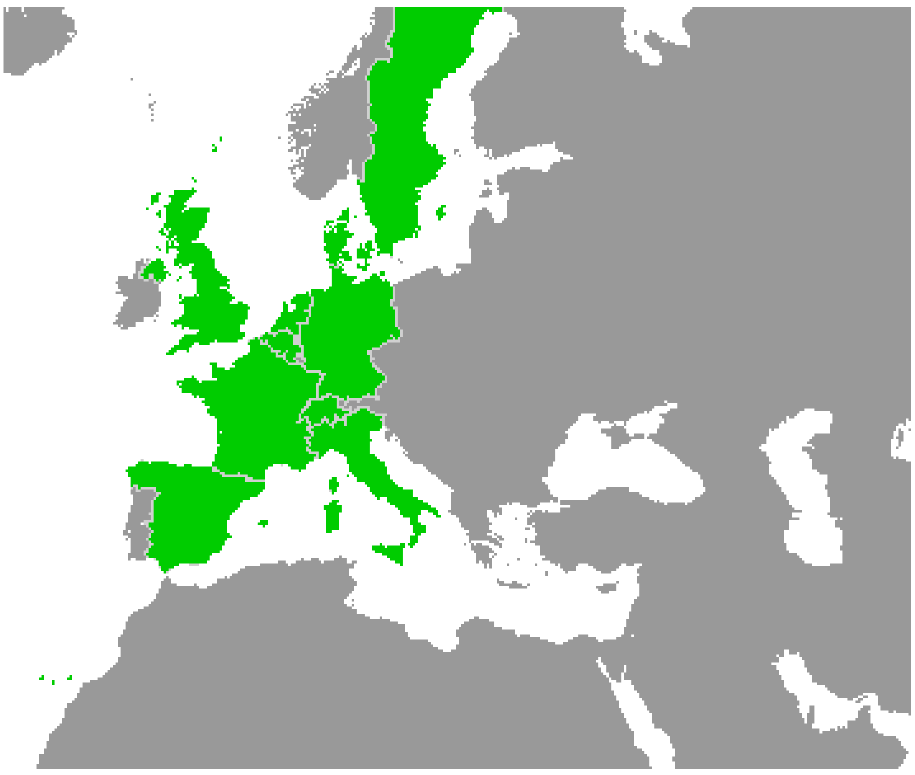
ITEA2 Project
2012-2015

Work-Package 2 : “Requirements”

API Requirements for OpenETCS – v1.0Draft

G. Guillaume

March 2013



AMENDMENT RECORD

Rev.[1]	Author	Version	Date	§	Modifications
	G. Guillaume	1.0Draft	01/03/2013	New	Draft version for review meeting plan on week1311

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

TABLE OF CONTENT

1. Introduction

1.1 Subject

This document explains how to use the services of the ERTMS Board Application interface.

It lists the different kinds of services with their characteristics, gives calling conventions and describes the parameters.

This document is the Application Programming Interface (API) Specification for ERTMS Core Application.

It defines the operational and functional requirements of the interface between the ERTMS CORE Board Application interface and the basic software, including:

- Entry routines the application provides.
- The basic software functions which the application should invoke.
- Information that the basic software team needs to know in order use the interface to the application software.

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

1.2 Field of application

This document is applicable to all programs using the ERTMS CORE Board Application interface.

1.3 Document description

This document presents the general description of the different kinds of services and presents the way to use them.

ADA specifications will be used to define the application entry points called by the monitor software. These have been written for use from the perspective of both the Application Software and the Basic Software Teams.
This document should be read in conjunction with the software requirements description for the ERTMS CORE software, [SRD], which provides more contextual and functional information.

The Application Programming Interface (API) Specification for ERTMS Core Application is implemented by two Ada packages Ertms_trainborn_generic_api which define all the services and Ertms_trainborn_generic_api_types which define the types needed to provide/receive data to/from the services. No other package of the ERTMS Core Application may be called by anyone outside the application.

2. Documents & terminology

2.1 Reference documents

/1/System Requirements Specification, ref. SUBSET-026, v2.3.0

/2/Software Requirement Document, ref. /BSI/Sr/GATC/M/SPEC/0070, [*]

2.2 Applicable documents

/1/Software Quality Plan, BSI/Sr/GATC/SYS/PLAN/0020 v1.3

2.3 Definitions

Airgap	:	Communication with devices external to the train (balise, radio)
Component	:	Package or set of packages corresponding to the model
Model	:	Documentation corresponding to a component
Module	:	Ada functional package or generic
Subsystem	:	Set of functionality's of the whole system
Binary message	:	Most of the time, objects have a data structure that includes a fixed length message of binary information's, from which only one variable part is used. We call "binary message" this array of binary information's

2.4 Abbreviations

ADB, ADS	:	Ada Body, Ada Specification (files extensions)
ADD	:	Architectural Design Document (such as this document)
API	:	Application Programming Interface
APP	:	Application
ASW	:	Application Software
BTM	:	Balise Transmission Module
BSW	:	Basic Software
ERTMS	:	European Railways Traffic Management System
EVC	:	European Vital Computer
JRU	:	Juridical Recorder Unit
KM	:	Key Manager
LLRU	:	Lowest Level Replaceable Unit
MGT	:	Management
MMI	:	Man Machine Interface
MMU	:	Movement Measurement Unit
NOVRAM	:	Non Volatile RAM
N/A	:	Not Applicable

[*] For applicable versions, please refer to BSI_Sr_GATC_sys_plan_0014_app4_540_yymmdd.xls, where dd/mm/yy is the date of the latest update.

PU	:	Power Up
RAM	:	Random Access Memory
RTM	:	Radio Transmission Module
SRD	:	Software Requirements Document
SRS	:	System Requirements Specification
STM	:	Specific Transmission Module
S/W	:	Software
TBD	:	To Be Defined
TIU	:	Train Interface Unit

3. INTERFACE SPECIFICATIONS

There are four types of procedure.

Those to allow the application to execute, those to provide input data and those to retrieve output data, as well as report faults, see figure 1.

