ITEA2 PROJECT

2012-2015

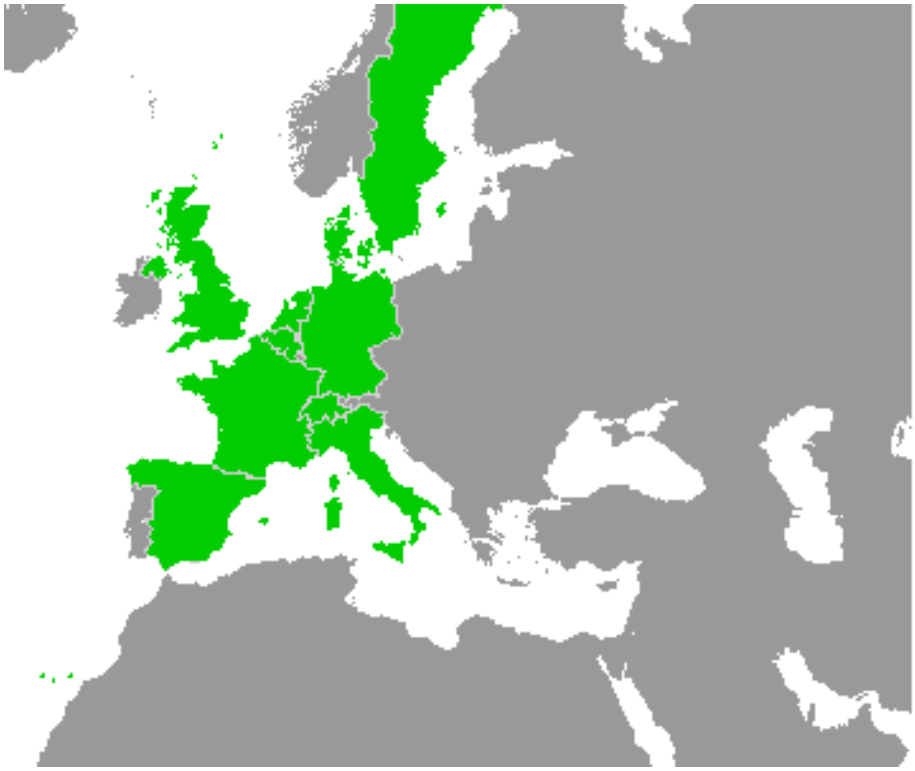
Frame to be used to indicate a customer reference number.

Client :	C/Ref. :

Work-Package 2 : “Requirements”

API Requirements for OpenETCS – v1.4

N. Boverie September 2014



ITEA2 PROJECT

2012-2015

Amendment record

Rev.[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)

1 M : meeting review, R : read-back process

ITEA2 PROJECT

2012-2015

Table of Content

ITEA2 PROJECT

2012-2015

1. INTRODUCTION

1.1 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).

1.2 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.

1.3 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

ITEA2 PROJECT

2012-2015

- 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

2. DOCUMENTS & TERMINOLOGY

2.1 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

2.2 APPLICABLE DOCUMENTS

/5/ API Requirements for OpenETCS – appendix - Application layer, v1.2
/6/ API Requirement for OpenETCS – appendix - Functional Data Dictionary, v1.1

2.3 DEFINITIONS

Refer also to /2/

2.4 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/

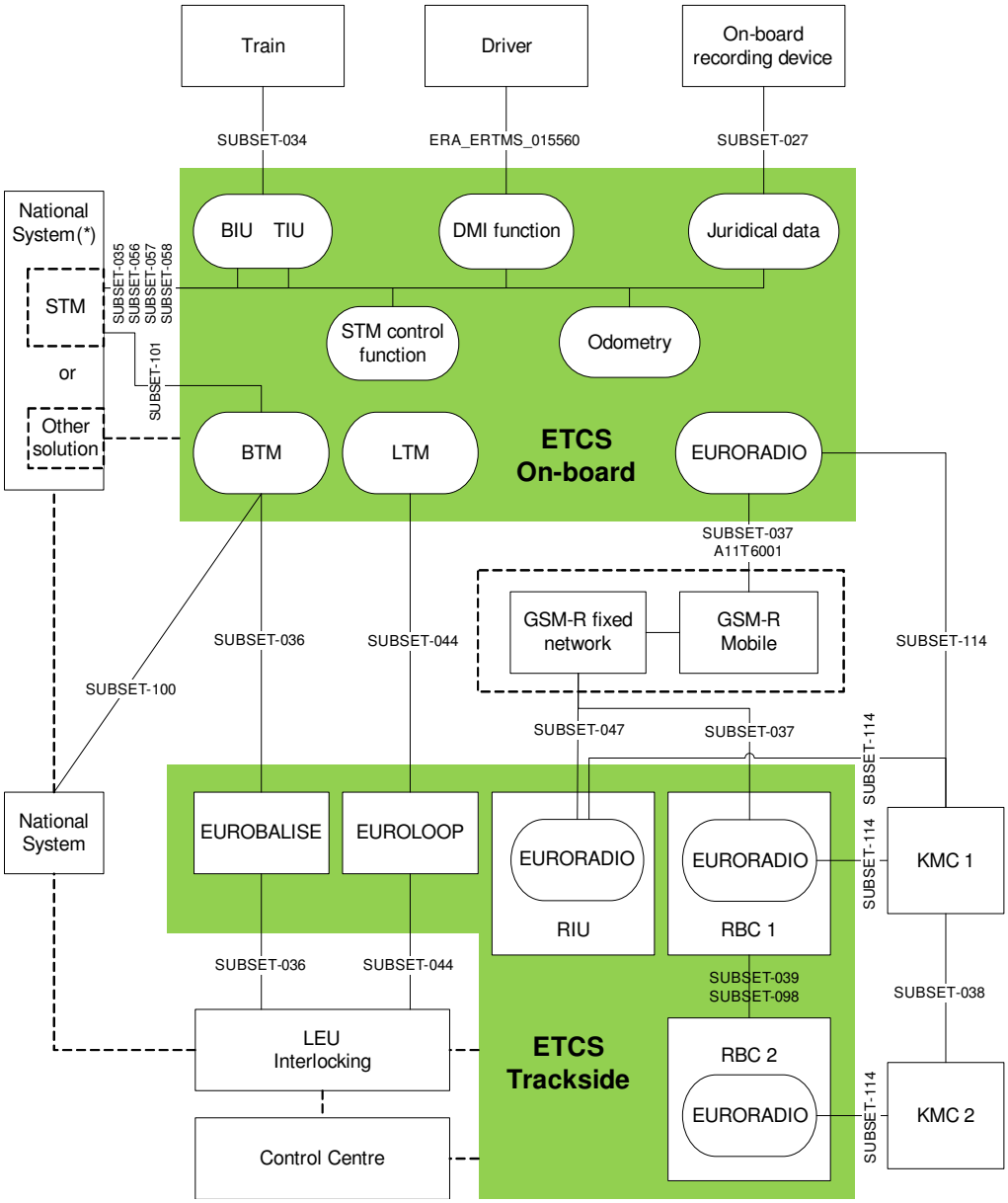
General remark about abbreviations: This document intends to use as much as possible the abbreviations commonly used by the Unisig and the ETCS community in general (See document /2/).

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.

