS – v1.4

N. Boverie September 2014



Amendment record

| Rev.**[[1]](#footnote-1)** | Author | Version | Date | § | Modifications |
| --- | --- | --- | --- | --- | --- |
|  | G. Guillaume | 1.0Draft | 01/03/2013 | New | Draft version for review meeting plan on week1311 |
|  | N. Boverie | 1.1Draft | 06/02/2014 | All | \* General format of the document  \* Comment sheet and responses :  - “OETCS\_API Requirements\_v1.0Draft\_130301\_COMT\_PD\_reply Alstom.pdf”  - “Review Sheet API Requirements 2.7\_reply\_Alstom\_2013\_02\_06.xls” |
|  | N. Boverie | 1.2 | 12/02/2014 |  | Inclusion of Alstom internal comments |
|  | N. Boverie | 1.3 | 03/07/2014 | All | \* According to review sheet OETCS\_API\_review\_2014\_07\_03\_SINGLE\_sheet.xlsx  \* §4.7.3 : Add SPEED\_VALUE\_T |
|  | N. Boverie | 1.4 | 08/09/2014 | §1.2, §2.2  §4.4, §4.14 | Update of the DMI interface (Baseline 3)  Update of the Radio interface (Baseline 3) |

Table of Content

1. Introduction 7

1.1 Subject 7

1.2 Field of application 7

1.3 Document description 7

2. Documents & terminology 9

2.1 Reference documents 9

2.2 Applicable documents 9

2.3 Definitions 9

2.4 Abbreviations 9

3. overview 11

3.1 definition of the OpenETCS 11

3.2 OpenETCS API services overview 12

3.3 General principles and rules 13

3.3.1 CORE board : SW architecture overview 13

3.3.2 CORE board : SW execution overview 14

3.3.3 OpenETCS API services and calling sequence 15

3.3.4 Dynamic behaviour and performances 18

3.3.5 Asynchronous behaviour 20

4. INTERFACE SPECIFICATIONS 22

4.1 General usage of message lists 22

4.1.1 Message lists related to EUROCAB peripherals input/output data 22

4.1.2 Message lists related to EURORADIO input/output data 23

4.1.3 Other message lists 24

4.2 Control interface 24

4.2.1 Introduction 24

4.2.2 Functional data flows(see /6/) 24

4.2.3 Service INITIALIZE 25

4.2.4 Service ACTIVATE\_CYCLE 25

4.2.5 SW API extract (ADA Source Code) 25

4.3 Configuration interface 26

4.3.1 Functional data flows (see /6/) 26

4.3.2 Service WRITE\_CONFIG\_DATA 26

4.3.3 Service WRITE\_ETCS\_ID 27

4.3.4 SW API extract (ADA Source Code) 27

4.4 Permanent data interface 27

4.4.1 General description 27

4.4.2 Functional data flows (see /6/) 28

4.4.3 Service WRITE\_PERMANENT\_DATA and WRITE\_PERMANENT\_RADIO\_NETWORK\_INFO 28

4.4.4 Service WRITE\_PROTECTED\_PERMANENT\_DATA 29

4.4.5 Service READ\_PERMANENT\_DATA 29

4.4.6 Service READ\_PROTECTED\_PERMANENT\_DATA 30

4.4.7 SW API extract (ADA Source Code) 30

4.4.8 Additional explanation : 31

4.5 TESTS interface 31

4.5.1 Functional data flows (see /6/) 31

4.5.2 Service POWER\_UP\_TESTS\_FINISHED 32

4.5.3 Service START\_BTM\_TESTS\_REQUIRED 32

4.5.4 Service BTM\_TESTS\_FINISHED 33

4.5.5 SW API extract (ADA Source Code) 33

4.6 TIME interface 34

4.6.1 Functional data flows (see /6/) 34

4.6.2 Service WRITE\_TIME 34

4.6.3 SW API extract (ADA Source Code) 35

4.7 MMU (Odometry) interface 36

4.7.1 Functional data flows (see /6/) 36

4.7.2 Service WRITE\_MMU\_DATA 36

4.7.3 SW API extract (ADA Source Code) 39

4.8 Eurobalise interface 40

4.8.1 Functional data flows (see /6/) 40

4.8.2 Service WRITE\_BTM\_INFO 40

4.8.3 Service WRITE\_BTM\_ANTENNA 41

4.8.4 Service BAD\_BALISE\_RECEIVED 42

4.8.5 Service READ\_BTM\_INFO 42

4.8.6 Service METAL\_MASS\_INFO 43

4.8.7 SW API extract (ADA Source Code) 43

4.9 Euroloop interface 44

4.9.1 Functional data flows (see /6/) 44

4.9.2 Service WRITE\_LOOP\_MESSAGE 45

4.9.3 Service SS\_CODE\_FOR\_LOOP 45

4.9.4 SW API extract (ADA Source Code) 45

4.10 Train Interface Unit (TIU) 46

4.10.1 Functional data flows (see /6/) 46

4.10.2 Service WRITE\_TIU\_MESSAGE 47

4.10.3 Service READ\_TIU\_MESSAGE 47

4.10.4 Service TIU\_MESSAGE\_QUEUE\_IS\_EMPTY 48

4.10.5 Service EB\_REQUESTED 48

4.10.6 Service EVC\_ISOLATION\_IS\_REQUESTED 49

4.10.7 Service OCCUPIED\_CABIN 49

4.10.8 SW API extract (ADA Source Code) 49

4.11 Driver Machine Interface (DMI) 50

4.11.1 Functional data flows (see /6/) 50

4.11.2 Service WRITE\_MMI\_MESSAGE 51

4.11.3 Service READ\_MMI\_MESSAGE 51

4.11.4 Service MMI\_MESSAGE\_QUEUE\_IS\_EMPTY 52

4.11.5 SW API extract (ADA Source Code) 52

4.12 JRU Interface 54

4.12.1 Functional data flows (see /6/) 54

4.12.2 Service WRITE\_JRU\_MESSAGE 54

4.12.3 Service READ\_JRU\_MESSAGE 54

4.12.4 Service JRU\_MESSAGE\_QUEUE\_IS\_EMPTY 55

4.12.5 SW API extract (ADA Source Code) 55

4.13 STM Interface 57

4.13.1 Functional data flows (see /6/) 57

4.13.2 Service WRITE\_STM\_MESSAGE 57

4.13.3 Service READ\_STM\_MESSAGE 57

4.13.4 Service STM\_MESSAGE\_QUEUE\_IS\_EMPTY 58

4.13.5 Service STM\_INFO 59

4.13.6 SW API extract (ADA Source Code) 59

4.14 Euroradio interface 61

4.14.1 Functional data flows (see /6/) 61

4.14.2 Service NUMBER\_OF\_HANDABLE\_RTM\_COMMUNICATION\_SESSION 62

4.14.3 Service WRITE\_MOBILE\_CONTEXT 62

4.14.4 Service READ\_RTM\_MESSAGE 63

4.14.5 Service RTM\_MESSAGE\_QUEUE\_IS\_EMPTY 64

4.14.6 Service WRITE\_RTM\_MESSAGE 65

4.14.7 Service TRAIN\_IS\_IN\_A\_RADIO\_HOLE 65

4.14.8 Complementary expected behaviour and usage of WRITE\_MOBILE\_CONTEXT, READ\_RTM\_MESSAGE and WRITE\_RTM MESSAGE 66

4.14.9 SW API extract (ADA Source Code) 67

4.15 Event report 71

4.15.1 Functional data flows (see /6/) 71

4.15.2 Service EVENT\_REPORT 72

4.15.3 SW API extract (ADA Source Code) 72

4.16 Faults (ERROR MANAGEMENT) 73

4.16.1 Functional data flows (see /6/) 73

4.16.2 Service ERROR\_MANAGEMENT 73

4.16.3 SW API extract (ADA Source Code) 74

5. Appendix 1 – application layer (telegram definition) 75

6. Appendix 2 – Functional data dictionary 75

# Introduction

## Subject

This document is the ALSTOM proposal for the Application Programming Interface (API) Specification of the OpenETCS Onboard Application Software.

This specification is directly based on the Application Programming Interface (API) Specification of the ALSTOM ERTMS Onboard CORE Application Software.

Therefore, in the present document : “ERTMS CORE (Onboard) Application”, “ETCS (Onboard) application”, “OpenETCS (Onboard) Application” or simply “the application” shall all be understood as “the ALSTOM proposal for the OpenETCS Onboard Application”.

This specification defines the operational and functional requirements of the interface between the OpenETCS Application Software and its environment (i.e the Basic Software), including:

* Entry routines the application shall provides
* Input and Output data flows
* The Basic Software functions which the application shall invoke
* Constraints to be respected in order to develop and to use the Application Software.

This is an input document for the subsequent phases of development.

**This document gives the current status of the Alstom API for baseline 3, at the date of edition. As Baseline 3 is still a work in progress, both from standardisation and implementation point of view, this API might be adjusted in the future. This is particularly true with respect to DMI interface (L1LS on going discussions).**

## Field of application

This document is to be considered in the frame of the OpenETCS program.

This specification is compliant to Unisig Baseline 3 of the ETCS Onboard unless explicitly mentioned in the document.

As the ALSTOM development for the ETCS Baseline 3 is still in progress, this document could be modified in the future.

Note:

- The main modification of the present version of the document and its “data dictionary” (appendix /6/) is : the radio communication interface.

- The main modification of the “application interface” (appendix /5/) is : the DMI interface.

## Document description

The document will first provide a context overview of the OpenETCS application:

* Definition of the OpenETCS application within the on-board environment
* OpenETCS API services overview
* General principles and rules : dynamic behaviour and SW architecture

Then this document will present the general description of the different kinds of services and presents the way to use them:

* Definition
* Input and Output Functional data flow
* Application Layer (telegrams) definition (when relevant)
* Constraints
* ADA Software source code extract

# Documents & terminology

## Reference documents

1. System Requirements Specification, ref. SUBSET-026, v3.3.0
2. Glossary of terms and abbreviations, ref. SUBSET-023, v3.0.0
3. FIS Juridical Recording, ref SUBSET-027, v3.0.0
4. FFFIS STM Application Layer, ref SUBSET-058, v3.0.0

## Applicable documents

1. API Requirements for OpenETCS – appendix - Application layer, v1.2
2. API Requirement for OpenETCS – appendix - Functional Data Dictionary, v1.1

## Definitions

Refer also to /2/

## Abbreviations

|  |  |
| --- | --- |
| API | Application Programming Interface |
| APP | Application |
| ASW | Application Software |
| BSW | Basic Software |
| BTM | Balise Transmission Module (is a component of the EVC; is made of the CTE/CIE boards) |
| CORE | The CORE board is the main processing board of the ALSTOM EVC |
| CPBI | Can Profibus Board Interface (is a component of the EVC) |
| CIE | Carte d’Interface Eurobalise (part of the BTM) |
| CTE | Carte de Traitement Eurobalise (part of the BTM) |
| DMI | Driver Machine Interface |
| DRU | Diagnostic Recording Unit |
| EB | Emergency Brake |
| EVC | European Vital Computer |
| GEOS | Generic Enhanced Odometry Subsystem (made of : a processing part which is a component of the EVC and external sensors) |
| JRU | Juridical Recorder Unit |
| KM | Key Manager |
| LTM | (Euro)Loop Transmission Module |
| MMU | Movement Measurement Unit |
| MMI | Man Machine Interface (Obsolete, now called DMI) |
| NOVRAM | Non Volatile RAM memory |
| OETCS | OpenETCS |
| RTM | Radio Transmission Module (is a component of the EVC) |
| SDMU | Speed and Distance Measurement Unit (made of : a processing part which is a component of the EVC and external sensors) |
| STM | Specific Transmission Module |
| SW | Software |
| TBD | To Be Defined |
| TIU | Train Interface Unit |

Refer also to /2/

General remark about abbreviations: This document intends to use as much as possible the abbreviations commonly used by the Unisig and the ETCS community in general (See document /2/).

Nevertheless, for consistency reason, some abbreviations used by Alstom since a long time have been kept in the present document although they have become “obsolete”. It is usually the case when those Alstom specific/obsolete abbreviations are used in the SW model documentation and in the SW source code.

# overview

## definition of the OpenETCS

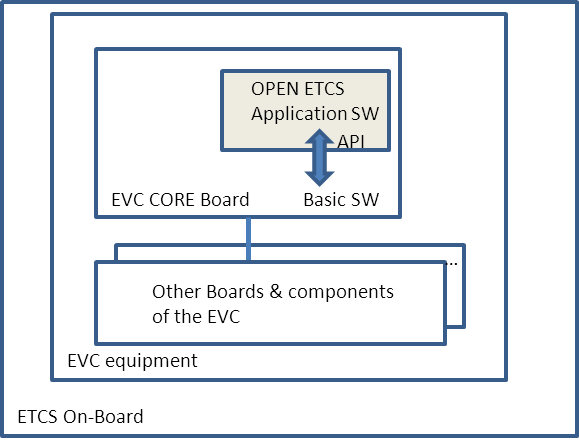
An overview of the ETCS on-board and its interfaces is provided in §2 of the subset-026 (/1/)



The aim of the OpenETCS is to implement the application functions of the ETCS On-board (also named “ETCS Kernel functions”). It does not cover the implementation of the peripheral functions and modules such as : BTM, LTM, EURORADIO, Odometry, Juridical Data, DMI, TIU/BIU and STM control (NTC).

In the frame of the Alstom Onboard architecture, the OpenETCS application shall be implemented on the CORE board of the EVC platform.

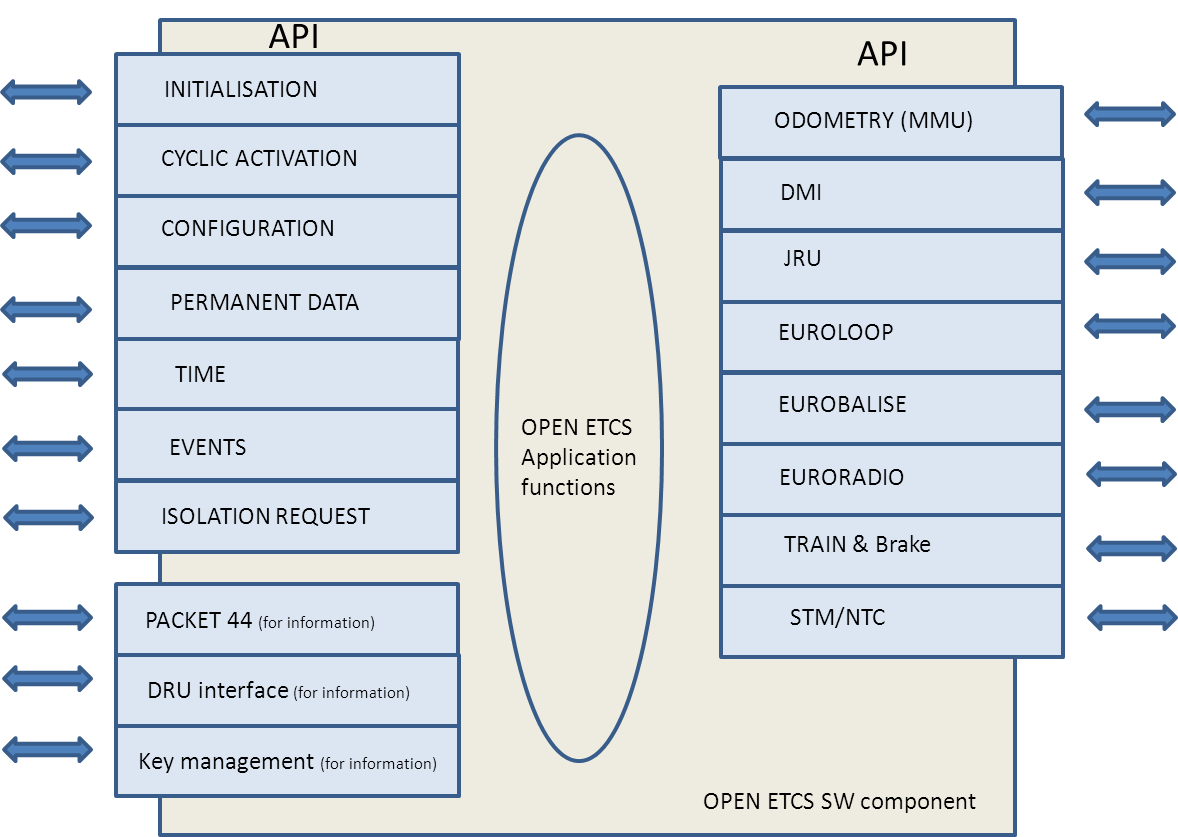
The OpenETCS API shall be the interface between the Application SW and the Basic SW of the CORE board. (notice that the API SW belongs to the Application SW component)



## OpenETCS API services overview

The OpenETCS API shall provide services of different kinds:

* services which are needed in order to achieve the communication between the OpenETCS application and the various peripheral functions and modules of the ETCS Onboard such as defined in the subset 26 and the §3.1 above : e.g DMI,BTM, TIU, etc…
* services which are needed in order to achieve the communication between the OpenETCS application and the various peripheral functions and modules of the ETCS Onboard that are specific to the implementation of some constructor (ALSTOM for instance) : e.g DRU, Key management, … (In this document, some of these "constructor specific" services will simply be listed for information only)
* services which are typically needed in order to implement any Application SW on a target such as : configuration, initialisation, activation, providing the time, error management, …



In the present document, the following services will not be specified because they are specific to the Alstom implementation of the ETCS Onboard : the DRU interface, Packet 44 and Key management.