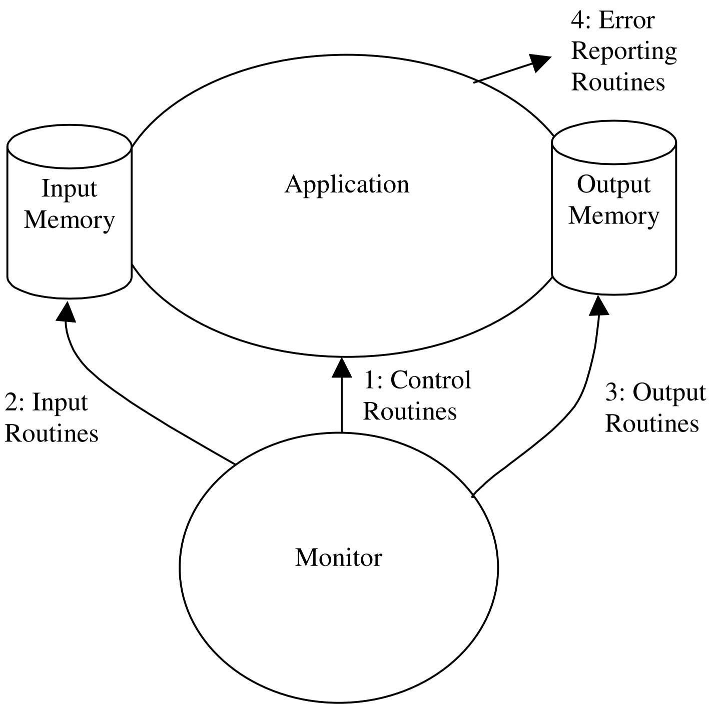


Figure 1: API Overview

After elaboration and initialisation of the basic software and the application, The Basic Software will start an infinite loop where he will first call the input routines, then call routine ACTIVATE to execute the application and finally he will call the output routines. It will restart this loop forever except if a fatal error occur. See figure 2.

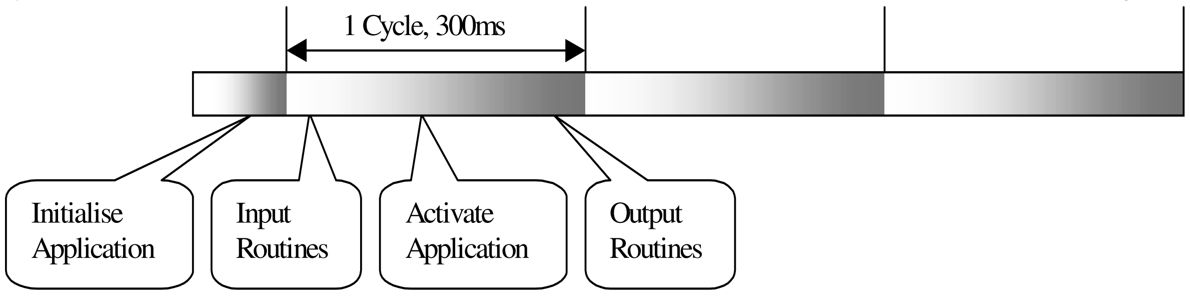


Figure 2: Basic Software Timeline

The ERTM board Application software has the following interfaces:

- Control
- Input/Output
  - Time

  - Configuration
  - Permanency
  - Events
  - Tests
  - Movement Management Unit (MMU)
  - Loop
  - Balise Transmission Module (BTM)
  - Key Manager
  - Radio Transmission Module (RTM)
  - Man Machine Interface (MMI)
  - Specific Transmission Module (STM)
  - Train Interface Unit (TIU)
  - Juridical Recorder Unit (JRU)
  - Diagnostic Recorder Unit (DRU)
  - LLRU Maintenance Data
  - Packets 44 for serial link
- Errors
- Data monitoring

3.1 Control

3.1.1 Description

The control interface allows the monitor to initialise and activate the application at the right times. Monitor activates the application on a cycle base, usually about 300ms. See figure 2 for a graphical description.
When the program is elaborated, initialisation services (INIT_DATA_MONITORING and then INITIALIZE) have to be called. Then the application configuration has to be performed (refer to §3.2.2 and §3.2.3 input services).
After the initialisation, at each cycle the monitor 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 then call the output routines to access the outputs data's and messages and apply or send them to the rest of the system.

The INIT_DATA_MONITORING provides an access to the procedure that shall be called when data to monitor are received. It returns as an ouput parameter the maximum size of a monitor data message.

3.1.2 API definitions

```
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;

  -- define the function to call to monitor the received data
  procedure INIT_DATA_MONITORING
   (WRITE_ACCESS : in     ERTMS_TRAINBORN_GENERIC_API_TYPES.MONITOR_DATA_ACCESS_T;
    MAX_SIZE     :    out TYPES.WORD_T);
.
.
.

end ERTMS_TRAINBORN_GENERIC_API;
```

3.2 Input/Output

These routines are called by the monitor. “Write” procedures with “in” parameters are called at the begin of the cycle to send input data to the application. “Read” procedures with “out” parameters or functions are called at the end of the cycle to receive computed data from the application.
It is permitted for the input routines to perform processing on the data passed to the application. However there are two issues to consider when designing a system that does this.

1) The order in which the basic software calls the application input routines is not defined. So, vital processing cannot be guaranteed to be performed first.
2) If the processing relies on other inputs, they may not yet have been updated for this cycle.

For these reasons, most processing should be performed when the application is activated. This is the way used by ERTMS CORE Board Application Software which store the non independent input data in buffer and process them during the Activate_Cycle.

- The input routines may be called in the order the caller want but all the inputs must be done before the Activate_Cycle.
- The following input services must be called before each Activate_Cycle.
  - WRITE_TIME
  - WRITE_MMU_DATA
- The outputs routine may be called in the order the caller want but a READ routine may be called only if the queue is not empty and all the output queue must be emptied before the end of the cycle except for the JRU output queue (see 3.2.11.1). For each output queue, there is a function to know if the queue is empty or not (xxx_MESSAGE_QUEUE_IS_EMPTY).

3.2.1 Time

3.2.1.1 Description

The time is provided by the basic software. Time is accurate to the hundredth of second and must increase.

3.2.1.2 API definition

```
-- 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);
.
.  -- procedure used by bsw to write the current cycle times
  procedure WRITE_STAT_COUNTERS(counters: in ERTMS_TRAINBORN_GENERIC_API_TYPES.STATISTIC_COUNTERS_T);
.

end ERTMS_TRAINBORN_GENERIC_API;
```

3.2.2 Configuration

3.2.2.1 Description

The configuration service should be called once before the initialisation control. It sends modifiable configuration parameters (plug data) to the ERTMS CORE Board Application Software.

3.2.2.2 API definition

```
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);

  -- procedure activated once at the initialisation to sent euroradio key management parameters
  procedure WRITE_KM_CONFIG (MANAGE_MANUAL_KEY_MGT_REQUEST : in TYPES.BOOLEAN_T;
                             DISPLAY_KEY_MGT_TEXT_MSGS     : in TYPES.BOOLEAN_T);
.
.
.
end ERTMS_TRAINBORN_GENERIC_API;
```

3.2.3 Permanency

3.2.3.1 Description

The WRITE_PERMANENT_DATA and WRITE_PROTECTED_PERMANENT_DATA services must be called once before the initialisation control. It sends the last application state saved before the Power Off to the ERTMS CORE Board Application Software.

