ERTMSFormalSpecs User Guide

Version 1.0.0

1 Table of contents

2 Table of Images:

3 Index of tables

4 Table of equations

5 Revision history

Version	Date	Name	Description	Paragraphs
0.2.1	15/12/2010	Stanislas Pinte	First version	All
0.2.2	15/12/2010	Laurent Ferier	Document revision	All
0.2.3	06/01/2011	Svitlana Lukicheva	Added in/out variables	5.3, 6.4
0.3.1	28/01/2011	Laurent Ferier	Document revision according to release 0.3	All
0.3.2	21/02/2011	Svitlana Lukicheva	Added chapter "Reports"	8
0.4.0	04/03/2011	Laurent Ferier	Document revision according to release 0.4	All
0.5.0	05/04/2011	Laurent Ferier	Document revision according to release 0.5	All
0.6.0	13/05/2011	Svitlana Lukicheva, Laurent Ferier	Document revision according to release 0.6	All
0.7.0	17/10/2012	Svitlana Lukicheva, Laurent Ferier	Document revision according to release 0.7	All
1.0.0	15/01/2014	Luis Marcos, Laurent Ferier	Document revision according to release 1.35	All

6 Introduction

This document is a User's Guide to the ERTMSFormalSpecs Workbench (EFSW). It details the features and functions of the ESFW.

6.1 References

Index	Title	Date
[1]	EFSW Technical design	15/12/2010
[2]	Subset-026 System requirements Specification V3.4.0	06/01/2015

6.2 Terms and definitions

Term	Definition
BL	Baseline
ESFW	ERTMSFormalSpecs Workbench
EFS	ERTMSFormalSpecs
EFSM	ERTMSFormalSpecs Model
EFST	ERTMSFormalSpecs Test
EVC	European Vital Computer
BNF	Bourne Normal Form

7 ERTMSFormalSpecs Workbench Introduction

7.1 The language

The ERTMSFormalSpecs language is an ad-hoc language, developed by *ERTMS Solutions* for the sole purpose of modelling ERTMS specifications.

The EFS model is made up of a data dictionary, containing requirements, traceability information, types, functions, procedures, variables, all the rules and tests, organized hierarchically, to represent the behaviour of the trainborne or trackside systems, as described in ERA's Subsets.

For instance, the EFS model for trainborne system is a non-trivial artefact (estimated at 2000 rules, 500 variables, 1500 tests, 500 pages of specifications), far too large and complex to be managed without ad hoc tools.

7.2 The workbench

The ERTMSFormalSpecs Workbench (EFSW) is a graphical tool designed to develop, maintain, and document model-based development for the Subset-026, Subset-027, Subset-034; Subset-076 and Optional Documents related with the ETCS/ERTMS system, such as "*DMI Start and Stop Conditions*". It is a desktop application, running on the *Microsoft Windows platform*. EFSW has also been successfully used to model trackside systems, such as Interlocking or RBC.

The EFSW provides high-level features:

- Requirements analysis tool, used to manage the requirements related to the system and take care of traceability between requirements, model and tests (section 8).
- Model browser, used to define the system's structure and dynamics (section 9).
- Test browser and execution environment, used to test the model and animate it (section 10)
- A specific tool used to automatically translate tests as described in Subset-076 (section 11).

The EFSW provides all features and functions required to support these high-level features, explained in details in the following sections.

7.3 Workbench installation

The complete EFS environment is distributed in a setup file which automatically installs the tool in the installation directory

C:\Program Files (x86)\ERTMSSolutions\ERTMSFormalSpecs

The path where the EFS environment is installed can be changed during the setup phase.

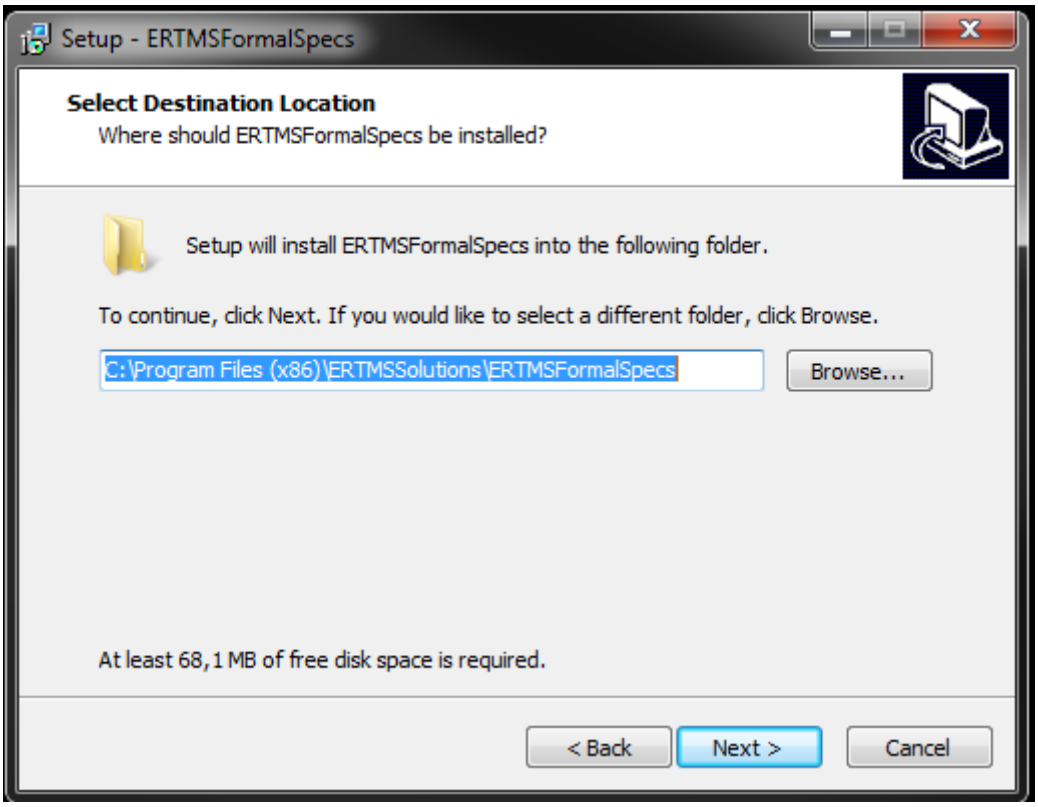


Figure 1: Path selection during installation

The installation process creates icons on the windows start menu to launch the EFSW and access the user guide of the ERTMSFormalSpecs Workbench. Table 1 indicates the minimum requirements:

Architecture	64 bits
Operating System	Windows 7
Framework	.NET 4.0 or higher

Table 1: Minimum requirements for installing EFSW

The installer also checks that the required .NET framework 4.0 is installed on the target machine before performing the installation.

7.4 Initial workbench state

The initial state of the EFSW main window executed just after being clicking on the icon present on the Start menu on windows is empty as Figure 2 shows.

Figure 2: Initial EFSW main window status.

7.4.1 Opening a EFS file

The EFS files are stored on:

${installationDirectory}\doc\specs\

Where the ${installationDirectory} represents the location where the EFSW has been installed. By default it is: C:\Program Files (x86)\ERTMSSolutions\ERTMSFormalSpecs. Otherwise, it is the path selected during the installation.

This directory holds, between others, the model and test files used to describe the trainborne system. Starting modelling the trainborne system requires to open one of these files, as Figure 3 and Figure 4 depict:

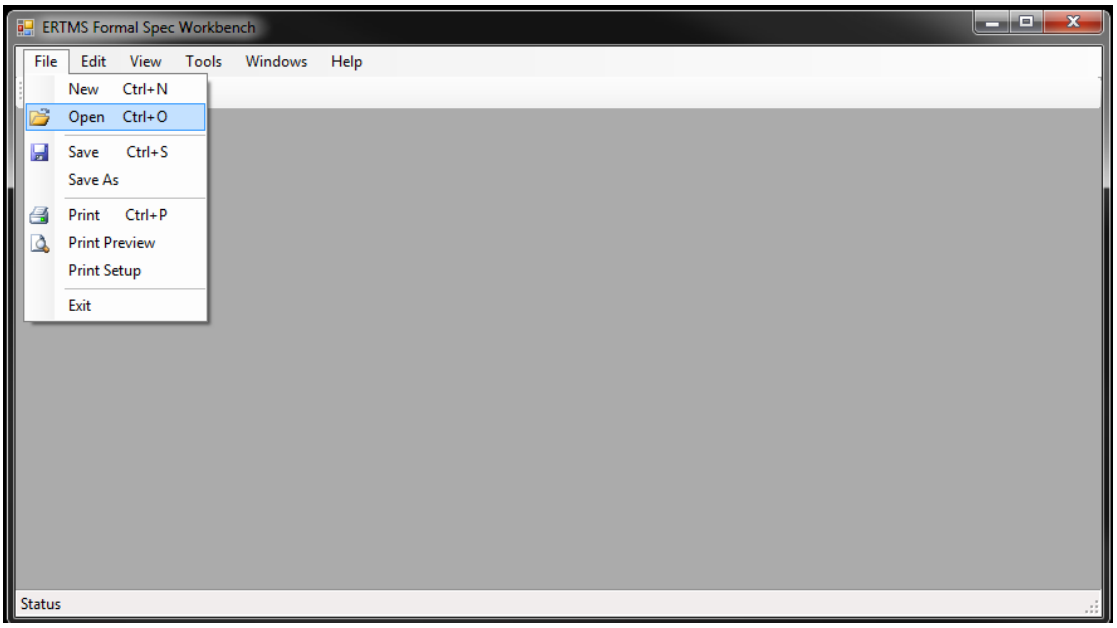


Figure 3: Open an EFS file.

To open a file, go to: File -> Open, select an .efs File and click on Open. Files with the .efs extension are the root files describing the model. They rely on other files to describe the system

- Files with the extension .efs_ns. Such files describe the namespaces used in the model
- Files with the extension .efs_tst which describe the tests used to verify the model

Such files are located in directories below the .efs file they refer to.

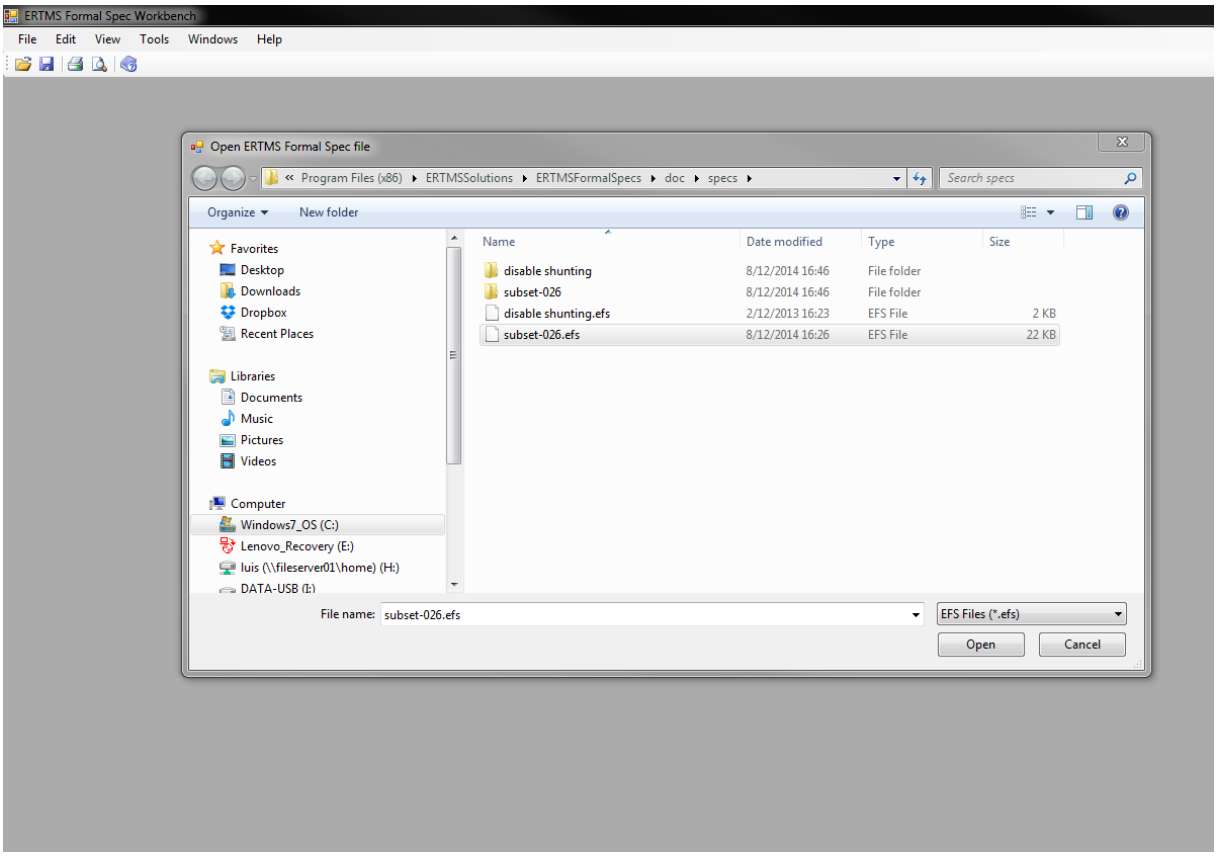


Figure 4: EFS file selection.

When an EFS file is loaded, specification browsers (see Chapter 8), data dictionary browsers (see Chapter *9*) and the test browser (see *Chapter 10*) are opened automatically, as displayed in the following figures.

The data dictionary browser is displayed by default (as Figure 5 depicts)

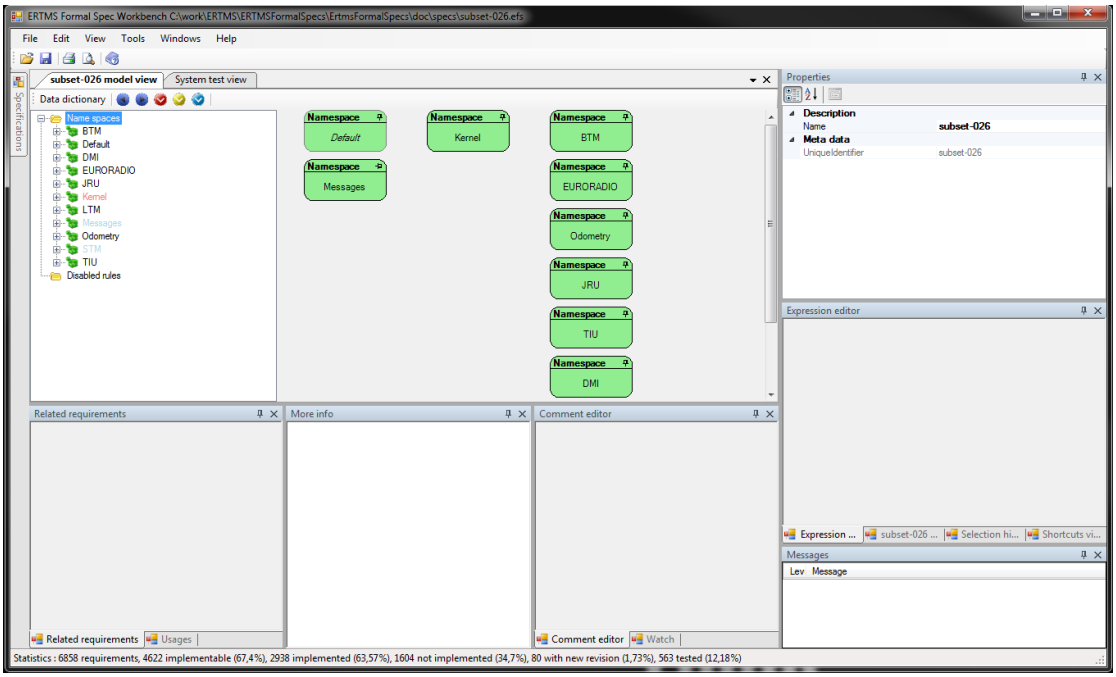


Figure 5: Data dictionary browser

The test browser is the second view displayed in the main part of the EFSW (Figure 6):

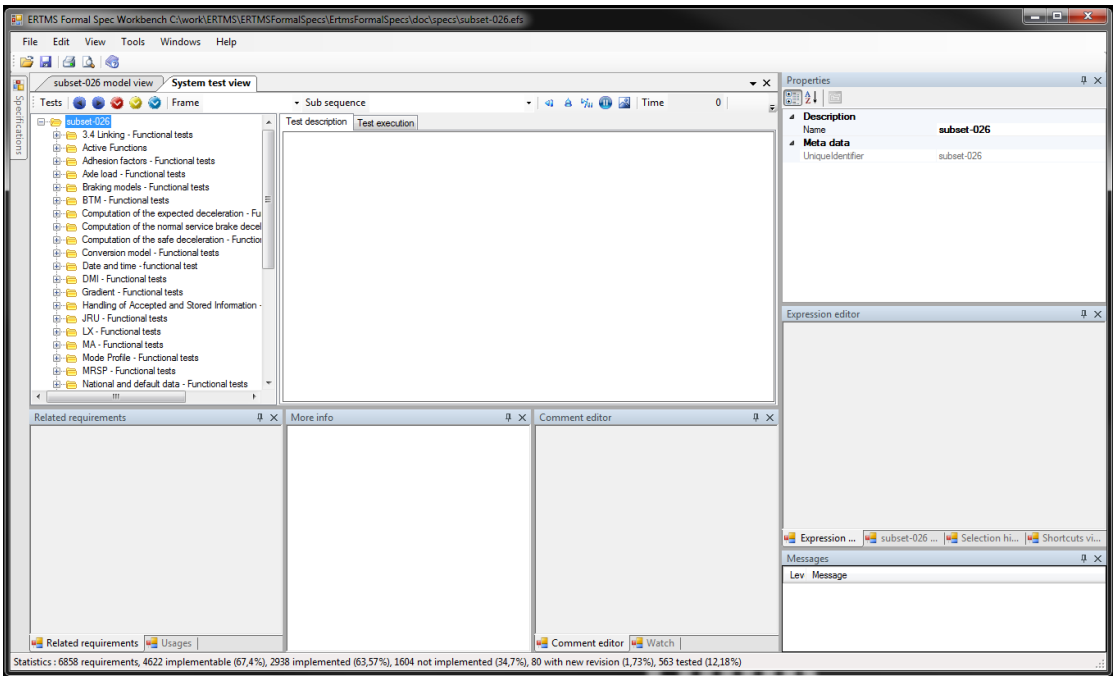


Figure 6: Test browser and execution environment

The Specification browser can be seen by expanding the left part of the EFWS (Figure 7):

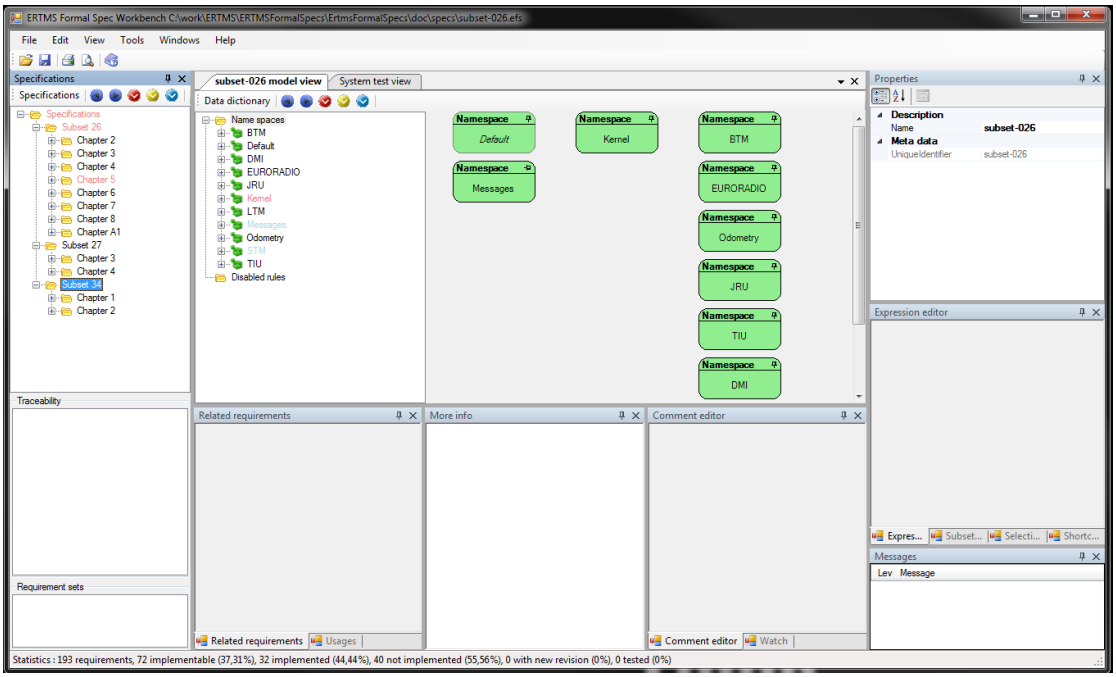


Figure 7: Subset-026 Specification browser

Several files can be opened at the same time. The resulting modelled EFS system is the combination of all those files. This allows, for instance, to add project specific behaviour to a given system, or to dissociate specific tests from the model. For instance, braking tests with specific train data are located in a separated (test) model, named braking curves verification.efs.

7.5 Messages on the ERTMSFormalSpecs Workbench:

7.5.1 Messages window

The EFSW provides information about the status of the selected item in the Message window (lower right part of the Workbench main window). Figure 8 presents a typical example of the message window's contents.

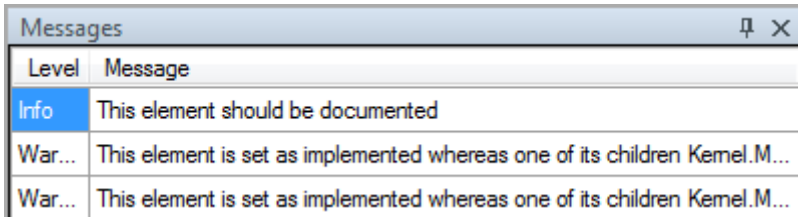


Figure 8: message window showing messages related with one of the requirements.

Messages are composed of a Level and the informative message. The Level column is related with the type of the message:

- **Info**: additional information related with the result of a search or an indication related with a requirement.
- **Warning**: indication related with an element which may become a problem and provoke an Error.
- **Error**: indication related with an element which will provoke the system not to work.

7.5.2 Messages colours:

The EFSW use different colours to identify the elements which have generated a Message, Figure 9 depicts a situation where several messages are highlighted on the data dictionary browser:

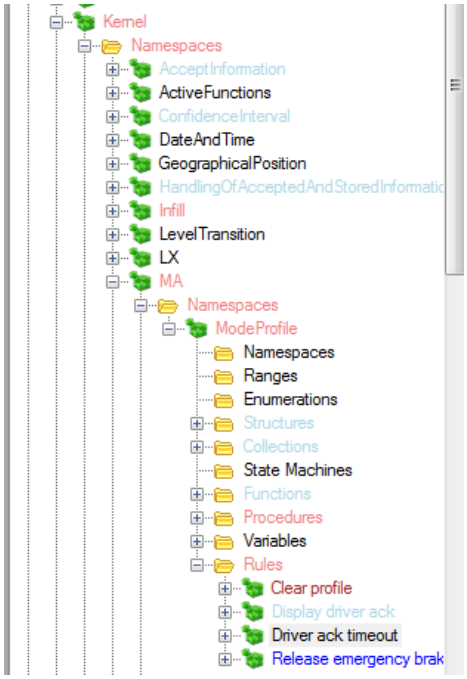


Figure 9: representation of different colours used on the EFSW.

Table 2 presents the colours used to denote the presence of a message. They are ordered in increasing precedence order, meaning that if there is a choice to display a colour, the system will select the colour the furthest in the table. For instance, if Light blue and Orange can be selected, the system will colour the node in Orange because it is the furthest element in the table.

Colour	Level	Meaning
Light blue	Info	There is at least one Info message on the hierarchical tree under this element.
Dark blue	Info	The current element is directly associated with an Info message.
Rose	Warning	There is at least one Warning message on the hierarchical tree under this element.
Brown	Warning	The current element is directly associated with a Warning message.
Orange	Error	There is at least one Error message on the hierarchical tree under this element.
Red	Error	The current element is directly associated with an Error message.

Table 2: messages and colours representation

Figure 10 depicts the elements of the model containing only a messages of type Info.

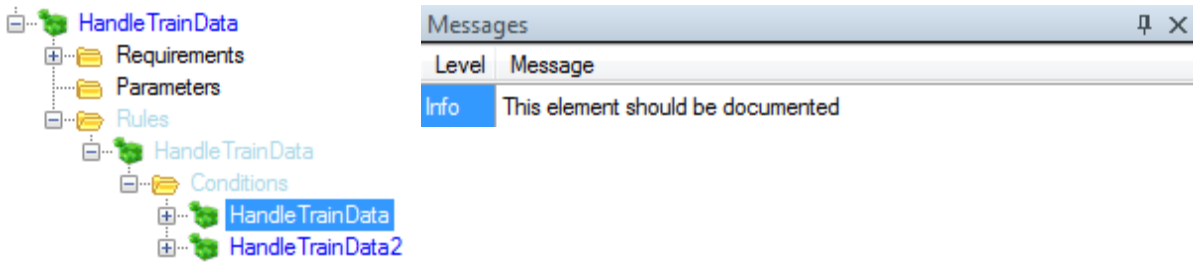


Figure 10: representation of the information messages on EFSW

Figure 11 depicts how the warning messages are represented on the EFS data dictionary browser and the associated explanation on the messages view window. The warning message indication on the data dictionary tree are done following the colours representation of Table 2.

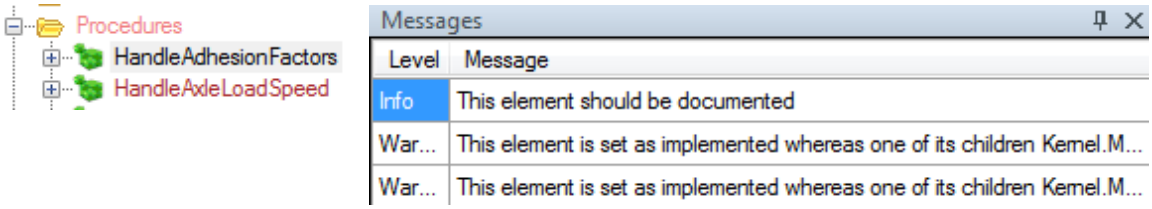


Figure 11: representation of the warning messages on EFSW.

Figure 12 depicts the error messages representation on the data dictionary hierarchical tree and the linked messages to the highlighted model item. Table 2 describes the colours to be used for the model nodes and elements.

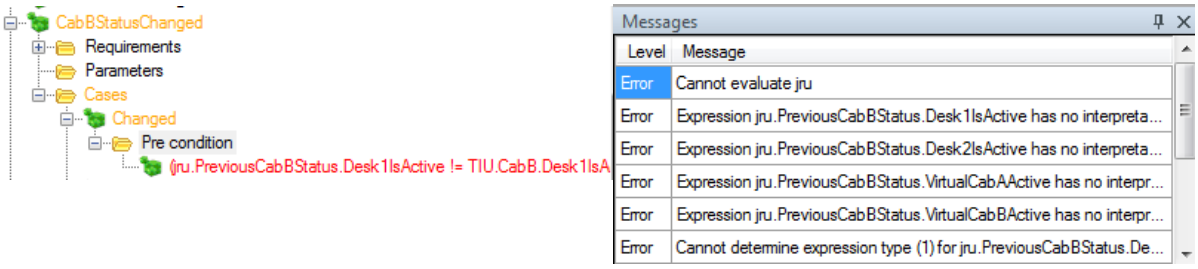


Figure 12: representation of the error messages on EFSW.

7.5.3 Messages navigation buttons

The buttons  ,  and  can be used to navigate to respectively the next error, warning or information message, according to the selected node in the tree.

7.6 ERTMSFormalSpecs Workbench properties

All the elements in EFSW have several metadata associated to them – properties – and are displayed in the Property view, on the upper right side of the EFSW main window, Figure 13 illustrates the location of the property window.

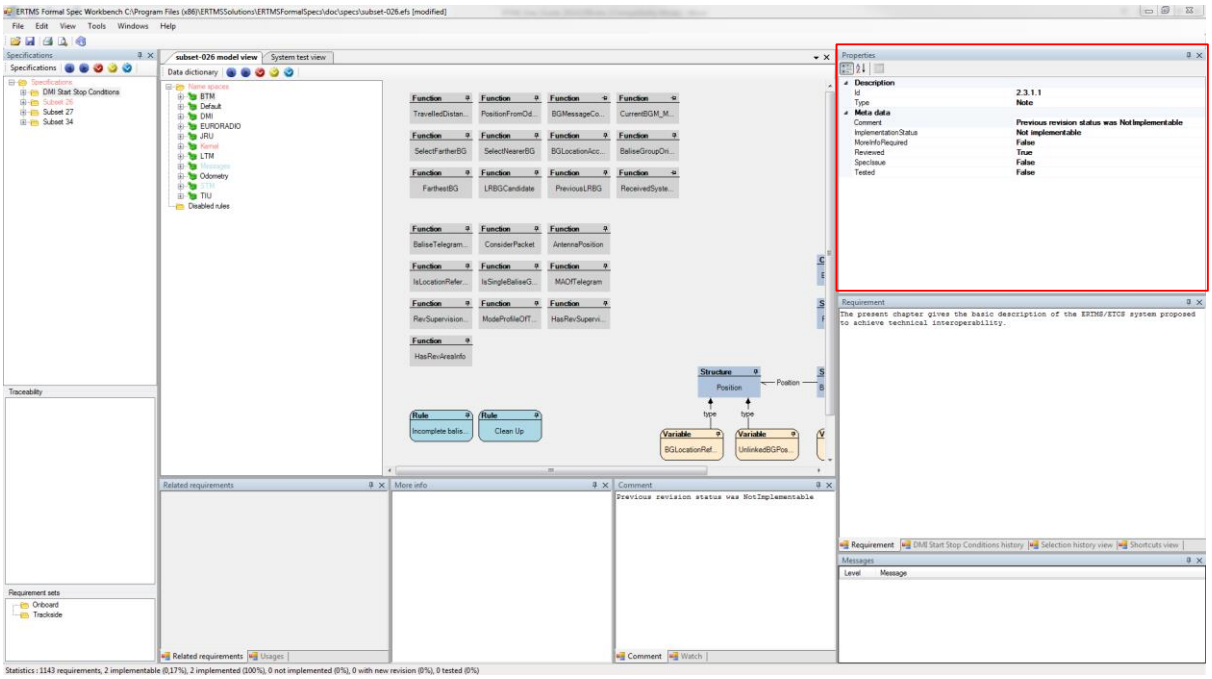


Figure 13: Properties location on the EFSW main window.

For instance, Figure 14 presents a typical content of the property window. Specific properties will be explained when describing each component of the EFS model.

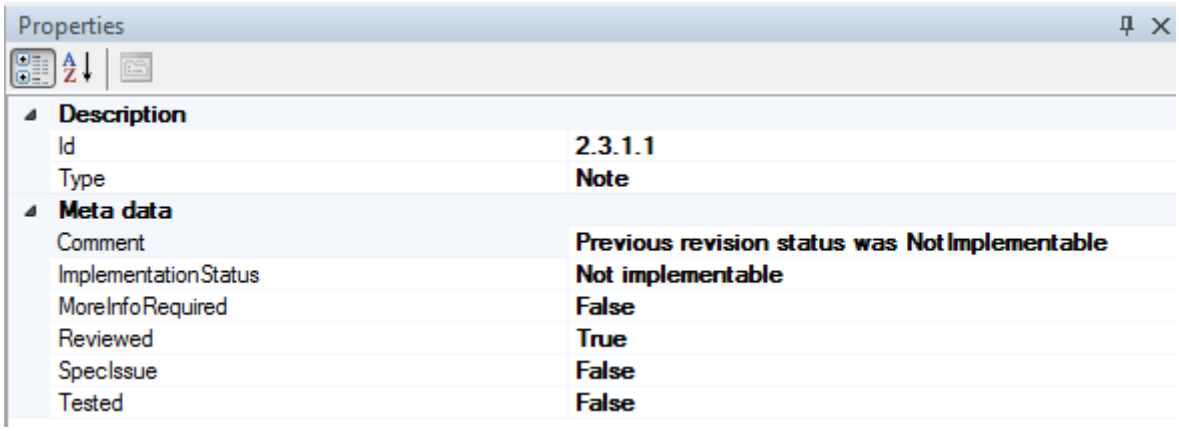


Figure 14: detailed view of the properties window of an EFSW element.

Along most of the elements on the EFSW contain some of the following properties.

- **Comment**: additional information giving an explanation of the current selected element of the hierarchical tree.
- **Name**: the given name of the current selected item of the hierarchical tree.

- **Unique identifier**: represents the unique name given to the current selected element of the hierarchical tree on the namespace where it belongs.
- **Needs requirement**: indicates that this part of the model requires requirement to be linked to it.
- **Implemented**: indicates the status of implementation of the current node of the hierarchical tree. When this flag is set to true means that it has been fully implemented.
- **Verified**: gives the related information to the revision status of the current node of the hierarchical test browser tree. When set to true, means that the current element of the test browser has been successfully reviewed.

8 Specification browser

The specification browser is used to display the requirements to be modelled. In the case of the trainborne system, it contains requirements from several Subsets, and also optional documents.

8.1 Opening the specification browser

The specification browser is automatically opened when launching the EFSW and it is reduced on the upper left corner on the EFSW main window.

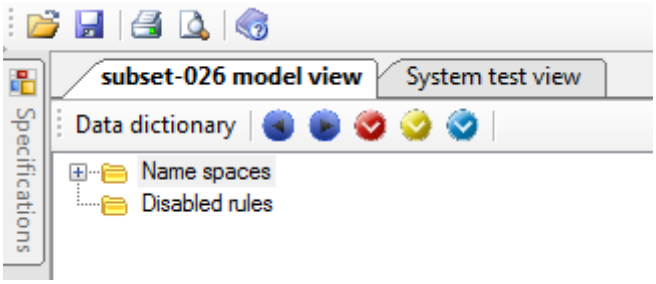


Figure 15: location of the Specification browser when launching

In case the model window tab is closed clicking on the close button located on the top right corner of the specifications tab, it can also be re-opened using the available option on View\Show specification view menu. Figure 16 depicts how to re-open the specification window.

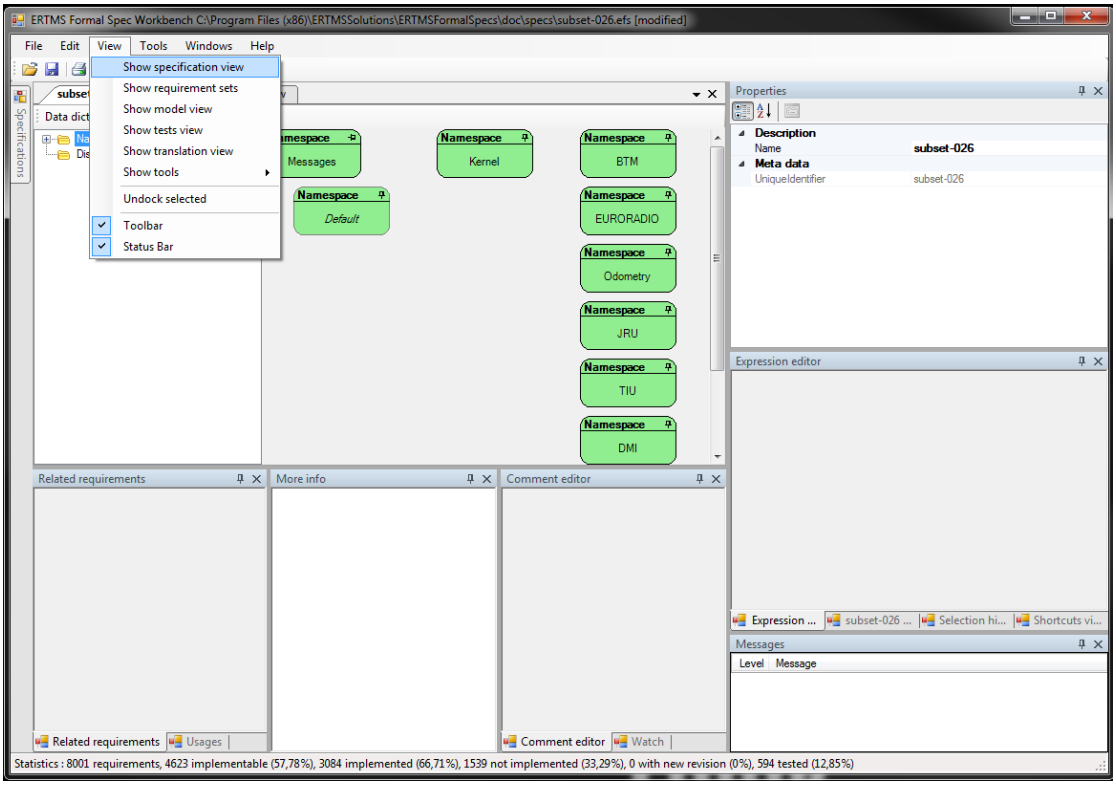


Figure 16: opening the specifications browser clicking the view contextual menu.

Figure 17 displays the general overview of the EFSW when the specification browser is pinned up as part of the main window. As could be seen the specification browser is allocated on the left side of the EFSW main window.

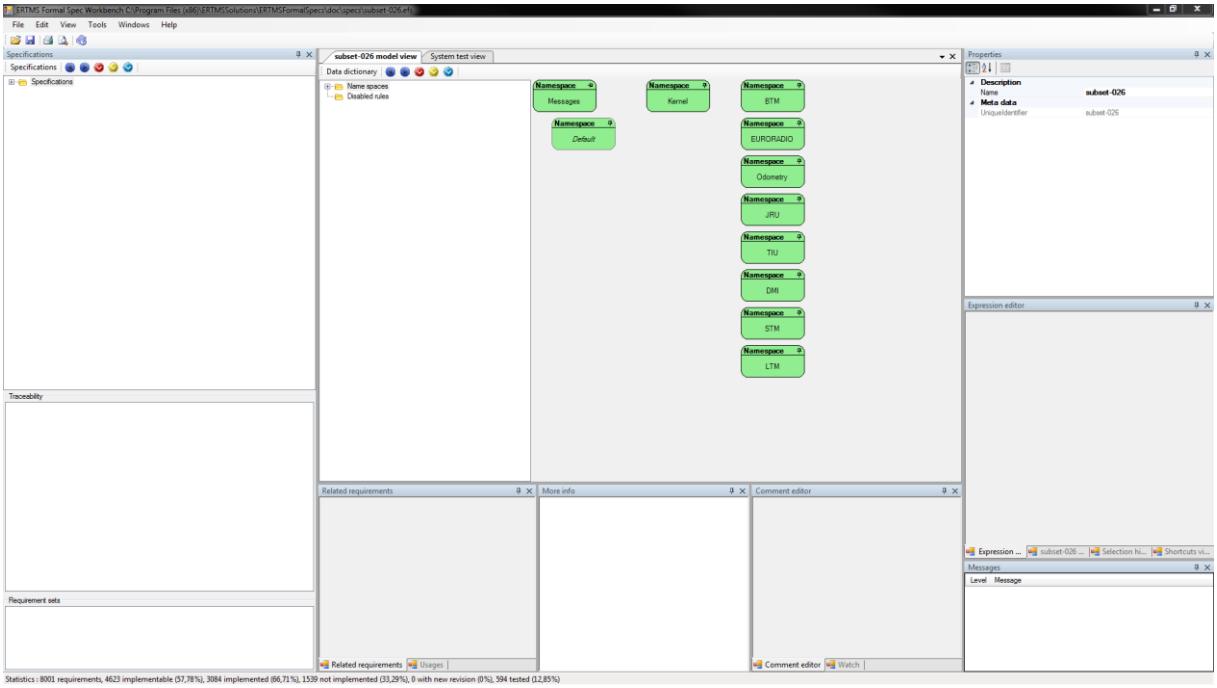


Figure 17: General overview of the specification browser.

8.2 Specification browser description

8.2.1 Overview

The specification browser is decomposed into several parts, as Figure 17 indicates.

- The left side contains the hierarchical tree specification browser, traceability and the requirement sets tab.
- On the upper left side the hierarchical tree view is found. It displays the input **specifications** in hierarchical order. It faithfully mirrors the content of the selected Subset as depicted by Figure 18.

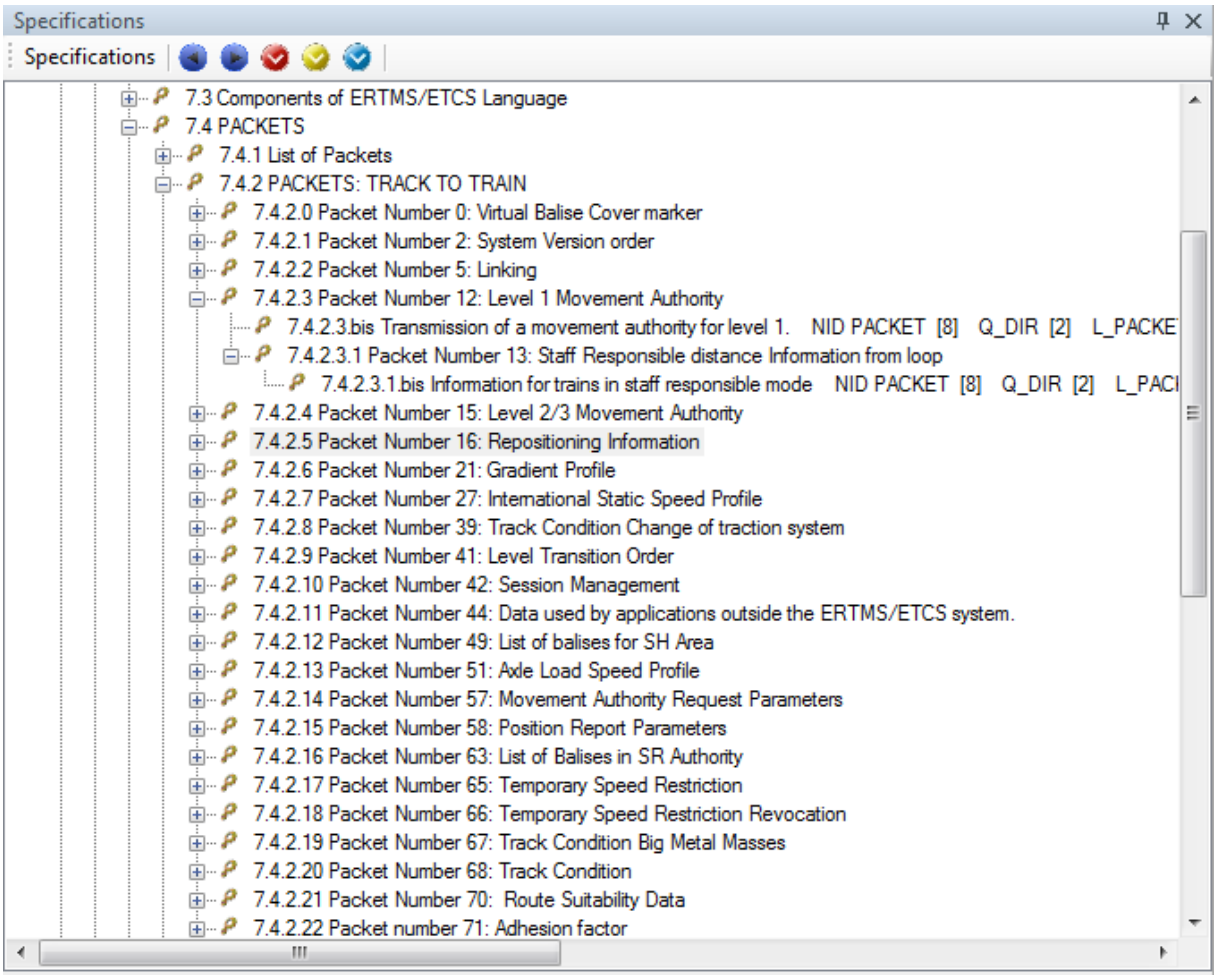


Figure 18: Selected Subset Specification view.

- The middle left part of the specification browser window displays the **traceability**: see Section 8.4 for further details about the traceability.

