Requirement Specification

Contents

License Information

This is a free document: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This document is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this artifact. If not, see <*http://www.gnu.org/licenses/*>.

Acknowledgement

The BUSMASTER application suite evoluted from its incipient stage of monolithic single bus - single controller architecture towards the present stage of modular & reusable component based architecture, easily and seamlessly extendable to support multiple vehicle buses, exhibiting a rich set of carefully crafted feature list. Reaching of such a milestone was possible only by the valuable contributions from many dedicated and skilful professionals who were involved at different cycles of the development process defining the trajectories of the product's evolution, translating many innovative ideas into reality. The contributors so far are Mrs. Jalaja K.V, Mr. Amitesh Bharti, Mr. Amarnath Shastry, Mr. Ratnadip Choudhury, Mr. Pemmaiah B.D, Mr. Raja N, Mr. Rajesh Kumar R, Mr. Anish Kumar, Mr. Pradeep Kadoor, Mr. Arunkumar Karri and Mr. Venkatanarayana Makam.

This RS document is the summation of all the relevant RS documents, notes and analysis reports produced and refined throughout the development process, and hence is an integrated version prepared to roll out the OSS version of BUSMASTER.

List of Abbreviations

DFD	Data Flow Diagram
E-R	Entity Relationship
OSS	Open Source Software
RBEI	Robert Bosch Engineering and Business Solutions Limited
RS	Requirement Specification

Prolegomena

Purpose of the document

This document combines purposes of both RS and RA. The former one focuses on understanding and stating the requirements of the BUSMASTER suite whereas the later one comments on the analysis in different aspects, especially from the 4+1 view architecture, and feedbacks towards possible modification of the former. Emphasis is given on the motivation behind each of the modules. The RS is covered from different functionality viewpoints.

Project Scope

The project is taken up as product development Initiative. Its scope is to develope an automotive bus monitoring, analyzing and node simulation suite named as BUSMASTER. This can be used to fulfill the family engineering aspect; both from the viewpoint of commonality and predicted variability; to be used as the development infrastructure. The application engineering part focuses on creating the concrete product that targets a particular bus. End product can be either a desktop based GUI application or an automation server. The goal is also to support tools for multiple buses under the same umbrella.

Tagging

Tagging follows the below convention:

RS_<module number>_<incremental number of RS for this module>

All the module names and the number assigned to each of them are tabulated below:

Module name	Module No.
Error Logger	1
License Validating Module	2
Utility library	3
DataType library	4
SimulationENG	5
Project Configuration	6
DIL_FlexRay	7
Main UI (Control unit)	8
ConfigDialogsDIL	9
Fibex Editor	10
Resilience (aspect)	11
Frame Logger	12
Node Simulation	13
DIL_CAN	14
DIL_MCNet	15
Generic Message Window	16
Tx Window	17
Signal Watch Window	18
Replay Window	19

Module name	Module No.
Filter	20
Common Class	21
Signal Graph	22
DIL_Interface	23
Bus Statistics	24
Installation	25
DIL_J1939	26
Test Automation	27

Definitions, Acronyms and Abbreviations

CAN	Controller Area Network
CSV	Comma Separated Variable
RA	Requirement Analysis
RS	Requirement Specification

References

No.	Title	Version	Author	Date	Source/ Location
1	Design Patterns: Elements of Reusable Object-Oriented software	-	Gamma, Helm, Johnson & Vlissides	August, 1994	Paperback edition
2	Architectural Blueprints - The "4+1" View Model of Software Architecture	-	Philippe Kruchten	November, 1995	Paper published in IEEE Software 12 (6)
3	Programming Applications for Microsoft Windows	4th edition	Jeffrey Richter	-	Microsoft press

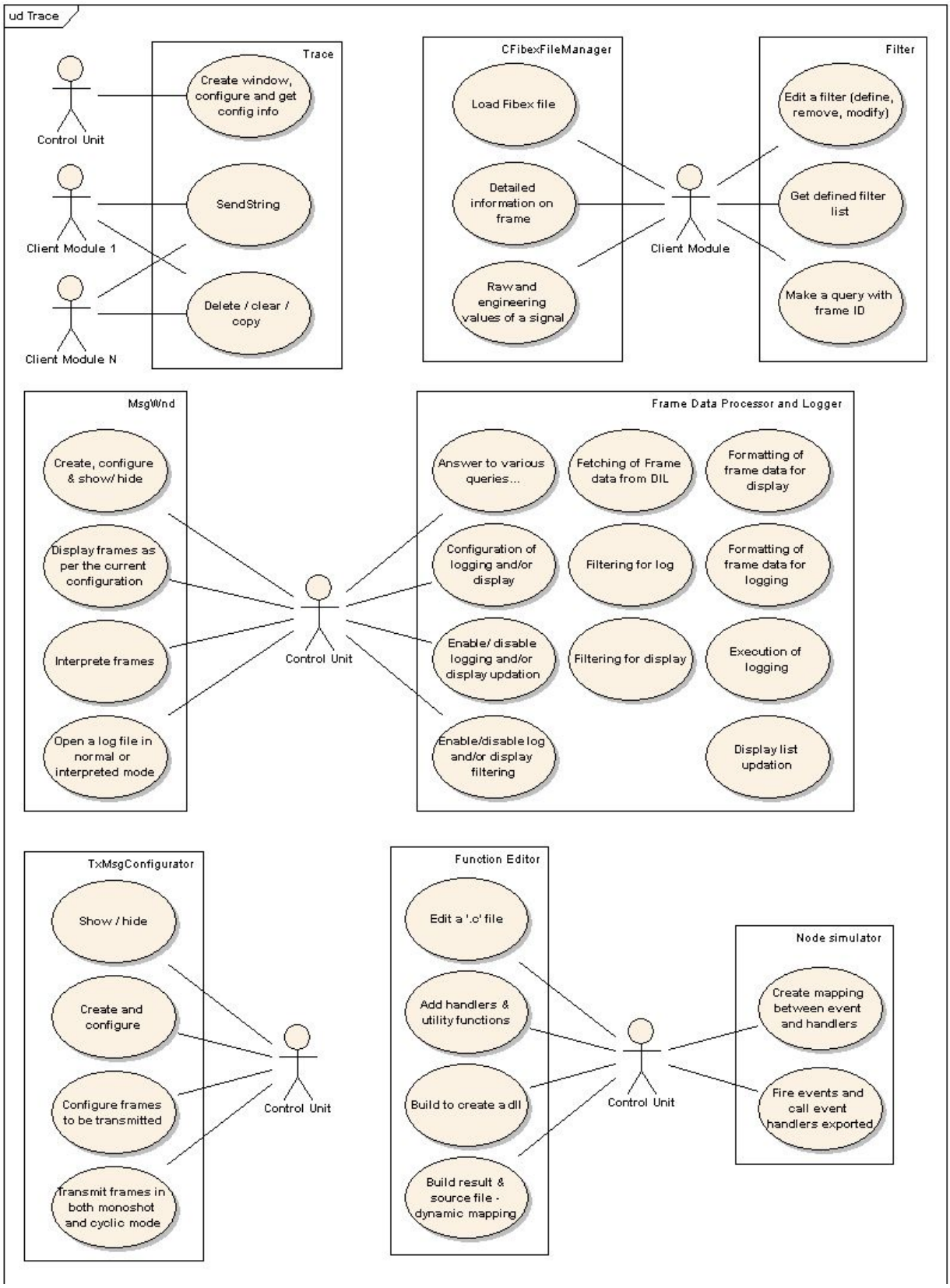
Software Requirements

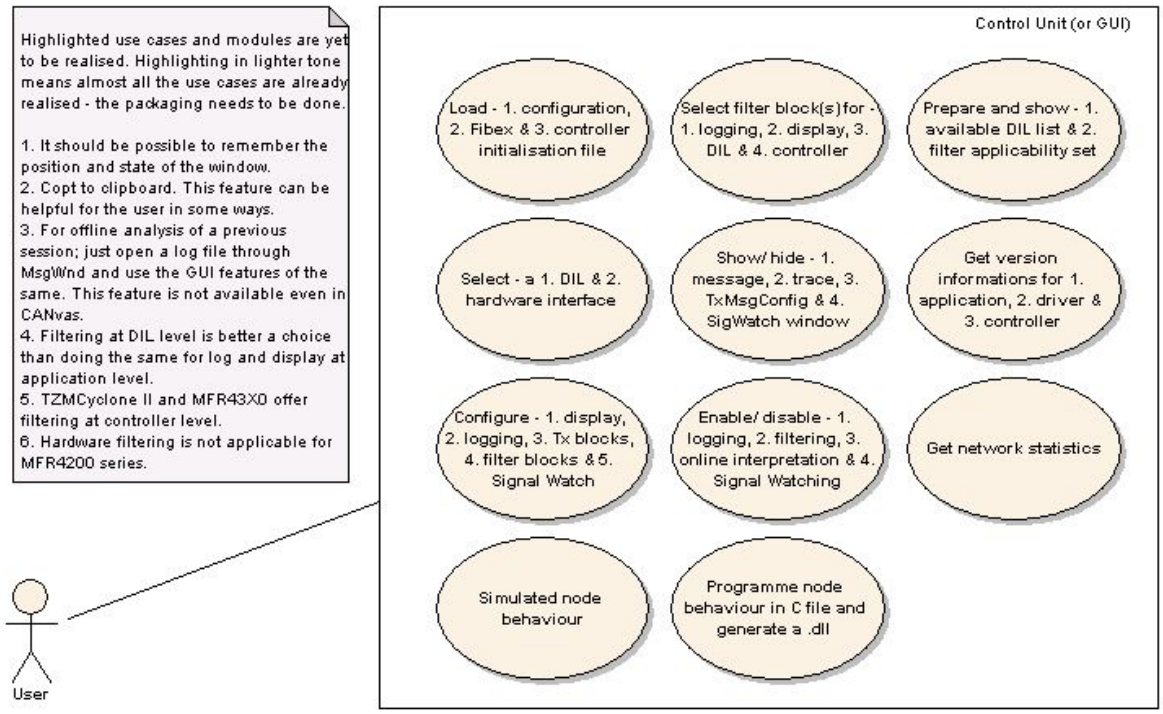
Operational Requirements

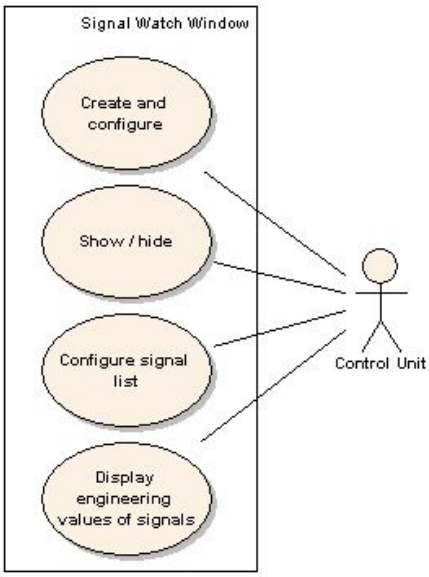
- Background information: Objective of development of the application suite BUSMASTER is to bring seamless support to any vehicle bus standard, all occurring under the umbrella of "automotive bus monitoring and node simulation tool" where the variability factor is the target automotive bus. From feature and functionality viewpoint, there are substantial amount of generalness to consider them to be of the same family. Towards that goal, the features and functionalities including the modules realizing them have been identified and analyzed from feasibility viewpoint, till the crystallization of the requirements into tagged statements.
- This means, there shall be a base version supporting the most widely used CAN standard, and easily extendable at run time with the addition of components for any other bus e.g., FlexRay, TTCAN, LIN; other CAN higher layer protocols like J1939, ISO-TP etc.
- The obvious addendum in the pool of development ideas is reusability, not only in terms of artifacts but also in terms of framework, algorithms & process. Hence it has been decided to employ the family engineering approach of product line engineering, which will extract the reuse possibility to the maximum possible extent using ideas of "commonality and predicted variability".
- Target environment: No module or component has been written targeting a particular windows environment and hence the product suite is expected to run on Windows 2k/XP/Vista/7 without any problem. However, running on XP is guaranteed for the time being. Being inline to the organization’s policy, we assume no responsibility for Win2K.
- User characteristics: This is an engineering tool targeting an automotive bus because of which the primarily intended users are the ECU developers, system engineers and testers.
- Assumptions and Dependencies of this Software on other systems: No such occurs till now.

Functional Requirements

Below are the use case diagrams:

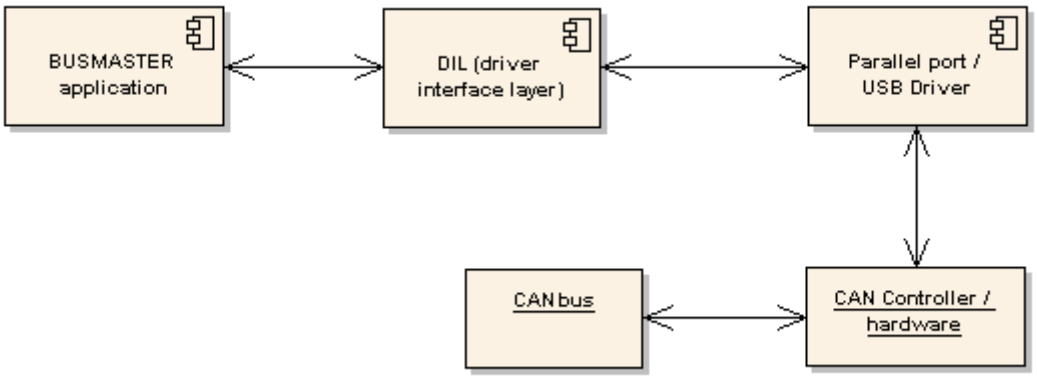

ud Trace
Trace
Control Unit
Create window, configure and get config info
Client Module 1
SendString
Client Module N
Delete / clear / copy
CFibexFileManager
Load Fibex file
Detailed information on frame
Raw and engineering values of a signal
Client Module
Filter
Edit a filter (define, remove, modify)
Get defined filter list
Make a query with frame ID
MsgWnd
Create, configure & show/ hide
Display frames as per the current configuration
Interprete frames
Open a log file in normal or interpreted mode
Control Unit
Frame Data Processor and Logger
Answer to various queries...
Fetching of Frame data from DIL
Formatting of frame data for display
Configuration of logging and/or display
Filtering for log
Formatting of frame data for logging
Enable/ disable logging and/or display updation
Filtering for display
Execution of logging
Enable/disable log and/or display filtering
Display list updation
TxMsgConfigurator
Show / hide
Create and configure
Configure frames to be transmitted
Transmit frames in both monoshot and cyclic mode
Control Unit
Function Editor
Edit a '.c' file
Add handlers & utility functions
Build to create a dll
Build result & source file - dynamic mapping
Control Unit
Node simulator
Create mapping between event and handlers
Fire events and call event handlers exported





DIL Interface

DIL is abbreviation for driver interface layer. Here the idea is to bring in a necessary abstraction between the application and the driver including the user interfaces for the controller configuration. Additionally in absence of the real driver, this contains the driver simulation. Below goes a component diagram of DIL usage:



Hence the objective of DIL may be outlined in the following way -

- Easy interfacing to stub / any hardware without any change in the DIL interface. So the API set shall be prototyped in such a way to support easy extendibility for any controller for a bus.
- Minimal change in DIL for addition of any new hardware support. “Change” mostly comprises of sheer code addition. The final goal is to realize “plug and play” feature.
- Optimum usage of resource. This involves minimum increase in code and minimum occupation of processor time. The later may be realized by minimizing redundant codes (e.g., if … else statements).
- The first two points needs a fair amount of general streamlining of the API set of drivers of automotive bus hardware.
- Configuration procedures of each of the supporting controllers are assumed to be different from each other. So there shall be UI specific to each one of them. Citing example from CAN bus controllers; configuration of ICS neoVI is different from PCAN USB hardware. This means for addition of a new hardware support, the application also needs to add a new UI.
- In view of the above – the change in application code may be prevented if the UI related code is shifted to DIL. In this approach application performs a query to receive list of all the controllers been supported. On selection of a controller from the list, DIL shall be instructed to return handle of a configuration dialog box (or any other UI). The entire hardware configuration related code shall remain inside the UI object and hence the application only calls a few functions from the API set.

Clearly, the services of DIL can be categorized in two groups, one bus the general type of services whereas the other one is of bus specific type. The tagged requirements of the general type are tabulated below:

			CAN	**FlexRay**	**J1939**
23		DIL Interface			
[RS_23_01]	Feature	Getter for the DIL list (number of driver interfaces)	X	X	-
[RS_23_02]	Feature	Selecting a driver from the DIL list	X	X	-
[RS_23_03]	Feature	Getter for the presently selected driver	X	X	-
[RS_23_04]	Feature	Registration of a client to simulate a node	X	X	X
[RS_23_05]	Feature	Unregistering of an existing client	X	X	X

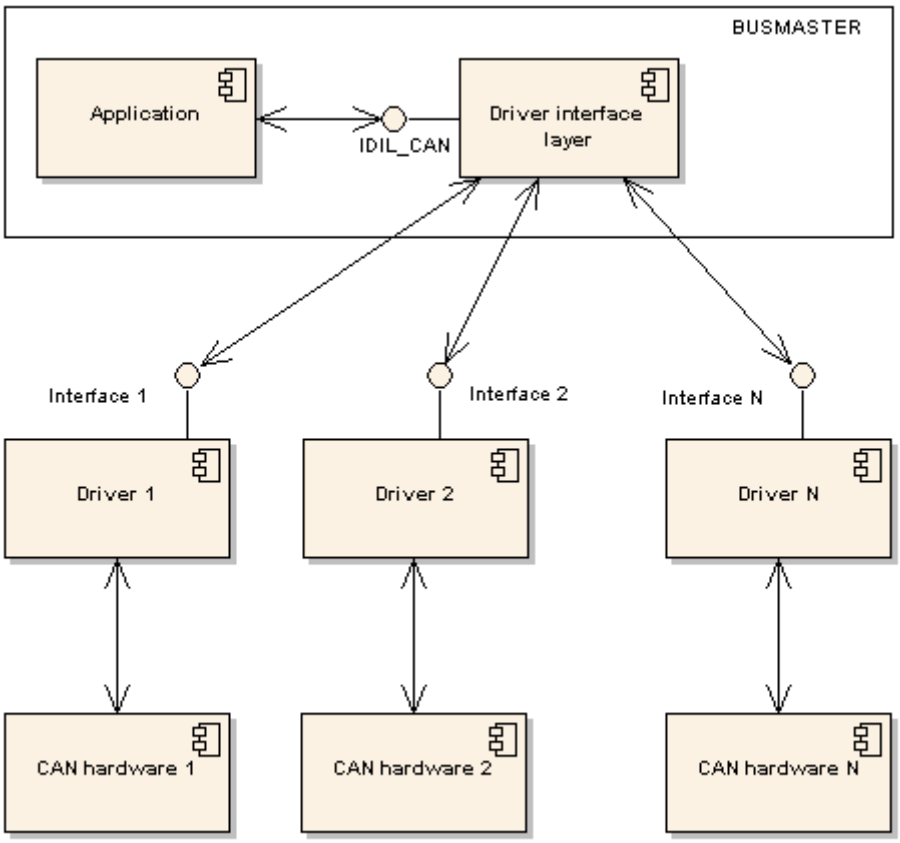
			CAN	**FlexRay**	**J1939**
[RS_23_06]	Feature	Addition of a message repository buffer of an existing client	X	X	X
[RS_23_07]	Feature	Deletion of a message repository buffer of an existing client	X	X	X
[RS_23_08]	Feature	Carry out initialization operations	X	X	X
[RS_23_09]	Feature	Carry out closure operations	X	X	X
[RS_23_10]	Feature	Getter for the time mode mapping (usually the 64-bit time stamp by the driver)	X	X	X
[RS_23_11]	Feature	Listing of the controllers for the current driver	X	X	-
[RS_23_12]	Feature	Selection of a controller from the hardware interface list	X	X	-
[RS_23_13]	Feature	Deselection of the presently selected controller	X	X	-
[RS_23_14]	Feature	Display the configuration dialog box of the present controller	X	X	-
[RS_23_15]	Feature	Setting of the configuration data for the present controller	X	X	-
[RS_23_16]	Feature	Start the presently selected controller (or connect)	X	X	X
[RS_23_17]	Feature	Stop the presently selected controller (or disconnect)	X	X	X

			CAN	FlexRay	J1939
[RS_23_18]	Feature	Reset the presently selected controller	X	X	-
[RS_23_19]	Feature	Transmit a frame	X	X	X
[RS_23_20]	Feature	Getter for the version information of the DIL for the present bus	X	X	X
[RS_23_21]	Feature	In case of any error, a function returns the associated string of the last error	X	X	X
[RS_23_22]	Feature	To set pass filters at hardware level.	X	X	X
[RS_23_23]	Feature	To set stop filters at hardware level.	X	X	X
[RS_23_24]	Feature	Getter for controller status by callback mechanism	X	X	X

To be noted that for J1939, some of the requirements directly don't hold good. The reason is that J1939 is based on CAN and hence they still hold good, albeit indirectly.

DIL CAN

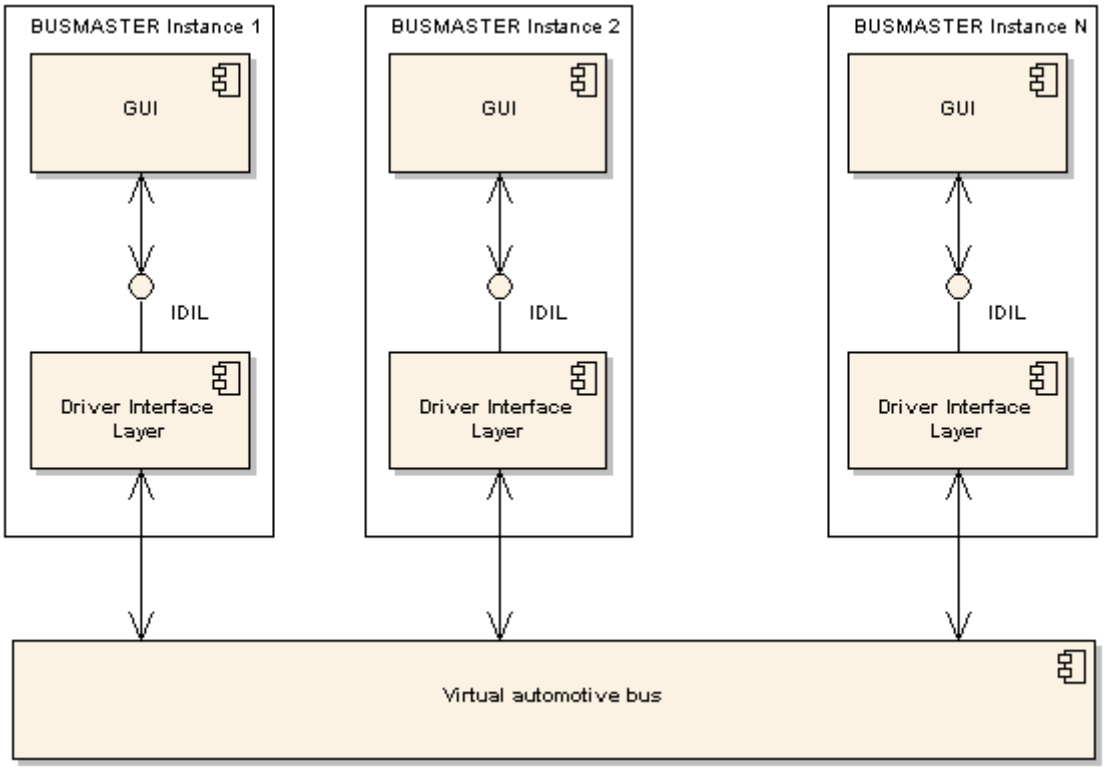
The requirement set of this derives from the general DIL characteristics. The diagram below indicates DIL_CAN scope.



Simulation Engine (Stub) component

Simulation engine realizes a virtual bus in the workstation making it possible for any instance of the application to act as a node and communicate via this communication channel. By design this should be able to simulate any bus from the viewpoint of data frame. Also, the same instance of server can cater to any number of buses with any number of nodes simultaneously.

Below diagram presents the usage of the simulation engine component.



This clearly shows that any DIL can employ the service of the same simulation engine process at any time.

Tagged requirements:

- A bus communication channel will be simulated in the system [RS_05_01].
- Each instance of BUSMASTER shall act as a node on the bus [RS_05_02].
- When a node is connected to the bus, it will be able to send / receive messages [RS_05_03]. Else it won't be [RS_05_04].
- Communication takes place when there are multiple nodes connected [RS_05_05]. In other words it must be a multi-node cluster connected to the bus.
- When a node sends a message, all of the other nodes except the sender shall receive the message [RS_05_06].
- While the communication is ON, for an assembly of 3 or more nodes, any node may get disconnected from the bus without causing any interruption / impact to the ongoing communication [RS_05_07]. For two nodes, only the communication will be OFF [RS_05_08].
- While the communication is ON, any node may join the cluster without causing any interruption / impact to the ongoing communication [RS_05_09].
- In order to facilitate the above two points (6 & 7), simulation of the virtual communication channel shall be realized as a separate process [RS_05_10]. From now onwards this will be defined as simulation engine.
- When the last node exits (on other words, the last client process exits), simulation engine process also exits [RS_05_11].

Project Configuration library

Purpose of this library is to provide an API set for the saving and retrieval of project configuration related information. Targeted data store may be either file system or database. In case of the former, the accessing shall be direct whereas in case of the later the interfacing shall be via ODBC. Following are the RS for the API set:

Function to ~

1. Set the present data storage configuration [RS_06_01].
2. Get the presently selected data storage configuration [RS_06_02].
3. Perform data storage operation after selection. Options are OPEN, SAVE and CLOSE [RS_06_03].
4. Get total number of projects in the project table [RS_06_04].
5. Get project name list from the project table [RS_06_05].