The READ_PERMANENT_DATA and READ_PROTECTED_PERMANENT_DATA must be called at each cycle after the Activate_Cycle. The returned value must be stored in a non-volatile memory so it can be send to the application at the next Power Up.
At the first Power Up or after a non-volatile memory cleanup, an BYTE_DATA_T with SIZE set to 0 must be used as parameter to the WRITE_PERMANENT_DATA and WRITE_PROTECTED_PERMANENT_DATA services.

Remark : the protected permanent data cannot not be erase by a non_volatile memory cleanup.

3.2.3.2 API definition

```
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_ID (VALUE : ERTMS_TRAINBORN_GENERIC_API_TYPES.RADIO_NETWORK_ID_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;
```

3.2.4 Test

3.2.4.1 Description

The POWER_UP_TESTS_FINISHED service must be called once at the end of the power up test done by the BSW.

The START_BTM_TESTS_REQUIRED function should be called after each Activate_Cycle and the basic software must start a BTM antenna test if required.
When such a BTM antenna test is finished, the procedure BTM_TESTS_FINISHED must be called to give the test result to the application.

3.2.4.2 API definition

```
 -- 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;
```

3.2.5 MMU

3.2.5.1 Description

The MMU information is used by ERTMS CORE Board Api to determine the train position.
The application must be able to cope with odometry data that contains UNKNOWN fields.
The COORDINATE is that of the train.
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.

The COORDINATE and SPEED are measured over a time interval appropriate for the Basic Software but one TIME_STAMP and COORDINATE bounded values are provided at each cyles. The highest value, the lowest value and the nominal value (i.e. the most probable) are absolute values. TIME_STAMP is the MMU acquisition time. The SPEED is never negative.
The MOTION_DIRECTION indicates whether the train is moving with cab A first or cab B first (increasing or decreasing locations). The MOTION indicates whether or not the train is in motion.
If the motion can not 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 MMU_INFO provides some information stored on board : the line speed and gradient over the carriage where the EVC and the accelerometers are located (if not a 'one EVC per train configuration' → it is over the engine), the gradient over the whole train, the level of supervision, some information about the states and the requests of the emergency brakes and service brakes (also its delay for full application and minimum guaranteed deceleration when fully applied), the slippery character of the track and the state of the traction cut off and its current status computed by the TIU. It must be call at most once at each cycle.

The line speed field contains the minimum line speed over the line occupied by a part of the train (the engine or the carriage holding the EVC in an 'one EVC per train configuration'). The gradient field contains the minimum gradient over the line occupied by a part of the train (the engine or the carriage holding the EVC in an 'one EVC per train configuration' and the whole train). If the returned value is NO_VALUE for one field, it means that the corresponding information (line speed or gradient) is not known for all the wished train length area.

Several services allow to CORE Board Api to modify odometry parameters like wheel diameters and radar coefficients.

At initialisation, the WRITE_MMU_PARAMETERS service gives odometry parameters currently stored either in database or in novram and which must be used by ERTMS CORE Board Api. A parameter having a NOT_RELEVANT state mean that it is not present in database and must not be used.

At each cycle, the basic software call CHECK_MMU_PARAMETERS service in order to know if a check of odometry parameters shall be performed. If a check must be performed, the basic software read odometry parameters provided by ERTMS CORE Board Api through a call of READ_MMU_PARAMETERS service. The WRITE_MMU_PARAMETERS service returns result of this check by setting state of each parameters at RANGE_ERROR, CONSISTENCY_ERROR or VALID.

Furthermore, at each cycle the basic software call RECORD_MMU_PARAMETERS service in order to know if a store of odometry parameters shall be performed too. If a store must be performed, the basic software read odometry parameters provided by ERTMS CORE Board Api through a call of READ_MMU_PARAMETERS service and store in novram only the valid one.