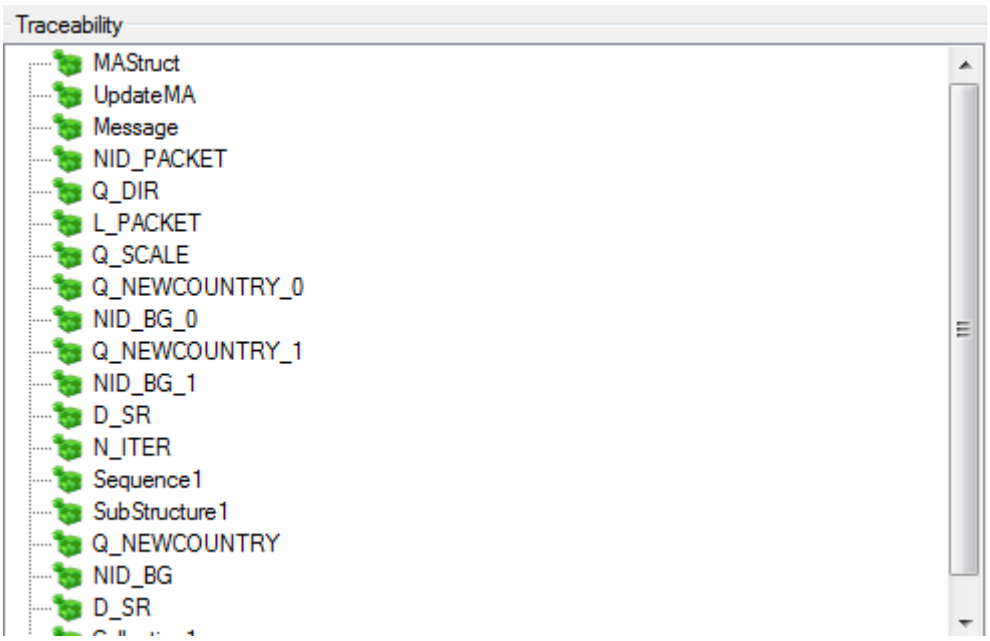


Figure 19: traceability section on the specification browser.

- The lower left part (the **requirement sets**) indicates to which set of requirement this requirement belongs. In this case, the requirement belongs to two requirement sets: Onboard & trackside.

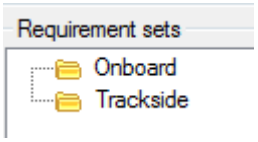


Figure 20: requirement sets section on the specification browser.

For further details about the requirement sets, see Section 8.2.2.

- The right side contains the properties, messages and different tabs containing the requirement information, historical information, selection history and shortcuts view.
- **Properties**: located on the upper right corner. See section 8.2.2.
- On the middle right side several different tab are located.
- **Requirement:** allows to edit the requirement text of the selected requirement.

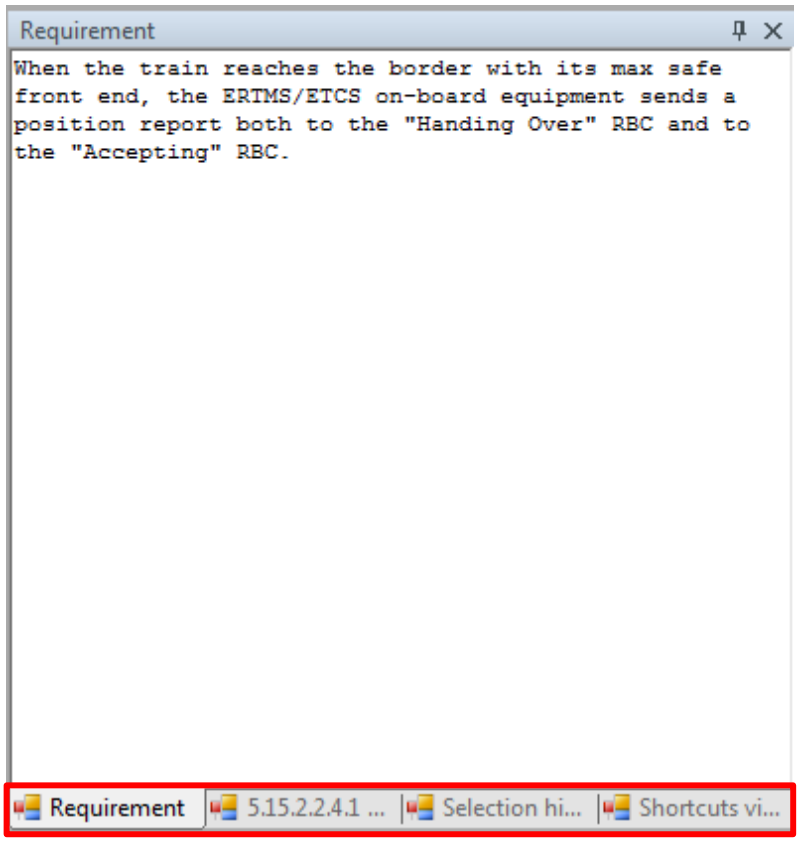


Figure 21: detailed view of the middle right side of the main window.

- **Messages:** located on the lower right corner. For further details see section 7.5.

8.2.2 Requirements classification

Requirements are classified according to their role in the document:

- Requirements contained on a table are further divided in sub-requirements, the identifier of this sub-requirements is composed by the identifier of the table and an extra number for the row identifier. (For instance a requirement coming from [2], Chapter 4, paragraph 4.7.2 “DMI versus Mode table”) and adding before the requirement number the keyword.
- Table for the table header.
- Entry for each table line.
- When a requirement is further split into several requirements, a sub number is added to the requirement number

Figure 22 and Figure 23 depict the strategy of classification used on the EFSW:

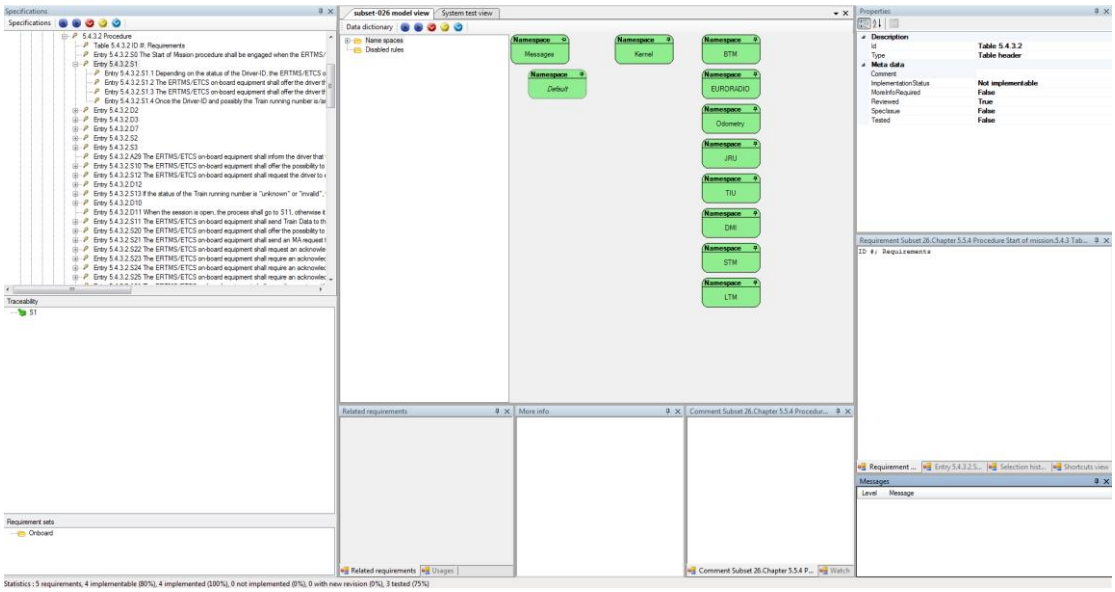


Figure 22: overview of the given names for tabluar requirements

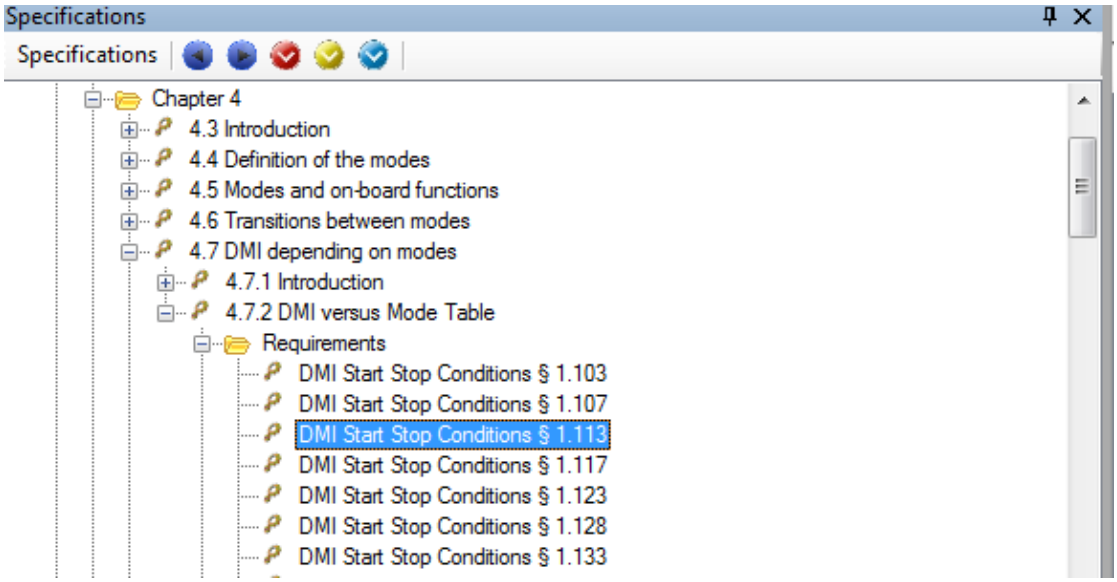


Figure 23: Handling of tabular requirements in specifications browser

8.2.3 Specifications properties

The properties of the specifications give a representation related with the development of the model and additional information not absolutely mandatory for the development process.

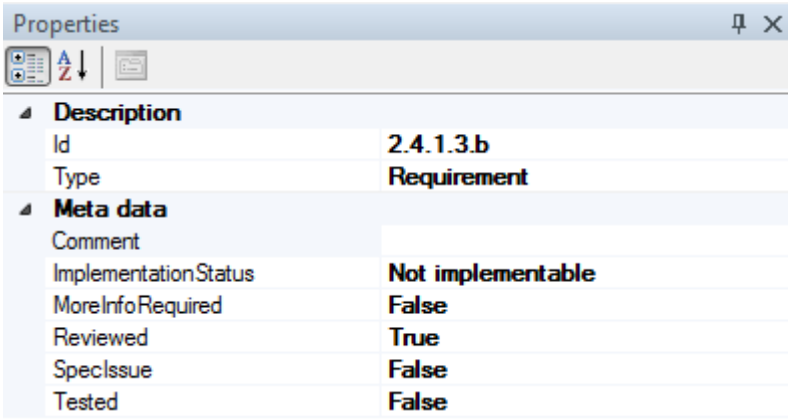


Figure 24: specifications properties view

Next to follow, the common properties for most of the specification elements are described.

- **Identifier**: represents the unique identifier of the requirement on the document where it was defined.
- **Type**: it is related with the kind of element. The possible types are:
- **Title**: related with the names of the different sub-sections which compose a chapter.
- **Note**: different comments present on the specifications.
- **Definition**: the definition of a concept.
- **Requirement**: a requirement of the specifications which need be modelled.
- **Table**: set of requirements which are grouped and related.
- **Implementation status**: indicates the current status of the model related to this requirement
- Implemented, the requirement has been completely modelled. There should be traceability information between this requirement and the related model
- not implemented, a model is required for this requirement, but it has not (yet) been created
- not implementable, the requirement cannot be modelled for various reasons, for instance because it is a requirement which constraints the environment in which the model is meant to exist.
- new version available : this indicates that the requirement has been modelled for a previous version, but it has changed and the model should be reviewed to take that change into account
- **More info required**: indicates that the requirement is not precise enough and should provide more information. Usually, there is a comment attached to requirement set to "More info required" to indicate the kind of information which is required.
- **Reviewed**: indicates that the content of the requirement has been reviewed and matches the textual representation provided (Word document, paper, …).
- **Spec issue**: represents the existence of an issue related with the requirement. Usually, a comment is provided to explain the issue encountered when modelling this requirement.
- **Tested**: Indicates that tests have been created to ensure that the model correctly implements the requirement's expectations. When a requirement is set to tested, it should hold traceability information to the corresponding tests.

8.2.3.1 Properties of Requirement Document:

The properties present on a Requirement Document give the information to identify it in a unique way. Figure 25 displays the properties of a Subset or optional document.

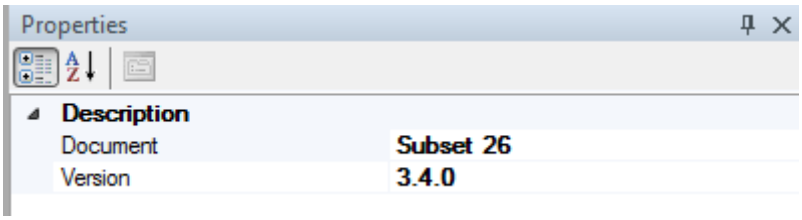


Figure 25: Subset or Optional Document properties

- **Document**: represents the name of the selected Requirement Document.
- **Version**: indicates what the version of the selected Subset or Optional Document is.

8.2.3.2 Properties of a Chapter on a Requirement Document

The properties present on a Chapter describe the information to determine to which chapter makes reference. Figure 26 depicts the properties of a chapter.

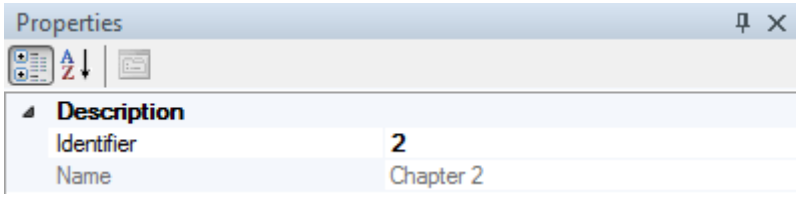


Figure 26: Chapter properties.

- **Identifier**: unique identifier of the Chapter on the document to whom belongs.

8.2.3.3 Properties of Paragraph:

- The properties of the Paragraphs are the same for all of them, and give additional information. The description of these properties is done just following, see Figure 27.

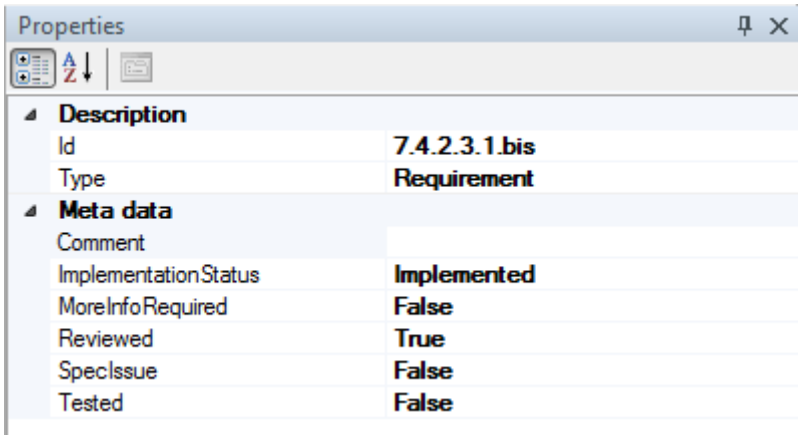


Figure 27: Properties contents

- The **Id** which univocally identifies the Paragraph in the Specification tree.

8.3 Requirement sets

Requirements can be classified in **requirement sets**, which allows to manage several requirements as a group. For instance, for the onboard unit, two main requirement sets are defined

- Scope. This allows to identify to which part of the system the requirement is related. The scope holds several sub requirement sets
  - **Trackside**: requirements are related to the trackside
  - **On Board**: requirements are related to the on board unit
  - **Rolling stock**: requirements are related to the rolling stock.
- Functional block, which split the system into a set of functions
  - Levels and transitions.
  - Recording of juridical data.
  - Management of radio communication
  - Acceptance of received information.
  - Speed and distance monitoring commands.
  - Train interface
  - …

A requirement can be classified on one or more, categories. For example a requirement can be linked to the **On-board** requirements and linked to the **Train interface** requirement set.

8.3.1 Managing the requirement sets

The **Requirement sets** of the requirements can be seen by: View -> Show requirement sets. Figure 28 depicts how to display the **Requirement sets**.

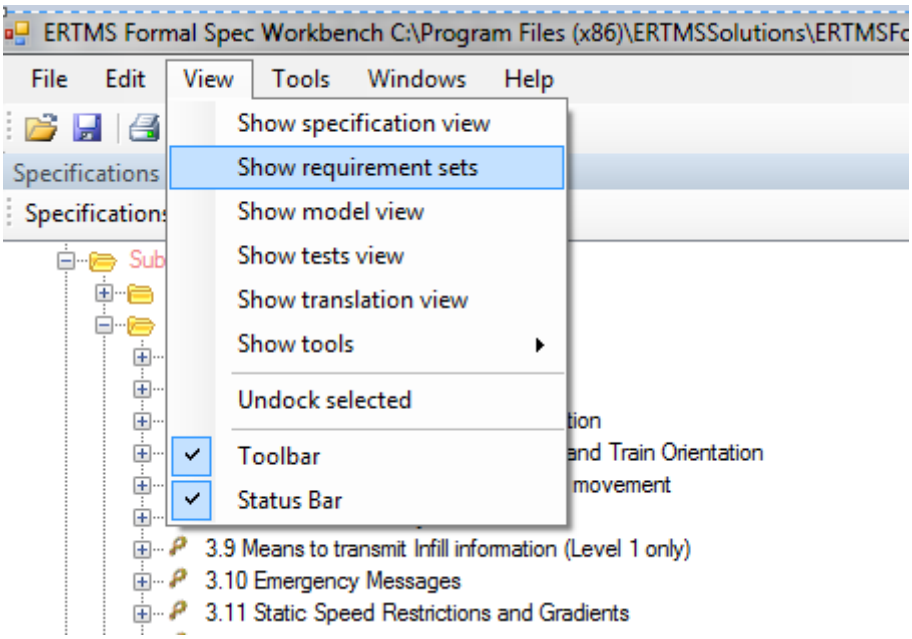


Figure 28: Selecting the requirements sets.

After clicking on the highlighted option, the EFSW displays the Requirement sets window for the selected Subset. Figure 29 illustrates the result of clicking on this option.

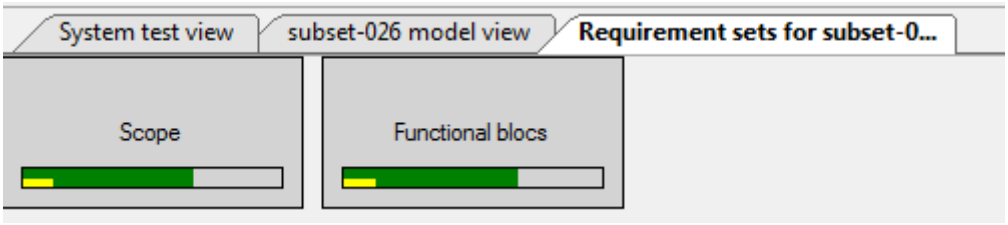


Figure 29: Requirement sets window for the selected Subset.

Figure 29 depicts the two highest levels of the requirement sets, and on their representation are displayed two progress bars.

- Yellow progress bar: indicates the percentage of each requirement sets that has been tested.
- Green progress bar: provides the percentage of each requirement sets that has been modelled.

Double clicking on the **Scope** Requirement sets the first classification of the requirements is shown. See Figure 30.

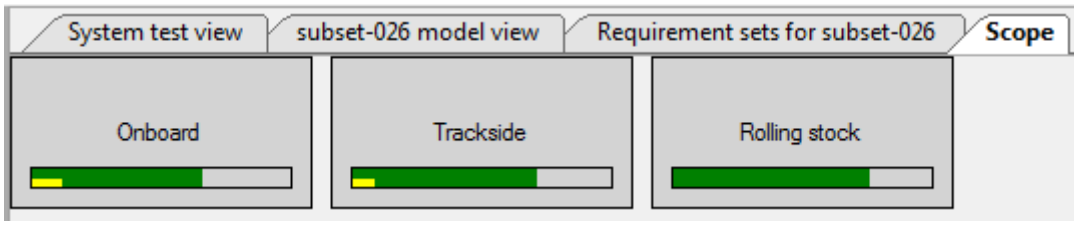


Figure 30: Representation of the possible scopes.

Right-clicking on a **Scope** and selecting the "Select paragraphs" option displays the paragraphs related to the selected requirement set. See Figure 31.

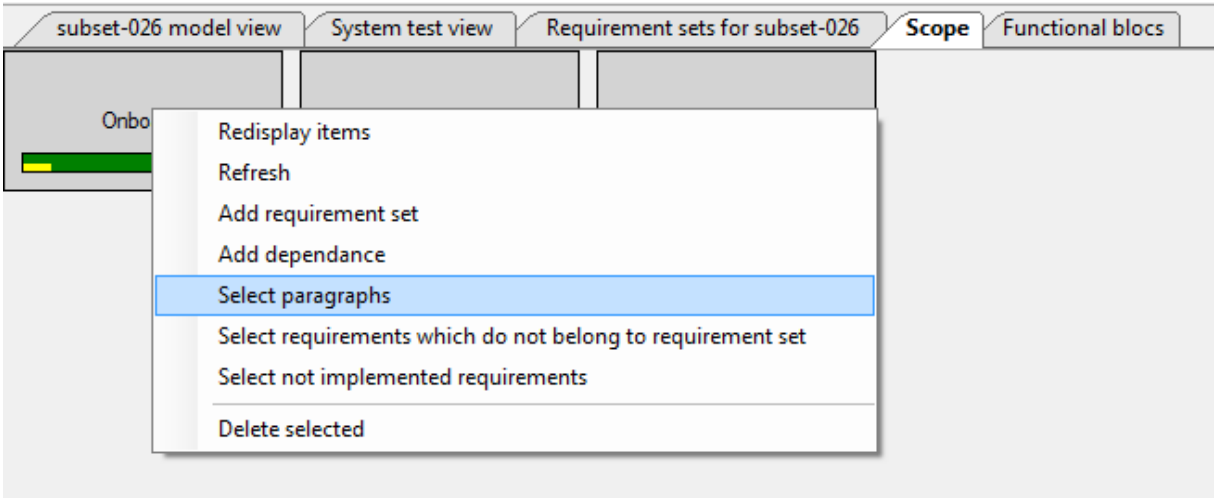


Figure 31: procedure to display the related paragraphs to a requirement set.

The related paragraphs to the selected Scope are displayed on light blue on the Specification browser:

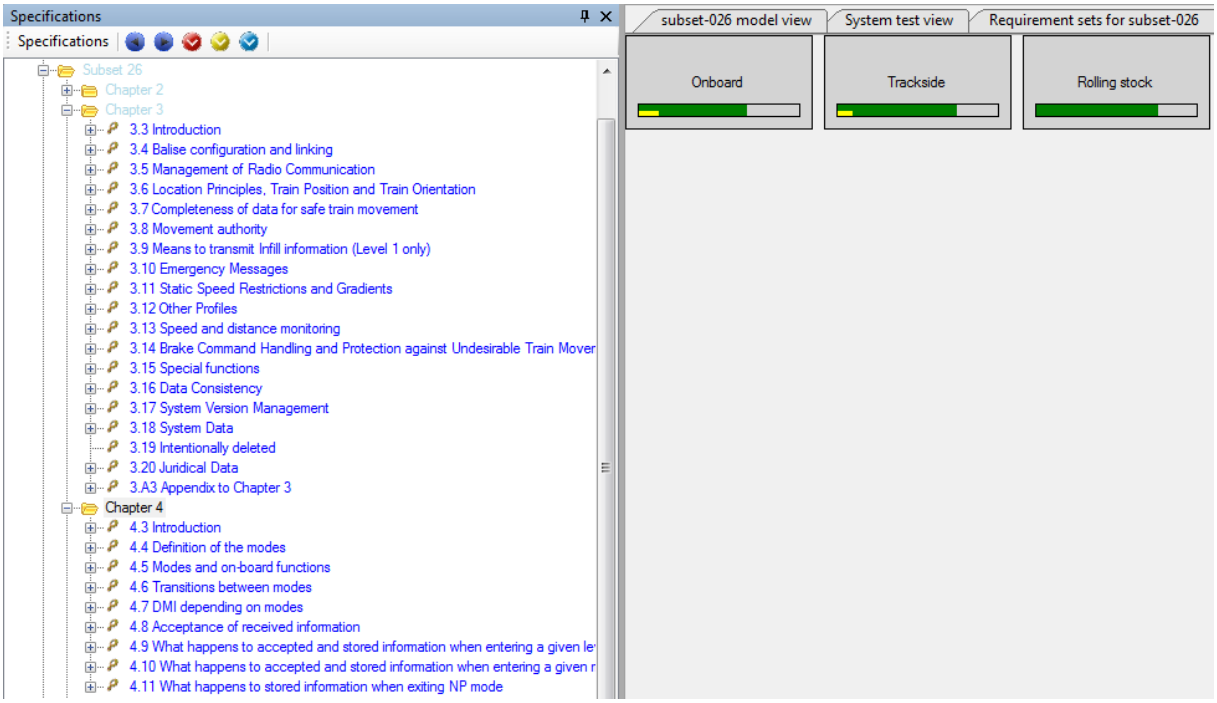


Figure 32: paragraphs related to the requirement sets

A requirement can be linked to a **Scope** by drag & dropping the requirement on the selected **Scope**. Drag the selected requirement on the selected **Scope** and drop it. The EFSW verifies automatically if the selected requirement is already present on the **Scope** and in case it already exists there, the requirement is not added.

Double clicking on a **requirement set** shows the sub requirement sets. For instance, double clicking on the **Functional block** requirement set shows all the **requirements sets** that belong to "**Functional block**". Figure 33 shows the available **Functional blocks** that have been defined for the onboard model.

Figure 33: requirement sets nesting.

8.3.2 Requirement sets properties:

Figure 29 displays the two highest level of the possible categories used to classify the **requirement sets**. When clicking on one of them, **scope** or **functional blocks**, a new properties window is displayed, see Figure 34 .

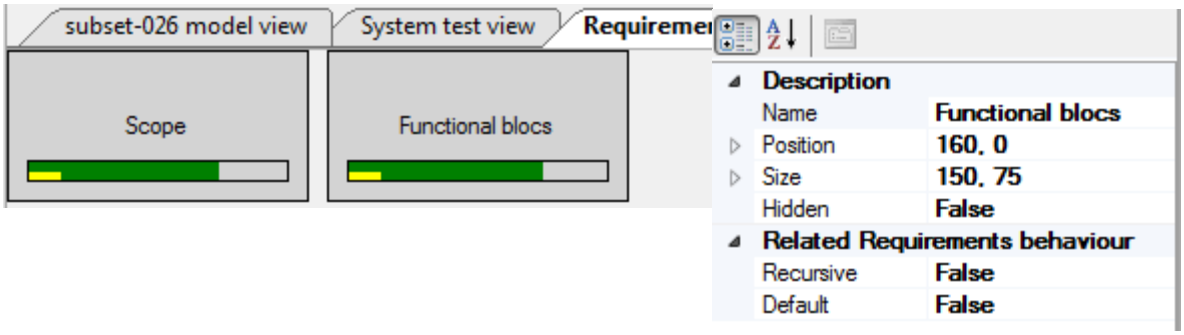


Figure 34: requirement sets properties window

Figure 35 shows a detailed view of the requirement sets window.

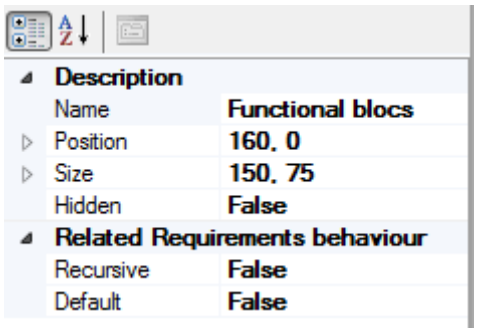


Figure 35: detailed view of the requirement sets properties window

The properties of a requirement set are the following

- **Recursive**: when this flag is set to true the subrequirement of the requirement which belong to this requirement set also belong (implicitely) to the same requirement set.
- **Default**: when adding a new requirement to the specification tree, that new requirement automatically belongs to the requirement set where the Default flag is set to true.

8.4 Requirements Traceability:

The **Traceability** tool is located on the middle part of the Specifications tab, as Figure 36 displays:

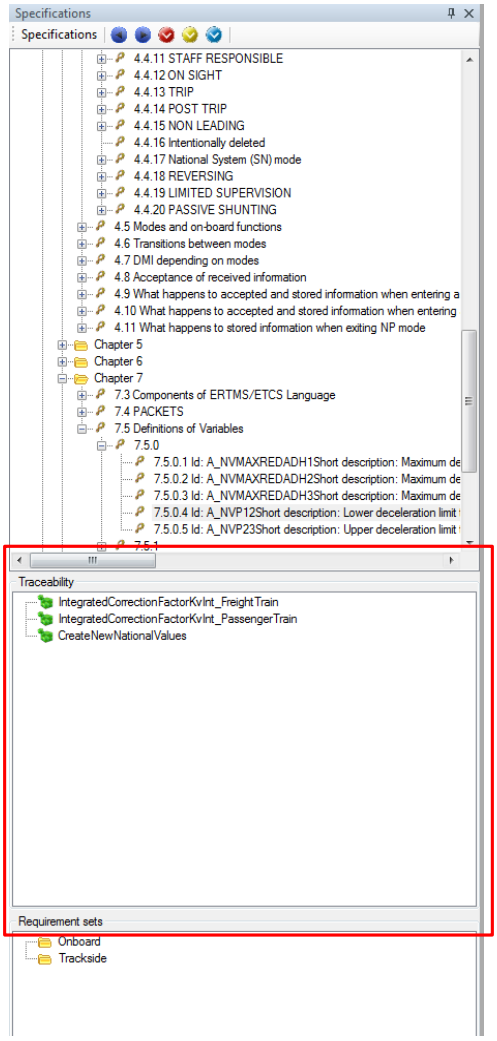


Figure 36: Location of the Traceability window on the Specifications tab.

The **Traceability** is used to link requirements to the elements of the model or to tests. Figure 37 shows that the requirement which identifier is 7.5.0.1 is related to four different elements of the model:

- *MaxDecelValueUnderReducedAdhesionCond1:* component of a model variable.
- *CreateNewNationalValues:* a function.
- *A_MAXREDADH*: function.
- *Check A_ MAXREDADH:* sub-step of the test.

Linking with model element is described in Chapter 9, whereas tracebility to tests is described in Chapter 10

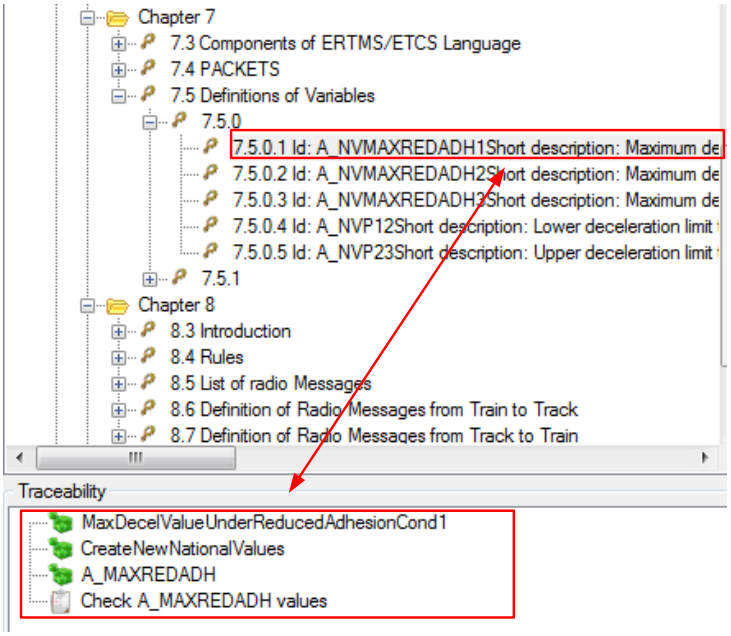


Figure 37: components of the model or the test linked with the selected requirement.

8.5 Tools related to the specification view

Figure 38 shows the actions which can be executed on the specification.

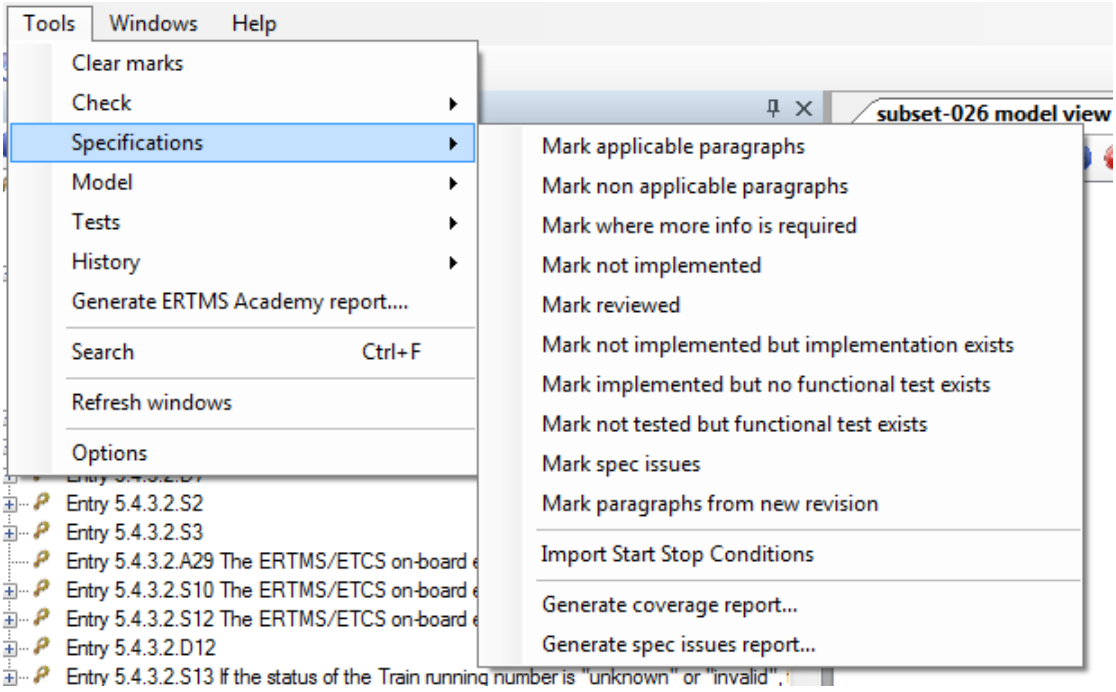


Figure 38: Actions could be performed on an element of the Specification tree.

8.5.1 Search for applicable and non-applicable paragraphs

Not all the requirements require implementation model. The EFSW can be used to find the requirements which require an implementation using the Mark applicable tool, which marks them in blue

In the same way, the Mark non applicable tool allows to mark paragraphs which do not require a model.

8.5.2 Search for paragraphs where more information is required

The paragraphs requiring more information to be implemented can be found using the Mark where more info is required tool. This marks all requirements for which the **More Info Required** property has been set.

8.5.3 Search for not implemented requirements

Requirements for which no implementation has been provided can be found using the Mark not implemented tool. This marks all requirements whose implementation is not complete. The implementation of a requirement is complete when

- The requirement status is Implemented
- All model elements associated to this requirement are also marked as Implemented.

8.5.4 Search for requirements that have not been verified

Requirements that have not successfully passed the review process can be found using the Mark not reviewed tool. This marks all requirements which have not been marked as verified. The verification phase ensures that the requirement, as encoded in the tool, corresponds to the requirement from the original source (paper, word document, …).

8.5.5 Search for requirements have not been implemented but implementation exists

The Mark not implemented but implementation exists tool allows to show all requirements for which an implementation is available (they are linked to at least a model element), but their status is still not implemented.

8.5.6 Search for implemented requirements without functional tests

The requirements that are implemented but not yet covered by a functional test can be found using the Mark implemented but no functional test tool. This marks all requirements that are marked as implemented but are not yet covered by functional tests.

8.5.7 Search for requirements which are incomplete or erroneous

If a requirement contains an error, or is incomplete, it is marked as spec issue. The Mark spec issues tool allows to mark all the spec issues that have already been detected.

8.5.8 Search for requirements from a new revision

If a requirement comes from a new revision of the selected Specification, can be found on the Mark paragraph from new revision contextual menu.

8.6 Processes related to requirements

Among the possible actions which can be performed over a requirement, the following have special importance: Review the requirements and Update the status of a requirement after implementation.

8.6.1 Review the requirements

Requirements must be reviewed for several reasons

- The conversion process between the selected Specification file and XML file is manual, hence error prone.
- Review the type and the scope associated to the requirement.
- Requirements expressed in the selected Specification file are not directly ready for modelling: some requirements are too small and must be merged with their neighbours; some others are too big and should be split.

As soon as requirements have been reviewed, the reviewer changes the Reviewed status of the requirement as **True**. Figure 39 depicts the available values for the **reviewed** property.

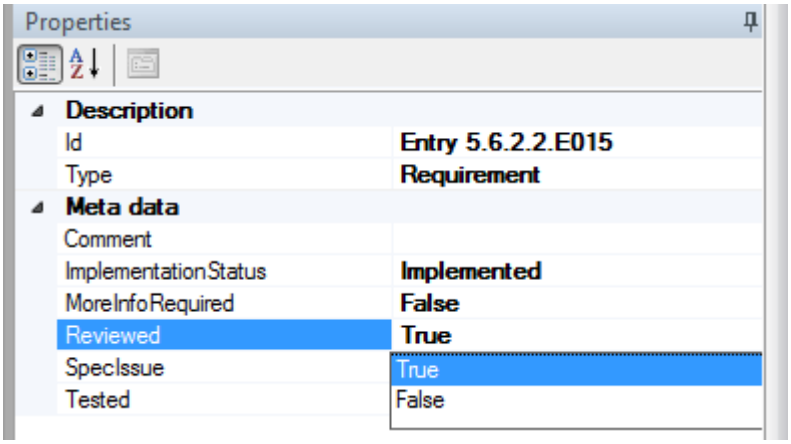


Figure 39: Requirement has been reviewed

The tool presented in Section 8.5.4 allows to easily detect the requirements that still require to be reviewed.

8.6.2 Create the model for the requirement

Once the requirement has been modelled, the engineer changes the Implementation status of the requirement to **Implemented**. The tool presented in Section 8.5.3 allows to detect the requirements that still require some modelling.

8.6.3 Update the model of an Implemented requirement

As soon as a model related to a requirement changes, the implementation status of the requirement must be manually changed from **Implemented** to the proper current status**.** Additionally, when the text of a requirement changes, the status is automatically set to new **revision available**

9 ERTMSFormalSpecs Model

The EFSM is used to model the specifications described on Chapter 8. This window can be used to model several items

- **Types** used to structure the model
- **Variables** which provide the system's state
- **Functions** which factorize common computations on the system
- **Procedures** which allow to sequence operations. Otherwise, the model described in EFS is purely functional
- **Rules** which provide the system dynamics by checking a set of preconditions before applying changes on the system's state.
- **State machines** are used to describe the system dynamic behaviour according to the inputs.

Figure 40 show a typical view of the EFSW when opening a model.

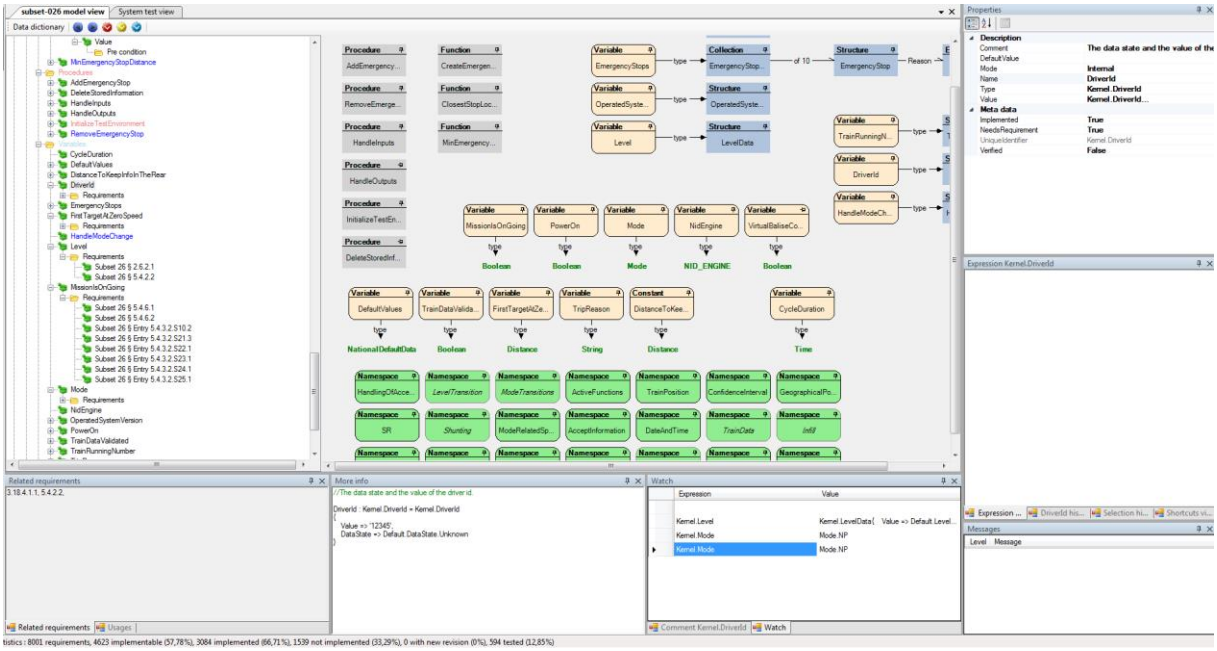


Figure 40: Data dictionary main window view and the representation of the close button for the model data dictionary.

9.1 Opening the data dictionary browser

The data dictionary main window is automatically opened, but also could be closed clicking on the close button on EFSW main window view. Figure 41 illustrates the location of the close button on EFSW main window.

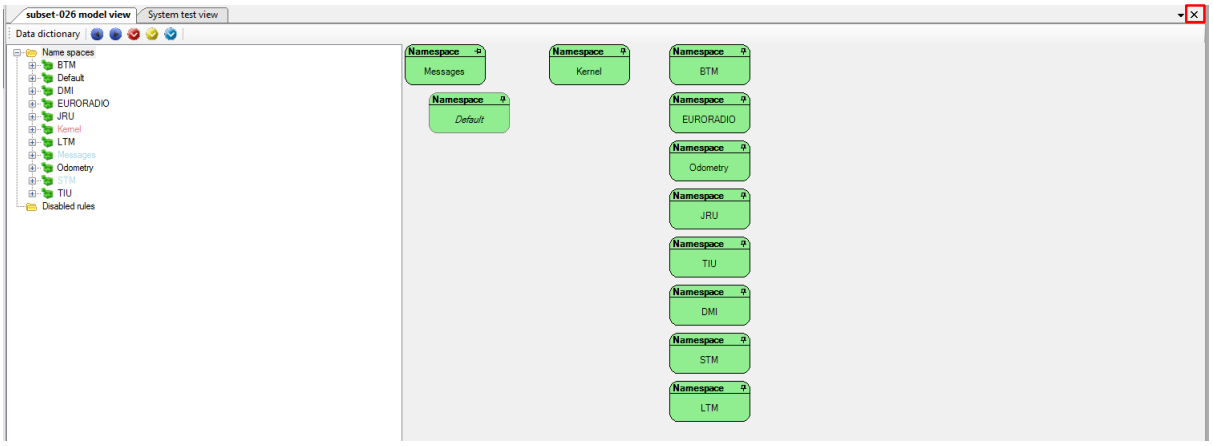


Figure 41: location of the close button on the model data dictionary window

The same window can be re-opened using the View\ Show model view contextual menu. Figure 42 depicts how to re-open the model view window.

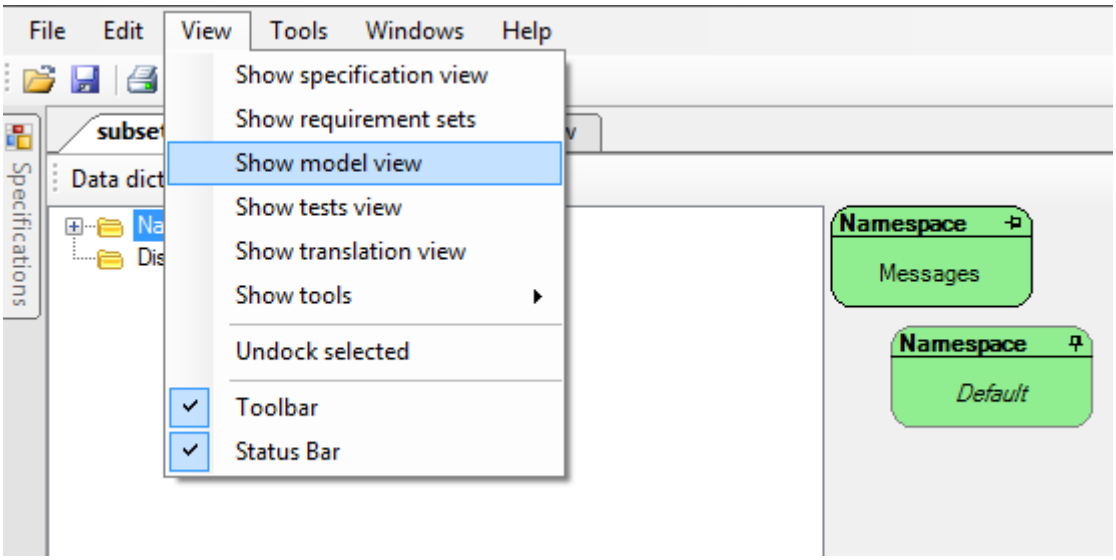


Figure 42: re-opening the model view window

If several EFS files have been opened, a dialog box is provided to select the EFS file on which the data dictionary browser should be opened as Figure 43 displays.