## General principles and rules

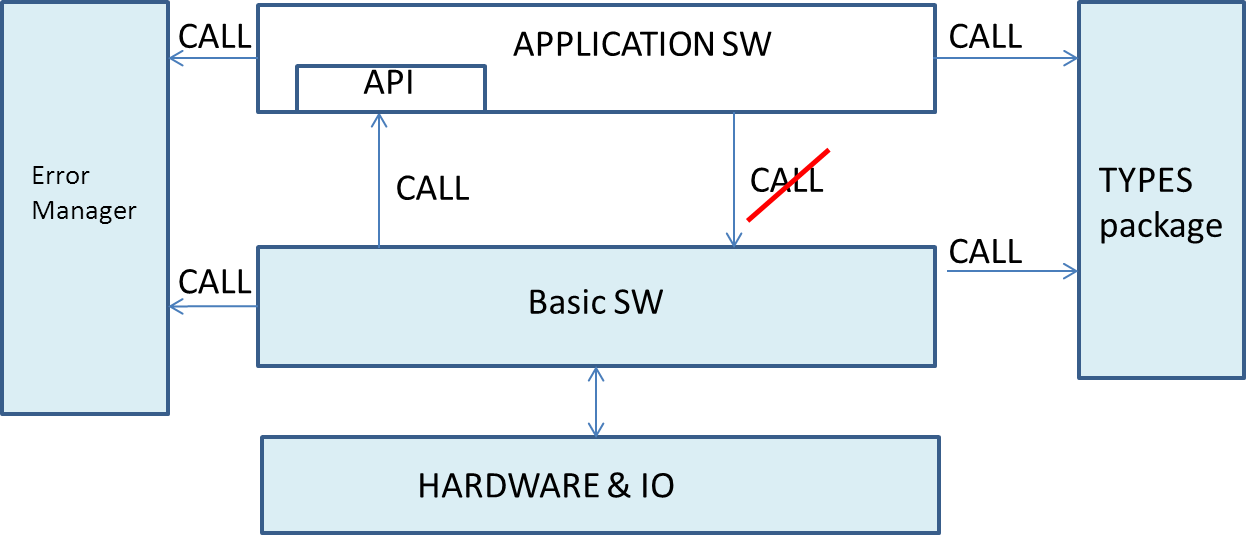
### CORE board : SW architecture overview

From a structural point of view, the Basic Software is in charge of calling all services of the Application SW (through the API).

The Application SW then must execute the actions related to the services requested by the Basic SW and provide back the results to the Basic SW.

In no case shall the Application SW call any service of the Basic SW except for 2 cases :

* The “TYPES” SW component is providing basic services that have to be used by the Application SW (e.g elementary data types such as Integer type, Boolean type, … and the related elementary operations such as “+”, “-“, “or”, “and”, …).   
  Floating point calculation must be supported by the platform (either by HW or SW).
* The “ERROR\_MANAGER” SW component that has to be used by the Application SW in order to communicate the faults to the Basic SW and possibly request actions such as safe platform shutdown, etc…



Remark: The Application SW has minimum dependencies with other SW components and the Hardware. Therefore this SW architecture allows portability of the Application Software on other platforms (PC, others, ..).

### CORE board : SW execution overview

The CORE board Basic SW execution is essentially sequential at the level of the background task (for information, in the Basic SW, some elementary I/O acquisition and low level HW interface tasks run in parallel at higher priority levels but this has no direct relation with the Application SW).

The CORE board OpenETCS Application SW, which is activated by the Basic SW (see above at §3.3.1), shall therefore be fully sequential at the level of the background task.

The main phases of the SW execution sequence, driven by the Basic Software, will be :

* The Initialisation :

At power on, the Basic SW shall first perform internal power-up tests (test of ROM, RAM, …) and low level initialisation.

Then the Basic SW shall initialise the OpenETCS SW (using the API services) by providing this later with its configuration and with its saved NOVRAM data (“permanent data” is the application data saved from the previous mission and restored to the application at initialisation).

The Basic SW then finishes its own initialisation and verifies the communication with all the peripheral boards.

* Cycle (normal working) :

After successful initialisation, the Basic SW will start to run in cycles. There shall be a repetitive and sequential execution of the same actions at each cycle (also called the “mainloop”) :

1. the Basic SW shall read the inputs from HW I/O interfaces

2. the Basic SW shall prepare and provide the inputs to the OpenETCS ASW (using the API services)

3. the Basic SW shall executes (activate) the OETCS ASW (using the API services)

4. the Basic SW shall collect the outputs from OETCS ASW (using the API services)

5. the Basic SW shall prepare and write the outputs to the HW I/O interfaces

* Failure : the normal working cycles shall be interrupted in case of failure (HW failure, System failure, …) detected by the Basic or the Application SW.

In case the Basic SW or the Application SW detects that the platform is not safely operational anymore, a failure manager routine shall be called. This failure manager shall execute minimal treatment (e.g record the failure code in NOVRAM) and then shall enter into a safe failure mode in which the Emergency Brake shall be safely applied. Besides that, no other treatment shall be achieved (no call to Basic SW routines, no call to Application routines). To exit this failure mode, the platform needs to powered-down and then powered-up again.

### OpenETCS API services and calling sequence

There are basically 3 types of API services that are invoked by the Basic SW :

* Services called by the Basic SW to provide the input data to the Application SW (“input routines” in the following figure; from SW architecture point of view “Input Memory” belongs to the application SW)
* Services called by the Basic SW to activate functions of the Application SW (“control routines” in the following figure)
* Services called by the Basic SW to collect the outputs of the Application SW (“output routines” in the following figure; from SW architecture point of view “Output Memory” belongs to the application SW)

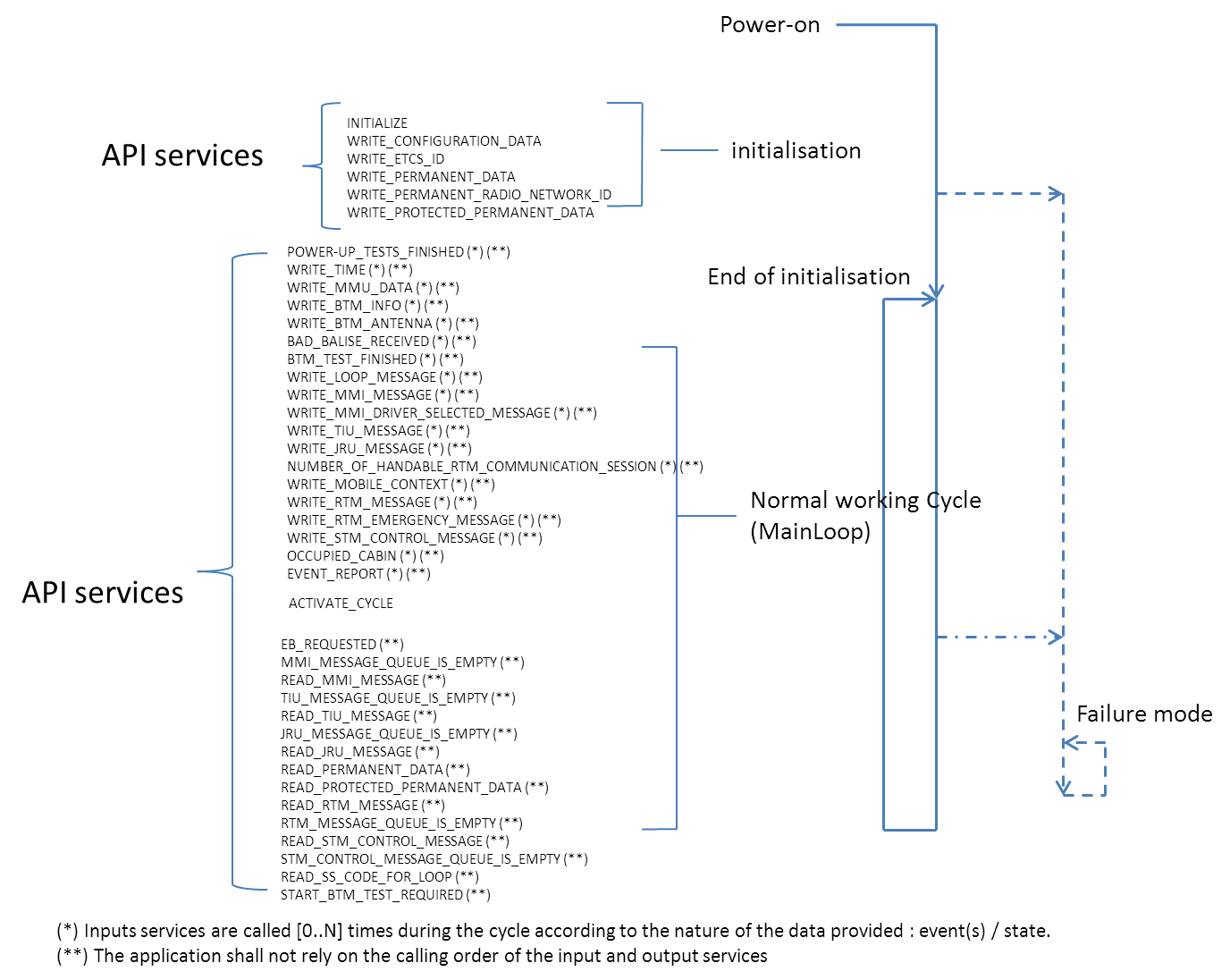
A fourth type is the calling of the ERROR\_MANAGER by the Application SW (“error reporting routines” in the following figure). The calling to the ERROR\_MANAGER shall be used to shutdown the platform in case a safety failure is detected (parameter set to “shutdown request”); see also §3.3.2. Simple events (diagnostic information) can also be reported by the application SW using this service (in that case, there is no request to shutdown the platform and, after its treatment, the Basic SW returns to the Application SW).

In the following figure, the “monitor” is a part of the Basic SW : it achieves the interface between the platform HW and the application. It is also the scheduler of the SW.

In the following figure, the arrows represent “control flows” (calling of routines).



The main phases of the SW execution (see §3.3.2), as well as the calling sequence of the API services, are illustrated (not formal) here after :



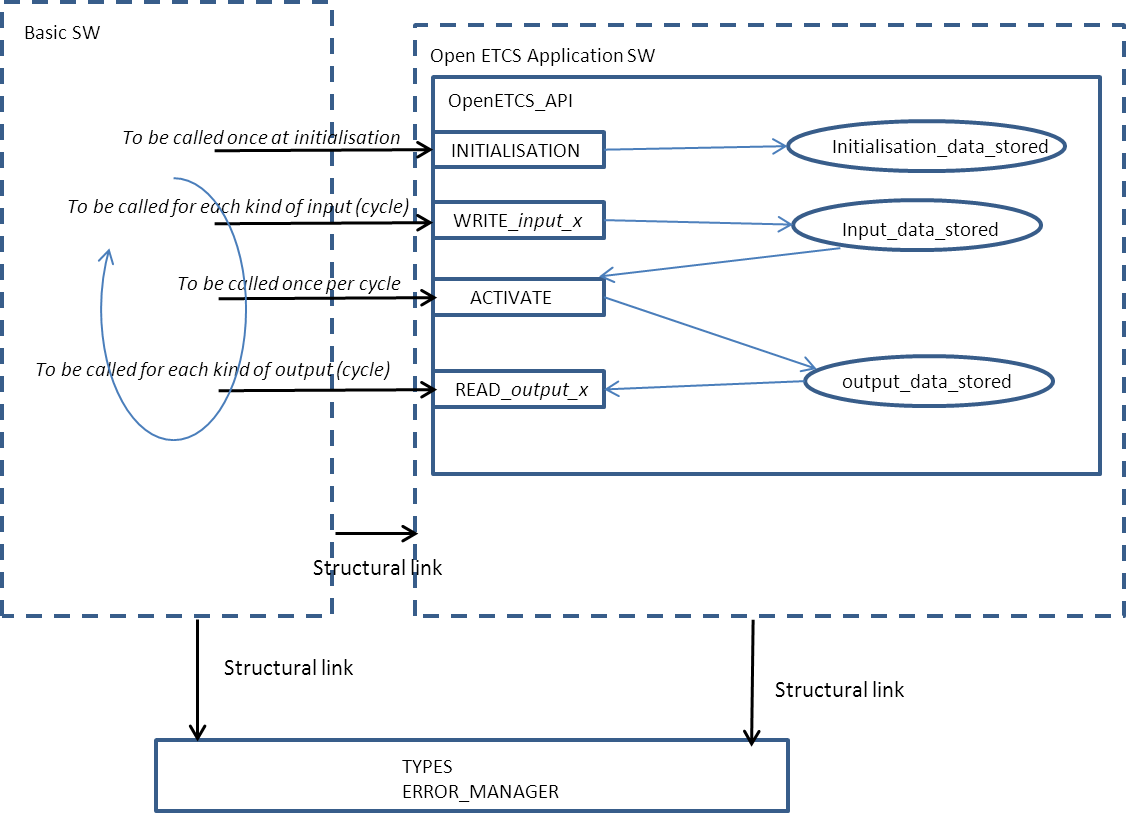
The order into which the inputs and output services of the Application Software are called is NOT defined. It does not have to be formally defined because the Application Software shall not take hypothesis on that order.

Therefore, most of the processing should be performed when the application is activated (ACTIVATE\_CYCLE).

For each “Write inputs” service, it is advised to implement simple treatment such as e.g input data buffering and possibly some filtering; and in all cases, to implement treatment that does not rely on the reception of the data of other “write inputs” services.

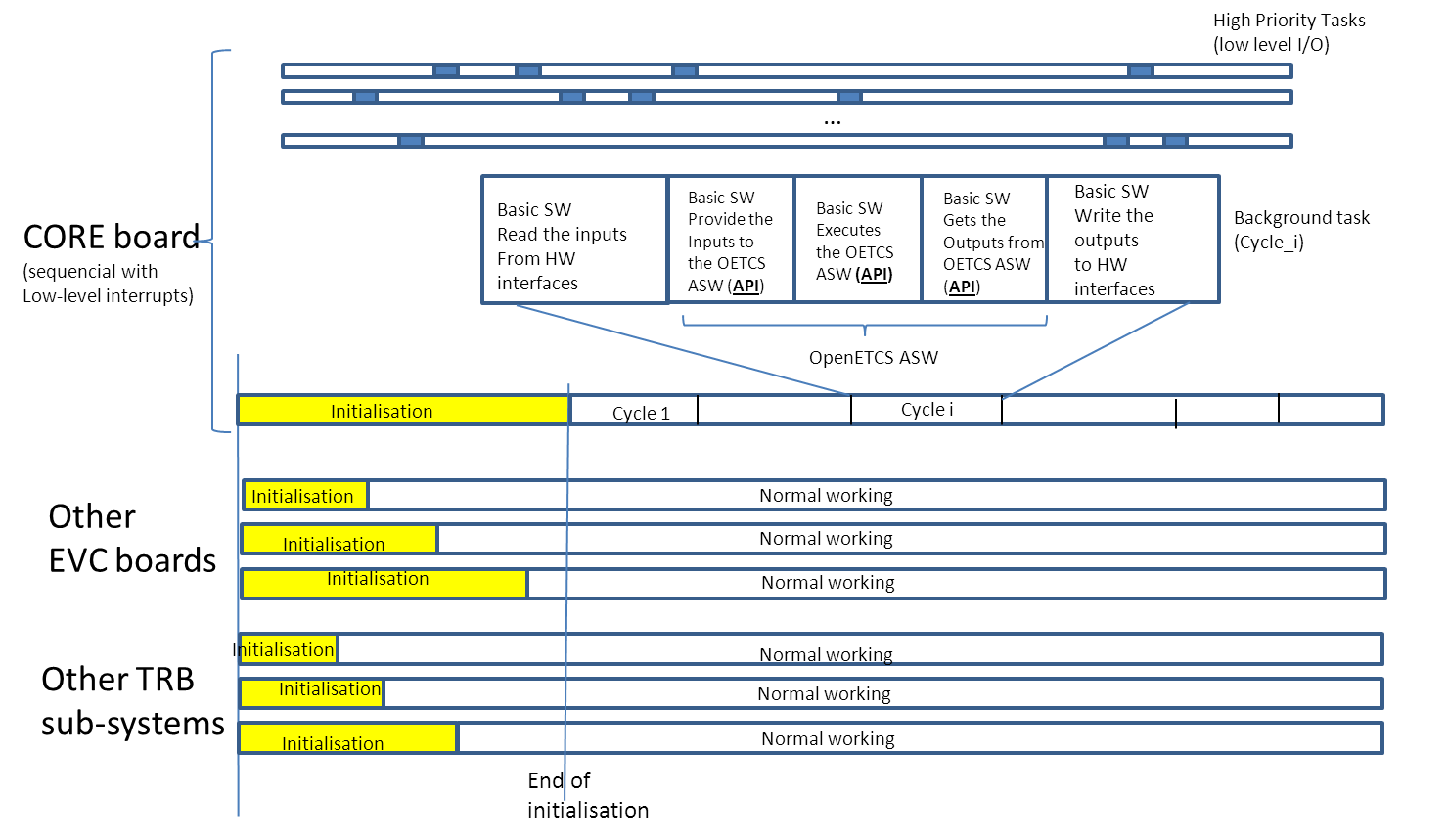
The main application processing shall be executed (when the Basic Software calls the “application activation” procedure) only after all “write inputs” have been called by the Basic Software meaning that all the newest input data of the current cycle are provided to the application.