3.2.5.2 API definition

```
---------------------------------------------------------------------------------------------------
-- MMU interface types

 -- 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 BOUNDED_SPEED_T is
  record
    NOMINAL_VALUE : SPEED_VALUE_T;
    UPPER_VALUE   : SPEED_VALUE_T;
    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);
-- Definition of the counter nominal_GAUGES
type MMU_COUNTER_NOMINAL_GAUGES_T is
  record
    ERROR_AFTER_0_METER    : MMU_COUNTER_T;
    ERROR_AFTER_100_METER  : MMU_COUNTER_T;
    ERROR_AFTER_1000_METER : MMU_COUNTER_T;
  end record;
-- Definition of the speed nominal_GAUGES
type MMU_SPEED_NOMINAL_GAUGES_T is
  record
    ERROR_AT_0_KMH      : SPEED_VALUE_T;
    SLOPE_AFTER_30_KMH : TYPES.REAL_T;
  end record;
-- Definition of the data related to the gradient sent to the SDMU
type GRADIENT_DATA_T is
  record
    GRADIENT_IS_AVAILABLE : TYPES.BOOLEAN_T;
    GRADIENT_OVER_TRAIN   : GRADIENT_VALUE_T;
    GRADIENT_ACC          : GRADIENT_VALUE_T;
  end record;
NUL_GRADIENT_DATA : constant GRADIENT_DATA_T :=
  (GRADIENT_IS_AVAILABLE => FALSE,
   GRADIENT_OVER_TRAIN   => GRADIENT_VALUE_T_FIRST,
   GRADIENT_ACC          => GRADIENT_VALUE_T_FIRST);
-- Definition of the data related to the Service brake sent to the SDMU
type SB_DATA_T is
  record
    SB_INTERVENTION_REQUESTED : TYPES.BOOLEAN_T;
    SB_APPLIED                : TYPES.BOOLEAN_T;
    SB_BRAKING_CAPACITY       : ACCELERATION_VALUE_T; -- REAL_T in m/s²   -- Not corrected by the adhesion coefficient !
    SB_APPLICATION_DELAY      : DURATION_VALUE_T; -- REAL_T in sec
  end record;
NUL_SB_DATA : constant SB_DATA_T :=
  (SB_INTERVENTION_REQUESTED => FALSE,
   SB_APPLIED                => FALSE,
   SB_BRAKING_CAPACITY       => (VALUE => 0.0),
   SB_APPLICATION_DELAY      => (VALUE => 0.0));
-- Definition of the data related to the Emergency brake sent to the SDMU
type EB_DATA_T is
  record
    EB_REQUESTED : TYPES.BOOLEAN_T := FALSE;
    EB_APPLIED   : TYPES.BOOLEAN_T := FALSE;
  end record;
```

```
NUL_EB_DATA : constant EB_DATA_T := (EB_REQUESTED => FALSE,
                                     EB_APPLIED   => FALSE);
-- Definition of the traction status computed by the TIU and sent to the SDMU
type TRACTION_STATUS_T is (NULL_STATE, POSITIVE, NEGATIVE, NOT_NULL, FAIL, NOT_AVAILABLE);
-- Definition of the data sent to the SDMU
type MMU_INFO_T is
  record
    LINE_SPEED       : SPEED_T;
    GRADIENT_DATA    : GRADIENT_DATA_T;
    SB_DATA          : SB_DATA_T;
    EB_DATA          : EB_DATA_T;
    SLIPPERY_TRACK   : TYPES.BOOLEAN_T;
    TRACTION_CUT_OFF : TYPES.BOOLEAN_T;
    TRACTION_STATUS  : TRACTION_STATUS_T;
  end record;
NUL_MMU_INFO : constant MMU_INFO_T :=
  (LINE_SPEED       => (QUALITY => NO_VALUE),
   GRADIENT_DATA    => NUL_GRADIENT_DATA,
   SB_DATA          => NUL_SB_DATA,
   EB_DATA          => NUL_EB_DATA,
   SLIPPERY_TRACK   => FALSE,
   TRACTION_CUT_OFF => FALSE,
   TRACTION_STATUS  => NOT_AVAILABLE);


.
.
.

subtype UNSIGNED11_T is TYPES.UNSIGNED_WORD_T range 0 .. 2047;
 UNSIGNED11_DEF_C : constant UNSIGNED11_T := 0;

 subtype INTEGER9_T is TYPES.WORD_T range -(2**8) .. (2**8) - 1;
 INTEGER9_DEF_C : constant INTEGER9_T := 0;

 subtype RADAR_COEFF_T is TYPES.LONG_T range 0 .. 100_000;
 RADAR_COEFF_DEF_C : constant RADAR_COEFF_T := 10_000; -- 100%

  -- LOCAL time in SECOND since 01/01/2000 00:00:00.000
  type LOCAL_TIME_VALUE_T is new TYPES.UNCHECKED_LONG_T;
  LOCAL_TIME_VALUE_DEF_C : constant LOCAL_TIME_VALUE_T := 0;

  type LOCAL_TIME_QUALIFIER_T is (VALUE, NO_VALUE);
  LOCAL_TIME_QUALIFIER_DEF_C : constant LOCAL_TIME_QUALIFIER_T := NO_VALUE;

  type LOCAL_TIME_T (QUALITY : LOCAL_TIME_QUALIFIER_T := NO_VALUE) is
    record
      case QUALITY is
        when VALUE =>
          VALUE : LOCAL_TIME_VALUE_T;
        when NO_VALUE =>
          null;
      end case;
    end record;

  LOCAL_TIME_DEF_C : constant LOCAL_TIME_T := (QUALITY => LOCAL_TIME_QUALIFIER_DEF_C);

  type DATA_STATUS_T is (NOT_RELEVANT, RANGE_ERROR, CONSISTENCY_ERROR, VALID);
  DATA_STATUS_DEF_C : constant DATA_STATUS_T := NOT_RELEVANT;

  type WHEEL_DIAMETER_DATA_T is
   record
     STATUS : DATA_STATUS_T;
     VALUE  : UNSIGNED11_T;
  DATE   : LOCAL_TIME_T;
 end record;
WHEEL_DIAMETER_DATA_DEF_C : constant WHEEL_DIAMETER_DATA_T := (STATUS => DATA_STATUS_DEF_C,
                                                VALUE  => UNSIGNED11_DEF_C,
                                                DATE   => LOCAL_TIME_DEF_C);

type M_INTER_COEF_DATA_T is
 record
  STATUS : DATA_STATUS_T;
  VALUE  : INTEGER9_T;
  DATE   : LOCAL_TIME_T;
 end record;
M_INTER_COEF_DATA_DEF_C : constant M_INTER_COEF_DATA_T := (STATUS => DATA_STATUS_DEF_C,
                                              VALUE  => INTEGER9_DEF_C,
                                              DATE   => LOCAL_TIME_DEF_C);

type M_DOPPLER_COEF_DATA_T is
 record
  STATUS : DATA_STATUS_T;
  VALUE  : INTEGER9_T;
  DATE   : LOCAL_TIME_T;
 end record;
M_DOPPLER_COEF_DATA_DEF_C : constant M_DOPPLER_COEF_DATA_T := (STATUS => DATA_STATUS_DEF_C,
                                                VALUE  => INTEGER9_DEF_C,
                                                DATE   => LOCAL_TIME_DEF_C);

type RADAR_COEF_DATA_T is
 record
  STATUS : DATA_STATUS_T;
  VALUE  : RADAR_COEFF_T;
  DATE   : LOCAL_TIME_T;
 end record;
RADAR_COEF_DATA_C : constant RADAR_COEF_DATA_T := (STATUS => DATA_STATUS_DEF_C,
                                       VALUE  => RADAR_COEFF_DEF_C,
                                       DATE   => LOCAL_TIME_DEF_C);

 type ACC_BIAS_COEF_DATA_T is
   record
     STATUS : DATA_STATUS_T;
     VALUE  : ACC_BIAS_COEFF_T;
     DATE   : LOCAL_TIME_T;
   end record;

 subtype ACC_BIAS_COEFF_T is TYPES.WORD_T range -200 .. 200;
 ACC_BIAS_COEFF_C : constant ACC_BIAS_COEFF_T := 0; -- 0 mm/s2

 ACC_BIAS_COEF_DATA_C : constant ACC_BIAS_COEF_DATA_T := (STATUS => DATA_STATUS_DEF_C,
                                             VALUE  => ACC_BIAS_COEFF_C,
                                             DATE   => LOCAL_TIME_DEF_C);

type MMU_PARAMETERS_T is
  record
   WHEEL_DIAMETER_A  : WHEEL_DIAMETER_DATA_T;
   WHEEL_DIAMETER_B  : WHEEL_DIAMETER_DATA_T;
   M_INTER_COEF_A    : M_INTER_COEF_DATA_T;
   M_INTER_COEF_B    : M_INTER_COEF_DATA_T;
   M_DOPPLER_COEF_A1 : M_DOPPLER_COEF_DATA_T;
   M_DOPPLER_COEF_B1 : M_DOPPLER_COEF_DATA_T;
   RADAR_1_COEF      : RADAR_COEF_DATA_T;
   RADAR_2_COEF      : RADAR_COEF_DATA_T;
   ACC_BIAS_COEF     : ACC_BIAS_COEF_DATA_T;
  end record;
 MMU_PARAMETERS_DEF_C : constant MMU_PARAMETERS_T
:= (WHEEL_DIAMETER_A  => WHEEL_DIAMETER_DATA_DEF_C,
   WHEEL_DIAMETER_B  => WHEEL_DIAMETER_DATA_DEF_C,
   M_INTER_COEF_A    => M_INTER_COEF_DATA_DEF_C,
   M_INTER_COEF_B    => M_INTER_COEF_DATA_DEF_C,
   M_DOPPLER_COEF_A1 => M_DOPPLER_COEF_DATA_DEF_C,
   M_DOPPLER_COEF_B1 => M_DOPPLER_COEF_DATA_DEF_C,
   RADAR_1_COEF => RADAR_COEF_DATA_C,
   RADAR_2_COEF => RADAR_COEF_DATA_C,
   ACC_BIAS_COEF     => ACC_BIAS_COEF_DATA_C);

 type MMU_PARAMETERS_KIND_T is (WHEEL_DIAMETERS, RADAR_COEFFICIENTS, SDMU_RADAR_COEFFICIENTS);
 type MMU_PARAMETERS_BY_KIND_T (KIND : MMU_PARAMETERS_KIND_T := MMU_PARAMETERS_KIND_T'FIRST) is
   record
     case KIND is
       when ERTMS_TRAINBORN_GENERIC_API_TYPES.WHEEL_DIAMETERS =>
         WHEEL_DIAMETER_A : WHEEL_DIAMETER_DATA_T;
         WHEEL_DIAMETER_B : WHEEL_DIAMETER_DATA_T;
       when ERTMS_TRAINBORN_GENERIC_API_TYPES.RADAR_COEFFICIENTS =>
         M_INTER_COEF_A    : M_INTER_COEF_DATA_T;
         M_INTER_COEF_B    : M_INTER_COEF_DATA_T;
         M_DOPPLER_COEF_A1 : M_DOPPLER_COEF_DATA_T;
         M_DOPPLER_COEF_B1 : M_DOPPLER_COEF_DATA_T;
       when ERTMS_TRAINBORN_GENERIC_API_TYPES.SDMU_RADAR_COEFFICIENTS =>
         RADAR_1_COEF      : RADAR_COEF_DATA_T;
         RADAR_2_COEF      : RADAR_COEF_DATA_T;
        when ACC_BIAS_COEFFICIENT =>
          ACC_BIAS_COEF     : ACC_BIAS_COEF_DATA_T;
     end case;
   end record;
 MMU_PARAMETERS_BY_KIND_DEF_C : constant MMU_PARAMETERS_BY_KIND_T := (KIND => MMU_PARAMETERS_KIND_T'FIRST,
                                   WHEEL_DIAMETER_A => WHEEL_DIAMETER_DATA_DEF_C,
                                   WHEEL_DIAMETER_B => WHEEL_DIAMETER_DATA_DEF_C);



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.BOUNDED_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);

 function MMU_INFO return ERTMS_TRAINBORN_GENERIC_API_TYPES.MMU_INFO_T;

.
.
.
 function CHECK_MMU_PARAMETERS return TYPES.BOOLEAN_T;
 function RECORD_MMU_PARAMETERS return TYPES.BOOLEAN_T;
 procedure READ_MMU_PARAMETERS
```

```
  (MMU_PARAMETERS : out ERTMS_TRAINBORN_GENERIC_API_TYPES.MMU_PARAMETERS_BY_KIND_T);
 procedure WRITE_MMU_PARAMETERS
  (MMU_PARAMETERS : in ERTMS_TRAINBORN_GENERIC_API_TYPES.MMU_PARAMETERS_T);

end ERTMS_TRAINBORN_GENERIC_API;
```