Function that, given name of the project ~

1. And project data, adds a project entry in the project table or modifies an existing one [RS_06_06].
2. Deletes the project entry from the project table [RS_06_07].
3. Retrieves project data from the project table [RS_06_08].
4. And section name, add a section or modify an existing one in the section table of the project [RS_06_09].
5. And section name, deletes the section from the section table of the project [RS_06_10].
6. And section name, gets information of that particular section from the section table of the project [RS_06_11].
7. Receives total number of sections from the section table of the project [RS_06_12].
8. Retrieves list of all the section names from the section table of the project [RS_06_13].

Frame Logger

This generates log files for a monitoring session. A log file is human readable ASCII text file containing the fundamental or bare minimum information of the said monitoring session. A log file consists of the three sections namely:

- File header which presents the following information
  - <Application version information>
  - <Declaration of the start of the logging session>
  - <Start date and time>
  - <Numeric format>
  - <Time mode>
- Data section which contains the frame data that is specific to the current vehicle bus standard. Nevertheless, so far the following data fields are observed to be consistently present in all the vehicle bus hitherto considered.
  - <Time stamp>
  - <Rx/Tx status>
  - <Channel>
  - <Frame ID>
  - <Frame type>
  - <Data length code>
  - <Data bytes>
- File footer concludes the end of the logging session specifying the time and date of the end of the logging session.

Clearly, frame logger consists of two sub-modules; one serves the purpose configuration tool whereas the other actually carries out the logging process.

Configuration

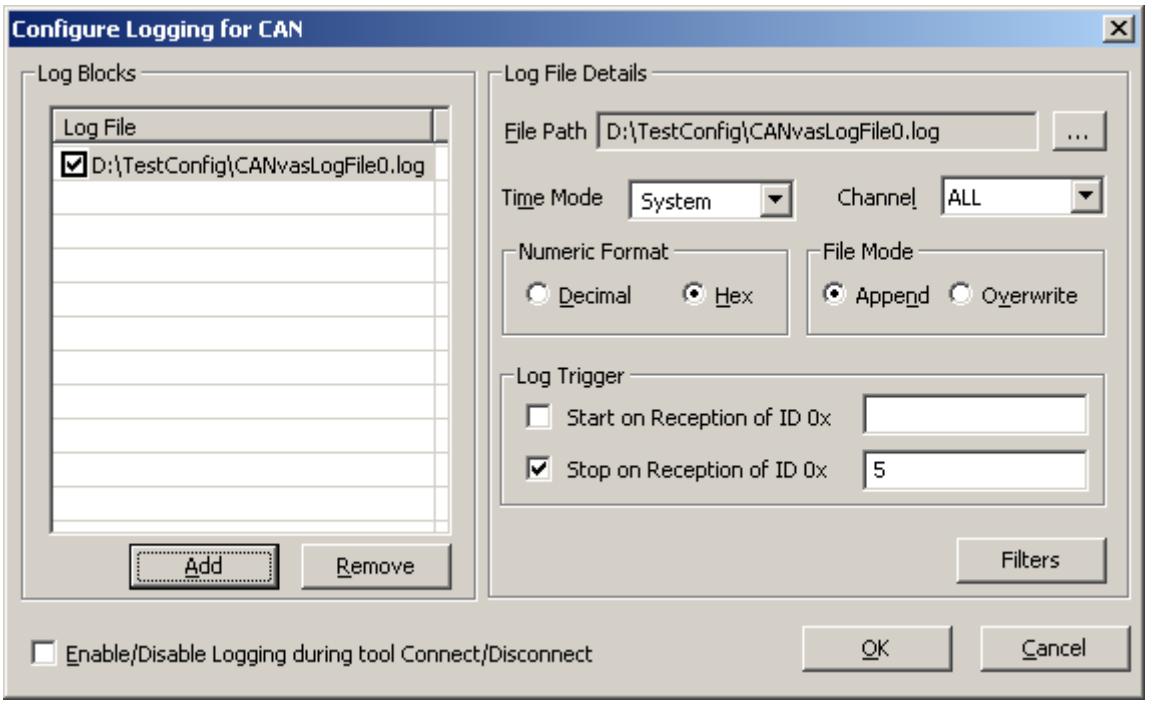
- Addition of a logging block
- Deletion of a logging block
- Enabling / disabling of a logging block

A logging session is defined in terms of a logging block. In order to start a logging session the user must define a logging block. So the first step is to define logging blocks. Clearly, the functionalities expected are -

A logging block is defined with a number of parameters namely,

- Absolute log file path
- Numeric mode (decimal / hexadecimal)
- Time mode (system / relative / absolute)
- File updating mode (overwrite / append)
- Log triggering parameters (optional). Logging can start at the reception of a particular frame. Similarly, it can stop based on the reception of a particular frame.
- Filters to be associated with the logging block

The user interface specification for CAN is presented below:



Additionally, the following operations shall be supported:

- Commit and rollback operations
- Saving and retrieving of configuration data

Log executor

This generates the log files as defined by the enabled logging blocks and provides all the controls necessary which are listed below:

- Enabling / disabling of the logging process
- Enabling / disabling of logging filter

If so needed, it shall be possible to use the logging module as a library without taking resource to the GUI. This means separation of front end from the back end operation and data.

The tagged requirement details are tabulated below:

			CAN	**FlexRay**	**J1939**
12		Frame Logger			
		Configuration			
[RS_12_01]	Feature	Addition of a logging block	X	X	X
[RS_12_02]	Feature	Deletion of a logging block	X	X	X
[RS_12_03]	Feature	Enabling of a logging block	X	X	X
[RS_12_04]	Feature	Disabling of a logging block	X	X	X
		Editing a logging block			
[RS_12_05]	Feature	- Updating the target file path	X	X	X
[RS_12_06]	Feature	- Selecting a time mode	X	X	X

			CAN	FlexRay	J1939
		(system / absolute / relative)			
[RS_12_07]	Feature	- Selecting a channel	X	X	X
[RS_12_08]	Feature	- Selecting the desired numeric mode (decimal / hexadecimal)	X	X	X
[RS_12_09]	Feature	- Selecting the file updation mode (append / overwrite)	X	X	X
		- Selecting trigger condition			
[RS_12_10]	Feature	- Start trigger on reception of a frame	X	X	X
[RS_12_11]	Feature	- Stop trigger on reception of a frame	X	X	X
[RS_12_12]	Feature	- Enable start trigger	X	X	X
[RS_12_13]	Feature	- Disable start trigger	X	X	X
[RS_12_14]	Feature	- Enable stop trigger	X	X	X
[RS_12_15]	Feature	- Disable stop trigger	X	X	X
[RS_12_16]	Feature	- Associating filters	X	X	X
[RS_12_17]	Feature	Saving of present configuration	X	X	X
[RS_12_18]	Feature	Loading of configuration	X	X	X
[RS_12_19]	Feature	Committing of the present updation	X	X	X
[RS_12_20]	Feature	Rolling back the present updation	X	X	X
		Logging execution			
[RS_12_21]	Feature	Enable logging filter	X	X	X
[RS_12_22]	Feature	Disable logging filter	X	X	X
[RS_12_23]	Feature	Start logging	X	X	X
[RS_12_24]	Feature	Stop logging	X	X	X

			CAN	FlexRay	J1939
[RS_12_25]	Design	Make logging module employable as a library	X	X	X

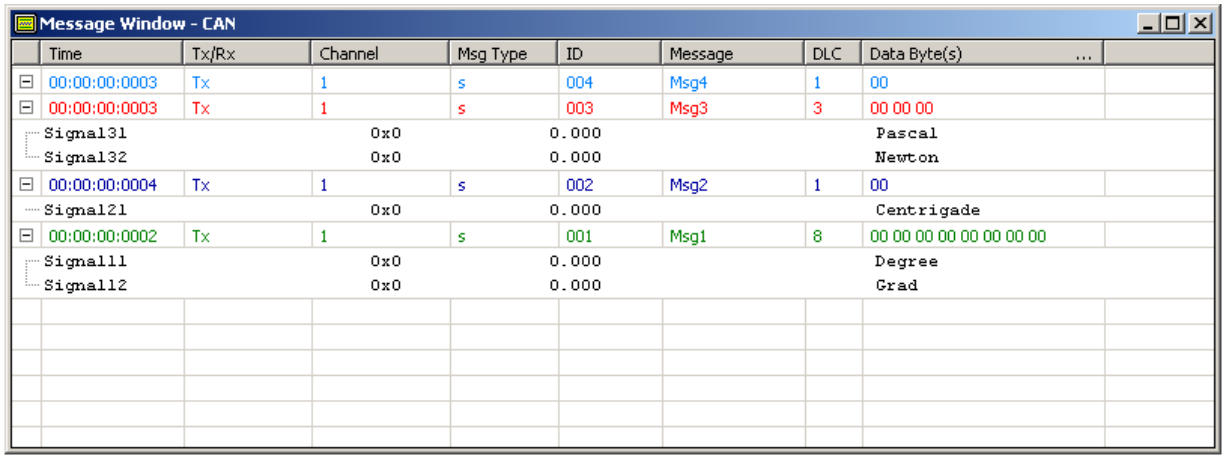
Message Window

It is the visual means of the bus data monitoring with dynamic listing of the frames. The window does the listing in a soft real time manner with the minimum possible time latency. Both data and error frames are accounted in the listing. Each raw corresponds to a frame entry whereas each of the columns to the data property like time stamp, receive status (Rx / Tx), frame identifier, channel, DLC, data bytes etc.

Clearly, visual presentation can be of two types, the first one is called append mode and the second one is called overwrite mode. In append mode frames appear in chronological order whereas for the later, the whole set of frames is grouped according to the frame identifier and the list contains only the latest entries of all the different frame identifiers. The apparently unintelligible data bytes needs to be presented in suitable way with all the signal names and values indicated and this is defined as online interpretation. This shall be possible only when the message window is associated with a database or a number of it. Two numeric modes are supported namely, decimal and hexadecimal. Three different time modes are possible. System time mode presents the time stamp in the local time. In absolute time format, time stamp is presented as the time elapsed since the application is connected to the channels. Relative time manifests differently in append and overwrite visual modes. In the former one, it is the time difference with respect to the last received frame. In the later one the time difference is with respect to the previous instance of the frame with the same id. During the execution it should be possible to select any such setting without losing of bus data. For better performance and control, the refresh rate shall be configurable. Also on the same token, there is an upper limit to the frame entries in either of the visual modes configurable at run time.

Clearly, the aforementioned paragraph discusses about the general characteristics of message window not hinting anything on the vehicle bus data. The columns are selected according to the specific bus data.

Below goes the UI specification of message window for CAN:



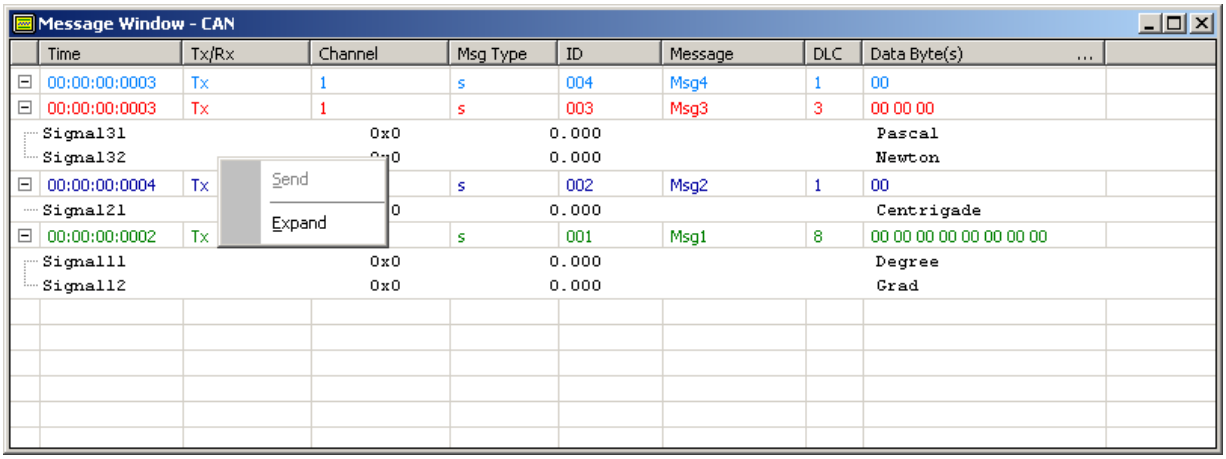
This example shows message window with overwrite visual mode, using hexadecimal numeric mode and with relative time stamp.

Clearly, in message overwrite mode there will be only one instance of a message with specific frame ID and direction on the message window. Subsequent messages that occur with the same values for ID and direction will overwrite the current message entry. In append mode new messages are appended at the end of the existing list of messages on the window even though that entry is already existing in the list.

The presentation is in interpreted mode. The message database can be either proprietary or a standard one (e.g., Fibex). From the loaded database are extracted the necessary frame / message and signal information and engineering values of all the signals using signal encoding technique are calculated. When message interpretation option is selected, the signals present in the frames should be displayed in the message window. In the message

window at the left hand side there shall be +/- option for each and every frame to expand / collapse the particular frame if that contains signals.

There are some miscellaneous features of message window available. For example – any frame entry can be selected and transmitted. The GUI specification depicting this property is shown below:



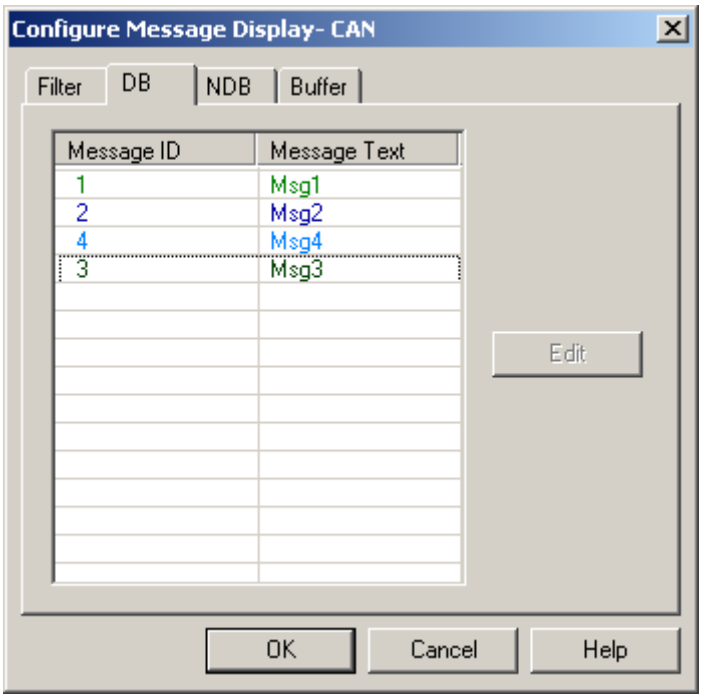
Message Window configuration

The different configurations aspects potentially related with message window are explained below:

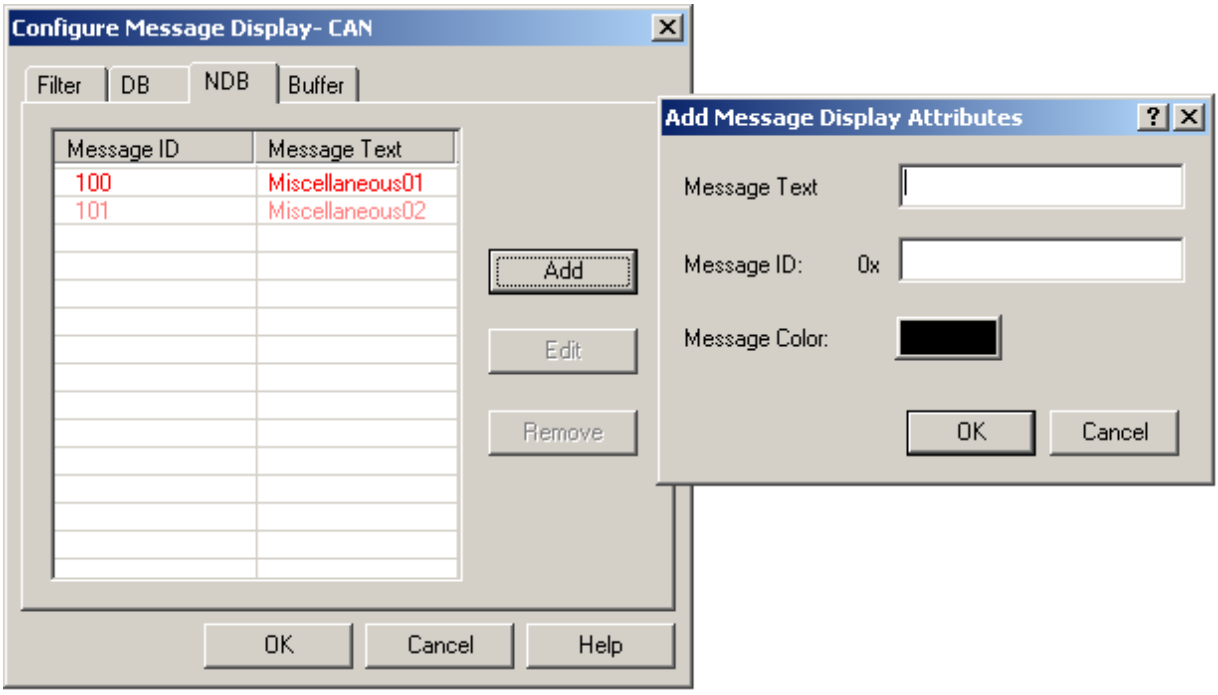
- Visual indication for defined frame entries - A particular frame can be made easily visually distinguishable from others with different colour. Both database and non-database message shall be configurable, albeit with a restriction in the configuration extent. For database frames the frame list is available and different colour can be associated with each one of them. There shall be a default colour (black).
- Visual indication for undefined frames - It shall be possible to define a frame with an ID, name and colour code. Again the default colour is black.
- Usage of filters – Filters can be applied for message window as well to boost up the performance. The defined filters shall be available from where the selected filters can be confirmed. Again each of the filters can be enabled / disabled.
- Buffer and refresh parameters – Characteristically it shall be possible to switch to any visual, numeric and time mode – both online and offline. This is possible because a message repository is maintained for each of the visual modes. The size of the list is configurable at run-time. Also, the refresh rate is configurable. Both of them render the user a better control over message window.

Message window is configurable through a property sheet where the property pages are logically linked with the aforementioned configuration aspects.

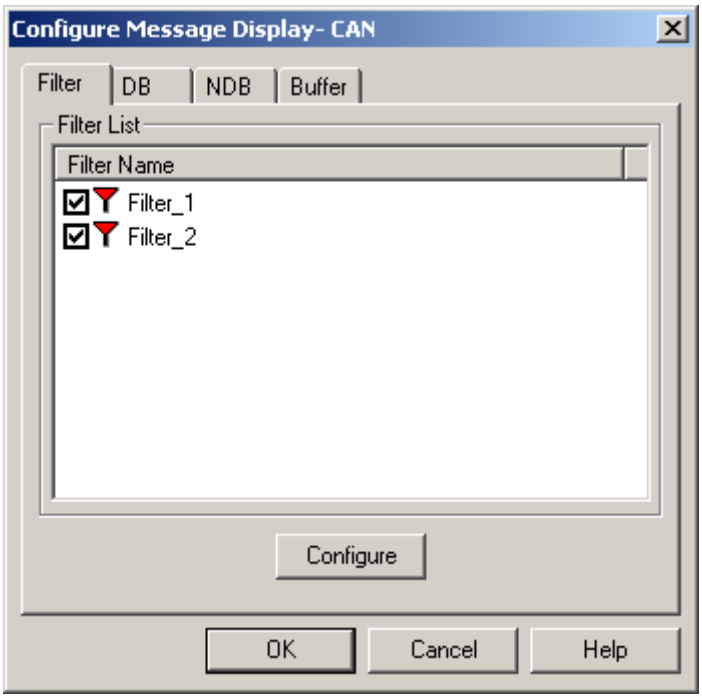
Database frame configuration UI specification:



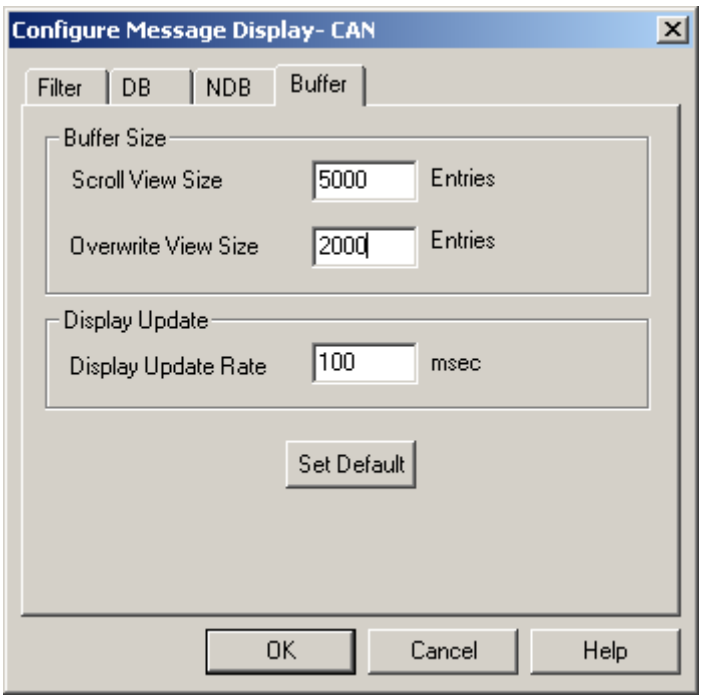
Non-database frame configuration UI specification:



Filter configuration UI specification:



Other configuration UI specification:



			CAN	FlexRay	J1939
16		Message Window			
	Definition	BUSMASTER is connected to the bus through one or more controllers. A controller can be	X	X	X

			CAN	FlexRay	J1939
		either a real or a virtual one. A communication channel of a controller can be perceived as a connection instance to a bus. Objective of message window for a particular bus is to list messages in all the related communication channels that the application is connected. The flag 'Tx' or 'Rx' ascribed for a message entry is done from the perspective of the channel.			
		Display modes			
[RS_16_01]	Feature	Two display modes namely, overwrite and append shall be supported.	X	X	X
[RS_16_02]	Definition	A message entry signature := <CAN id> + <type (standard | extended | RTR)> + <Rx | Tx> + <channel>	X	-	-
[RS_16_03]	Definition	A message entry signature := [<PGN Number> <Type (Request | data | broadcast)>]	-	-	X
[RS_16_04]	Definition	A message entry signature := <Rx | Tx> | <Frame ID> | <Cycle number> | <channel (A | B | AB)>	-	X	-
[RS_16_05]	Feature	In overwrite mode, an entry shall be overwritten by	X	X	X

			CAN	FlexRay	J1939
		the one with the same signature.			
[RS_16_06]	Feature	In append mode message entries shall be listed in chronological order.	X	X	X
[RS_16_07]	Feature	In append mode number of message entries shall be kept restricted and the number shall be configurable.	X	X	X
[RS_16_08]	Design	In overwrite mode, the condition of entry restriction shall not come into the picture. This is because of the fact that compared to the maximum allowable number of entries in append mode, number of entries in overwrite mode is expected to be just a fraction.	X	X	X
[RS_16_09]	Feature	In append mode when the list is full, the newest entry replaces the oldest one. As a result, the list scrolls up by one entry.	X	X	X
[RS_16_10]	Feature	It shall be possible to toggle between the two display modes without loss of data.	X	X	X
[RS_16_11]	Feature	The above shall be possible both online and offline.	X	X	X
		Time stamp types			
	Definition	Unit for a message			

			CAN	FlexRay	J1939
		time stamp is hundreds of microseconds. There are three different time modes. System time mode expresses the time stamp in local time. In append mode relative time stamp indicates time elapsed since the last message whereas in overwrites it indicates the time elapsed since message of same signature. In absolute time mode it is the time elapsed since the connection.			
[RS_16_12]	Feature	Support of system time stamp.	X	X	X
[RS_16_13]	Feature	Support of relative time stamp.	X	X	X
[RS_16_14]	Feature	Support of absolute time stamp.	X	-	X
[RS_16_15]	Feature	It shall be possible to transit from one time stamp mode to another without any loss of data.	X	X	X
[RS_16_16]	Feature	The above shall be possible both online and offline	X	X	X
		Numeric modes			
[RS_16_17]	Definition	Numeric mode dictates numeric representation of frame / CAN ID and data bytes.	X	X	X
[RS_16_18] [RS_16_19]	Feature	Shall support two numeric	X	X	X

			CAN	FlexRay	J1939
		modes namely, decimal and hexadecimal.			
[RS_16_20]	Feature	It shall be possible to toggle between the two numeric modes without loss of data	X	X	X
[RS_16_21]	Feature	The above shall be possible both online and offline Column configurations	X	X	X
[RS_16_22]	Definition	A section of an entry in a bus occur under a particular columns.	X	X	X
[RS_16_23]	Definition	Sections of a message entry := { <time stamp>, <CAN id>, <Rx | Tx>, <description>, <type>, <channel>, <DLC (0 - 8)>, <data bytes> }	X	-	-
[RS_16_24]	Definition	Sections of a message entry := { <time stamp>, <channel>, <CAN id>, <PGN>, <PGN name>, <Type>, <Src>, <Dest>, <Priority>, <Tx/ Rx>, <DLC>, <data bytes> }	-	-	X
[RS_16_25]	Definition	Sections of a message entry := { <time stamp>, <frame id>, <Rx | Tx>, <description>, <cycle number>, <channel (A | B | AB)>, <DLC>, <data bytes> }	-	X	-
[RS_16_26]	Definition	A selected set of columns shall have sorting	X	X	X