3. OVERVIEW

3.1 DEFINITION OF THE OPENETCS

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

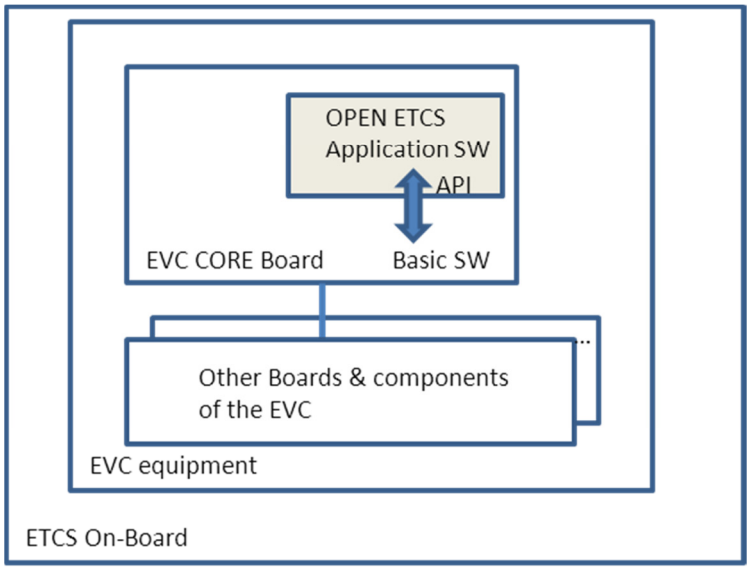


(*) Depending on its functionality and the desired configuration , the national system can be addressed either via an STM using the standard interface or via another national solution

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.

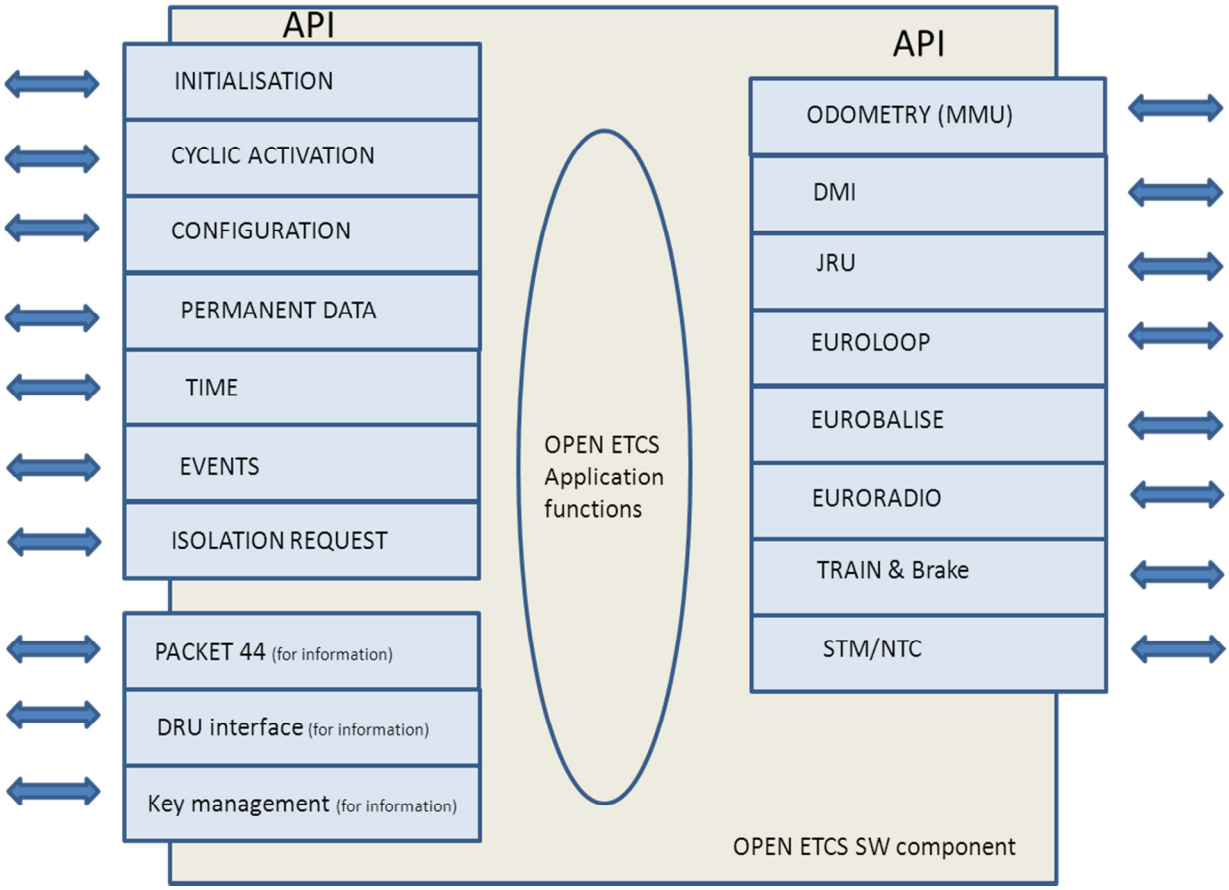
The OpenETCS API shall be the interface between the Application SW and the Basic SW of the CORE board. (notice that the API SW belongs to the Application SW component)



3.2 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.

3.3 GENERAL PRINCIPLES AND RULES

3.3.1 CORE board : SW architecture overview

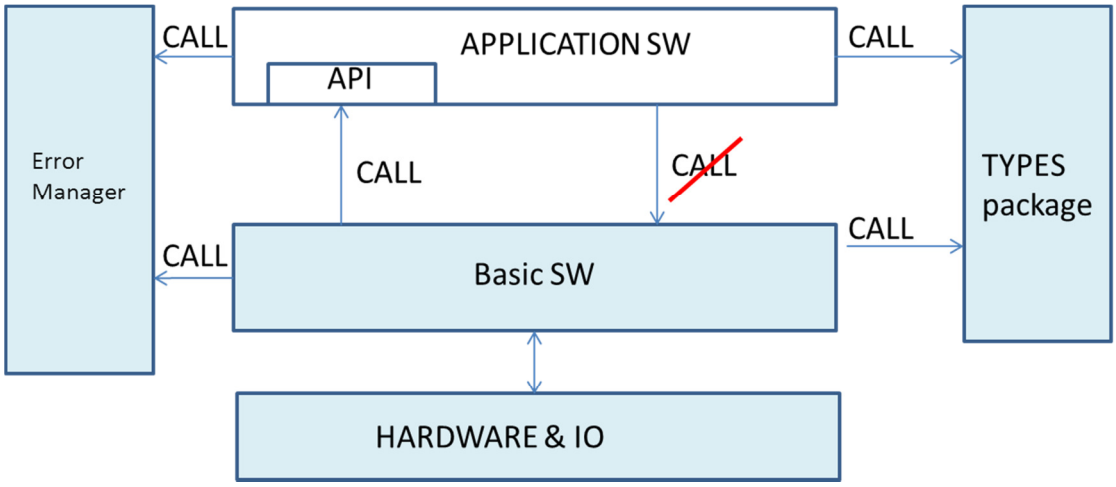
From a structural point of view, the Basic Software is in charge of calling all services of the Application SW (through the API).

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, ..).

3.3.2 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.

3.3.3 OpenETCS API services and calling sequence

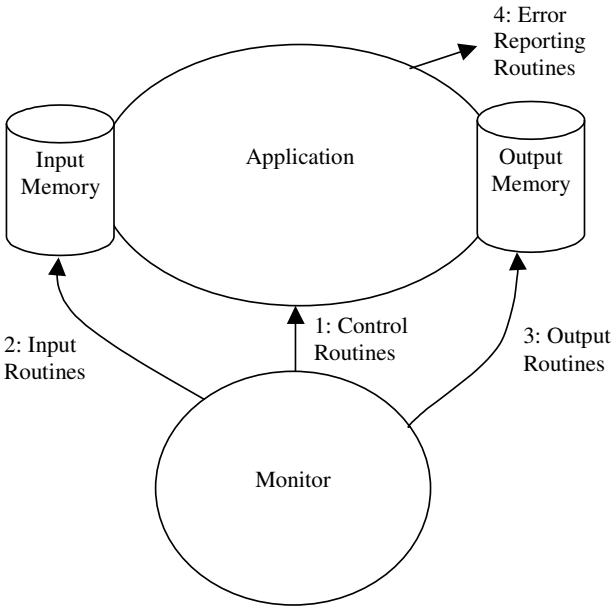
There are basically 3 types of API services that are invoked by the Basic SW :