3.2.6 BTM

3.2.6.1 Description

The READ_BTM_INFO function returns the type of modulation to use for the BTM antenna and the antenna the basic software has to active according to the current active cab. It must be called after each Activate_Cycle.

The WRITE_BTM_INFO procedure is used to provide information coming from the track that has been received from the balise. This procedure must be called once per balise with the last information received. The WRITE_BTM_INFO procedure could be called multiple times in one cycle if multiple balises has been crossed during this cycle, so the application should bufferize the data it receives.
The LOCATION is that of the centre of the balise. The co-ordinate types use the same frame of reference as the odometry – so they may be compared.
There is an uncertainty in the balise centre measurement described by LOCATION_ACCURACY. The TELEGRAM is the data received from the Balise. It has not yet been decoded.
The TIME_STAMP field on the Balise info is the time the basic software loses the contact with the balise. This Time stamp is in the same reference as the time used in the WRITE_TIME service.
The USED_ANTENNA field on the Balise info is the identification of the BTM antenna who has read the telegram.

The WRITE_BTM_ANTENNA procedure is used to inform the application about the current BTM antenna in service. The NONE value means none of the antenna are usable.
This procedure must be called once at the end of BSW initialisation and at each antenna change.

The METAL_MASS_INFO function should be called after each Activate_Cycle and
the basic software must manage the BTM antenna state and test following this information.
If the BTM acquisition device detect a balise with integrity problem (bad CRC, …) the BAD_BALISE_RECEIVED service must be called.

3.2.6.2 API definition

The Ada definition is as follows:

```
 ---------------------------------------------------------------------------------------------------
 -- 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;
```

3.2.7 RTM

3.2.7.1 Description

Actually, the board is the "master" in the radio communication management. It's the board, which take the initiative to open or close a communication.
There are two types of RTM services.

- Services to know about radio environment
- Services to manage the radio connections and messages

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 BSW has to wait the end of the radio hole before to try to restore it.
The service NUMBER_OF_HANDABLE_RTM_COMMUNICATION_SESSION is used by the application to know the number of communication session, which are possible simultaneously. This value is dynamic according to the hardware equipment status but once it reaches 0 it never increase anymore.
The WRITE_MOBILE_CONTEXT is used by the application to know the status of each mobile and associated network ID.
The NETWORK_REQUEST is used to request the mobiles registration to given a network ID .
The CONNECTION_REQUEST is used to open a connection and the DISCONNECTION_REQUEST is used to close then. Only one communication may be established with a RADIO_DEVICE at the same time.
If the basic software has establish the connection with the required RADIO_DEVICE a CONNECTION_CONFIRMATION is returned to the application. Otherwise a CONNECTION_FAILURE is returned.
As soon as an established connection is seen as broken by the basic software, it shall send a CONNECTION_LOST. An INFINITE_CONNECTION_RETRIES message shall then be send by the application software to indicate to the expected behavior. If after three attemps it is not able to re-establish the connection, a CONNECTION_FAILURE (finite retries configuration) or CONNECTION_NOT_RE_ESTABLISHED (infinite retries configuration) with the origin of the problem shall be returned to the application software.
If the radio connection is opened but at the application level, it seems the be lost (no application message received during a given time) a RESET_CONNECTION is send to the BSW. 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, the basic software may delete the message. By usable i mean the connection is established and the basic software is not attempt to reconnect it.
So there is no obligation of delivery of a message by the BSW. If a radio message is lost, the sender will not receive the answer and will repeat it.