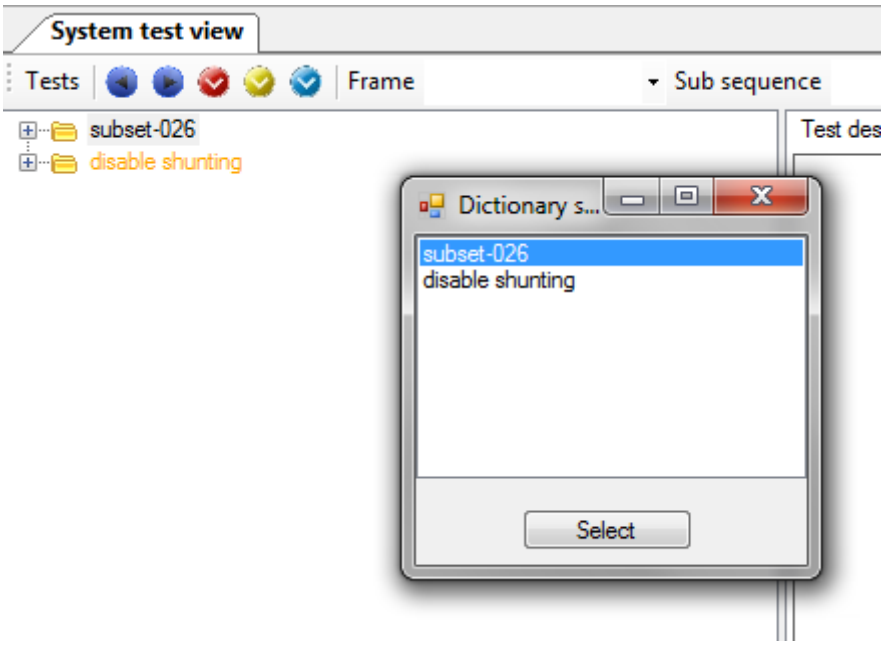


Figure 43: Dialog box for selecting the model to be open.

9.2 Data dictionary browser description

9.2.1 Overview

The data dictionary browser is decomposed into several parts highlighted on red. Figure 44 displays the parts of the data dictionary browser.

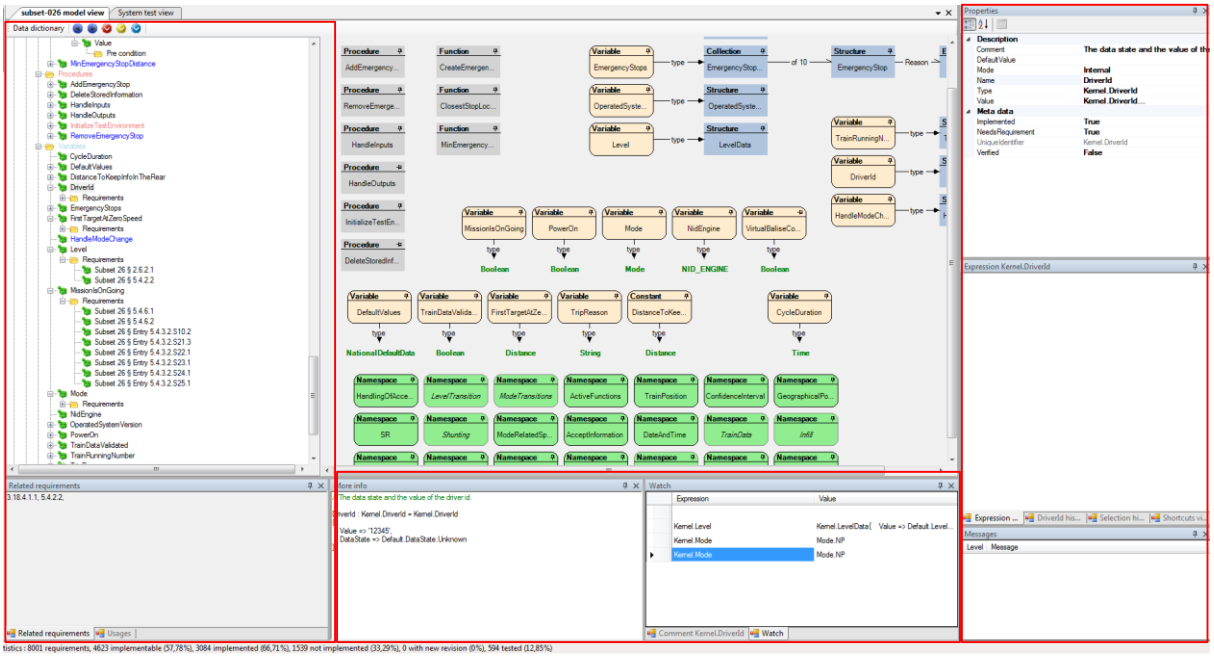


Figure 44: representation of the different components of the data dictionary.

- The left side contains the hierarchical tree view which allows to select an element in the model. To see the hierarchical tree in detail see Figure 45.

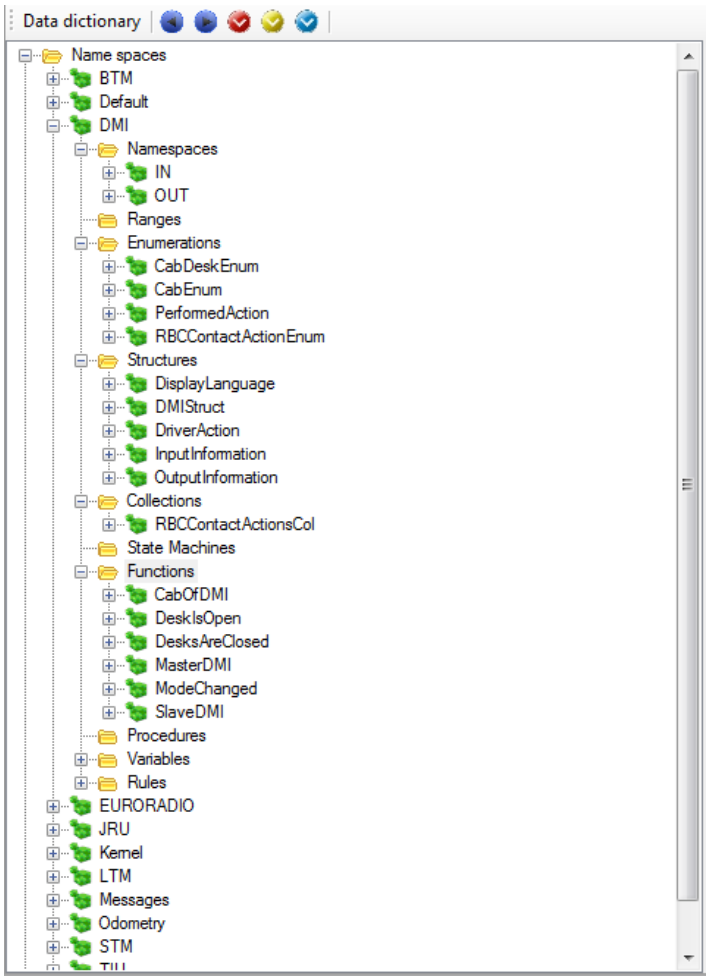


Figure 45: hierarchical tree which contains all the model elements.

- On the upper right corner, a property view, allows to visualize details of the element selected in the tree view. For further details about properties, see section 7.6
- On the lower right corner, a messages view, can be used to visualize messages provided by the Workbench on the selected element (see section 7.5).

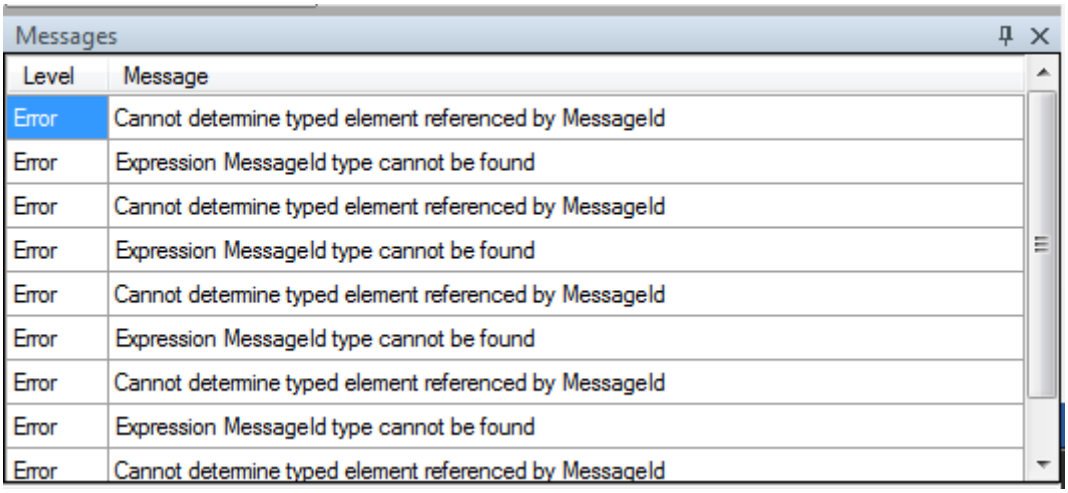


Figure 46: message view of a selected element from the model.

- **More info**: located on the lower middle part. This view provides a textual description of the current item using a pseudo-code representation. For example, see Figure 47.

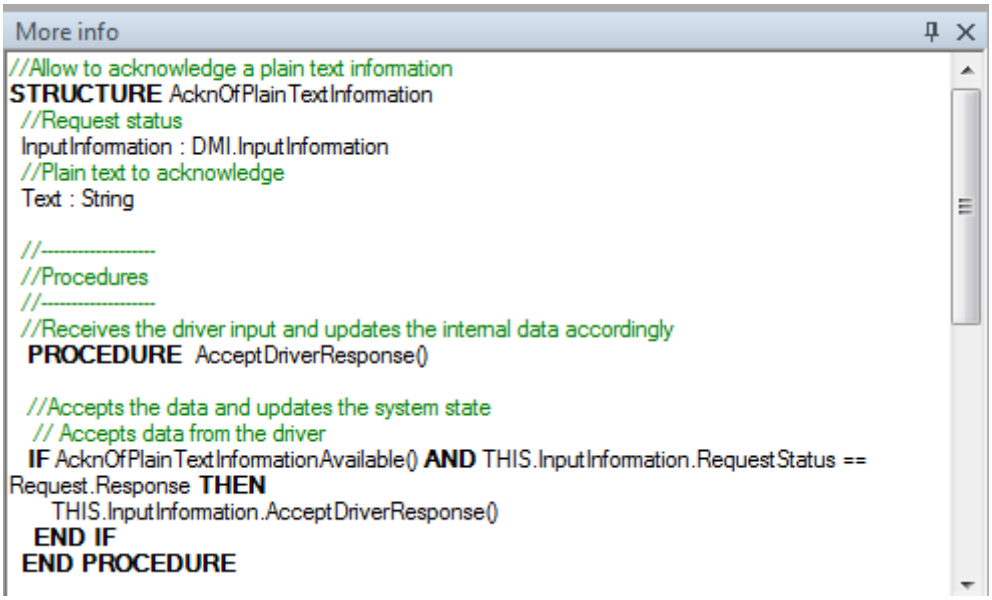


Figure 47: representation of the More Info window.

- **Expression**: located on the middle right parte. It is used to display and edit the expression related to the selected element, such as **Cases**, **Preconditions**, **Actions**, **Expectations**, ... Figure 48 depicts an example of the Expression window.

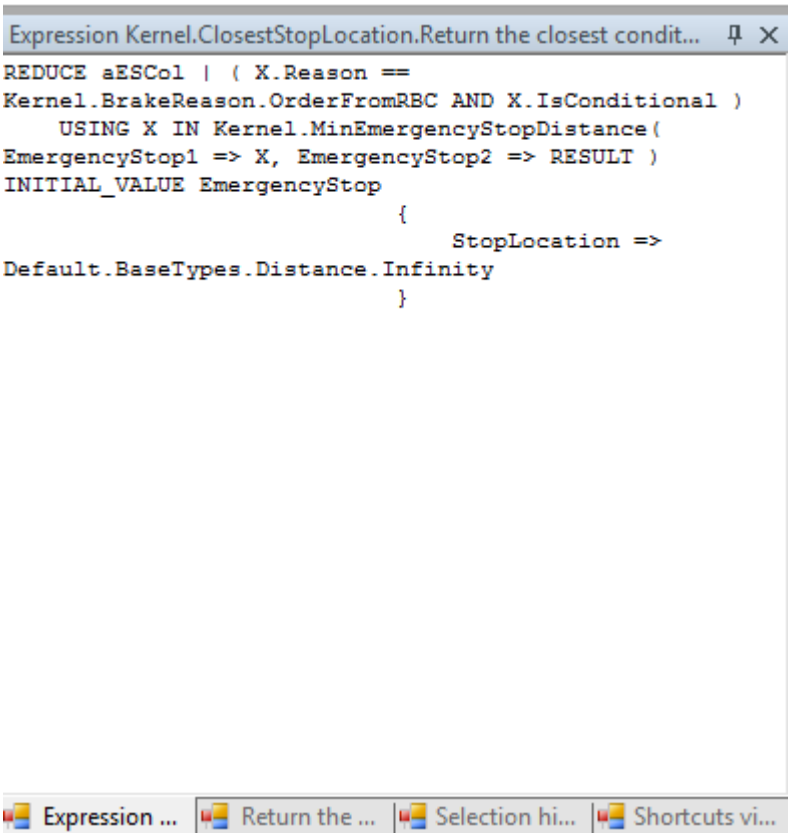


Figure 48: Representation of the Expression.

- **Comment**: provides meta-information related to the selected item. This window is used to displays and edit the selected item's comments. These comments can also be accessed using the properties of the element (see section 7.6).

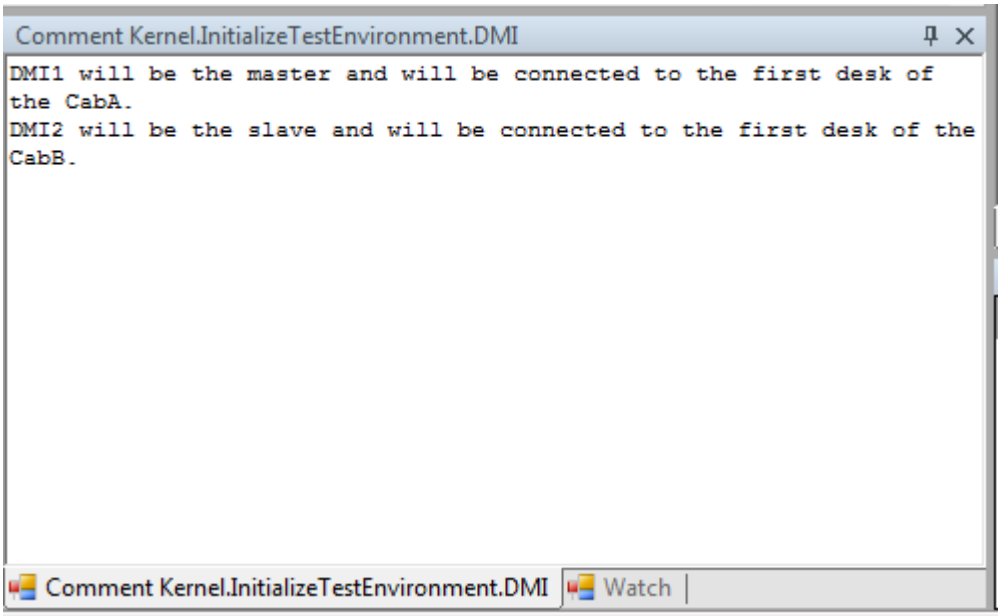


Figure 49: representation of the comment section.

- **Watch**: during execution of a sequence, the watch window provides the value related to user's expressions. Drag & drop variables can be used to add visualization of such variables in the Watch window.

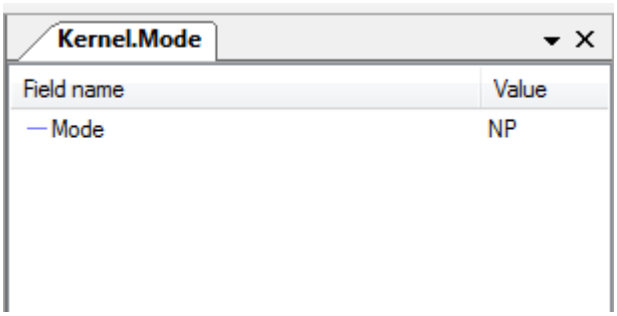


Figure 50: Representation of the Display window

- Double clicking on the **Expression** section of the **Watch** window opens the **Expression** editor and allows to edit the expression. See Figure 51.

Figure 51: Expression editor.

- **Usage**: the usage view allow to see where a specific model element has been used. It is available for
- **types**: it provides the variables that are instances of the corresponding type
- **variables:** it provides the locations where the variables is either read (using the symbol ) or written (using the symbol )
- **functions:** it provides the location where a function is called (using the symbol ).

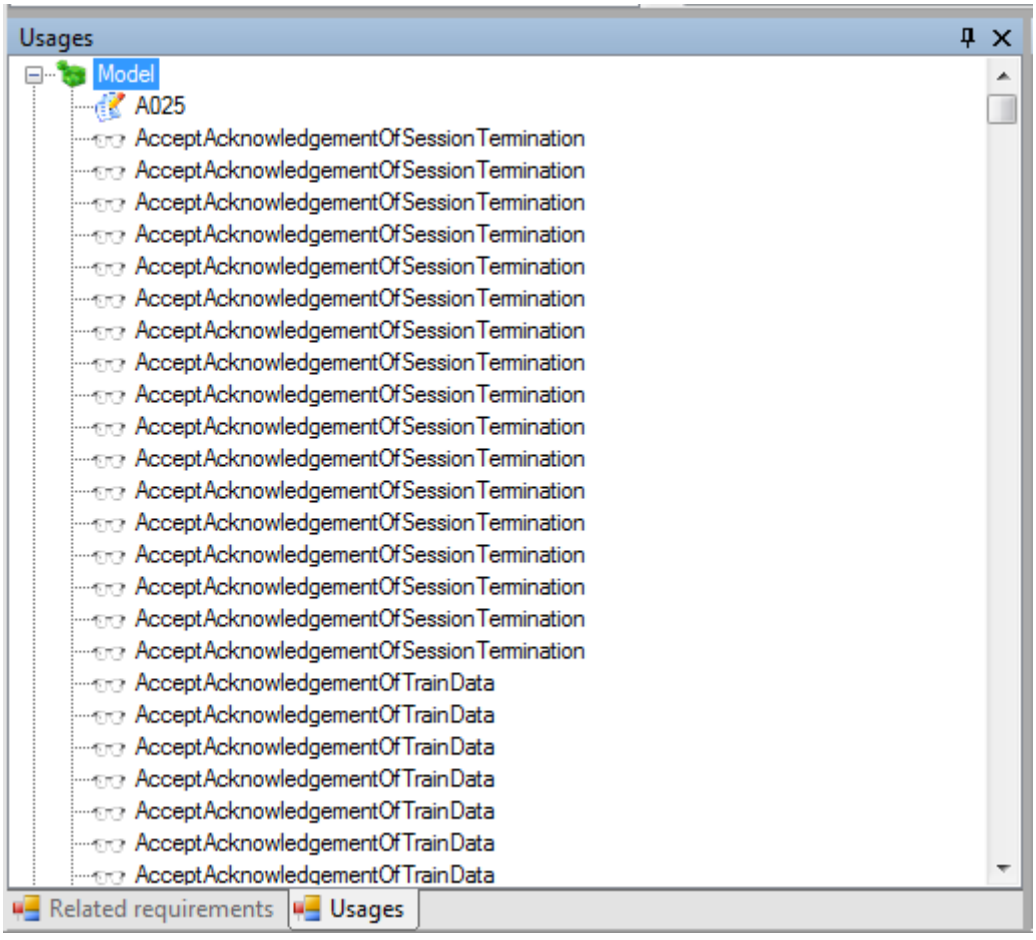


Figure 52: usage representation.

- **Related Requirements**: displays the requirements which are related to the selected element of the model.

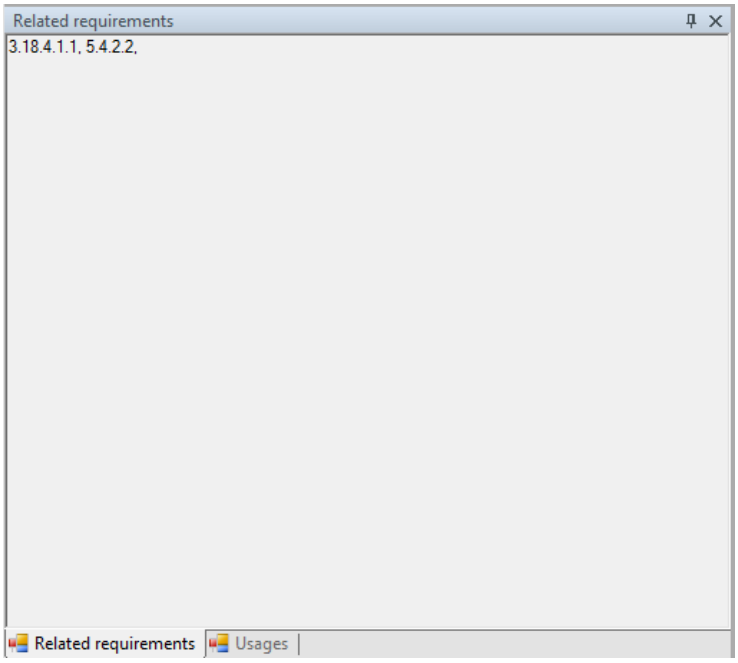


Figure 53: representation of the Related Requirements on the EFSW.

9.2.2 Common properties

Next to follow all the common properties for the data types are described.

- **Default value**: indicates the given value of the current selected element of the model when creating an instance of it.
- **Default value**: it is the given value if there is no value assigned to the selected item of the hierarchical tree when using it.
- **Implemented**: indicates the implementation status of the current selected item of the hierarchical tree. If this metadata is set to true means that the current selected range has been implemented, otherwise it is set to false.
- **Verified**: indicates the revision status of the element of the hierarchical tree. If the flag is set to true, it means that has been satisfactory reviewed, otherwise must be set to false.

9.3 The model

9.3.1 Data types

The data types which are described on this section are the built-in data types present on the EFSW. According to the needs of the modeller new data types can be created; the created data types are always result of inheriting from the built-in data types and have the same properties.

9.3.1.1 Operations performed on the ERTMSFormalSpecs Model for all the data types

The operations which can be performed over all the ranges are the following:

- Create new
- Delete existing

9.3.1.1.1.1 Add a new element on the hierarchical tree

To create a new element, select the proper folder in which the element should be created, right-click to open a contextual menu and select Add to add the new element. For instance, Figure 54 shows how to add a new range in the model.

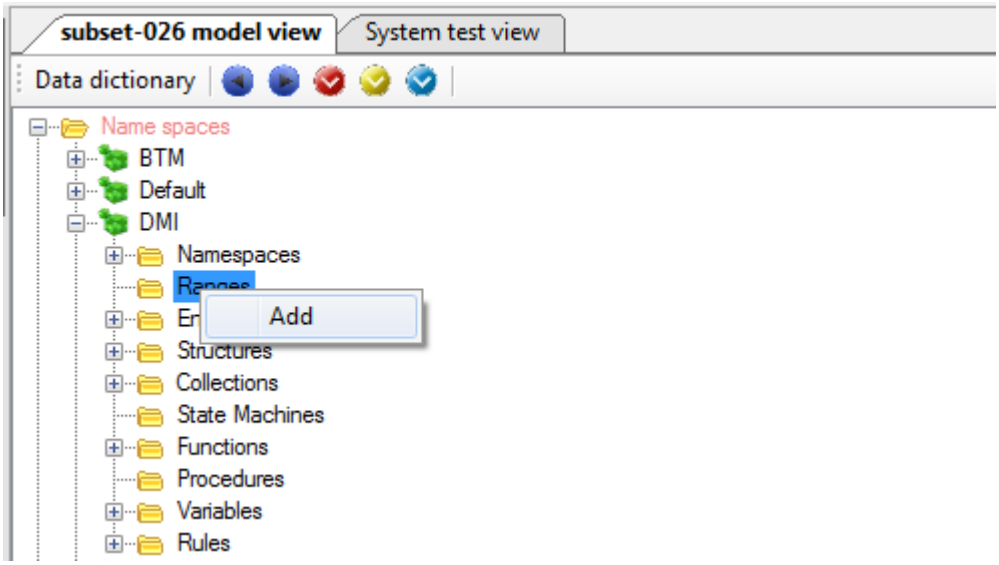


Figure 54: add a new element on the hierarchical tree

The newly created element is added with a default name, which should be modified using the property view (see 9.2.2), along with all the properties specific to that element.

9.3.1.1.1.2 Delete an existing element of the hierarchical tree

Additionally the EFSW allows to delete an existing element on the model. Select the element to be deleted, right-click on it and choose the Deleted option on the contextual menu. Once the delete option has been used the element disappears from the hierarchical tree.

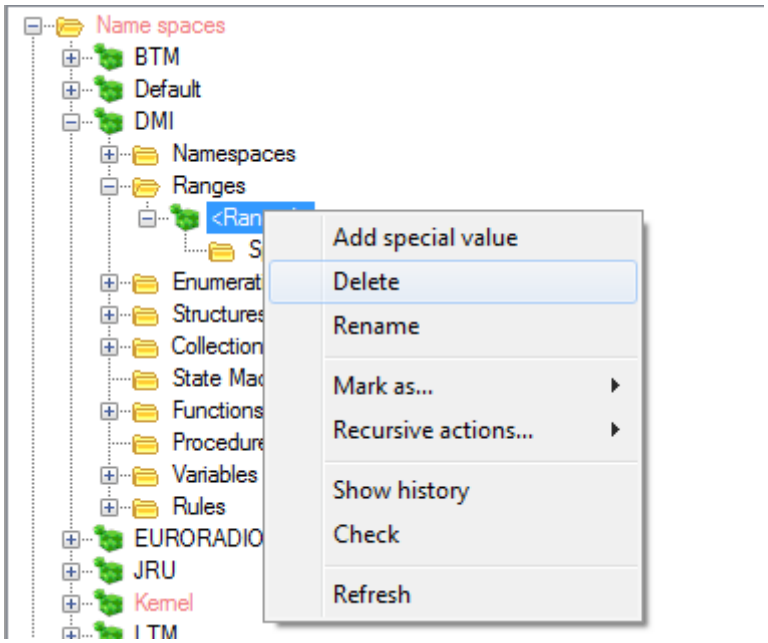


Figure 55: delete an existing element from the hierarchical tree

9.3.1.2 Ranges data type

The **Ranges** are used for numerical representations with a high and low bound. For all the ranges must be given the precision of the range. That means the numerical separation between two consecutive elements of the range. There are two cases:

- Integer: the minimal separation is a natural unit[1].
- Floating point: the minimal separation is a fraction of a natural unit[2].

Additionally to the described possible values, **ranges** can have values with a special meaning for the model, these are called **special values**.

All the model elements defined as a range data type have the same properties which are described below. Figure 56 shows the properties of ranges[3]:

- **Max value**: represents the high limit of the possible values a range can be given.
- **Min value**: represents the low limit of the possible values a range can be given.
- **Precision**: separation between two consecutive values of the range. The possible options for this property are:
- **Integer**: the values of the elements of the range are integers.
- **Floating** point: the possible values of the range are not integers.

[1] The natural unit is given by 1.
[2] A fraction of the natural unit is given by: 1/X; where X > 1
[3] For further information about ranges, see section 9.3.1.2.

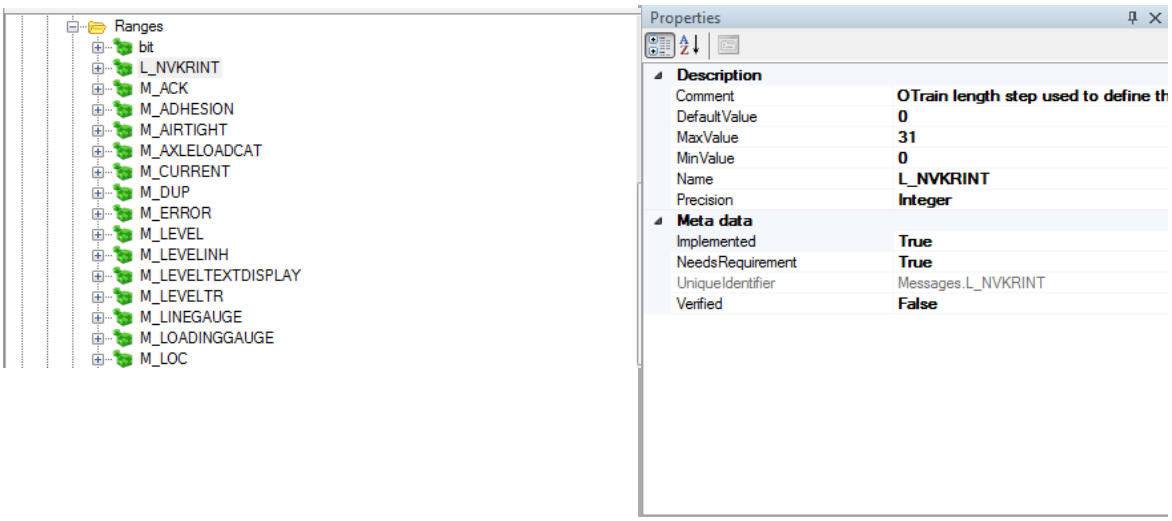


Figure 56: ranges properties

9.3.1.3 Enumeration data type

Enumerations specify a set of discrete values identified by a name. Normally the possible discrete values defined on an enumeration are represented by chains of characters and has an integer number associated to each discrete value.

An enumeration can contain sub-enumerations, representing subgroups of the enumeration's values. For instance the Mode enumeration (representing the modes of a European Vital Computer on a trainborne) has a sub-enumeration "Mode.MA", which contains all the possible modes that can be received in a Movement Authority. Figure 57 describes the properties of an enumeration.

- **Values**: this is the collection of possible values that can be given to the current enumeration.

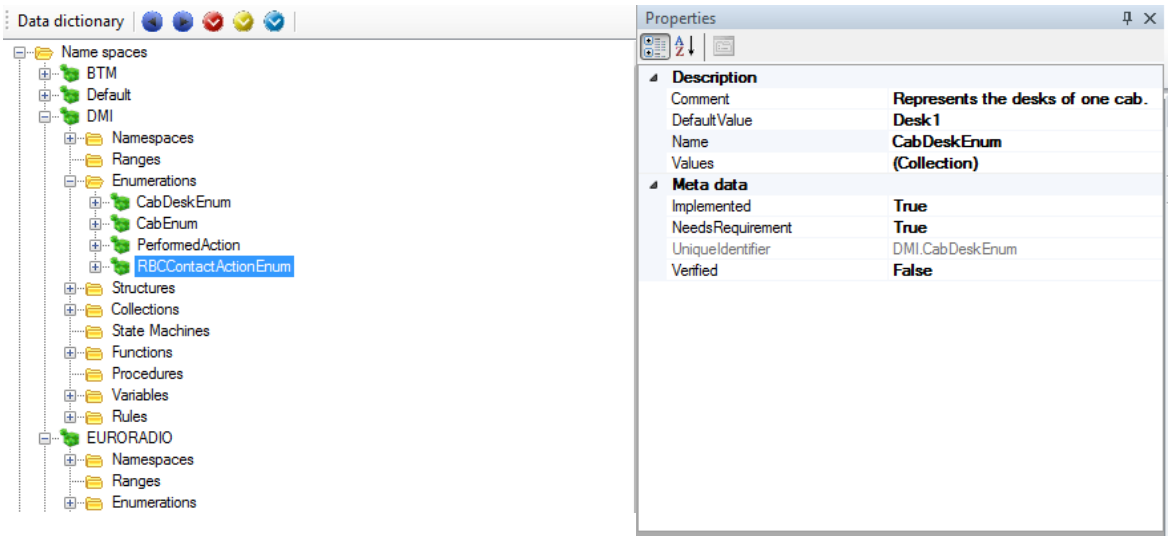


Figure 57: enumeration properties

9.3.1.4 Structure data type

The EFS **Structure** matches the C-like *struct*. All the structures present on the EFSW can contain different sub-elements with different types, for instance, a EFS structure can contain other structures (called sub-structures), rules, procedures, ….

The presence of **procedures** and **rules** on the structures provide to it a dynamic behavior (see sections 9.3.2.3 and 9.3.2.5 for further details). Figure 58 shows the structures properties

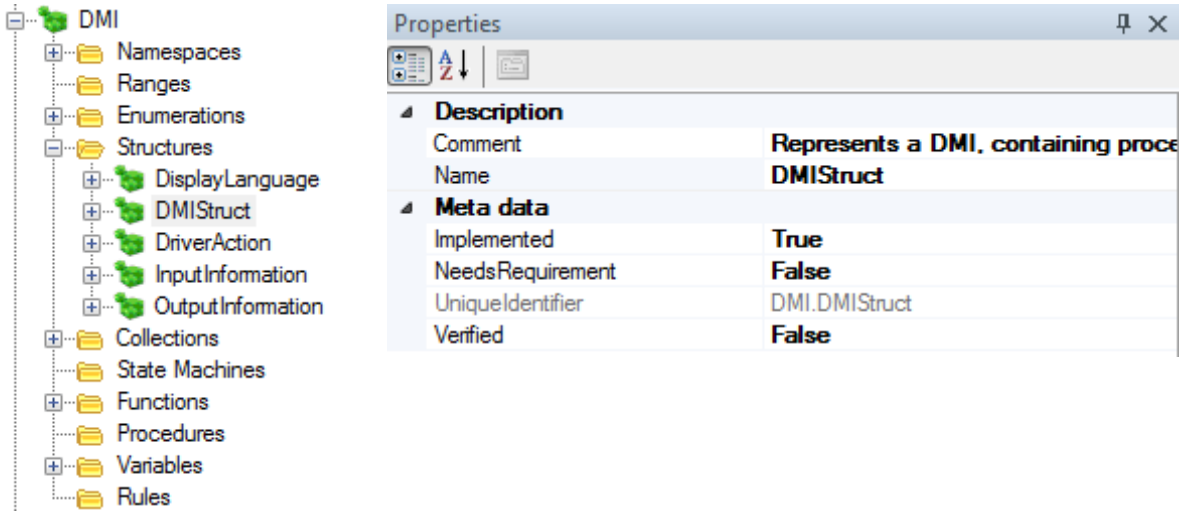


Figure 58: structure properties

9.3.1.5 Collection data type

EFSM **Collections** allow listing and grouping several elements of the same type. As a general fact, most of the collections present on EFSW contains empty for the default value at its creation time. So it means that for using them, at least, on element need to be added to the collection.

In order to add a new value on a collection it is used a dynamic component of the EFSM, as a procedure or rule; for further details about these components see section 9.3.2.3 and section 9.3.2.5.

Collections have a fixed maximum size Figure 59 shows the properties of a collection:

- **Max size**: represents the highest amount of elements that can be allocated on the current selected collection
- **Type**: represents the kind of elements to be allocated on the selected collection. Only elements which match the type of the collection can be allocated on it.

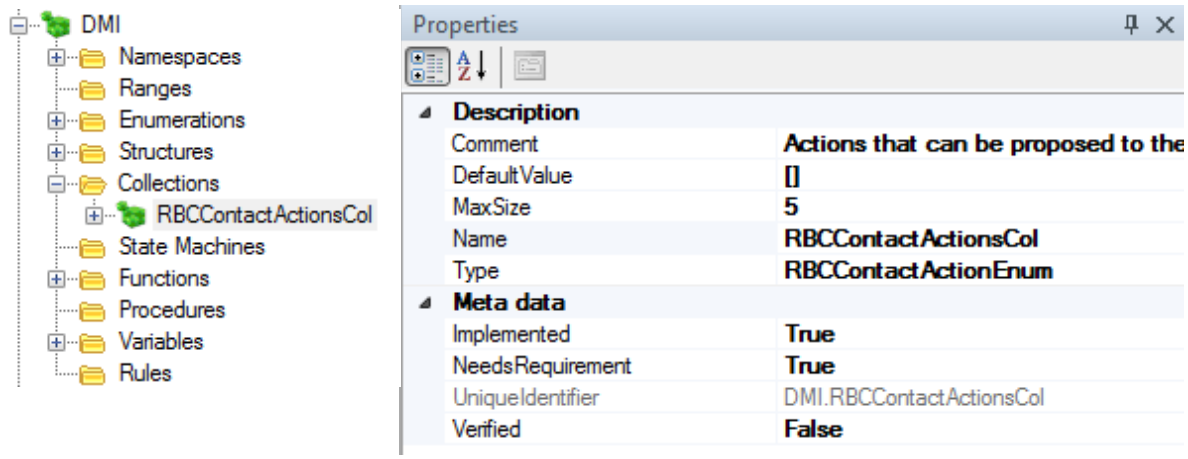


Figure 59: collection properties

9.3.1.6 State machine data type

The EFSW **state machine** matches the traditional concept of finite state machine. Is built by an initial state[4], a group of possible states and a set of **rules** (see section 9.3.2.5 for further details about **rules**).

All the state machines are graphically represented by a state diagram. The state diagram representation of the state machine is available through the state diagram view, as depicted by Figure 61. Figure 60 illustrates in detail the properties and miscellaneous of a state machine:

- **Initial state**: indicates which the initial state of the selected state machine is.

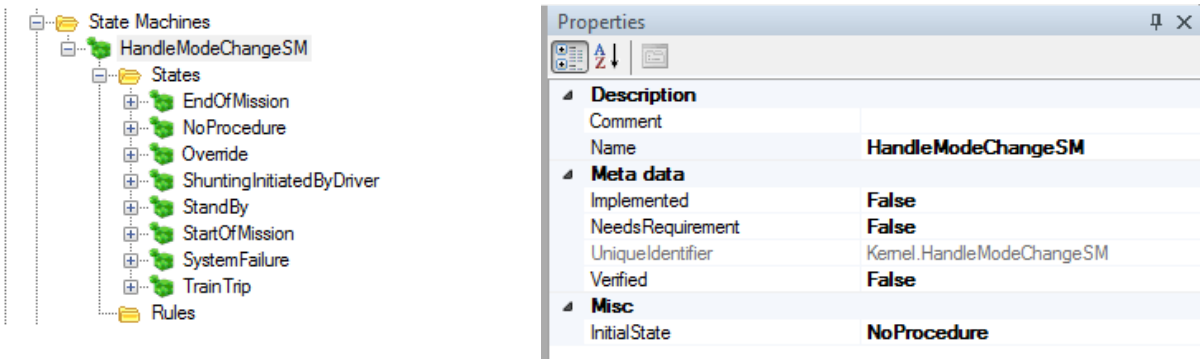


Figure 60: state machine properties

[4] The Initial State is a particular case of a State.

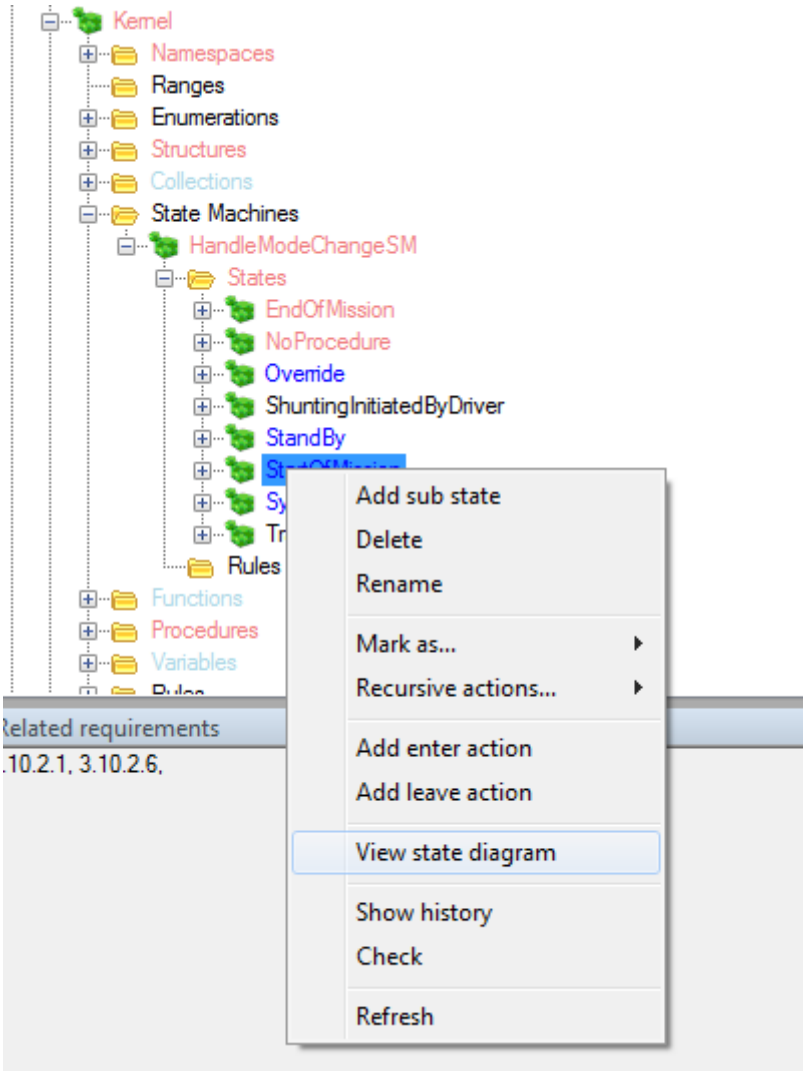


Figure 61: opening the state diagram view.

A **state machine**, as said before, is built by a set of possible states. A state is the basic block used to build a **State Machine**, and it is characterized by a name which identifies it on the current **state machine**. Each state can be decomposed by a new state machine or sub-states.

The second building block for state machines are transitions. Transitions are inferred from the model using the following pattern: a rule which, is either defined in state S1 or which checks in its preconditions that the system is in a specific state S1, and modifies the state to a new value S2 is a transition from S1 to S2, and displayed as an arrow from state S1 to state S2 in the state diagram. That arrow may take one of the two colors

- Black: the rule is defined in a state of the state machine
- Purple: the rule is defined in the model, outside any state of the state machine

This has the advantage to differentiate transitions which are explicitly described in the state machine in Subset-026 (for instance transition from S0 to S1 (see section 9.3.2.5.1.1), or from S1 to D2; Figure 62 depicts both cases of explicit definition of transitions) from the implicit transitions, that is transition which are defined somewhere else in the specification (for instance, transition named

"After D11" from S24 to S10. This transition has been created according to requirement 5.4.5.1 from Subset-026).

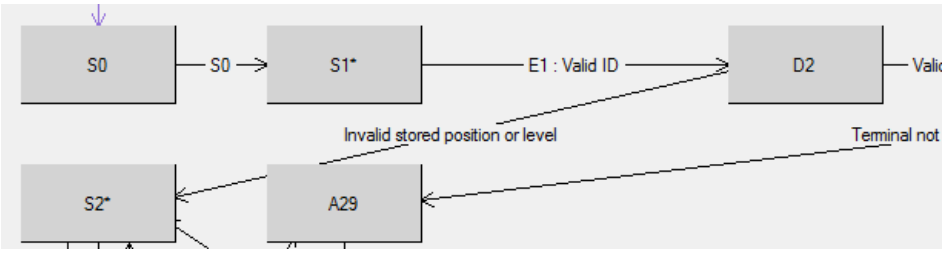


Figure 62 : explicit transition in a state diagram

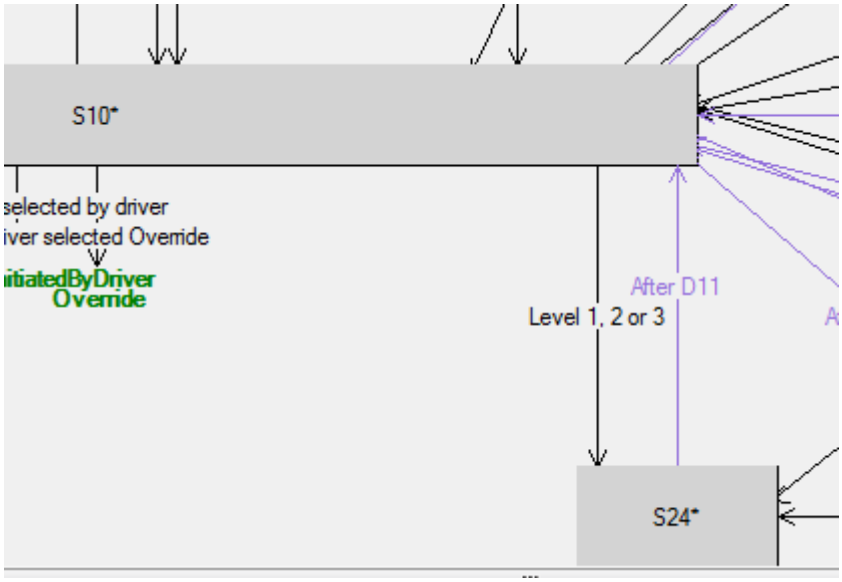


Figure 63 : implicit transition in a state diagram

9.3.1.6.1.1 State diagrams

The state diagram of a state machine described on EFSW can be visualized using state diagram view.