- 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)

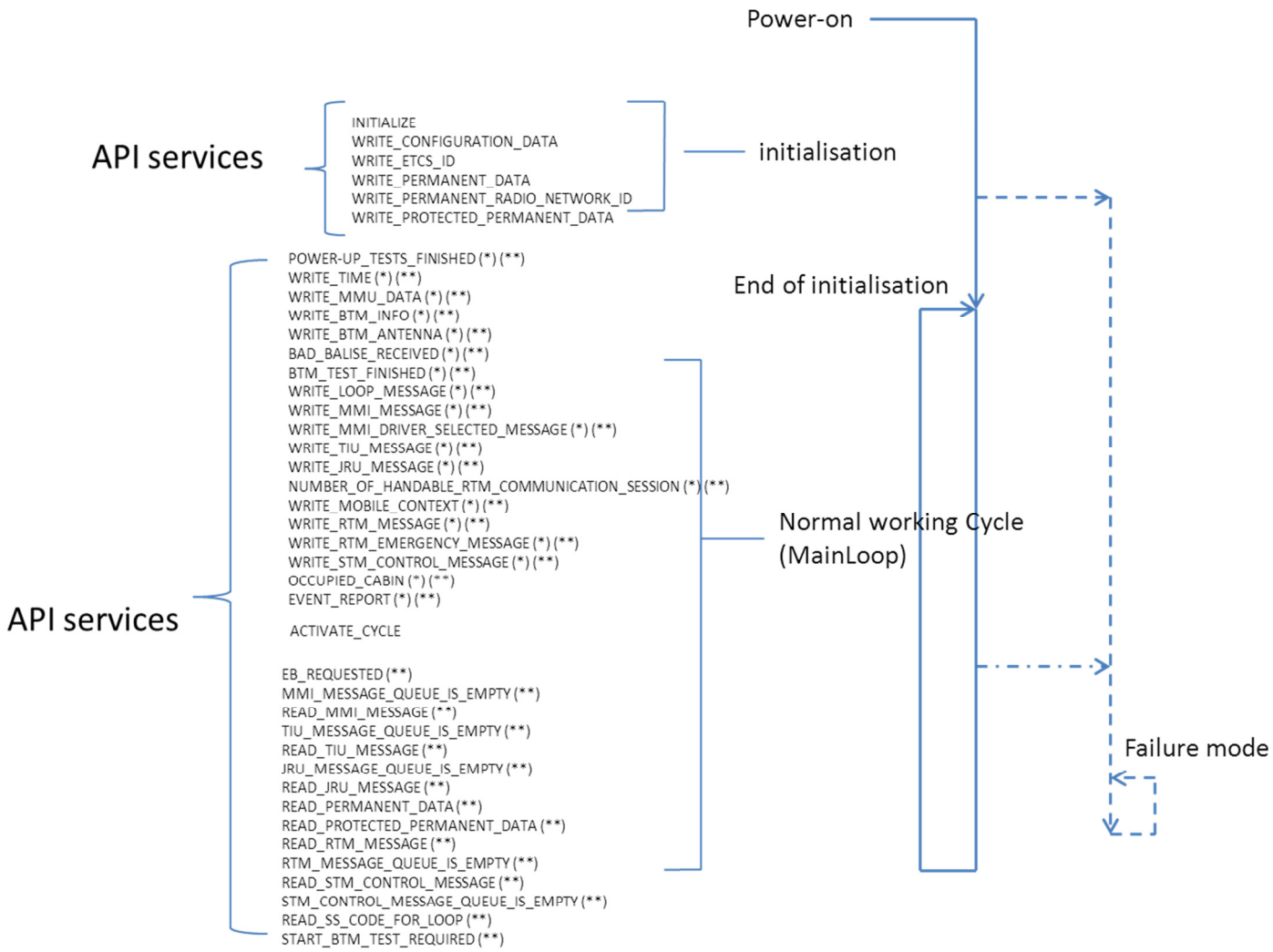
A fourth type is the calling of the ERROR_MANAGER by the Application SW ("error reporting routines" in the following figure). The calling to the ERROR_MANAGER shall be used to shutdown the platform in case a safety failure is detected (parameter set to "shutdown request"); see also §3.3.2. Simple events (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 :



(*) Inputs services are called [0..N] times during the cycle according to the nature of the data provided : event(s) / state.
(**) The application shall not rely on the calling order of the input and output services

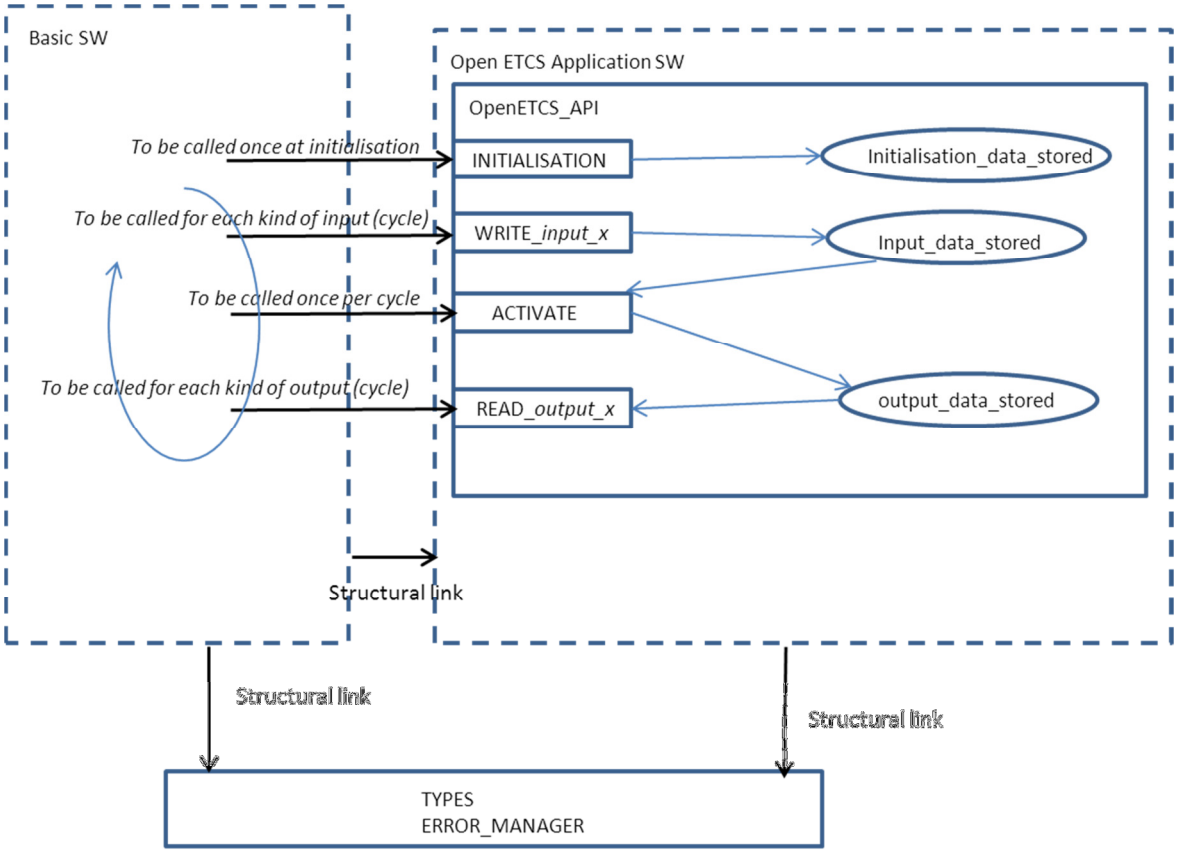
The order into which the inputs and output services of the Application Software are called is NOT defined. It does not have to be formally defined because the Application Software shall not take hypothesis on that order.