See another illustration on the following figure :



### Dynamic behaviour and performances

The following figure provides an overview of the dynamic behaviour of the CORE board, the EVC and the other Trainborne sub-systems :



The total execution time taken by the OpenETCS Application SW during initialisation shall be limited to 100ms.

The total execution time taken by the OpenETCS Application SW in one cycle shall be limited to 100ms. This duration includes all Application SW services to be executed during 1 cycle.

The nominal (typical) CORE board cycle duration shall be 300ms on the ALSTOM EVC.

The duration of the CORE board cycle does not have to be constant (It can be constant, it can be variable). The Application SW shall be able to manage constant and variable cycle duration.

The user of the ALSTOM EVC has to be aware that, exceptionally, cycle durations of the CORE board (and the other boards of the EVC) could be higher than the nominal values in case of un-probable happening of several simultaneous conditions. In those cases, a peak of load could be observed due to e.g. ;

* exceptional amount of input/output data to treat;
* exceptional load on the buses and serial lines.

In any case, the maximum duration of the Onboard ETCS (and therefore the maximum duration of the OpenETCS application too) shall be compliant to Unisig Subset-41 performance constraints.

The cycle duration of the CORE board (as well as the other EVC boards) are safely and permanently monitored (by the safe EVC platform) not to be greater than the safe limits, channel by channel (for instance, in the ALSTOM EVC safe 2 out of 3 architecture). In case of overpassing these limits, the faulty component(s)/channel(s) shall be safely shut-down by the safe EVC platform (and possibly the whole EVC shall be shut-down if required).

### Asynchronous behaviour

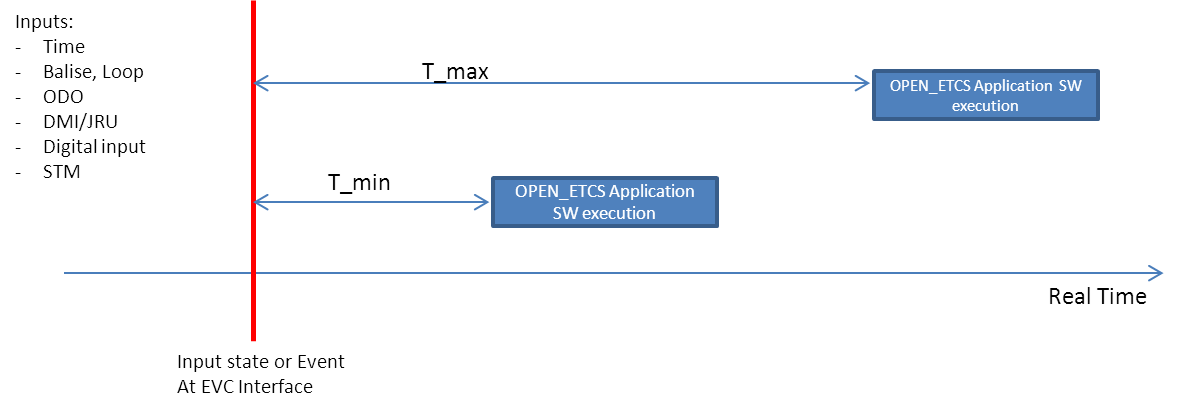
In addition to the “cyclic execution” principle (§3.3.2, §3.3.3), the application must take into account that the various peripheral boards/sub-systems in charge of the input/output treatment are not synchronised and that their performances are not identical, nor constant in time;

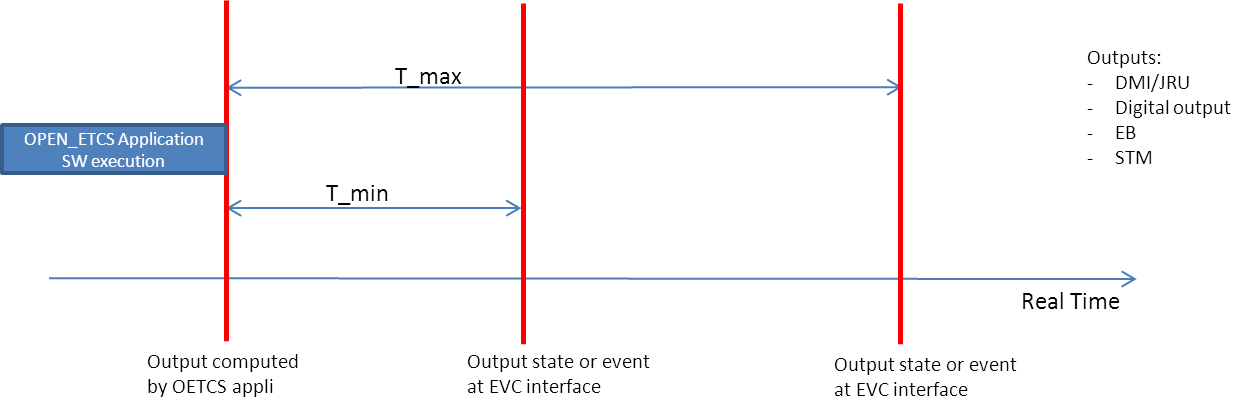
* e.g filtering of analog data such as GEOS sensors, train digital inputs, …
* e.g normal propagation durations on buses for which the performance can vary (chained treatments).

Therefore,

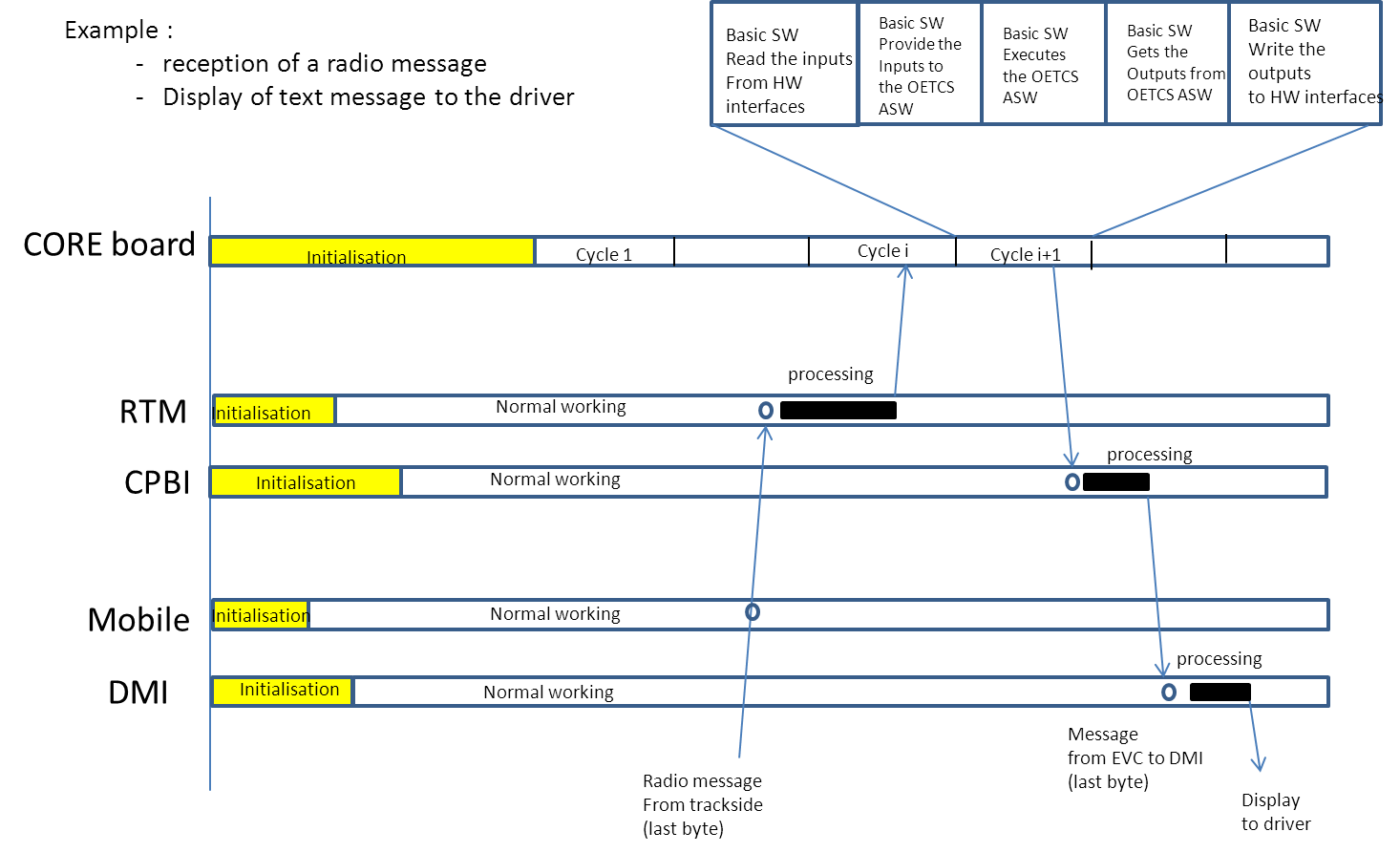
* The input data present at the interface of the Onboard ETCS/EVC will be provided with a variable delay to the application (delay, jitter);
* The application output data are provided to the EVC/Onboard ETCS interface with a variable delay (delay, jitter).

For each input and for each output data, a T\_min (minimum delay of EVC input/output treatment) and a T\_max (maximum delay of EVC input/output treatment) will be defined as shown on the following diagrams :





The Application Software shall have to deal with the various input and output data that are not synchronised as shown in the following example :



Notice that when the input data are time-stamped at the moment they are produced (by the source), the receiving applications may apply a correction in order to manage the delay of transmission of the data; assuming that the clocks of the various calculators are synchronised (e.g. the accuracy of the clock synchronisation within the ALSTOM EVC is 1ms).

# INTERFACE SPECIFICATIONS

## General usage of message lists

The present paragraph presents a general explanation about the OpenETCS\_API services related to reading and writing lists of messages (message stacks).

### Message lists related to EUROCAB peripherals input/output data

The message lists concerned by this paragraph are:

* application data from/to the DMI peripheral
* application data from/to the JRU peripheral
* application data from/to the STM peripheral

Those data links include a common connection/disconnection procedure which is managed by the Application Software and by the Basic Software as explained afterwards.

The Basic Software shall establish the connection with the peripherals during the initialisation of the trainborne system and maintain it during the normal working of the system.

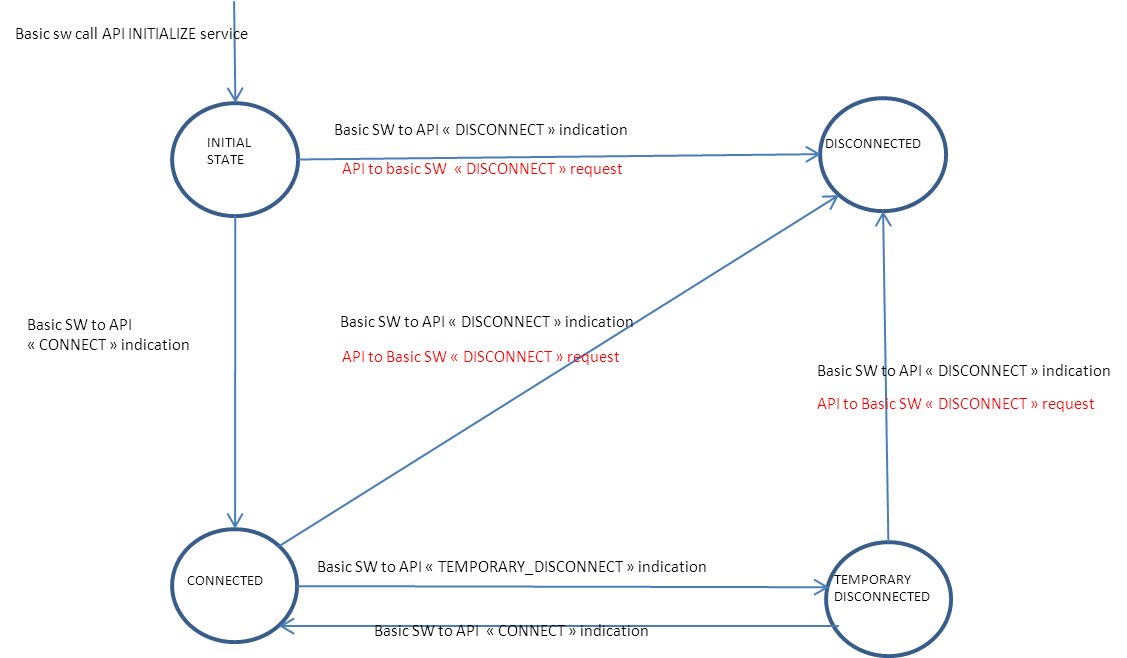
The Basic Software will start to request the first connection with the peripherals not sooner than 20s and not later than 80s after power-up.

Both the basic and the Application SW can request a disconnection.

The following states are defined for each link:

* Initial state
* Connected state
* Temporary\_disconnected state
* Disconnected state

The following state transitions shall be expected (see the following figure) :



In each state, the following actions shall be executed in the current cycle by the Basic SW and by the Application SW:

|  |  |  |
| --- | --- | --- |
| State | Basic SW | OpenETCS application SW |
| Connected | - before ACTIVATE\_CYCLE, will provide all input data available to the application (potentially several calls to the WRITE\_INPUT service)  - after ACTIVATE\_CYCLE, will read all application output data available (potentially several calls to the READ\_OUTPUT service until the output stack is emptied).  In case no output data is available (the output stack is empty), no call to the READ\_OUTPUT will be achieved. | - during ACTIVATE\_CYCLE, will treat all input data provided by basic SW. After treatment, the input stack is supposed to be empty.  - during ACTIVATE\_CYCLE, will generate the output data |
| Initial state  /  Temporary\_Disconnected  /  Disconnected | The same actions as for the “connected state” will be executed except that:  - The Basic SW will not provide any useful DATA to the application ; only connection or disconnection indication will be provided.  - The Basic SW will read the application output data but useful DATA are not expected; if existing useful DATA would be discarded. | The same actions as for the “connected state” will be executed except that:  - the application shall not generate any output DATA; only disconnect request could be generated |

### Message lists related to EURORADIO input/output data

Concerning the RTM (radio), the connection/disconnection procedure is specific : it is managed by the application and the Basic SW and it is described in the related paragraph further in this document (see §4.14).

### Other message lists

The other message lists concerned by this paragraph are:

* application data from/to TIU
* application data from BTM (Eurobalise)
* application data from LTM (Euroloop)

The TIU, BTM and LTM data links may include a connection/disconnection procedure but, if existing, the management of that procedure is fully achieved by the Basic SW.

The following usage rules are applicable :

|  |  |
| --- | --- |
| Basic SW | OpenETCS application SW |
| - before ACTIVATE\_CYCLE, will provide all input data available to the application (potentially several calls to the WRITE\_INPUT service)  - after ACTIVATE\_CYCLE, will read all application output data available (potentially several calls to the READ\_OUTPUT service until the output stack is emptied).  In case no output data is available (the output stack is empty), no call to the READ\_OUTPUT will be achieved. | - during ACTIVATE\_CYCLE, will treat all input data provided by the basic SW. After treatment, the input stack is supposed to be empty.  - during ACTIVATE\_CYCLE, will generate the output data |

## Control interface

### Introduction

The control interface allows the Basic SW to initialise and activate the application at the right times. The Basic SW activates the application on a cycle base, typically 300ms. (please refer to §3.3.3 for more explanation)

### Functional data flows(see /6/)

#### Input

None.

#### Output

None.

### Service INITIALIZE

#### Description

This aim of this service for the Application SW is to elaborate its internal SW data.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

When the Basic SW is elaborated, the initialisation service (INITIALIZE) has to be called by the Basic SW.

Only after, the application configuration data shall be provided by the Basic SW (refer to §4.3 and §4.4).

### Service ACTIVATE\_CYCLE

#### Description

The aim of this service for the application is to execute its main processing once per cycle.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

After the EVC initialisation, at each cycle, the Basic SW will call the input routines, then ACTIVATE\_CYCLE which will run the main application functions. The application will process its inputs and calculate its outputs. Then the Basic Software will call the output routines to access the outputs data’s and messages and apply or send them to the rest of the system.

### SW API extract (ADA Source Code)

package ERTMS\_TRAINBORN\_GENERIC\_API is

.

.

.

-------------------------------------------------------------------------------------------------

-- Control services

-------------------------------------------------------------------------------------------------

-- procedure activated on each cycle.

-- Asks the application to perform a single processing cycle.

procedure ACTIVATE\_CYCLE;

-- procedure activated once at the initialisation of the system (power up)

-- Asks the application to performs all the initialisation actions for the application.

procedure INITIALIZE;

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API;

## Configuration interface

### Functional data flows (see /6/)

#### Input

- basic\_to\_generic\_app\_info .coded\_config\_data

- basic\_to\_generic\_app\_info .specific\_config\_data

#### Output

None.

### Service WRITE\_CONFIG\_DATA

#### Description

This service is used to sends the configuration parameters (data plug) to the Application Software.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| PLUG | PLUG\_BIT\_STREAM\_T | in | The configuration file related to the Application. |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

The configuration service shall be called by the Basic SW once during the initialisation of the EVC, after the initialisation control (INITIALIZE service, see 4.2.3). The application shall decode and store its configuration file.