			CAN	FlexRay	J1939
		property. Also, this feature shall be available only in offline mode.			
[RS_16_27]	Feature	For a CAN message window, the following columns shall feature sorting property: 1. <time stamp>, 2. <CAN id> and 3. <channel> Next priority columns in decreasing order of importance may be 4. type, 5. direction and 6. description.	X	-	-
[RS_16_28]	Feature	For a FlexRay message window, the following columns shall feature sorting property: 1. <time stamp>, 2. <frame id> and 3. <cycle number> Next priority columns in decreasing order of importance may be 4. channel, 5. direction and 6. description.	-	X	-
[RS_16_29]	Feature	For a J1939 message window, the following columns shall feature sorting property: 1. <time stamp>, 2. <channel>, 3. <CAN ID>, 4. <PGN>, 5. <Src> and 6. <Dest> Next priority columns in	-	-	X

			CAN	FlexRay	J1939
		decreasing order of importance may be 4. CAN id, 5. direction, 6. type and 7. message name / code.			
[RS_16_30]	Feature	Sorting shall be possible both in overwrite and append modes.	X	X	X
[RS_16_31] [RS_16_32]	Feature	It shall be possible to show / hide any column.	X	X	X
[RS_16_33]	Feature	It shall be possible to reorder message columns (by dragging).	X	X	X
[RS_16_34]	Feature	It shall be possible to revert to the default setting of column order.	X	X	X
		Message interpretation			
[RS_16_35]	Design	Message interpretation means apparently incomprehensible raw data bytes of a message entry are translated into and conveyed in human-readable signal values. Finally it is the signal set that are of prime concern for the user. Presentation of a signal involves the following parameters: 1. signal name, 2. raw value, 3. engineering value and 4. unit. Interpreting a message means presenting its	X	X	X

			CAN	FlexRay	J1939
		signal using the four parameters.			
[RS_16_36]	Feature	It shall be possible to load a message database (Fibex or proprietary) and extract out the information necessary for message interpretation using signaling encoding technique.	X	X	X
[RS_16_37]	Feature	When this is ON, defined signals are interpreted with the four parameters.	X	X	X
[RS_16_38]	Feature	Interpretation shall be possible only in overwrite mode.	X	X	X
[RS_16_39]	Feature	Interpretation can be toggled between active and inactive states.	X	X	X
[RS_16_40]	Design	Hence the following states of an entry result - 1. NON_INTERPRETABLE, 2. INTERPRETABLE and 3. INTERPRETING	X	X	X
[RS_16_41]	Design	Also, the following message window states are defined - 1. APPEND, 2. OVERWRITE and 3. OVERWRITE_INTERPRET	X	X	X
[RS_16_42]	Feature	Every interpretable message entry shall have a '+' sign against it to indicate presence of interpreted signals.	X	X	X

			CAN	**FlexRay**	**J1939**
[RS_16_43]	Feature	Clicking on the '+' sign shall be taken as message entry expansion action and the entry shall be expanded into the full set of interpreted signals and the sign shall change into '-'.	X	X	X
[RS_16_44]	Feature	Clicking on the '-' sign shall be taken as message entry collapsing action and the entry shall be collapsed and the sign shall change back into '+'.	X	X	X
[RS_16_45]	Feature	It shall be possible to transit to any message window state without any loss of data.	X	X	X
[RS_16_46]	Feature	The above shall be possible both online and offline. Configuration retention	X	X	X
[RS_16_47]	Design	Message window configuration data := { 1. numerical mode, 2. display mode, 3. time mode, 4. displayed column list, 5. individual column width (in proportion with respect to overall window width), 6. displayed column order, 7. window dimension, 8. window state, 9. active interpretation mode, 10. interpretation	X	X	X

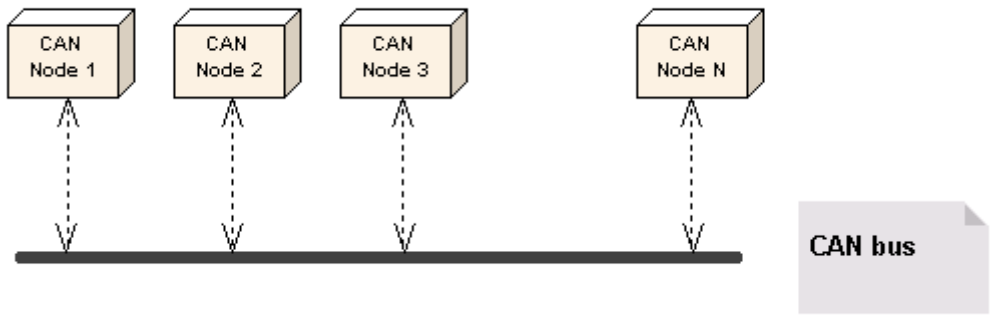
			CAN	**FlexRay**	**J1939**
		data and 11. expanded message entries }.			
[RS_16_48]	Feature	It shall be possible to save the present configuration in a byte stream.	X	X	X
[RS_16_49]	Feature	It shall be possible to load a configuration from a byte stream.	X	X	X
[RS_16_50]	Feature	In case of absence of configuration data, default setting shall be assumed. Utilities and others	X	X	X
[RS_16_51]	Performance	Display updation shall be flicker-free	X	X	X
[RS_16_52]	Feature	Shall be possible to to clear message window, both during online and offline.	X	X	X
[RS_16_53]	Design	Message window exactly reflects the message data in the buffer (but obviously not the representation). Both append and overwrite display modes have their own data buffer.	X	X	X
[RS_16_54]	Design	Hence clearing the message window means clearing the buffers as well.	X	X	X
[RS_16_55]	Feature	It shall be possible to specify frequency / refresh rate of message window	X	X	X

			CAN	FlexRay	J1939
[RS_16_56]	Feature	It shall be possible to specify size of the append buffer and this shall have lower and upper limits.	X	X	X
[RS_16_57]	Design	Multiple message windows even for a single bus shall be possible.	X	X	X

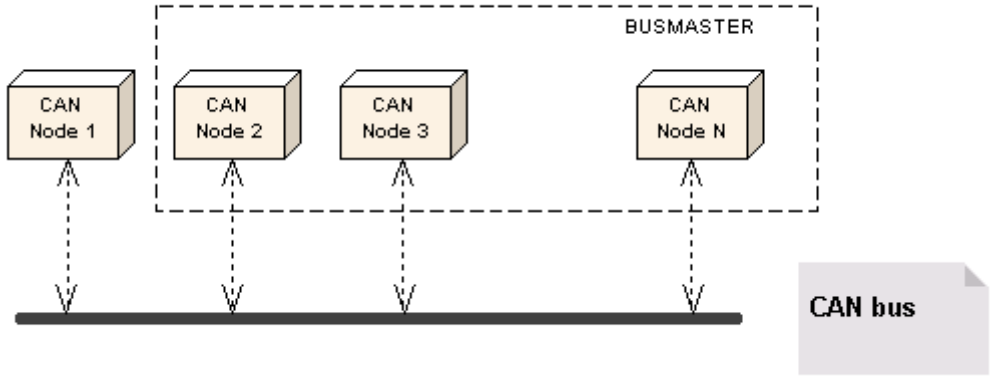
Node Simulation

This addresses the programming and configuration aspect of the node simulation.

At the outset the typical node assembly scenario may be recalled to consider the role of such application in such a scenario. Below goes a schematic view of the node ensemble:



Node simulation feature makes it possible for the application to simulate one or multiple node characteristics which may be depicted by the following schematic view.



Typically, in BUSMASTER parlance and viewpoint, a simulated system consists of a few nodes and the CAN bus is realized by a number of simulated systems. Hence,

A CAN bus (full / partial) = {Simulated system 1, Simulated system 2, …, Simulated system N }

A simulated system = {Node 1, Node 2, …, Node N }

A bus node is realized in the desired way by programming the set of handlers such a node is associated with. Following this, below handlers may be edited:

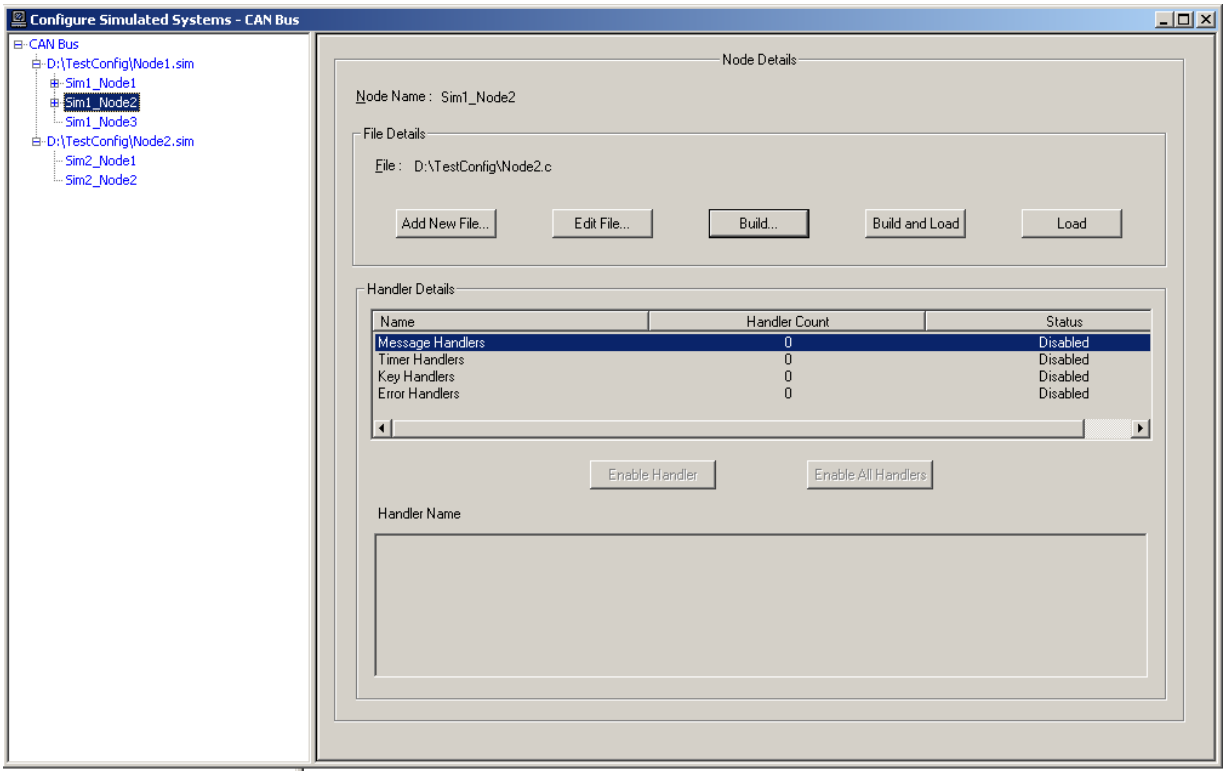
- Data message handlers: Message handling is based on certain directives on association which can be the following:
  1. On a list of discrete frame identifiers
  2. On a range of frame identifiers
  3. All received frames
- Timer handlers: Here the triggering is based on timer in MS.
- Key handlers: This basically corresponds to a user event which is realized as a key press.
- Error handler: This focuses on the bus error conditions and hence bus specific.
- Connect / disconnect handler: These handlers are called when the node joins the bus and is disconnected from it.

From deployment viewpoint, a node is realized by the application as a DLL. A node is programmed with C language. Hence, connect / disconnect handler in this scenario may thought to be logically associated with loading / unloading events of the DLL.

So the handlers are added as C global functions. Besides, addition of utility functions and global variables are supported for obvious programming reason. To carry out any domain specific operation (e.g., transmission of a frame, start / stop logging, tracing any message, going online / offline etc) the source programme requires a suitable API. Moreover, the framework should also make it possible for the make the loaded database (messages and signals defined) available for usage. The function editor framework provides these services. The former one is available through an API of the application and the later one is made accessible through a header file generated on the fly as a result of the database loading.

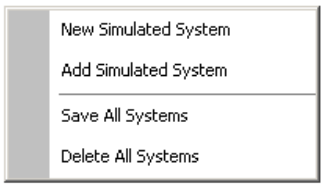
The final phase of node programming is the building where the C file is compiled and a DLL is built. This library is appreciated as the node characteristics deployed. The building result is available. In case of any error, it shall be possible to navigate to the erroneous source line.

Below presented the user interface specifications with due explanation:

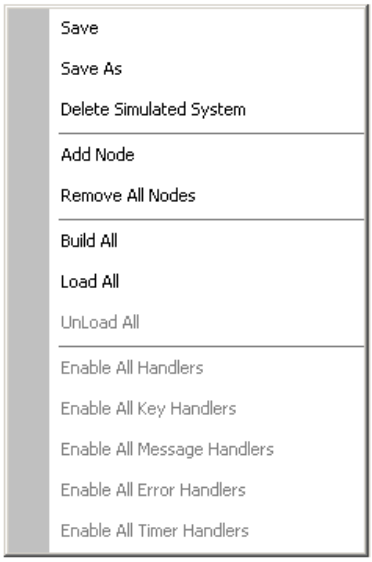


The above diagram shows defining a simulated system to the point of a node configuration. The GUI clearly shall reflect the whole assembly of the simulated systems and the nodes under each of them. It shall be possible to edit

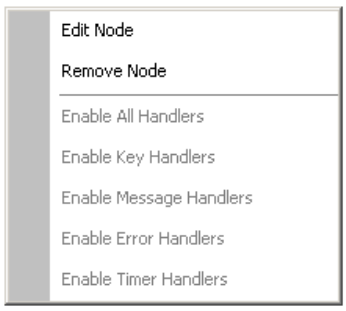
each of them with the related fundamental functionalities readily available through a number of popup menus. The below diagram shows the functionalities segregated into different popup menus resulted from the right click on bus, a simulated system and a node respectively.



Defining a simulated system



Working with a simulated system

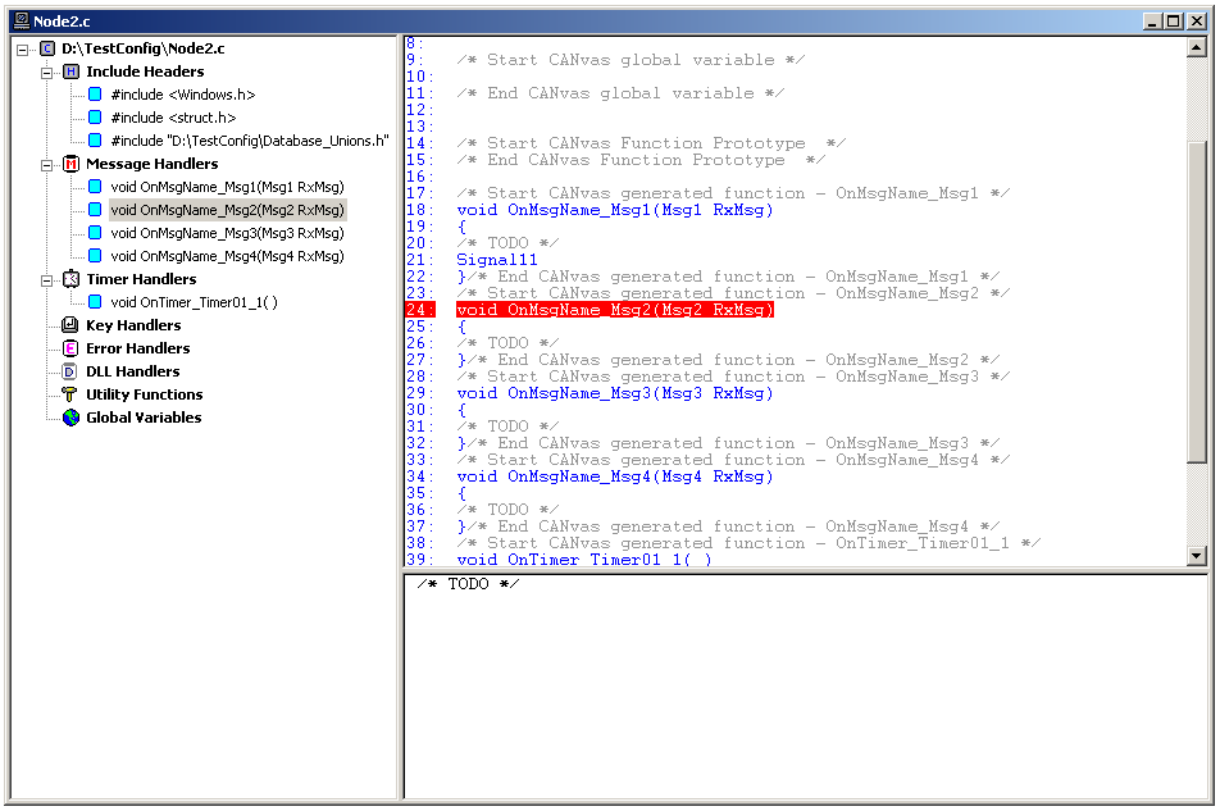


Working with a node

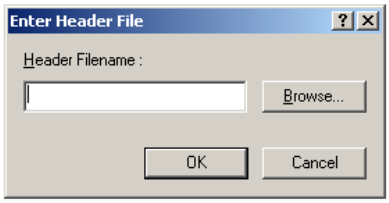
Programming a node involves associating the same with a C file. Function editor shall provide a programming framework. The various functionalities shall be provided by the framework:

- Addition / deletion / editing of include headers
- Addition / deletion / editing of message handlers
- Addition / deletion / editing of timer handlers
- Addition / deletion / editing of error handlers
- Addition / deletion / editing of key handlers
- Addition / deletion / editing of DLL handlers
- Addition / deletion / editing of utility functions
- While working in a handler or function body, listing of the following ~
    - API set of the application
    - Available frames of the loaded database
    - Available signals of the loaded database

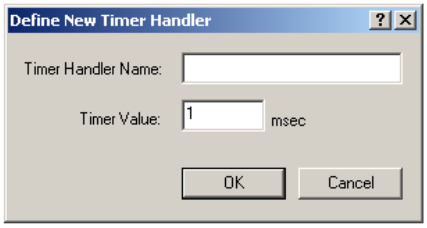
Below is presented the GUI specification of function editor for a source file.



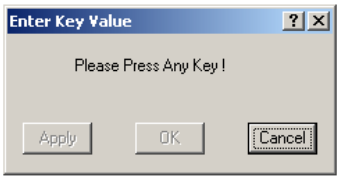
The GUIs involved in editing all the tree nodes under the source file are shown below:



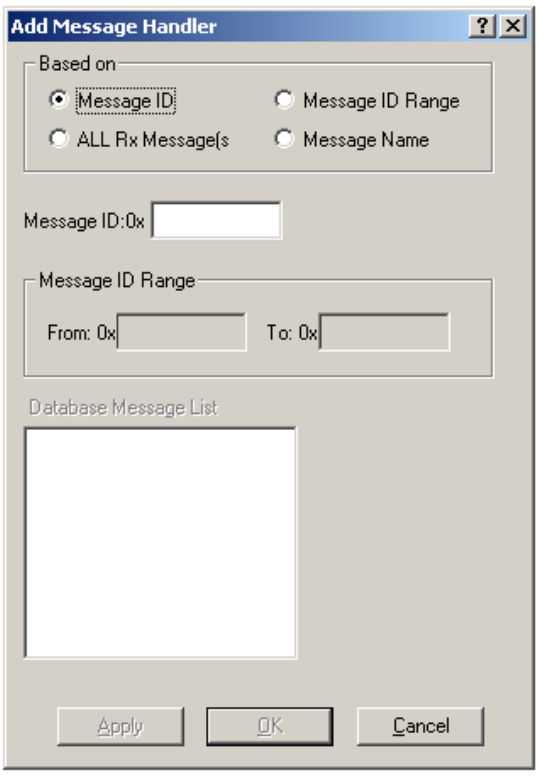
Header file



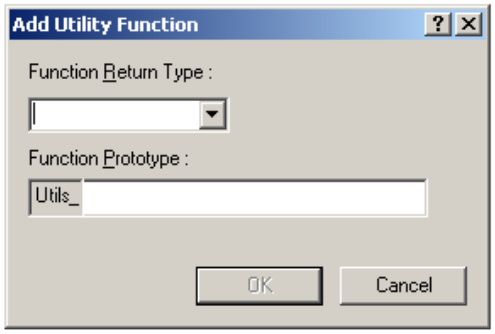
Timer handler



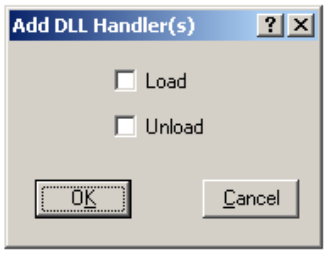
Key event



Message handler



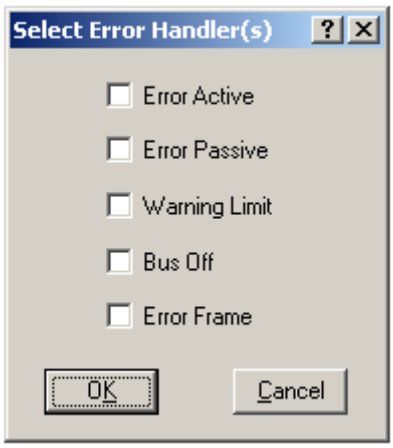
Utility function



DLL handler

The general popup menu resulted with a right click on a node

The error handler has a direct relation to the bus. Below is shown the one for CAN:



The tagged requirement details are provided below:

- User shall be able to select "Simulated Systems" window by clicking "Configure ->Simulated System(s)" menu
- On right clicking the root of the tree "CAN Bus" in the left pane, a pop up menu shall appear. It shall have options: "New Simulated System" , "Add Simulated System", "Save All systems" and "Delete All Systems". On right clicking any simulated system in the left pane, ref fig 2.3.4 a pop up menu shall appear having options: "Save", "Save As", "Delete Simulated System", "Add Node", "Remove All Nodes", " Build All" , "Load All" , "Unload All" , "Enable / Disable All Handlers", "Enable / Disable All Key Handlers", "Enable / Disable All Message Handlers", "Enable / Disable All Error Handlers", "Enable / Disable All Timer Handlers". [RS_13_1]
- If there are no simulated systems under the "CAN Bus" the option "Delete All Systems" shall be disabled. [RS_13_2]

- A new simulated system shall be added by right clicking the root of the tree in the left pane, "CAN Bus" and selecting "New Simulated System" from the pop up menu. A browser shall appear to enter the name of the new simulated system. [RS_13_3] ref Fig 2.3.3
- On giving the new simulated system name and selecting the Save button in the browser, the name of the new simulated system shall be shown under the "CAN Bus" [RS_13_3.1] and the configuration file shall be saved under the specified directory. [RS_13_4]
- The path of the new simulated system's configuration file shall be included in the main configuration file. [RS_13_5]
- On giving the name of the new simulated system as an existing one which is not connected to the "CAN Bus", a message box shall pop up saying "The file already exits. Do you want to replace it?". [RS_13_6]
- On selecting "Yes", the old simulated system configuration shall be replaced by the new one only if the simulated system is not present in the tree view of the simulated system window. [RS_13_7]
- On selecting "No", the focus shall be on the new simulated system file name to enable the user to enter the file name again. [RS_13_8]
- If the simulated system is already included in the tree view then a message box shall pop up saying "The simulated system is already present" and replacing the file shall not take place. [RS_13_9]
- If the simulated system is not present in the tree view of the simulated system window the selected simulated system configuration file's path shall be updated to the new one in the main configuration file.[RS_13_11]
- The left pane , tree view of the "Simulated system" window shall also be updated with the new simulated system details.[RS_13_12]
- On selecting "No" there shall be no change in the "Simulated System" window and the action shall be cancelled.[RS_13_13]
- An existing simulated system shall be added by right clicking the root of the tree in the left pane, "CAN Bus" and selecting "Add Simulated System" from the pop up menu. A browser shall appear with all the existing simulated system's names, to select the name of the existing simulated system. If the simulated system is already present under the "Can Bus" in the tree of the simulated system window then it shall not added and it shall be indicated to the user. [RS_13_14]
- On selecting an existing simulated system name it shall be added under the "CAN Bus". [RS_13_15]
- If the selected simulated system has the same C file or DLL which is already present under any simulated systems present under the "CAN bus" in the tree view of the simulated system window , it shall not be added. [RS_13_10]
- The configuration of the existing simulated system shall be included in the main configuration file. [RS_13_16]
- On entering a non existing simulated system name in the browser a message box shall appear saying "File not found. Please check if the correct file name was entered". [RS_13_17]
- On selecting "Delete All Systems" option on right clicking the root of the tree "CAN Bus" in the left pane, a confirmation message is thrown whether the user is sure to delete them or not if no DLL under any simulated system is loaded. [RS_13_18]
- When the user selects "Yes" in the confirmation message box , if any DLL under any simulated system is loaded and "Delete All Systems" is selected a message box shall pop up saying "Unload all DLLs and try again!" to indicate the user that the simulated system cannot be deleted without unloading the DLL under it. [RS_13_19].
- The configuration paths of the simulated systems under the "CAN Bus" are deleted from the main configuration file if it is saved by selecting File->Configuration->Save. [RS_13_20]
- The simulated system details are deleted from the left pane under the "CAN Bus" and the right pane shall show an empty frame. [RS_13_21]
- On selecting "No" in the confirmation message box there shall be no change in the "Simulated Systems" window and the main configuration file.[RS_13_22]
- The configuration of the simulated system under the "CAN Bus" shall be saved by right clicking on that particular simulated system and selecting "Save" option in the menu. [RS_13_23] ref Fig 2.3.4
- The configuration of the simulated system under the "CAN Bus" shall be saved as a different configuration by right clicking on that particular simulated system and selecting "Save As" option in the menu. [RS_13_24]
- A new node shall be added by right clicking the simulated system under which it is to be added, in the left pane and selecting "Add Node" from the pop up menu. A Node details dialog box shall appear in which the user shall specify its name and load the existing DLL by selecting the browse button [RS_13_25]