3.2.7.2 API definition

```
----------------------------------------------------------------------------------------------
-- 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));

-- FAILED          : mobile out of order
-- NETWORK_REQUEST : mobile with a network-request in progress (NOT_REGISTERED)
-- NETWORK_CONFIRM : mobile has received a network-confirm and no communication is running (REGISTERED)
-- BUSY            : mobile with a communication running (IN COMMUNICATION)
```

```
type MOBILE_STATE_T is (FAILED, REGISTER_REQUEST, REGISTER_CONFIRM, BUSY);

type MOBILE_CONTEXT_T is
  record
    STATE   : MOBILE_STATE_T;
    NETWORK : 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));


-- 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 re-connection retries is not infinite)
-- CONNECTION_NOT_RE_ESTABLISHED : to send when it has not been possible to  re-establish the
                                connection after 3 attempts
                                (if re-connection retries is infinite)
-- DATA : euroradio data message from trackside
type RTM_IN_MESSAGE_KIND_T is (CONNECTION_CONFIRMATION,
                              CONNECTION_LOST,
                              CONNECTION_FAILURE,
                              CONNECTION_NOT_RE_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_RETRIES : order to try to reconnect infinitely after a connection loss
                               or a reset connection
-- DATA                       : euroradio data message from trainborn
type RTM_OUT_MESSAGE_KIND_T is (NETWORK_REQUEST,
                               CONNECTION_REQUEST,
                               DISCONNECTION_REQUEST,
                               RESET_CONNECTION,
                               INFINITE_CONNECTION_RETRIES, DATA);


-- Origin of a connection failure =>
-- AUTHENTIFICATION_FAILURE : the anomaly is due to a KMAC problem
-- TRACK                    : the anomaly is due to the track side.
-- BOARD                    : the anomaly is due to the board side.
-- UNDEFINED                : impossible to define the anomaly origin.
-- UNKNOWN                  : no reason/subreadon corresponding to a known anomaly.
```

```
type ORIGIN_T is (AUTHENTIFICATION_FAILURE,
                  TRACK,
                  BOARD,
                  UNDEFINED,
                  UNKNOWN);


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
         | CONNECTION_LOST =>
        null;

      when CONNECTION_FAILURE
         | CONNECTION_NOT_RE_ESTABLISHED =>
        ORIGIN : ORIGIN_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_ID : RADIO_NETWORK_ID_T;

      when CONNECTION_REQUEST =>
        RADIO_NUMBER : NIDRADIO_T;

      when DISCONNECTION_REQUEST =>
        null;

      when RESET_CONNECTION =>
        null;

      when INFINITE_CONNECTION_RETRIES =>
        null;

      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;
```

3.2.8 MMI

3.2.8.1 Description

The interface to the MMI contains only read and write services, the Basic software has to establish the connection with the MMI(s) during the initialisation of the system and maintain it during the system live. A cabin is always associated with a MMI message.
Once the connection is established, the CONNECTED message must be received in the WRITE_MMI_MESSAGE procedure.
If the connection cannot be established or an established connection is lost, a disconnected message must be received in WRITE_MMI_MESSAGE procedure. Depending on the type of disconnection, the message will be TEMPORARY_DISCONNECTED or DISCONNECTED.
DATA messages must only be used if the concerned MMI is connected.
A DISCONNECTION_REQUEST message is send when the application software want to close definitively the connection with a MMI.

3.2.8.2 API definition

```
------------------------------------------------------------------------------------------------
-- 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;


-- Driver language from Basic software
-- define the mmi language selected by the driver
type NID_DRV_LANG_T is array (1..2) of TYPES.ASCII_T;

-- define the mmi language receied from BSW
type DRIVER_LANGUAGE_T (QUALITY : VALUE_QUALIFIER_T := NO_VALUE) is
  record
    case QUALITY is
      when VALUE =>
        VALUE : NID_DRV_LANG_T;
      when NO_VALUE =>
        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;

.  -- this function is called to write the mmi selected language
procedure WRITE_MMI_DRV_SELECTED_LANGUAGE
       (DRV_LANG: in ERTMS_TRAINBORN_GENERIC_API_TYPES.DRIVER_LANGUAGE_T);

-- this function is called by the BSW to read the speed displayed on the DMI
function MMI_DISPLAYED_NUMERICAL_SPEED return ERTMS_TRAINBORN_GENERIC_API_TYPES.SPEED_T;

-- this function is called by the BSW it returns TRUE if the numerical speed readback shall be active
function MMI_SPEED_READBACK_IS_ACTIVE return TYPES.BOOLEAN_T;
.
.
end ERTMS_TRAINBORN_GENERIC_API;
```

3.2.9 STM

3.2.9.1 Description

The interface to the STM contains only read and write services, the Basic software has to establish the connection with the STM(s) during the initialisation of the system and maintain it during the system live. An ID is always associated with a STM message.
Once the connection is established, the CONNECTED message must be received in the WRITE_STM_CONTROL_MESSAGE or WRITE_STM_SPECIFIC_MESSAGE procedures.
If the connection cannot be established or an established connection is lost, a disconnected message must be received in WRITE_STM_CONTROL_MESSAGE and WRITE_STM_SPECIFIC_MESSAGE procedures. Depending on the type of disconnection, the message will be TEMPORARY_DISCONNECTED or DISCONNECTED.
DATA messages must only be used if the concerned STM is connected.
A DISCONNECTION_REQUEST message is send when the application software want to close the connection with a STM.

The function STM_INFO returns to the basic software the list of connected STM (from an application point of view) with their associated state.

3.2.9.2 API definition

```
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;
.
.
  ---------------------------------------------------------------------------------------------------
  -- STM specific services
  ---------------------------------------------------------------------------------------------------
  -- procedure to deliver STM specific message to the application
  procedure WRITE_STM_SPECIFIC_MESSAGE (THE_STM_MESSAGE : in
ERTMS_TRAINBORN_GENERIC_API_TYPES.STM_SPECIFIC_IN_MESSAGE_T);

  -- procedure to read STM specific message from the application
  -- this procedure may be called only if STM_SPECIFIC_MESSAGE_QUEUE_IS_EMPTY returns FALSE
  procedure READ_STM_SPECIFIC_MESSAGE (THE_STM_MESSAGE : out
ERTMS_TRAINBORN_GENERIC_API_TYPES.STM_SPECIFIC_OUT_MESSAGE_T);

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

```
 -- and returns FALSE otherwise
 function STM_SPECIFIC_MESSAGE_QUEUE_IS_EMPTY return TYPES.BOOLEAN_T;
.
end ERTMS_TRAINBORN_GENERIC_API;
```

3.2.10 TIU

3.2.10.1 Description

The interface to the TIU contains only read, write and error services, the Basic software has to establish the connection with the TIU during the initialisation of the system and maintain it during the system live. If the basic software is no more able to maintain the communication, it has to stop the system.
If the function EVC_ISOLATION_IS_REQUESTED returns TRUE The Basic software has to "isolate" the EVC from the train.
The function OCCUPIED_CABINE returns the current active cab wich has to be used as target MMI for the BSW mmi communications.

3.2.10.2 API definition