The parameters data will be provided in the form of a bit stream. The length of that bit stream will be a multiple of 8. There will be at most 7 spare bits at the end of the bit stream.

### Service WRITE\_ETCS\_ID

#### Description

This service is used to send the ETCS\_ID configuration parameter to the Application Software.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| ETCS\_ID | API\_TYPES.ETCS\_ID\_T | in | The ETCS ID parameter |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

The configuration service shall be called by the Basic SW once during the initialisation of the EVC, after the initialisation control (INITIALIZE service, see 4.2.3). The application shall store the ETCS\_ID.

### SW API extract (ADA Source Code)

package ERTMS\_TRAINBORN\_GENERIC\_API is

.

.

.

-------------------------------------------------------------------------------------------------

-- Configuration services

-------------------------------------------------------------------------------------------------

-- procedure activated once at the initialisation to sent the configuration of the system

procedure WRITE\_CONFIG\_DATA (VALUE : in INTERFACE\_LANGUAGE\_TYPES.CORE\_APP\_PLUG\_BIT\_STREAM\_T);

-- procedure activated once at the initialisation to sent the train ETCS ID

procedure WRITE\_TRAIN\_ETCS\_ID (VALUE : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.ETCS\_ID\_T);

.

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API;

## Permanent data interface

### General description

The “permanent data” are data that can be saved by the Application during its normal working in order to be restored at the next power-up.

The Basic SW is in charge of achieving the read-write interface with some Non-Volatile RAM memory (including the verification of the consistency of the data).

The “permanent data” are:

* PERMANENT\_DATA (and PERMANENT\_RADIO\_NETWORK\_ID): those data will be erased (reset) during a normal maintenance operation of the EVC (reading/resetting stored data on-board like diagnose data and permanent data, testing the EVC. Operation to be achieved by the maintenance staff).
* PROTECTED\_PERMANENT\_DATA : those data will not be erased (reset) during a normal maintenance operation of the EVC (reading/resetting stored data on-board like diagnose data and permanent data, testing the EVC).

In order to reset the protected\_permanent\_data, a full re-programming of the EVC is needed. (re-programming of the EVC SW with appropriate tools, procedures and staff)

### Functional data flows (see /6/)

#### Input

- basic\_to\_generic\_app\_info .coded\_data\_restored\_at\_power\_up

#### Output

- generic\_app\_to\_basic\_info .coded\_data\_to\_be\_restored\_at\_power\_up

### Service WRITE\_PERMANENT\_DATA and WRITE\_PERMANENT\_RADIO\_NETWORK\_INFO

#### Description

See 4.4.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| VALUE | API\_TYPES.PERMANENT\_DATA\_T | in | The last value saved in the Novram at the previous power-on or an empty data after a Novram reset or corruption detection. |

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| VALUE | API\_TYPES.PERMANENT\_RADIO\_NETWORK\_INFO | in | The last value saved in the Novram at the previous power-on or an empty data in case of corruption. |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

This service shall store the data into a local buffer in order to use it as starting value for the train configuration.

The Basic SW shall call this service once during the initialisation phase; after the initialisation of the Application SW (service INITIALIZE).

The data are protected against corruption by the Basic SW, but this later does not guarantee that the data shall always have a content that is valid compared with the current Application SW installed on the train.

It is advised that the Application SW adds a structure version in its data to ensure that the data are usable. If that version is not the expected one, the Application SW should not decode the data anymore (they were probably written with an older version of the software).

### Service WRITE\_PROTECTED\_PERMANENT\_DATA

#### Description

See 4.4.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| VALUE | API\_TYPES.PROTECTED\_PERMANENT\_DATA\_T | in | The last value saved in the Novram at the previous power-on or an empty data after corruption detection. |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

This service shall store the data into a local buffer in order to use it as starting value for the train configuration.

The Basic SW shall call this service once during the initialisation phase; after the initialisation of the Application SW (service INITIALIZE).

The data are protected against corruption by the Basic SW, but this later does not guarantee that the data shall always have a content that is valid compared with the current Application SW installed on the train.

It is advised that the Application SW adds a structure version in its data to ensure that the data are usable. If that version is not the expected one, the Application SW should not decode the data anymore (they were probably written with an older version of the software).

As these data CANNOT be erased by the “normal maintenance” team (see 4.4.1), the Application SW must ensure that the decoding of that PROTECTED\_PERMANENT\_DATA that it will achieve shall not lead to a failure (shutdown) of the EVC SW. If it is the case, the EVC shall never be able to start again until a new software is downloaded (re-programming of the EVC SW with appropriate tools, procedures and staff) to restore a correct content to that Novram data.

### Service READ\_PERMANENT\_DATA

#### Description

See 4.4.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | API\_TYPES.PERMANENT\_DATA\_T | - | The data to store in the Novram of the EVC |

#### Expected behaviour & usage

This service shall return the data to be stored in the Novram.

The Basic SW shall call this service once per cycle after the activation of the Application SW.

The value of each used byte must be defined (constrained - no spare bytes)

### Service READ\_PROTECTED\_PERMANENT\_DATA

#### Description

See 4.4.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | API\_TYPES.PROTECTED\_PERMANENT\_DATA\_T | - | The data to store in the Novram of the EVC |

#### Expected behaviour & usage

This service shall return the data to be stored in the Novram.

The Basic SW shall call this service once per cycle after the activation of the Application SW.

The value of each used byte must be defined (constrained - no spare bytes)

### SW API extract (ADA Source Code)

…

type NETWORK\_REGISTRATION\_INFO\_T (BEARER\_TYPE : BEARER\_CAPABILITY\_T := GSM\_R\_ONLY) is

record

NETWORK\_ID : RADIO\_NETWORK\_ID\_T;

case BEARER\_TYPE is

when GSM\_R\_ONLY =>

null;

when GPRS\_WITH\_GSM\_R\_FALLBACK =>

NID\_APN : APN\_IDENTITY\_T;

end case;

end record;

-- OpenETCS : BEARER\_TYPE shall always be GSM\_R\_ONLY

….

package ERTMS\_TRAINBORN\_GENERIC\_API is

.

.

.

-------------------------------------------------------------------------------------------------

-- Permanency services

-------------------------------------------------------------------------------------------------

-- procedure activated once at the initialisation to restore the previous state of the system.

-- VALUE shall be (VALUE.SIZE = 0) when the NOVRAM is empty to use the FIRST\_PERMANENT\_DATA\_C

-- permanent data.

-- A length coherence is done to be sur all given bytes are used during the conversion.

procedure WRITE\_PERMANENT\_DATA (VALUE : in TYPES.BYTE\_DATA\_T);

-- function read on each cycle to save the current state of the system. If the returned value is

-- too much long (more than MAX\_PERMANENT\_DATA\_MESSAGE\_SIZE\_C bytes), an empty BYTE\_STRING\_T is

-- returned and an ATBL error is raised.

function READ\_PERMANENT\_DATA return TYPES.BYTE\_DATA\_T;

-- procedure activated once at the initialisation to restore the RADIO NETWORK ID stored into the BSW NOVRAM

procedure WRITE\_PERMANENT\_RADIO\_NETWORK\_INFO (VALUE : ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.NETWORK\_REGISTRATION\_INFO\_T);

-- procedure activated once at the initialisation to restore the previous protected data of the system.

-- VALUE shall be (VALUE.SIZE = 0) when the NOVRAM is empty to use the FIRST\_PROTECTED\_PERMANENT\_DATA\_C

-- protected permanent data.

-- A length coherence is done to be sur all given bytes are used during the conversion.

procedure WRITE\_PROTECTED\_PERMANENT\_DATA (VALUE : in TYPES.BYTE\_DATA\_T);

-- function read on each cycle to save the current state of protected data of the system. If the returned value is

-- too much long (more than MAX\_PROTECTED\_PERMANENT\_DATA\_MESSAGE\_SIZE\_C bytes), an empty BYTE\_STRING\_T is

-- returned and an ATBL error is raised.

function READ\_PROTECTED\_PERMANENT\_DATA return TYPES.BYTE\_DATA\_T;

.

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API;

### Additional explanation :

When decoding the Novram data, the application SW should use "defensive programming" so that in case of inconsistent non-volatile memory data (e.g the version of the data structure is not good) , it will not crash the SW. Instead, it should continue with "default values", or prompt for a new data entry on the DMI, etc ... This is even more important for the NOVRAM area which is "protected" because in that case, the service tool of the EVC (“normal maintenance”) can't reset the NOVRAM data.

## TESTS interface

### Functional data flows (see /6/)

#### Input

- basic\_to\_generic\_app\_info .power\_up\_tests\_info

#### Output

None.

### Service POWER\_UP\_TESTS\_FINISHED

#### Description

Power-up tests are started automatically by the Basic SW during the Initialisation phase.

During those tests, Onboard ETCS and EVC components are tested. Typically:

* Eurobalise antenna
* ODOMETRIC sensors
* Radio sub-system (Mobile Terminals not included)
* Each channel of the EVC
* Digital I/O interfaces

The duration of the power-up tests can be higher than the initialisation phase (typically up to 1 minute).

The result of the power-up tests is then provided to the application during normal working.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| WITH\_RESULT | API\_TYPES.TEST\_RESULT\_T | in | The result of the target's external equipment power-up tests. (OK, NOT\_OK, REDUCED\_DISPONIBILITY); |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

The Basic SW shall call this service once after the end of the power-up tests. This shall be done typically about 30 sec to 1 minute (during the normal working phase) after the EVC power-on. The Application SW shall already have been initialised and some cycles elapse before the reception of this event.

The Application SW can use this service to indicate a status to the driver :

|  |  |  |
| --- | --- | --- |
| Status provided | Description | Expected action |
| OK | The EVC works in full availability mode | Maintenance event to the driver |
| REDUCED\_DISPONIBILITY | Some redundancies of the EVC are not working. The train is able to work but it may not be able to support a failure | Maintenance event to the driver or other action according to system team |
| NOT\_OK | The mission cannot be started | Maintenance event to the driver and stop the train. |

### Service START\_BTM\_TESTS\_REQUIRED

#### Description

The service is used by the Application to request a BTM test (and Eurobalise antenna test) test to the Basic Software.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | Boolean | Out | True : start test  False:no test to be started |

#### Expected behaviour & usage

The START\_BTM\_TESTS\_REQUIRED function should be called after each Activate\_Cycle. When a test is started, all BTM test request from the Application SW shall be ignored by the Basic SW until the test is finished.

### Service BTM\_TESTS\_FINISHED

#### Description

When the BTM antenna test is finished, the procedure BTM\_TESTS\_FINISHED must be called by the Basic SW to give the test result to the application.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| WITH\_RESULT | API\_TYPES.TEST\_RESULT\_T | in | The result of the BTM tests. (OK, NOT\_OK, REDUCED\_DISPONIBILITY); |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

The BTM\_TESTS\_FINISHED function shall be called once, before Activate\_Cycle, only when the BTM test is finished.

### SW API extract (ADA Source Code)

-- Possible result of the power up tests

type TEST\_RESULT\_T is (OK, NOT\_OK, REDUCED\_DISPONIBILITY);

package ERTMS\_TRAINBORN\_GENERIC\_API is

.

.

.

-------------------------------------------------------------------------------------------------

-- Test services

-------------------------------------------------------------------------------------------------

-- procedure to deliver balises telegrams and coordinate to the application

function START\_BTM\_TESTS\_REQUIRED return TYPES.BOOLEAN\_T;

-- procedure called when the btm tests are finished with the given result

procedure BTM\_TESTS\_FINISHED (WITH\_RESULT : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.TEST\_RESULT\_T);

-- procedure called when the power-up tests are finished with the given result

procedure POWER\_UP\_TESTS\_FINISHED

(WITH\_RESULT : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.TEST\_RESULT\_T);

.

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API;

## TIME interface

### Functional data flows (see /6/)

#### Input

- basic\_to\_generic\_app\_info .logical\_voted\_time

#### Output

None.

### Service WRITE\_TIME

#### Description

This service is used by the Basic SW layer to provide the current time to the Application SW.

The time provided to the application is sampled from the EVC clock (BSW Clock) once per cycle and is not refreshed until the next cycle.

The time provided to the application is equal to 0 at power-up of the EVC (it is not a “UTC time” nor a “Local Time”), then must increase at each cycle (unit = 0,01s), until it reaches its maximum value (i.e current EVC limitation = 24 hours)

The Application shall never use directly the Basic SW clock.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| VALUE | API\_TYPES.CLOCK\_T | in | The current time as to be taken by the Application SW. The unit of the time provided is the hundredth of second (10 ms) |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

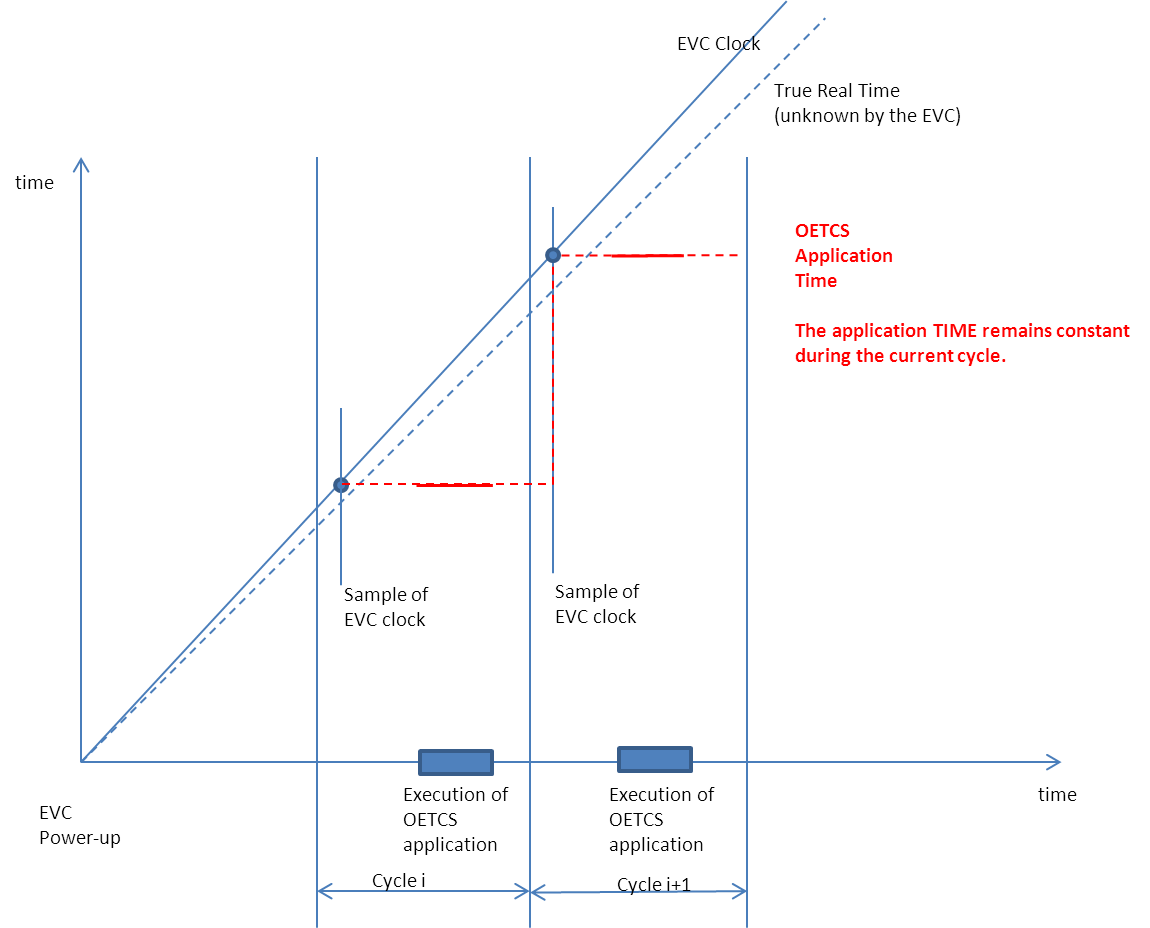
The Application SW should store the time and use it for all its internal computations.

As the cycle duration is not constant, using a cycle counter is not appropriate.

The Basic SW shall call this service once at the beginning of each cycle, before providing the other input data of the cycle.

Care must be taken that the time is sampled at the beginning of the cycle.

The following figure defines the way the time is provided to the Application SW:



### SW API extract (ADA Source Code)

-- System clock in HUNDREDTH\_OF\_SECOND

type CLOCK\_T is new TYPES.NATURAL\_LONG\_T;

.

.

.

package ERTMS\_TRAINBORN\_GENERIC\_API is

-------------------------------------------------------------------------------------------------

-- Time services

-------------------------------------------------------------------------------------------------

-- procedure activated on each cycle to give the current application time

procedure WRITE\_TIME

(VALUE : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.CLOCK\_T);

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API;

## MMU (Odometry) interface

### Functional data flows (see /6/)

#### Input

- basic\_to\_generic\_app\_info .MMU\_input\_info

#### Output

None.