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.

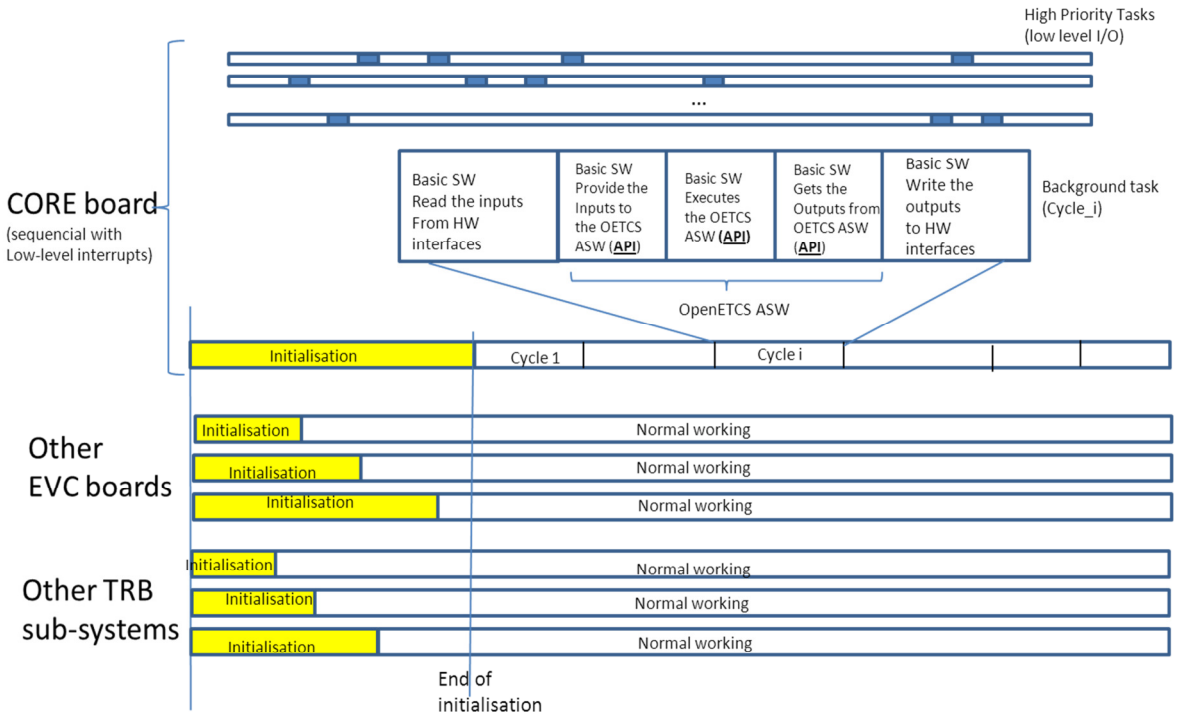
The main application processing shall be executed (when the Basic Software calls the "application activation" procedure) only after all "write inputs" have been called by the Basic Software meaning that all the newest input data of the current cycle are provided to the application.

See another illustration on the following figure :



3.3.4 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).

3.3.5 Asynchronous behaviour

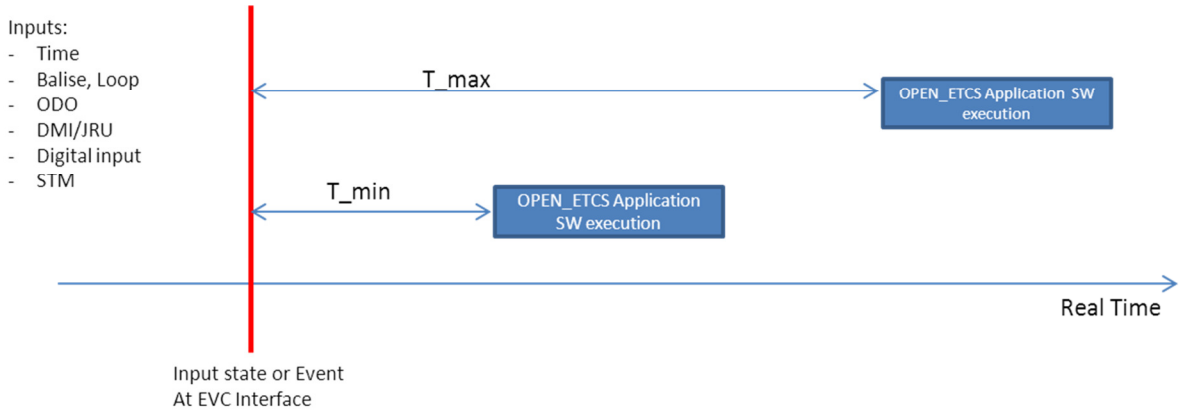
In addition to the "cyclic execution" principle (§3.3.2, §3.3.3), the application must take into account that the various peripheral boards/sub-systems in charge of the input/output treatment are not synchronised and that their performances are not identical, nor constant in time;

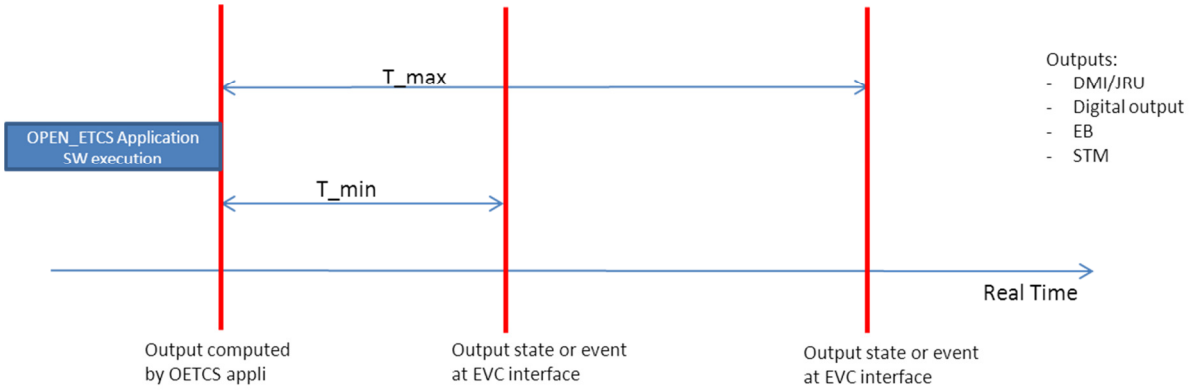
- e.g filtering of analog data such as GEOS sensors, train digital inputs, …
- e.g normal propagation durations on buses for which the performance can vary (chained treatments).