- The browser shall show the list of the DLL present. When the user specifies a DLL which is already loaded / present under any other simulated system under the "CAN Bus" , a message box shall pop up saying " Duplicate DLL found!". [RS_13_26]
- On entering a non existing DLL name in the browser a message box shall appear saying "DLL not found. Please check if the correct DLL name was entered". [RS_13_27]
- If the new Node is given an existing Node name under that particular simulated system and "OK" is selected, a dialog box saying "Duplicate Node name found" shall appear. [RS_13_28]
- The included DLL shall be shown under the node where it is included in the left pane under the "CAN Bus" in normal font indicating that the DLL is unloaded by default. [RS_13_29]
- By default the DLL under the Node shall be unloaded and the Node details frame in the right pane of the "Simulated system" window shall show "Load" button to enable the user to load the DLL if required. [RS_13_30]
- On selecting the "Load" button in the right pane of the "Simulated system" window" the name of the DLL under that particular node shall be made BOLD indicating that the DLL under the node is loaded. [RS_13_31]
- When the DLL is already loaded the button in the right pane of the "Simulated system" window shall toggle to "Unload". [RS_13_32]
- The DLL included under the node shall be unloaded by selecting the "Unload" button in the right pane of the "Simulated system" window in the Node details frame. [RS_13_33]
- Unloading the DLL shall change the font of the DLL's path under that particular node to normal from BOLD in the right pane.
- The node and its details shall be added pressing "OK" in the Node Details dialog box and the same shall be indicated under the "CAN Bus". [RS_13_34]
- The selected DLL through "Browse" button can be deleted by selecting "Clear" button. This is enabled only when there is any DLL selected. [RS_13_105]
- The new node details shall be added in the simulated system's configuration file under which it is included on selecting "Save" by right clicking the simulated system. [RS_13_35]
- Adding a node shall be cancelled by selecting "Cancel" in the Node Details dialog box. [RS_13_36]
- On selecting "OK" button in the Node Details dialog box, a Node Details frame shall appear on the left pane showing the details of the node entered and the handler details frame and the controls in it shall be disabled, ref Fig 2.3.6. [RS_13_37]
- When any C file is associated with the node "Open File" button shall be enabled in the "Node Details" frame. A confirmation box shall appear whether to continue to change the node details or not. [RS_13_106]
- When "yes" is selected in the confirmation box a browser shall appear which shall enable the user to select any C file. [RS_13_107]
- When "no" is selected in the confirmation box no change shall be observed. [RS_13_108]
- When DLL is loaded, an attempt to "Open File" shall indicate the user to unload the DLL first and then try. [RS_13_109]
- When the DLL is unloaded, "Open File" shall enable the user to associate another C file. If the corresponding DLL is also present in the same directory even the DLL name shall be changed for that particular node. The new C file shall be updated in the right view of the simulated system window in the File edit box. [RS_13_110]
- When the DLL is present for the same C file which is selected, the tree view of the simulated system window shall be updated with the new DLL under that particular node. [RS_13_111]
- When no DLL is associated with the Node the menu options "All Handlers", "Key Handlers", "Message Handlers", "Error Handlers", "Timer Handlers" shall be disabled on right clicking that particular Node in the left pane(tree view). [RS_13_38]
- When the DLL is already loaded and the corresponding C file is modified outside the change shall be reflected only when the corresponding C file is built and loaded using "Build and Load" button in the right pane of the simulated system window. [RS_13_39]
- The user shall edit the C file when the corresponding DLL is loaded by selecting "Edit File" button. The file name to be opened shall be taken from the Edit box "C File" in Node details frame. [RS_13_40]
- The edited file shall be saved by using the existing feature File->Function Editor-> Save. But the changes in it shall be reflected only when the file is unloaded and then built and load using button "Unload" , "Build and Load" in the Node details frame. [RS_13_41]
- If the C file is edited "Load" button shall be enabled. But it asks the user that the C file has been edited. Do u want to build and load or not. If "No" then load the DLL which is previously there. [RS_13_112]

- When the DLL is unloaded and the C file is edited and saved through Function Editor the toggle button "Load/Unload" is disabled as it has to be first built and then loaded. Only "Build" and "Build and Load" buttons shall be enabled.(change this to the previous requirement) [RS_13_42]
- If the DLL is already loaded and the corresponding C file is present , on selecting "Open File" button it shall indicate the user that he can associate another C file only after unloading the DLL. [RS_13_43]
- The "Edit file" button shall be disabled when there is no C file for the corresponding loaded DLL in the same directory. [RS_13_44]
- When a new C file is included the toggle button "Load/Unload" shall remain disabled until it is built using "Build" or "Build and Load" buttons in the Node details frame. [RS_13_45]
- A new.C file shall be created by selecting the "Open File" button in the "File Details" frame and giving a non existing c file name. This also opens the C file for editing. [RS_13_46]
- The path of the saved new.C file using Function Editor-> Save shall be updated under the "CAN Bus" and also in the File Details frame. [RS_13_47]
- The handler details frame shall be updated with the handler details of the loaded DLL, ref Fig 2.3.7.
- The handler details frame shall be updated with the handler details of the new file only when it is built using "Build" or "Build and Load" buttons in the "Node details" frame. [RS_13_48]
- The handler details frame shall show the details of the handler of the latest built DLL if it is already present and it is loaded. [RS_13_49]
- When the user selects "Unload", the DLL loaded shall be unloaded and the user shall have the provision to include a new file or existing one by selecting "Open File" or edit the existing C file by selecting "Edit File". All the buttons "Build", "Build and Load" and "Load" shall be enabled. The handler details frame shall be disabled when the DLL is unloaded. [RS_13_50]
- The edited .C file shall be saved by selecting File-> Function Editor-> Save menu option. [RS_13_51]
- If the C file is saved as an already existing C file in the same directory a confirmation shall be asked if the user wants to replace the old file. If the DLL of the same name is loaded in the "CAN Bus" then a message shall pop up saying "The DLL with the same name is already loaded. Unload it first" on selecting "Unload" button in the "Node details" frame in the right pane. [RS_13_52]
- If there is no DLL loaded from the "Node details" dialog box or no C file associated with that particular node then "Build", "Build and Load", "Load" and "Edit File" buttons are disabled. [RS_13_53]
- To build, build and load, load and unload, from the function editor buttons: "B", "BL", "L", "UL" in the toolbar which are already present shall be used. [RS_13_54]
- On right clicking any simulated system "Remove All nodes" shall be disabled if there are no nodes present under that simulated system. [RS_13_113]
- All the nodes under a particular simulated system shall be deleted by right clicking and selecting "Remove All Nodes" option. A confirmation message is thrown whether the user is sure to delete them or not if the DLL associated with it is not loaded. [RS_13_55]
- When the user selects "Yes" in the confirmation message box , the node details are deleted from the left pane under the selected simulated system and the right pane shows an empty frame. [RS_13_56]
- On selecting "No" in the confirmation message box there shall be no change in the "Simulated Systems" window. [RS_13_57]
- If the DLL associated with any of the nodes is loaded and "Remove All Nodes" is selected, the user shall be given an indication that some of the DLL are loaded. Unload it first and try again. Only when no DLL is loaded under a simulated system "Remove All Nodes" is possible. [RS_13_114]
- The user shall have a provision to Enable/Disable: All Handlers [RS_13_58], All Key Handlers [RS_13_59], All Message Handlers [RS_13_60], All Error Handlers [RS_13_61], All Timer Handlers [RS_13_62] by right clicking the required simulated system and selecting the appropriate option.
- When any of the DLL is loaded under the simulated system, all these options shall be enabled in the pop up menu. [RS_13_115]
- When no DLL under the simulated system is loaded , all these options shall be disabled in the pop menu. [RS_13_116]
- All the handlers of all the DLL loaded under the simulated system can be enabled by selecting "Enable All Handlers" on right clicking any simulated system. [RS_13_117]
- When "Enable All handlers" is selected , as all the handlers are enabled the text shall be toggled to "Disable All Handlers" , "Disable All Key Handlers" , "Disable All Message Handlers" , "Disable All Error Handlers" , "Disable All Timer Handlers". [RS_13_118]
- All the handlers of all the DLL loaded under the simulated system can be disabled by selecting "Disable All Handlers" on right clicking any simulated system. [RS_13_119]

- The text in the pop up menu shall be toggled to "Disable All handlers" only when "Enable All handlers" was selected. [RS_13_120]
- When "Disable All handlers" is selected , as all the handlers are disabled the text shall be toggled to "Enable All Handlers" , "Enable All Key Handlers" , "Enable All Message Handlers" , "Enable All Error Handlers" , "Enable All Timer Handlers". [RS_13_121]
- When "Enable All Key Handlers" is selected all the key handlers of all the DLL which are loaded under the simulated system shall be enabled and the text of the option is toggled to "Disable All Key Handlers". [RS_13_122]
- When "Enable All Message Handlers" is selected in the pop up menu of the tree view , all the key handlers of all the DLL which are loaded under the simulated system shall be enabled. So when any node under that simulated system is selected the handler details of the node shall show the status of the message handler as "Enabled". [RS_13_123]
- When the message handler is selected in the handler details frame , the button text is toggled from "Enable Handler" to "Disable Handler". [RS_13_124]
- When "Enable All Message Handlers" is selected all the message handlers of all the DLL which are loaded under the simulated system shall be enabled and the text of the option is toggled to "Disable All Message Handlers". [RS_13_125]
- When "Enable All Message Handlers" is selected in the pop up menu of the tree view , all the message handlers of all the DLL which are loaded under the simulated system shall be enabled. So when any node under that simulated system is selected the handler details of the node shall show the status of the key handler as "Enabled". [RS_13_126]
- When the message handler is selected in the handler details frame , the button text is toggled from "Enable Handler" to "Disable Handler". [RS_13_127]
- When "Enable All Error Handlers" is selected all the error handlers of all the DLL which are loaded under the simulated system shall be enabled and the text of the option is toggled to "Disable All Error Handlers". [RS_13_128]
- When "Enable All Error Handlers" is selected in the pop up menu of the tree view , all the error handlers of all the DLL which are loaded under the simulated system shall be enabled. So when any node under that simulated system is selected the handler details of the node shall show the status of the error handler as "Enabled". [RS_13_129]
- When the error handler is selected in the handler details frame, the button text is toggled from "Enable Handler" to "Disable Handler". [RS_13_130]
- When "Enable All Timer Handlers" is selected all the timer handlers of all the DLL which are loaded under the simulated system shall be enabled and the text of the option is toggled to "Disable All Timer Handlers". [RS_13_131]
- When "Enable All Timer Handlers" is selected in the pop up menu of the tree view , all the timer handlers of all the DLL which are loaded under the simulated system shall be enabled. So when any node under that simulated system is selected the handler details of the node shall show the status of the timer handler as "Enabled". [RS_13_132]
- When the timer handler is selected in the handler details frame, the button text is toggled from "Enable Handler" to "Disable Handler". [RS_13_133]
- A particular simulated system shall be deleted by right clicking on it and selecting "Delete Simulated System". A confirmation message box shall pop if the user is sure of deleting it or not. [RS_13_63]
- When the user selects "Yes" the DLL s loaded under that particular simulated system are unloaded and that simulated system details are deleted from the left pane under the "CAN Bus" and the right pane shall show an empty pane. [RS_13_64]
- Deleting the simulated system shall delete the path of that simulated system's configuration file from the main configuration when the main configuration is saved by File->Configuration->Save. [RS_13_65]
- If any DLL are loaded under the simulated system then it shall be intimated to the user that the one or more DLL are loaded. Unload it and try again! [RS_13_134]
- On right clicking any Node in the left pane, ref fig 2.3.8 a pop up menu shall appear having options : "Edit Node", "Remove Node", "Enable /Disable All Handlers", "Enable /Disable Key Handlers", "Enable / Disable Message Handlers", "Enable / Disable Error Handlers", "Enable / Disable Timer Handlers". [RS_13_66]
- The node details, "Node Name" and the "DLL" shall be edited by right clicking on that particular node in the left pane of the "Simulated Systems" window and selecting "Edit Node". The Node Details dialog box shall appear to enable the user to edit the details. [RS_13_67] ref Fig 2.3.8
- On selecting "OK" the details shall be updated in the Node Details frame shown in the right pane [RS_13_68] and if the name of the Node is edited then it is also updated under the "CAN Bus". [RS_13_69]

- Whenever there is a change in the node details it shall be updated in the simulated system's configuration file under which it is included when saved by right clicking on that simulated system and selecting "Save" option. [RS_13_70]
- The node shall be deleted from the "CAN Bus" by right clicking on that particular node and selecting "Remove Node" from the pop up menu. A confirmation message box shall pop up. [RS_13_71]
- On selecting "Yes" in the confirmation message box the DLL if loaded under it is unloaded [RS_13_72]and the node details shall be deleted from the simulated system's configuration file under which it is configured (if saved) [RS_13_73] and also in the left pane of the "Simulated System" window. [RS_13_74]
- On selecting "No" in the confirmation message box the node shall not be deleted and no change shall be observed under the "CAN Bus". [RS_13_75]
- If the DLL associated with the node is loaded, then intimation shall be given to the user that the DLL is loaded. Unload it and try again. [RS_13_135]
- When one or more nodes are configured under a simulated system, the updated node details shall be shown in the right pane on a single left click of a particular node in the left pane. [RS_13_76] ref Fig 2.3.7
- The handler details frame shall show the details of the handlers , ref Fig 2.3.7 associated with that particular node's Dll and also the details of individual handlers when a particular handler is selected from the handler details list box provided only when the DLL is loaded. [RS_13_77]
- All the handlers of that particular node shall be enabled or disabled by selecting the toggling button "Enable/ Disable All Handlers"[RS_13_78]
- This change shall be reflected in the pop up menu which appears on right clicking on that particular node in the tree view of the simulated system window, i.e. the text in the pop up menu shall also be changed to "Disable All handlers". [RS_13_136]
- When all the handlers are enabled the button shall toggle to show "Disable All Handlers" and vice a versa. When all the handlers are neither enabled or disabled the toggle button shall toggle to "Enable All Handlers" by default. [RS_13_79]
- The Start/Stop Timers dialog box shall be modified to deal with multiple nodes by providing a combo box with all the nodes configured, ref Fig 2.3.9.When a node is selected from the combo box the "Timer Handler Name" column shall be updated with the timer handlers configured for that particular node. Individual timer of each node shall be enabled or disabled by selecting the timer names. [RS_13_80]
- Individual handlers of that particular node shall be enabled or disabled by selecting a handler in the handler details frame and pressing the "Enable /Disable Handler" toggle button. [RS_13_81]
- When the selected handler is enabled the button toggles to "Disable Handler" and vice a versa. [RS_13_82]
- On enabling/disabling the timer handler in the handler details or by right clicking the simulated system or the node under the "CAN Bus" the Start/Stop Timers dialog box shall appear with the Node Name combo box entry as the node name from which it is called showing its Timer information [RS_13_83].
- The combo box for the Node Name shall be disabled if it is invoked for a particular node. [RS_13_84]
- When "Enable / Disable All Timer handlers" is selected by right clicking on any simulated system or from the Tool bar Set Reset Timer dialog box shall appear with all the active nodes (nodes whose DLL are loaded) in the combo box respectively. [RS_13_137]
- The user shall have a provision to Enable / Disable: All Handlers [RS_13_85], Key Handlers [RS_13_86], Message Handlers [RS_13_87], Error Handlers [RS_13_88], All Timer Handlers [RS_13_89] by right clicking the required node and selecting the appropriate option
- The whole "CAN Bus" configuration and its details shall be saved by : File-> Configuration-> Save. This file shall contain the list of the configuration paths of all the simulated systems under the "CAN Bus". [RS_13_90]
- If any simulated systems are present under the "CAN Bus", the focus shall be on the first simulated system showing the empty frame in the right pane when the user selects menu option Configuration->Simulated Systems. [RS_13_91]
- If the user tries to close the application or the "Simulated System" window without saving the configuration a message box shall appear saying "You have made changes to the configuration. Do you want to save it?" [RS_13_92]
- The configuration shall be saved by selecting "Yes" option and discarded by selecting "No". [RS_13_93]
- New buttons like "BA" [RS_13_94], "BLA" [RS_13_95], "LA" [RS_13_96], "ULA" [RS_13_97], build all, build and load all, load all, unload all shall be added in the toolbar.
- When "LA" is selected from the tool bar all the DLL of all the simulated systems configured under the present main configuration shall be loaded. [RS_13_138]

- This shall be indicated by making the DLL names BOLD in the tree view of the simulated system window. [RS_13_139]
- When "UA" is selected from the tool bar all the DLL of all the simulated systems configured under the present main configuration shall be unloaded. [RS_13_140]
- This shall be indicated by making the DLL names normal from BOLD in the tree view of the simulated system window. [RS_13_141]
- When the DLLs are loaded the state of the DLLs under any simulated system shall indicate that the DLLs are loaded. When right clicked on any simulated system whose DLL are loaded, the text in the pop up menu shall be toggled to "Unload All" and the same when right clicked on any node. [RS_13_142]
- "BA" , "BLA" in the tool bar shall be enabled only when at least one C file is present under any simulated system which is configured under that particular main configuration. [RS_13_143]
- "LA" shall be enabled only when at least one DLL is present under any simulated system which is configured under that particular main configuration. [RS_13_144]
- The tool bar buttons to enable / disable: all handlers , message handlers , timer handlers, key handlers, error handlers shall be modified to enable all the handlers or individual handlers for all the DLL which are loaded from all the simulated systems. [RS_13_145]
- These tool bar buttons shall be enabled only when at least one DLL is loaded and the tool is connected. [RS_13_146]
- When "BLA" , "LA" or "UA" tool bar buttons are selected the success or failure in the operation shall be indicated to the user in the Output Window. [RS_13_147]
- When "BLA" or "BA" is selected from the tool bar and if some of the files are not built properly the user shall be indicated about the errors in building the file in the Output window. Double Clicking on the Output window on the error , shall open the respective C file in the function editor if it not opened and point at the statement which has caused the error. [RS_13_148]
- The success or failure in the operation shall be indicated to the user in the Output Window when the user selects "Load All", "Unload All" , "Build All" on right clicking on any simulated system or when "Load", "Unload", "Build" , "Build All" is selected in the right view of the simulated system window. [RS_13_149]
- The global variables shall be accessed and modified across nodes. [RS_13_98]
- Two API's "Node Active" [RS_13_99] and "Node Passive" [RS_13_100] shall be implemented.
- The online help shall be updated to reflect the new features. [RS_13_101]
- The performance of the tool shall remain same as for the 4.00.01.X.XXX version tool. [RS_13_102]
- Appropriate validation message shall pop warning the user wherever it is required. [RS_13_103]
- The API's exposed from the application shall be updated internally to handle multiple DLLs. [RS_13_104]

Frame Transmission

From triggering viewpoint, this occurs in two different ways. First one is time-oriented whereas for the other, triggering comes from event handlers, user-programme, test cases in test automation etc. The second one features in activities such as node programming, test automation, automation server and hence is part of a larger scenario.

This section describes transmission module which deals with the first one thereby employing time-oriented triggers. Independent yet obvious another of the considerations is the frame data configuration that focuses on the bus data exchange aspect. So again the discussion proceeds with configuration and executor modules.

Configuration

The first step is to define transmission blocks which can be defined with a name. There is no specific nomenclature to be imposed. Nor is there any enforced principle on uniqueness of name. A transmission block is appreciated with a certain number of parameters which are the results of an at length analysis and collation of the various below characteristics.

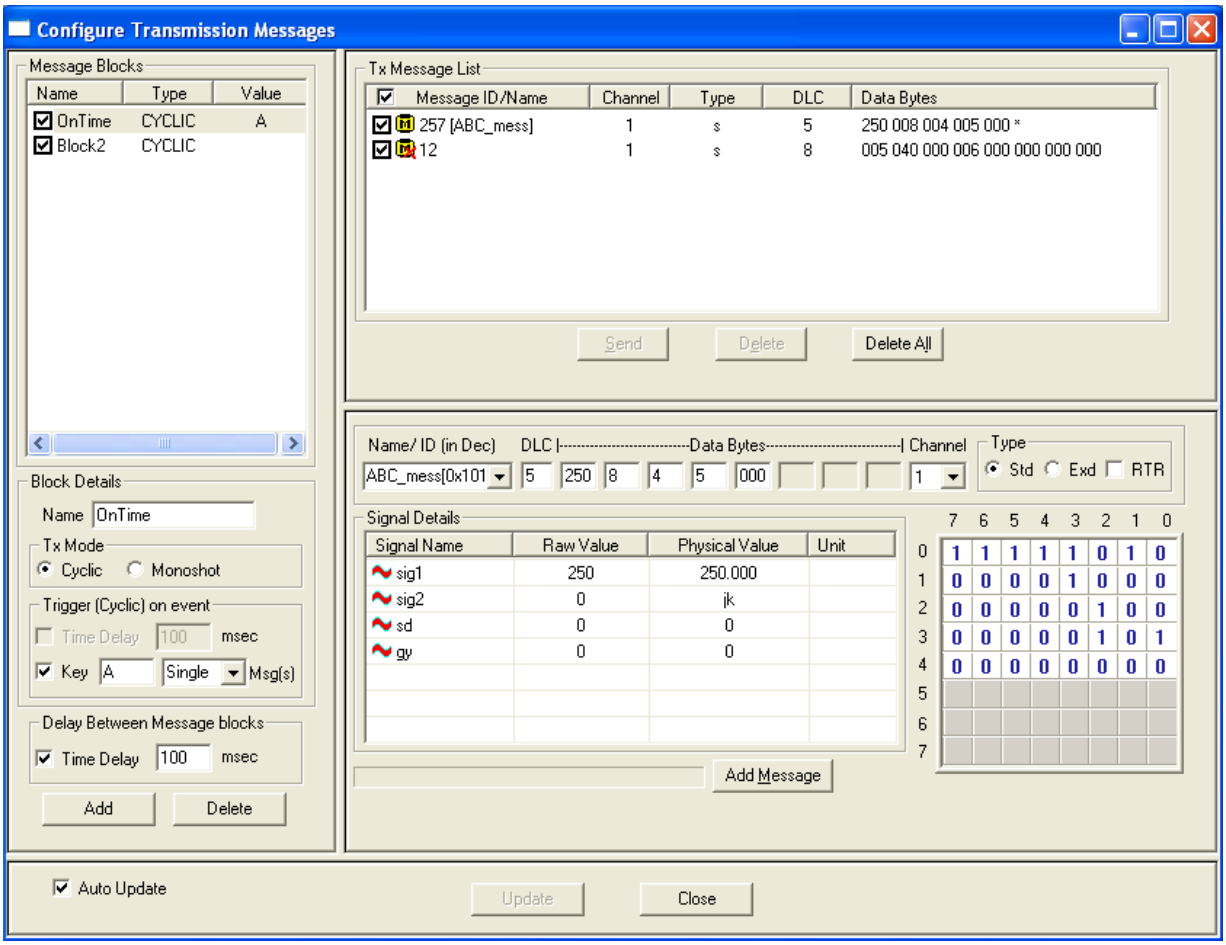
- Transmission block repeats with a time period specified (cyclic block)
- Transmission block doesn’t get repeated (monoshot block)
- Frames constituting a transmission block maintain a uniform time delay between them
- Transmission for a block takes place by a manual event like a key press
- Manual event transmission for a block sends one frame at a time
- Manual event transmission for a block sends all frames in a burst
- Transmission of an entire block at a time

The above listing results in the following transmission configuration parameters.

1. Name of transmission block
2. Activation status of a block
3. Frame list with which this block consists of. Each of the selected frames shall also be deactivated from transmission. This means only those selected can be transmitted.
4. Transmission mode which can be either cyclic or monoshot.
5. Trigger event: Triggering event for a frame transmission which can be a timer, a manual one such as a key press. When it's a manual event, either one frame gets transmitted at a time or the entire list in a burst. Delay between blocks will send an entire block in a single shot and will wait for the delay mentioned next to it, before sending the next block.
6. Automatic Updation: All the parameters set by the user should be automatically updated if the user desires it. "Auto Update" option serves this purpose. All the parameters will be set as soon as they loose focus.

Besides, there is another aspect which involves configuration of the message data. It should be possible to edit a defined message to the extent of its individual signals covering both its raw and engineering values.

The above listing translates into the following user interface specification for CAN~



Transmission

This transmits the frames defined in the transmission block list according to the time period and data specified. This must ensure the time integrity is maintained.