### Service WRITE\_MMU\_DATA

#### Description

The service is used by the Basic SW to provide the current train MMU data to the train.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| COORDINATE | API\_TYPES.MMU\_COORDINATE\_T | in | The current position of the train, bounded with the Upper and Lower values.  The position given is related to the train (does not change with the selected cabin).  For motions where cab A cross a point of the track before cab B, the location increases into the positive.  For motions where cab B cross a point of the track before cab A, the location decreases into the negative.  [m]  The highest value, the lowest value and the nominal value (i.e. the most probable) are absolute values. |
| SPEED | API\_TYPES.MMU\_SPEED\_T | in | The current speed of the train, bounded by the Upper and Lower values. In more details  - The upper estimation of the speed  - The lower estimation of the speed  - The safe nominal estimation of the speed which will be bounded between 98% and 100% of the upper estimation  - The raw nominal estimation of the speed which will be bounded between the lower and the upper estimations  The speed is always positive, ranging from 0 to 600 km/h  [m/s] |
| ACCELERATION | API\_TYPES.ACCELERATION\_VALUE\_T | in | Acceleration of the train  [m/s²] |
| MOTION\_STATE | API\_TYPES.MOTION\_STATE\_T | in | Indicate when the train is moving or not. |
| MOTION\_DIRECTION | API\_TYPES.MOTION\_DIRECTION\_T | in | Indicates the direction of the train |
| TIME\_STAMP | API\_TYPES.CLOCK\_T | in | The time at which the measure was valid  (same referencial as WRITE\_TIME service) |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

The Basic SW shall call this service once per cycle before the activation of the Application SW to provide the latest MMU data available.

The application must be able to cope with odometry data that contains UNKNOWN fields.

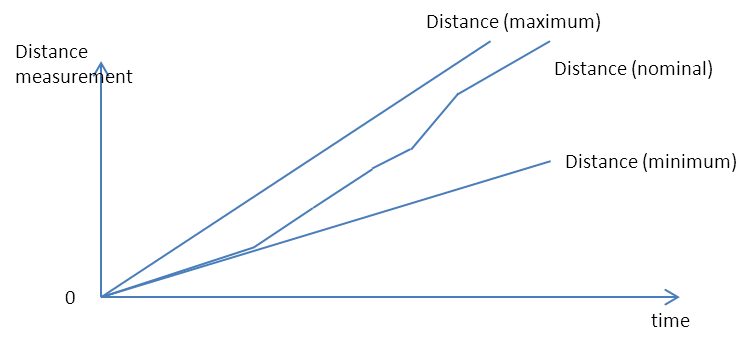
If the motion cannot be determined (i.e. due to a very low speed or small distance travelled), the MOTION\_STATE or MOTION\_DIRECTION may be set to UNKNOWN by the Basic Software.

For both speed and coordinate LOWER\_VALUE and UPPER\_VALUE are absolute values in the same reference as NOMINAL\_VALUE.

The ACCELERATION is provided for ease of use, but it is not a safe value, and is not used in safety relevant calculations. The provided acceleration data must be consistent with the maximum train acceleration and maximum train deceleration parameters.

The position of the train is always 0 at the power-up of the EVC and it corresponds to the head of the train.

The nominal distance will always be in the range of the minimum distance and the maximum distance. But it can be closer to one or the other boundary. (same principle for the speed measurement). An example is provided here after :



E.g. distance nominal bounded by distance maximum and minimum

The error range |Distance\_max – Distance\_min| will always increase during the time. Here-after is an example when the train is going first in forward and then after in reverse :

##### 

E.g Train direction forward and then reverse

When passing over a balise, the reset of the error range |Distance\_max – Distance\_min| is to be managed by the application.

Distance measurement, Speed measurement and motion detection data will always be consistent.

When the train starts moving, the Motion Direction may be set to “unknown” due to very low speed of the train (under ~1 km/h). In such case, the distance measurement data may vary as following (e.g) :

##### 

E.g. start of the train (zooming on the lapse of time during which the motion-direction of the train is “unknown”)

In the EVC, the MMU data estimation are computed by the GEOS board using a smoothing mechanism over the measurement samples obtained during the last 500 ms leading to a delay of 250 ms on the estimation (time-stamp). It is to notice that this 250ms delay is extrapolated directly by the GEOS board for the “distance minimum” and “distance maximum” data only.

Remark : train\_acceleration\_upper\_bound and train\_accelration\_lower\_bound are defined in the functional model but are not implemented in the SW API. Only the nominal value of the acceleration measure is available in the SW.

### SW API extract (ADA Source Code)

-------------------------------------------------------------------------------------------------

-- MMU interface types

-- Speed in m/sec range 0 .. 600 km/h

subtype SPEED\_VALUE\_T is TYPES.REAL\_T; -- range 0.0 .. TYPES.REAL\_T(600.0/3.6); --

-- Distance to reference point in m

type MMU\_COUNTER\_T is delta 0.01 range - 15\_000\_000.0 .. 15\_000\_000.0;

-- Definition of a bounded point coordinate

type MMU\_COORDINATE\_T is

record

NOMINAL\_VALUE : MMU\_COUNTER\_T := 0.0;

UPPER\_VALUE : MMU\_COUNTER\_T := 0.0;

LOWER\_VALUE : MMU\_COUNTER\_T := 0.0;

end record;

EMPTY\_MMU\_COORDINATE\_C : constant MMU\_COORDINATE\_T := (NOMINAL\_VALUE => 0.0,

UPPER\_VALUE => 0.0,

LOWER\_VALUE => 0.0);

-- Definition of a bounded speed value

type MMU\_SPEED\_T is

record

SAFE\_NOMINAL\_VALUE : SPEED\_VALUE\_T;

RAW\_NOMINAL\_VALUE : SPEED\_VALUE\_T;

RAW\_ UPPER\_VALUE : SPEED\_VALUE\_T;

RAW LOWER\_VALUE : SPEED\_VALUE\_T;

end record;

-- Definition of the motion state

type MOTION\_STATE\_T is (NO\_MOTION, MOTION);

-- Definition of the motion direction

type MOTION\_DIRECTION\_T is (UNKNOWN, CAB\_A\_FIRST, CAB\_B\_FIRST);

package ERTMS\_TRAINBORN\_GENERIC\_API is

.

.

.

-------------------------------------------------------------------------------------------------

-- MMU services

-------------------------------------------------------------------------------------------------

-- procedure activated on each cycle to send the current Movement data of the train

procedure WRITE\_MMU\_DATA

( COORDINATE : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.MMU\_COORDINATE\_T;

SPEED : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.MMU\_SPEED\_T;

ACCELERATION : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.ACCELERATION\_VALUE\_T;

MOTION\_STATE : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.MOTION\_STATE\_T;

MOTION\_DIRECTION : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.MOTION\_DIRECTION\_T;

TIME\_STAMP : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.CLOCK\_T);

.

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API;

## Eurobalise interface

### Functional data flows (see /6/)

#### Input

- basic\_to\_generic\_app\_info .EUROBALISE\_input\_info

#### Output

- generic\_app\_to\_basic\_info .EUROBALISE\_output\_info

- generic\_app\_to\_basic\_info .antenna\_to\_be\_activated\_for\_basic

- generic\_app\_to\_basic\_info .BTM\_configuration\_data\_to\_basic

#### Application layer (telegram definition)

DATA message from the eurobalise shall be compliant to /1/.

### Service WRITE\_BTM\_INFO

#### Description

The service is used by the Basic SW to provide the balise messages (BTM) to the application.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| THE\_BTM\_INFO | API\_TYPES.BTM\_INFO\_T | in | The BTM message as received from the track :  - balise telegram (useful data)  - center of balise position (nominal, min, max)  - center of balise position inaccuracy (1m)  - center of balise time-stamp (accuracy=cycle duration)  - Id of antenna used for reception |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

Before the activation of the Application SW, the Basic SW shall call this service once per balise crossed (since the previous cycle), respecting the chronology (older balises first), with the last information received from each balise (see further for details).

This service WRITE\_BTM\_INFO could be called multiple times in one cycle if multiple balises have been crossed during this cycle, so the application should bufferize the received data and achieve processing only at activation of the application.

The balises “short telegrams” (210 useful bits) could be sent to the application as “long telegrams”. In that case, padding bits are added to match the size of the long telegrams (830 bits). Appropriate decoding of the message by the application allows to extract the 210 useful bits.

When crossing a balise, the application shall assume that the Basic SW achieves the following treatment;

* invalid telegrams will be discarded (e.g wrong transmission of telegram in the airgap)
* in case of telegram change , only the last received telegram will be kept for the application. So, when crossing a balise, only one message shall be provided to the application.
* the balise telegram will be ready for the Basic SW only when the train antenna has reached the “end of the balise signal reception” point. As long as the train is stopped over the balise, the telegram will not be provided to the application.

##### 

Balise reception

The position of the center of the balise will be provided in the same coordinate system as the MMU distance calculation. So those data may be compared.

The timestamp of the center of the balise will be provided in the same referential as the WRITE\_TIME service. So those data may be compared. Notice that this timestamp correspond to the application time taken at reception of the balise data on the CORE board which accuracy is not as optimal as the position of the center of the balise.

The “center of balise position inaccuracy” data and the “center of balise time-stamp” data do not include the following inaccuracy sources : drift of the MMU computed distance, antenna positioning error, balise positioning error.

### Service WRITE\_BTM\_ANTENNA

#### Description

The service is used by the Basic SW to indicate to the Application SW the antenna currently used by the EVC

If the service is called with the antenna set to 'NONE', this means that no antenna is selected and that the EVC was not able to find a couple CTE-ANTENNA able to read balise messages

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| THE\_BTM\_ANTENNA | API\_TYPES.ANTENNA\_T | in | The currently selected antenna |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

The knowledge of the selected antenna can be used by the Application SW to compute the exact position of the train head compared to the balise read.

When the service is called with the parameter 'NONE', the EVC shall not be able to read the balise anymore. It can be used to indicate an error message to the driver and to take system actions.

The Basic SW shall call this service the first time at the end of the EVC initialisation and then whenever the selected antenna is updated.

### Service BAD\_BALISE\_RECEIVED

#### Description

If the BTM acquisition device detect a balise with integrity problem (bad CRC, …) the BAD\_BALISE\_RECEIVED service must be called by the Basic SW.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

According to the application specification.

### Service READ\_BTM\_INFO

#### Description

The READ\_BTM\_INFO function returns the type of modulation to use for the BTM antenna and which antenna the Basic Software has to active according to the current active cab (this latter information is relevant when one antenna is installed at each end of the train).

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| The antenna information | ANTENNA\_INFO\_T | out | -Modulation type to be activated : Eurobalise or KER  - Antenna to be activated : antenna\_1 or antenna\_2 |

#### Expected behaviour & usage

This service must be called by the Basic SW after each Activate\_Cycle.

### Service METAL\_MASS\_INFO

#### Description

This service is used by the Basic SW to inhibit its monitoring of antenna failures in areas known as potentially filled with metal masses.

The presence of metal masses may corrupt the result of the antenna tests even if the antenna is valid. This service is thus used to indicate to the BTM that the results of the antenna tests are not valid and that the antenna should be seen as valid even if the test results are wrong.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | API\_TYPES.METAL\_MASS\_INFO\_T | - | The information about the presence of metal masses on the track. |

#### Expected behaviour & usage

Provide the BTM with the state of the track. The information can be provided in two ways :

* Full tolerance : in that case the BTM does not monitor the state of the antenna
* For a given distance : in that case, the BTM shall monitor that the antenna does not have antenna tests invalid during more than the given distance.

The Basic SW shall call this service at multiple moments in the code, even during the management of the input where as it is actually an output of the Application SW.

### SW API extract (ADA Source Code)

-------------------------------------------------------------------------------------------------

-- BTM types

-- type of antenna modulation to use

type MODULATION\_T is (EUROBALISE, KER);

-- definition of the existing antennas

type ANTENNA\_T is (NONE, ANTENNA\_1, ANTENNA\_2);

subtype USED\_ANTENNA\_T is ANTENNA\_T range ANTENNA\_1 .. ANTENNA\_2;

-- Definition of BTM info sent back to the bsw

type ANTENNA\_INFO\_T is

record

MODULATION : MODULATION\_T;

ANTENNA\_ACTIVATED : ANTENNA\_T;

end record;

-- Definition of the BTM telegram and coordinate

type BTM\_INFO\_T is

record

TIME\_STAMP : CLOCK\_T;

USED\_ANTENNA : USED\_ANTENNA\_T;

BALISE\_CENTER\_LOCATION : MMU\_COORDINATE\_T;

BALISE\_CENTER\_DETECTION\_ACCURACY : DISTANCE\_VALUE\_T;

TELEGRAM : INTERFACE\_LANGUAGE\_TYPES.BTM\_TELEGRAM\_T;

end record;

-- Metal mass information

-- IMMUNITY\_DISTANCE : distance to ignored errors caused by big metal masses (NO\_VALUE -- means inifinite)

-- ACTIVATE\_METAL\_MASS\_IMMUNITY : any onboard supervision functions which may be sensitive to -- metal masses shall to be ignored for IMMUNITY\_DISTANCE

-- TRAIN\_IS\_ON\_A\_BIG\_METAL\_MASS : train is inside an announced big metal mass area

type METAL\_MASS\_INFO\_T is

record

IMMUNITY\_DISTANCE : DISTANCE\_T;

ACTIVATE\_METAL\_MASS\_IMMUNITY : TYPES.BOOLEAN\_T;

TRAIN\_IS\_ON\_A\_BIG\_METAL\_MASS : TYPES.BOOLEAN\_T;

end record;.

.

.

package ERTMS\_TRAINBORN\_GENERIC\_API is

.

.

.

-------------------------------------------------------------------------------------------------

-- BTM services

-------------------------------------------------------------------------------------------------

-- function which returns the modulation to use for the BTM antenna and the antenna to be activated

procedure READ\_BTM\_INFO (THE\_ANTENNA\_INFO : out ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.ANTENNA\_INFO\_T);

-- procedure to deliver balises telegrams and coordinate to the application

procedure WRITE\_BTM\_INFO (THE\_BTM\_INFO : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.BTM\_INFO\_T);

-- procedure to deliver the current active BTM ntenna to the application

procedure WRITE\_BTM\_ANTENNA (THE\_BTM\_ANTENNA : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.ANTENNA\_T);

-- function which returns metal mass information to the basic software to know how to manage the BTM antenna

function METAL\_MASS\_INFO return ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.METAL\_MASS\_INFO\_T;

-- basic software has detected a balise with integrity problem (CRC...)

procedure BAD\_BALISE\_RECEIVED;

.

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API;

## Euroloop interface

### Functional data flows (see /6/)

#### Input

- basic\_to\_generic\_app\_info .EUROLOOP\_input\_info

#### Output

- generic\_app\_to\_basic\_info .EUROLOOP\_output\_info

#### Application layer (telegram definition)

DATA message from the euroloop shall be compliant to /1/.

### Service WRITE\_LOOP\_MESSAGE

#### Description

The Service is used by the Basic SW to provide the received Euroloop message(s) from the trackside.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| THE\_LTM\_INFO | API\_TYPES.LOOP\_MESSAGE\_T | in | - Loop telegram (useful data)  - “SAME” indication : when the telegram is the same as the previous one, “SAME” indication is provided to the application SW |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

This service shall be called by the Basic for each message received from the trackside. This service could be called several times during the same cycle but always before the ACTIVATE\_CYCLE service.

### Service SS\_CODE\_FOR\_LOOP

#### Description

This Service shall be used by the Basic SW to know which spread spectrum code has to be used by the LTM module.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| THE\_SS\_CODE\_FOR\_LOOP | API\_TYPES.SS\_CODE\_FOR\_LOOP\_T | out | the spread spectrum code to be used by the LTM |

#### Expected behaviour & usage

This service shall be called by the Basic SW once at each cycle after the ACTIVATE\_CYCLE service.

### SW API extract (ADA Source Code)

package ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES is

.

.

.

-------------------------------------------------------------------------------------------------

-- Loop types

-------------------------------------------------------------------------------------------------

-- Spread Spectrum Code required to receive messages from a specific loop installation.

-- 15 : reserved value when no loop is expected

type SS\_CODE\_FOR\_LOOP\_T is range 0 .. 15;

type CONTINUOUS\_DATA\_KIND\_T is (DATA, SAME);

type LOOP\_MESSAGE\_T (KIND : CONTINUOUS\_DATA\_KIND\_T := DATA) is

record

TIME\_STAMP : CLOCK\_T;

case KIND is

when DATA =>

MESSAGE : INTERFACE\_LANGUAGE\_TYPES.LOOP\_MESSAGE\_T;

when SAME => -- same as previously DATA message received

null;

end case;

end record; .

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES;

package ERTMS\_TRAINBORN\_GENERIC\_API is

.

.

.

-------------------------------------------------------------------------------------------------

-- LOOP services

-------------------------------------------------------------------------------------------------

-- function which return the spread spectrum code of the expected loop

function SS\_CODE\_FOR\_LOOP return ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.SS\_CODE\_FOR\_LOOP\_T;

-- procedure to deliver loop message to the application

procedure WRITE\_LOOP\_MESSAGE (THE\_LOOP\_MESSAGE : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.LOOP\_MESSAGE\_T);

.

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API;

## Train Interface Unit (TIU)

### Functional data flows (see /6/)

#### Input

- basic\_to\_generic\_app\_info .TIU\_input\_info

#### Output

- generic\_app\_to\_basic\_info .TIU\_output\_msgs\_info

- generic\_app\_to\_basic\_info .isolation\_from\_other\_equipment\_is\_required

- generic\_app\_to\_basic\_info .cab\_status\_for\_basic

- generic\_app\_to\_basic\_info .EB\_intervention\_requested

#### Application layer (telegram definition)

Please refer to /5/.