9.3.1.6.1.2 Display a state diagram

To display a state diagram, select the corresponding state machine in the data dictionary window, in the contextual menu appearing after using the right button of the mouse, select View state diagram.

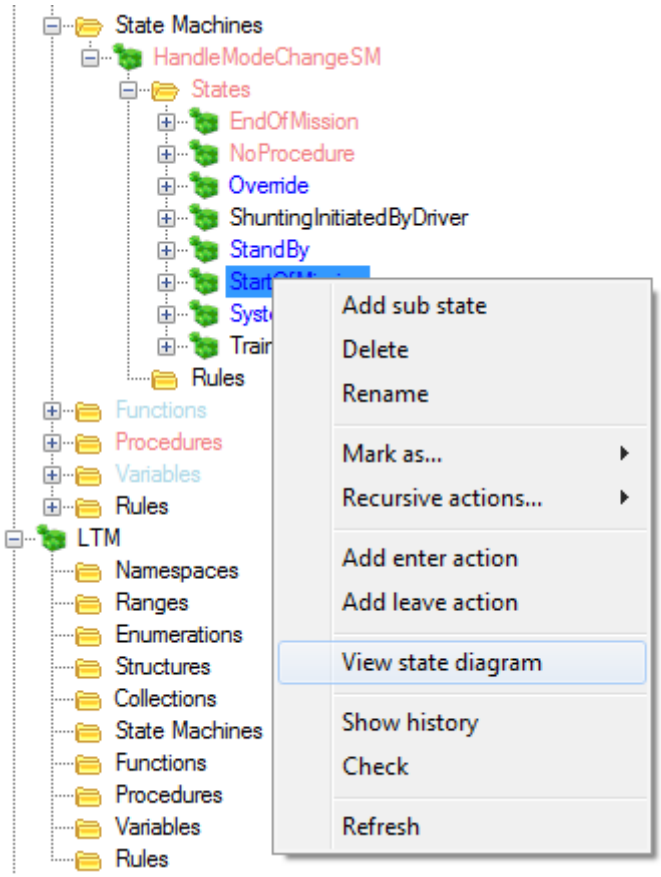


Figure 64: View the state diagram associated to a procedure

Based on the state machine information and on the rules of the model, the EFSW builds the state diagram representation of the state machine and displays it in a new window. Figure 65 depicts an example of state diagram.

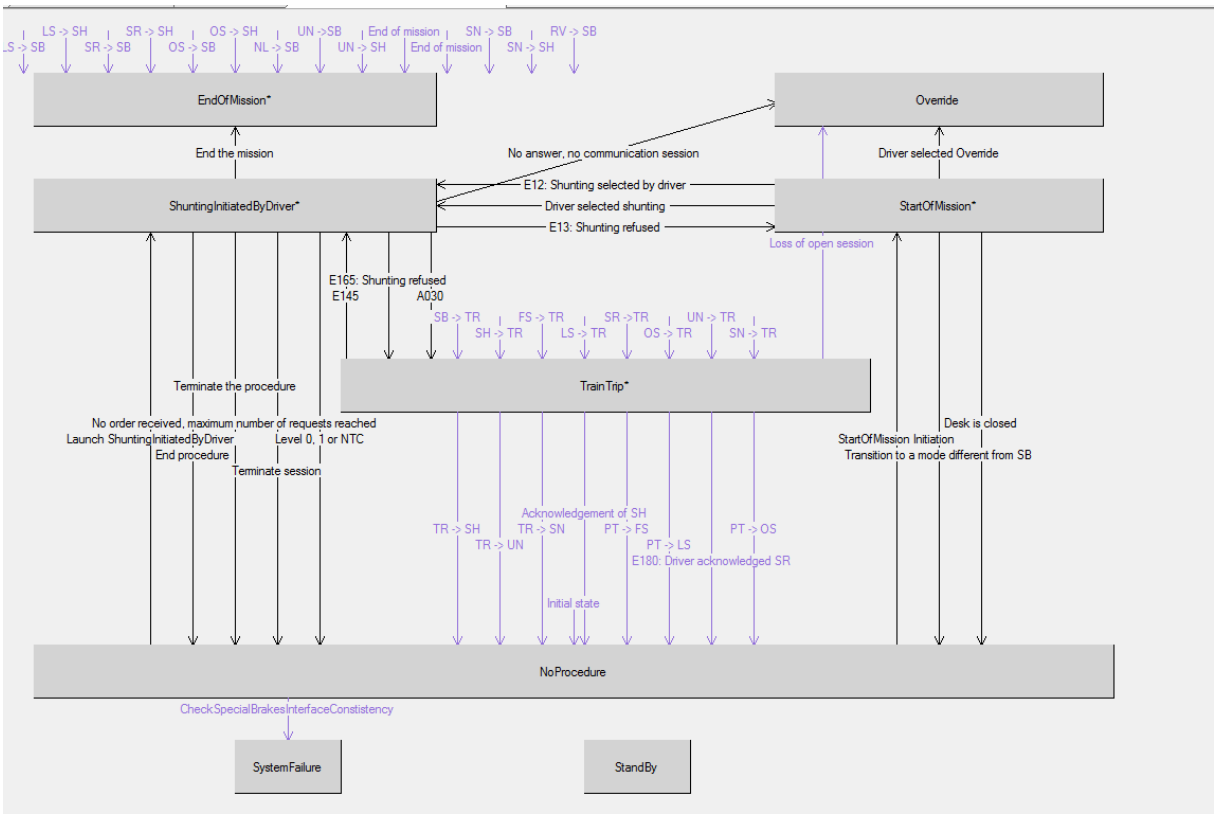


Figure 65: State diagram view

Double clicking on a state opens the sub-state machine related to that state. For instance, Figure 66 shows the sub state diagram related to the *StartOfMission* state machine.

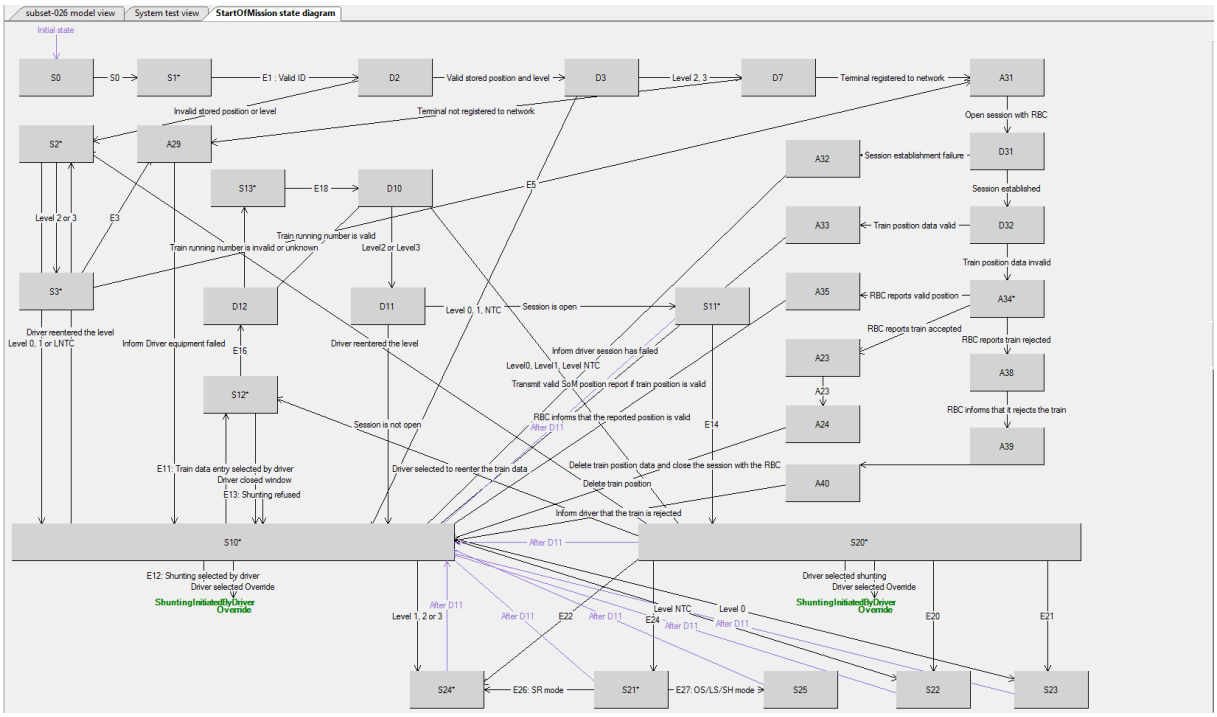


Figure 66: Sub-states of the state StartOfMission.

9.3.1.6.1.3 Manipulations of a state diagram

New states and new rules can be added in the state diagram using the graphical view, thanks to the contextual view.

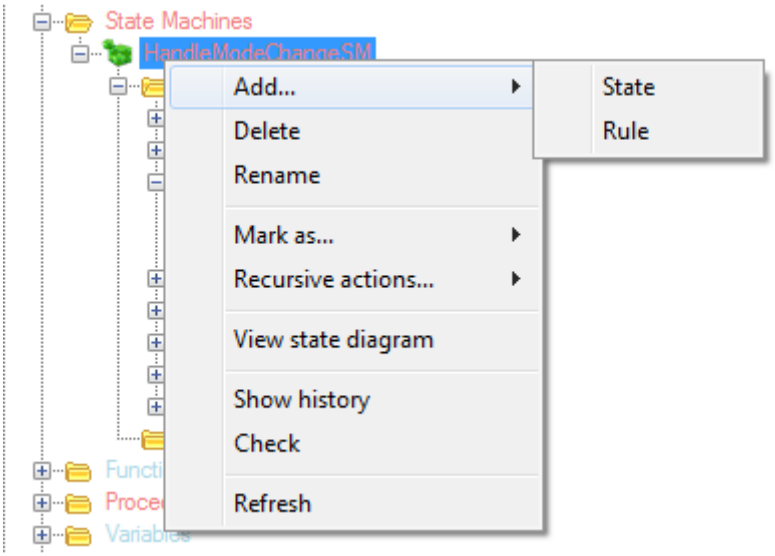


Figure 67: Contextual menu for a state diagram

9.3.1.6.1.4 Add a new state

To add a new state, select Add state in the state diagram window contextual menu. This creates a new state in the corresponding state machine and updates the state diagram accordingly.

9.3.1.6.1.5 Add a new transition

A new rule can be added by selecting Add rule in the same contextual window. This create a new transaction associated to this state machine.

9.3.1.6.1.6 Selecting element in a state diagram

When a state or a transition is selected, its details are displayed in the right side of the window and the corresponding element is selected in the Model.

9.3.1.7 Traceability

Traceability information can be added to data types by dragging a requirement to the related data type.

The **related requirements**, presented as sub-nodes of the data type on the left part of the window. They indicate the requirements that are modelled by the corresponding data type. One can add a new requirement to this list by **Drag&Drop** between the requirement expressed in the specification view and the data type requirement node.

9.3.2 Model elements description

9.3.2.1 Namespaces

Namespaces are used to group model elements together. Figure 68 displays a typical namespace graphical view, holding types, variables, functions, procedures and sub-namespaces. Each namespace allows to define

- sub namespaces
- Various **data types** needed to support the model.
- **Procedures** and **variables** which model the system state.
- **Functions** which factorize common computations.
- **Rules** which specify the system dynamics.

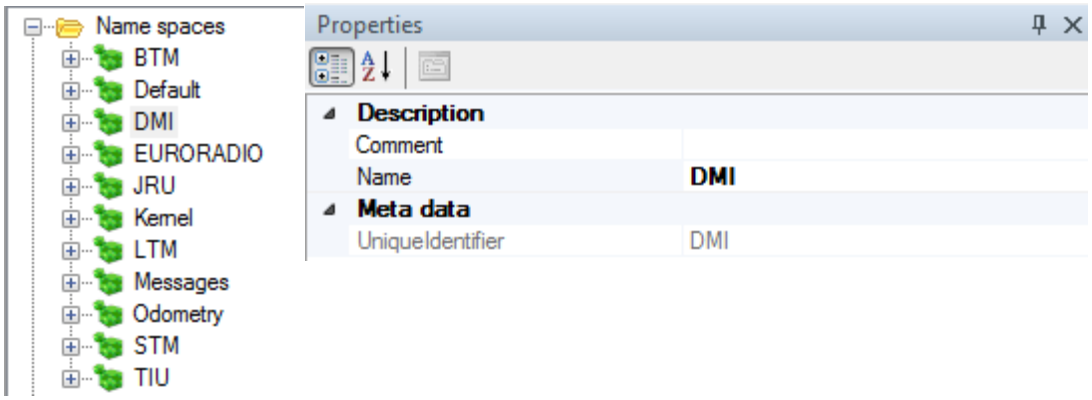


Figure 68: namespace view

For instance Figure 69 displays the following namespaces: *BTM*, *Default*, *DMI*, *EURORADIO*, *JRU*, *Kernel, Messages, Odometry*, *STM* and *TIU*.

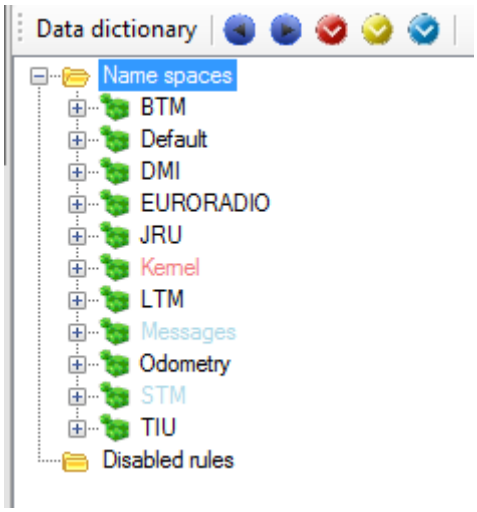


Figure 69: available namespaces on the EFSW

The EFSW allows to display the functional view of the namespaces. For this, right-clicking on the desired namespace and select the functional view option on the contextual menu. As shown in Figure 70, the functional view displays the present sub-namespaces on a namespace.

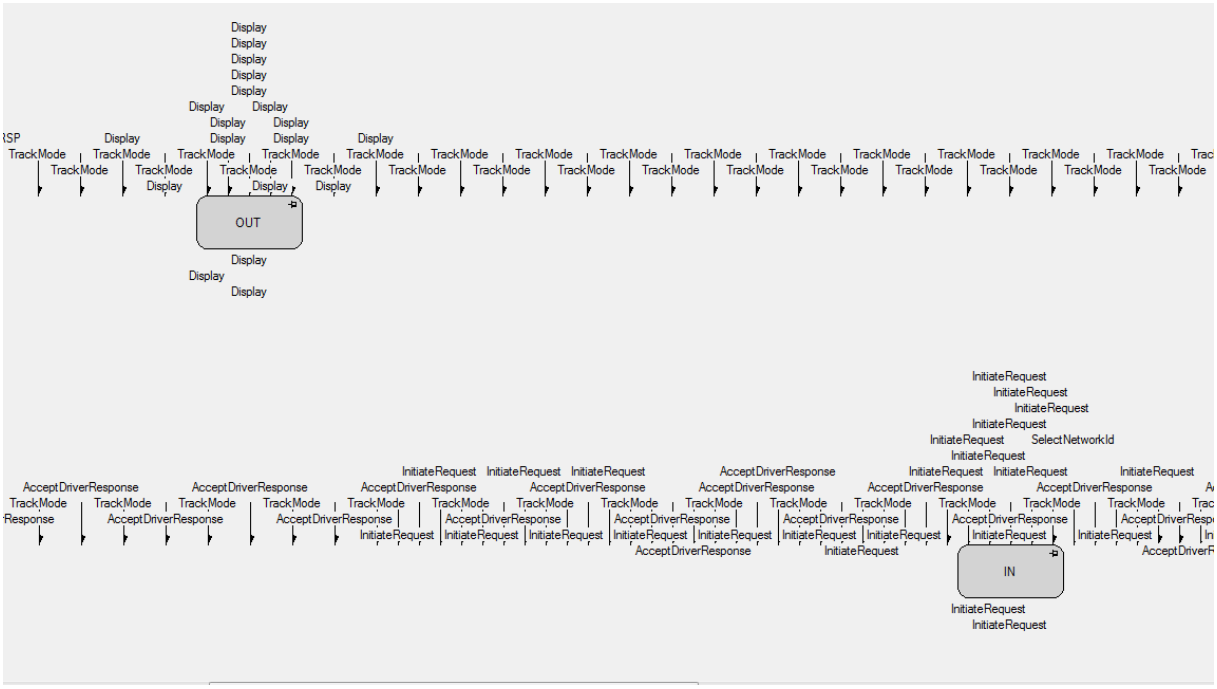


Figure 70: functional view of the DMI namespace

9.3.2.2 Functions

Functions allow to define a functional computation, e.g. allow to compute a value according to the parameters' values and the current system's state. A function is split in a set of cases, each one mutually exclusive. The first case whose preconditions are satisfied is used to compute the function's value.

$$f(x) = \begin{cases} x^2+1 & \text{if } x > 0 \text{ and } x < 100 \\ 2x & \text{if } x > 100 \end{cases}$$

Equation 1: function definition.

Figure 71 shows the properties of a function:

- **Is cacheable**: indicates that if each time the selected function is called during a cycle, the internal computation of the function must be done, or the result of the first call can be stored and re-used during a cycle. The flag set a true means that the result is stored after the first call on a cycle and re-used. When it is set to false each time it is called the result must be computed.
- **Type**: represents the kind of elements returned by the selected function as result of its computations.

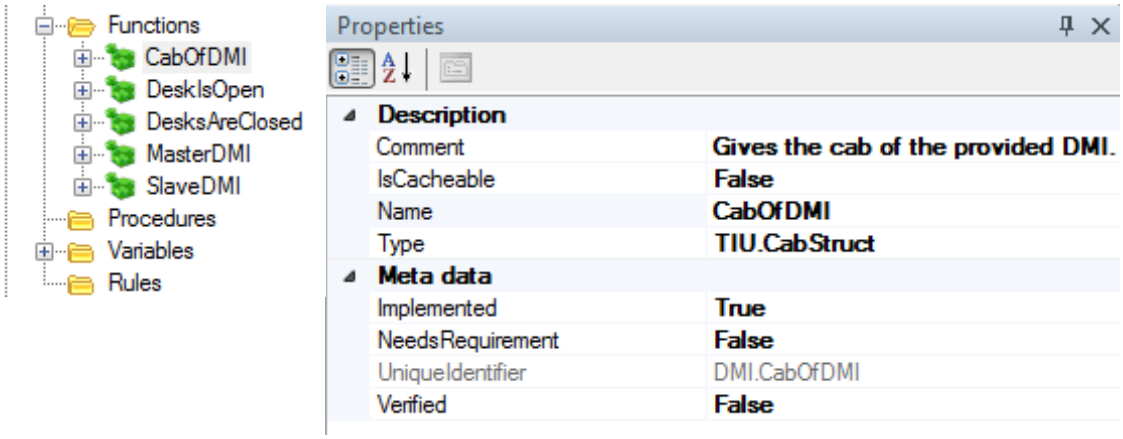


Figure 71: function properties

Functions allow factorizing common computations in a single location. They are identified by a unique name and their return type and can have several parameters. Functions can be used to model complex functions[5] such as:

Figure 72 depicts a graphical example of a function permitting to calculate the confidence interval of a train.

[5] This kind of function is mainly used in braking curve computation

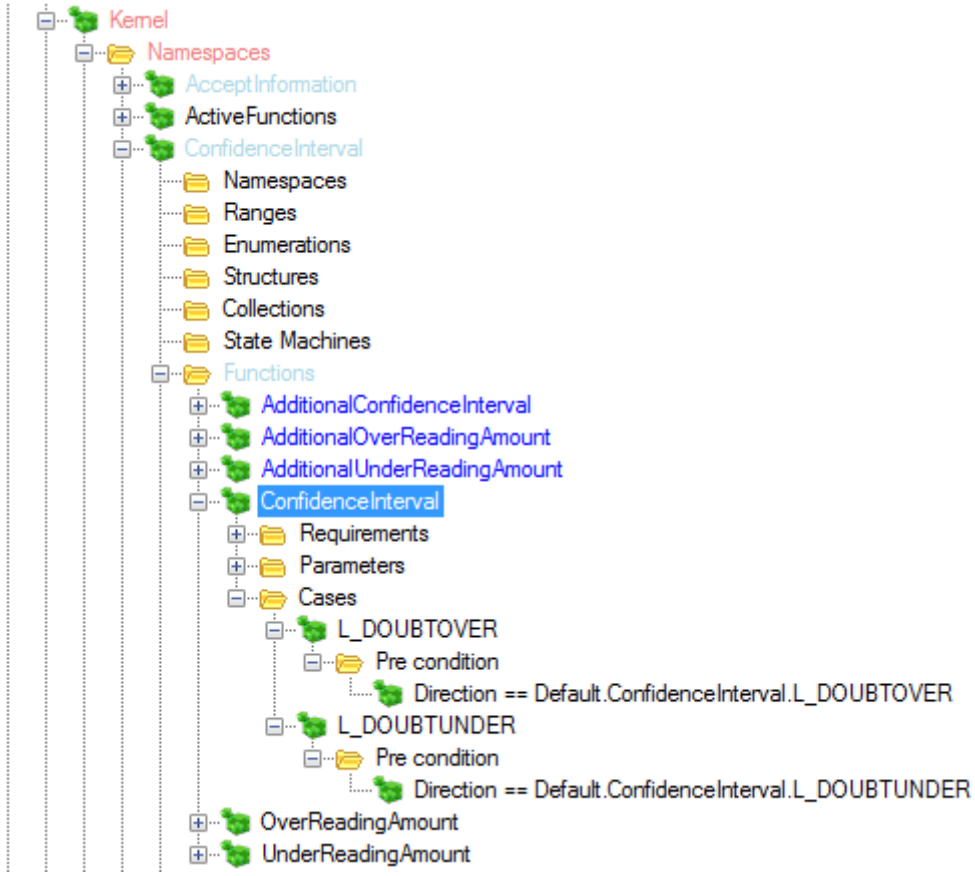


Figure 72: Example of a function

Functions manipulations (add/add Parameter/add Case/remove) are performed using the contextual menu, as Figure 73 and Figure 74 depict. Properties about functions are altered using the property view on the right part of the Model main window.

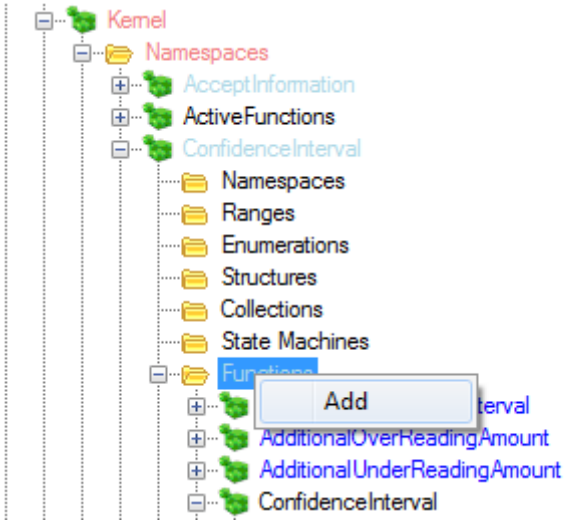


Figure 73: contextual menu used to add a function.

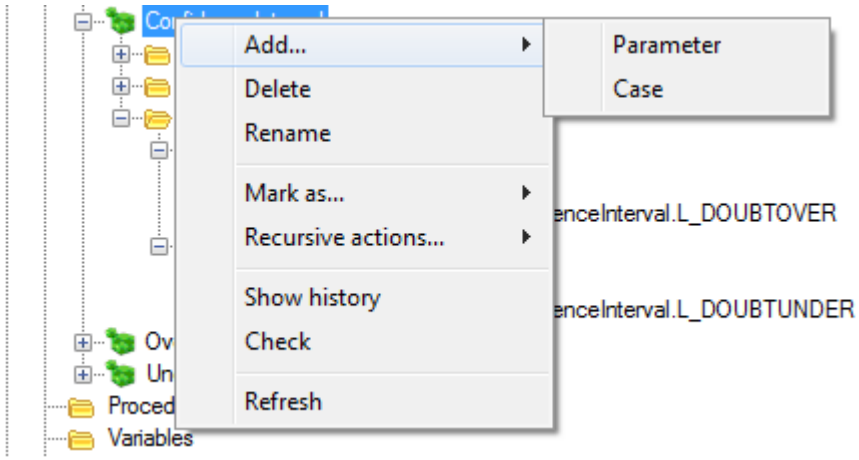


Figure 74: contextual menu which allows to add parameters or cases to a function and deleting a function

9.3.2.3 Procedures

Procedures are used to perform a sequence of actions, in a sequential fashion. This is the only case where imperative programming is available in EFS. In the case of a procedure, the first statement is executed before executing the next one. All procedures is characterized by

- A **name**, which identifies this procedure in the system.
- **Parameters**: the formal parameters for this procedure.
- **Rules**: the rules to be activated when the procedure is called.
- A **Comment,** which gives a brief description of the functionality of the current **Procedure**.

The procedure[6] properties are composed by its **name**, a **comment**, **implementation status**, **needs requirement**, **verification status** and its **unique identifier**. Figure 75 mirrors these properties.

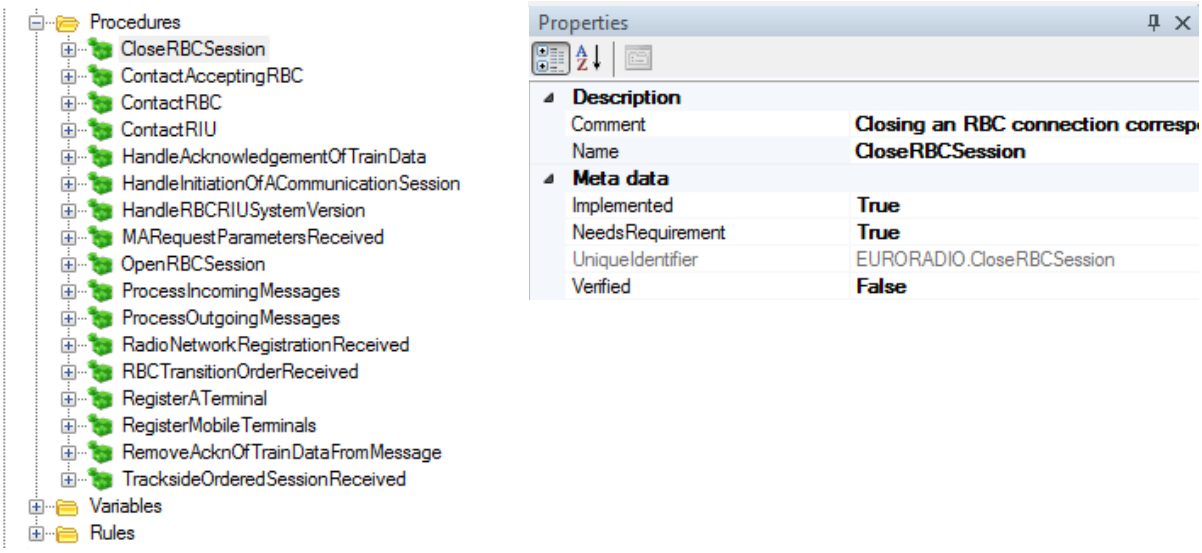


Figure 75: procedure properties

[6] For further information about variables, see section 9.3.2.4

9.3.2.4 Variables

Variables are used to store the value used to describe the system's state. **Variables** are characterized by:

- A **name**, which identifies, along with the enclosing sub system/variable, the variable in the system.
- A **type**, as defined above.
- A **default value**, which overrides the default value specified for the type (if any).
- A **mode**:
  - Internal: the variable is used by the EFS model only.
  - Out: the variable is written by the EFS model and read by the outside world.
  - In: the variable is written by the outside world and read by the EFS model.
  - In/Out: the variable is used by both the outside world and the EFS model.
  - Constant: the variable is initialized at EFS start up. Its values then never changes. It pertains to the "Data Prep".
- A **value**, which is the current value of the variable.

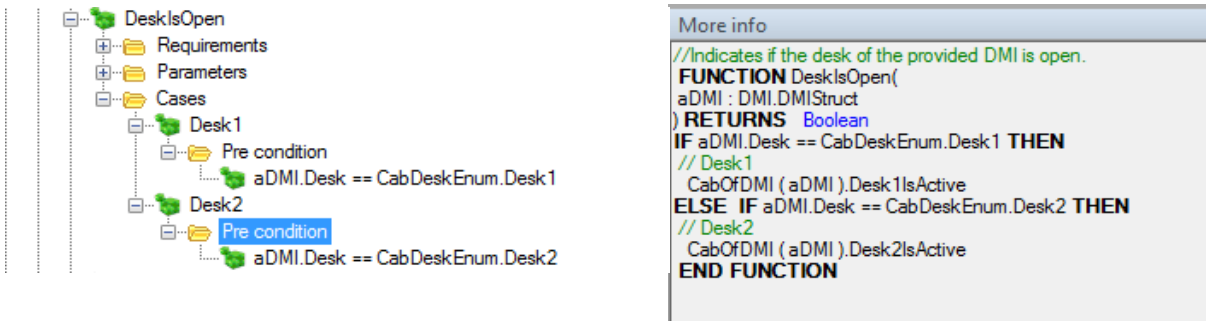


Figure 76: variable representation on EFSW

Figure 79 indicates the available properties of a variable[7]:

[7] For further information about variables, see section 9.3.2.4.

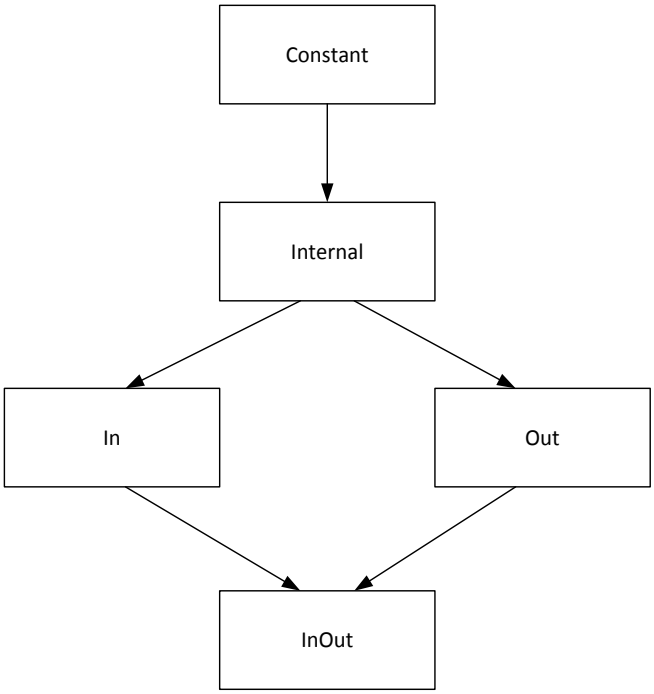


Figure 77: Order of the possible modes of a variable

The mode of a sub-variable must, at least, be higher than the mode of the variable where it is enclosed. For instance, a variable which mode is **In** can contain sub-variables which mode is **internal** or **constant** but never **Out** or **InOut**. Figure 77 depicts the hierarchy of the enclosed variables on a variable. Figure 78 depicts an existing situation whit a variable enclosed on other and both of them have different modes.

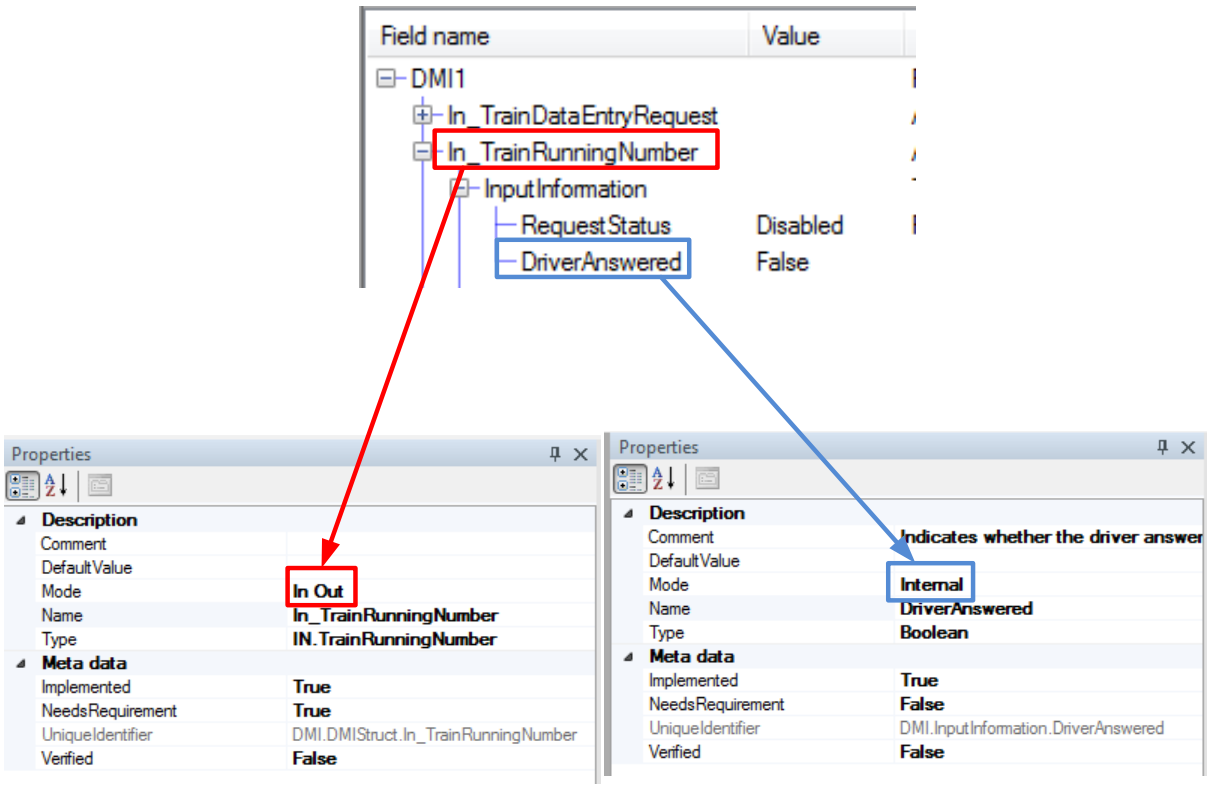


Figure 78: variable which mode is internal and is enclosed on an InOut varaible

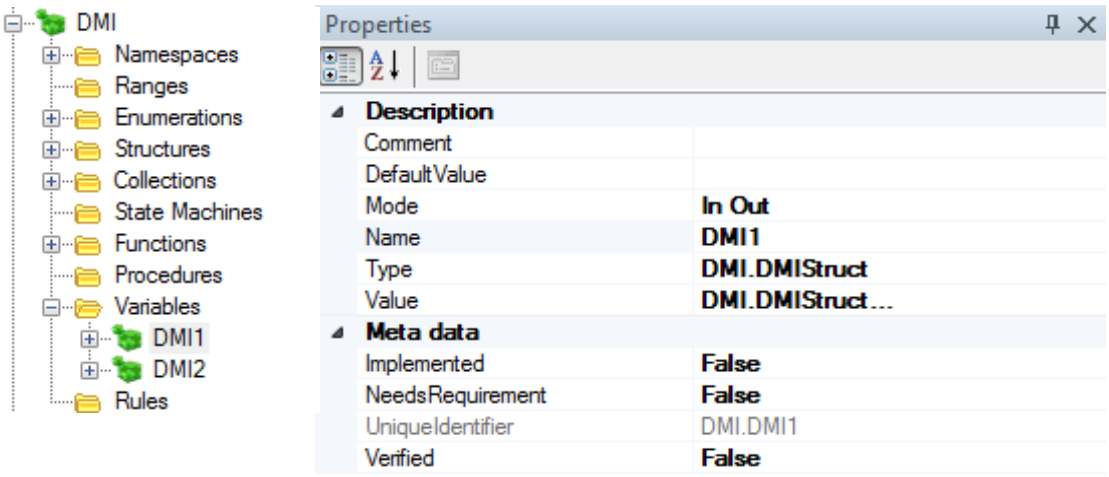


Figure 79: variable properties

Variable manipulations (add/delete) are performed using the contextual menu, see Figure 80 and Figure 81 Properties about variables are altered using the property view on the right part of the Model main window.

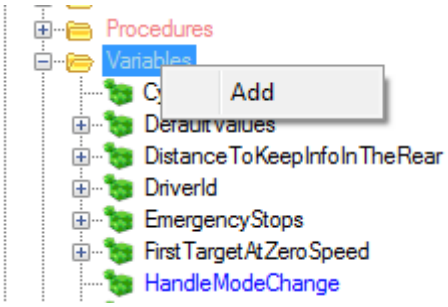


Figure 80: contextual menu for adding a variable.

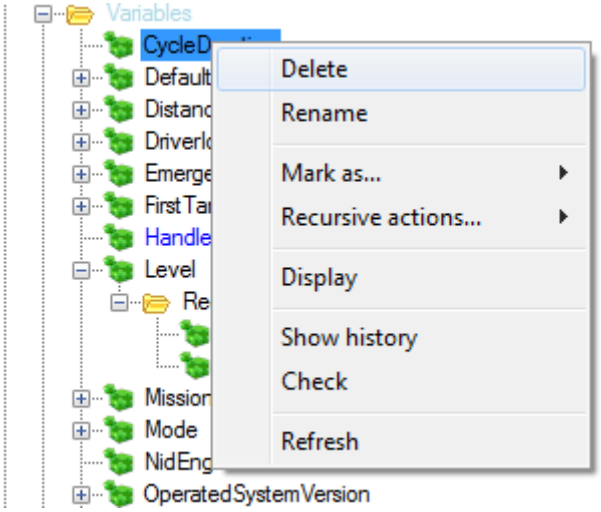


Figure 81: contextual menu for deleting a variable.

9.3.2.4.1.1 Sub variables

Variables on EFSM can contain different sub-elements – sub-variables-. Right-clicking on a variable is displayed the contextual menu offering the possibility of Display the variable's structure. See Figure 82.

After selecting the Display option of the contextual menu, the variable's structure representation is displayed on the right side of EFSW main window. Figure 83 displays the representation of the "*Kernel.Level*" and "*Kernel.FirstTargerAtZeroSpeed*" variables after activating the **Display** option.

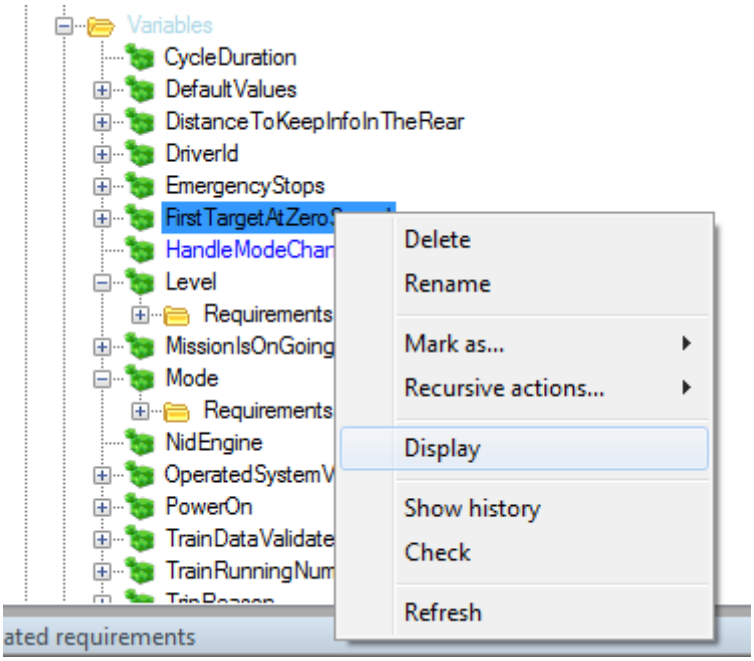


Figure 82: contextual menu for displaying the enclosed sub-variables

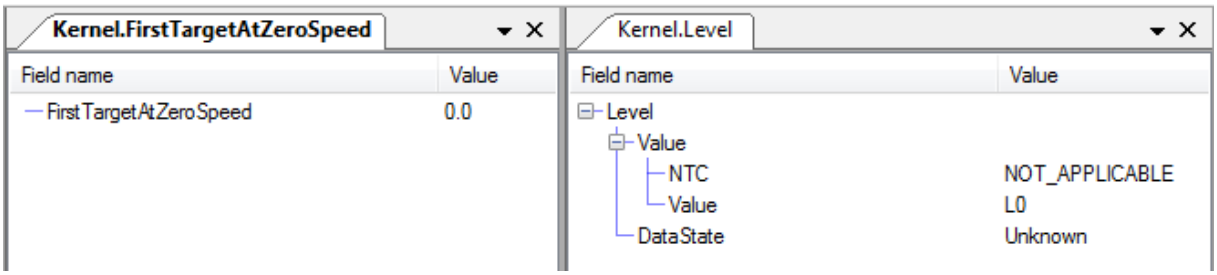


Figure 83: representation of the sub-variables enclosed on a variable.

9.3.2.5 Rules

Rules are used to describe the system's dynamics. All the rules are made by a set of preconditions, actions and sub-rules.

- **Pre-conditions**: Each rule has a list of one or more pre-conditions that must all be true for a rule to be triggered. Each pre-condition is defined by its name, containing the plain text of the expression associated to this pre-condition.
- **Actions**: Each rule contains a set of actions which are executed when the rule is activated. An action is defined by its name, containing the plain text of the expression associated to this action. The action can be a variable update or a procedure call.
- **Sub-rules**: It is possible that a **Rule** contains other rules in its hierarchical tree structure. These are called the sub-rules.
- The mutually exclusive rule conditions. Only one of the rule's conditions can be activated when a rule is activated. When several conditions can be activated, the EFS interpreter activates the first one.

When the rule's preconditions are satisfied, the rules actions are applied on the system, and sub rules can be evaluated.

Rules manipulations (add/remove) are performed using the contextual menu. Reordering of rule conditions inside a rule or of rules inside a procedure can be done by the **Drag&Drop** feature of the EFSW. In this case, for instance, drag the rule or rule condition to be moved and drop it on the rule or rule condition which will follow it while holding down the key "alt"[8]. Properties about rules are altered using the property view on the right part of the Model main window.

The processing followed by EFS follows a cycle: after inputs has been provided, an **Input Verification** phase is applied, followed by the **update of the internal state.** After this step completes, the actual **business process** is executed, followed by an **update of the output variables**. The external world can then retrieve the output values before a **clean up phase** is applied on the system. This is depicted by Figure 84.

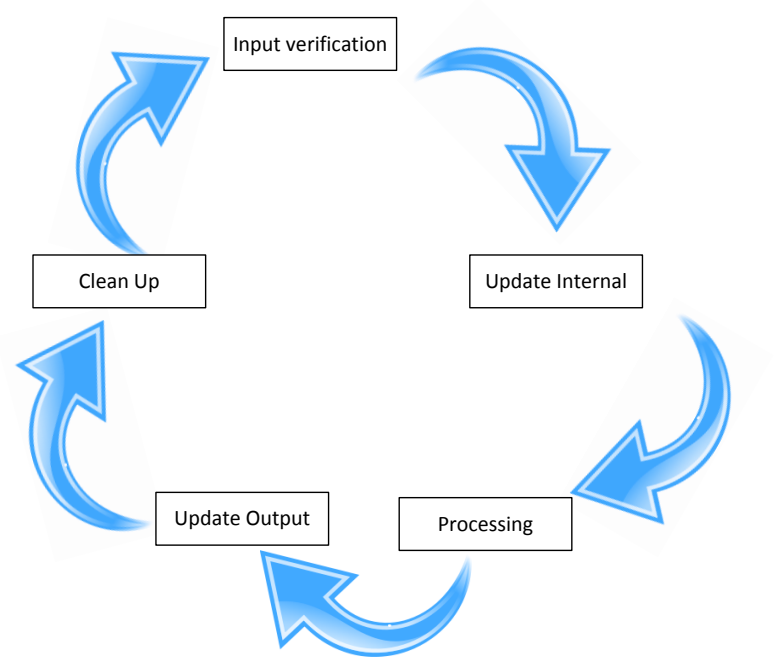


Figure 84: diagram of the cycle on ERTMSFormalSpecs

Figure 85 presents the properties of a rule[9]:

- The rule **priority**. This indicates the part of the activation cycle in which the rule can be activated. The priorities are the following
- **Input verification**: First part of the activation cycle. This phase handles verification of the IN variables.
- **Update internal** variables, this part of the cycle is performed after the input verification is complete. It updates the internal variables according to the data found in input data.
- **Processing.** This phase is performed after the update internal variable phase is complete. It is the main processing for the system.
- **Update output** variables. Based on the result of the processing phase, the output variables are updated during the update output phase.

[8] This is not only applicable for the rules or rules conditions but for all the elements on the hierarchical tree of the Model.
[9] For further information about rules, see section 9.3.2.5.