```
------------------------------------------------------------------------------------------------
-- 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;
.
.
.
end ERTMS_TRAINBORN_GENERIC_API;
```

3.2.11 JRU

3.2.11.1 Description

The interface to the JRU contains only read and write services, the Basic software has to establish the connection with the JRU during the initialisation of the system and maintain it during the system live.
Once the connection is established, the CONNECTED message must be received in the WRITE_JRU_MESSAGE procedure.
If the connection cannot be established or an established connection is lost, a disconnected message must be received in WRITE_JRU_MESSAGE procedure. Depending on the type of disconnection, the message will be TEMPORARY_DISCONNECTED or DISCONNECTED.
DATA messages must only be used if the concerned JRU is connected. The basic software is allowed to not empty the output queue immediately if the communication is not yet established or if it is currently saturated. This is an exception where an output queue may be not empty at the end of a cycle. (The application output queue will be designed to take this in account).
A DISCONNECTION_REQUEST message is send when the application software want to close definitively the connection with a JRU.

3.2.11.2 API definition

```
-----------------------------------------------------------------------------------------------
-- 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;
```

3.2.12 DRU

3.2.12.1 Description

The interface to the DRU contains only read and write services, the Basic software has to establish the connection with the DRU during the initialisation of the system and maintain it during the system live.
Once the connection is established, the CONNECTED message must be received in the WRITE_DRU_MESSAGE procedure.
If the connection cannot be established or an established connection is lost, a disconnected message must be received in WRITE_DRU_MESSAGE procedure. Depending on the type of disconnection, the message will be TEMPORARY_DISCONNECTED or DISCONNECTED.
DATA messages must only be used if the concerned DRU is connected.
A DISCONNECTION_REQUEST message is send when the application software want to close definitively the connection with a JRU.
Currently, there is no DRU -> EVC message defined, so the WRITE_DRU_MESSAGE is never used. It has been written for design purpose and future use.

A specific service "READ_DRU_ETCS_CONTEXT" exists to allow to build a DRU packet 3 to send to train tracer function via the CORE Basic Software.

3.2.12.2 API definition

```
----------------------------------------------------------------------------------------------------
-- DRU types
----------------------------------------------------------------------------------------------------

 type DRU_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.DRU_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 DRU_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_DRU_BIT_STREAM_T;

      when INTERFACE_LANGUAGE_TYPES.DISCONNECTION_REQUEST =>
        null;

    end case;
  end record;

 type DRU_ETCS_CONTEXT_T is
  record
    LRBG                : BALISE_GROUP_ID_T;
    DISTANCE            : DISTANCE_T;
    TRAIN_ORIENTATION   : DIRECTION_VALUE_T;
    FRONT_END_POSITION  : DIRECTION_VALUE_T;
    DOUBT_OVER          : DISTANCE_T;
    DOUBT_UNDER         : DISTANCE_T;
    SPEED               : SPEED_VALUE_T;
    SPEED_DIRECTION     : DIRECTION_VALUE_T;
    MODE                : MODE_T;
    LEVEL               : LEVEL_T;
    ACTIVE_CAB          : OCCUPIED_CABINE_T;
    ACTIVE_ANTENNA      : USED_ANTENNA_T;
    EQUIPMENT_ID        : TYPES.POSITIVE_LONG_T;
  end record;


package ERTMS_TRAINBORN_GENERIC_API is
.
.
 -----------------------------------------------------------------------------------------------
 -- DRU services
 -----------------------------------------------------------------------------------------------
 -- procedure to deliver JRU message to the application
 procedure WRITE_DRU_MESSAGE
  (THE_DRU_MESSAGE : in ERTMS_TRAINBORN_GENERIC_API_TYPES.DRU_IN_MESSAGE_T);

 -- procedure to read DRU message from the application
 -- this procedure may be called only if DRU_MESSAGE_QUEUE_IS_EMPTY returns FALSE
 procedure READ_DRU_MESSAGE
  (THE_DRU_MESSAGE : out ERTMS_TRAINBORN_GENERIC_API_TYPES.DRU_OUT_MESSAGE_T);

 -- this function returns TRUE if the output DRU MESSAGE QUEUE IS EMPTY
 -- and returns FALSE otherwise
 function DRU_MESSAGE_QUEUE_IS_EMPTY return TYPES.BOOLEAN_T;

 -----------------------------------------------------------------------------------------------
 -- DRU specific services
 ---------------------------------------------------------------------------------------------------
 procedure READ_DRU_ETCS_CONTEXT
   (CONTEXT : out ERTMS_TRAINBORN_GENERIC_API_TYPES.DRU_ETCS_CONTEXT_T);

.
.
end ERTMS_TRAINBORN_GENERIC_API;
```

3.2.13 LLRU Maintenance Data

3.2.13.1 Description

The WRITE_LLRU_STATUS procedure must be called to deliver the coded LLRU status list to the ASW. It must be activated at initialisation of the system (after the reception of the configuration and the permanent data) and at each modification of a LLRU status.

The LLRU_RESET_REQUESTED function must be called to known if the ASW request a reset of the LLRU status list. This request could be taken into account at the next power-on.

3.2.13.2 API Definition

```
---------------------------------
-- LLRU Maintenance Data Types --
---------------------------------
MAX_N_OF_LLRU_C        : constant := 160;
SIZE_OF_LLRU_STATUS_C  : constant := 3;   -- 3bits for one LLRU status

-- coded LLRU status/health list
subtype LLRU_STATUS_LIST_T is TYPES.BITS_T (1 .. MAX_N_OF_LLRU_C * SIZE_OF_LLRU_STATUS_C);

package ERTMS_TRAINBORN_GENERIC_API is
.
.
.
 --------------------------
 -- LLRU Maintenance Data --
 --------------------------
 -- procedure to deliver the coded LLRU status list
 -- activate at initialisation of the system (power up) and at each modification of a LLRU status
 procedure WRITE_LLRU_STATUS (DATA : in ERTMS_TRAINBORN_GENERIC_API_TYPES.LLRU_STATUS_LIST_T);

 -- Return TRUE if a reset of the LLRU status list is requested
 function LLRU_RESET_REQUESTED return TYPES.BOOLEAN_T;
.
.
.
end ERTMS_TRAINBORN_GENERIC_API;
```

3.2.14 Packet_44 for serial link

3.2.14.1 Description

The READ_PACKET_44_MESSAGE must be called at each cycle after the Activate_Cycle. The returned value must be sent onto the serial link (RS 485) by the basic software.

3.2.14.2 API Definition