### Service WRITE\_TIU\_MESSAGE

#### Description

The service is used by the Basic SW to provide the messages received from the TIU to the Application SW.

This service shall be compliant to the general requirements about message lists usage, refer to section 4.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| THE\_TIU\_MESSAGE | TIU\_LANGUAGE\_TYPES.TIU\_TO\_CORE\_BIT\_STREAM\_T | in | The message from the TIU |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

After initialisation, the Basic SW shall call this service on reception of each telegram data from the TIU.

This service shall potentially be called several times during the cycle, but always before the ACTIVATE\_CYCLE service.

The Basic SW is in charge of maintaining the connection with the TIU. In case of disconnection, the Basic SW shall disable the system.

### Service READ\_TIU\_MESSAGE

#### Description

The service is used by the Basic SW to get the messages from the Application SW that must be sent to the TIU.

This service shall be compliant to the general requirements about message lists usage, refer to section 4.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| THE\_TIU\_MESSAGE | TIU\_LANGUAGE\_TYPES.CORE\_TO\_TIU\_BIT\_STREAM\_T | out | The message to the TIU |

#### Expected behaviour & usage

The Basic SW calls this service only when the service TIU\_MESSAGE\_QUEUE\_IS\_EMPTY returns FALSE.

After having called the service, the Basic SW shall try again a call to the TIU\_MESSAGE\_QUEUE\_IS\_EMPTY to see if other messages must be sent to the TIU and read them if existing.

The Basic SW is in charge of maintaining the connection with the TIU. In case of disconnection, the Basic SW shall disable the system.

### Service TIU\_MESSAGE\_QUEUE\_IS\_EMPTY

#### Description

The service is used by the Basic SW to know if messages are to be output to the TIU.

This service shall be compliant to the general requirements about message lists usage, refer to section 4.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | TYPES.BOOLEAN\_T | out | Indicate if messages to the TIU are present in the Application SW output buffers. |

#### Expected behaviour & usage

Indicate to the Basic SW whether the output FIFO to the TIU is empty or not.

The Basic SW calls this service to know if it can read messages to the TIU. As long as this service returns FALSE, the Basic SW shall read messages to the TIU and dispatch them.

### Service EB\_REQUESTED

#### Description

This service is called by the Basic SW in order to know if the Emergency Brake application is requested by the application.

A similar service exit in the TIU interface (see §4.10.3). Both services have to be used by the application.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | TYPES.BOOLEAN\_T | out | returns TRUE if the EB is requested by the application  returns FALSE if the EB is not requested by the application |

#### Expected behaviour & usage

This service shall be called by the Basic SW once per cycle after the ACTIVATE\_CYCLE service.

The safe application of the Emergency Braking is insured by the Basic SW.

### Service EVC\_ISOLATION\_IS\_REQUESTED

#### Description

This service is called by the Basic SW in order to know if the EVC has to be isolated.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | TYPES.BOOLEAN\_T | out | returns TRUE if the EVC has to be isolated |

#### Expected behaviour & usage

This service shall be called by the Basic SW once per cycle after the ACTIVATE\_CYCLE service.

### Service OCCUPIED\_CABIN

#### Description

This service is called by the Basic SW in order to know which cabin is occupied.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| THE\_OCCCUPIED\_CABIN | API\_TYPES.OCCUPIED\_CABIN\_T | out | CAB\_A, CAB\_B, none |

#### Expected behaviour & usage

This service shall be called by the Basic SW once per cycle after the ACTIVATE\_CYCLE service.

### SW API extract (ADA Source Code)

-------------------------------------------------------------------------------------------------

-- TIU types

-------------------------------------------------------------------------------------------------

-- Possible state of the cabine occupation

type OCCUPIED\_CABINE\_T is (CAB\_A, CAB\_B, NONE);

package ERTMS\_TRAINBORN\_GENERIC\_API is

.

.

.

-------------------------------------------------------------------------------------------------

-- TIU services

-------------------------------------------------------------------------------------------------

-- procedure to deliver TIU message to the application

procedure WRITE\_TIU\_MESSAGE

(THE\_TIU\_MESSAGE : in INTERFACE\_LANGUAGE\_TYPES.TIU\_TO\_CORE\_BIT\_STREAM\_T);

-- procedure to read TIU message from the application

-- this procedure may be called only if TIU\_MESSAGE\_QUEUE\_IS\_EMPTY returns FALSE

procedure READ\_TIU\_MESSAGE

(THE\_TIU\_MESSAGE : out INTERFACE\_LANGUAGE\_TYPES.CORE\_TO\_TIU\_BIT\_STREAM\_T);

-- this function returns TRUE if the output TIU MESSAGE QUEUE IS EMPTY

-- and returns FALSE otherwise

function TIU\_MESSAGE\_QUEUE\_IS\_EMPTY return TYPES.BOOLEAN\_T;

-- this function returns TRUE if the EVC must be isolated from the train

-- and returns FALSE otherwise

function EVC\_ISOLATION\_IS\_REQUESTED return TYPES.BOOLEAN\_T;

-- this function returns TRUE if the EVC must be isolated from the train

-- and returns FALSE otherwise

function OCCUPIED\_CABINE return ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.OCCUPIED\_CABINE\_T;

.

.

-------------------------------------------------------------------------------------------------

-- Brakes services

-------------------------------------------------------------------------------------------------

-- function which TRUE if the emergency brakes are requested by the application

function EB\_REQUESTED return TYPES.BOOLEAN\_T;

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API;

## Driver Machine Interface (DMI)

### Functional data flows (see /6/)

#### Input

- basic\_to\_generic\_app\_info .EUROCAB\_input\_info

#### Output

- generic\_app\_to\_basic\_info .EUROCAB\_output\_info

#### Application layer (telegram definition)

Please refer to /5/.

### Service WRITE\_MMI\_MESSAGE

#### Description

The service is used by the Basic SW to provide the messages received from the DMI to the Application SW.

This service shall be compliant to the general requirements about message lists usage, refer to section 4.1.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| THE\_MMI\_MESSAGE | API\_TYPES.MMI\_IN\_MESSAGE\_T | in | The message from the DMI |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

The messages from the DMI can be of multiple kinds :

* DATA : a data message is received from the DMI. These messages can only be received after the link is connected and before the link is disconnected/Temporary\_disconnected.
* CONNECTED : the link with the peer is connected.
* TEMPORARY\_DISCONNECTED : the link with the DMI is no more connected but may be connected again in the future.
* DISCONNECTED : the link is no more connected and shall never be connected again.

The Basic SW calls this service, on reception of data from the DMI or when the state of the connection with the DMI is updated. In any cases, if it is called, it shall be done before the activation of the Application SW.

A cabin ID is always associated with a MMI message.

### Service READ\_MMI\_MESSAGE

#### Description

The service is used by the Basic SW to get the messages from the Application SW that must be sent to the DMI.

This service shall be compliant to the general requirements about message lists usage, refer to section 4.1.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| THE\_MMI\_MESSAGE | API\_TYPES.MMI\_OUT\_MESSAGE\_T | out | The message to the DMI |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

The messages to the DMI can be of multiple kinds :

* DATA : a data message to send.
* DISCONNECTION\_REQUEST : a request to kill the connection with the DMI. When received, the connection shall be closed and a confirmation shall be given to the Application SW. In that case the link shall never be opened again.

The Basic SW calls this service only when the service MMI\_MESSAGE\_QUEUE\_IS\_EMPTY returns FALSE.

After having called the service, the Basic SW shall try again a call to the MMI\_MESSAGE\_QUEUE\_IS\_EMPTY to see if other messages must be sent to the DMI and read them if existing.

A cabin ID is always associated with a MMI message.

### Service MMI\_MESSAGE\_QUEUE\_IS\_EMPTY

#### Description

The service is used by the Basic SW to know if messages are to be output to the DMI.

This service shall be compliant to the general requirements about message lists usage, refer to section 4.1.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | TYPES.BOOLEAN\_T | - | Indicate if messages to the DMI are present in the Application SW output buffers. |

#### Expected behaviour & usage

Indicate to the Basic SW if the output fifo to the DMI is empty or not.

The Basic SW calls this service to know if it can read messages to the DMI. As long as this service returns FALSE, the Basic SW shall read messages to the DMI and dispatch them.

### SW API extract (ADA Source Code)

-------------------------------------------------------------------------------------------------

-- MMI types

-------------------------------------------------------------------------------------------------

subtype CABINE\_T is OCCUPIED\_CABINE\_T range CAB\_A .. CAB\_B;

type MMI\_IN\_MESSAGE\_T

(KIND : INTERFACE\_LANGUAGE\_TYPES.IN\_MESSAGE\_KIND\_T

:= INTERFACE\_LANGUAGE\_TYPES.IN\_MESSAGE\_KIND\_T'FIRST) is

record

ORIGIN : CABINE\_T;

case KIND is

when INTERFACE\_LANGUAGE\_TYPES.DATA =>

MESSAGE : INTERFACE\_LANGUAGE\_TYPES.MMI\_TO\_CORE\_BIT\_STREAM\_T;

when INTERFACE\_LANGUAGE\_TYPES.CONNECTED =>

null;

when INTERFACE\_LANGUAGE\_TYPES.TEMPORARY\_DISCONNECTED =>

null;

when INTERFACE\_LANGUAGE\_TYPES.DISCONNECTED =>

null;

end case;

end record;

type MMI\_OUT\_MESSAGE\_T

(KIND : INTERFACE\_LANGUAGE\_TYPES.OUT\_MESSAGE\_KIND\_T

:= INTERFACE\_LANGUAGE\_TYPES.OUT\_MESSAGE\_KIND\_T'FIRST) is

record

DESTINATION : CABINE\_T;

case KIND is

when INTERFACE\_LANGUAGE\_TYPES.DATA =>

MESSAGE : INTERFACE\_LANGUAGE\_TYPES.CORE\_TO\_MMI\_BIT\_STREAM\_T;

when INTERFACE\_LANGUAGE\_TYPES.DISCONNECTION\_REQUEST =>

null;

end case;

end record;

package ERTMS\_TRAINBORN\_GENERIC\_API is

.

.

.

-------------------------------------------------------------------------------------------------

-- MMI services

-------------------------------------------------------------------------------------------------

-- procedure to deliver MMI message to the application

procedure WRITE\_MMI\_MESSAGE

(THE\_MMI\_MESSAGE : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.MMI\_IN\_MESSAGE\_T);

-- procedure to read MMI message from the application

-- this procedure may be called only if MMI\_MESSAGE\_QUEUE\_IS\_EMPTY returns FALSE

procedure READ\_MMI\_MESSAGE

(THE\_MMI\_MESSAGE : out ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.MMI\_OUT\_MESSAGE\_T);

-- this function returns TRUE if the output MMI MESSAGE QUEUE IS EMPTY

-- and returns FALSE otherwise

function MMI\_MESSAGE\_QUEUE\_IS\_EMPTY return TYPES.BOOLEAN\_T;

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API;

## JRU Interface

### Functional data flows (see /6/)

#### Input

- basic\_to\_generic\_app\_info .EUROCAB\_input\_info

#### Output

- generic\_app\_to\_basic\_info .EUROCAB\_output\_info

#### Application layer (telegram definition)

Please refer to /5/. (DATA message to JRU shall be compliant to /3/)

### Service WRITE\_JRU\_MESSAGE

#### Description

The service is used by the Basic SW to provide the messages received from the JRU to the Application SW.

This service shall be compliant to the general requirements about message lists usage, refer to section 4.1.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| THE\_JRU\_MESSAGE | API\_TYPES.JRU\_IN\_MESSAGE\_T | in | The message from the JRU |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

The messages from the JRU can be of multiple kinds :

* DATA : a data message is received from the JRU. These messages can only be received after the link is connected and before the link is disconnected/Temporary\_disconnected.
* CONNECTED : the link with the peer is connected.
* TEMPORARY\_DISCONNECTED : the link with the JRU is no more connected but may be connected again in the future.
* DISCONNECTED : the link is no more connected and shall never be connected again.

The Basic SW calls this service, on reception of data from the JRU or when the state of the connection with the JRU is updated. In any cases, if it is called, it shall be done before the activation of the Application SW.

### Service READ\_JRU\_MESSAGE

#### Description

The service is used by the Basic SW to get the messages from the Application SW that must be sent to the JRU.

This service shall be compliant to the general requirements about message lists usage, refer to section 4.1.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| THE\_JRU\_MESSAGE | API\_TYPES.JRU\_OUT\_MESSAGE\_T | out | The message to the JRU |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

The messages to the JRU can be of multiple kinds :

* DATA : a data message to send.
* DISCONNECTION\_REQUEST : a request to kill the connection with the JRU. When received, the connection shall be closed and a confirmation shall be given to the Application SW. In that case the link shall never be opened again.

The Basic SW calls this service only when the service JRU\_MESSAGE\_QUEUE\_IS\_EMPTY returns FALSE.

After having called the service, the Basic SW shall try again a call to the JRU\_MESSAGE\_QUEUE\_IS\_EMPTY to see if other messages must be sent to the JRU and read them if existing.

### Service JRU\_MESSAGE\_QUEUE\_IS\_EMPTY

#### Description

The service is used by the Basic SW to know if messages are to be output to the JRU.

This service shall be compliant to the general requirements about message lists usage, refer to section 4.1.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | TYPES.BOOLEAN\_T | - | Indicate if messages to the JRU are present in the Application SW output buffers. |

#### Expected behaviour & usage

Indicate to the Basic SW if the output fifo to the JRU is empty or not.

The Basic SW calls this service to know if it can read messages to the JRU. As long as this service returns FALSE, the Basic SW shall read messages to the JRU and dispatch them.

### SW API extract (ADA Source Code)

-------------------------------------------------------------------------------------------------

-- JRU types

-------------------------------------------------------------------------------------------------

type JRU\_IN\_MESSAGE\_T (KIND : INTERFACE\_LANGUAGE\_TYPES.IN\_MESSAGE\_KIND\_T := INTERFACE\_LANGUAGE\_TYPES.IN\_MESSAGE\_KIND\_T'FIRST) is

record

case KIND is

when INTERFACE\_LANGUAGE\_TYPES.DATA =>

MESSAGE : INTERFACE\_LANGUAGE\_TYPES.JRU\_TO\_CORE\_BIT\_STREAM\_T;

when INTERFACE\_LANGUAGE\_TYPES.CONNECTED =>

null;

when INTERFACE\_LANGUAGE\_TYPES.TEMPORARY\_DISCONNECTED =>

null;

when INTERFACE\_LANGUAGE\_TYPES.DISCONNECTED =>

null;

end case;

end record;

type JRU\_OUT\_MESSAGE\_T (KIND : INTERFACE\_LANGUAGE\_TYPES.OUT\_MESSAGE\_KIND\_T := INTERFACE\_LANGUAGE\_TYPES.OUT\_MESSAGE\_KIND\_T'FIRST) is

record

case KIND is

when INTERFACE\_LANGUAGE\_TYPES.DATA =>

MESSAGE : INTERFACE\_LANGUAGE\_TYPES.CORE\_TO\_JRU\_BIT\_STREAM\_T;

when INTERFACE\_LANGUAGE\_TYPES.DISCONNECTION\_REQUEST =>

null;

end case;

end record;

.

.

.

package ERTMS\_TRAINBORN\_GENERIC\_API is

.

.

.

-------------------------------------------------------------------------------------------------

-- JRU services

-------------------------------------------------------------------------------------------------

-- procedure to deliver JRU message to the application

procedure WRITE\_JRU\_MESSAGE

(THE\_JRU\_MESSAGE : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.JRU\_IN\_MESSAGE\_T);

-- procedure to read JRU message from the application

-- this procedure may be called only if JRU\_MESSAGE\_QUEUE\_IS\_EMPTY returns FALSE

procedure READ\_JRU\_MESSAGE

(THE\_JRU\_MESSAGE : out ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.JRU\_OUT\_MESSAGE\_T);

-- this function returns TRUE if the output JRU MESSAGE QUEUE IS EMPTY

-- and returns FALSE otherwise

function JRU\_MESSAGE\_QUEUE\_IS\_EMPTY return TYPES.BOOLEAN\_T;

.

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API;

## STM Interface

### Functional data flows (see /6/)

#### Input

- basic\_to\_generic\_app\_info .EUROCAB\_input\_info

#### Output

- generic\_app\_to\_basic\_info .EUROCAB\_output\_info

#### Application layer (telegram definition)

DATA message to/from STM shall be compliant to /4/.

### Service WRITE\_STM\_MESSAGE

#### Description

The service is used by the Basic SW to provide the messages received from the STM to the Application SW.

This service shall be compliant to the general requirements about message lists usage, refer to section 4.1.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| THE\_STM\_MESSAGE | API\_TYPES.STM\_IN\_MESSAGE\_T | in | The message from the STM |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

The messages from the STM can be of multiple kinds :

* DATA : a data message is received from the STM. These messages can only be received after the link is connected and before the link is disconnected/Temporary\_disconnected.
* CONNECTED : the link with the peer is connected.
* TEMPORARY\_DISCONNECTED : the link with the STM is no more connected but may be connected again in the future.
* DISCONNECTED : the link is no more connected and shall never be connected again.

The Basic SW calls this service, on reception of data from the STM or when the state of the connection with the STM is updated. In any cases, if it is called, it shall be done before the activation of the Application SW.