The tagged requirement details are tabulated below:

			CAN	FlexRay	J1939
[RS_17_01]	Feature	Configuration Addition of a transmission block	X	X	X

			CAN	**FlexRay**	**J1939**
[RS_17_02]	Feature	Deletion of a transmission block	X	X	X
[RS_17_03]	Feature	Enable / disable a transmission block	X	X	X
		Editing characteristics of a transmission block			
[RS_17_04]	Feature	- Editing of transmission block	X	X	X
[RS_17_05]	Feature	- Choosing between 'Cyclic' and 'Monoshot' transmission mode	X	X	X
[RS_17_06]	Feature	- Enable / disable time delay characteristics.	X	X	X
[RS_17_07]	Feature	- Update time delay	X	X	X
[RS_17_08]	Feature	- Enable / disable key event trigger for transmission	X	X	X
[RS_17_09]	Feature	- Configure key event trigger by specifying the key stroke	X	X	X
[RS_17_10	Feature	Transmission of an entire block at a time and waiting for specified delay before transmission of next block	X		
[RS_17_11]	Feature	- Specify if key event for a single message or a burst of message	X	X	X
		Editing a transmission block data			
[RS_17_12]	Feature	- Addition of a frame, defined / undefined	X	X	X
[RS_17_13]	Feature	- Deletion of a frame	X	X	X

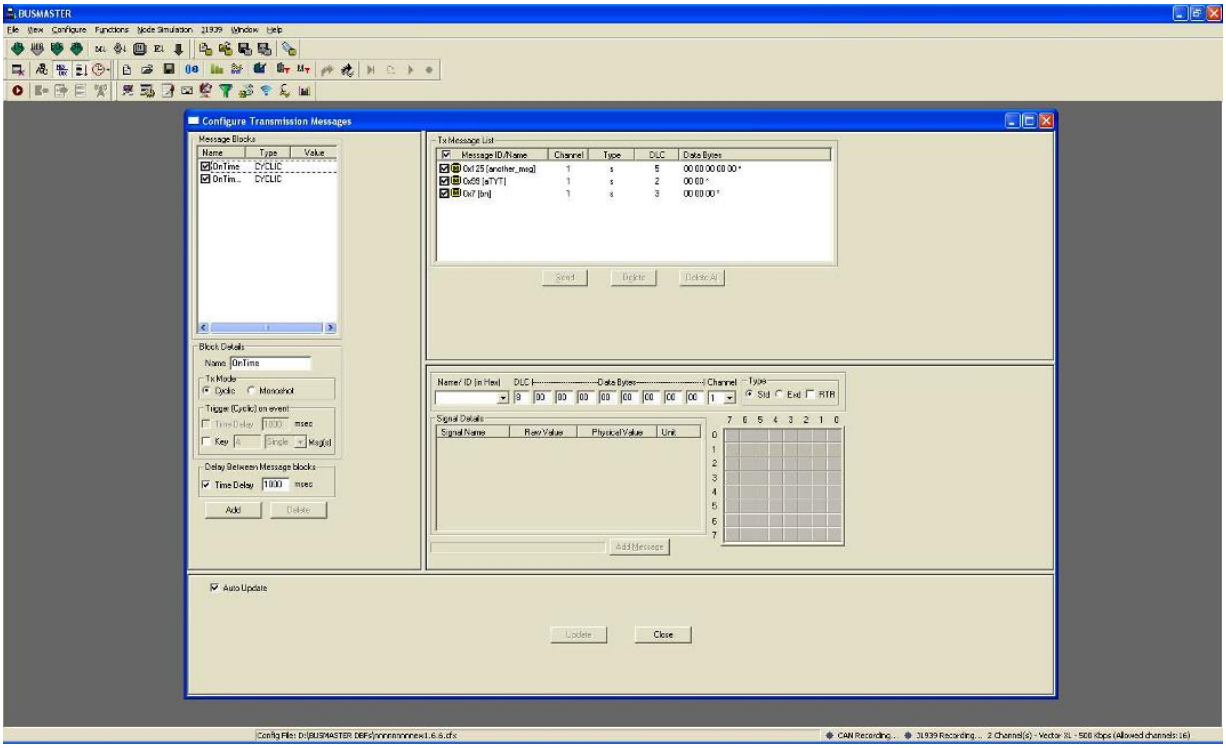
			CAN	**FlexRay**	**J1939**
[RS_17_14]	Feature	- Clearing the frame list	X	X	X
[RS_17_15]	Feature	- Enable / disable a frame from transmission	X	X	X
		Editing of a frame data in a block			
[RS_17_16]	Feature	- Edit the frame characteristics	X	X	X
[RS_17_17]	Feature	- Editing data for an undefined frame	X	X	X
[RS_17_18]	Feature	- Editing data for a defined frame	X	X	X
[RS_17_19]	Feature	Automatically saving the parameters as they are set	X		
[RS_17_20]	Feature	Committing the transmission block list	X	X	X
[RS_17_21]	Feature	Saving of configuration data	X	X	X
[RS_17_22]	Feature	Retrieving of configuration data	X	X	X
		Transmission			
[RS_17_23]	Feature	Manual transmission of a frame	X	X	X
[RS_17_24]	Feature	Start transmission activity	X	X	X
[RS_17_25]	Feature	Stop transmission activity	X	X	X
[RS_17_26]	Design	Maintaining integrity of the transmission frequency	X	X	X
[RS_17_27]	Design	Ensure substantial number of transmission block doesn't overburden the system	X	X	X

Time Delay Between Blocks

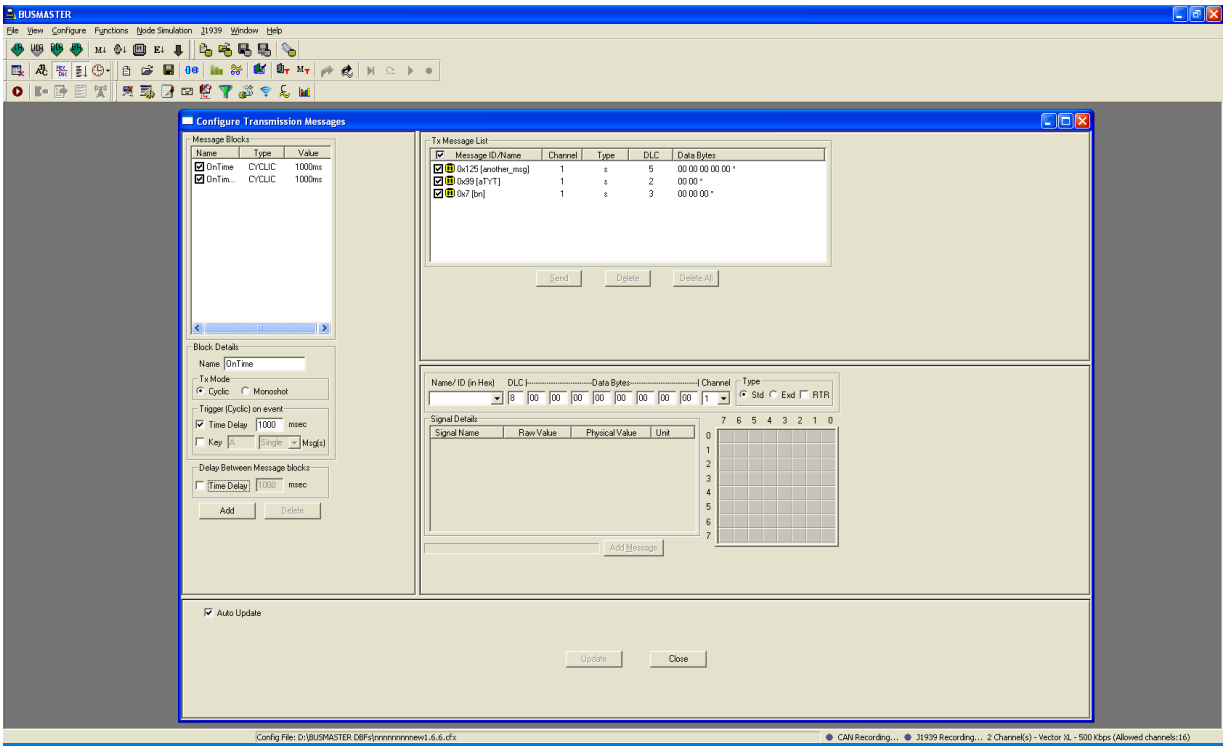
- Abbrevations:
- TDBB- Time Delay Between Blocks
- TDBM- Time Delay Between Messages

UI Behavior:

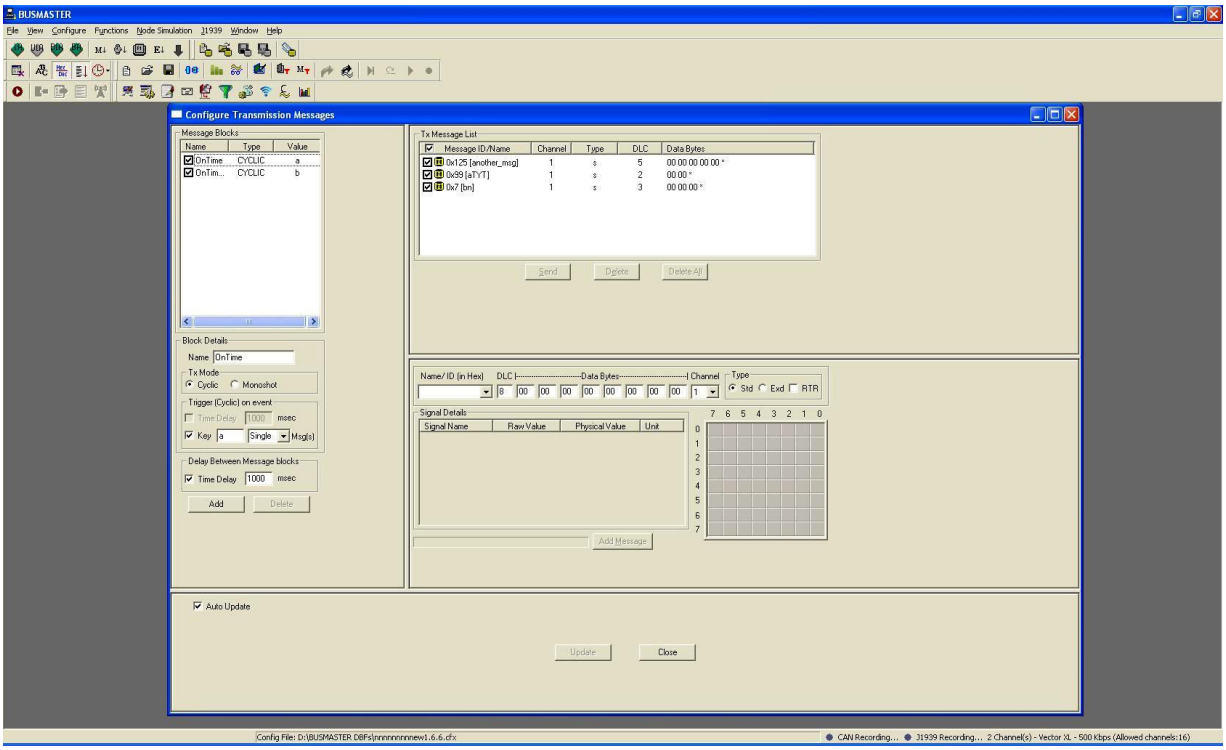
- Scenario 1:
- 1.Check TDBB
- 2.TBDM and its delay edit box should be disabled

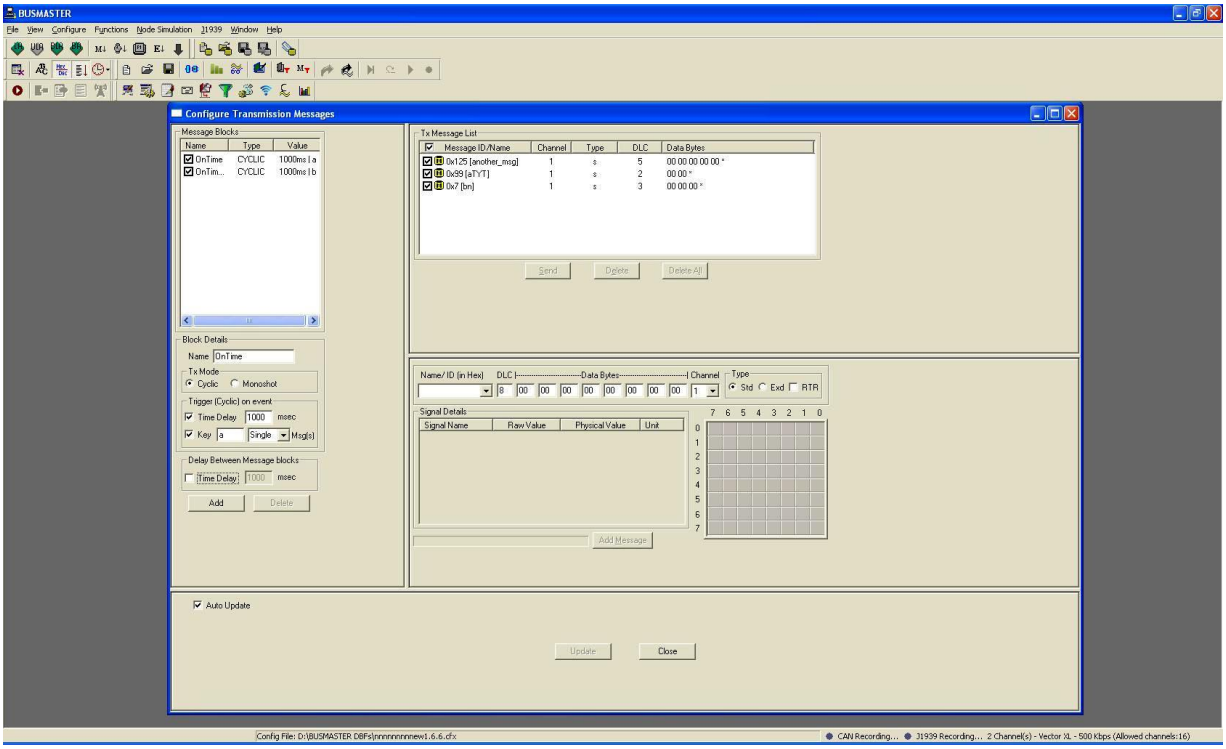


- Scenario 2:
- 1.Uncheck TDBB
- 2.TBDM and its delay edit box should be enabled



- Scenario 3:
- 1.Check / Uncheck TDBM
- 2.Check / Uncheck TDBB, the previous stats of TDBM and its time delay has to be retained.
- 3.Also the key trigger values has to be retained





Functional Behavior:

- Scenario 1:
- 1.Check TDBB
- 2.Add multiple blocks in the Tx window with some messages in each block
- 3.When the transmission is started, the messages of first block has to be sent first 4.Messages of second block should be sent after the given time delay. By default the time delay should be 100 ms
- 5.Cyclic / monoshot functionality to be same as that of TBDM checked

- Scenario 2:
- 1.Select TBDD. Input some time delay value
- 2.Connect and transmit messages

- 3.Uncheck TBDD and check TBDB. Connect and transmit again. Time delay specified for TBDM should be used for message transmission and vice versa.
- Scenario 3:
- 1.Select key press either for one block or for multiple blocks
- 2.Key press should not depend or wait for any time delay specified either for TBDD or TBDM.
- 3.Once the key is pressed the message should be sent based on the Single / All selection
- Scenario 4:
- 1.Message transmission should transmit or not transmit the messages based on the check / uncheck status of each message during run time
- 2.If a message is checked before transmission starts, the message should be sent when transmission is started.
- 3.If a message is unchecked before transmission starts, the message should not be sent when transmission is started.
- Scenario 5:
- 1.Add multiple messages with some messages in each block
- 2.Connect and transmit messages
- 3.During transmission if a message in one block is checked / unchecked or any other data of a message is changed, then all the messages from the first should be sent again as a single shot.
- Scenario 6:
- 1.If TBDD is unchecked then the list control in the Message Blocks view should not display any time delay in it to the right side of the message name

Note: All other scenarios should work normally as like that of TDBM. This can be either with / without AutoUpdate selected. Closing the window, closing the application and loading the same configuration again etc. should retain the values. Etc. Such functionalities are expected.

Session Replay

Here the objective is to play back a recorded monitoring session using the generated log files from that session in question, as the input data source. Play back a session is characterized by transmitting all the data frame entries in the input data file, both transmitted and received. To be noted that realization of session replay feature doesn't qualify for its name ad verbatim. The only reason behind this apparent aberration is that otherwise, for received frames the application node shall be dependent on other nodes resulting in a distributed testing environment with higher degree of implementation complexity, and so far there is no such demand from current user group.

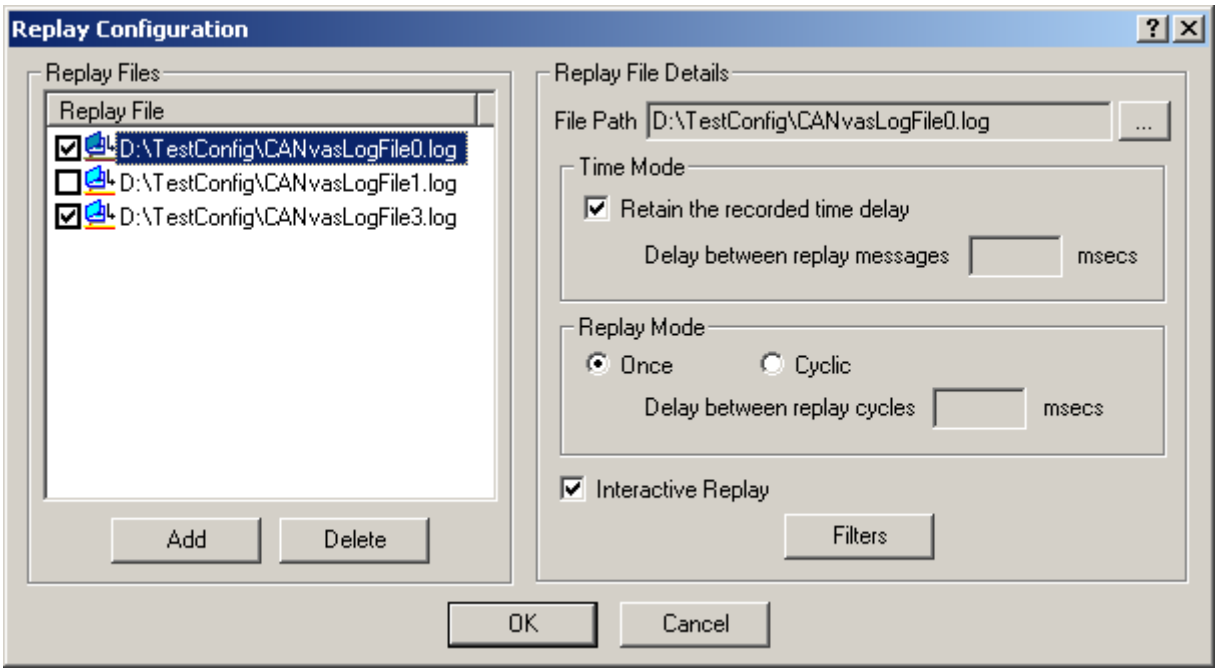
Session reply occurs in two sub-modules the first one deals with the necessary configuration whereas the last one with the execution activity.

Configuration

This manages the replay entries. A replay entry is parameterized with the following ~

1. Log file: An existing log file.
2. Time mode: Time mode is defined as the expected time delay between frames transmitted, which is either the recorded time delay in the associated log file or a time delay specified by the user. The time unit here is millisecond.
3. Repetition mode: Repetition mode can have two values. 'Once' means the replay file will be played only once. 'Cyclic' means it will be repeated with the interval specified by the user.
4. Replay mode: Replay mode shuttles between interactive replay more and non-interactive replay mode. The former one involves human intervention in which the user can set breakpoints against some selected entries. Transmission halts at those entries awaiting user-event to go ahead. It is also possible for the user to transmit one frame at a time. All these together provide the user a better control over session play back. This is a manifestation of the typical debugging scenario in this domain. Meaning of the later one is obvious and hence needs no further explanation.
5. Filter set: The filter set associated. This is optional.

Below goes the user interface specification:

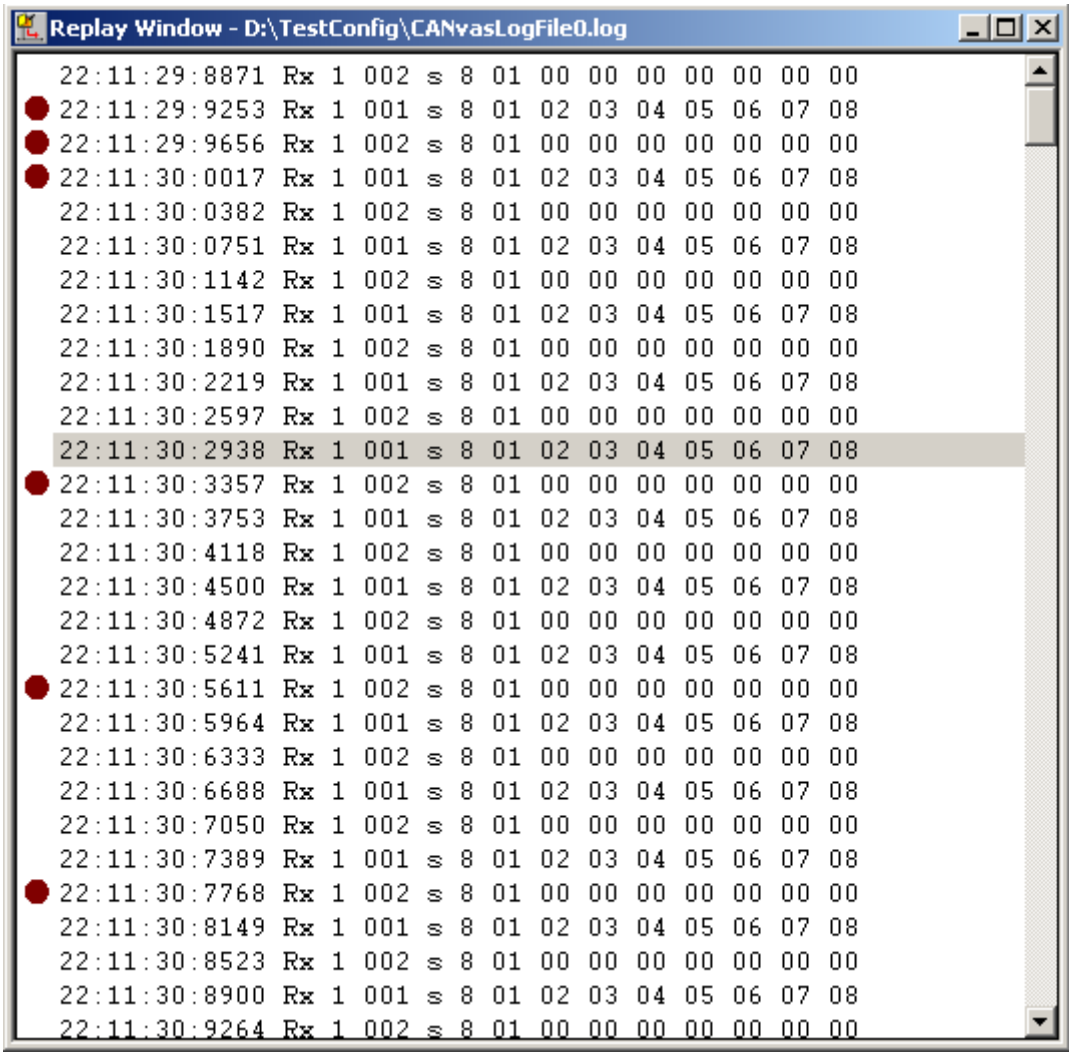


Replay Window

This carries out the execution part and transmits frames according to the rules set by the configuration module. Execution activity set involves steps resulting in full control in the replay execution spectrum. These can be listed as below:

- Add / remove breakpoints
- Start replay session and halt at a breakpoint
- Select any entry to make it the current execution location
- Transmit current entry and move forward one step
- Transmit current entry and move forward transmitting until hitting next breakpoint
- Skip current entry and move one step
- Stop the replay activity

The user interface specification is presented below:



The tagged requirement details are tabulated below:

			CAN	FlexRay	J1939
19		Session Replay Configuration			
[RS_19_01]	Feature	Existing log file shall be identifiable as the input data source	X	-	-
[RS_19_02]	Feature	Multiple input data source shall be employable	X	-	-
[RS_19_03]	Feature	It shall be possible to add / remove any log file from the input data source set	X	-	-
[RS_19_04]	Feature	In the input data source set, any entry can	X	-	-

			CAN	FlexRay	J1939
		be enabled / disabled			
[RS_19_05]	Feature	It shall be possible to configure / reconfigure an existing input data source by -	X	-	-
[RS_19_06]	Feature	- Changing the log file absolute path	X	-	-
[RS_19_07]	Feature	- Choosing between recorded time delay and user-specified delay in ms	X	-	-
[RS_19_08]	Feature	- Choosing between 'Once' and 'Cyclic' repetition mode	X	-	-
[RS_19_09]	Feature	- Choosing between interactive and non-interactive replay mode	X	-	-
[RS_19_10]	Feature	- Using a filter set	X	-	-
[RS_19_11]	Feature	It shall be possible to commit the changes or to roll back	X	-	-
		Replay Window			
[RS_19_12]	Feature	Add / remove breakpoints	X	-	-
[RS_19_13]	Feature	Select eny entry to make it the current execution location	X	-	-
[RS_19_14]	Feature	Start / resume replay session	X	-	-
[RS_19_15]	Feature	Halt at a breakpoint	X	-	-
[RS_19_16]	Feature	Step on (current entry is transmitted and move by one step)	X	-	-
[RS_19_17]	Feature	Step over (Move by one	X	-	-

			CAN	FlexRay	J1939
		step without transmitting the current entry)			
[RS_19_18]	Feature	Transmit current entry and move forward until meeting the next breakpoint	X	-	-
[RS_19_19]	Feature	Terminate current replay session	X	-	-

Filter

This feature aims at reducing the runtime load on the application and also to higher efficiency by extracting from the channel only those frames identified as relevant by the user. A filter defines a condition by which a set of one or multiple frames can be identified so that a blocking mechanism is applied on that set or on the absolute complement of it. In the former one the usage is called stop (blocks the identified frames) whereas in the later one the filter usage is the pass usage (blocks every frame except those in the filter).