Therefore,

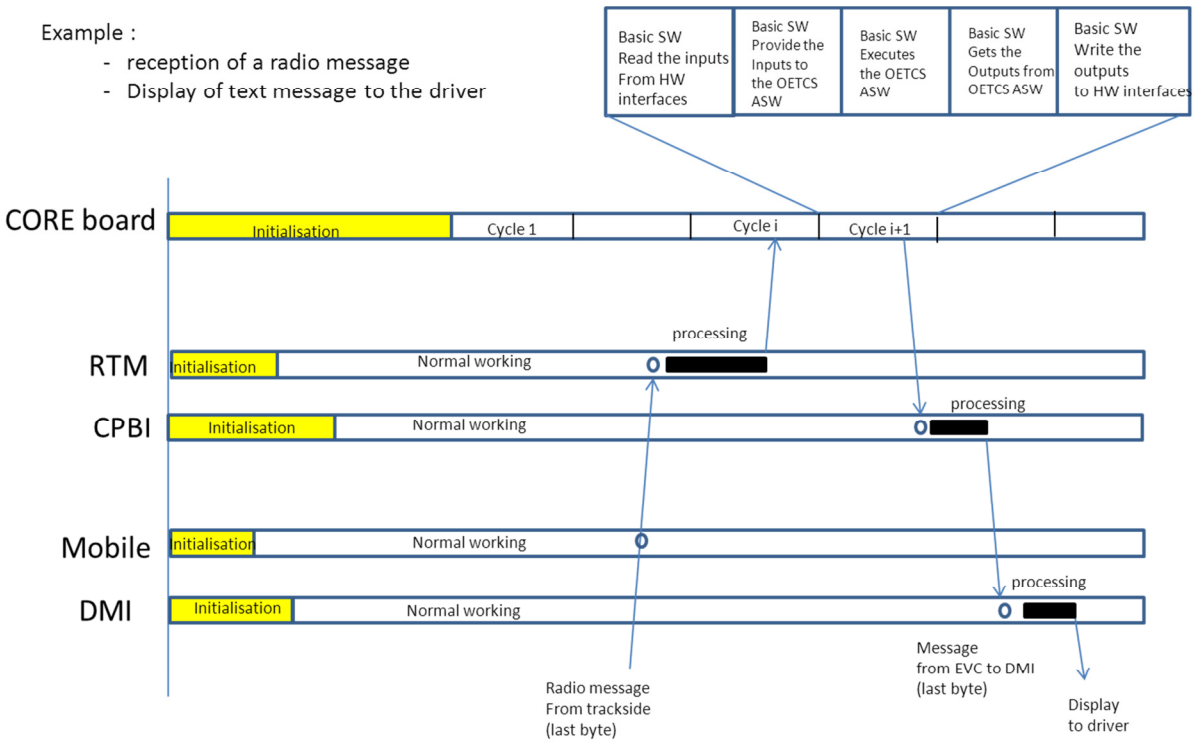
- The input data present at the interface of the Onboard ETCS/EVC will be provided with a variable delay to the application (delay, jitter);
- The application output data are provided to the EVC/Onboard ETCS interface with a variable delay (delay, jitter).

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).

4. INTERFACE SPECIFICATIONS

4.1 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).

4.1.1 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.

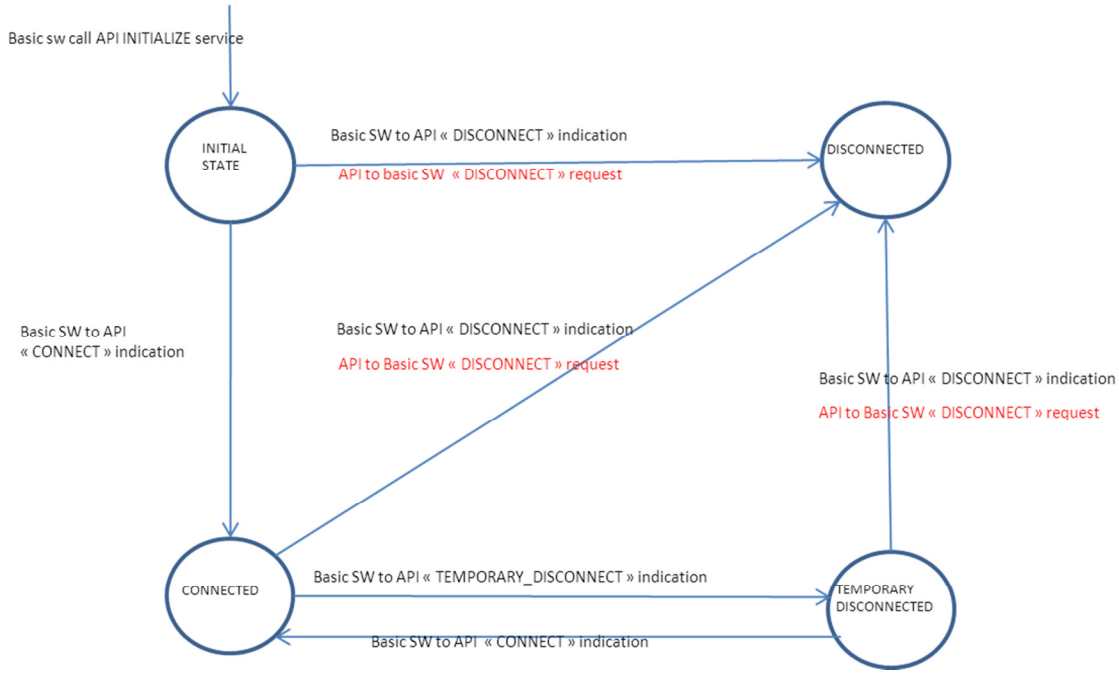
The Basic Software will start to request the first connection with the peripherals not sooner than 20s and not later than 80s after power-up.

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

4.1.2 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).

4.1.3 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

4.2 CONTROL INTERFACE

4.2.1 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)

4.2.2 Functional data flows(see /6/)

4.2.2.1 Input

None.

4.2.2.2 Output

None.

4.2.3 Service INITIALIZE

4.2.3.1 Description

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

4.2.3.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.2.3.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.2.3.4 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).

4.2.4 Service ACTIVATE_CYCLE

4.2.4.1 Description

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

4.2.4.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.2.4.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.2.4.4 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.

4.2.5 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;
```

4.3 CONFIGURATION INTERFACE

4.3.1 Functional data flows (see /6/)

4.3.1.1 Input

- basic_to_generic_app_info .coded_config_data
- basic_to_generic_app_info .specific_config_data

4.3.1.2 Output

None.

4.3.2 Service WRITE_CONFIG_DATA

4.3.2.1 Description

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

4.3.2.2 Parameter

Name	Type	Direction	Description
PLUG	PLUG_BIT_STREAM_T	in	The configuration file related to the Application.

4.3.2.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.3.2.4 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.

4.3.3 Service WRITE_ETCS_ID

4.3.3.1 Description

This service is used to send the ETCS_ID configuration parameter to the Application Software.

4.3.3.2 Parameter

Name	Type	Direction	Description
ETCS_ID	API_TYPES.ETCS_ID_T	in	The ETCS ID parameter

4.3.3.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.3.3.4 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.

4.3.4 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;
```

4.4 PERMANENT DATA INTERFACE

4.4.1 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)

4.4.2 Functional data flows (see /6/)

4.4.2.1 Input

- basic_to_generic_app_info .coded_data_restored_at_power_up

4.4.2.2 Output

- generic_app_to_basic_info .coded_data_to_be_restored_at_power_up

4.4.3 Service WRITE_PERMANENT_DATA and WRITE_PERMANENT_RADIO_NETWORK_INFO

4.4.3.1 Description

See 4.4.1.

4.4.3.2 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.

4.4.3.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.4.3.4 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).

4.4.4 Service WRITE_PROTECTED_PERMANENT_DATA

4.4.4.1 Description

See 4.4.1.

4.4.4.2 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.

4.4.4.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.4.4.4 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.

4.4.5 Service READ_PERMANENT_DATA

4.4.5.1 Description

See 4.4.1.

4.4.5.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.4.5.3 Returned value

Name	Type	Direction	Description

-	API_TYPES.PERMANENT_DATA_T	-	The data to store in the Novram of the EVC

4.4.5.4 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)

4.4.6 Service READ_PROTECTED_PERMANENT_DATA

4.4.6.1 Description

See 4.4.1.