- **Clean Up**: Cleans the contents of variables used during this cycle. This is the last step of the cycle.

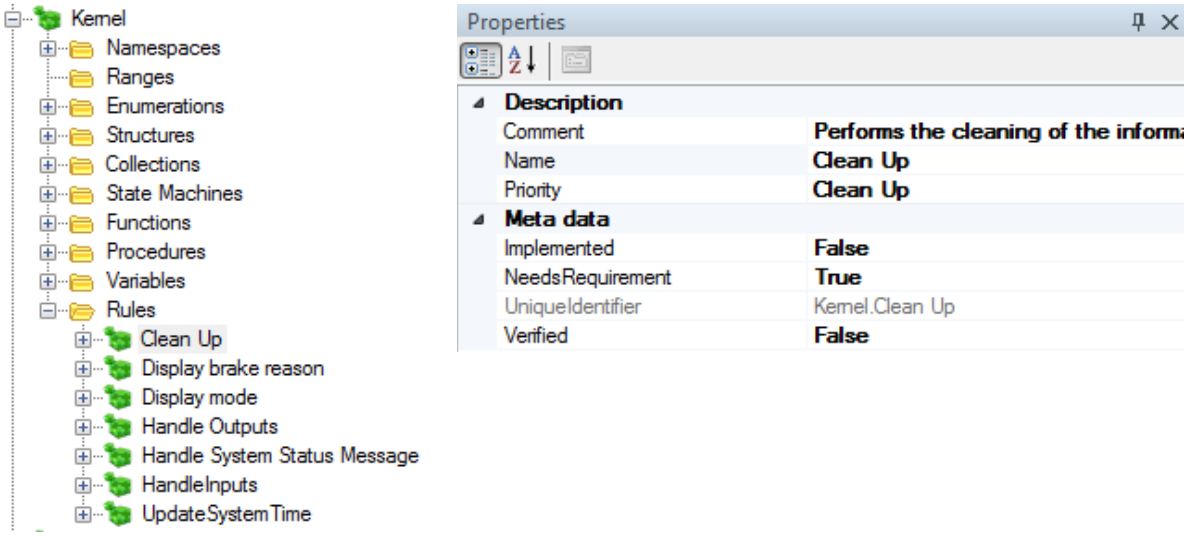


Figure 85: rule properties

Figure 86 illustrates an example of a rule. This rule is in charge of deleting the present messages on the JRU messages collection at the end of each cycle; which means the rule is executed on the clean-up part of the cycle.

Figure 86: rule representation

9.3.2.5.1.1 Specific case: a rule declared in a state

When a rule is declared in a state of a **State Machine**, it is only triggered when (as usual) its pre-conditions are satisfied but also when the current state of the **State Machine** corresponds to the state in which the rule is declared. For instance, Figure 88 displays the sub-rule related to the state StartOfMission.A40.

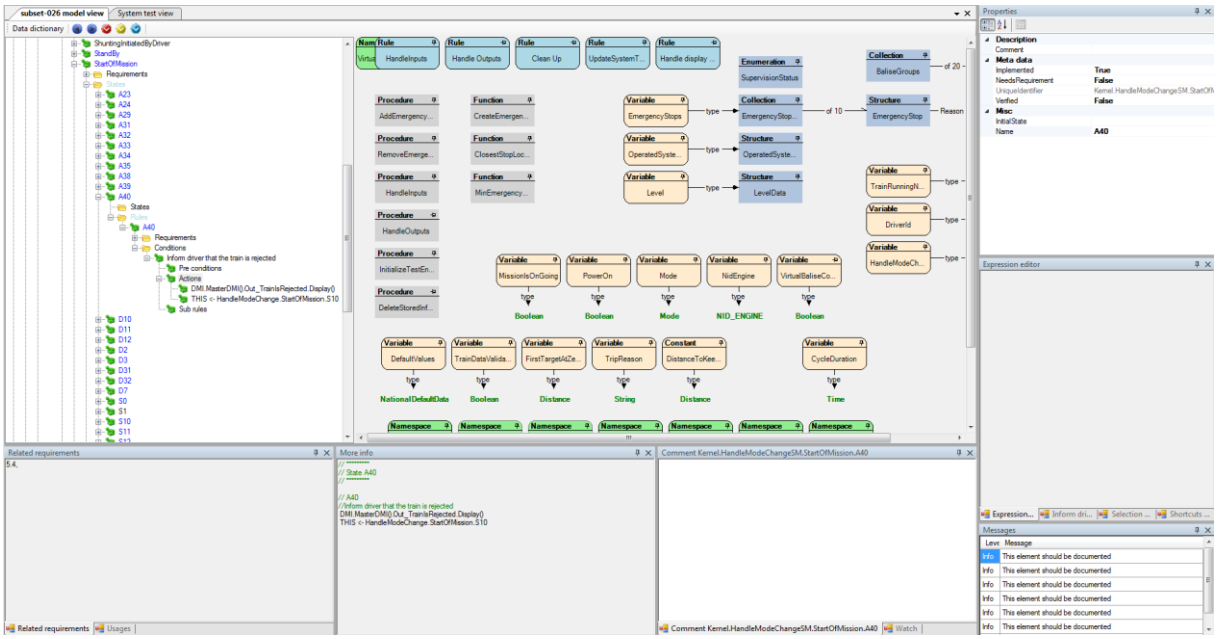


Figure 87: General view of a rule on a state of a State Machine, Start of Mission state A40.

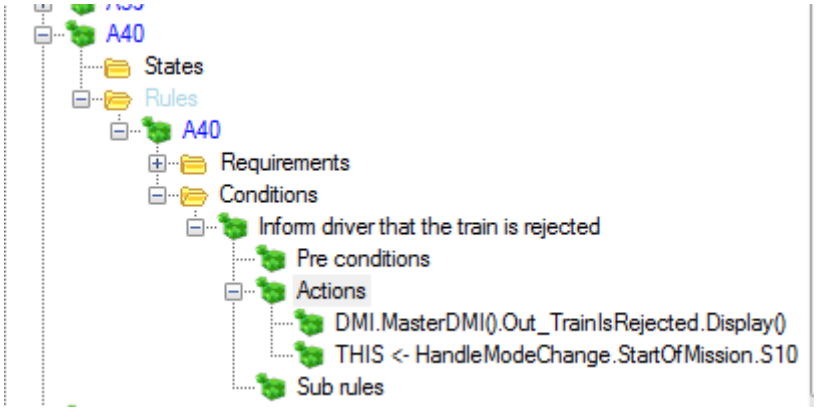


Figure 88: Detailed view of a rule on a state of a State Machine, Start of Mission state A40

The rule A40 is triggered only when its pre-condition is satisfied (always true) and the current state of the state machine is A40 or a sub-state of A40). When that rule is activated, it executes two following actions:

- The **State Machine HandleModeChange** changes its state to the state S10.
- The procedure **DisplayTrainIsRejected** of the master DMI is called and informs the driver that the train is rejected.

Please note that this also corresponds to a transition in a state diagram, as it is represented as such in the corresponding state diagram.

9.4 Shortcuts view

EFSW provides a way to quickly access some model elements, without searching them on the model hierarchical tree. The shortcuts are opened after opening a data dictionary and they are located on the middle left side of the EFSW main window on the shortcuts tab; Figure 89 depicts the shortcut

view window. In case of closing it, can be re-opened. On the menus, select: View/Show tools/Show shortcuts view or using “Crtl+E”.

To add an element of the model to the shortcuts use the **Drag&Drop** feature of the EFSW; they can then be assembled in different folders of the shortcut view. The shortcuts can be created, deleted and renamed using the contextual menu actions. The contextual menu appears right-clicking on the desired element.

To select in the model view the model element corresponding to a shortcut, just double-click on it.

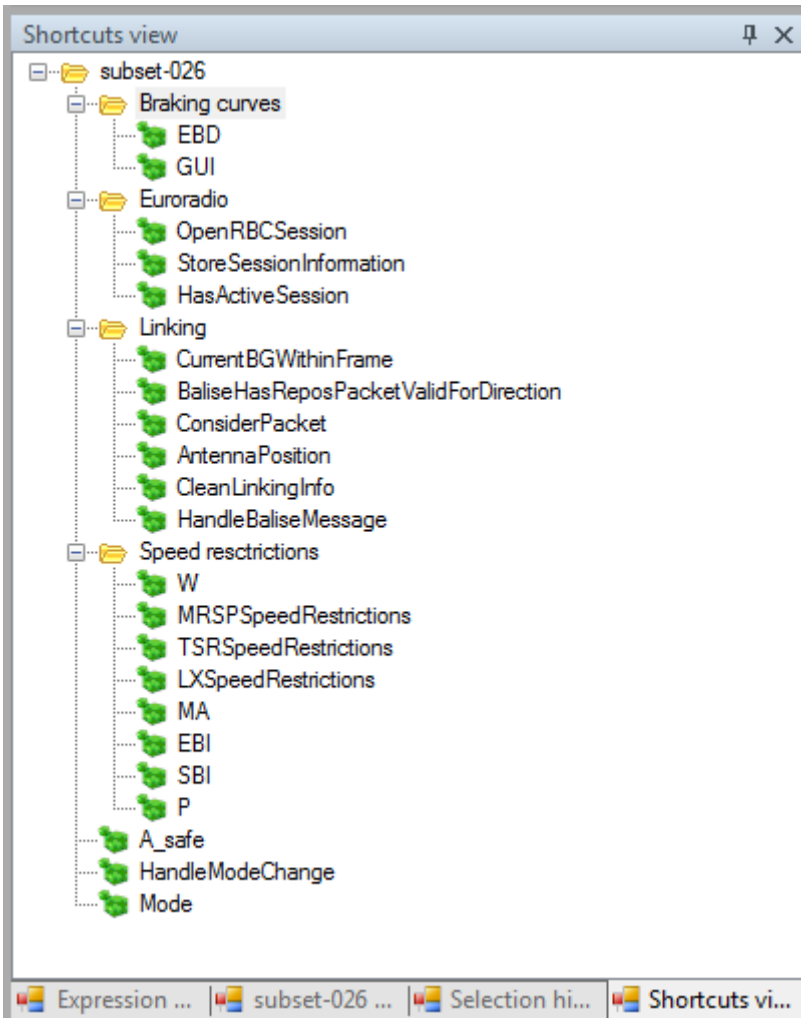


Figure 89: Shortcuts view

9.5 Tools related to the data dictionary view

Figure 90 shows all the tools related with the EFSM which are described on the following sub-sections.

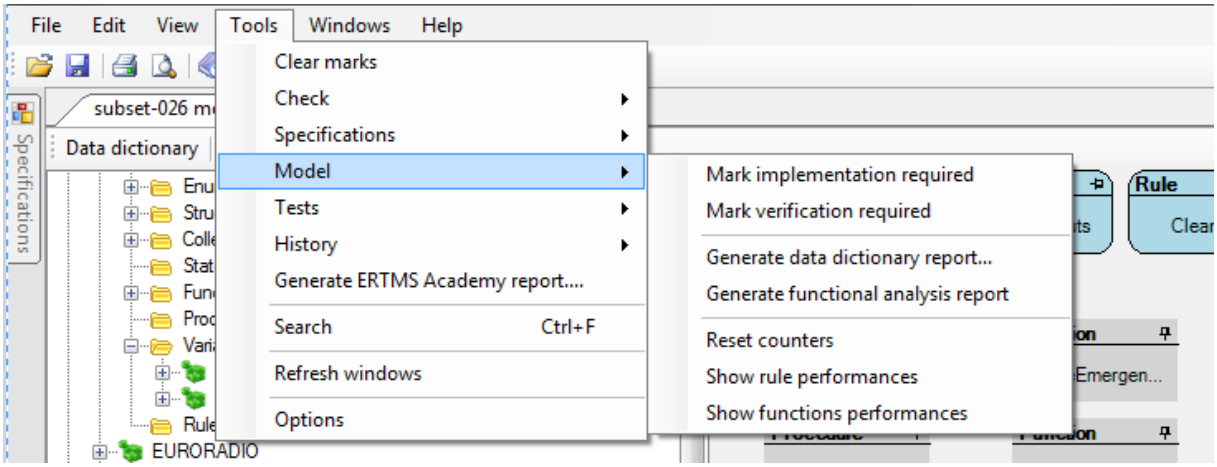


Figure 90: tools related with the ERTMSFormalSpecs Model

9.5.1 Search for elements requiring an implementation

The Mark implementation required tool allows to display all the model elements that have not yet been marked as implemented. Section 7.5.2 describes the colours legend associated for the messages. In the case of the Mark implementation required the kind of generated messages is Info, which means the path to the marked element is displayed in light blue, whereas the element which is not marked as implemented is marked in blue, as Figure 91 shows.

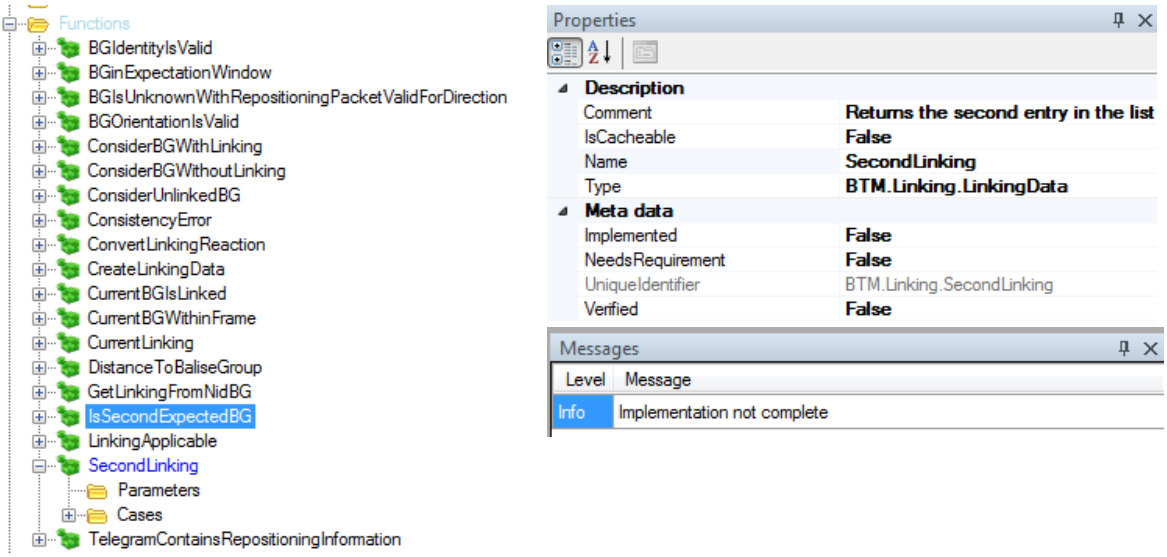


Figure 91: representation of elements requiring implementation.

9.5.2 Search for elements requiring a verification

The Mark verification required tool allows to display all the model elements which implementation has not yet been verified. Elements are marked following the same rule as the one presented on section 9.5.1 and following the described colour legend of section 7.5.2. See Figure 92.

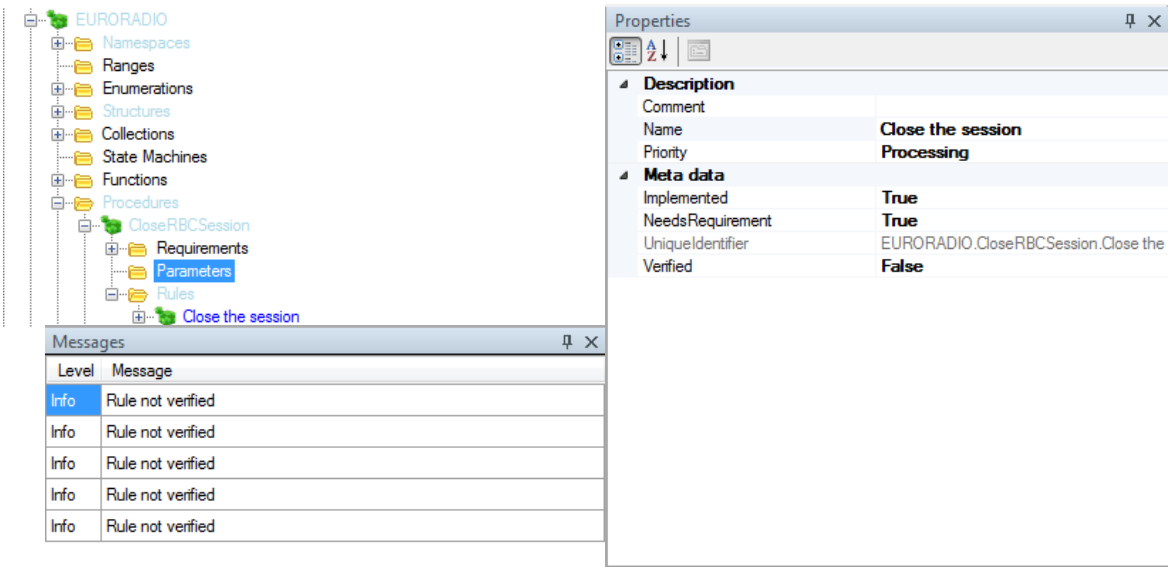


Figure 92: representation of the elements requiring verification.

9.5.3 Show rule performance

The performance of the rules on the EFSW can be measured taking into account the following traces:

- *ExecutionTime*: indicates the total time needed to execute a rule.
- *ExecutionCount*: amount of times a rule has been executed.
- *Average*: the relationship between the ExecutionTime and the ExecutionCount.

RuleName	ExecutionTime	ExecutionCount	Average
Kernel.Handle Outputs	264	4	66
Kernel.HandleOutputs.Send JRU messages	233	4	58
JRU.JRU.HandleMessagesOut.SendMessage	202	4	50
Kernel.InitializeTestEnvironment.Train data	94	1	94
Kernel.TrainData.Initialize.TrainData	94	1	94
Kernel.TrainData.InitializeTrainData.Train data	94	14	6
JRU.JRU.InitializeMessage.Send DMI Symbol Status message	62	4	15
JRU.JRU.HandleMessagesOut.UpdateVariable	31	84	0
DMI.InputInformation.CleanUp	16	320	0
EURORADIO.PositionReport.Handle second RBC/RBC handover position report	16	0	0
JRU.JRU.InitializeMessage.Send Magnetic Shoe Brake Status message	16	1	16
DMI.OUT.MRSP.Output when mandatory	16	0	0
DMI.DMIStruct.UpdateOUTVariables.Update local time	15	4	3
Kernel.HandleOutputs.Update the DMI	15	4	3
JRU.JRU.InitializeMessage.Send Additional Brake Status message	15	1	15
Kernel.InitializeTestEnvironment.Train position	15	5	3
JRU.JRU.InitializeMessage.Send Additional Data message	15	4	3
DMI.OUT.RBCContactInformation.Display.Updates the request status	0	0	0
DMI.OUT.TrainRunningNumber.TrackMode.Updates the status of the request according to the mode	0	0	0
DMI.IN.ErtmsEtcsLevelEntryRequest.InitiateRequest.InitiateRequest	0	0	0
DMI.IN.DriverIdEntryRequest.InitiateRequest.InitiateRequest	0	1	0

Figure 93: representation of the different rules performance

9.5.4 Show functions performance

The performance of the functions on the EFSW can be measured taking into account the following traces:

- *ExecutionTime*: indicates the total time needed to execute a rule.
- *ExecutionCount*: amount of times a rule has been executed.
- *Average*: the relationship between the ExecutionTime and the ExecutionCount.

FunctionName	ExecutionTime	ExecutionCount	Average
JRU.CreateHeader	62	34	1
JRU.CreateDMISymbolStatusChangeMessage	62	9	6
Kernel.DateAndTime.Now	61	242	0
JRU.DriverActed	32	4	8
JRU.CreateAdditionalDataMessage	31	8	3
JRU.CreateMagneticShoeBrakeStatusJruMessage	16	1	16
JRU.LevelChangedToNtc	16	5	3
DMI.OUT.MRSPMandatory	16	8	2
JRU.SendAdditionalDataMessage	16	4	4
JRU.SendNewMessage	16	4	4
JRU.DMISymbolsConvertors.LE04Activated	16	9	1
EURORADIO.OrderToTerminateHandingOverRBCIsReceived	16	4	4
JRU.LevelChanged	16	13	1
DMI.MasterDMI	16	1225	0
JRU.DMISymbolsConvertors.LE11Activated	15	9	1
JRU.CreateAditionalBrakeStatusMessage	15	1	15
JRU.SendTrainDataMessage	15	4	3
JRU.TrainDataHaveChanged	15	6	2
Kernel.DateAndTime.LocalTime	15	4	3
JRU.SendDMIStatusMessage	15	5	3
Kernel.AcceptInformation.ModeRules.AcceptSRDistanceInformationFromLoop	0	0	0
Kernel.AcceptInformation.ModeRules.AcceptDefaultGradientForTSR	0	0	0

Figure 94: representation of function performance

9.5.5 Reset counters

The Reset counters sets all the counters related with the rules performance and functions performance. See Show rule performance (section 9.5.3) and Show functions performance (section 9.5.4) for further details.

9.6 Model validations

9.6.1 Check for dead model

The model validation engine can detect functions or procedures which are never used by the model. They are called dead functions or dead procedures. Detecting such procedures and functions can be performed using the contextual menu Tools/Check/Check for dead model or by using "*Ctrl+D*".

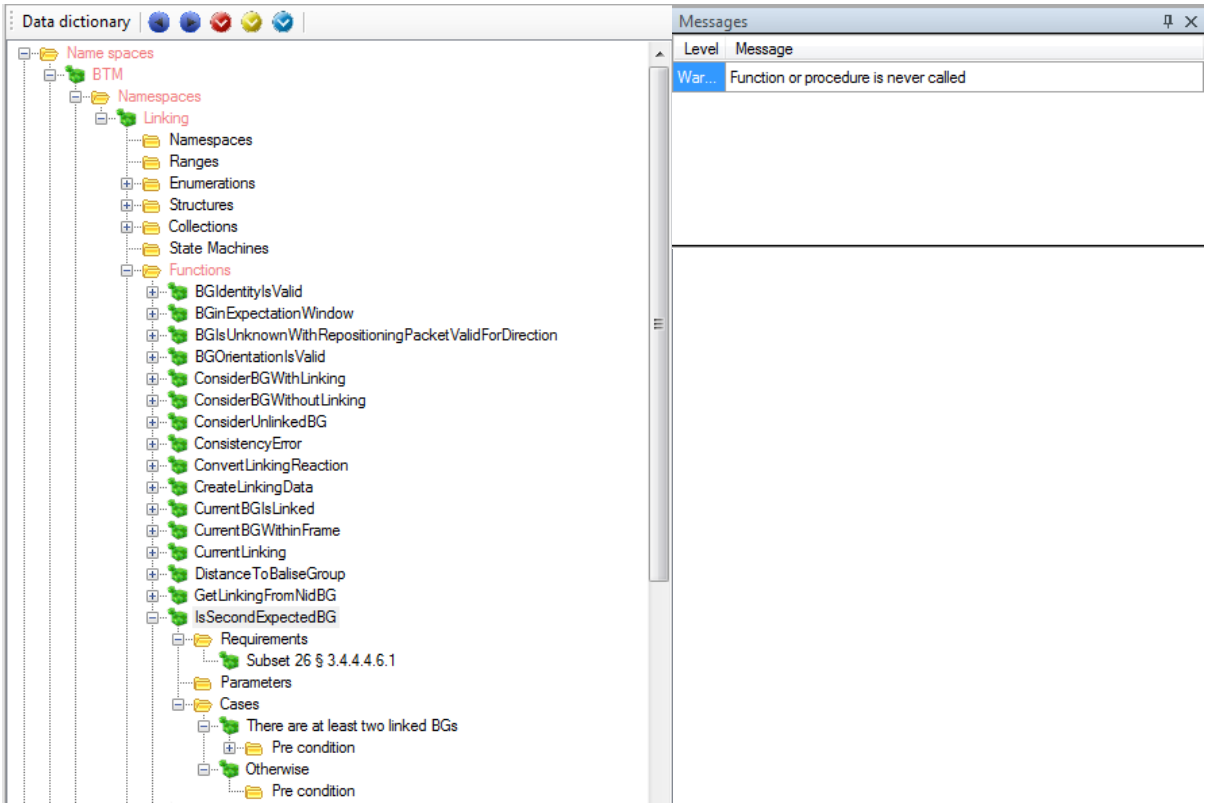


Figure 95: validation done on dead functions or procedures

9.6.2 Check model

ERTMSFormalSpecs allows to perform static tests on the model. During this process, the tool performs syntactical and semantical verifications on the model. The model validation engine can be activated selecting the contextual menu Tools/Check/Check model or using "*Ctrl+R*". The next table describes in detail the verifications performed by the EFSW:

Model	Description	Severity
Action	The variable which is modified by the action must exist in the system	Error
Action	The variable type must match the expression type	Error
All model elements	If the implementation flag of the model element is set to implemented, all its children must have the same value on this marker.	Warning
Assignation	When assigning a value to a variable of type range, the value must match the range's minimum and maximum values	Error
Case	The expression type of the type must be the same as the function	Error
Collections	All the collections must define a maximal size.	Error
Enumeration	Enumeration values must be unique	Error
Enumeration	The same numerical value cannot be assigned to two different elements of an enumeration.	Error
Enumeration	The enumeration identifier must be valid	Error
Expectation	The variable which is checked by the expectation must exist in the system	Error
Expectation	The variable which is checked by an expectation must match the expression type. The resulting value must be a Boolean.	Error
Expression	IN operator should be used instead of == between expressions of type StateMachine. This is used to verify that expressions of the form `CurrentState == S1` is not used, since this does not take sub states of S1 into consideration and would result to false if `CurrentState` is `S1.subS1`	Warning
Expression	Expressions must be syntactically correct	Error
Expression	Expressions must be type correct	Error
Expression	Designators used in expressions must refer to a valid model element	Error
Expression	Designators used in expressions must have a valid type	Error
Expression	Collections cannot be compared with EMPTY, compare with [] instead.	Error

<table>
<tr><td>Field of a structure</td><td>The mode of a field of a structure must be at least more restrictive than the mode of its enclosing field according to the following table
<table>
<tr><th>Enclosing field</th><th colspan="5">Field</th></tr>
<tr><th></th><th>Constant</th><th>Incoming</th><th>In/Out</th><th>Internal</th><th>Outgoing</th></tr>
<tr><td>Constant</td><td>✓</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Incoming</td><td>✓</td><td>✓</td><td></td><td>✓</td><td></td></tr>
<tr><td>In/Out</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td></tr>
<tr><td>Internal</td><td>✓</td><td></td><td></td><td>✓</td><td></td></tr>
<tr><td>Outgoing</td><td>✓</td><td></td><td></td><td>✓</td><td>✓</td></tr>
</table></td><td>Warning</td></tr>
<tr><td>Frame</td><td>The Cycle time duration does not resolve to a type that is compatible with Time</td><td>Error</td></tr>
<tr><td>Function</td><td>This element should be documented.</td><td>Info</td></tr>
<tr><td>Function</td><td>All the functions must have a return type.</td><td>Error</td></tr>
<tr><td>Paragraphs</td><td>Paragraph state does not correspond to implementation status. This can occur in several cases
The paragraph has been modelled but is not applicable
The paragraph model state is N/A or is Not Implementable but the paragraph is applicable.</td><td>Warning</td></tr>
<tr><td>Paragraph</td><td>All paragraphs must be linked to, at least, one Scope.</td><td>Warning</td></tr>
<tr><td>Paragraphs</td><td>Sub-paragraphs must have the same Scope as all their children.</td><td>Warning</td></tr>
<tr><td>Paragraphs</td><td>Two paragraphs cannot have the same identifier</td><td>Error</td></tr>
<tr><td>PreCondition</td><td>Operator == should not be used for state comparison</td><td>Warning</td></tr>
<tr><td>PreCondition</td><td>Operator != should not be used for state comparison</td><td>Warning</td></tr>
<tr><td>PreCondition</td><td>The variable on which the pre-condition is computed must exist in the system</td><td>Error</td></tr>
<tr><td>PreCondition</td><td>The variable type must match the Operand type, according to the operator semantics. The resulting value must be a Boolean.</td><td>Error</td></tr>
<tr><td>Procedure</td><td>This element should be documented.</td><td>Info</td></tr>
<tr><td>Range</td><td>Special values value cannot be duplicated</td><td>Error</td></tr>
<tr><td>Ranges</td><td>Special values names cannot be duplicated</td><td>Error</td></tr>
<tr><td>Ranges</td><td>Default value's precision does not correspond to the type's precision.</td><td>Error</td></tr>
<tr><td>Requirement link</td><td>The link to a requirement must refer to a valid requirement</td><td>Error</td></tr>
</table>

Requirement related	A model element related to a requirement (type, variable, procedure, function, rule) should refer to at least one requirement. This is only true if the flag **NeedsRequirement** is set to true. Either set the flag **NeedsRequirement** to false or provide a link to a requirement.	Info
Requirement related	When the implementation of a requirement is completed, all model element which implement that requirement must also be marked as "implementation completed".	Warning
Rule	This element should be documented.	Info
Rule	An incoming variable cannot be modified by the EFS.	Error
Rule	An outgoing variable cannot be read by the EFS.	Error
Rule	If one of the pre-condition is in the form *Variable == 'Request.Response'* there must be an action which sets that variable to *'Request.Disabled'*. This ensures that all requests for which a response was received are disabled.	Error
Special values	The precision of a Special value does not corresponds to the type's precision.	Error
State machine	This element should be documented	Info
State machine	The initial state of a state machine is not empty and corresponds to a state of that state machine	Error
State machine	The name of states in a state machine are valid (e.g. do not contain space)	Error
Statement	Assignation of EMPTY cannot be performed on variables of collection. Use [] instead.	Error
Structure elements	The type of the elements on a structure must be the same as the type of the default value.	Error
Sub-sequence	Sub sequences should hold at least one test case	Warning
Sub-sequence	First test case of a subsequence should hold at least one step. Only for the first step of the first test case on the sub-sequence	Warning
Sub-sequence	First step of the first test case of a subsequence should be used to setup the system, and should hold 'Setup' or 'Initialize' in its name. Sub-sequence. Only for the first step of the first test case on the sub-sequence	Warning
Subset-076 Steps	Cannot find Balise messages for this step. Only available for translation rules where a key word has been found.	Warning
Subset-076 Steps	Cannot find Euroloop messages for this step. Only available for translation rules where a key word has been found.	Warning
Subset-076 Steps	Cannot find RBC message for this step. Only available for translation rules where a key word has been found.	Warning
Translation	Translations must contain action and/orexpectations, or must be linked to a requirement.	Warning

Translation	Source text of the translations rules must be unique	Error
Type	This element should be documented.	Info
Type	Types are uniquely identified	Error
Type	A type should define a valid default value	Error
Typed Element	The declaration of a typed element (variable, structure element, function) defined in the model must have a valid type	Error
Typed Element	Recursive types are not allowed	Error
Variable	The variable semantics cannot be empty. Fill the field comment defined for the variable	Info
Variable	When its type has no comment an Info message is displayed also on the variable.	Info
Variable	The type of a variable must be the same as the type of the default value.	Error
Variable	The expression of the default value of a variable must be correct	Error

Table 3: checks verification on EFS

After the validation is performed, each node of the data dictionary view is displayed according to the following rule in the following colour:

- Red: an error has been found on the corresponding item.
- Orange: an error has been found in one of the sub elements.
- Brown: a warning (and no error) has been found on the item.
- Pink: a warning (and no error) has been found in one of the sub elements.
- Blue: an informative message (and no error nor warning) has been found on the item.
- Light blue: an informative message (and no error nor warning) has been found in one of the sub elements.
- Black: nothing has been found either on this item or one of its sub elements.

Figure 96 shows an example of the result of the Check model action. In this case, there is an error on one of the actions of the rule Update LRBG Values, and Info messages on several items of the tree view.

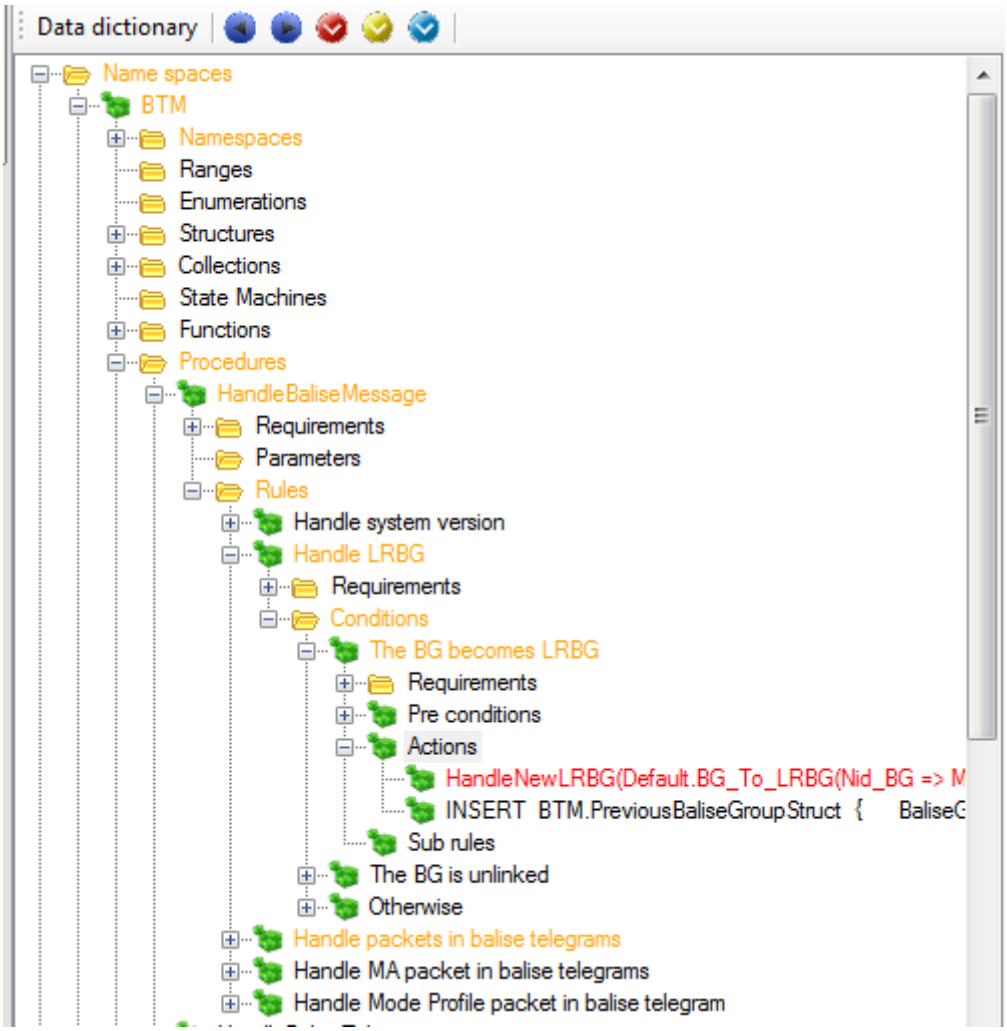


Figure 96: Example of model warnings/errors

9.7 Expressions

Expressions are used by the EFSW to evaluate values for pre-conditions, actions (these two cases are described on Section 9.3.2.5) and expectations (see Section 10.3). They follow the BNF grammar described in the EFSW Technical Guide (see [1]).

10 ERTMSFormalSpecs Test browser and execution environment

The EFSW provides an environment to define and run several kinds of tests:

- UNISIG Subset-076 test sequences
- Additional functional tests targeted at EFS model coverage
- User-defined tests

Tests are used to ensure that the model corresponds to the expected behavior. In some sense, tests provide an alternate model of the system. They set the system in a specific situation and check the system behavior. Tests which are defined on EFSW follows the structure presented on Subset-076: they are grouped in frames, each frame is split in one or more sub-sequence, each sub-sequence is divided in test cases and each test case is further split in steps. Last, and this does not correspond to Subset-076 structure, steps are split in sub-steps. This allows to seamlessly translate a Subset-076 step which requires several atomic actions to be performed.

10.1 Opening the test browser

The test browser allows to edit tests and execute them against the model. The test browser is automatically opened when launching the EFSW and in case it is closed by the user it can be re-opened using the View\Show tests view contextual menu as depicted by Figure 97.

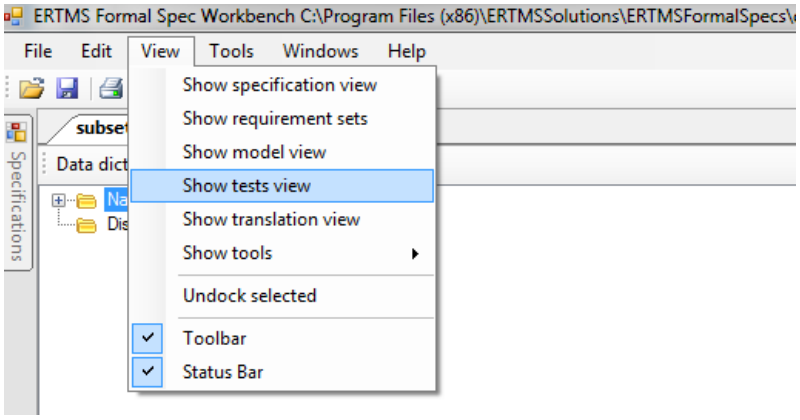


Figure 97: opening the test view

10.2 Overview

The test browser is composed by several parts. See Figure 98.

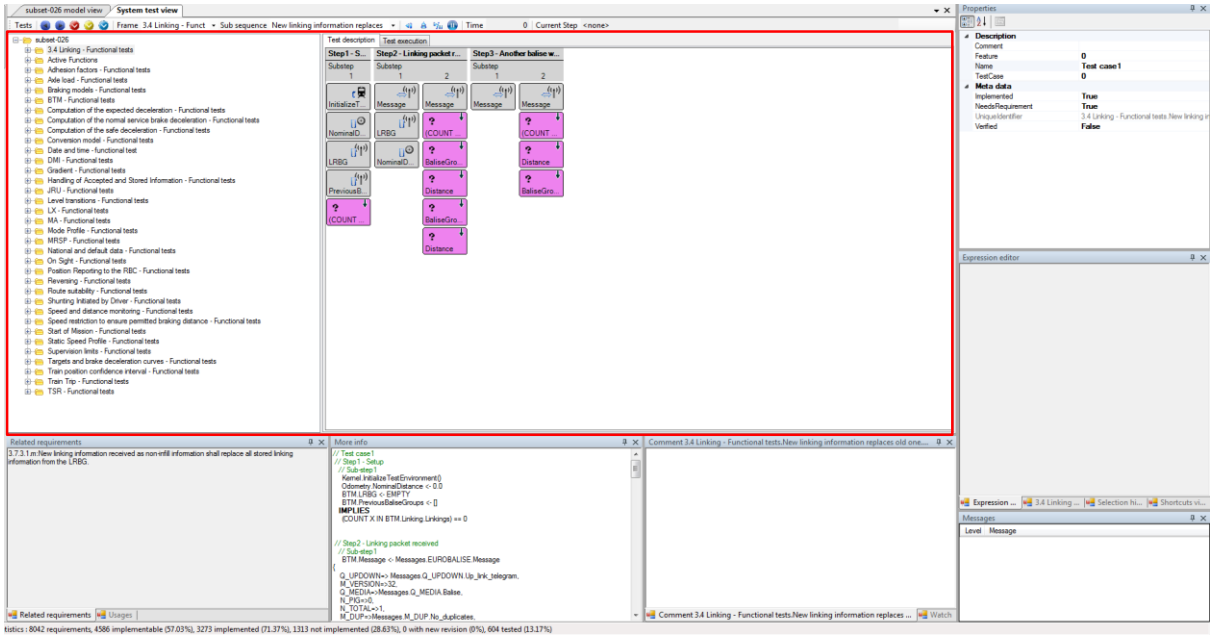


Figure 98: general overview of the system test view

The left tree view represents the tests structure and allows to select a test element. As stated before, EFS Tests are structured according to the structure of Subset-076 tests (frames, sub sequences, test cases, steps and sub-steps). This is shown in Figure 99.

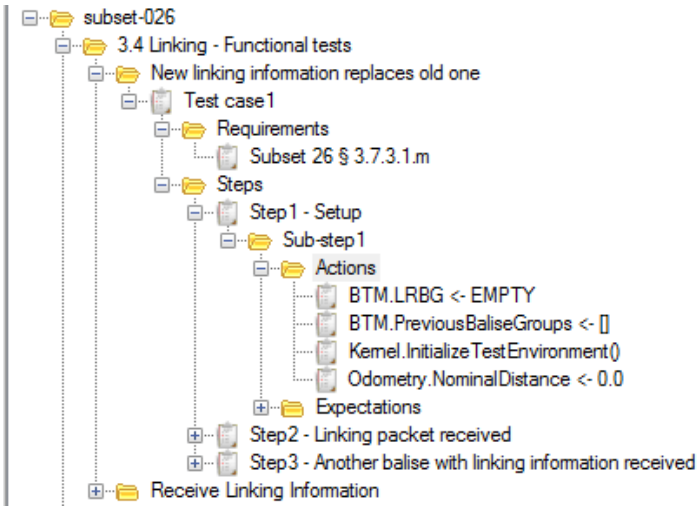


Figure 99: test tree view

The **property**, **expression editor**, **history**, **selection history view**, **shortcuts** and **description** windows have been described previously and keep the same functions and characteristics as described on section 9.2.

10.3 Test structure

EFSW graphically displays the tests structure using on the time lines. Figure 100 shows the two kinds of time lines: **test description** and **test execution**.

- **Test description**: graphical description of the test that should be executed in terms of actions and expectations.
- **Test execution**: shows the result of the execution of a test.

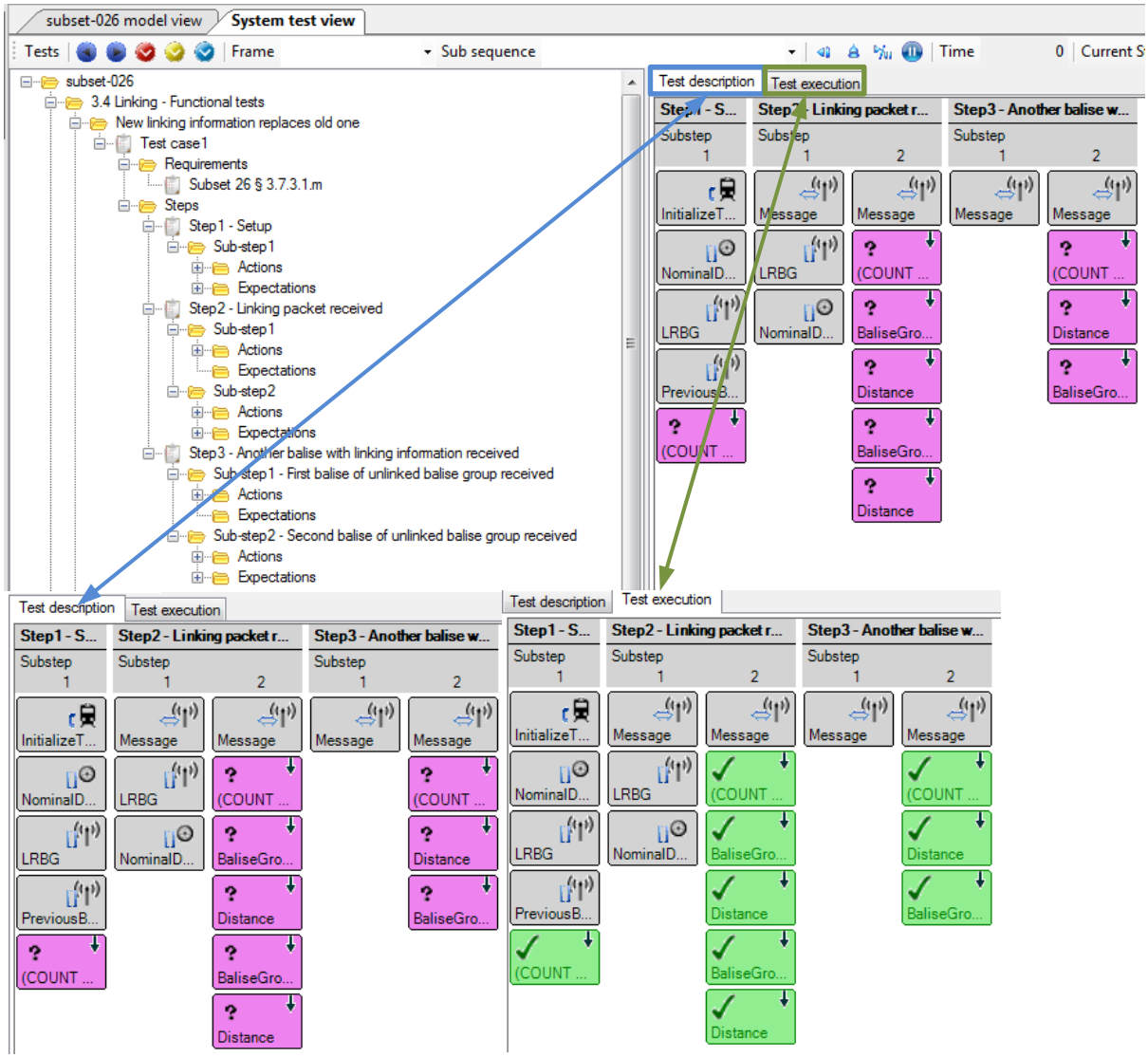


Figure 100: general overview of the EFST main window: test description and execution tabs

In both cases, time lines show the following elements:

- **Steps**: are used to group all the sub-steps involved. Usually, the first step of a test case is called *Step 1 – Initialize test environment,* and is used to initialize the system (typically using the procedure call *InitializeTestEnvironment()*). Steps are depicted by the topmost box in the time line.
- Each step contains one or more **sub-steps:** sub-steps are used to group the test actions and test expectations. Sub-steps are depicted by the grey boxes just below the step box.
- Test **actions**: represents a modification performed on the state of the different variables which conform the system. This corresponds to the test triggering. Actions are depicted by grey boxes with rounded corners, and are placed below the sub-step boxes.
- Test **expectations**: conditions that need to be satisfied by the system after the triggering has been applied, to consider the test successful. Those conditions can be either **punctual** or **last** a given duration in time. They can also **block** the scenario until they are satisfied, or simply need be satisfied once in a given timespan, **without blocking** the scenario execution. Expectations are depicted by a non-grey box with rounded corners, below the sub-step box. Its colour depends on the situation (see below).