As an identification condition can be a combination of individual identification conditions, a filter also follows suit with the priority being on the blocked set.

A filter, from the implementation perspective, realizes a set of frames. We denote a filter as F and the set A. We also denote FA to be the function which performs the filtering action. Let the parameterized usage be FA (<Bus Data>, <Filter>, <Usage>). The two use cases shall be the following:

FA (<Bus Data>, F1, Pass) = A1; Outcome is the set of available frames

And,

FA (<Bus Data>, F1, Stop) = A1C; Outcome is the set of available frames

Assume two filters F1 and F2, and the associated sets A1 and A2. Hence, the final outcome of F1 and F2 on a bus data set shall be equal to

FA (<Bus Data>, F1, <Usage>) ∩ FA (<Bus Data>, F2, <Usage>)

Filtering can take place from two levels:

- Hardware level: This is the most efficient type of filtering as the controller blocks the unwanted frames thereby taking off unnecessary load from the application. Prerequisite is that the controller must support filtering feature.
- Software level: This is not as efficient as the previous type of filtering, but this has guaranteed availability and no dependency on third party.

Signal Watch Window

In signal watch window the data are looked at a different perspective, having the focus rather on signals. Here the user identifies from the frame database a set of signals of choice. Upon confirmation, signal watch window monitors engineering and raw values of the chosen signal set. The different fields in the signal watch window are message / frame name, signal name, engineering / physical value and raw value.

Naturally this has two sub-modules namely, the configuration module and the executor.

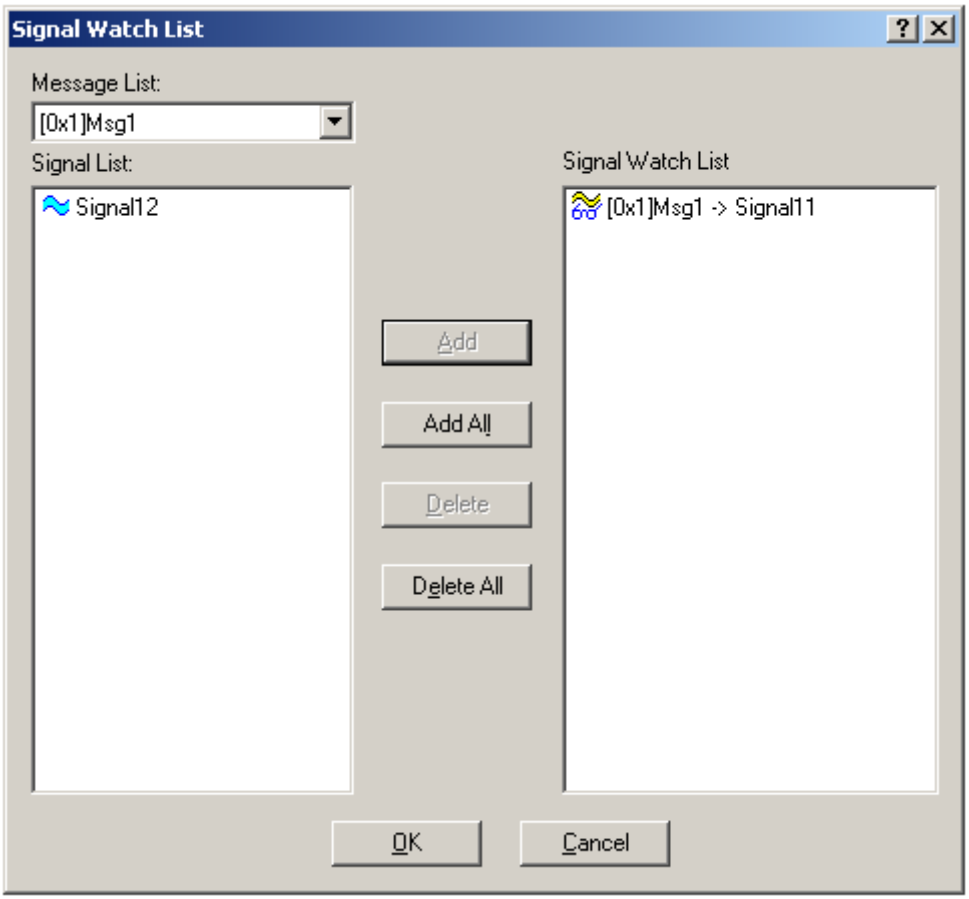
Configuration - This is used to configure the signal watch activity. Naturally this consists of the following functionalities:

1. Listing of the available frames
2. Listing of the signals for each frame
3. Selection of one or multiple or all the signals under a frame
4. Deselection of one or multiple or all the signals under a frame
5. Retrieval of the available and selected list data

6. Confirming the present selection
7. Setting the refresh rate of signal watch window

Below is a schematic diagram of the configuration user interface. Clearly, this consists of the following sections in general:

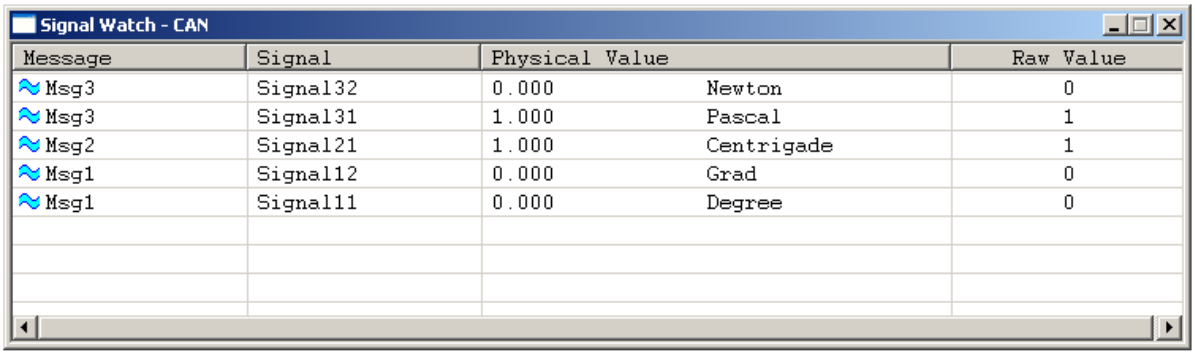
1. A master list of the frames
2. Available signals for all frames, shown for a selected frame at a time
3. Selected signal list for signal watching showing the container frame



Watch Window - This is the module that actually does the monitoring of the selected signals. This consists of the following functionalities:

1. Showing of the four different aspects of signal data which are the owner frame name, signal name, raw value and engineering value with unit
2. Updating the window with latest signal data

The user interface diagram for the watch window is the following:



The title bar indicates the bus. This means for each bus supported, there shall be a watch window.

The detailed tagged requirements are given below:

			CAN	**FlexRay**	**J1939**
18		Signal Watch Window			
	Definition	Signal watch window monitors a group of signals selected by the user. Monitoring takes place in terms of their raw and engineering values.	X	X	X
		Configuration			
[RS_18_01]	Feature	Signal watch configuration dialog box lists frames / messages from the loaded database (combo box / list box)	X	X	X
[RS_18_02]	Feature	There are two lists maintained - one for the available signals for a frame (available list) and the other one for the signals selected (selected list)	X	X	X
[RS_18_03]	Feature	There are four selection buttons maintained - 'Add', 'Add All', 'Delete' and "Delete All'	X	X	X
[RS_18_04] [RS_18_05] [RS_18_06] [RS_18_07] [RS_18_08] [RS_18_09]	Feature	Enable / disable status of the buttons: **1.** At least one selected entry in available list - 'Add' and 'Add All' buttons enabled. **2.** At least one selected entry	X	X	X

			CAN	FlexRay	J1939
		in selected list - 'Delete' and 'Delete All' buttons enabled. **3.** No entry in available list - 'Add' and 'Add All' buttons disabled. **4.** No entry in selected list - 'Delete' and 'Delete All' buttons disabled. **5.** At least one unselected entry in available list - 'Add All' button enabled. **6.** At least one unselected entry in selected list - 'Delete All' button enabled.			
[RS_18_10]	Feature	Both the lists support multiple selection.	X	X	X
[RS_18_11]	Feature	Add' button moves the selected item(s) from the available list into the selected list	X	X	X
[RS_18_12]	Feature	Delete' button moves the selected item(s) from the selected list into the available list	X	X	X
[RS_18_13]	Feature	Add All' button moves all the entries into the selected list.	X	X	X
[RS_18_14]	Feature	Delete All' button moves all the entries into the available list.	X	X	X

			CAN	**FlexRay**	**J1939**
[RS_18_15]	Feature	Entry in selected list: <Frame Name>-><Signal Name>	X	X	X
[RS_18_16] [RS_18_17]	Feature	Ok' button confirms the selection whereas 'Cancel' button causes no change of the current signal watch selection. Signal watch activity	X	X	X
[RS_18_18]	Feature	Signal watch window title bar indicates name of the current bus standard	X	X	X
[RS_18_19] [RS_18_20]	Feature	There are four columns in the window - 'Message / frame' showing name of the frame under which the signal occurs, 'Signal' which shows the signal name, 'Raw value' reflecting raw value of the signal and 'Engineering value' under which the fields are updated with the engineering values with unit.	X	X	X
[RS_18_21]	Feature	On right click a popup menu is displayed.	X	X	X
[RS_18_22]	Feature	Signal watch window is updated at the refresh rate set by the user	X	X	X
[RS_18_23]	Feature	Items of the popup menu - 'Clear' (clears the signal watch window).	X	X	X

Signal Graph Window

With regard to data presentational content, signal graph window is similar to signal watch window. The basic difference is that instead of textual the presentation here is visual, i.e., values of a signal are shown with a real-time line diagram where the X-axis represents the time axis whereas the Y-axis the signal values. In addition to the ECU data, bus statistics related data are included in signal graph window too. The information is again vehicle bus standard specific.

Configuration – The configuration module takes into account both the ECU and bus statistics data. Loading a valid database is a prerequisite for configuring database messages whereas selection of a valid bus standard is the obvious prerequisite for statistical data configuration. Both the engineering and raw values of a signal can be selected for monitoring. So database messages / frames category lists entries frame-wise. On the other hand, statistical category list has channels as the highest level of hierarchy under it. This also ensures that signals are represented by different colours, to make them visually distinguishable from each other.

Signal graph window – The already indicated dynamic chart with signal value (both raw and engineering) and time as its two axes is the central theme of this module. This provides analyzing and visual functionalities like showing / hiding of the grid, auto-ranging, auto-fitting of signals, tracking of delta-values of a signal at any two instances of time – based on the choice the values can be either exact or indicative.

The detailed tagged requirements are given below:

			CAN	FlexRay	J1939
22		Signal Graph Window			
	Definition	Signal Graph Window supports plotting graph for signal values and statistics parameters. This includes raw and physical values of a signal. Network statistics parameters can be added to plot graph. The number of graphs plotted are configured by user.	X	X	X
[RS_22_01]	Feature	Signal Graph window shall be bus independent.	X	X	X
[RS_22_02]	Feature	The number of Signal Graph windows shall be dependent on the number of active buses. This means there shall be one window per bus.	X	X	X
[RS_22_03]	Feature	Signal Graph window shall	X	X	X

			CAN	FlexRay	J1939
		be completely decoupled from application.			
[RS_22_04]	Design	A common buffer (or channel in abstract jargon) for all the buses.	X	X	X
[RS_22_05]	Design	Each entry in buffer shall follow the format: <BUS><DLC><Msg Data Bytes>	X	X	X
[RS_22_06]	Design	Signal Graph Window should be passed the list of signals to be plotted, by the application.	X	X	X
[RS_22_07]	Design	There will be a separate interface for each bus to interpret the configured message signals.	X	X	X
[RS_22_08]	Design	Using the information of above two listings, the raw / engineering values are calculated and displayed by Signal Graph Window.	X	X	X
[RS_22_09]	Design	Interpretation for a particular bus needs <MSG ID>, <Data bytes>, since the interpretation interface is constructed according to reduced set of selected signals which optimises performance.	X	X	X
[RS_22_10]	Design	As a configuration procedure, the client passes the following: 1. BUS (ID	X	X	X

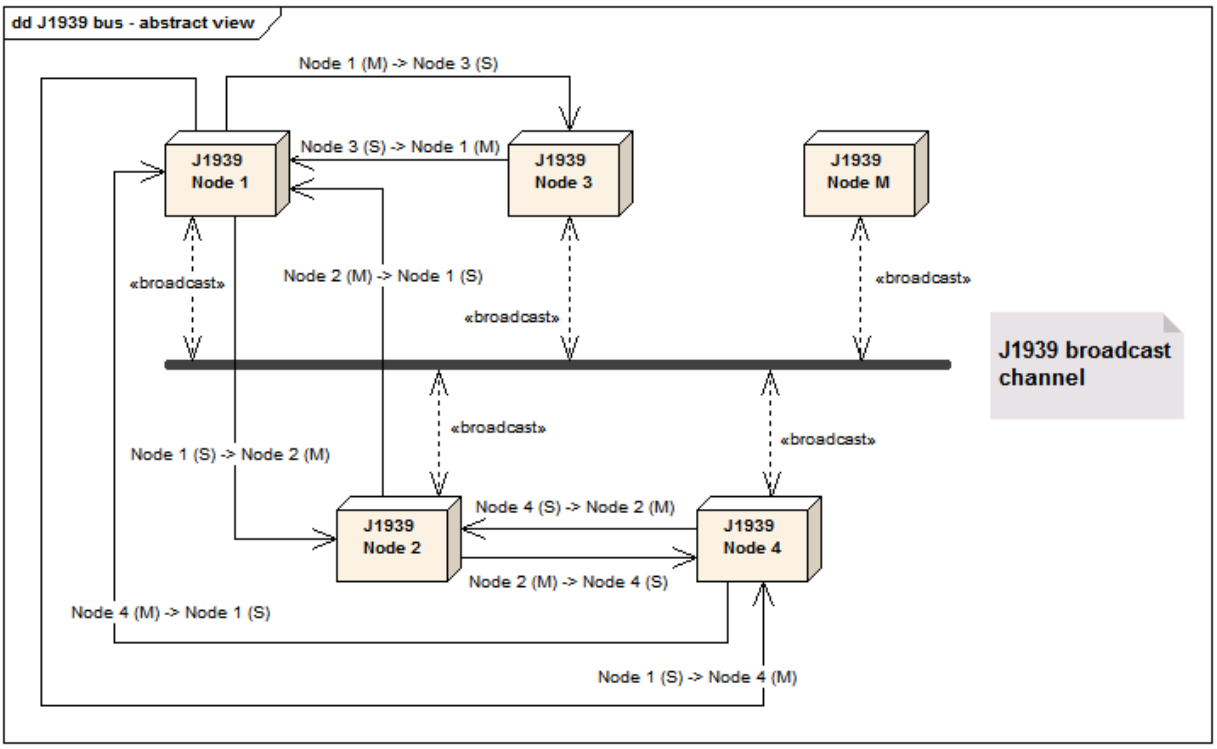
			CAN	FlexRay	J1939
		and Name). 2. Message ID set with their names. 3. For a message, the signal names list and the type of value chosen (raw or engineering).			
[RS_22_11]	Design	The client is responsible for pumping bus data, interpreting the message using the respective bus interpreter and supplying the interpreted values to Signal Graph Window for display.	X	X	X
[RS_22_12]	Design	The common buffer belongs to Signal Graph. It only passes the base class pointer to the client.	X	X	X
[RS_22_13]	Design	There should be another set of configuration data that is purely GUI related. This includes window placement etc.,	X	X	X
[RS_22_14]	Design	Refresh rate should be directly configurable by the user.	X	X	X
[RS_22_15]	Feature	The Signal Graph Window should support two display types for signal: **1.** Normal Line display. **2.** Step Mode display	X	X	X
[RS_22_16]	Feature	All the signals in a Signal Graph Window shall shuttle	X	X	X

			CAN	FlexRay	J1939
		between normal line and step mode display whenever the display type changes.			
[RS_22_17]	Feature	Each signal in Signal Graph Window can be configured to be either Normal Line or Step Mode display.	X	X	X
[RS_22_18]	Feature	The Signal Graph supports cursors for signal tracking.	X	X	X
[RS_22_19]	Feature	The Signal Graph Window should support show / hide by the user.	X	X	X
[RS_22_20]	Performance	If Graph Window is not displayed, then Graph Window module should not claim processor power.	X	X	X