```
package ERTMS_TRAINBORN_GENERIC_API_TYPES is
.
.
.
  -----------------------------------------------------------------------------------------
  -----  PACKET_44 MESSAGES ---------------------------------------------------------------
  -----------------------------------------------------------------------------------------

  subtype PACKET_44_MESSAGE_RANGE_T is TYPES.LONG_T range 0 .. MAX_PACKET_44_MESSAGE_LENGTH_C;
  type PACKET_44_OUT_T is
    record
      LENGTH  : PACKET_44_MESSAGE_RANGE_T := PACKET_44_MESSAGE_RANGE_T'LAST;
      MESSAGE : TYPES.BYTE_STRING_T (1 .. TYPES.NEW_INDEX_T(PACKET_44_MESSAGE_RANGE_T'LAST));
    end record;

.
.
.
end ERTMS_TRAINBORN_GENERIC_API_TYPES;

package ERTMS_TRAINBORN_GENERIC_API is

---------------------------------------------------------------------------------------------
  -- PACKET_44 specific services
  -------------------------------------------------------------------------------------------

  -- procedure to read PACKET 44 message from the application
  -- this procedure may be called only if PACKET_44_QUEUE_IS_EMPTY returns FALSE
  procedure READ_PACKET_44_MESSAGE (THE_PACKET_44 : out   ERTMS_TRAINBORN_GENERIC_API_TYPES.PACKET_44_OUT_T);

  -- this function returns TRUE if the output PACKET 44 QUEUE IS EMPTY
  -- and returns FALSE otherwise
  function PACKET_44_MESSAGE_QUEUE_IS_EMPTY return TYPES.BOOLEAN_T;.
.
.
 end ERTMS_TRAINBORN_GENERIC_API;
```

3.2.15 Events

3.2.15.1 Description

The EVENT_REPORT procedure must be used by the BSW to inform the ASW of an event leading to actions which can only be performed by the ASW (ex : display of a text message on the DMI). This procedure must be called once when the event is detected.

3.2.15.2 API Definition

```
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;
```

3.2.16 Loop

3.2.16.1 Description

The WRITE_LOOP_MESSAGE procedure is used to provide information coming from the track that has been received from a loop. If the received loop message is the same as the previous one transmits to the ASW, the message kind shall be SAME. The procedure could be called multiple times in one cycle if multiple message have been received during this cycle.

The SS_CODE_FOR_LOOP function returns the spread spectrum code of the expected loop.

3.2.16.2 API Definition

```
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;
```

3.2.17 Key Manager

3.2.17.1 Description

The WRITE_KM_INFO procedure is used to provide information coming from the key manager functionnality in the BSW. It supplies a table of events and shall be call each time an event occurs. If several events occur in the same cycle, each occurred events shall be set at TRUE.

The KM_REQUEST function returns ASW requests linked to the keys for the BSW. It shall be checked at each cycle.

3.2.17.2 API Definition

```
package ERTMS_TRAINBORN_GENERIC_API_TYPES is
.
.
.
 ---------------------------------------------------------------------------------------------------
 -- Key Manager types
 ---------------------------------------------------------------------------------------------------

 -- Events linked to the euroradio key management to be managed by the application
 type KM_EVENT_T is
   (DIALOGUE_WITH_KMC_NOT_POSSIBLE,
    DIALOGUE_WITH_KMC_POSSIBLE,
    DIALOGUE_WITH_KMC_ON_GOING,
    DIALOGUE_WITH_KMC_FAILURE,
    KEY_MGT_INFO_UPDATED,
    KEY_DB_UPDATED);

  -- Table of events to manage several events at once
  type KM_EVENT_TABLE_T is array (KM_EVENT_T) of TYPES.BOOLEAN_T;

  -- Requests linked to the euroradio keys management
  -- NONE    : no request
  -- UPDATE  : indicates if the update of the key database has been requested
  -- INSTALL : indicates if the complete (re-)installation of the key database has been requested
  type KM_REQUEST_T is
   (NONE,
    UPDATE,
    INSTALLATION);
.
.
end ERTMS_TRAINBORN_GENERIC_API_TYPES;

package ERTMS_TRAINBORN_GENERIC_API is
.
.
.
  ----------------------------------------------------------------------------------------------------
  -- Key Manager services
  ----------------------------------------------------------------------------------------------------

  -- This procedure has to be called when one or more events defined in
  -- ERTMS_TRAINBORN_GENERIC_API_TYPES.KM_EVENT_T occur in the euroradio key management
  procedure WRITE_KM_INFO (THE_KM_INFO : in ERTMS_TRAINBORN_GENERIC_API_TYPES.KM_EVENT_TABLE_T);

  -- This function has to be called at each cycle to check requests linked to the euroradio keys management
  function KM_REQUEST return ERTMS_TRAINBORN_GENERIC_API_TYPES.KM_REQUEST_T;
.
.
.
end ERTMS_TRAINBORN_GENERIC_API;
```

3.2.17.3 Brakes services

3.2.17.4 Description

The EB_REQUESTED function is used to provide the application emergency brakes state order to the BSW.

3.2.17.5 API Definition

```
package ERTMS_TRAINBORN_GENERIC_API is
.
.
.
  ----------------------------------------------------------------------------------------------------
  -- 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;
```

3.3 Faults

3.3.1 *Description*

If an error is detected within the application and the application wishes to report it, an explicitly defined ATBL_ERROR.INFO_ERROR_T will be used. The names of these errors are to be determined by the design.
The BSW provided an error reporting routine ERROR_MANAGEMENT that requires specific error codes and reaction to be defined. When this ERROR_MANAGEMENT routine is called, the BSW has to take the predefined actions (memorisation, brake application, shutdown of the system, …).

3.3.2 *API definition*

```
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;
```

3.4 Data monitoring

3.4.1 *Description*

The BSW provide a service to the application to monitor some critical data.
This function has two goals
1. Monitoring all the inputs and the outputs of the application to be able to understand them and replay a recorded scenario.
2. Comparing the outputs of the application to be sure that there is no difference between the three channel of the system.

These data will be sent to a spy and / or voted according to the parameters values.

3.4.2 *API definition*

```
package MONITOR_DATA is
.
.
.
  procedure WRITE (
          MESSAGE       : in TYPES.BYTE_STRING_T;
          SEND_TO_VOTE    : in TYPES.BOOLEAN_T;
          SEND_TO_SPY     : in TYPES.BOOLEAN_T := TRUE
         );
.
.
.
end MONITOR_DATA;
```

3.5 OTHER TYPES INTERFACE

There is one types package that is common to the Application and the Basic Software : Ertms_Trainborn_Generic_Api_Types. It contains data definitions that are as far as useful strongly typed. This packages only depends on high level types packages to avoid dependencies between ASW and BSW.