For further details about the test elements and the test environment see section 10.4.

Figure 101 displays a typical test description. **Actions** are depicted as grey boxes with rounded corners, whereas **expectations**, in this case, are depicted as purple boxes with rounded corners and a question mark (?).

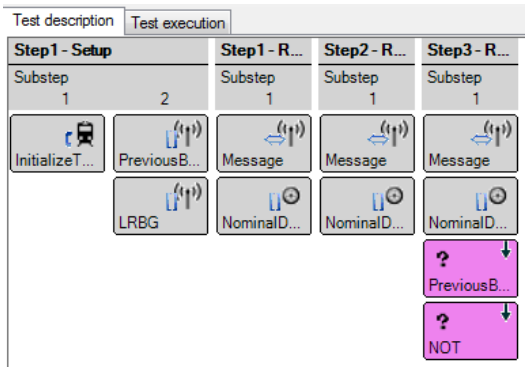


Figure 101: time line representation on EFST

On the other hand, the results of a test execution are depicted in Figure 102: **actions** are still depicted as grey boxes with rounded corners, but **successful expectations** are displayed as green boxes with rounded corners and a ✓ sign.

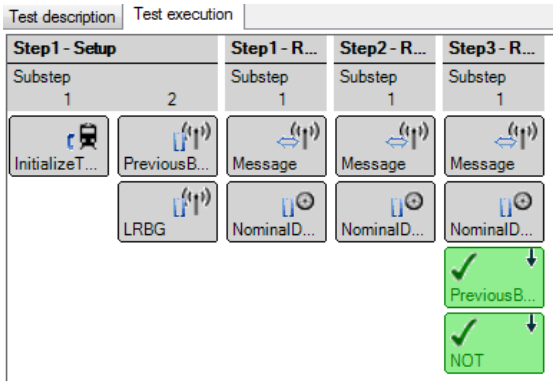


Figure 102: test execution view

In case an **expectation** has not been successfully satisfied, shall be displayed as a red box with a red cross (✖).

10.4 Test description

The test description tab presents a graphical display used to describe and edit a test. The following sub-sections describe the operations available on test descriptions. Same actions can be performed directly on the test hierarchical tree by right-clicking on the desired element and select the corresponding menu entry.

10.4.1 Add a test frame

Test frames are the highest component on the hierarchical tree after the root Test frame are used to group different sub-sequences, at least a test frame contains one sub-sequence.

To add a new test frame, select the subset where the test frame should be added, right click to display contextual menu. See Figure 103.

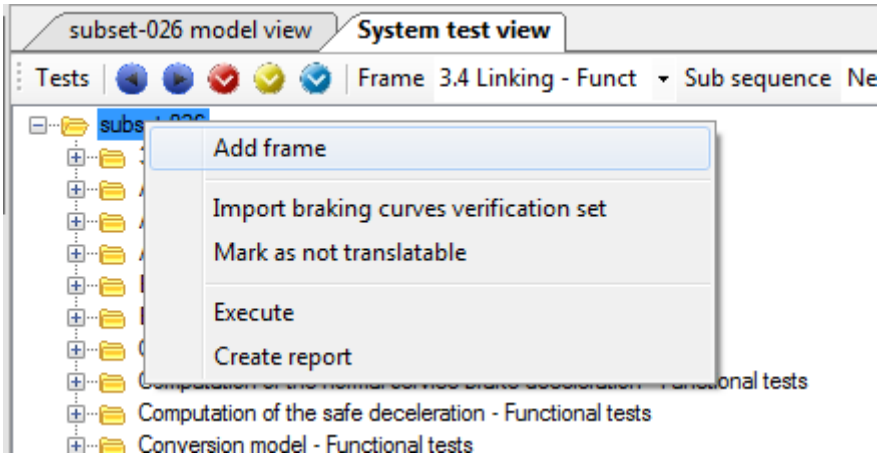


Figure 103: add a new test frame

Select **Add frame** to add the new test frame is the dictionary.

The specific properties related with a test frame are:

- **Cycle duration**: indicates the time required to compute a cycle on EFSW (see section 9.3.2.5).

10.4.2 Add a sub-sequence

Sub-sequences are used to sequence several test cases within a test. Sub-sequences are independent on to the other, hence, should begin with a complete system setup. Right-clicking on a test frame and selecting **Add sub-sequence** in the contextual menu adds a new sub-sequence to the selected test frame.

Specific properties for all the sub sequences are

- **Completed**: when set to true, it indicates that the current sub-sequence has been fully implemented. This flag is used by the continuous integration process to determine when a regression occurs: only **completed** sub sequences are checked by the continuous integration process.

Sub sequences can be either created manually or imported from Subset-076 (see section 10.6.3). When the sub-sequence is imported from a Subset-076 database, the Subset-076 Description part of the properties presents the information imported from Subset-076 test sequences.

- **D_LRBG**: indicates the distance to the last relevant balise group.
- **Level**: provides the related information to the current EVC level.
- **Mode**: provides the related information to the current EVC mode.
- **NID_LRBG**: indicates the identifier of the last relevant balise group
- **Q_DIRLRBG**: indicates the orientation of the train in relation to the direction of the LRBG
- **Q_DIRTRAIN**: provides the related information about the direction of train movement in relation to the LRBG orientation
- **Q_DLRBG**: provides the related information about on which side of the LRBG the estimated front end of the train is.
- **RBC_ID**: RBC unique identifier.
- **RBC_Phone**: telephone number of the RBC

10.4.3 Add new test case

The test cases consist of a sequence of steps to follow during the execution of the test. Right-clicking on a sub-sequence and selecting **Add test case** in the contextual menu adds a new test case to the selected sub-sequence.

Requirements can be linked to test cases by dragging that requirement and dropping it on the related test case. To remove a requirement from a test case, select the requirement and right-click on it, select **Delete** in the contextual menu.

Properties specific to a test case are

- **Feature**: linked to Subset-076.

- **Test case**: linked to Subset-076. The couple **feature** and **test case** identifies the test case on Subset-076.

10.4.4 Add new steps, sub-steps, actions and expectation

Steps, sub-steps, actions and expectations definitions can be found on the beginning of this section (section 10.3).To add a new step, right-click on the empty space of the test description tab. Figure 104, illustrates how to add a new step on the current test:

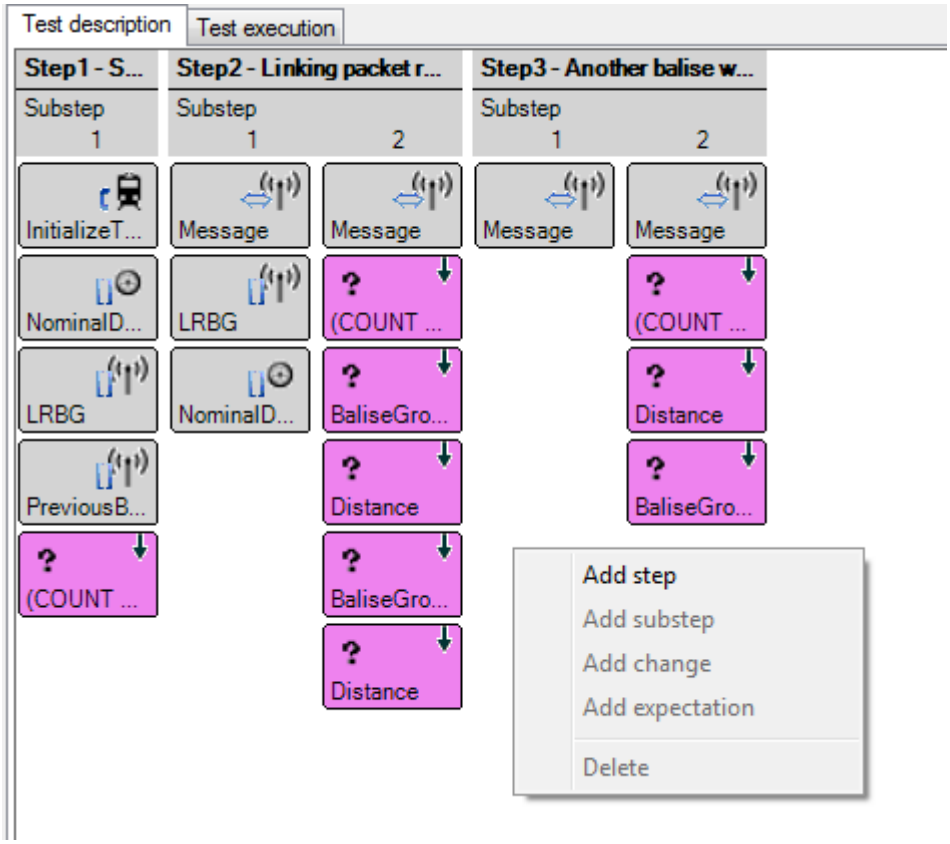


Figure 104: adding a new step using the contextual menu

Specific properties of steps are:

- **Completed**: when set to True, it indicates if the current step has been fully implemented.

Additionally, steps may also contain information related to Subset-076. These properties are displayed under the tag **Subset76**. They are:

- **Distance**: indicates the distance where the step occurs
- **Input Output**: indicates the mode of the variable.
- **Interface**: represents the interface to which must be applied.
- **Order**: the order of the sub-step on the test sub-sequence.
- **Test Level In**: indicates the EVC level at the beginning of the test step. For further information about the levels see Section 2.6 of [2].
- **Test Level Out**: indicates the EVC level after executing the selected step.
- **Test Mode In**: indicates the EVC mode at the beginning of the step. For further details about the Modes see Section 4.4 of [2].
- **Test Mode Out**: indicates the mode after the execution of the test step.
- **Translated**: indicates if the current step has been already translated using the tool described on Section 11.3.
- **Need Translation**: indicates if the selected sub-step needs to be translated as described in section 11.3.
- **User Comment**: this comment comes directly from Subset-076 database, and cannot be modified using EFS.

To add a new sub-step on an existing step, right-click on the step, and select the **add sub-step** in the contextual menu.

Properties specific to a sub-step are

- **Skip engine**: when set to true indicates that the engine of EFS must not be activated after having executed the actions related to this step. This flag is used to automatically translate steps in Subset-076 where, for instance, a new expectation is added in the test, but the system should not run.

To add an action or an expectation in a sub-step, select the sub-step then right-click to select the element to add in the contextual menu.

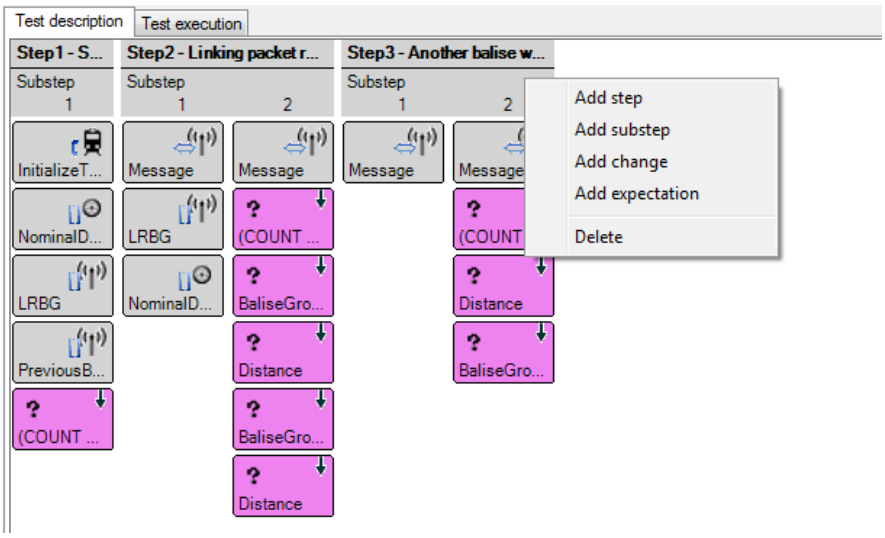


Figure 105: adding a new test element on an already existing item

Figure 106 shows the properties related with an action:

- **Expression**: the expression indicates the operations to be performed when the action becomes active. This expression can also be edited in the Expression window.

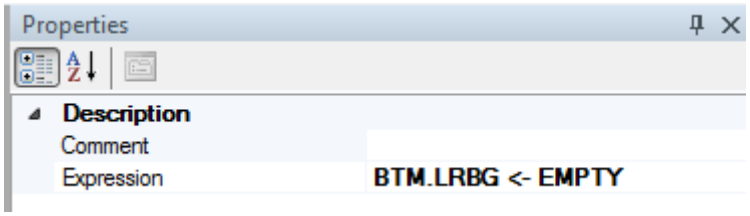


Figure 106: action properties

Expectations hold the following information:

- **Blocking**: when set to true, indicates that the following sub-step cannot be activated as long as this expectation is not reached.
- **Condition**: The condition in which the expectation should be verified. If the condition is not satisfied, the expectation is not taken into account during the verification phase.
- **Cycle phase**: the phase of the system's processing cycle when the expectation is verified, see section 9.3.2.5 for more information about processing cycles.
- **Expression:** indicates the element to be checked in order to determine if the expectation has been satisfied or has failed.
- **Kind:** indicates if the evaluation of the expectation is punctual or continuously checked:
- **Instantaneous:** the expectation should be satisfied at least once before the time specified, otherwise, the expectation is considered as **Failed**.
- **Continuous:** the expectation should always be satisfied during the duration specified, otherwise the expectation is considered as **Failed**.**DeadLine**: provides duration associated to the expectation. The semantics of this duration depends on the expectation kind (see above).

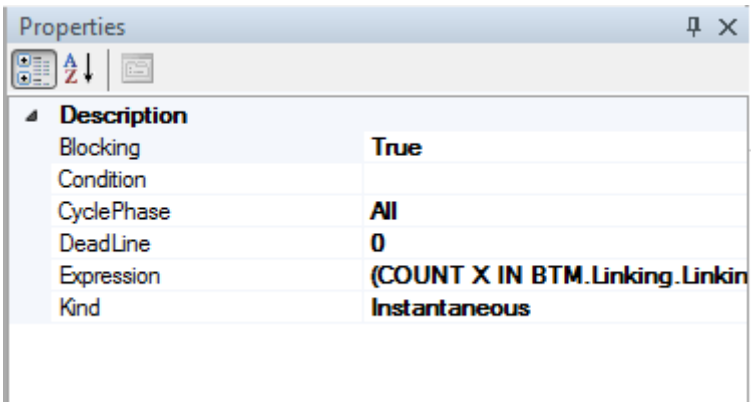


Figure 107: expectation properties

10.4.5 Remove steps, sub-steps, actions and expectation

One can also delete elements from the test using the contextual menu. Select the corresponding element, right click to open the contextual menu, and select **Delete**, as shown in Figure 108.

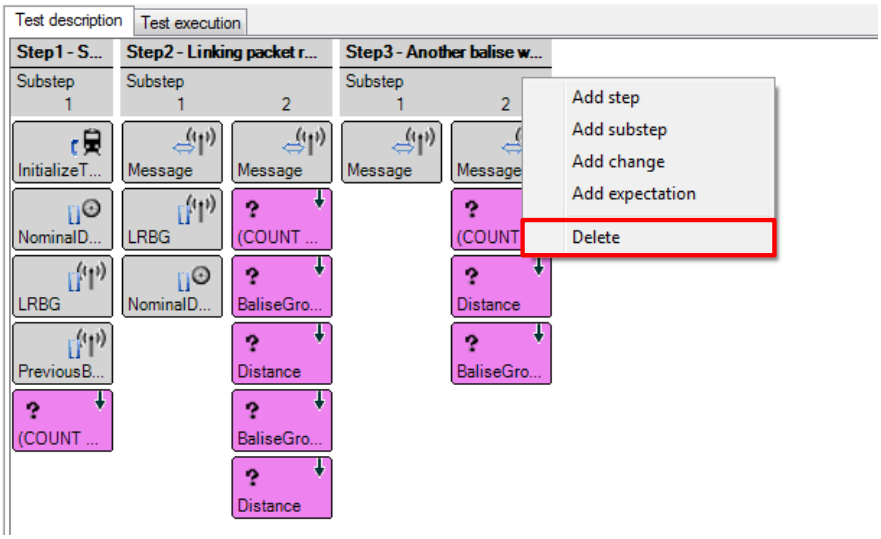


Figure 108: delete option of the contextual menu

10.4.6 Remove test frames or sub-sequences

Deleting test frames or sub-sequence cannot be performed on the description time line. This is performed using the hierarchical tree: right-clicking on the selected test frame or sub-sequence to remove and select **Delete** in the contextual menu. Note that you can also delete test cases, steps, sub-steps, actions or expectations using the hierarchical tree.

10.5 Test execution

As stated before, the test browser can also be used to execute tests to ensure that the model behaviour satisfied the expectations expressed in the test, according to the triggers (actions) the test defines. Test execution and results are also displayed using time lines.

To execute a sub-sequence step by step, first select the test frame, as shown in Figure 109.

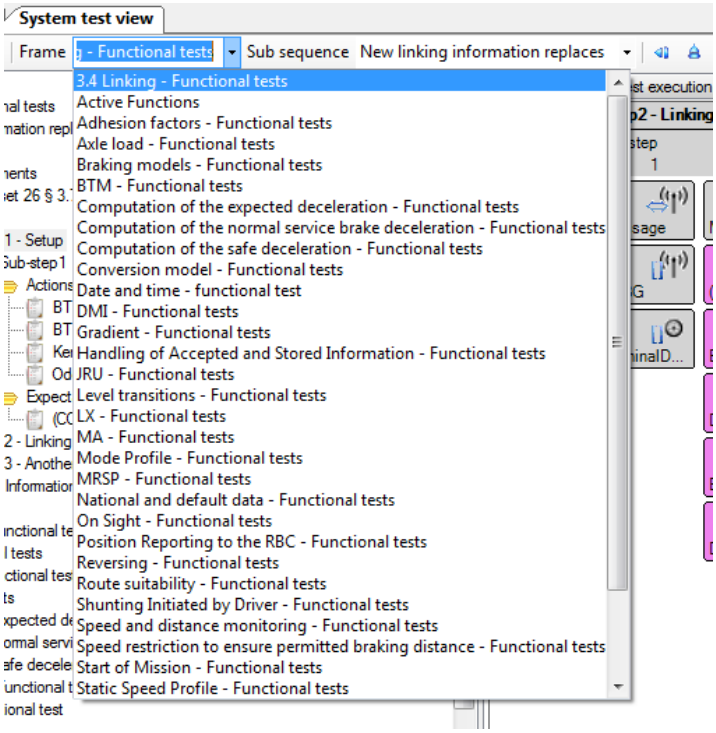


Figure 109: test frame selection

Once the frame has been selected, select the sub-sequence, as shown in Figure 110.

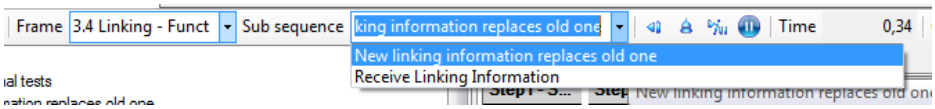


Figure 110: test frame selection

Finally, to animate the system use the animation buttons depicted on the Figure 111.

- **Step once ( )**: executes the next step of the current test. Each time this button is pushed a new step is displayed on the execution time line, and all windows are updated.
- **Step back ( )**: navigates one step back on the test execution and updates all windows.
- **Restart ( )**: empties the time line and restarts the current test.
- **Pause ( )**: used to suspend model execution when external visualizer interact with EFS, such as the ERTMS Solutions' DMI or the ERTMS Solutions' TrackMap display.

Figure 111: buttons controlling test actions

10.5.1 Watch window

During the execution of a test, the values of a model element can be seen on the watch window. See Figure 112.

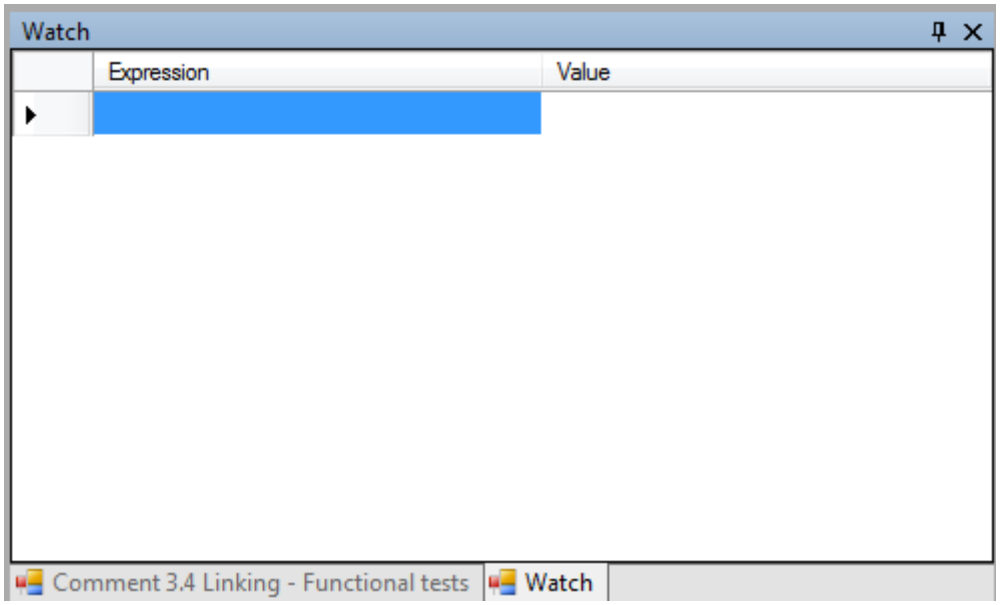


Figure 112: watching window

Double-clicking on the blue highlighted rectangle displays the expression editor. Edit the expression to select the component of the model to be seen and close the window.

One can also **Drag&Drop** a variable from the hierarchical tree displayed in the model view to add it in the watch window.

10.5.2 Test triggerings (actions) and expectations

Several colours are used to display results of a test execution, as shown in Figure 113.

- **Variable updates** due to actions defined in the test are displayed in grey. Left click navigates to the corresponding action in the test browser. Double click displays the explanation view, which describes all the computations performed by the engine to alter the variable's value..
- **Unfulfilled expectations** are displayed in violet. Left click navigates to the corresponding expectation in the test browser.
- **Fulfilled expectations** are displayed in green. As for all expectations, Left click navigates to the corresponding expectation in the test browser. Double click displays the explanation view, which describes all the computations performed by the engine to compute the expectation's value.
- **Failed expectations** are displayed in red. As for all expectations, Left click navigates to the corresponding expectation in the test browser. Double click displays the explanation view, which describes all the computations which lead to this expectation failure.

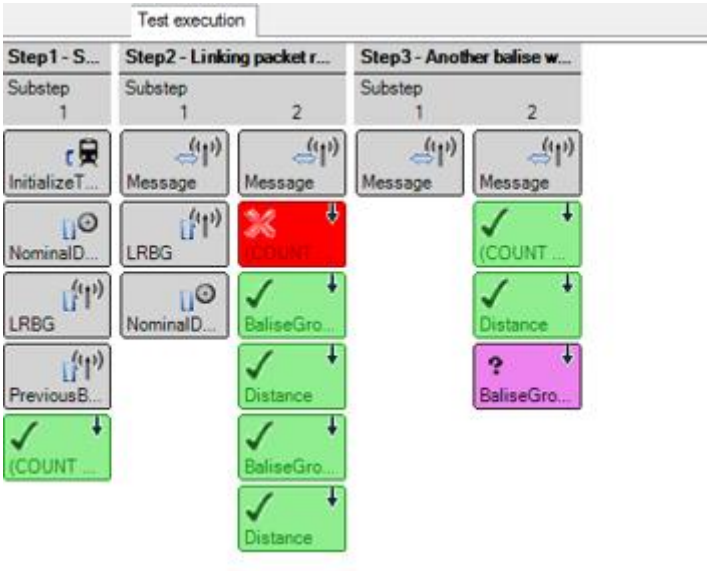


Figure 113: timeline component

10.5.3 Activated rules and variable updates

Additionally, the execution time line can also display the rules that have been activated, and the variable updates performed during the model execution. By default, to limit their number, this information is filtered out, but the filter can be configured thanks to Configure filter available in the timeline's contextual menu, as depicted below.

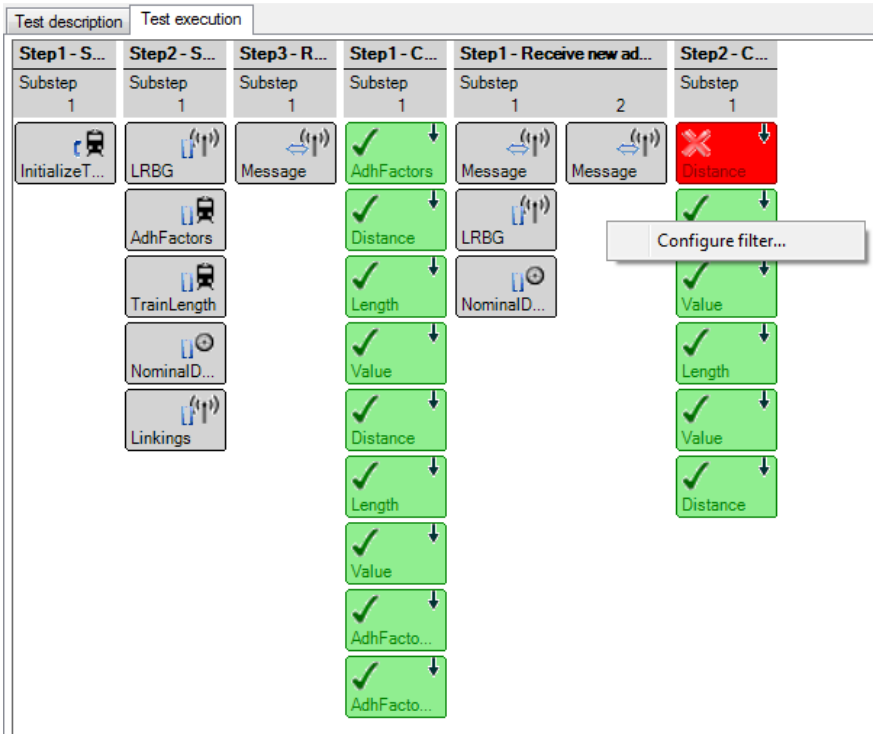


Figure 114: Contextual menu to configure the time line filter

This opens the filtering dialog, as shown in Figure 115 which provide the following options

- Show or hide **rule activations**: when selected, the execution time line displays the rules of the model which have been activated during the execution cycle (e.g. whose precondition have been evaluated to true). Activated rules are displayed as blue boxes with rounded corners.
- Show or hide **variable updates**: when selected, the execution time line displays the variables whose value changed during the execution cycle. They are represented as pink salmon squared boxes with rounded corners.
- Show or hide **expectations**: when selected, expectations are shown in the execution time line (see section 10.3).
- **Namespace** and **Variables** of the selected namespace filtering. Only the selected namespaces or variables are displayed in the execution time line.
- Regular expression filtering. When displaying elements in the execution time line, filters out all variable updates or rule activates which do not match the regular expression.

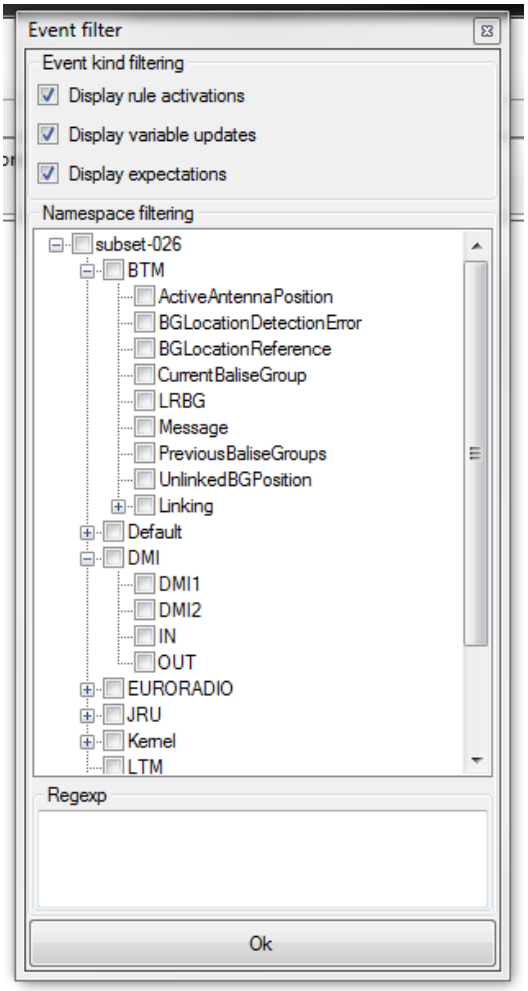


Figure 115: Filter options

Figure 116 shows a typical result after selecting the namespace EURORADIO in the filter view.

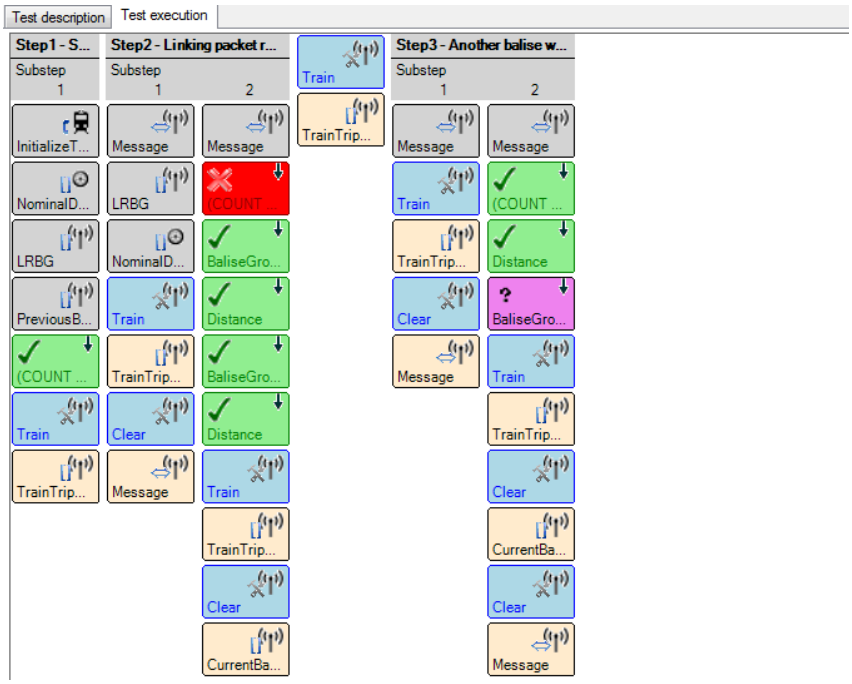


Figure 116: Rule activations and variable update display on a test

Several icons are used in Figure 116. Table 4 summarises the available icons and their associated meaning.

Variables and expressions	
	The affected variable is defined as IN
	The affected variable is defined as OUT
	The affected variable is defined as IN OUT
	The affected variable is defined as INTERNAL
	Procedure call
Namespaces	
	Related to namespace BTM or EURORADIO
	Related to namespace JRU
	Related to namespace Odometry
	Related to namespace DMI
	Related to namespace Kernel
Expectations	
	Continuous expectation
	Instantaneous expectation
	Failed expectation
	Successful expectation
	Expectation whose state has not yet been determined.
Rule activation	
	Rule activation marker

Table 4: icons used on the execution time line

10.5.4 Expectation failure

Figure 117 shows an example of expectation failed during the execution of a test. When an expectation is failed, the corresponding hierarchical tree in the test view is coloured using the scheme presented in 7.5.2 : a failed expectation corresponds to an error.

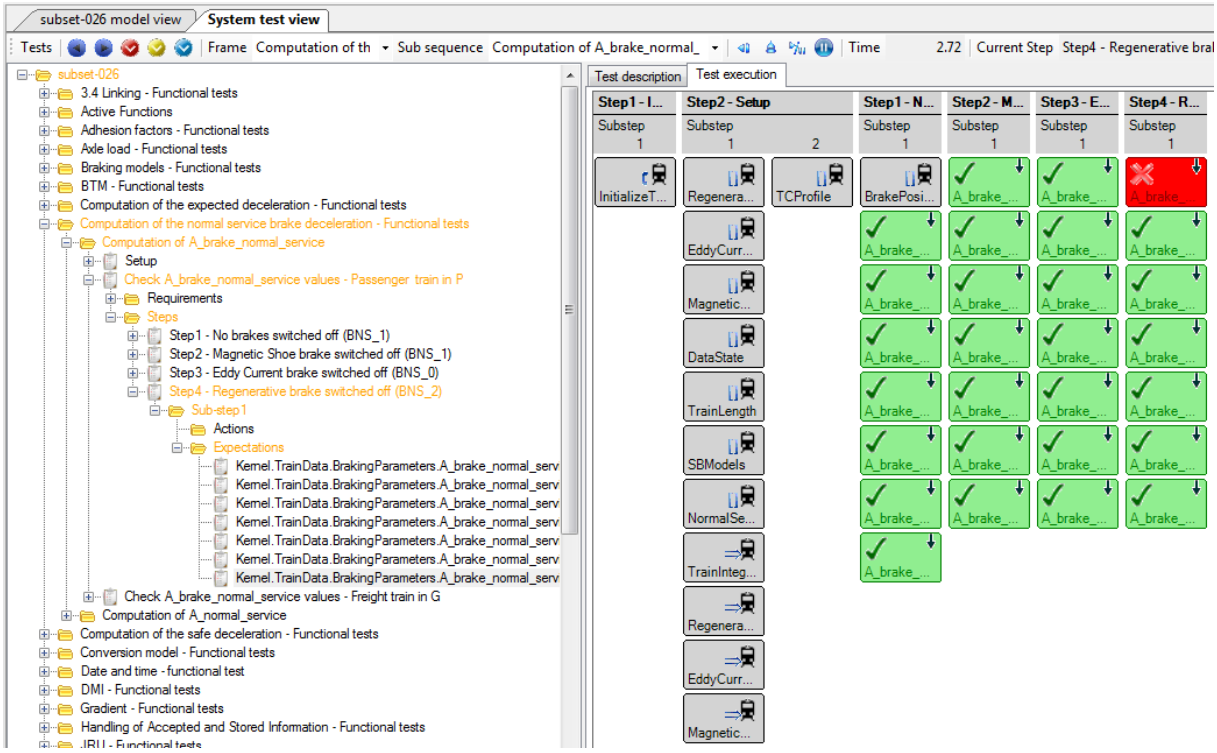


Figure 117: Failed expectation

The relevant message is displayed in the Messages window, as displayed in Figure 118.

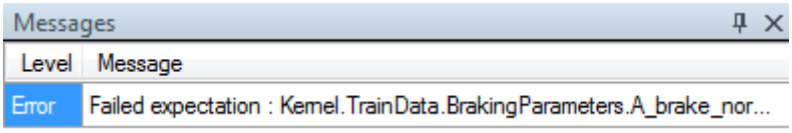


Figure 118: error message related with the failed expectation

10.5.5 Execute several steps at once

Tests execution on EFSW is not limited to the execution of a single step each time. EFSW allows to execute several steps at the same time.

10.5.5.1 Executing test steps checking its expectations are satisfied

Select the desired target step on the hierarchical tree and right-click on it. On the contextual menu select the Run until expectation reached entry. This executes all the steps in the sub-sequence until the expectations of the selected step are satisfied. Steps after the selected step are not executed. See Figure 119.

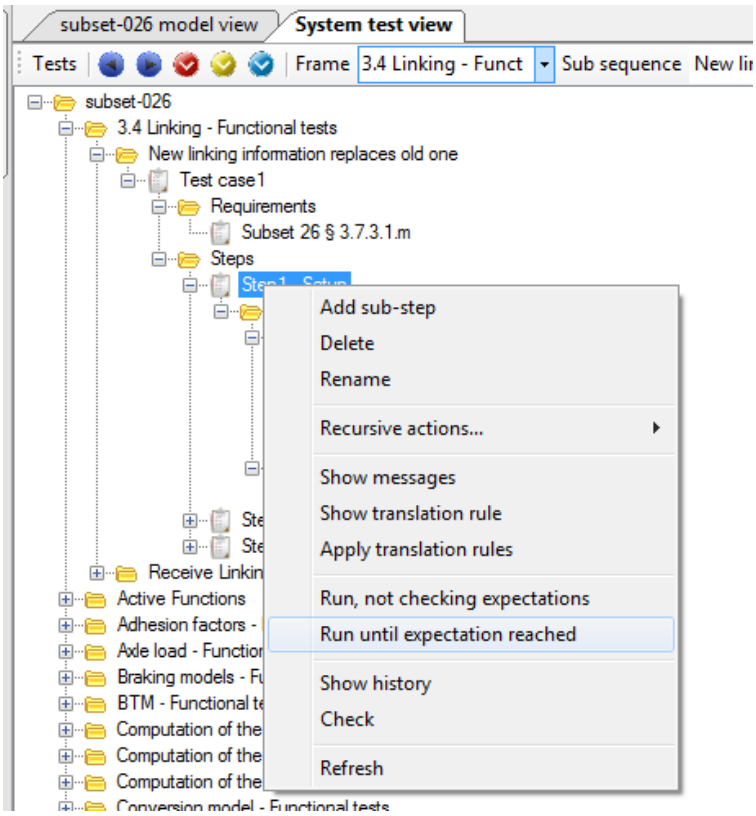


Figure 119: execute a step until expectation reached

This way of executing test steps is compatible with executing a test step by step described on section 10.5.

10.5.5.2 Execute steps without checking its expectations are satisfied

The entry “Run, not checking expectations” executes the test as described in 10.5.5.1, the execution stops before the step expectation are satisfied..

10.5.5.3 Executing a sub-sequence

To execute a complete sub-sequence, right-click on the selected test subsequence in the hierarchical tree view and select the **Execute** contextual menu entry, as displayed in Figure 120. This executes all steps of the sub-sequence, even if an expectation is failed during the execution. Results are displayed in the execution time line.

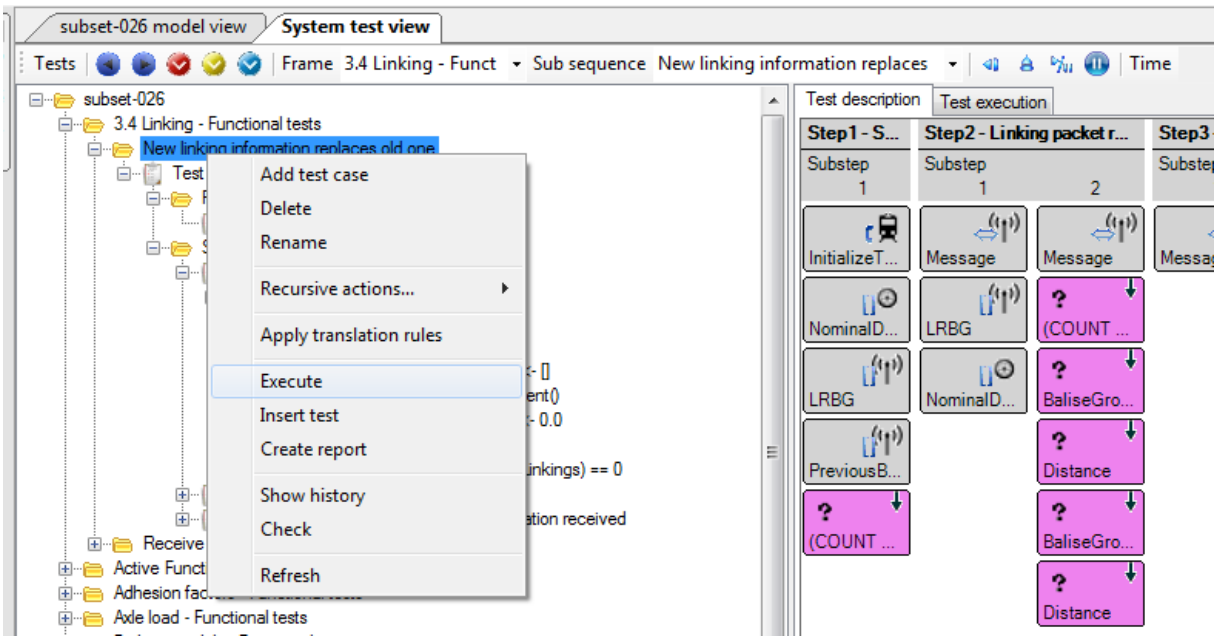


Figure 120: executing a sub-sequence by the contextual menu

10.5.5.4 Execute a test frame

Tests can also be executed at the frame level, using the corresponding **Execute** entry in the frame's contextual menu, as shown in Figure 121.

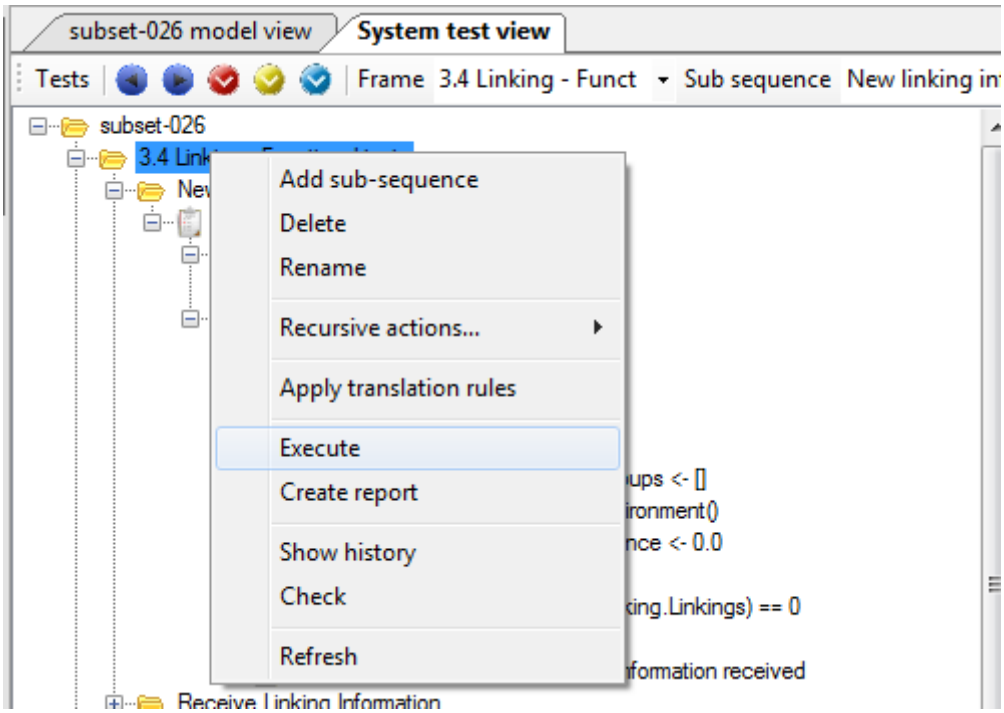


Figure 121: executing a test frame

Selecting this option executes all sub-sequences defined in the frame, and does not produce a graphical result on the execution time line, since a time line can only presents the result for a single sub-sequence. However, after executing completes, EFSW provides a summary of the test execution: how many sub-sequences were executed, how many of them were successfully and the amount of failures.

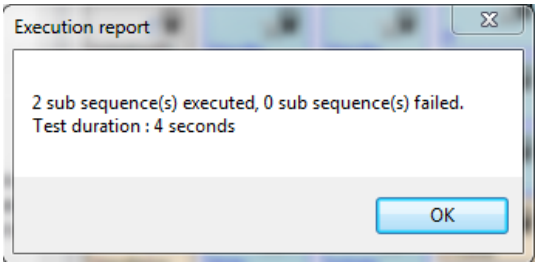


Figure 122: test frame execution result

10.5.5.5 Execute all the tests related to a dictionary

The EFSW allows to execute all the test frames present in a dictionary. Select the desired dictionary in the test window, right-click and select **Execute** in the contextual menu. See Figure 123.