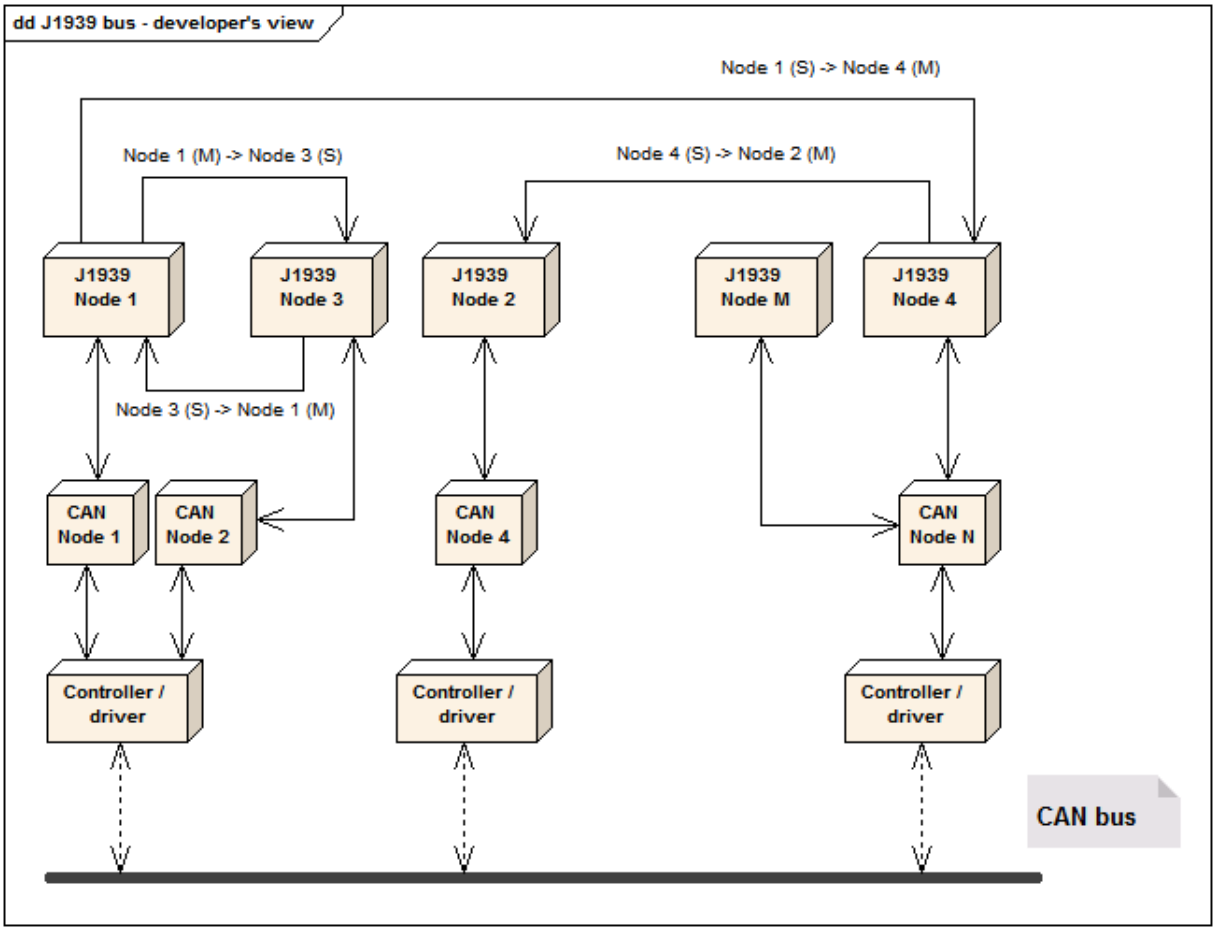
J1939 Basic

Developer Diagrams

J1939 bus - abstract view

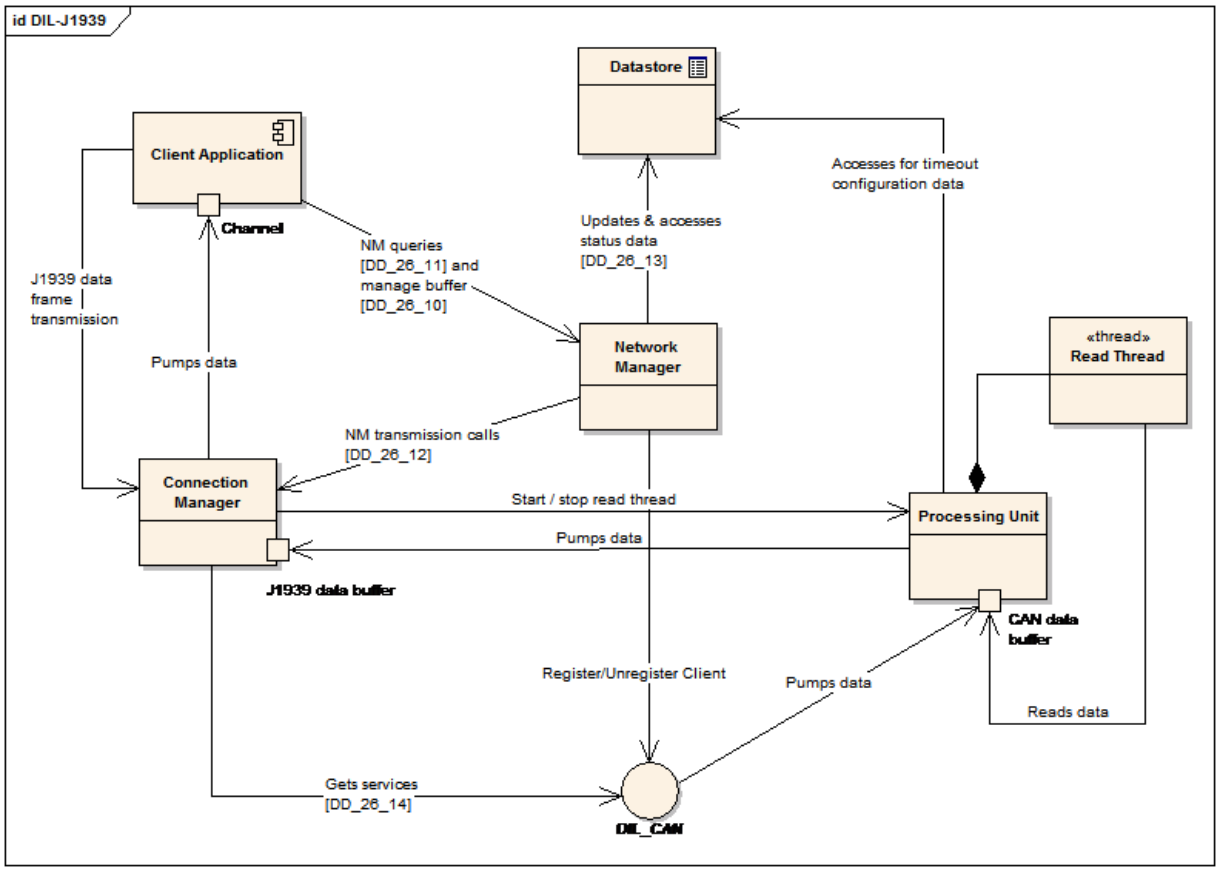


J1939 bus - developer's view



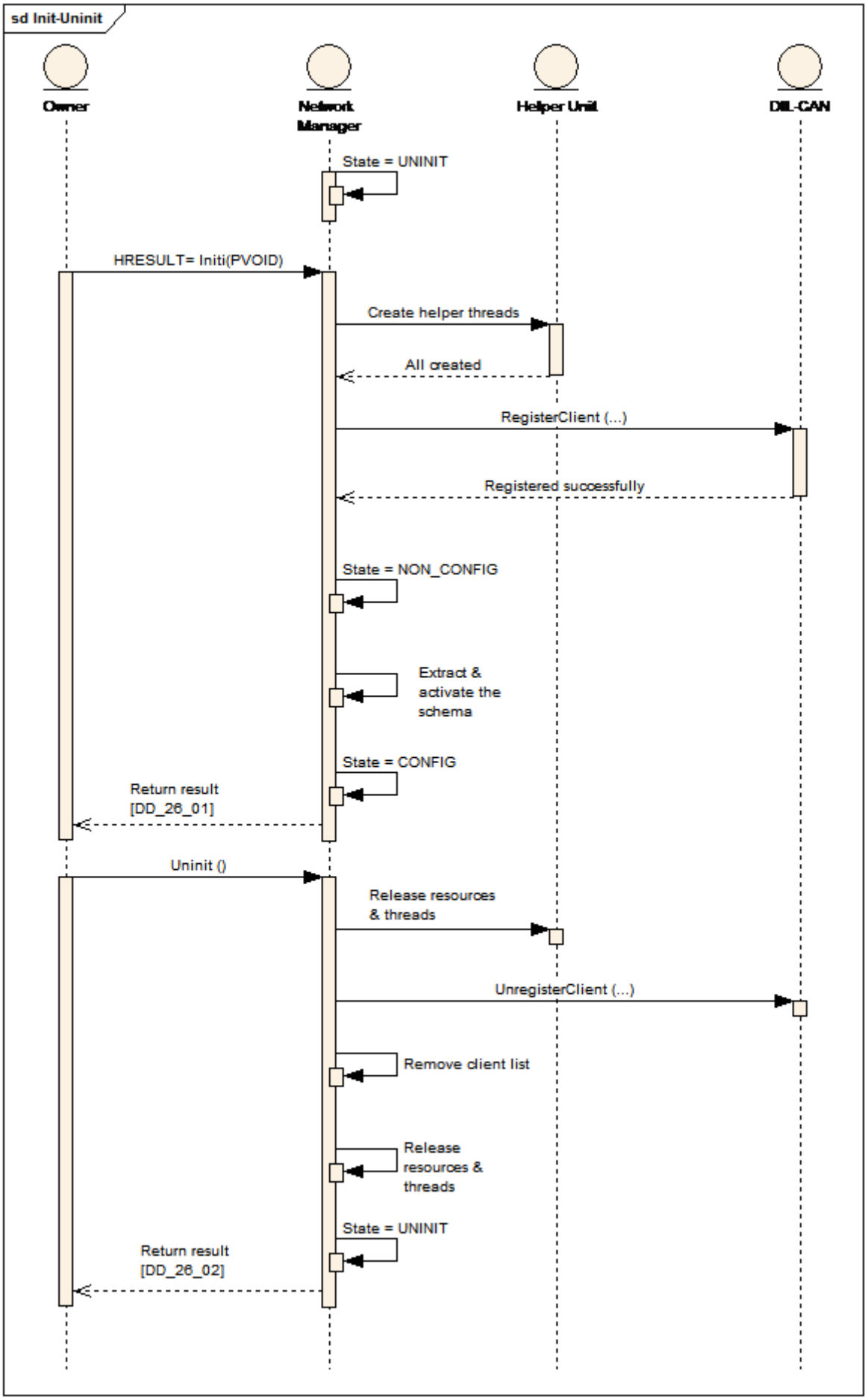
Interface Diagrams

DIL-J1939

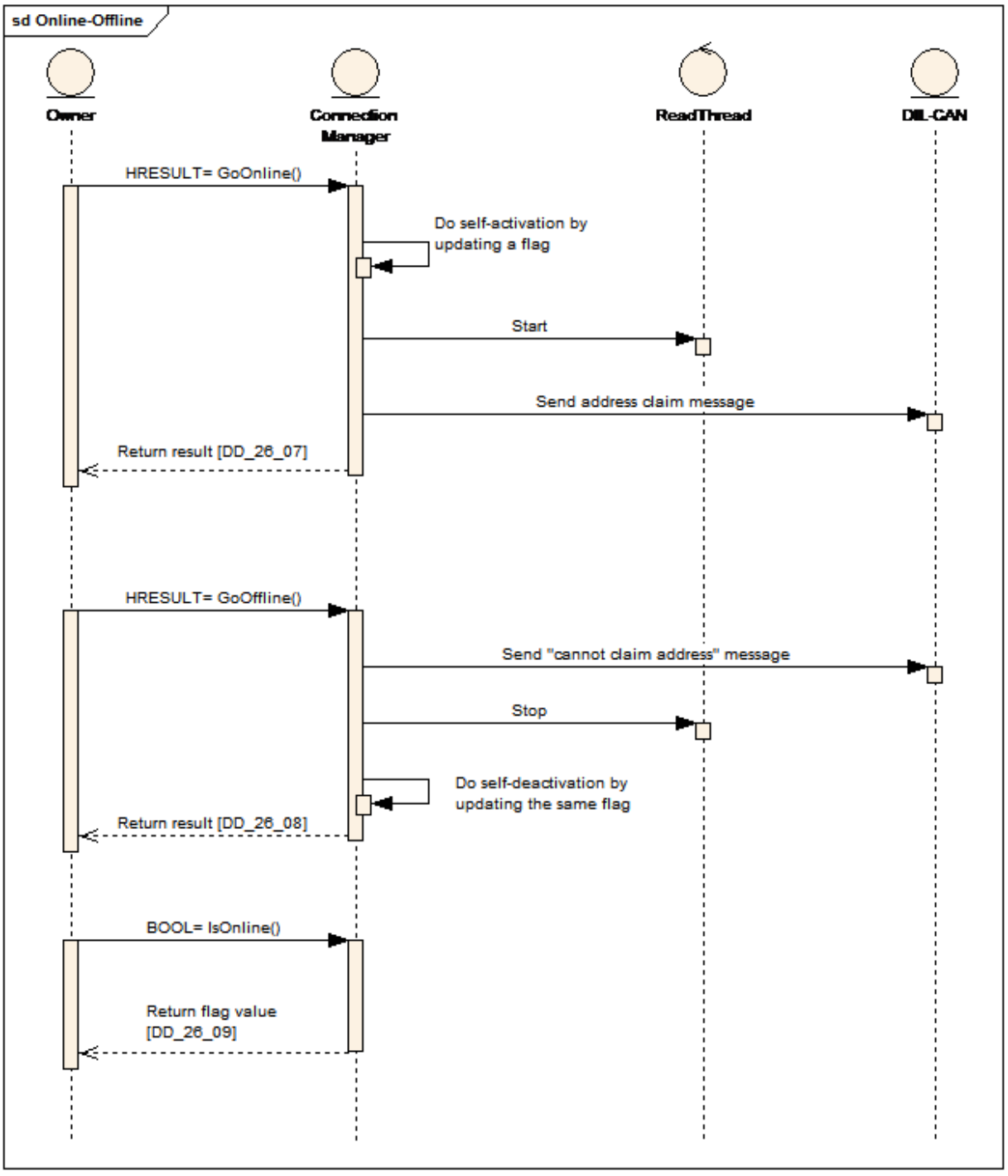


State Diagrams

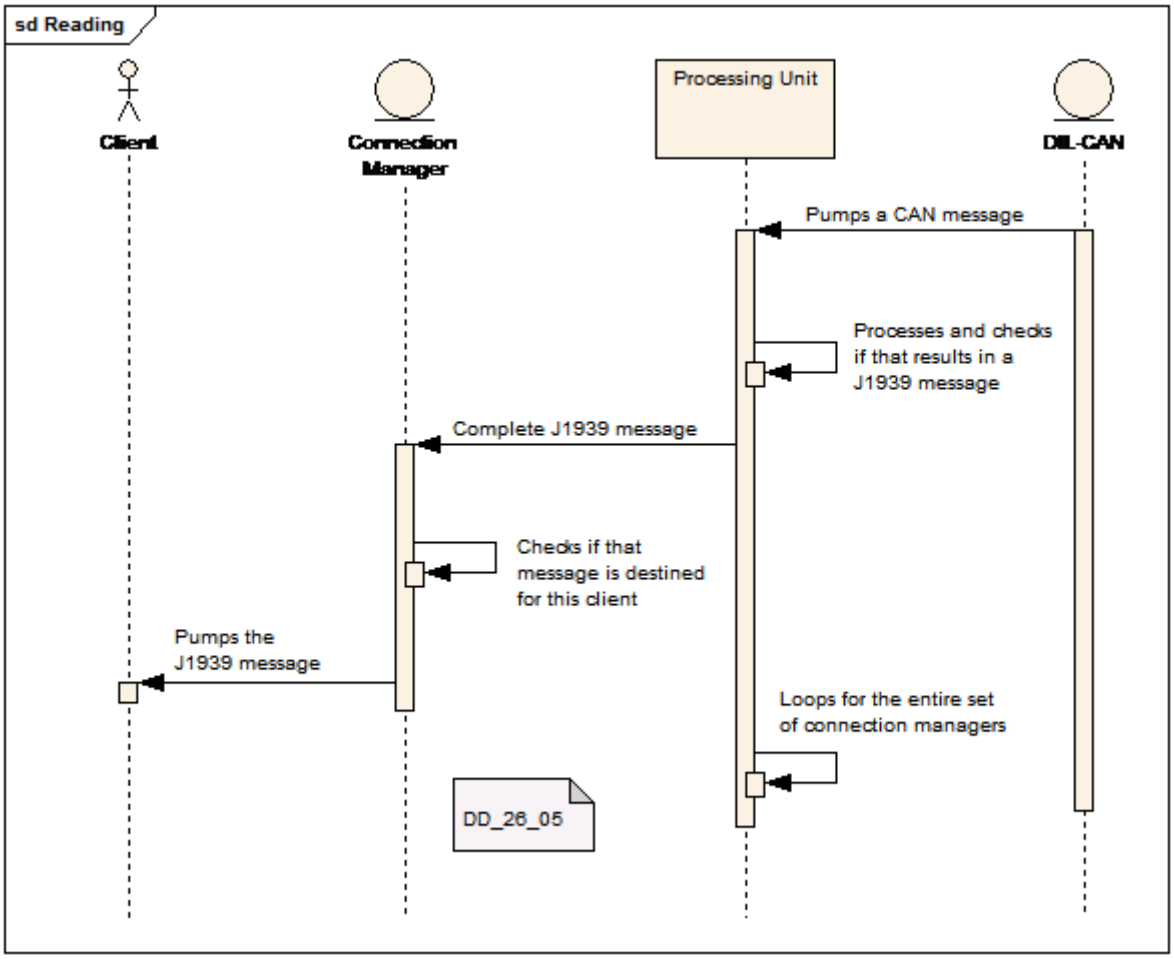
Init-Uninit


sd Init-Uninit
Owner
Network Manager
Helper Unit
DIL-CAN
State = UNINIT
HRESULT= Initi(PVOID)
Create helper threads
All created
RegisterClient (...)
Registered successfully
State = NON_CONFIG
Extract & activate the schema
State = CONFIG
Return result [DD_26_01]
Uninit ()
Release resources & threads
UnregisterClient (...)
Remove client list
Release resources & threads
State = UNINIT
Return result [DD_26_02]

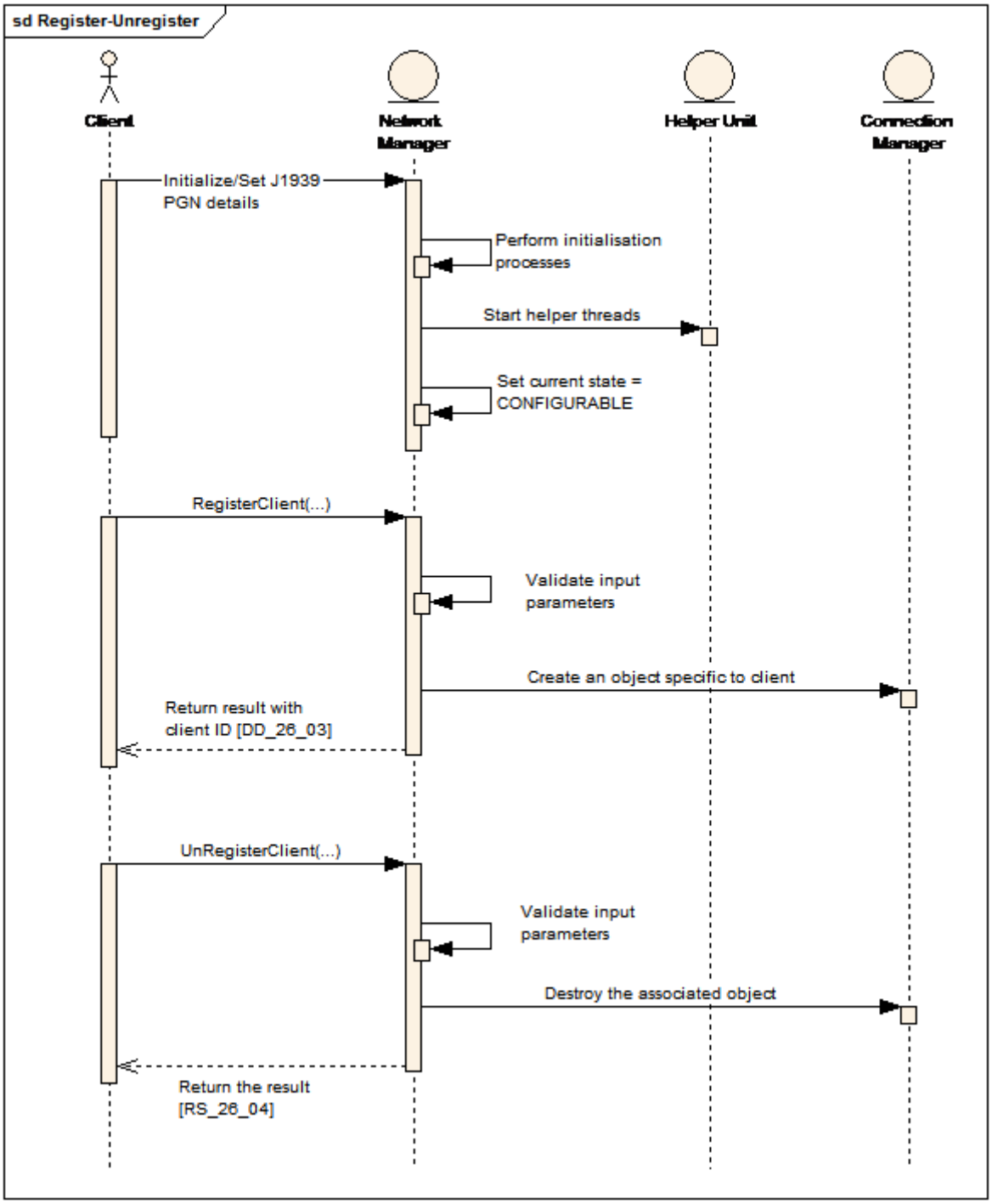
Online-Offline



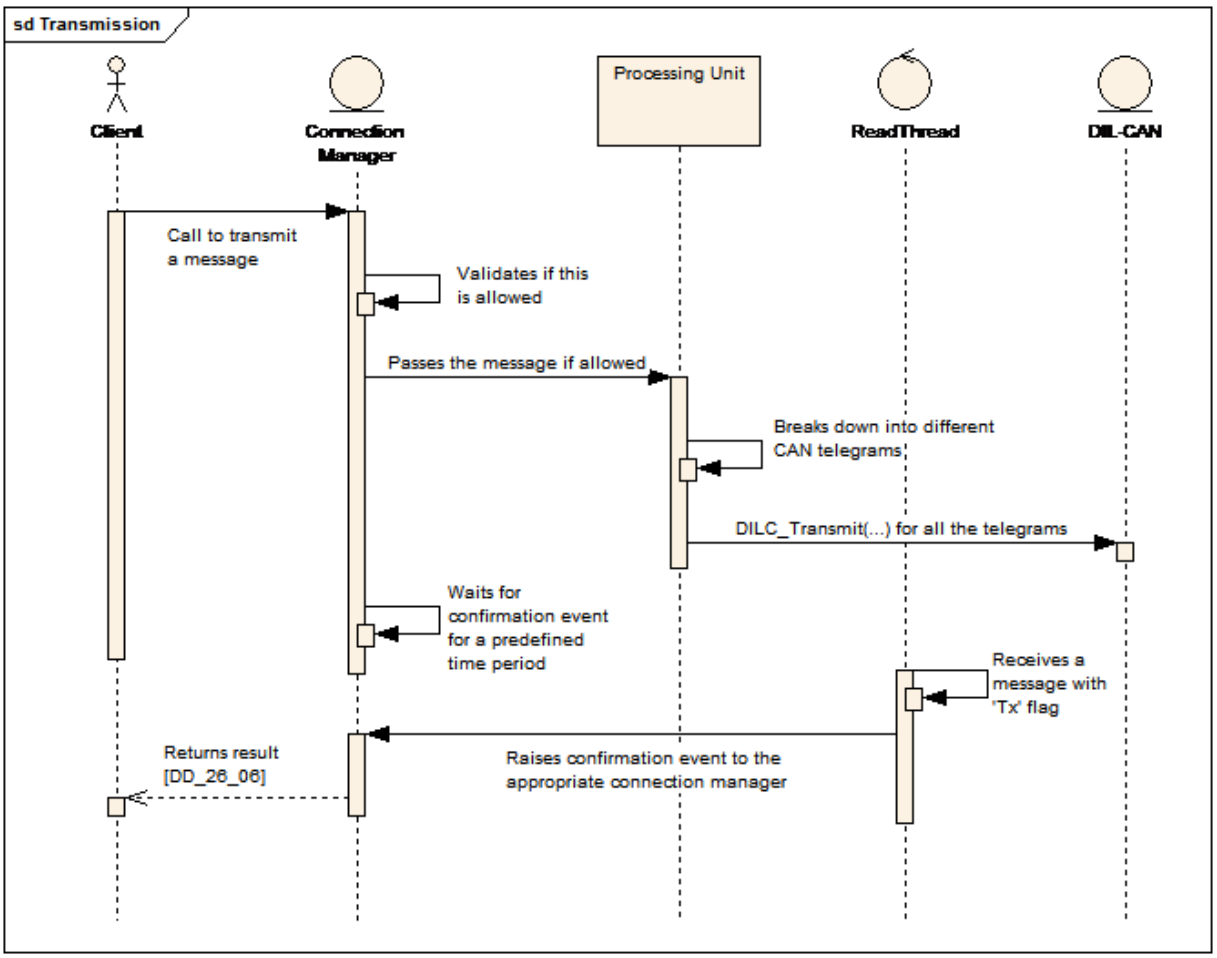
Reading



Register-Unregister

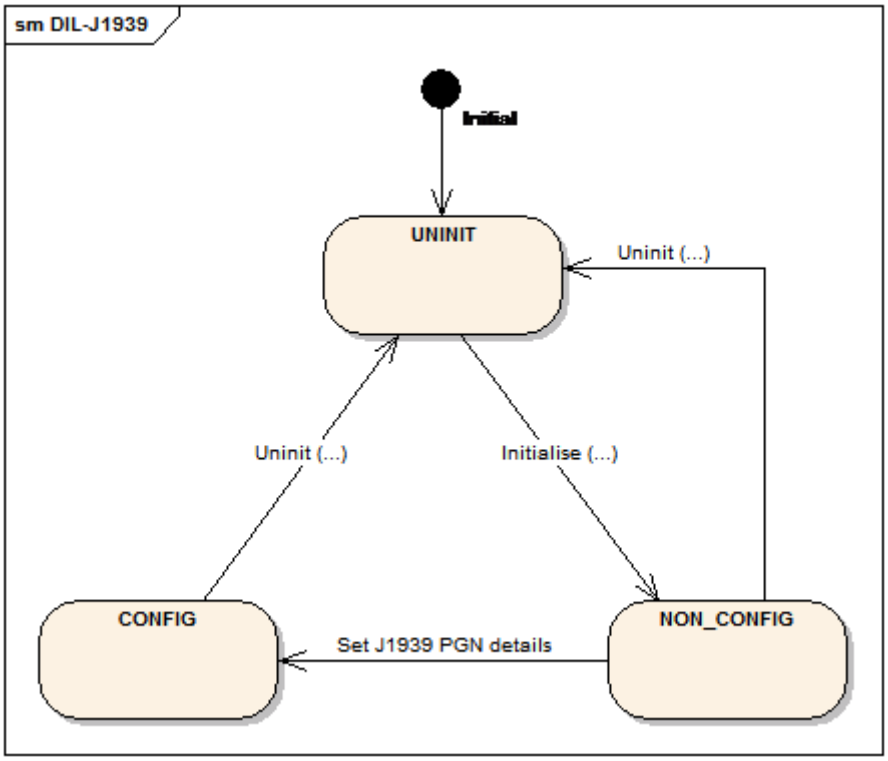


Transmission



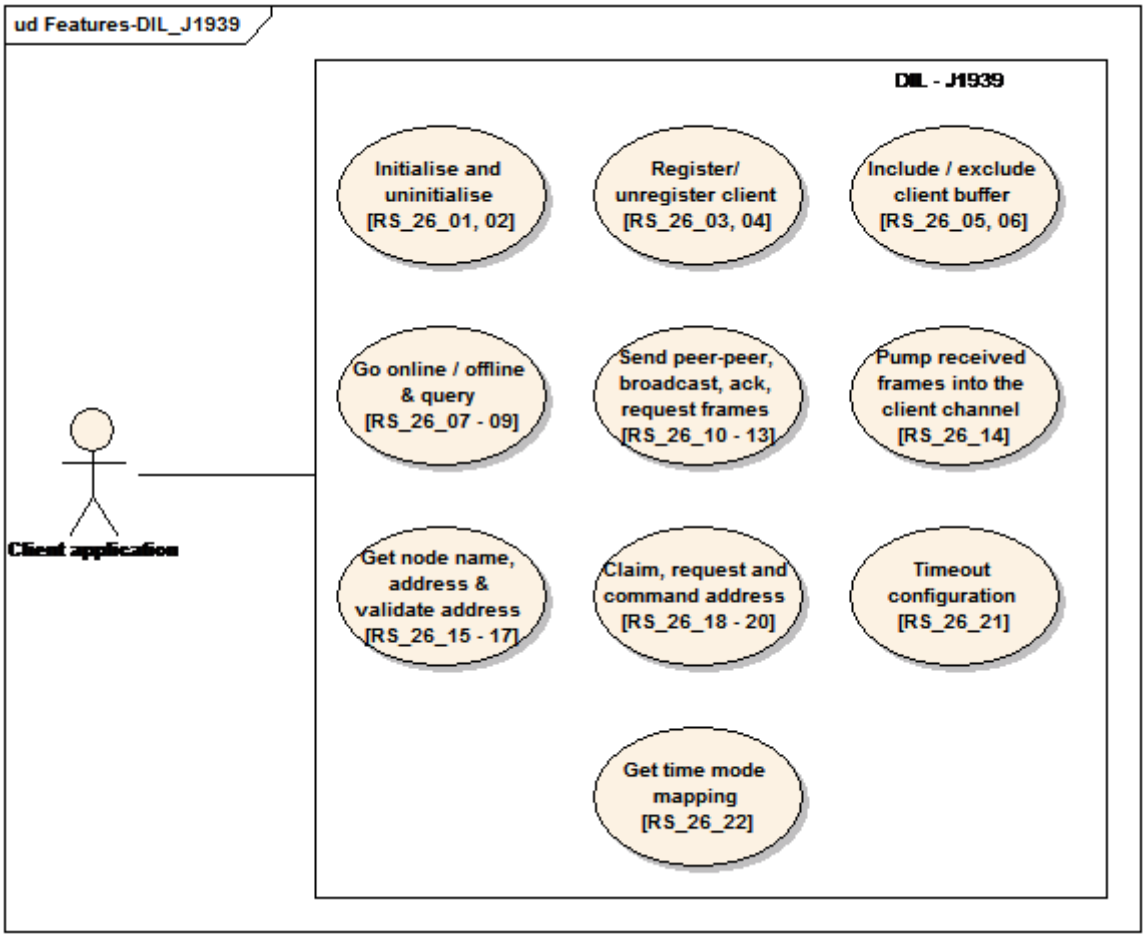
State Machines

DIL-J1939



Usecase Diagrams

Features-DIL_J1939



Requirement Tags

- Initialise and uninitialise [RS_26_01][RS_26_02]
- Register / unregister client [RS_26_03][RS_26_04]
- Include / excluce client buffer [RS_26_05][RS_26_06]
- Go online / offline & query [RS_26_07][RS_26_08][RS_26_09]
- Send peer-peer, broadcast, ack, request frames [RS_26_10][RS_26_11][RS_26_12][RS_26_13]
- Pump received frames into client channel [RS_26_14]
- Get node name, address & validate address [RS_26_15][RS_26_16][RS_26_17]
- Claim, request and command address [RS_26_18][RS_26_19][RS_26_20]
- Timeout configuration [RS_26_21]
- Get time mode mapping [RS_26_22]

Test Automation

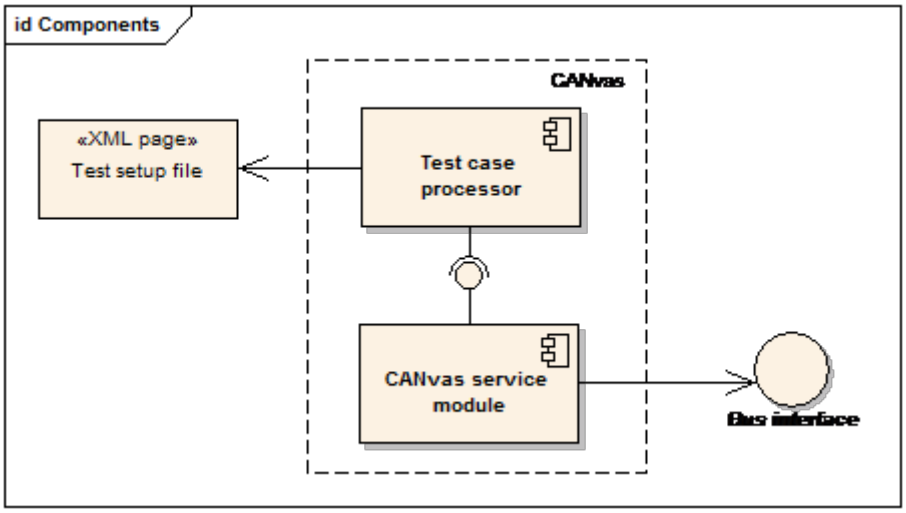
This automates the ECU testing by using certain general test models or patterns. So objective of this is to realize more abstraction in testing where the user only needs to define the test cases that may be taken directly as input parameters for the execution of a testing session carried out by the tool. This means user can expend more time for writing proper test cases rather than implementation issues of the same.

As a prelude, some technical jargons used in the parlance of Test Automation of BUSMASTER are explained first. For obvious reason the different sections of a test setup file shall be most highlighted in the discussion. The different sections are perceived to occur as different nodes of certain respective levels in a tree.