4.4.6.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.4.6.3 Returned value

Name	Type	Direction	Description
-	API_TYPES.PROTECTED_PERMANENT_DATA_T	-	The data to store in the Novram of the EVC

4.4.6.4 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)

4.4.7 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;
```

4.4.8 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.

4.5 TESTS INTERFACE

4.5.1 Functional data flows (see /6/)

4.5.1.1 Input

- basic_to_generic_app_info .power_up_tests_info

4.5.1.2 Output

None.

4.5.2 Service POWER_UP_TESTS_FINISHED

4.5.2.1 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.

4.5.2.2 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);

4.5.2.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.5.2.4 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.

4.5.3 Service START_BTM_TESTS_REQUIRED

4.5.3.1 Description

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

4.5.3.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.5.3.3 Returned value

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

4.5.3.4 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.

4.5.4 Service BTM_TESTS_FINISHED

4.5.4.1 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.

4.5.4.2 Parameter

Name	Type	Direction	Description
WITH_RESULT	API_TYPES.TEST_RESULT_T	in	The result of the BTM tests. (OK, NOT_OK, REDUCED_DISPONIBILITY);

4.5.4.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.5.4.4 Expected behaviour & usage

The BTM_TESTS_FINISHED function shall be called once, before Activate_Cycle, only when the BTM test is finished.

4.5.5 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;
```

4.6 TIME INTERFACE

4.6.1 Functional data flows (see /6/)

4.6.1.1 Input

- basic_to_generic_app_info .logical_voted_time

4.6.1.2 Output

None.

4.6.2 Service WRITE_TIME

4.6.2.1 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.

4.6.2.2 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)

4.6.2.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.6.2.4 Expected behaviour & usage

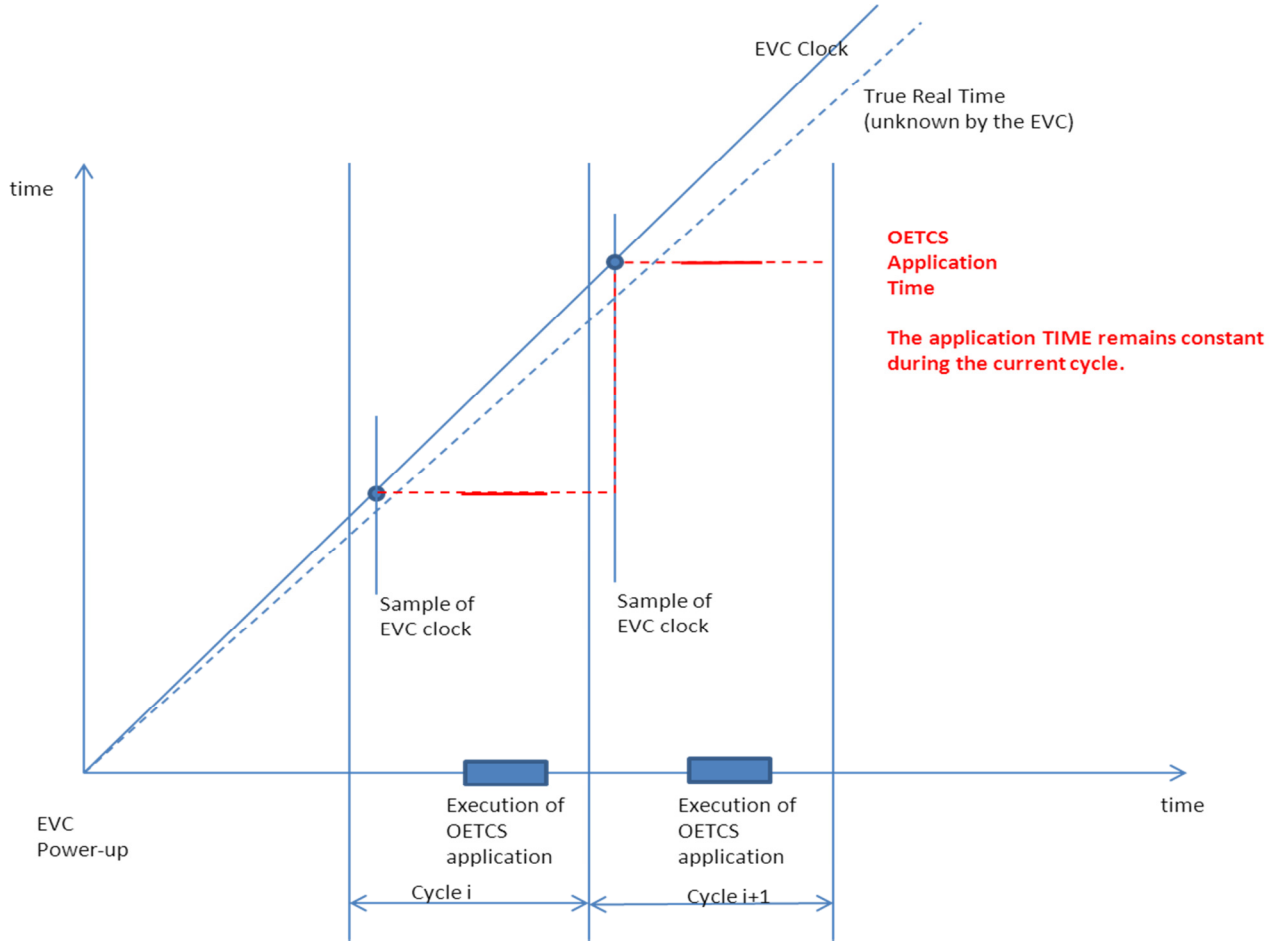
The Application SW should store the time and use it for all its internal computations.

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:



4.6.3 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;
```

4.7 MMU (ODOMETRY) INTERFACE

4.7.1 Functional data flows (see /6/)

4.7.1.1 Input

- basic_to_generic_app_info .MMU_input_info

4.7.1.2 Output

None.

4.7.2 Service WRITE_MMU_DATA

4.7.2.1 Description

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

4.7.2.2 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)

4.7.2.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.7.2.4 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.

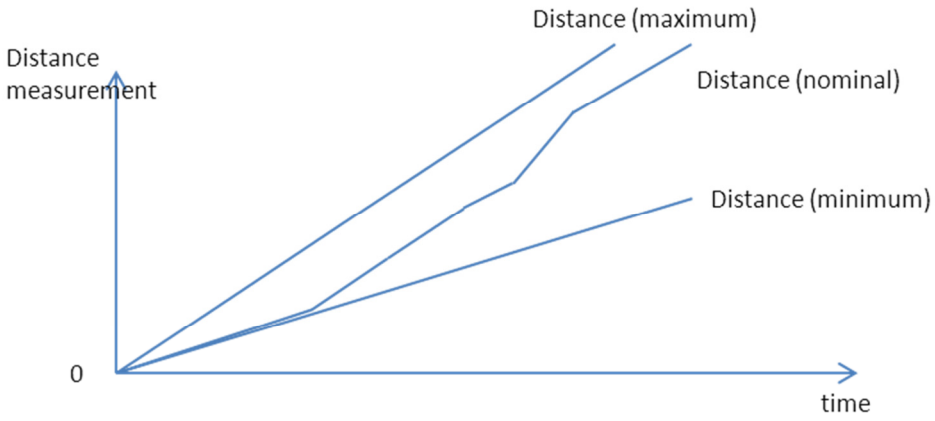
If the motion cannot be determined (i.e. due to a very low speed or small distance travelled), the MOTION_STATE or MOTION_DIRECTION may be set to UNKNOWN by the Basic Software.

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.

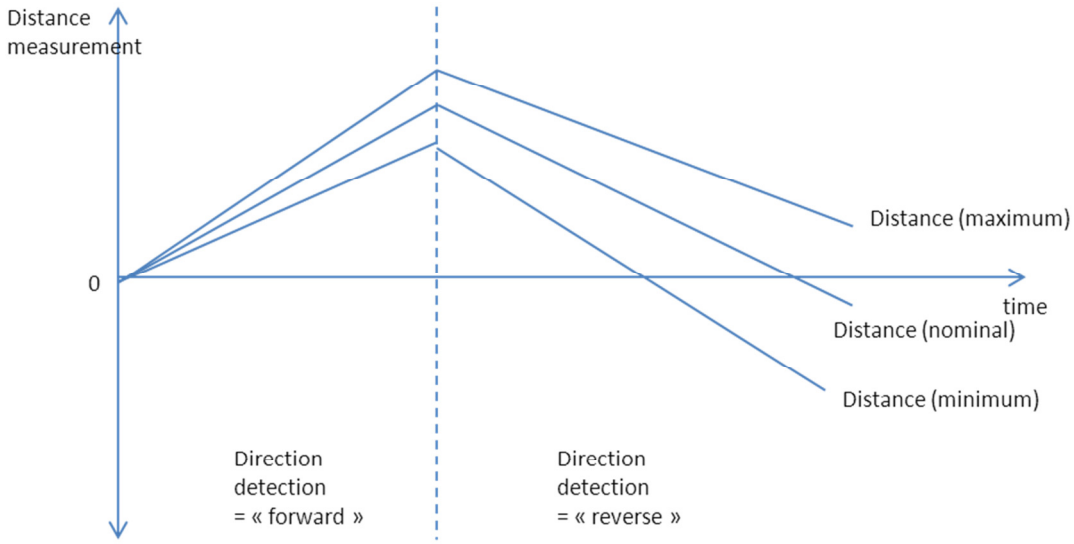
The position of the train is always 0 at the power-up of the EVC and it corresponds to the head of the train.

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 :

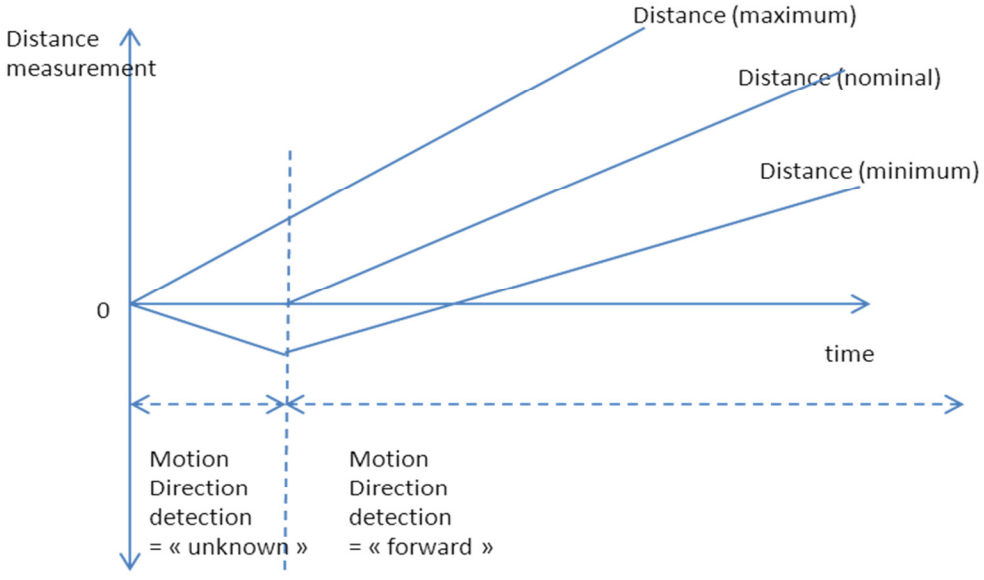


E.g Train direction forward and then reverse

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.

4.7.3 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;

4.8 EUROBALISE INTERFACE

4.8.1 Functional data flows (see /6/)

4.8.1.1 Input

- basic_to_generic_app_info .EUROBALISE_input_info

4.8.1.2 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

4.8.1.3 Application layer (telegram definition)

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

4.8.2 Service WRITE_BTM_INFO

4.8.2.1 Description

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

4.8.2.2 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

4.8.2.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.8.2.4 Expected behaviour & usage

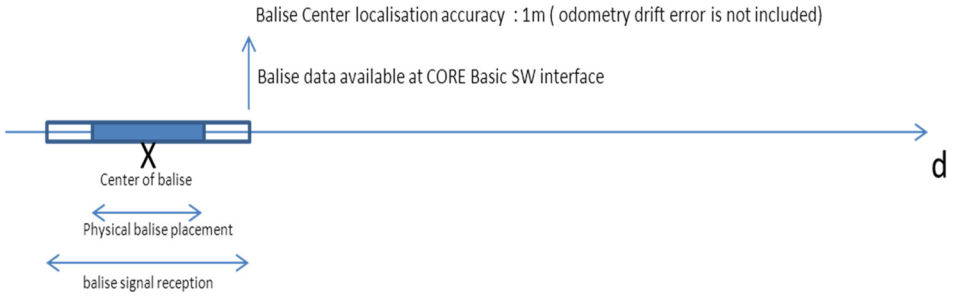
Before the activation of the Application SW, the Basic SW shall call this service once per balise crossed (since the previous cycle), respecting the chronology (older balises first), with the last information received from each balise (see further for details).

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.

4.8.3 Service WRITE_BTM_ANTENNA

4.8.3.1 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

4.8.3.2 Parameter

Name	Type	Direction	Description
THE_BTM_ANTENNA	API_TYPES.ANTENNA_T	in	The currently selected antenna

4.8.3.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.8.3.4 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.

4.8.4 Service BAD_BALISE_RECEIVED

4.8.4.1 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.

4.8.4.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.8.4.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.8.4.4 Expected behaviour & usage

According to the application specification.

4.8.5 Service READ_BTM_INFO

4.8.5.1 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).

4.8.5.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.8.5.3 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

4.8.5.4 Expected behaviour & usage

This service must be called by the Basic SW after each Activate_Cycle.

4.8.6 Service METAL_MASS_INFO

4.8.6.1 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.

4.8.6.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.8.6.3 Returned value

Name	Type	Direction	Description
-	API_TYPES.METAL_MASS_INFO_T	-	The information about the presence of metal masses on the track.

4.8.6.4 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.

4.8.7 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;
```

4.9 EUROLOOP INTERFACE

4.9.1 Functional data flows (see /6/)

4.9.1.1 Input

- basic_to_generic_app_info .EUROLOOP_input_info

4.9.1.2 Output

- generic_app_to_basic_info .EUROLOOP_output_info

4.9.1.3 Application layer (telegram definition)

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

4.9.2 Service WRITE_LOOP_MESSAGE

4.9.2.1 Description

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

4.9.2.2 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

4.9.2.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.9.2.4 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.

4.9.3 Service SS_CODE_FOR_LOOP

4.9.3.1 Description

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

4.9.3.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.9.3.3 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

4.9.3.4 Expected behaviour & usage

This service shall be called by the Basic SW once at each cycle after the ACTIVATE_CYCLE service.

4.9.4 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;
```

4.10 TRAIN INTERFACE UNIT (TIU)

4.10.1 Functional data flows (see /6/)

4.10.1.1 Input

- basic_to_generic_app_info .TIU_input_info

4.10.1.2 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

4.10.1.3 Application layer (telegram definition)

Please refer to /5/.

4.10.2 Service WRITE_TIU_MESSAGE

4.10.2.1 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.

4.10.2.2 Parameter

Name	Type	Direction	Description
THE_TIU_MESSAGE	TIU_LANGUAGE_TYPES.TIU_TO_CORE_BIT_STREAM_T	in	The message from the TIU

4.10.2.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.10.2.4 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.

4.10.3 Service READ_TIU_MESSAGE

4.10.3.1 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.

4.10.3.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.10.3.3 Returned value

Name	Type	Direction	Description
THE_TIU_MESSAGE	TIU_LANGUAGE_TYPES.CORE_TO_TIU_BIT_STREAM_T	out	The message to the TIU

4.10.3.4 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.

4.10.4 Service TIU_MESSAGE_QUEUE_IS_EMPTY

4.10.4.1 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.

4.10.4.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.10.4.3 Returned value

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

4.10.4.4 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.

4.10.5 Service EB_REQUESTED

4.10.5.1 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.

4.10.5.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.10.5.3 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

4.10.5.4 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.

4.10.6 Service EVC_ISOLATION_IS_REQUESTED

4.10.6.1 Description

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

4.10.6.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.10.6.3 Returned value

Name	Type	Direction	Description
-	TYPES.BOOLEAN_T	out	returns TRUE if the EVC has to be isolated

4.10.6.4 Expected behaviour & usage

This service shall be called by the Basic SW once per cycle after the ACTIVATE_CYCLE service.

4.10.7 Service OCCUPIED_CABIN

4.10.7.1 Description

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

4.10.7.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.10.7.3 Returned value

Name	Type	Direction	Description
THE_OCCCUPIED_CABIN	API_TYPES.OCCUPIED_CABIN_T	out	CAB_A, CAB_B, none

4.10.7.4 Expected behaviour & usage

This service shall be called by the Basic SW once per cycle after the ACTIVATE_CYCLE service.

4.10.8 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;
```

4.11 DRIVER MACHINE INTERFACE (DMI)

4.11.1 Functional data flows (see /6/)

4.11.1.1 Input

- basic_to_generic_app_info .EUROCAB_input_info

4.11.1.2 Output

- generic_app_to_basic_info .EUROCAB_output_info

4.11.1.3 Application layer (telegram definition)

Please refer to /5/.