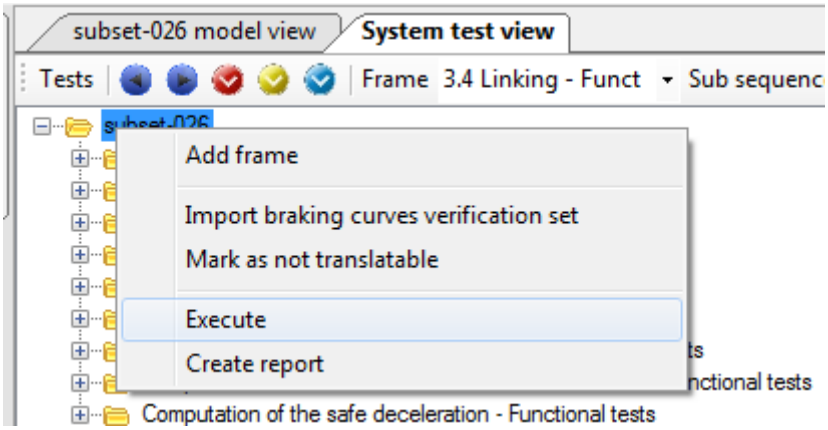


Figure 123: subset test sequences execution

The result of executing all the tests linked to a subset is the same as the result described on section 10.5.5.4.

10.6 Test tools

Figure 124 shows the actions which can be executed on the tests structure.

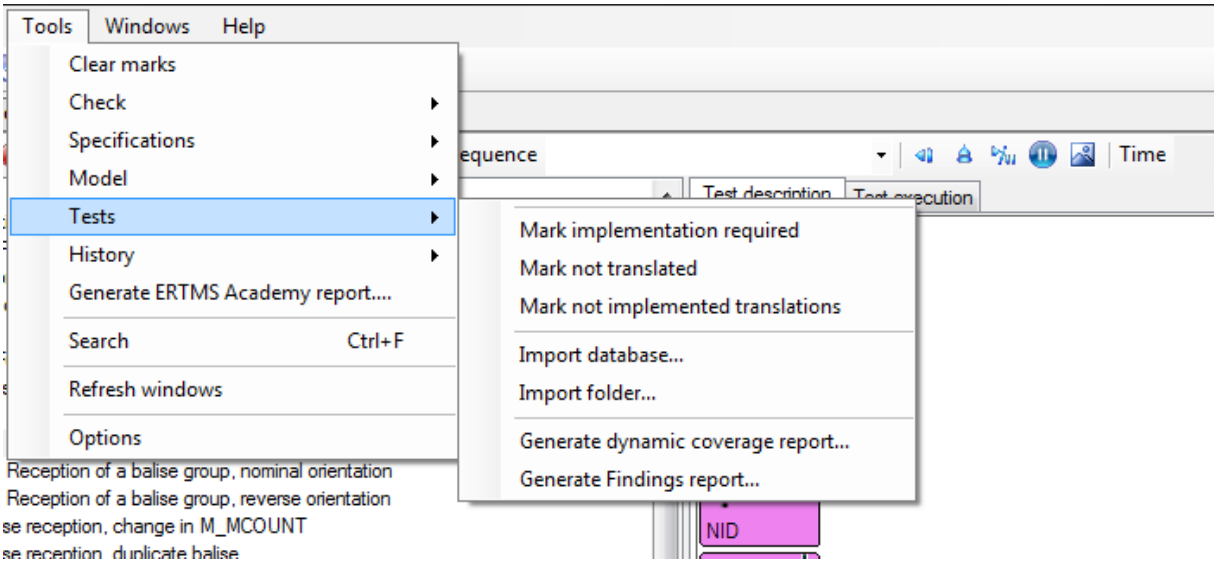


Figure 124: actions which can be performed on the tests.

10.6.1 Search for test elements requiring an implementation

The Mark implementation required tool allows to display all the test elements that have not yet been marked as implemented.

10.6.2 Search for test elements not translated

The Mark not translated tool allows to display all the test elements that have not yet been translated (see section 11 about translations).

10.6.3 Importing database/folder

10.6.3.1 Import data base:

Import database allows to import the ERA Subset-076 a data base file and to incorporate it to new test frame. The given name to the new test frame is by default subset-076.

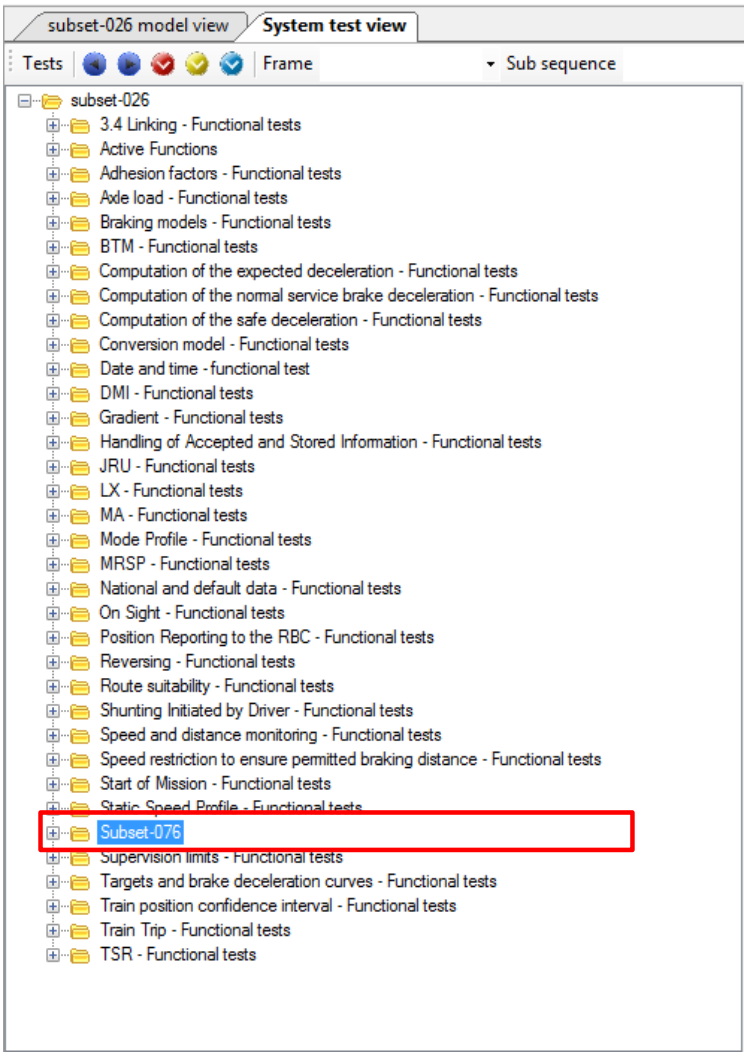


Figure 125: import database file.

10.6.3.2 Import folder

When more than one database file need be imported, the system allows to select the folder where these files are stored. The EFSW imports all the files at once and creates a unique test frame with as much sub-sequence as data base files available in the selected folder.

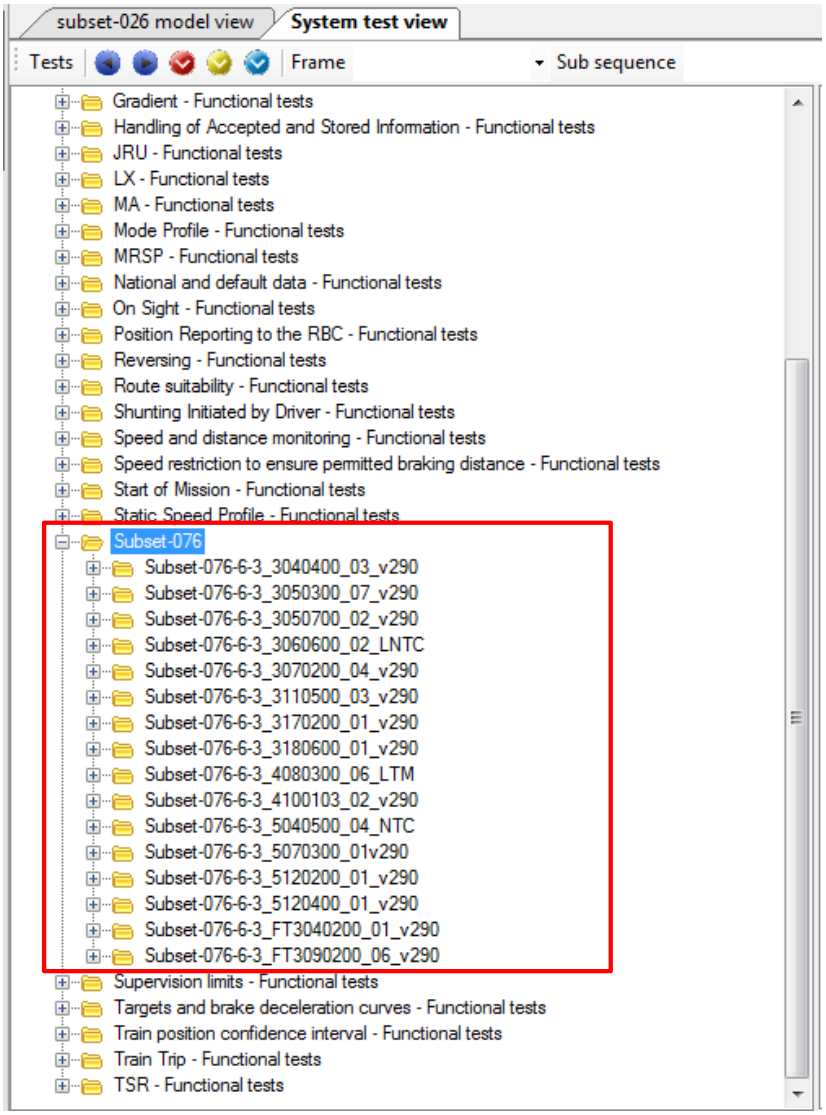


Figure 126: import folder result

11 Translations

EFS can import Subset-076 tests, as presented in section 10.6.3. However, these tests are expressed as a text and cannot be directly applied on the EFS model. EFW defines a way to automatically translate Subset-076 textual tests into a sequence of actions and expectations using a translation dictionary, and provides a tool to fill the translation dictionary. This is the scope of this section.

11.1 Open the translation view

The translation view can be opened using the tool menu entry View/Show translation view, as shown in Figure 127.

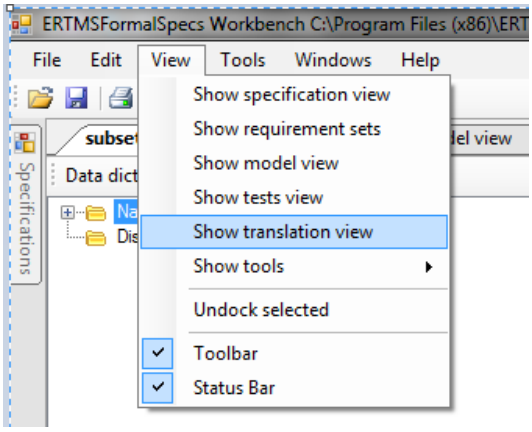


Figure 127: opening the translation rules view

As usual, if several EFS files are opened – which is usually the case when considering Subset-076 tests – a dialog box is provided to select the EFS file from which the translation view should be opened. See Figure 128.

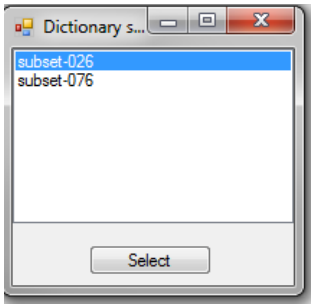


Figure 128: selection for the translation rules having opened two EFS files

In this case, select subset-076.

11.2 Overview

The translation browser allows to browse through the available translations. It proposes, as shown in Figure 129, hierarchical view of the translations, grouped using folders. Each translation contains two parts

- a set of source texts, which represent the pattern to be matched to apply the translation rule.
- one or more sub-steps, actions and expectations.

In addition, some translations are linked to requirements, for traceability purposes.

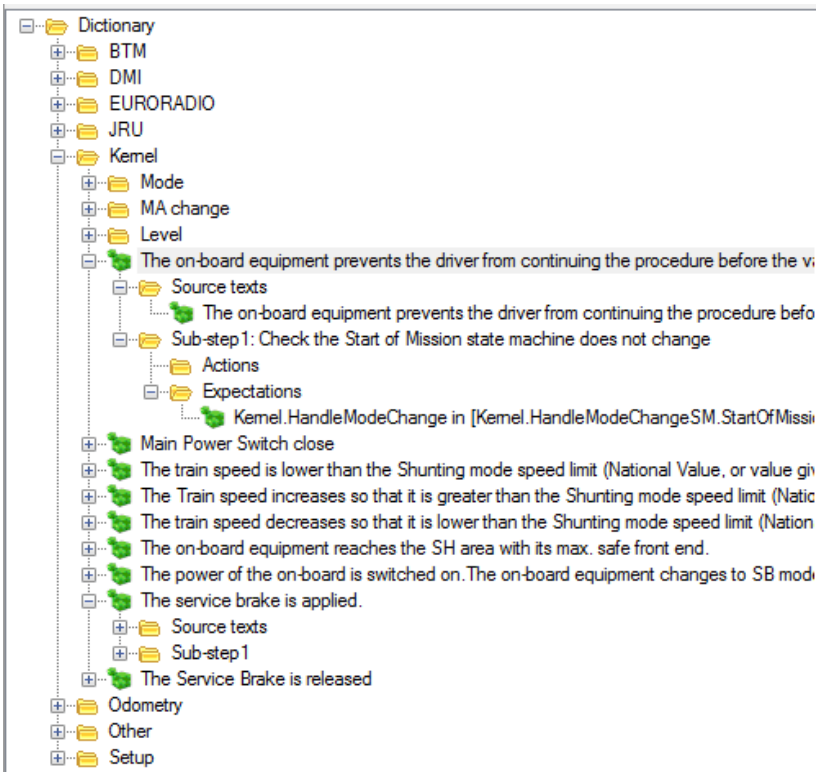


Figure 129: Translation view

Each translation can be edited as presented in section 10.4, using a description time line, as shown in Figure 130.

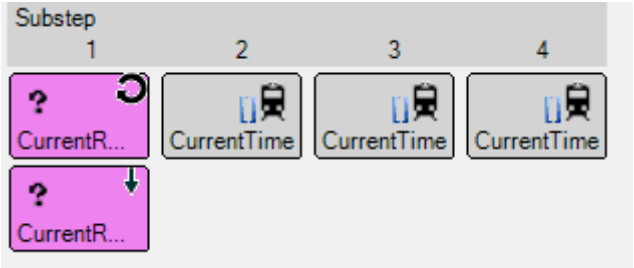


Figure 130: translation description representation

11.3 Translating

To apply translation rules, select the node which needs to be translated in the hierarchical tree view of the test browser, and select Apply translation rules in the contextual menu, as shown below.

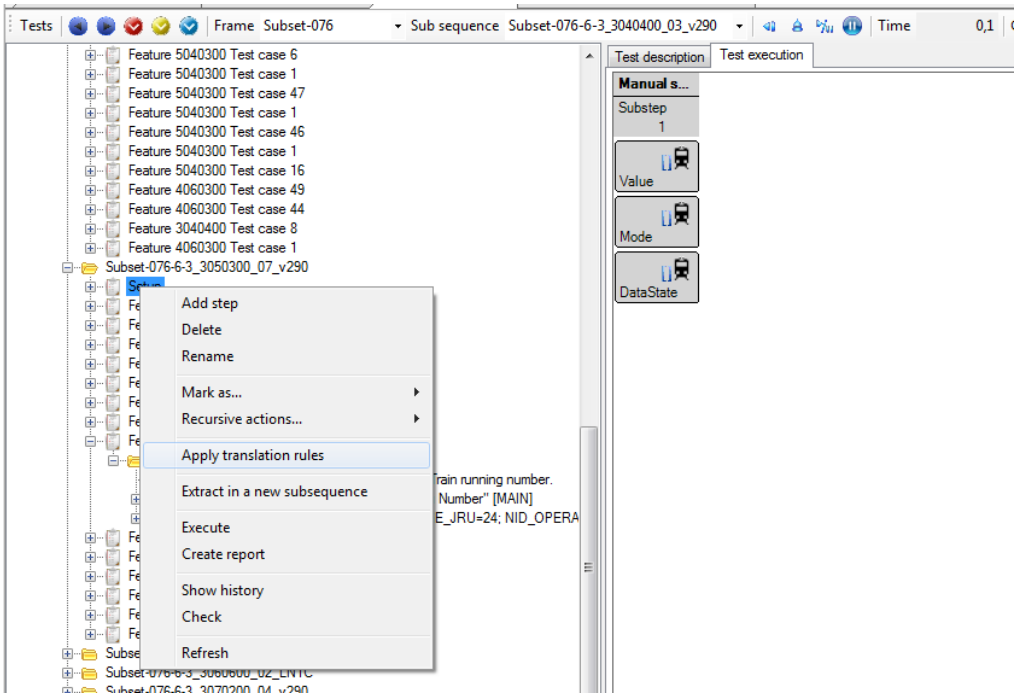


Figure 131: Apply translation rules

Translation is performed according to the following process. For each step that must be translated – a step with **TranslationRequired** flag set to true, as defined in section 10.4.4 – EFS searches for a translation which matches the step text and (possibly) comment. The translation's sub steps are then added to the step.

When translating a sub-sequence using the translation dictionary, EFS performs the following operation, as depicted in Figure 132:

1. For each step in the sequence, EFS finds a matching translation. The translation matches when it contains a source test for which
    a. The source text name corresponds to the step description
    b. If the source text holds comments, one of these comments matches the step comment.
    c. If two translations match the step, the more specific translation is used, that is, the one which matches both **description** and **comment**.
2. The translation's sub-step are copied in the step
3. The action and expectation are adapted to that specific step, by replacing the template variables with the values described in the step, according to following table.

Sub sequence related		
%D_LRBG	DLRBG	Number
%Level	Level	Enumeration Default.Level
%Mode	Mode	Enumeration Default.Mode
%NID_LRBG	NID_LRBG	Number
%Q_DIRLRBG	Q_DIRLRBG	Number
%Q_DIRTRAIN	Q_DIRTRAIN	Number
%Q_DLRBG	Q_DLRBG	Number
%RBC_ID	RBC_ID	Number
%RBCPhone	RBC phone number	String
Step		
%Step_Distance	Distance at which the step takes place	Number
%Step_LevelIN	Level before the step occurs	Enumeration Default.Level
%Step_LevelOUT	Level after the step occurs	Enumeration Default.Level
%Step_ModeIN	Mode before the step occurs	Enumeration Default.Mode
%Step_ModeOUT	Mode after the step occurs	Enumeration Default.Mode

Table 5: replacements for template variables

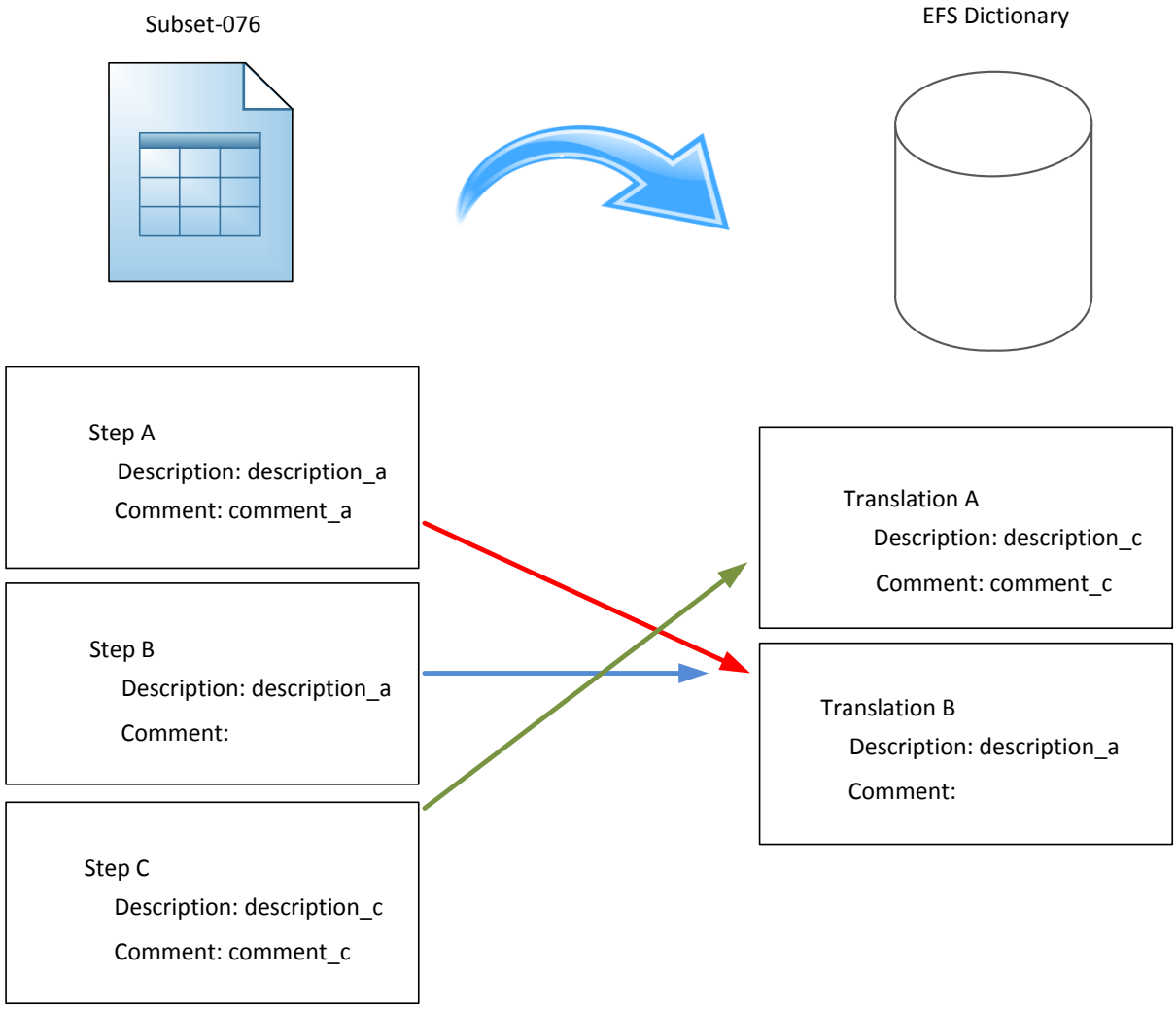


Figure 132: translation process

In addition, if the distance of the current step is different from the distance of the previous step, the translation process adds a new action in the step which follows the template

```
OdometryInterface.UpdateDistance ( %Step_Distance )
```

11.4 Adding translations in the translation dictionary

A new translation can be added in the translation dictionary by either using the contextual menu in the hierarchical view.

Alternatively, dragging a step from the test window in a folder of the translation view creates a new translation, fully configured to match the selected step.

11.5 Navigation

EFSW provides a way to easily navigate to the translation that would be used to translate a step, using the **Show Translation Rule** in the step contextual menu, as shown in Figure 133.

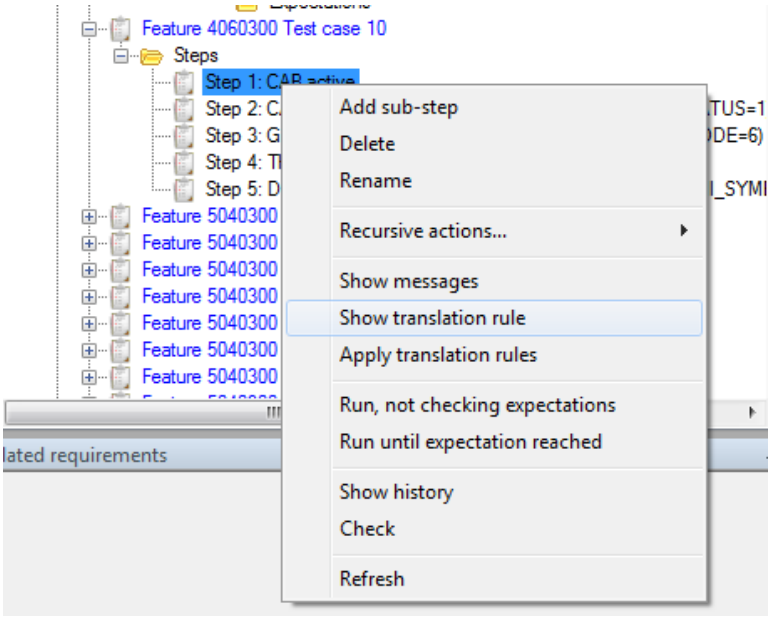


Figure 133: show translation rule contextual menu

11.6 Show messages

Along with the test textual description, Subset-076 also provides the messages exchanged between the onboard and trackside system. These messages are imported when importing the Subset-076 database and can be visualised using Show Message in a step contextual menu, as presented in Figure 134.

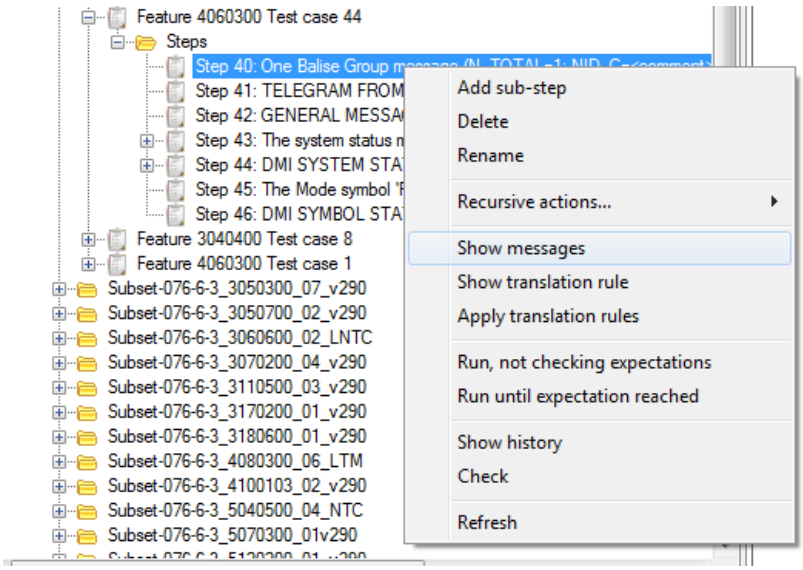


Figure 134: show message

This opens a structured view of the message, as presented in Figure 135 .

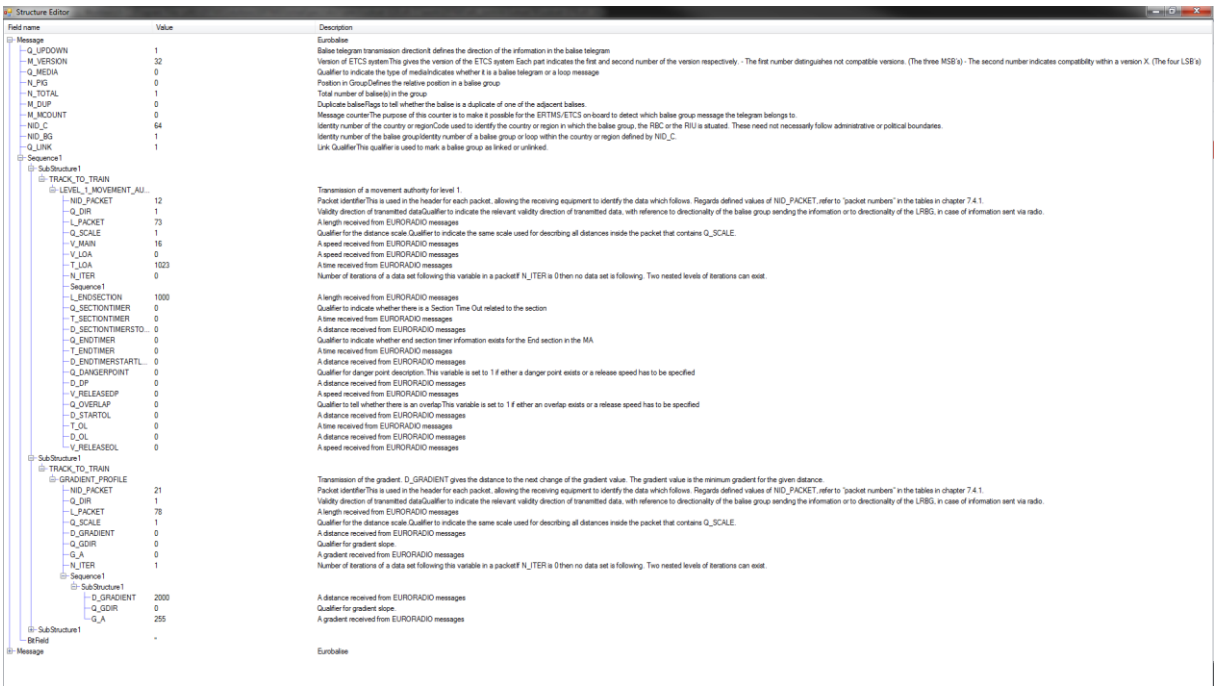


Figure 135: general overview of the message show window

11.7 Add requirement to the translation view browser

Requirements can be linked to the translation rules present on the dictionary browser simply by dragging and dropping the selected requirement on the translation in the translation dictionary.

12 History

EFSW allows to access the underlying Git[10] repository to provide historical data on requirements, model and tests. These features are only available when the model belongs to a git repository. To access the history information for the trainborne model, you should clone the repository located at

https://github.com/openETCS/ERTMSFormalSpecs

Moreover, git should be installed on the machine and accessible in the $PATH.

12.1 History and model comparison

Access to the model history and comparison features is available in the Tools/History as presented in Figure 136.

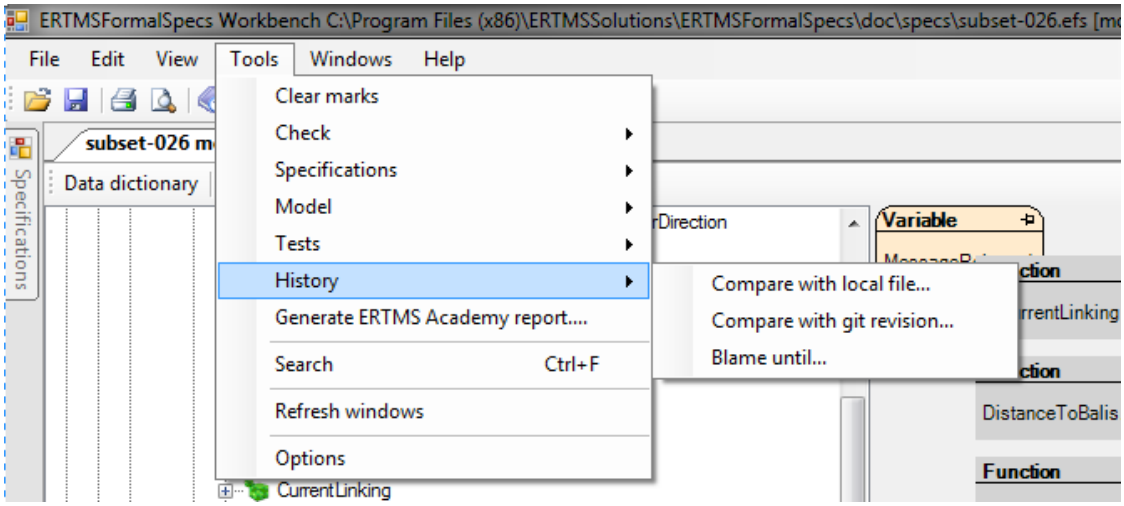


Figure 136: possible history actions

12.1.1 Compare with local file

Two .efs files can be compared in a structured way, using **Compare with local file…** The comparison is done between the .efs file currently loaded and the one selected in the file browser. Differences between those two files are marked as Informative message (see section 7.5 as displayed in Figure 137.

The message has the following structure:

CHANGED <fieldname> FROM text1 TO text2,

which indicates that the field <fieldname> has the value text2 in the currently opened file, and has value text1 in the selected file.

[10] More information about Git can be found at http://git-scm.com/

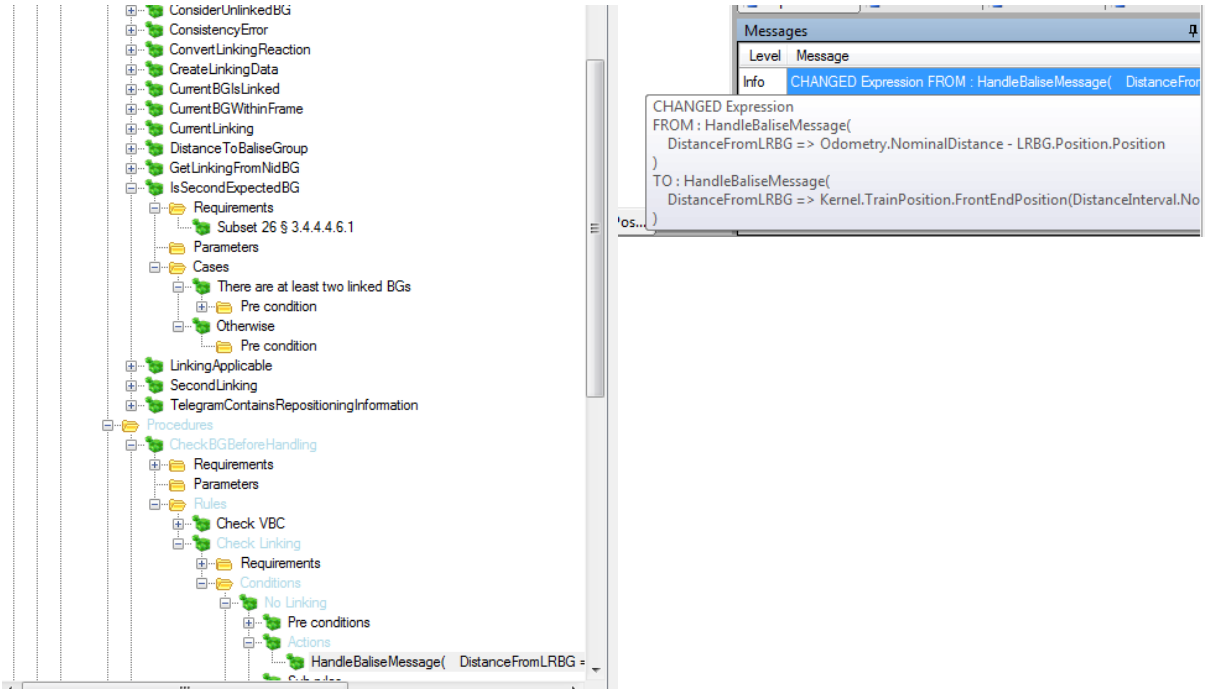
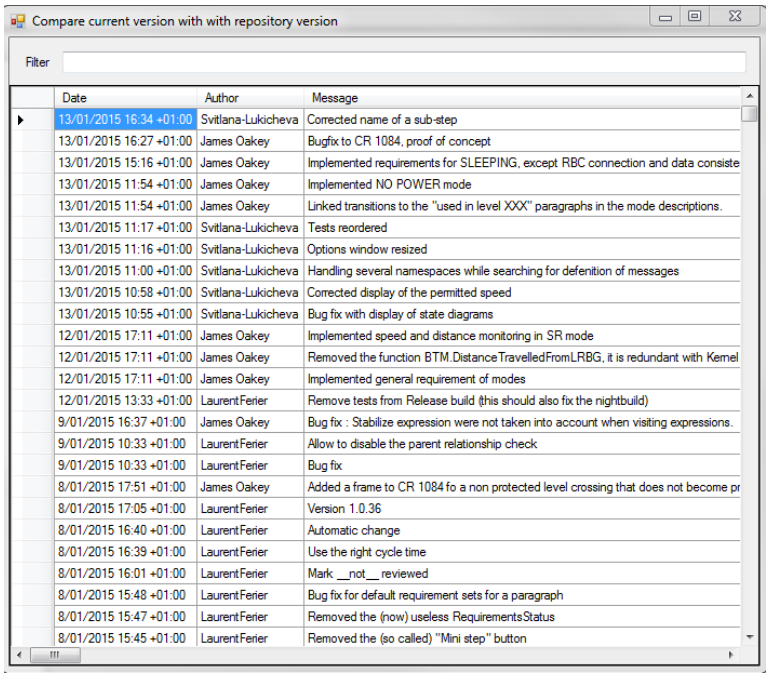


Figure 137: result of comparing two definitions of Subset-026

12.1.2 Compare with git revision

The comparison described on section 12.1.1 can be performed with a previous version of the current model. When executing **Compare with git repository…** , EFW opens the revision selector which displays the commit date, author and commit message of all commits available in the repository, as shown in Figure 138. Selecting the version to compare to the current version is done by double clicking on it.



Date	Author	Message
13/01/2015 16:34 +01:00	Svitlana-Lukicheva	Corrected name of a sub-step
13/01/2015 16:27 +01:00	James Oakey	Bugfix to CR 1084, proof of concept
13/01/2015 15:16 +01:00	James Oakey	Implemented requirements for SLEEPING, except RBC connection and data consiste
13/01/2015 11:54 +01:00	James Oakey	Implemented NO POWER mode
13/01/2015 11:54 +01:00	James Oakey	Linked transitions to the "used in level XXX" paragraphs in the mode descriptions.
13/01/2015 11:17 +01:00	Svitlana-Lukicheva	Tests reordered
13/01/2015 11:16 +01:00	Svitlana-Lukicheva	Options window resized
13/01/2015 11:00 +01:00	Svitlana-Lukicheva	Handling several namespaces while searching for defenition of messages
13/01/2015 10:58 +01:00	Svitlana-Lukicheva	Corrected display of the permitted speed
13/01/2015 10:55 +01:00	Svitlana-Lukicheva	Bug fix with display of state diagrams
12/01/2015 17:11 +01:00	James Oakey	Implemented speed and distance monitoring in SR mode
12/01/2015 17:11 +01:00	James Oakey	Removed the function BTM.DistanceTravelledFromLRBG, it is redundant with Kernel
12/01/2015 17:11 +01:00	James Oakey	Implemented general requirement of modes
12/01/2015 13:33 +01:00	LaurentFerier	Remove tests from Release build (this should also fix the nightbuild)
9/01/2015 16:37 +01:00	James Oakey	Bug fix : Stabilize expression were not taken into account when visiting expressions.
9/01/2015 10:33 +01:00	LaurentFerier	Allow to disable the parent relationship check
9/01/2015 10:33 +01:00	LaurentFerier	Bug fix
8/01/2015 17:51 +01:00	James Oakey	Added a frame to CR 1084 fo a non protected level crossing that does not become pr
8/01/2015 17:05 +01:00	LaurentFerier	Version 1.0.36
8/01/2015 16:40 +01:00	LaurentFerier	Automatic change
8/01/2015 16:39 +01:00	LaurentFerier	Use the right cycle time
8/01/2015 16:01 +01:00	LaurentFerier	Mark __not__ reviewed
8/01/2015 15:48 +01:00	LaurentFerier	Bug fix for default requirement sets for a paragraph
8/01/2015 15:47 +01:00	LaurentFerier	Removed the (now) useless RequirementsStatus
8/01/2015 15:45 +01:00	LaurentFerier	Removed the (so called) "Mini step" button

Figure 138: remote GIT version to be selected for comparison

After comparison, elements that have been modified are highlighted in blue (see section 12.1.1).

12.1.3 Blame until

The tool **Blame until…** builds a database of the changes between the current version and a version selected from the Git repository, using the selector dialog presented in Figure 139. Double click on an entry to select it.

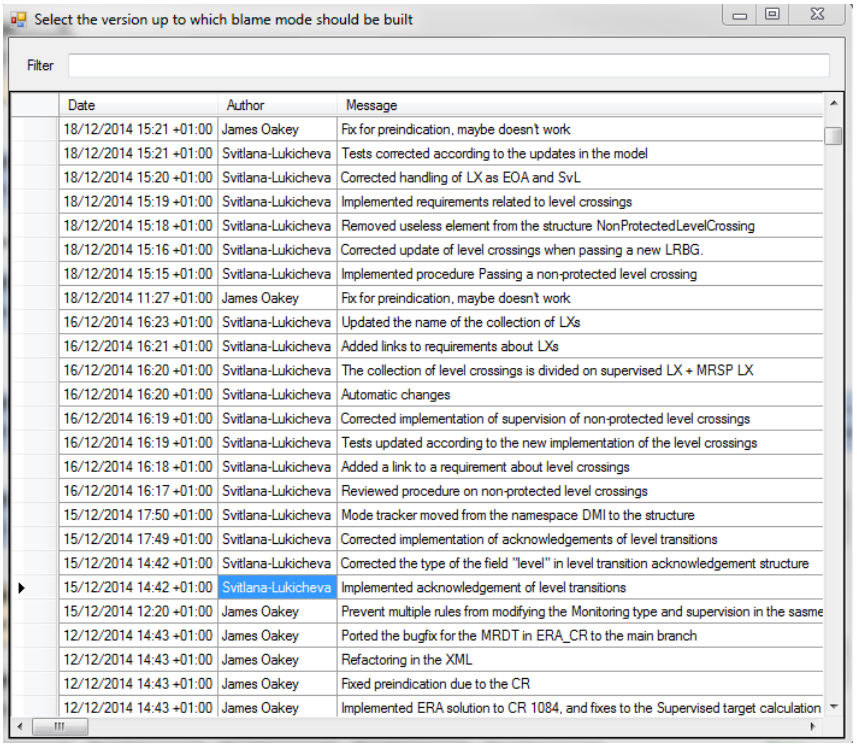


Figure 139: selection of the blame until filter

During the process, EFSW displays the dialog presented inFigure 140.

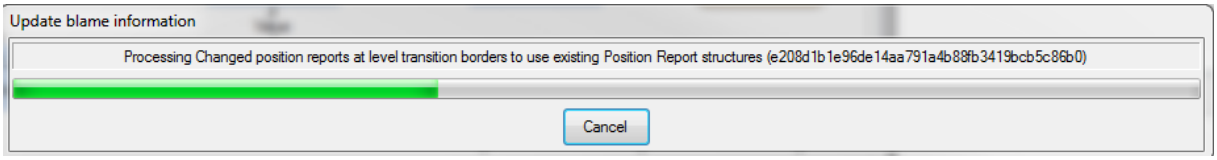


Figure 140: processing the information for the blame until option

Data gathered during this process are used to fill the History window presented in Figure 141. This window is automatically updated when selecting an element (requirement, model object, test, …) in EFSW.

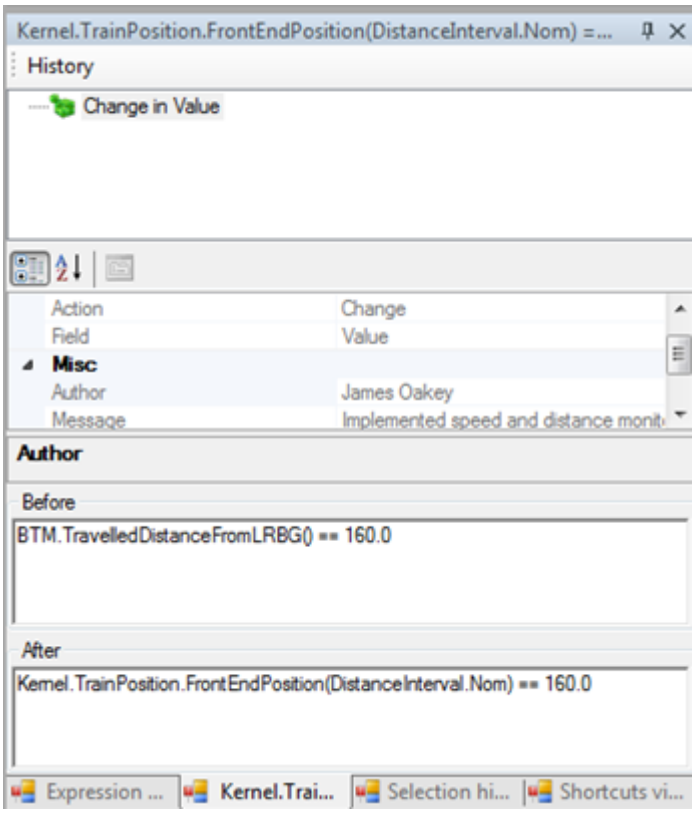


Figure 141 : history window

This window holds the following information

- **History**: traces all changes in the selected element.
- **Properties** of the change.
  - **Action**: indicates the operation performed, which can be either Add (a new element has been added), Remove (an element has been removed) or Change (a modification has been performed in the element)
  - **Field**: Indicates which field of the element has been altered
  - **Author**: provides the name of the author of the commit
  - **Message**: provides the message the author provided during the commit
  - **Date**: the date of the commit
- **Before**: in case of a change or a deletion, provides the value of the field before the action was performed.
- **After**: in case of a change or an addition, provides the value of the field before the action was performed.

13 ERTMSFormalSpecs reports

13.1 Specification coverage report

13.1.1 Purpose

The purpose of the specification coverage report is to provide requirement coverage information based on the modelled elements, along with implementation statistics.