### Service READ\_STM\_MESSAGE

#### Description

The service is used by the Basic SW to get the messages from the Application SW that must be sent to the STM.

This service shall be compliant to the general requirements about message lists usage, refer to section 4.1.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| THE\_STM\_MESSAGE | API\_TYPES.STM\_OUT\_MESSAGE\_T | out | The message to the STM |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

The messages to the STM can be of multiple kinds :

* DATA : a data message to send.
* DISCONNECTION\_REQUEST : a request to kill the connection with the STM. When received, the connection shall be closed and a confirmation shall be given to the Application SW. In that case the link shall never be opened again.

The Basic SW calls this service only when the service STM\_MESSAGE\_QUEUE\_IS\_EMPTY returns FALSE.

After having called the service, the Basic SW shall try again a call to the STM\_MESSAGE\_QUEUE\_IS\_EMPTY to see if other messages must be sent to the STM and read them if existing.

### Service STM\_MESSAGE\_QUEUE\_IS\_EMPTY

#### Description

The service is used by the Basic SW to know if messages are to be output to the STM.

This service shall be compliant to the general requirements about message lists usage, refer to section 4.1.1.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | TYPES.BOOLEAN\_T | - | Indicate if messages to the STM are present in the Application SW output buffers. |

#### Expected behaviour & usage

Indicate to the Basic SW if the output fifo to the STM is empty or not.

The Basic SW calls this service to know if it can read messages to the STM. As long as this service returns FALSE, the Basic SW shall read messages to the STM and dispatch them.

### Service STM\_INFO

#### Description

This function returns to the Basic SW the list of connected STMs (from an application point of view) with their associated state. This service is specific to Alstom use.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| list of STMs with their state | API\_TYPES.STM\_INFO\_T | out | - NID\_STM  -Mode: Power On, Configuration, Data entry, cold Standby, Hot standby, Data available, Failure |

#### Expected behaviour & usage

This service may be called by the Basic SW at each cycle after the ACTIVATE\_CYCLE service.

### SW API extract (ADA Source Code)

MAX\_NBR\_OF\_STMS\_C : constant := 12;

type STM\_IDENTITY\_T is range 0 .. 255;

type STM\_STATE\_T is -- NID\_STMSTATE

(POWER\_ON,

CONFIGURATION,

DATA\_ENTRY,

COLD\_STANDBY,

HOT\_STANDBY,

DATA\_AVAILABLE,

FAILURE);

type STM\_INFO\_T is

record

ID : STM\_IDENTITY\_T := STM\_IDENTITY\_T'LAST; -- NID\_STM

STATE : STM\_STATE\_T := STM\_STATE\_T'FIRST; -- NID\_STMSTATE

end record;

type STM\_INFO\_LIST\_LENGTH\_T is range 0 .. MAX\_NBR\_OF\_STMS\_C;

type STM\_INFO\_MAP\_T is array (STM\_INFO\_LIST\_LENGTH\_T range <>) of STM\_INFO\_T;

type STM\_INFO\_LIST\_T (LENGTH : STM\_INFO\_LIST\_LENGTH\_T := STM\_INFO\_LIST\_LENGTH\_T'FIRST) is

record

LIST : STM\_INFO\_MAP\_T (1 .. LENGTH);

end record;

type STM\_CONTROL\_IN\_MESSAGE\_T (KIND : INTERFACE\_LANGUAGE\_TYPES.IN\_MESSAGE\_KIND\_T := INTERFACE\_LANGUAGE\_TYPES.IN\_MESSAGE\_KIND\_T'FIRST) is

record

CONNECTION\_ID : INTERFACE\_LANGUAGE\_TYPES.STM\_CONNECTION\_IDENTITY\_T;

case KIND is

when INTERFACE\_LANGUAGE\_TYPES.DATA =>

MESSAGE : INTERFACE\_LANGUAGE\_TYPES.STM\_TO\_STM\_CONTROL\_BIT\_STREAM\_T;

when INTERFACE\_LANGUAGE\_TYPES.CONNECTED =>

null;

when INTERFACE\_LANGUAGE\_TYPES.TEMPORARY\_DISCONNECTED =>

null;

when INTERFACE\_LANGUAGE\_TYPES.DISCONNECTED =>

null;

end case;

end record;

type STM\_CONTROL\_OUT\_MESSAGE\_T (KIND : INTERFACE\_LANGUAGE\_TYPES.OUT\_MESSAGE\_KIND\_T := INTERFACE\_LANGUAGE\_TYPES.OUT\_MESSAGE\_KIND\_T'FIRST) is

record

CONNECTION\_ID : INTERFACE\_LANGUAGE\_TYPES.STM\_CONNECTION\_IDENTITY\_T;

case KIND is

when INTERFACE\_LANGUAGE\_TYPES.DATA =>

MESSAGE : INTERFACE\_LANGUAGE\_TYPES.STM\_CONTROL\_TO\_STM\_BIT\_STREAM\_T;

when INTERFACE\_LANGUAGE\_TYPES.DISCONNECTION\_REQUEST =>

null;

end case;

end record;

-------------------------------------------------------------------------------------------------

-- STM specific types

-------------------------------------------------------------------------------------------------

type STM\_SPECIFIC\_IN\_MESSAGE\_T (KIND : INTERFACE\_LANGUAGE\_TYPES.IN\_MESSAGE\_KIND\_T := INTERFACE\_LANGUAGE\_TYPES.IN\_MESSAGE\_KIND\_T'FIRST) is

record

CONNECTION\_ID : INTERFACE\_LANGUAGE\_TYPES.STM\_CONNECTION\_IDENTITY\_T;

case KIND is

when INTERFACE\_LANGUAGE\_TYPES.DATA =>

MESSAGE : INTERFACE\_LANGUAGE\_TYPES.STM\_TO\_EVC\_SPECIFIC\_BIT\_STREAM\_T;

when INTERFACE\_LANGUAGE\_TYPES.CONNECTED =>

null;

when INTERFACE\_LANGUAGE\_TYPES.TEMPORARY\_DISCONNECTED =>

null;

when INTERFACE\_LANGUAGE\_TYPES.DISCONNECTED =>

null;

end case;

end record;

type STM\_SPECIFIC\_OUT\_MESSAGE\_T (KIND : INTERFACE\_LANGUAGE\_TYPES.OUT\_MESSAGE\_KIND\_T := INTERFACE\_LANGUAGE\_TYPES.OUT\_MESSAGE\_KIND\_T'FIRST) is

record

CONNECTION\_ID : INTERFACE\_LANGUAGE\_TYPES.STM\_CONNECTION\_IDENTITY\_T;

case KIND is

when INTERFACE\_LANGUAGE\_TYPES.DATA =>

MESSAGE : INTERFACE\_LANGUAGE\_TYPES.EVC\_SPECIFIC\_TO\_STM\_BIT\_STREAM\_T;

when INTERFACE\_LANGUAGE\_TYPES.DISCONNECTION\_REQUEST =>

null;

end case;

end record;

package ERTMS\_TRAINBORN\_GENERIC\_API is

.

.

.

-------------------------------------------------------------------------------------------------

-- STM services

-------------------------------------------------------------------------------------------------

-- function which returns the information (STM ID, STM state) on connected (from an application

point of view) STMs

function STM\_INFO return ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.STM\_INFO\_LIST\_T;

-- procedure to deliver STM message to the application

procedure WRITE\_STM\_CONTROL\_MESSAGE

(THE\_STM\_MESSAGE : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.STM\_CONTROL\_IN\_MESSAGE\_T);

-- procedure to read STM message from the application

-- this procedure may be called only if STM\_CONTROL\_MESSAGE\_QUEUE\_IS\_EMPTY returns FALSE

procedure READ\_STM\_CONTROL\_MESSAGE

(THE\_STM\_MESSAGE : out ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.STM\_CONTROL\_OUT\_MESSAGE\_T);

-- this function returns TRUE if the output STM CONTROL MESSAGE QUEUE IS EMPTY

-- and returns FALSE otherwise

function STM\_CONTROL\_MESSAGE\_QUEUE\_IS\_EMPTY return TYPES.BOOLEAN\_T;

.

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API;

## Euroradio interface

### Functional data flows (see /6/)

#### Input

- basic\_to\_generic\_app\_info .EURORADIO\_input\_info

#### Output

- generic\_app\_to\_basic\_info .EURORADIO\_output\_info

#### Application layer (telegram definition)

DATA message to/from RBC shall be compliant to /1/.

Remark : The Euroradio API has been updated in present document for baseline 3 purpose and now also includes GPRS features. **Those GPRS features are not applicable and are not to be used in the frame of Open ETCS.**

### Service NUMBER\_OF\_HANDABLE\_RTM\_COMMUNICATION\_SESSION

#### Description

This service will provide to the application the amount of communication sessions that are possible simultaneously. It represents actually the amount of Mobile Terminals that are available.

This value is dynamic according to the hardware equipment status but once it reaches 0 it never increases anymore until the next EVC power-up.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | API\_TYPES. RTM\_COMMUNICATION\_SESSION\_NBR\_T | out | 0,1 or 2 |

#### Expected behaviour & usage

The Basic SW shall call this service at each cycle (before the call to ACTIVATE\_CYCLE). Notice that the selftests of the Mobile Terminals will be finished 80s after the power-up of the EVC equipment.

### Service WRITE\_MOBILE\_CONTEXT

#### Description

This service will provide to the application the context of each mobile terminal and their associated network ID.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| THE\_CONTEXT | API\_TYPES.MOBILE\_TABLE\_T | in | For Mobile\_1 and Mobile\_2 :  - the associated network-id  - the Mobile “health” status  - the Mobile network registration status  - the Mobile communication status |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & Usage

The mobile “health” status may be:

* OK : mobile is operational
* NOK : mobile is out of order

The mobile network registration status may be:

* NETWORK\_NOT\_REGISTERED : no network registration is on-going
* NETWORK\_REGISTER\_REQUEST: mobile with a network registration request in progress
* NETWORK\_REGISTER\_CONFIRM : mobile is registered to the network

The mobile communication status may be :

* Mobile free
* Mobile in safe com : a safe connection (ETCS) is on-going via the mobile. More precisely, it means that a e.g. ION

More explanation of the expected behaviour is to be found at section 4.14.8.

The Basic SW shall call this service at the first cycle and then at each change (at most once per cycle, before the call to ACTIVATE\_CYCLE).

More explanation of the usage is to be found at section 4.14.8.

### Service READ\_RTM\_MESSAGE

#### Description

This service is to be used by the Basic SW to get the messages (data and control) from the Application SW that must be sent to the Radio sub-system.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
|  |  |  |  |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| RADIO\_DEVICE | API\_TYPES.RADIO\_DEVICE\_T | in | ETCS\_id of the destination radio device |
| THE\_RTM\_MESSAGE | API\_TYPES.RTM\_OUT\_MESSAGE\_T | out | radio sub-system messages to be sent |

#### Expected behaviour & usage

The messages to be sent to the radio sub-system may be of several kinds:

* NETWORK\_REQUEST : order to register the mobiles to a network. In such case, the network-id is provided by the application.
* CONNECTION\_REQUEST : order to connect. In such case, the NID\_RADIO (phone number) is provided by the application and the RADIO\_PRIORITY\_LEVEL (HIGH or LOW) aswell.
* CHANGE\_OF\_PRIORITY\_LEVEL : change the priority level (HIGH / LOW) of the radio connection.
* DISCONNECTION\_REQUEST : order to disconnect. In such case, the application provides a diagnostic code.
* RESET\_CONNECTION : order to disconnect and then reconnect In such case, the application provides a diagnostic code.
* INFINITE\_CONNECTION\_ATTEMPTS : order to try to reconnect infinitely after a connection loss or a reset connection.
* DATA : euroradio data message from trainborne. In such case, the application provides the radio data message which is a bit stream (maximum length = 8\*500 bits)

More explanation of the expected behaviour is to be found at section 4.14.8.

The Basic SW may call this service only when the service RTM\_MESSAGE\_QUEUE\_IS\_EMPTY returns FALSE.

After having called the service, the Basic SW shall try again a call to the RTM\_MESSAGE\_QUEUE\_IS\_EMPTY to see if other messages must be sent and read them if existing.

More explanation of the usage is to be found at section 4.14.8.

### Service RTM\_MESSAGE\_QUEUE\_IS\_EMPTY

#### Description

The service is used by Basic SW to know if messages are to be output to the Radio sub-system.

#### Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Return value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | TYPES.BOOLEAN\_T | out | Indicate if messages to the Radio sub-system are present in the application output buffers. |

#### Expected behaviour & usage

Indicate to the Basic SW whether the output FIFO to the radio sub-system is empty or not.

The Basic SW calls this service to know if it can read messages to the radio sub-system. As long as this service returns FALSE, the Basic SW shall read messages and treat them.

### Service WRITE\_RTM\_MESSAGE

#### Description

This service is used by the Basic SW to provide the received radio sub-system messages (data and command) to the Application SW.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| RADIO\_DEVICE | RADIO\_DEVICE\_T | in | ETCS\_id of the source radio device |
| THE\_RTM\_MESSAGE | API\_TYPES.RTM\_IN\_MESSAGE\_T | in | received radio sub-system message |

Other parameters are depending on the message kind.

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

The received messages from the radio sub-system may be of several kinds:

* CONNECTION\_CONFIRMATION : provided to the application at every connection or reconnection
* CONNECTION\_LOST : provided when the connection is lost (if it was previously established). In such case the origin of the failure is provided by the Basic SW to the application.
* CONNECTION\_FAILURE : provided when it has not been possible to (re)establish the connection after 3 attempts (if re-connection attenpts is not infinite). In such case the origin of the failure is provided by the Basic SW to the application .
* CONNECTION\_NOT\_ESTABLISHED : provided when it has not been possible to re-establish the connection after 3 attempts (if re-connection attenpts is infinite) In such case the origin of the failure is provided by the Basic SW to the application.
* DATA : euroradio data message from trackside. In such case, the Basic SW provides the radio data message which is a bit stream (maximum length = 8\*500 bits)

The Basic SW calls this service on reception of data from the radio sub-system or when the state of the connection with the radio device is updated. In any cases, if it is called, it shall be done before the activation of the Application SW.

More explanation of the usage is to be found at section 4.14.8.

### Service TRAIN\_IS\_IN\_A\_RADIO\_HOLE

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | boolean | out | train is currently in a radio hole |

#### Expected behaviour & usage

The service TRAIN\_IS\_IN\_A\_RADIO\_HOLE is used by the Basic Software to know if a disconnection is expected or not. When the train is in a radio hole and the connection is lost, the basic software has to wait the end of the radio hole before to try to restore it.

### Complementary expected behaviour and usage of WRITE\_MOBILE\_CONTEXT, READ\_RTM\_MESSAGE and WRITE\_RTM MESSAGE

#### Network registration

The Basic SW shall :

* At start-up, send the network registration command to the radio sub-system for both Mobile Terminals with the last used network-id (to be stored/read in Novram by the Basic SW; only the last requested network-id is memorised.) or else a default network-id stored in data prep of the Basic SW
* In normal working, send the network registration commands to the radio sub-system with the network-id from the application; and deliver to the application a status of the network registration.

The application will have the possibility to (re-)enter the network-id (send the network registration with new network-id to the Basic SW) during the mission if needed.

#### Radio session management

The radio session is always initiated by the Trainborne sub-system.

The application shall not try to establish a radio connection in any other state than NETWORK\_REGISTER\_CONFIRM.

Only one communication may be established with a given radio device.

If the Basic SW has established the connection with the required RADIO\_DEVICE a CONNECTION\_CONFIRMATION is returned to the application. Otherwise a CONNECTION\_FAILURE or a CONNECTION\_NOT\_ESTABLISHED is returned.

As soon as an established connection is seen as broken by the Basic SW, it shall send a CONNECTION\_LOST.

An INFINITE\_CONNECTION\_ATTENPTS message shall be sent by the application to indicate the expected behaviour of the Basic SW: finite attempts configuration (by default) or infinite attenpts configuration.

By default (“infinite connection attempts” is not requested by the application), when a radio connection link is broken, the Basic SW shall:

* indicate a CONNECTION\_LOST event to the ASW;
* try to re-connect at most 3 times;
* in case the radio connection could not be re-established after the 3 trials, then the BSW shall stop the connection re-tries and declare a CONNECTION\_FAILURE (with the origin of the problem) to the ASW.

If “infinite connection attempts” is requested by the application for a given radio session; when the given radio connection link is broken, the Basic SW shall:

* indicate the CONNECTION\_LOST event to the application;
* try to re-connect periodically with unlimited number of trials;
* in case the radio connection is not re-established after 3 trials, then the Basic SW shall indicate a “CONNECTION\_NOT\_ESTABLISHED event to the application (with the origin of the problem).

When the application requests a radio disconnection, then the Basic SW shall stop the connection re-tries in all cases.

When a radio connection has been opened, if it seems to be lost at application level (no application message received during a given time) : a RESET\_CONNECTION may be sent to the Basic SW by the application. The goal is to “refresh” the connection by disconnecting and reconnecting the RADIO\_DEVICE.

DATA is used to send and receive messages to/from the specified RADIO\_DEVICE.

If a READ procedure is used with a RADIO\_DEVICE which do not correspond to an established and usable connection (usable connection meaning : the connection is established and the Basic SW is not trying to reconnect it), the Basic SW may delete the message.

So there is no obligation of delivery of a message by the Basic SW. If a radio message is lost, the sender will not receive the answer and shall repeat it.