4.11.2 Service WRITE_MMI_MESSAGE

4.11.2.1 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.

4.11.2.2 Parameter

Name	Type	Direction	Description
THE_MMI_MESSAGE	API_TYPES.MMI_IN_MESSAGE_T	in	The message from the DMI

4.11.2.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.11.2.4 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.

4.11.3 Service READ_MMI_MESSAGE

4.11.3.1 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.

4.11.3.2 Parameter

Name	Type	Direction	Description
THE_MMI_MESSAGE	API_TYPES.MMI_OUT_MESSAGE_T	out	The message to the DMI

4.11.3.3 Returned value

Name	Type	Direction	Description

-	-	-	-

4.11.3.4 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.

4.11.4 Service MMI_MESSAGE_QUEUE_IS_EMPTY

4.11.4.1 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.

4.11.4.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.11.4.3 Returned value

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

4.11.4.4 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.

4.11.5 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;
```

4.12 JRU INTERFACE

4.12.1 Functional data flows (see /6/)

4.12.1.1 Input

- basic_to_generic_app_info .EUROCAB_input_info

4.12.1.2 Output

- generic_app_to_basic_info .EUROCAB_output_info

4.12.1.3 Application layer (telegram definition)

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

4.12.2 Service WRITE_JRU_MESSAGE

4.12.2.1 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.

4.12.2.2 Parameter

Name	Type	Direction	Description
THE_JRU_MESSAGE	API_TYPES.JRU_IN_MESSAGE_T	in	The message from the JRU

4.12.2.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.12.2.4 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.

4.12.3 Service READ_JRU_MESSAGE

4.12.3.1 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.

4.12.3.2 Parameter

Name	Type	Direction	Description
THE_JRU_MESSAGE	API_TYPES.JRU_OUT_MESSAGE_T	out	The message to the JRU

4.12.3.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.12.3.4 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.

4.12.4 Service JRU_MESSAGE_QUEUE_IS_EMPTY

4.12.4.1 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.

4.12.4.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.12.4.3 Returned value

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

4.12.4.4 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.

4.12.5 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;

4.13 STM INTERFACE

4.13.1 Functional data flows (see /6/)

4.13.1.1 Input

- basic_to_generic_app_info .EUROCAB_input_info

4.13.1.2 Output

- generic_app_to_basic_info .EUROCAB_output_info

4.13.1.3 Application layer (telegram definition)

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

4.13.2 Service WRITE_STM_MESSAGE

4.13.2.1 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.

4.13.2.2 Parameter

Name	Type	Direction	Description
THE_STM_MESSAGE	API_TYPES.STM_IN_MESSAGE_T	in	The message from the STM

4.13.2.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.13.2.4 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.

4.13.3 Service READ_STM_MESSAGE

4.13.3.1 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.

4.13.3.2 Parameter

Name	Type	Direction	Description
THE_STM_MESSAGE	API_TYPES.STM_OUT_MESSAGE_T	out	The message to the STM

4.13.3.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.13.3.4 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.

4.13.4 Service STM_MESSAGE_QUEUE_IS_EMPTY

4.13.4.1 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.

4.13.4.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.13.4.3 Returned value

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

4.13.4.4 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.

4.13.5 Service STM_INFO

4.13.5.1 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.

4.13.5.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.13.5.3 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

4.13.5.4 Expected behaviour & usage

This service may be called by the Basic SW at each cycle after the ACTIVATE_CYCLE service.

4.13.6 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;
```

4.14 EURORADIO INTERFACE

4.14.1 Functional data flows (see /6/)

4.14.1.1 Input

- basic_to_generic_app_info .EURORADIO_input_info

4.14.1.2 Output

- generic_app_to_basic_info .EURORADIO_output_info

4.14.1.3 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.**

4.14.2 Service NUMBER_OF_HANDABLE_RTM_COMMUNICATION_SESSION

4.14.2.1 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.

4.14.2.2 Parameter

Name	Type	Direction	Description
-	-	-	-

4.14.2.3 Returned value

Name	Type	Direction	Description
-	API_TYPES. RTM_COMMUNICATION_SESSION_NBR_T	out	0,1 or 2

4.14.2.4 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.

4.14.3 Service WRITE_MOBILE_CONTEXT

4.14.3.1 Description

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

4.14.3.2 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

4.14.3.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.14.3.4 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 communication is established or that the EVC is trying to establish a communication. (e.g. The last command from the application to the radio sub-system was a CONNECTION_REQUEST or RESET_CONNECTION)

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.

4.14.4 Service READ_RTM_MESSAGE

4.14.4.1 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.

4.14.4.2 Parameter

Name	Type	Direction	Description

4.14.4.3 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

Other parameters are depending on the message kind.

4.14.4.4 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.

4.14.5 Service RTM_MESSAGE_QUEUE_IS_EMPTY

4.14.5.1 Description

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

4.14.5.2 Parameters

Name	Type	Direction	Description
-	-	-	-

4.14.5.3 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.

4.14.5.4 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.

4.14.6 Service WRITE_RTM_MESSAGE

4.14.6.1 Description

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

4.14.6.2 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.

4.14.6.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.14.6.4 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.

4.14.7 Service TRAIN_IS_IN_A_RADIO_HOLE

4.14.7.1 Parameter

Name	Type	Direction	Description

		on	
-	-	-	-

4.14.7.2 Returned value

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

4.14.7.3 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.

4.14.8 Complementary expected behaviour and usage of WRITE_MOBILE_CONTEXT, READ_RTM_MESSAGE and WRITE_RTM MESSAGE

4.14.8.1 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.

4.14.8.2 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.

4.14.9 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;
```

4.15 EVENT REPORT

4.15.1 Functional data flows (see /6/)

4.15.1.1 Input

- basic_to_generic_app_info .events_from_basic

4.15.1.2 Output

None.

4.15.2 Service EVENT_REPORT

4.15.2.1 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.

4.15.2.2 Parameter

Name	Type	Direction	Description
EVENT	API_TYPES.EVENT_T	in	The event that occurred on the target.

4.15.2.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.15.2.4 Expected behaviour & usage

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

4.15.3 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;
```

4.16 FAULTS (ERROR MANAGEMENT)

4.16.1 Functional data flows (see /6/)

4.16.1.1 Input

None.

4.16.1.2 Output

- generic_app_to_basic_info .channels_extinction_is_required

4.16.2 Service ERROR_MANAGEMENT

4.16.2.1 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, …).

4.16.2.2 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

4.16.2.3 Returned value

Name	Type	Direction	Description
-	-	-	-

4.16.2.4 Expected behaviour & usage

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

4.16.3 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;
```

5. APPENDIX 1 – APPLICATION LAYER (TELEGRAM DEFINITION)

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

6. APPENDIX 2 – FUNCTIONAL DATA DICTIONARY

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