The **specification coverage** part of the report provides the status of all the requirements specified in the requirement document. This status can be one of the followings:

- **Implemented**: indicating that the modelling of the requirement is complete. This also ensures that all model elements which model this requirement are also marked as implemented.
- **Not implementable**. Indicates that the paragraph does not need to be implemented because it is not a requirement.
- **Not implemented**: indicating that the modelling of the requirement is not completed.
- **N/A**. indicates that the implementation status of this paragraph is not yet known. In this case, implementation is not performed.

The **requirement coverage report** provides the list of all the requirements of the specification along with the model elements that implement them.

The **model coverage report** provides the list of requirements associated to each model element.

13.1.2 Structure

The specification coverage report can be composed of four following chapters:

- **Specification coverage**. This chapter holds two sections:
  - **Statistics**. This section provides a table with the following information:
    - Total number of specification paragraphs.
    - Number of applicable paragraphs. The paragraph is applicable if its scope is "OBU" (on board unit) or "OBU and Track" and if its type is "Requirement".
    - Number and percentage of covered paragraphs. This percentage is computed from the total number of applicable paragraphs. A paragraph is considered as covered if it is marked as "implemented" and all the EFS elements that are related to this paragraph are marked as "implemented".
  - **Specification**. This section provides a table with an entry for each specification paragraph. For each paragraph the table provides its scope ("OBU", "OBU and Track" or "Track"), its type and its implementation status.
- **Covered requirements**. This chapter provides the list of requirements covered by the model and can be composed by the two following sections:
  - **Statistics**. This section provides a table with the following information:
    - Number of applicable paragraphs.
    - Number and percentage of covered requirements.

- **Covered requirements**. This section provides a table with an entry for each covered requirement. For each covered requirement, the table provides the list of the model elements that implement it with the associated comment.

- **Non-covered requirements**. This chapter provides the list of requirements that are not yet covered by the model and can be composed of two following sections:
  - **Statistics**. This section provides a table with the following information:
    - Number of applicable paragraphs.
    - Number and percentage of non-covered requirements.
  - **Non-covered requirements**. This section provides the list with the non-covered requirements.
- **Model coverage**. This chapter provides the list of implemented model elements and can be composed of the following sections:
  - **Statistics**. This section provides a table with the following information:
    - Number of implemented model elements.
    - Number and percentage of modelled paragraphs.
  - **Implemented rules**. This section provides a table containing an entry for each implemented rule. For each implemented rule the table provides the (list of) the paragraph(s) it implements.
  - **Implemented types**. This section provides a table containing an entry for each implemented type. For each implemented type the table provides the (list of) the paragraph(s) it implements.
  - **Implemented variables**. This section provides a table containing an entry for each implemented variable. For each implemented variable the table provides the (list of) the paragraph(s) it implements.

13.1.3 Launch the specification reporting

The specification coverage report creation window is accessible via Tools/Specifications/Generate coverage report... (See Figure 142).

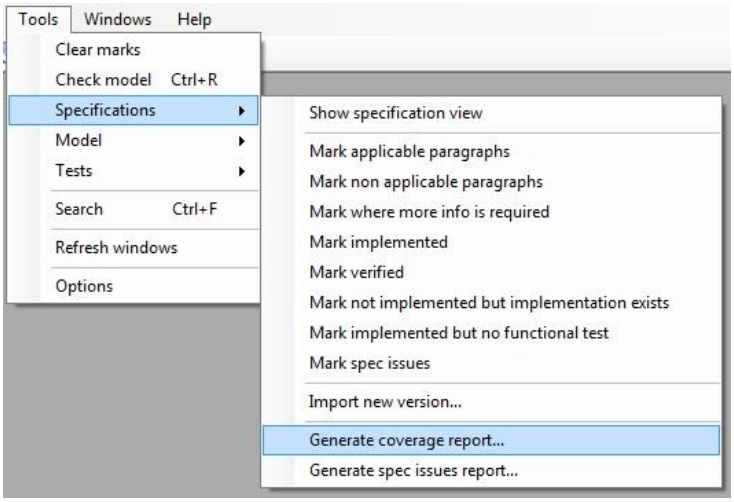


Figure 142: launch the specification coverage report

This opens the dialog which allows to select the report options, as depicted below

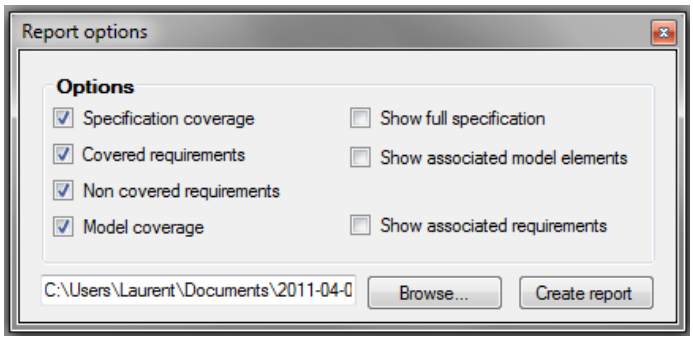


Figure 143: Specification coverage report options

The figure above shows the window used to select the different report options. The check boxes in the left column allow to create the corresponding report chapters with their "Statistics" section. The check boxes of the right column allow to add additional information described below:

- **Show full specification** check box is available only when the "Specification coverage" check box is selected. This option allows to add the "Specification" section to "Specification coverage" chapter.
- **Show associated model elements** check box is available only when the "Covered requirements" check box is selected. This option allows to add the information about the elements implementing the covered requirements in "Covered requirements" section of "Covered requirements" chapter.
- **Show associated requirements** check box is available only when the "Model coverage" check box is selected. This option allows to add the list of requirements modelled by each model element in the sections "Implemented rules", "Implemented types" and "Implemented variables" of "Model coverage" chapter.

The "Browse" button allows to select the folder and the name of the generated report.

13.2 Generate spec issue report

The purpose of this report, is to summarise the specification issues encountered during analysis. One can generate this report using the menu item Tools/Specifications/Generate spec issues report… .

13.2.1 Structure

The report is divided in different chapters.

- **More information needed**: indicates the requirements for which the description is not precise enough and requires more information. These issues are composed by:
  - **Description**: the requirement text, as written in the specification.
  - **Comment**: a comment made by the developer.
- **Specification issues report**: provides the requirements which pose problems, and cannot be modelled as such. Each issue is composed of
  - **Description**: the requirement text, as written in the specification.
  - **Comment**: a comment made by the developer.
- **Design choices**: Provides the list of new requirements requires to model the system. They are composed of
  - **Description**: the new requirement text
  - **Comment**: an optional comment.

13.2.2 How to create a specification issue

An element of the specifications appears on the SpecIssue report when its **SpecIssue** flag is set to true. Section 8.2.2 describes the properties of the specifications.

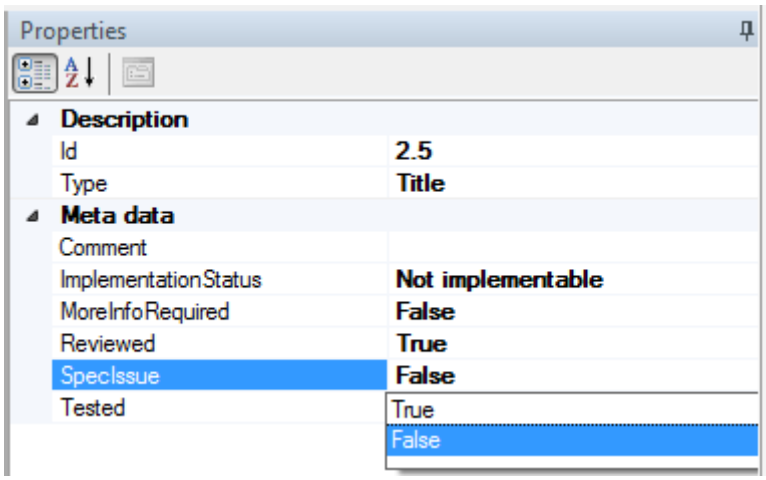


Figure 144: setting to true the SpecIssue flag

This action should be performed to all the requirements which contains any kind of specification issue.

13.2.3 Launch the report

To generate the Specs issues report, go to Tools/specifications/Generate spec issues report. After clicking on the option, a menu indicating the different options contained on the report appears.

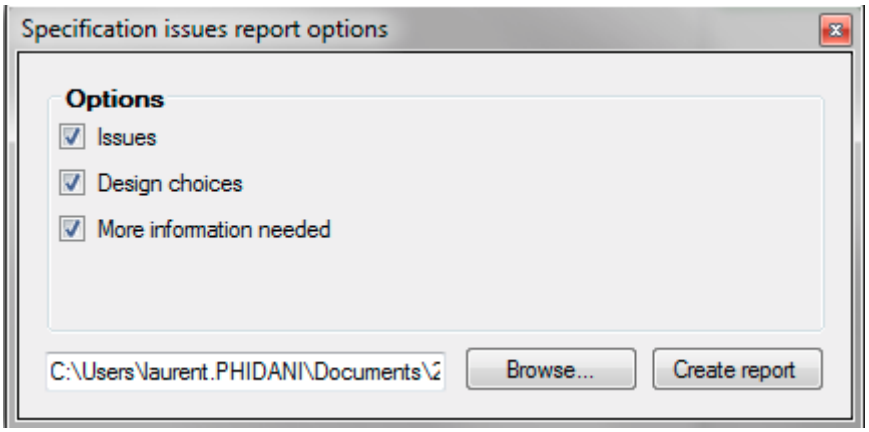


Figure 145: contents of the specification issues report

Select the place to store the resulting report and then create the report. Figure 146 shows an example of the specification issues report.

More information needed

3.5.5.1.b is not precise enough

3.5.5.1.b	
Description	If an error condition requiring the termination of the communication session is detected on-board (e.g., not compatible system versions between on-board and trackside).
Comment	[stan@ertmssolutions.com] Any other errors requiring this?

3.6.6.9.b is not precise enough

3.6.6.9.b	
Description	it is told not to do so, OR
Comment	How can it be told not to do so?

3.13.9.3.5.6 is not precise enough

3.13.9.3.5.6	
Description	In case the calculation of the GUI curve is enabled, for display purpose only, the P speed related to SBD shall be calculated for the estimated train front end as follows: V_P_EOA (d_estfront) = min { V_SBD (d_estfront + Vest * (T_driver + T_bs1)), V_GUI_EOA (d_estfront) } V_P_EOA (d_estfront) = 0 if d_estfront + Vest * (T_driver + T_bs1) >= d_EOA
Comment	V_GUI_EOA is not defined.

Figure 146: extract of the specification issues report

13.3 Generate data dictionary report

13.3.1 Purpose

The purpose of the data dictionary report is to provide information about the model. The report can be created on two different levels of details:

- Default level: the report provides only the list of implemented model elements together with their associated requirements.
- Detailed level: the report provides all the available details for each implemented model element.

13.3.2 Structure

The report is divided in several chapters each one corresponding to one of the data dictionary namespace. Depending on the user's choice, each chapter can contain information about its

- Ranges
- Enumerations
- Structures
- Collections
- Functions
- Procedures
- Variables
- Rules

For each element described above, the data dictionary report provides a comment describing its utility. The report indicates the implementation and verification status of each model element.

13.3.3 Launch the model report

The data dictionary report creation window is accessible via Tools/Model/Generate data dictionary report... (See Figure 147).

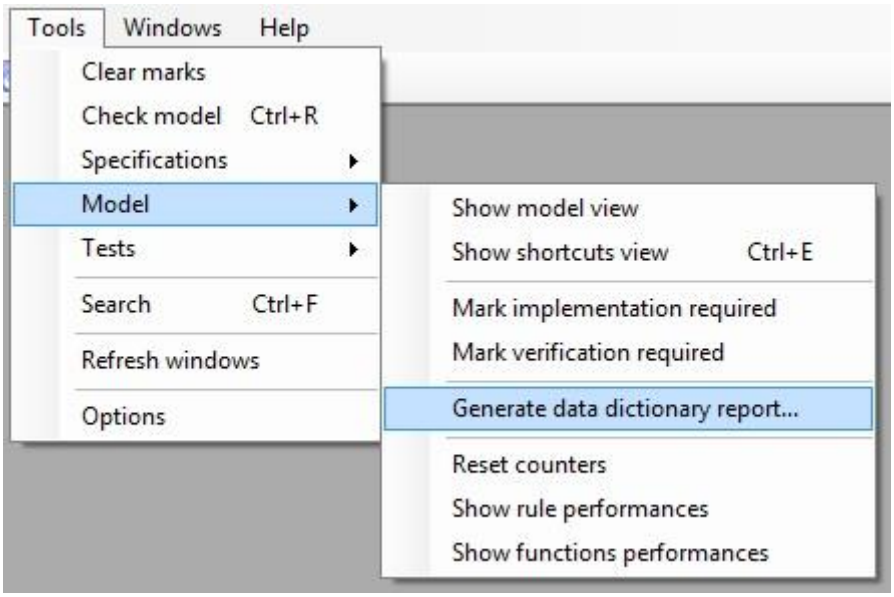


Figure 147: Launch the data dictionary report

This opens the dialog which allows to select the report options, as depicted below.

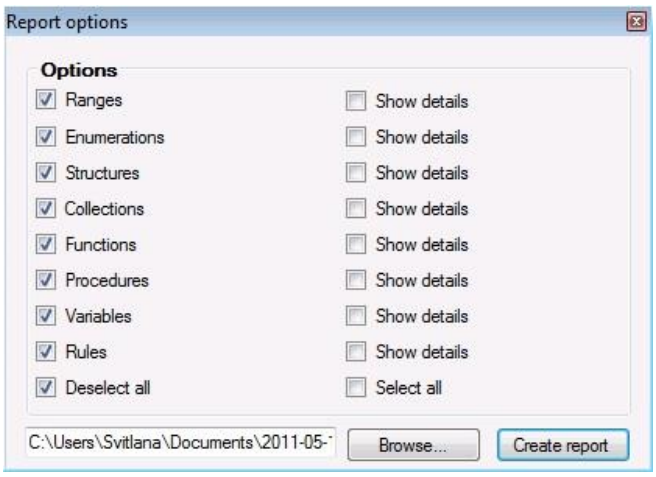


Figure 148: Data dictionary report options

The different check boxes allow to select the type of elements to be included in the report, and precise if the latter has to describe the details of these elements.

13.4 Generate functional analysis report

13.4.1 Purpose

Generates a report for all the functions and procedures in the system, along with the relationship between namespace and the function/procedure. It clearly indicates the functions that are exposed by a namespace and the locations where that function is used.

13.4.2 Structure

The functional analysis report is divided in several chapters, one per namespace, and presents, for one of them, the namespace's **Exposed functions/procedures**. An exposed function or exposed procedure is defined in a namespace and used in another. This report provides the relationship between namespaces.

Each entry presents the following information, as shown in Figure 149.

- Function or procedure **name**
- Function or procedure **parameters,** identified by a name and a type
- Function return value
- Requirements related to the function or procedure
- Known **usages**: provides the list of package which are using the function or procedure.

Function MaxSpeedFunction

MaxSpeedFunction	
This function provides the maximum speed	
Parameters	
Name	**Type**
Distance	Double
Return value	
Default.BaseTypes.Speed	
Related requirements	
No requirements related to this element	

Known usages

Usage
Kernel.TrackDescription.AxleLoad
Kernel.SpeedAndDistanceMonitoring.DecelerationCurves.GUI
Kernel.LX
Kernel.MRSP
Kernel.TrackDescription.PermittedBrakingDistance
Kernel.TrackDescription.StaticSpeedProfile
Kernel.TSR

Figure 149: extract of a functional analysis report.

13.4.3 Launch the functional analysis report

The Generate functional analysis report can be accessed by Tools/Model/Generate functional analysis report.

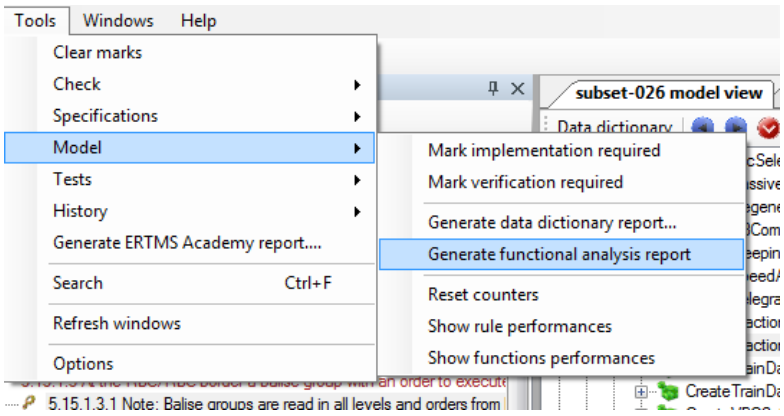


Figure 150: launching the functional analysis report tool

13.5 Dynamic tests coverage report

13.5.1 Purpose

The purpose of the dynamic tests coverage report is to provide model dynamic coverage by either all or a set of tests. The report can be created for the complete tests set of the model or for a certain element of the tests tree. In that case the level of a selected element is the topmost. For example, a report can concern

- The whole tests tree, containing information of all its levels
- All the available frames

- All the frames and all the sub sequences
- A particular sub sequence with all it sub cases
- A particular test case

13.5.2 Structure

For each selected level, the report provides the percentage of activated rules of the EFS model and (if selected) the list of activated rules and/or the list of rules that weren't activated.

13.5.3 Launch the test coverage reporting

The specification coverage report creation window is accessible via Tools/Tests/Generate dynamic coverage report... (Figure 151).

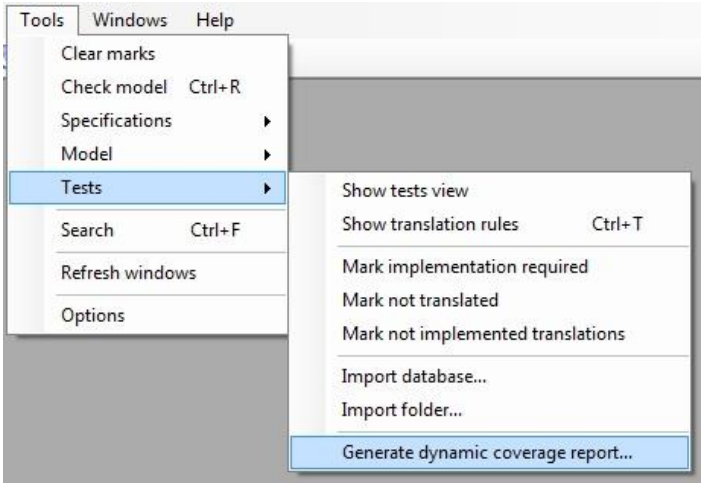


Figure 151: Launch the dynamic test coverage report

The option of a **partial** report creation for a selected item of the tests tree is accessible via the "Create report" option from its contextual menu.

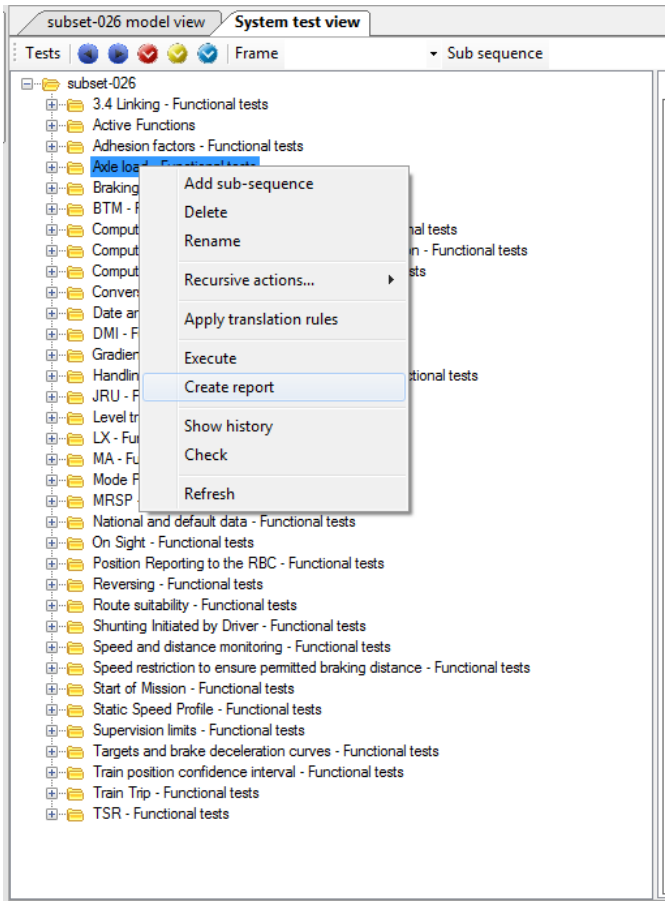


Figure 152: Report creation for a specific element on the test hierarchical tree

This action opens the dialog box displayed below.

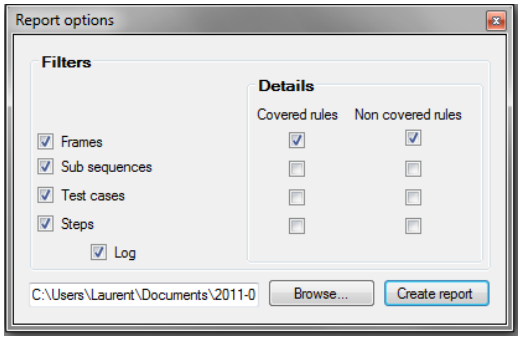


Figure 153: Dynamic tests coverage report options

Figure 153 shows the dialog which allows to select the different report options. The check boxes in "Filters" group box allow to select the different levels of the report. The corresponding check boxes in the "Details" group box allow to add the list of covered and/or not covered rules to the corresponding level. "Log" check box allows to display the log information for the different steps.

The "Browse" button allows to select the name and the folder of the generated report.

13.6 Generate findings report

13.6.1 Purpose

The findings report contains the findings detected while modelling Subset-076 tests:

- **Comments**: possible improvements to the subset-076 specification.
- **Questions**: elements of the Subset-076 which its explanation and description is not clear enough.
- **Bugs**: problem detected on the test specification.

13.6.2 Structure

The report is divided in two different chapters. The first one contains the currently open findings and on the second one the findings which have been addressed.

13.6.3 Launch the findings report

To activate the findings report, open Tools/Tests/Generate Findings Report… Figure 154 illustrates how to access to the findings report

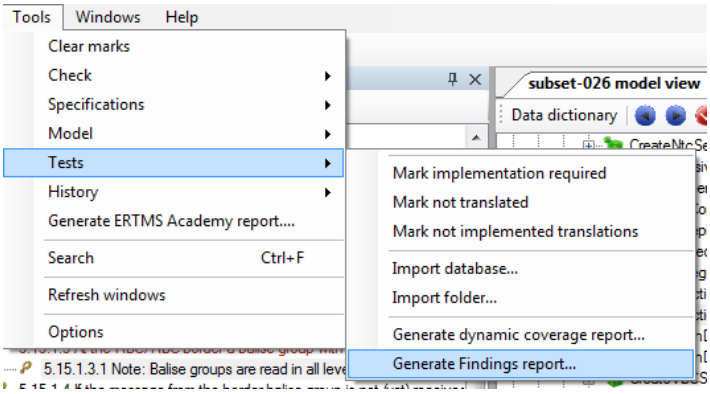


Figure 154: activating the findings report

Then, select a place to save it and click on create report. Figure 155 illustrates the contextual menu related with this procedure.

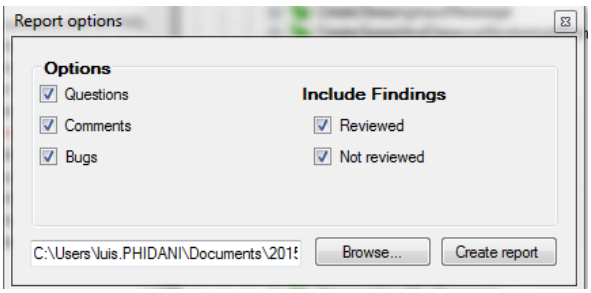


Figure 155: selecting the contents of the findings report

The file contents are described in section 13.6.1. The findings described on the report can be classified according their revision status.

13.7 Generate ERTMS academy report

13.7.1 Purpose

The ERTMS academy report describes the evolution, in terms of number of implemented requirements and test, of a student during a given period of time.

13.7.2 Structure

The report is composed by a single chapter which contains all the information related with the student's progress. It holds a list of addition and deletion in the model. Each entry on the list contains:

- **Author**: student responsible of the addition or deletion.
- **Comment**: explicative message written before committing and pushing the modifications on GIT.
- **Statistics**: brief of the elements modified, deleted or added on the model.

Figure 156 shows a typical entry for this list.

Added on 17/02/2014 15:20:28 +01:00	
Author	luis(luis@ertmssolutions.com)
Comment	Test Coment from las Pull Request
Statistics	JRU.efs_ns 2 addition(s), 2 deletion(s) JRU - Functional tests.efs_tst 91 addition(s), 0 deletion(s) 2 file(s) changed, 93 addition(s), 2 deletion(s)

Figure 156: ERTMS Academy report extract.

13.7.3 Launch the ERTMS academy report

For activating the ERTMS academy report open Tools/Generates ERTMS Academy report… Figure 157 proves the procedure to generate an ERTMS Academy report.

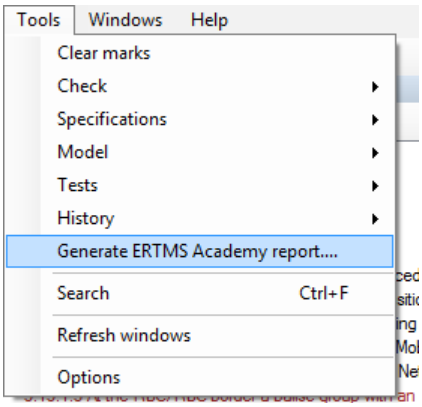


Figure 157: activating the ERTMS Academy report

Then, provide the name of the student and the point from the last ERTMS Academy report done. See Figure 158.

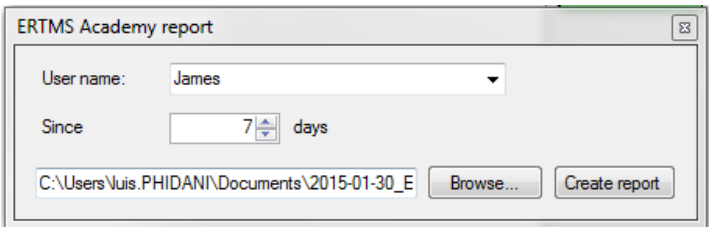


Figure 158: configuration for creating the ERTMS Academy report

14 ERTMSFormalSpecs general tools

14.1 Clear marks

This tool is located on Tools/Clear marks, as Figure 159 shows.

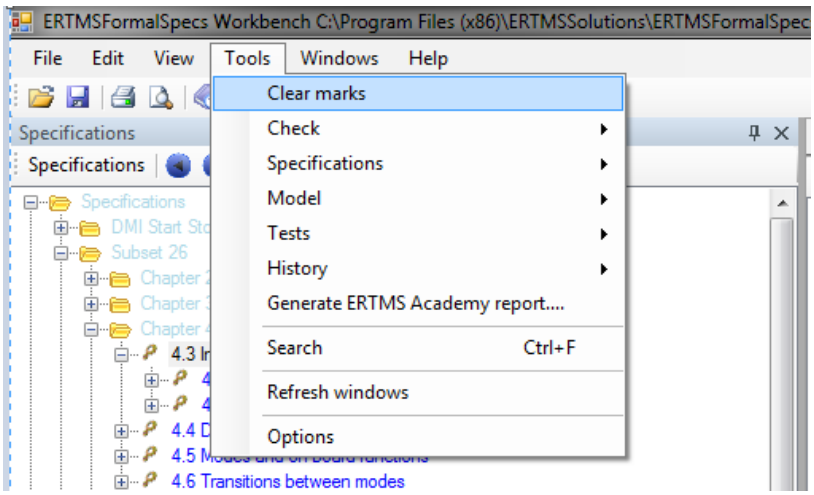


Figure 159: clear marks option location

It lets the user to unmark all marked elements in the system (remove all messages), see section 7.5.

14.2 Search

EFSW provides a way to search elements in the entire model (requirements, model, tests, …). This feature is located on Tools/Search or can be activated using the Crtl+F, which displays the following search dialog (Figure 160).

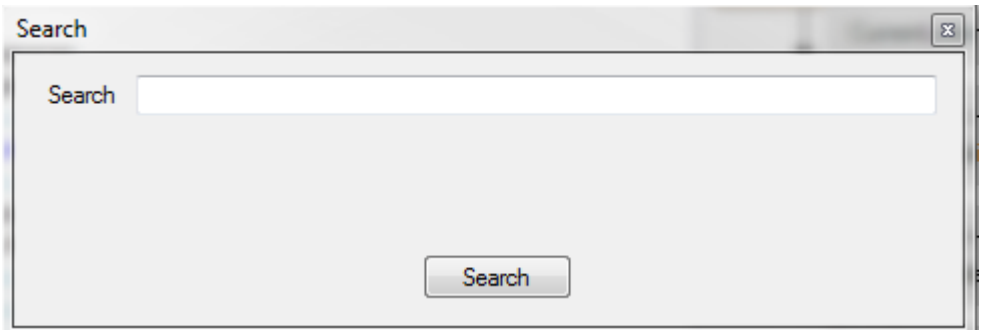


Figure 160: contextual menu for searching

The elements related with the search criteria are highlighted in blue as shown in Figure 161.

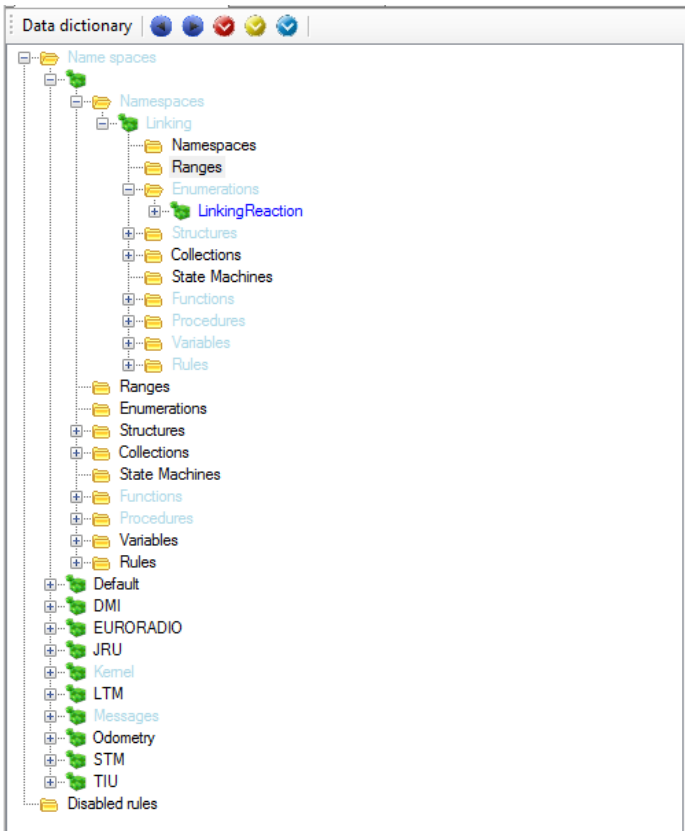


Figure 161: results of the search feature

14.3 Refresh windows

To refresh the windows contents, use the Tools/Refresh windows. This re-computes and re-draws all windows.

14.4 Options

EFSW behaviour can be configured using the Option dialog box. To open the Options dialog, click on Tools/Options. This dialog allows to edit the following options

- Display all variables in structure editor. This option is used to indicate if the structure editor should display sub-variables whose value is EMPTY or not.
- Enclosing messages. This option is used to indicate that the messages related to the enclosing elements should be displayed when selecting a model element.
- Requirements as a list. This modifies the behaviour of the requirements window, when set to true, the window only displays the requirement identifier, and otherwise, the window displays the requirement identifier along with the requirement text.
- Lock opened files. When set to true, EFS locks the files opened during a session. This forbids external applications, such as GIT, a text editor, … to access those files, hence work on an inconsistent set of sources.

15 Shortcuts

The following table summarises the shortcuts available in EFSW.

Shortcut	Meaning	Location
Ctrl+N	New file	File
Ctrl+O	Open a file	File
Ctrl+S	Save modifications	File
Ctrl+P	Print	File
Ctrl+Z	Undo	Edit
Ctrl+Y	Redo	Edit
Ctrl+X	Cut	Edit
Ctrl+C	Copy	Edit
Ctrl+V	Paste	Edit
Ctrl+A	Select all	Edit
Ctrl+E	Show shortcuts view	View
Ctrl+R	Check the model	Tools
Ctrl+D	Check for dead model	Tools
Ctrl+F	Search	Tools
Ctrl+F1	contents	Help

Table 6: quick access controls

15.1.1 Auto completion

The expression editor provides a feature to auto-complete the expression. It becomes active when typing "Ctrl+space". It is applicable for all the elements present on the model, and provides the name of the related element. In case there are several elements enclosed, the auto completion feature offers a list of possible names. Figure 162 illustrates an example of the auto-completion feature.

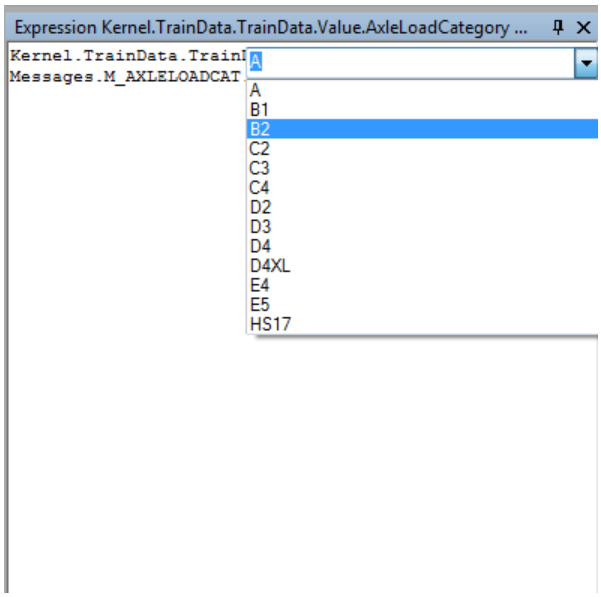


Figure 162: auto completion feature suggestion list

15.1.2 Quick navigation to a model element

EFSW allows to easily navigate from element usage to definition, using "Ctrl+click" as shown in Figure 163. This is only available in the *Expression editor* and *More info* view.

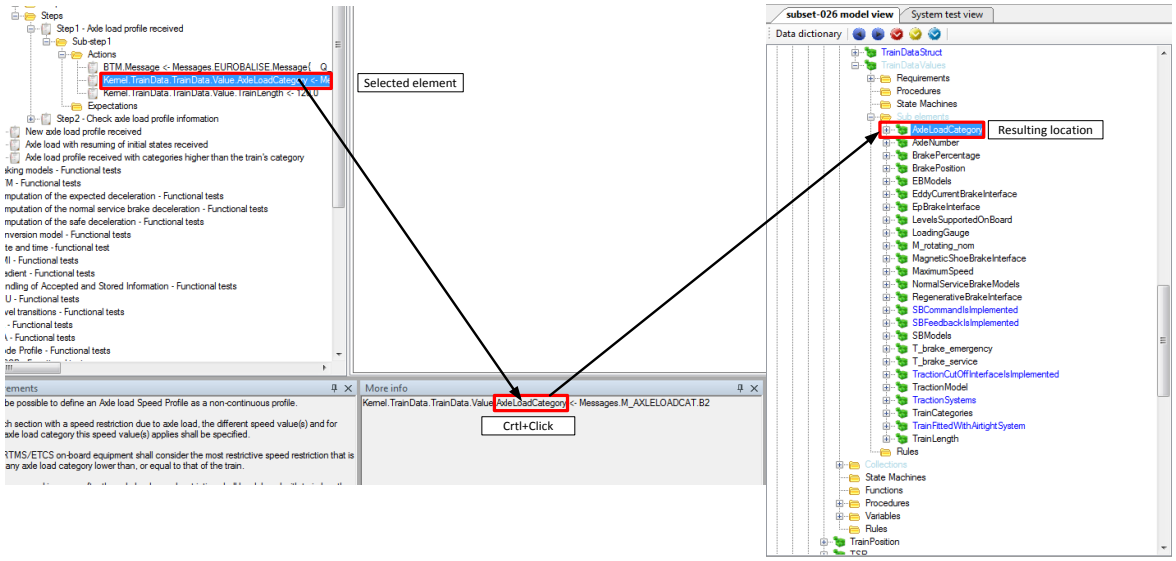


Figure 163: representation of the Crtl+Click shortcut

Moreover, right clicking on an element in the *More info* window or in the *Expression editor* displays that element's short description, as presented in Figure 164.

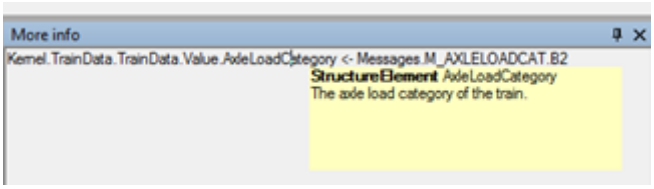


Figure 164: quick access to the selected element related information

16 Frequent Answers and Questions

16.1 Usages view and navigation

The usages window displays the locations where the corresponding element is used, grouped using two different categories:

- Model: other places of the model where the selected element is used.
- Test: displays the test where the selected element of EFSM is used.

To navigate to one of those locations, double click on the left icon.

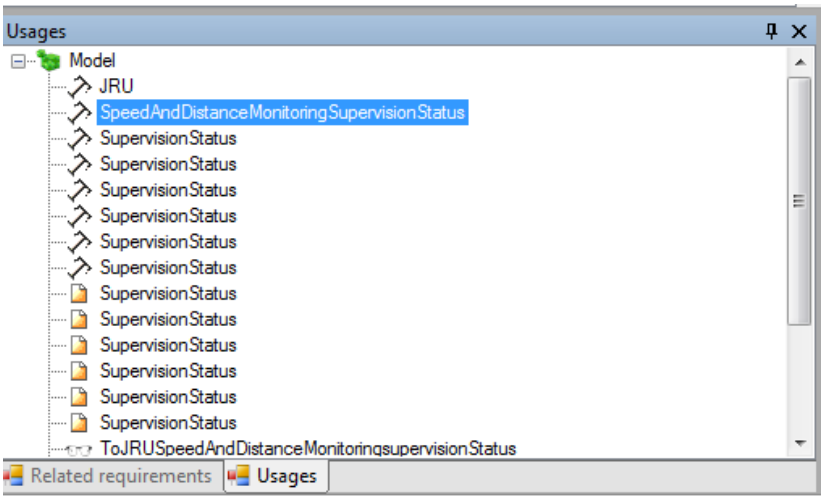


Figure 165: list displaying other locations where an element of the model he is used

16.2 ERTMSFormalSpecs Workbench windows

Sub-windows of EFSW can be moved. To change the location of a sub-window, select it; click and keep clicked on its frame and drag to the desired location. Figure 166 depicts an example to re-allocate a component of the EFSW main window, in this case the selected window is the messages window. Figure 167 depicts the new location of the messages window on the EFSW.

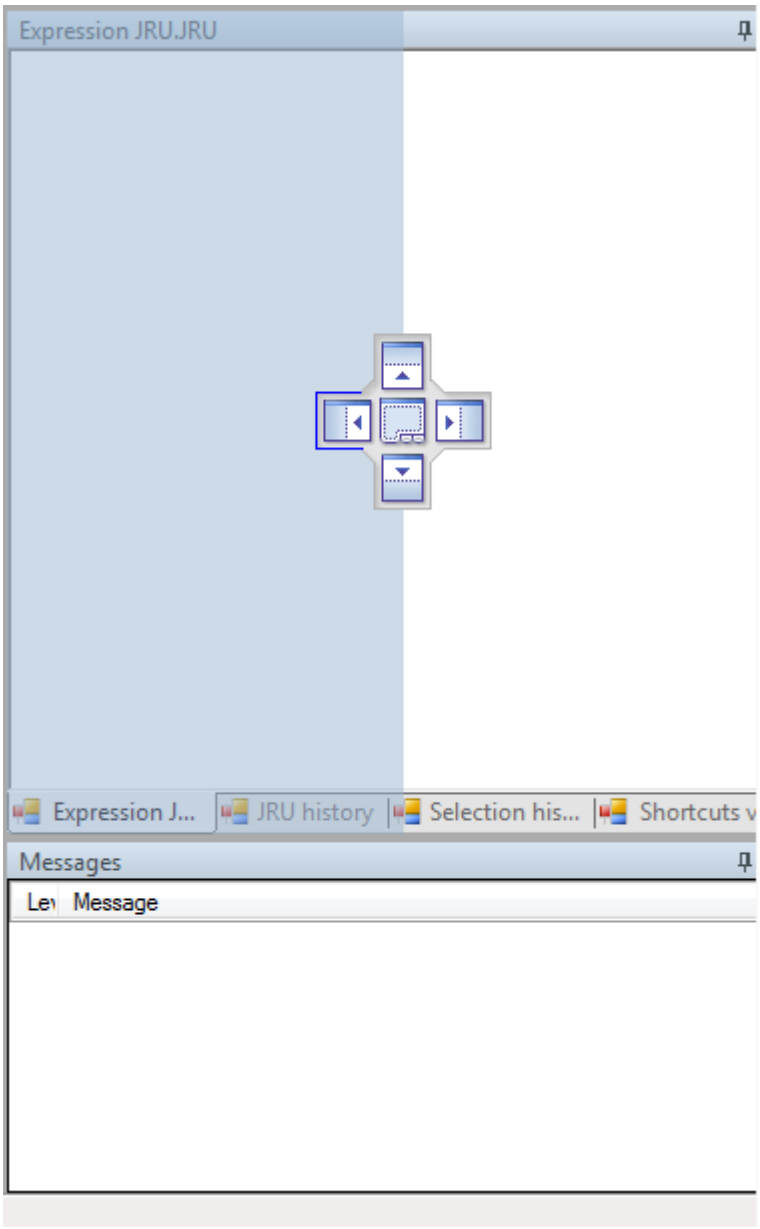


Figure 166: re-allocation the messages window

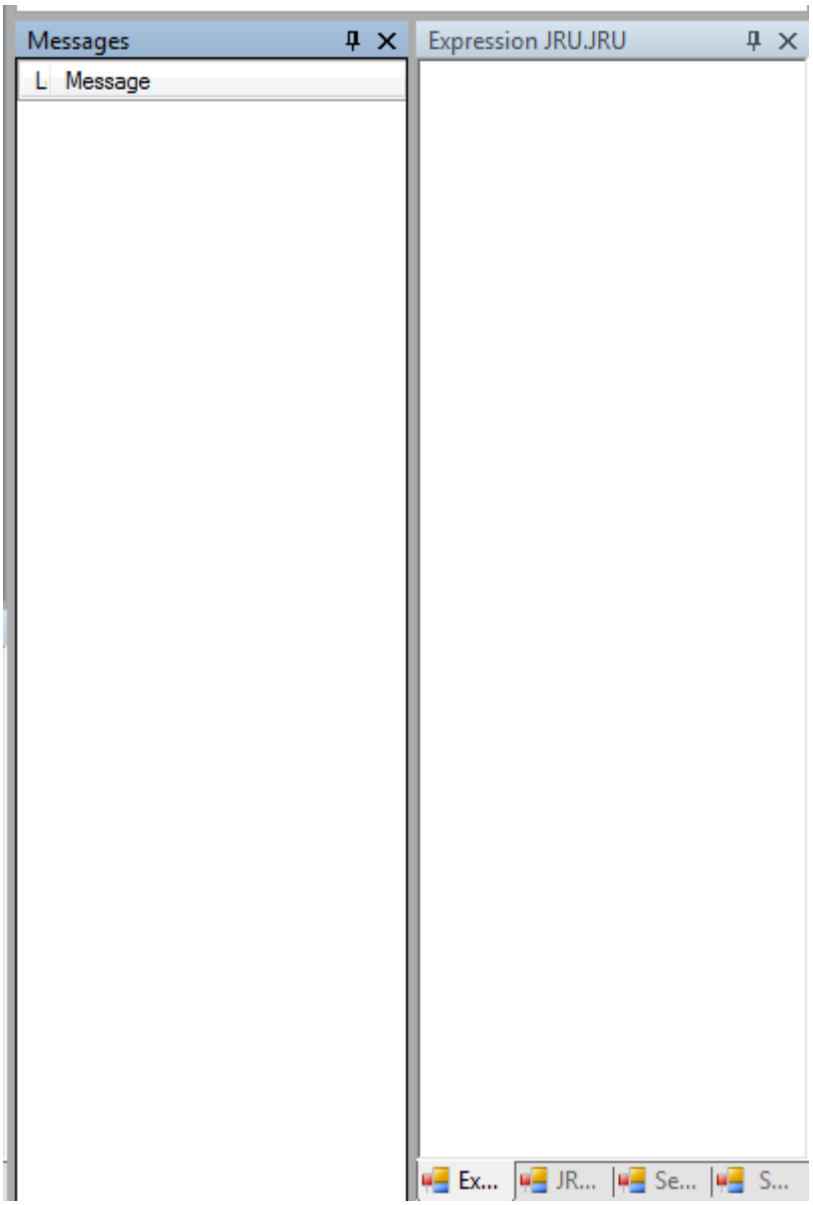


Figure 167: messages window re-allocation result