### SW API extract (ADA Source Code)

-------------------------------------------------------------------------------------------------

-- RTM types

-------------------------------------------------------------------------------------------------

-- Number of RTM physical communication session

type RTM\_COMMUNICATION\_SESSION\_NBR\_T is range 0 .. 2;

-- types used for the identity of radio network

subtype NETWORK\_ID\_DIGIT\_T is TYPES.UNSIGNED\_BYTE\_T range 0 .. 9;

type RADIO\_NETWORK\_ID\_LENGTH\_T is range 0 .. 6;

type RADIO\_NETWORK\_ID\_MAP\_T is array (RADIO\_NETWORK\_ID\_LENGTH\_T range <>) of NETWORK\_ID\_DIGIT\_T;

type RADIO\_NETWORK\_ID\_T (LENGTH : RADIO\_NETWORK\_ID\_LENGTH\_T := RADIO\_NETWORK\_ID\_LENGTH\_T'FIRST) is

record

ID : RADIO\_NETWORK\_ID\_MAP\_T (1 .. LENGTH);

end record;

UNKNOWN\_RADIO\_NETWORK\_ID\_C : constant RADIO\_NETWORK\_ID\_T := (LENGTH => 0, ID => (others => 0));

type BEARER\_CAPABILITY\_T is (GSM\_R\_ONLY, GPRS\_WITH\_GSM\_R\_FALLBACK); -- Q\_RADIO

-- OpenETCS : BEARER\_CAPABILITY shall always be GSM\_R\_ONLY

type APN\_IDENTITY\_T is range 0 .. 255; -- 0 means ETCS\_MNC\_MCC\_GPRS -- NID\_APN

-- OpenETCS : APN\_IDENTITY is not applicable, it shall not be used in the API

type NETWORK\_REGISTRATION\_INFO\_T (BEARER\_TYPE : BEARER\_CAPABILITY\_T := GSM\_R\_ONLY) is

record

NETWORK\_ID : RADIO\_NETWORK\_ID\_T;

case BEARER\_TYPE is

when GSM\_R\_ONLY =>

null;

when GPRS\_WITH\_GSM\_R\_FALLBACK =>

NID\_APN : APN\_IDENTITY\_T;

end case;

end record;

UNKNOWN\_NETWORK\_REGISTRATION\_INFO\_C : constant NETWORK\_REGISTRATION\_INFO\_T := (BEARER\_TYPE => GSM\_R\_ONLY,

NETWORK\_ID => UNKNOWN\_RADIO\_NETWORK\_ID\_C);

type MOBILE\_NETWORK\_REGISTRATION\_T is (NETWORK\_NOT\_REGISTERED, NETWORK\_REGISTER\_REQUEST, NETWORK\_REGISTER\_CONFIRM);

-- NETWORK\_NOT\_REGISTERED : the mobile has no registration on-going

-- NETWORK\_REGISTER\_REQUEST : the mobile is under registration

-- NETWORK\_REGISTER\_CONFIRM : the mobile is registered to the network

type MOBILE\_PDP\_CONTEXT\_T is (PDP\_NOT\_ACTIVATED, PDP\_ACTIVATION\_REQUEST, PDP\_ACTIVATION\_CONFIRM);

-- PDP\_NOT\_ACTIVATED : the mobile has no PDP context activation on-going

-- PDP\_ACTIVATION\_REQUEST : the mobile is activating a PDP context

-- PDP\_ACTIVATION\_CONFIRM : the mobile PDP context is activated

-- OpenETCS :MOBILE\_PDP\_CONTEXT shall always be PDP\_NOT\_ACTIVATED

type MOBILE\_COMM\_STATUS\_T is (MOBILE\_FREE, MOBILE\_IN\_SAFE\_COMM, MOBILE\_IN\_NON\_SAFE\_COMM);

-- MOBILE\_FREE : the mobile is free

-- MOBILE\_IN\_SAFE\_COMM : a safe connection (ETCS) is on-going via the mobile

-- MOBILE\_IN\_NON\_SAFE\_COMM : a non safe connection is on-going via the mobile

-- OpenETCS :MOBILE\_COMM\_STATUS shall always be MOBILE\_FREE or MOBILE\_IN\_SAFE\_COMM

type MOBILE\_CALL\_TYPE\_T is (CSD, PSD);

-- CSD : the mobile is used in CSD mode (Circuit Switched Data)

-- PSD : the mobile is used in PSD mode (Packet Switched Data)

-- OpenETCS : MOBILE\_CALL\_TYPE shall always be CSD

type MOBILE\_CONTEXT\_T is

record

IS\_OK : TYPES.BOOLEAN\_T;

COMM\_STATUS : MOBILE\_COMM\_STATUS\_T;

CALL\_TYPE : MOBILE\_CALL\_TYPE\_T;

NETWORK\_REGISTRATION : MOBILE\_NETWORK\_REGISTRATION\_T;

PDP\_CONTEXT : MOBILE\_PDP\_CONTEXT\_T;

NETWORK\_ID : RADIO\_NETWORK\_ID\_T;

end record;

type MOBILE\_T is (MOBILE\_1, MOBILE\_2);

type MOBILE\_TABLE\_T is array (MOBILE\_T) of MOBILE\_CONTEXT\_T;

-- ETCS\_ID type ( European Train Control System IDentification ).

type L\_ETCS\_ID\_T is range 1 .. 3;

type ETCS\_ID\_T is array (L\_ETCS\_ID\_T) of TYPES.UNSIGNED\_BYTE\_T;

UNKNOWN\_ETCS\_ID\_C : constant ETCS\_ID\_T := (others => TYPES.UNSIGNED\_BYTE\_T'LAST);

-- NIDRADIO types is used for radio subscriber number

type RADIO\_DIGIT\_T is range 0 .. 9;

for RADIO\_DIGIT\_T'SIZE use 4 \* TYPES.BITS;

type NIDRADIO\_LENGTH\_T is range 0 .. 16; -- Length of the phone number.

type NIDRADIO\_MAP\_T is array (NIDRADIO\_LENGTH\_T range <>) of RADIO\_DIGIT\_T; -- Phone Number.

type NIDRADIO\_T (LENGTH : NIDRADIO\_LENGTH\_T := NIDRADIO\_LENGTH\_T'FIRST) is

record

LIST : NIDRADIO\_MAP\_T (1 .. LENGTH) := (others => 0);

end record;

UNKNOWN\_NIDRADIO\_C : constant NIDRADIO\_T := (LENGTH => 0, LIST => (others => 0));

type CALL\_TYPE\_T is (CSD\_ONLY,

PSD\_WITH\_FALLBACK,

PSD\_ONLY); -- PSD\_ONLY is only used for KMC management in the BSW layer.

-- OpenETCS : CALL\_TYPE\_T shall always be CSD\_ONLY

type RADIO\_QUALITY\_OF\_SERVICE\_T is range 0 .. 255;

-- OpenETCS : RADIO\_QUALITY\_OF\_SERVICE is not applicable, it shall not be used in the API

type CALL\_INFO\_T (CALL\_TYPE : CALL\_TYPE\_T := CSD\_ONLY) is

record

NIDRADIO : NIDRADIO\_T;

case CALL\_TYPE is

when CSD\_ONLY =>

null;

when PSD\_WITH\_FALLBACK

| PSD\_ONLY =>

QUALITY\_OF\_SERVICE : RADIO\_QUALITY\_OF\_SERVICE\_T;

end case;

end record;

UNKNOWN\_CALL\_INFO\_C : constant CALL\_INFO\_T := (CALL\_TYPE => CSD\_ONLY,

NIDRADIO => UNKNOWN\_NIDRADIO\_C);

type RADIO\_PRIORITY\_LEVEL\_T is (LOW, HIGH);

type RTM\_DISCONNECTION\_REASON\_T is range 0 .. 255;

type RTM\_DIAGNOSTIC\_CODE\_T is

record

REASON : RTM\_DISCONNECTION\_REASON\_T;

SUBREASON : RTM\_DISCONNECTION\_REASON\_T;

end record;

RADIO\_AUTHENTIFICATION\_FAILURE\_REASON\_C : constant RTM\_DIAGNOSTIC\_CODE\_T := (REASON => 4,

SUBREASON => 2);

NORMAL\_REQUEST\_FROM\_APPLI\_REASON\_C : constant RTM\_DIAGNOSTIC\_CODE\_T := (REASON => 0,

SUBREASON => 0);

-- CONNECTION\_CONFIRMATION : to send at every connection or reconnection

-- CONNECTION\_LOST : to send when the connection is lost (if it was previously established)

-- CONNECTION\_FAILURE : to send when it has not been possible to (re)establish the connection after 3 attempts (if connection attempts is not infinite)

-- CONNECTION\_NOT\_ESTABLISHED : to send when it has not been possible to (re)establish the connection after 3 attempts (if connection attempts is infinite)

-- DATA : euroradio data message from trackside

type RTM\_IN\_MESSAGE\_KIND\_T is (CONNECTION\_CONFIRMATION,

CONNECTION\_LOST,

CONNECTION\_FAILURE,

CONNECTION\_NOT\_ESTABLISHED,

DATA);

-- NETWORK\_REQUEST : order to register the mobiles to a network

-- CONNECTION\_REQUEST : order to connect

-- DISCONNECTION\_REQUEST : order to disconnect

-- RESET\_CONNECTION : order to disconnect and then reconnect

-- INFINITE\_CONNECTION\_ATTEMPTS : order to try to connect infinitely

-- CHANGE\_OF\_PRIORITY : order to modify the priority management of a radio connection.

-- DATA : euroradio data message from trainborn

type RTM\_OUT\_MESSAGE\_KIND\_T is (NETWORK\_REQUEST,

CONNECTION\_REQUEST,

DISCONNECTION\_REQUEST,

RESET\_CONNECTION,

INFINITE\_CONNECTION\_ATTEMPTS,

CHANGE\_OF\_PRIORITY,

DATA);

type RTM\_IN\_MESSAGE\_T (KIND : RTM\_IN\_MESSAGE\_KIND\_T := RTM\_IN\_MESSAGE\_KIND\_T'FIRST) is

record

RADIO\_DEVICE : ETCS\_ID\_T;

case KIND is

when CONNECTION\_CONFIRMATION =>

null;

when CONNECTION\_LOST

| CONNECTION\_FAILURE

| CONNECTION\_NOT\_ESTABLISHED =>

REASON : RTM\_DIAGNOSTIC\_CODE\_T;

when DATA =>

DATA : INTERFACE\_LANGUAGE\_TYPES.RTM\_MESSAGE\_T;

end case;

end record;

type RTM\_IN\_EMERGENCY\_MESSAGE\_T is

record

RADIO\_DEVICE : ETCS\_ID\_T;

DATA : INTERFACE\_LANGUAGE\_TYPES.RTM\_EMERGENCY\_MESSAGE\_T;

end record;

type RTM\_OUT\_MESSAGE\_T (KIND : RTM\_OUT\_MESSAGE\_KIND\_T := RTM\_OUT\_MESSAGE\_KIND\_T'FIRST) is

record

RADIO\_DEVICE : ETCS\_ID\_T;

case KIND is

when NETWORK\_REQUEST =>

NETWORK\_INFO : NETWORK\_REGISTRATION\_INFO\_T;

when CONNECTION\_REQUEST =>

CALL\_INFO : CALL\_INFO\_T;

PRIORITY : RADIO\_PRIORITY\_LEVEL\_T;

when DISCONNECTION\_REQUEST

| RESET\_CONNECTION =>

REASON : RTM\_DIAGNOSTIC\_CODE\_T;

when INFINITE\_CONNECTION\_ATTEMPTS =>

null;

when CHANGE\_OF\_PRIORITY =>

NEW\_PRIORITY : RADIO\_PRIORITY\_LEVEL\_T;

when DATA =>

DATA : INTERFACE\_LANGUAGE\_TYPES.RTM\_MESSAGE\_T;

end case;

end record;

.

.

.

package ERTMS\_TRAINBORN\_GENERIC\_API is

.

.

.

-------------------------------------------------------------------------------------------------

-- RTM services

-------------------------------------------------------------------------------------------------

-- Services to know about radio environment

-------------------------------------------

-- function which returns TRUE if the train is in an expected radio hole (a tunnel for instance)

-- this function is used by the basic software to know if a disconnection is expected or not

function TRAIN\_IS\_IN\_A\_RADIO\_HOLE return TYPES.BOOLEAN\_T;

-- procedure to send to the application the number of communication sessions which are possible

-- simultaneously this value is dynamic folowing the hardware equipment status

-- but once it reachs 0, it never increases anymore

procedure NUMBER\_OF\_HANDABLE\_RTM\_COMMUNICATION\_SESSION

(SESSION\_NBR : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.RTM\_COMMUNICATION\_SESSION\_NBR\_T);

-- procedure to send to the application the context of each mobile

procedure WRITE\_MOBILE\_CONTEXT (THE\_CONTEXT : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.MOBILE\_TABLE\_T);

---------------------------------------------------

-- procedures to manage the messages and connection

---------------------------------------------------

-- procedure to deliver a radio message to the application

procedure WRITE\_RTM\_MESSAGE

(THE\_RTM\_MESSAGE : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.RTM\_IN\_MESSAGE\_T);

-- procedure to read a radio message from the application

procedure READ\_RTM\_MESSAGE

(THE\_RTM\_MESSAGE : out ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.RTM\_OUT\_MESSAGE\_T);

-- this function returns TRUE if the output RTM MESSAGE QUEUE IS EMPTY

-- and returns FALSE otherwise

function RTM\_MESSAGE\_QUEUE\_IS\_EMPTY return TYPES.BOOLEAN\_T;

-- procedure to deliver emergency radio message to the application

procedure WRITE\_RTM\_EMERGENCY\_MESSAGE

(THE\_RTM\_MESSAGE : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.RTM\_IN\_EMERGENCY\_MESSAGE\_T);

.

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API;

## Event report

### Functional data flows (see /6/)

#### Input

- basic\_to\_generic\_app\_info .events\_from\_basic

#### Output

None.

### Service EVENT\_REPORT

#### Description

The service is used by the Basic SW to indicate to the Application SW that a given event has occurred.

This service must be used to inform the Application SW of a events leading to actions which can only be performed by the Application SW (ex : display of a text message on the DMI).

The events and the related actions have to be defined.

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| EVENT | API\_TYPES.EVENT\_T | in | The event that occurred on the target. |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

This procedure must be called by the Basic SW once each time an event is detected.

### SW API extract (ADA Source Code)

package ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES is

.

.

.

-------------------------------------------------------------------------------------------------

-- Event types

-------------------------------------------------------------------------------------------------

type EVENT\_T is

(EXTERNAL\_SMALL\_AVAILABILITY,

LOOP\_RECEIVER\_FAILURE,

FVL\_ENTER\_HE\_MODE,

FVL\_EXIT\_HE\_MODE,

...

);

-------------------------------------------------------------------------------------------------

.

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES;

package ERTMS\_TRAINBORN\_GENERIC\_API is

.

.

.

-------------------------------------------------------------------------------------------------

-- Event services

-------------------------------------------------------------------------------------------------

-- procedure to call once when the event is detected

procedure EVENT\_REPORT (EVENT : in ERTMS\_TRAINBORN\_GENERIC\_API\_TYPES.EVENT\_T);

.

.

.

end ERTMS\_TRAINBORN\_GENERIC\_API;

## Faults (ERROR MANAGEMENT)

### Functional data flows (see /6/)

#### Input

None.

#### Output

- generic\_app\_to\_basic\_info .channels\_extinction\_is\_required

### Service ERROR\_MANAGEMENT

#### Description

As described at §3.3.1; this service belongs to Basic SW and is to be called by the Application SW : if an error is detected within the application and the application wishes to report it, an explicitly defined INFO\_ERROR\_T will be used. The names of these errors are to be determined by the design.

The Basic SW provides an error reporting routine ERROR\_MANAGEMENT that requires specific error codes and reaction to be defined. When this ERROR\_MANAGEMENT routine is called, the Basic SW has to take the predefined actions (memorisation, brake application, shutdown of the system, …).

#### Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| Error code | INFO\_ERROR\_T | out | The error code |
| Error is present | Boolean | out | The error is present or not |

#### Returned value

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Direction | Description |
| - | - | - | - |

#### Expected behaviour & usage

The error codes list and the corresponding reaction of the Basic SW are to be defined.

### SW API extract (ADA Source Code)

package ATBL\_ERROR is

.

.

.

procedure ERROR\_MANAGEMENT

(INFO\_ERROR : in INFO\_ERROR\_T;

ERR\_PRESENT : in TYPES.BOOLEAN\_T := TRUE;

COUNTER\_VALUE : in TYPES.UNCHECKED\_BYTE\_T := 1;

NBR\_OF\_CHANNELS : in NBR\_CHANNELS\_IN\_FAILURE\_T := 0;

CHANNEL1, CHANNEL2 : in GLOBAL\_DATA\_3CH.ORDER\_3CH\_T := GLOBAL\_DATA\_3CH.OWN);

.

.

.

end ATBL\_ERROR;

# Appendix 1 – application layer (telegram definition)

Please refer to the separate document “API Requirements for OpenETCS – appendix - application layer v.1.0” (/5/)

# Appendix 2 – Functional data dictionary

Please refer to the separate document “API Requirements for OpenETCS – appendix - functional data dictionary v.1.0” (/6/)

1. M : meeting review, R : read-back process [↑](#footnote-ref-1)