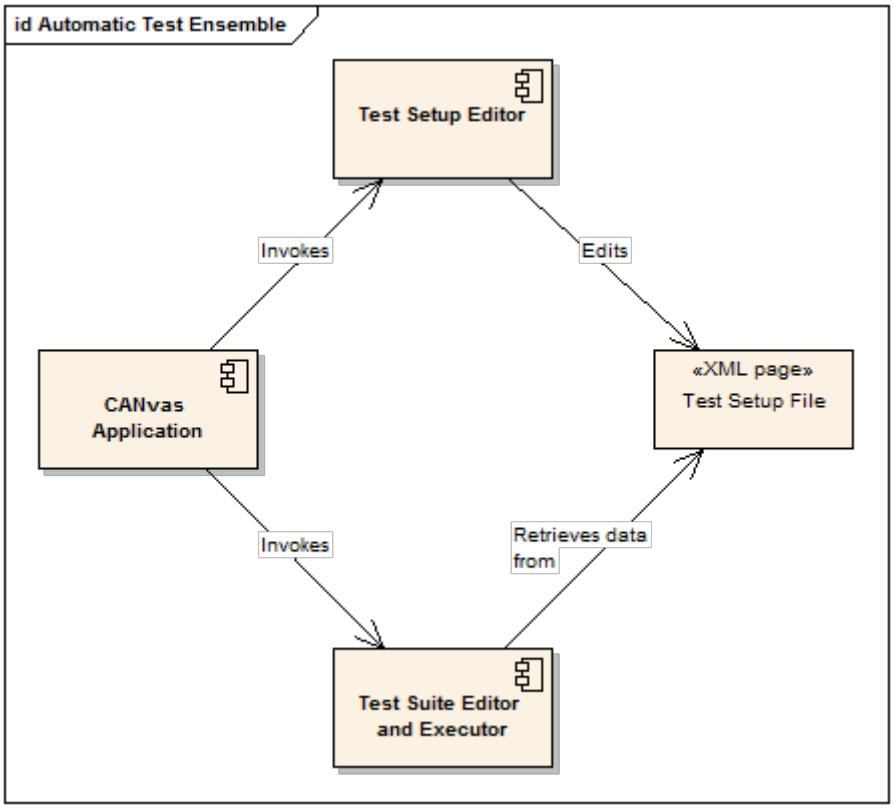
Expression	Explanation	Associated Keyword in Test Setup File
Test Suite	Explanation A collection of test setup. Root of the tree.	-
Test Setup	A collection of test cases	testsetup
Test case	Self-explanatory	testcase
Wait	Instruction to wait for a certain period expressed in milliseconds.	Wait
Replay	Instructs to replay a file.	Replay
Send	Instrction to send a set of messages under this node.	Send
Send_message	Details of the message to be transmitted.	Send_message
Verify	Instruction to carry out a validation operation	Verify
Verify_message	Details of a message verification procedure	Verify_message

General concept

Functional requirement is explained with the following concept diagram of the setup:



In a generic perspective executing a test case consists of transmitting one or more messages, waiting for a suitable response (i.e., one or multiple messages), if received - verifying individual signal values, reporting the result and continue / terminate the process. Test cases are most conveniently defined in a test setup file realised as a .xml file. Clearly, this leads to the evolving of another module which may be used to edit a test setup file. The scenario in toto may be depicted with the following component diagram:



To be noted – this is neither a substitute nor a variant of node programming. In the former the actions are predetermined; it is only the parameter set (both in terms of signal value and time axis) that varies, whereas in the later the node behaviour / logic are programmed. The former is an extension of the tool to simplify the process of carrying out tests of a genre and generating the report.

This calls for describing the three namely, Test Setup file, Test Setup Editor and Test Suite Editor and Executor.

Test Setup File

A test setup is realised as a XML file fed to the test case processor, which contains the instruction set of the various test cases.

Below is the grammar of the test setup file -

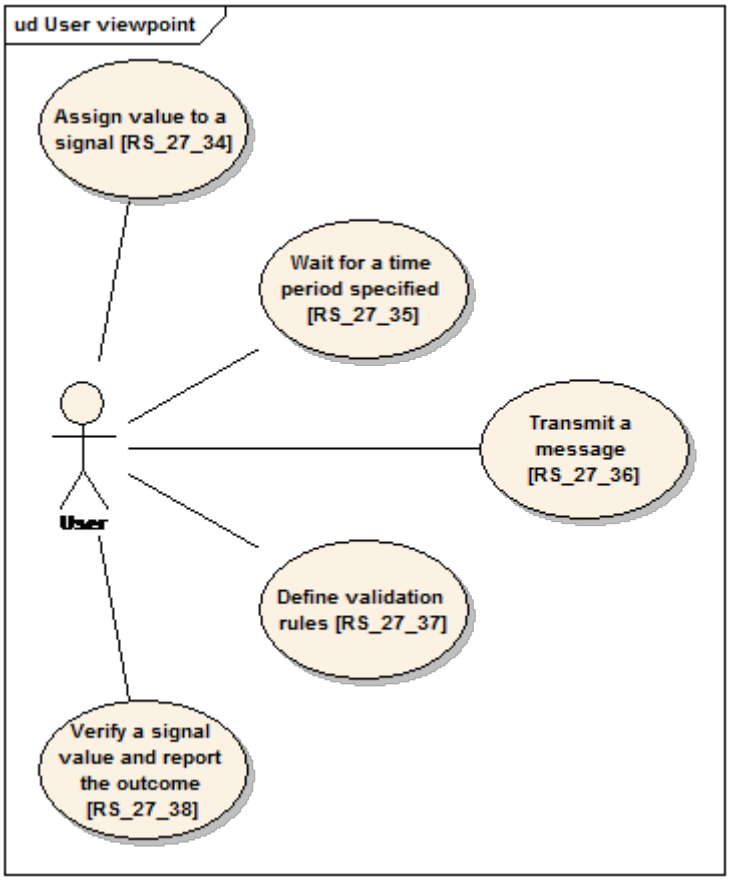
Test setup := <Header> + <Test case list>

Test case list := {test case 1, test case 2, …, test case N}, where N >= 1

Test case := {test step 1, test step 2, …, test step N}, N >= 2

Test step := transmission / wait / replay / verification operation

The below use case diagram shows the most basic operations needed to run a test case.



1. Transmission of a message: There can be three modes of message transmission namely, mono-shot transmission, cyclic transmission and transmission for a specific number of times. The signals not been specified for the initialisation shall have the default initialisation of 0. [RS_27_36]
2. Waiting: The time period shall be accepted as milliseconds. [RS_27_35]
3. Set value of a signal: User shall be entitled to set values of a signal both in engineering value mode and raw value mode. This shall be made possible per test case basis. If nothing is specified, the default mode shall be assumed. [RS_27_34]
4. A signal must need a message which it is a part of and shall be referred to. The message name should be specified before the signal name. [RS_27_38]
5. Validation conditions: The user needs to formulate the validation condition. In the formula current signal value shall be denoted by ‘x’ following the algebraic notation. The presently supported logical operators are the eight: ==, >, <, >=, <=, !=, ||, and && [RS_27_37]

By combining them suitably the following validation operations may be carried out:

1. Range of values; e.g., (x <= 10) && (x >= 50)
2. Set of discrete values; e.g., (x == 10) || (x == 20) || (x == 50)

3. Formulation of any other validation procedure.

Here goes a sample test setup file.

The table below contains a concise description of each of the section details touching upon the exceptional condition procedures.

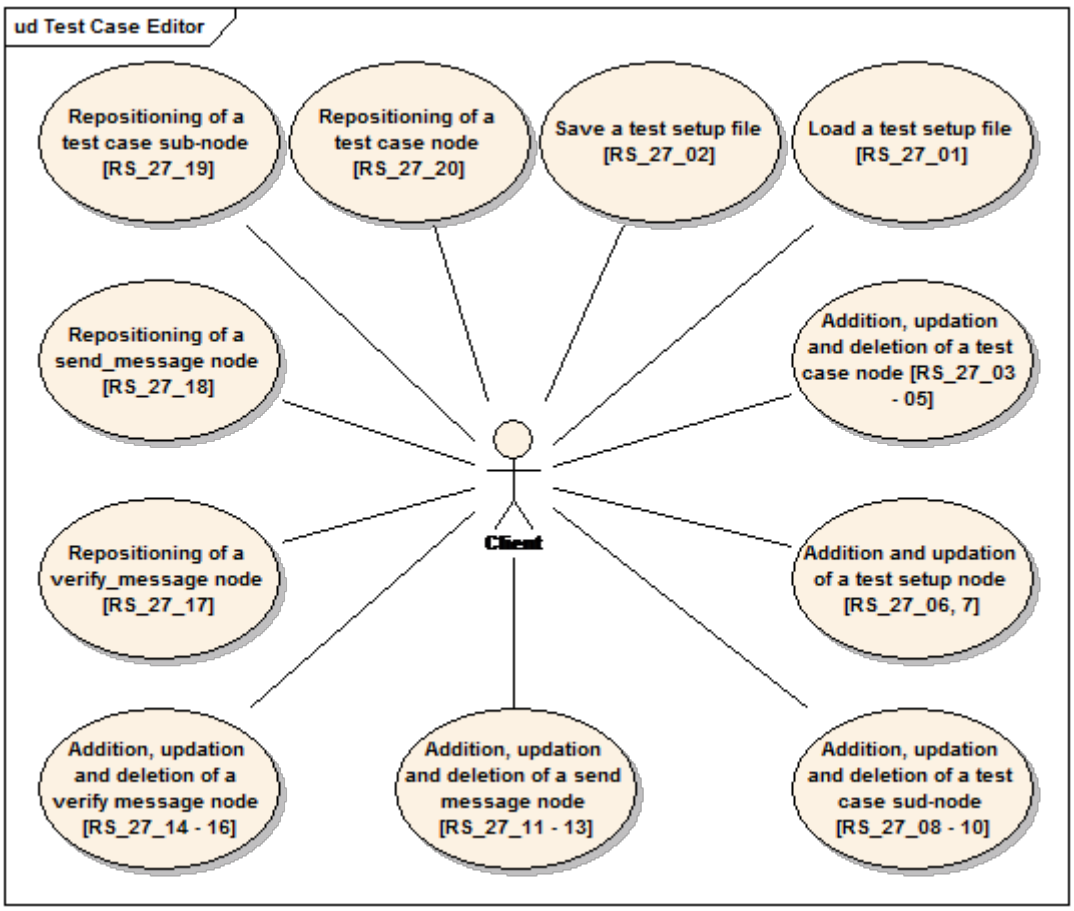
Table 1: 2

Section (tag)	Description	If absent, Assumed value	If absent, Error condition
Test setup (testsetup)	The root node. Accompanied by its title (title) and version information (version)	-	Fatal [RS_27_39]
Header (header)	Header section	-	Fatal [RS_27_40]
Database file (database)	Database file		Fatal [RS_27_41]
Version (version)	Version information of the test setup	-	No error [RS_27_42]
Module Name (module name)	Module focused on for the testing	-	No error [RS_27_43]
Engineer's name	Test engineer’s name	-	No error [RS_27_44]
Engineer's role	And role / designation	-	No error [RS_27_45]
Report file path (path)	Particular of the report file to be generated.	Current working directory with the name same as the test setup.	Error [RS_27_46]
Report file format (format)	Format of the report file. Can be one of TXT and HTM	TXT	Warning [RS_27_47]
Report file time mode (timemode)	Time mode. Can be one of SYS (system), REL (relative) and ABS (absolute)	SYS	Warning [RS_27_48]
Bus type (bustype)	Bus type. At present can be only CAN	-	Fatal [RS_27_49]
Test case list (list_of_test_cases)	Collection of test cases.	Nil	Error. There must be at least one entry. [RS_27_50]
Test case (testcase)	Collection of test steps. A test case contains identifier, title and exception handler. The last one instructs if in case of failure to continue or exit.	Nil	Error. There must be at least one entry. [RS_27_51]
Test case identifier (identifier)	Identifier of the test case. Must be unique.	Nil	Error. There must be an identifier. [RS_27_52]
Test case title (title)	Test case title	-	No error [RS_27_53]
Test case exception handler (exp_handler)	Instructs if in case of failure the testing process exists or continues. Can be one of continue or exit	continue	Warning [RS_27_54]

Section (tag)	Description	If absent, Assumed value	If absent, Error condition
Transmission (send)	Collection of the messages to be transmitted.	-	No error [RS_27_55]
Transmittion message details (send_message)	Details of the message list to be transmitted.	Nil	Warning [RS_27_56]
Send message id (identifier)	Identifier of the message.	-	Error. The test case shall be dropped. [RS_27_57]
Send message unit (unit)	Unit type of the signals. Can be either raw (raw) or enginneering (engg)	Engg	Warning [RS_27_58]
Signal (signal)	Details the signal with its name (name) and value.	-	Error. The test case shall be dropped. [RS_27_59]
Verification (verify)	Verification instruction set. Contains a collection of verification messages.	Nil	Error. A test case must have a validation routine. Test case shall be dropped. [RS_27_60]
Failure classification (failure)	For a verification procedure – how to classify validation failure. Can be one of warning, error, fatal	Error	Warning. [RS_27_61]
Verify message details (verify_message)	Details of the message list to be verified / validated.	Nil	Error. The test case shall be dropped. [RS_27_62]
Verify message signal detail (signal)	Details of a signal under a verify message node. The attribute required is the signal's name (name). The node value shall be a string with formulation of the condition. This shall follow the syntax mentioned in table 2.	Nil	Error. The test case shall be dropped. [RS_27_63]

Test Setup Editor

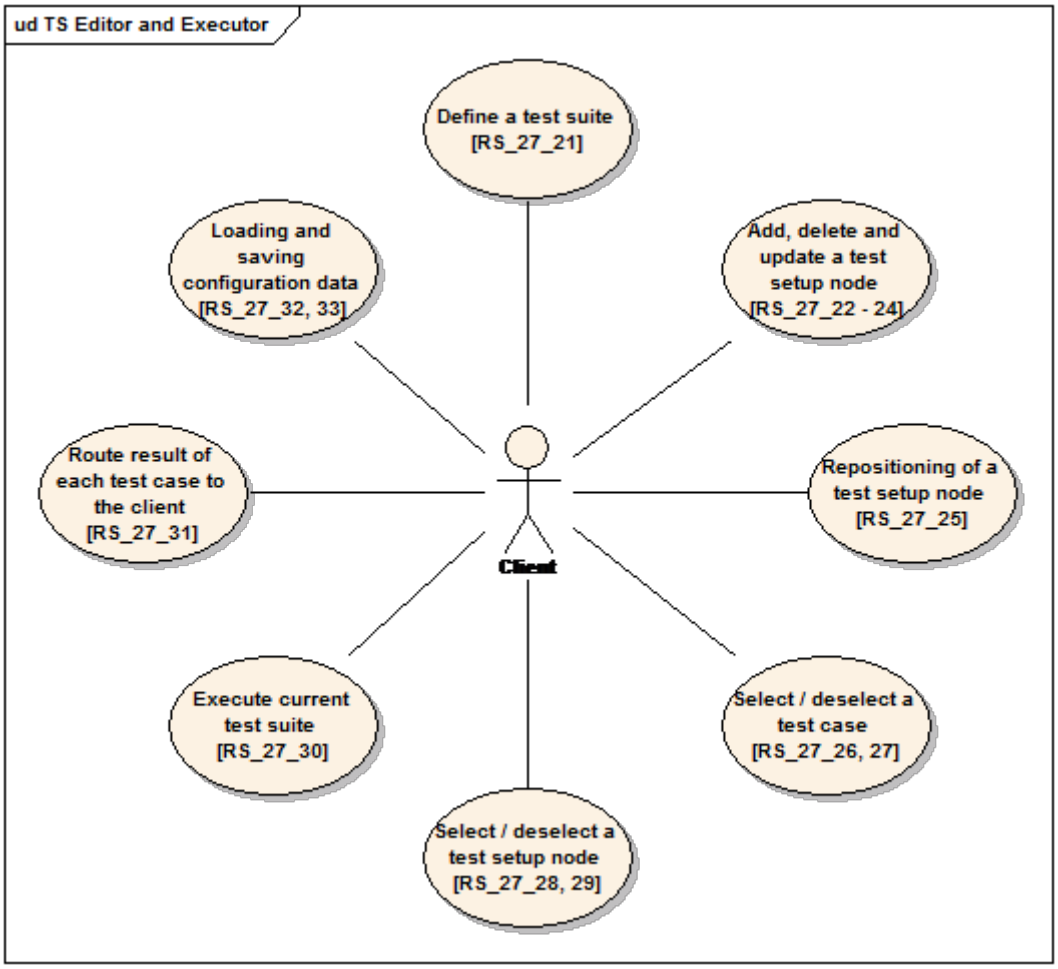
This shall be used to create and edit a test setup file. We begin by presenting a use case diagram describing the different services rendered by this module:



- Load a test setup file [RS_27_01]
- Save a test setup file [RS_27_02]
- Addition, updation and deletion of a test case node [RS_27_03][RS_27_04][RS_27_05]
- Addition and updation of a test setup node [RS_27_06][RS_27_07]
- Addition, updation and deletion of a test case sub-node [RS_27_08][RS_27_09][RS_27_10]
- Addition, updation and deletion of a send message node [RS_27_11][RS_27_12][RS_27_13]
- Addition, updation and deletion of a verify message node [RS_27_14][RS_27_15][RS_27_16]
- Repositioning of a very_message node [RS_27_17]
- Repositioning of a send_message node [RS_27_18]
- Repositioning of a test case sub-node [RS_27_19]
- Repositioning of a test case node [RS_27_20]

Test Suite Editor and Executor

This defines / edits a test suite and can run the test setups included under the test suite. The test suite may be saved as a part of the configuration file and accordingly may be retrieved from the same. The services to be rendered by this module are presented by the following use case diagram:



- Define a test suite [RS_27_21]
- Add, delete and update a test setup node [RS_27_22][RS_27_23][RS_27_24]
- Repositioning of a test setup node [RS_27_25]
- Select / deselect a test case [RS_27_26][RS_27_27]
- Select / deselect a test setup node [RS_27_28][RS_27_29]
- Execute current test suite [RS_27_30]
- Route result of each test case to the client [RS_27_31]
- Loading and saving configuration data [RS_27_32][RS_27_33]

Bus statistics window

Objective of this module is to monitor the bus status in terms of the detailed statistical data of frames for each channel and also the controller. So the monitored data set must be chosen in such a manner that together they mirror the necessary bus status information [RS_24_01].

Bus statistics data are available for all the channels the application presently is accessing [RS_24_02]. Both directions (reception and transmission) are considered [RS_24_03]. The information set consists of the following types of information namely, bus data [RS_24_04], bus status [RS_24_05], bus load status [RS_24_06] and controller status information [RS_24_07].

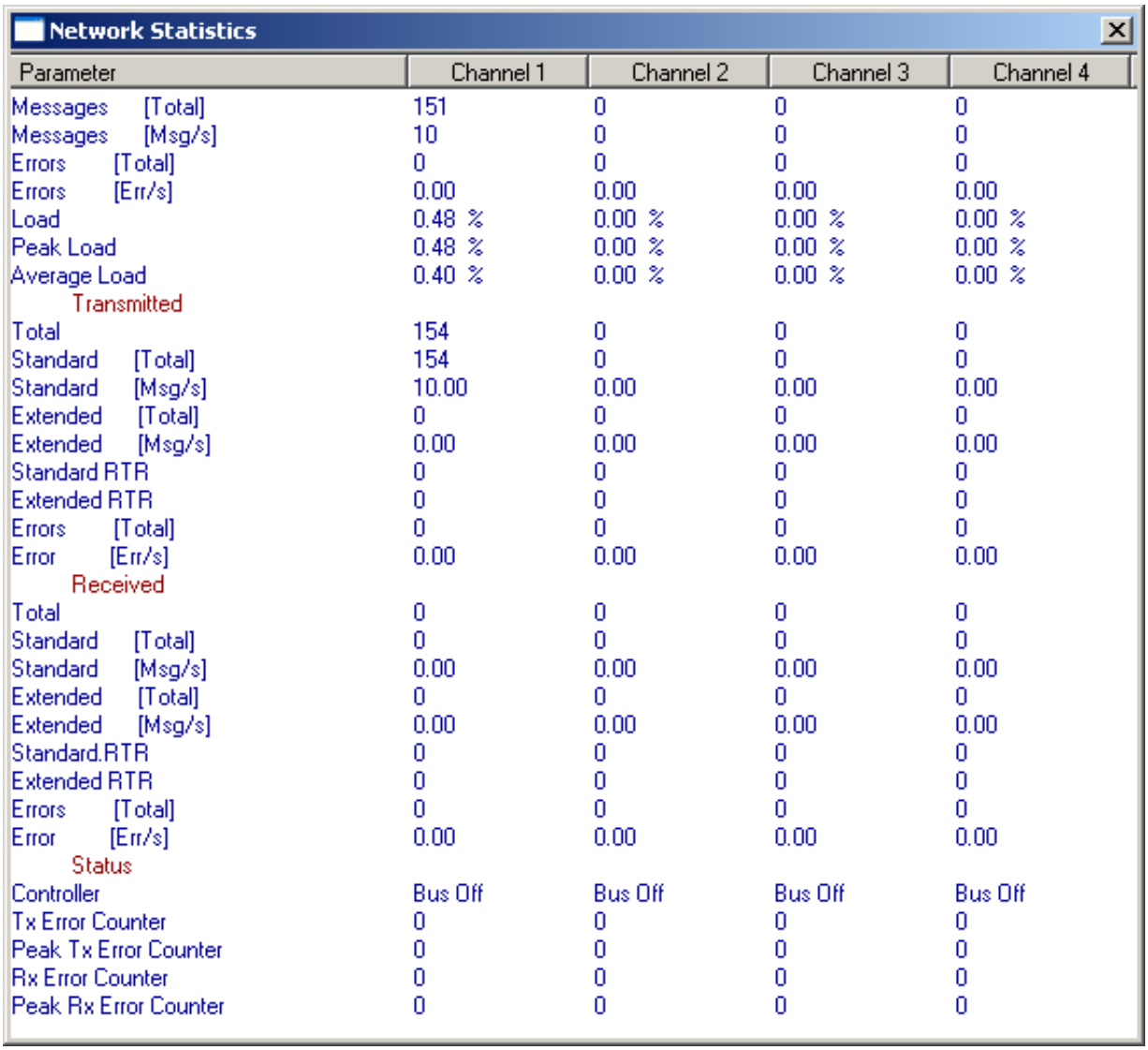
For individual bus standard the three aforementioned kinds of information shall be detailed.

CAN

For bus data standard [RS_24_08], extended [RS_24_09] and RTR [RS_24_10] frames are considered. Error frames occurs under the purview of status data [RS_24_11]. Bus load status data set includes current load [RS_24_12], peak load [RS_24_13] and average load [RS_24_14]. Measurement period always begins (or resets) from the time of connection [RS_24_15]. For each of the above, both total [RS_24_16] and present moment data shall be presented [RS_24_17]. Controller status data covers present controller status [RS_24_18], Tx Error

Counter (both peak and present) [RS_24_19][RS_24_20] and Rx error counter (both peak and present again) [RS_24_21][RS_24_22].

The user interface specification for CAN is shown below:



The above diagram shows a bus statistics window for CAN. The title bar shall indicate the bus standard [RS_24_23] and the refresh rate configurable [RS_24_24].

Installer

The installer manages setting up the deployment phase where all executables, libraries, components and other relevant artifacts are copied into the target system to make the application suite operational. Installation activities occur in three distinct modes namely, new installation, maintenance (modification / repairing) and uninstalling.

Installations are of different types - ‘restricted’, ‘free’ and ‘evaluation’. The first one is controlled by suitable keys from vendor, ‘free’ installation and ‘evaluation’ where a validation period for running the application suite is applied.

Below goes the detailed tagged specification.

			CAN	FlexRay	J1939
25		Installer			
[RS_25_01]	Feature	Installation doesn't proceed	X	X	X

			CAN	FlexRay	J1939
		if the target platform isn't indicated in the target platform support matrix.			
[RS_25_02]	Feature	Installation doesn't proceed if the user doesn't have the required access rights.	X	X	X
[RS_25_03]	Feature	Accepts the target location for the installing application suite	X	X	X
[RS_25_04]	Feature	Copies all the relevant components in the required location.	X	X	X
[RS_25_05]	Feature	Registers all the COM components.	X	X	X
[RS_25_06]	Feature	In case the target system doesn't have some necessary system libraries which are freely redistributable, the same shall be copied.	X	X	X
[RS_25_07]	Feature	In case this is a commercial application suite, license key shall be mandatory to go ahead with the process.	X	X	X
[RS_25_08]	Feature	A license key is for single usage only.	X	X	X
[RS_25_09]	Feature	Commercial application suite validates the installation authenticity before starting.	X	X	X
[RS_25_10]	Feature	The installation application shall be usable for new installation, repairing and uninstallation.	X	X	X

			CAN	FlexRay	J1939
[RS_25_11]	Feature	Repairing and uninstallation shall be possible from Control Panel too.	X	X	X
[RS_25_12]	Feature	Uninstallation shall completely remove the whle application suite.	X	X	X
[RS_25_13]	Feature	Uninstallation process generates an uninstallation key which can be treated as a proof of that activity.	X	X	X
[RS_25_14]	Feature	For evaluation copy, the licence key dictates the validation period	X	X	X
[RS_25_15]	Feature	Application suite stops working if the validation period is over	X	X	X
[RS_25_16]	Feature	It shall be possible to extend the validation period with another suitable license key	X	X	X
[RS_25_17]	Design	All keys (activation, license, uninstallation) shall not be comprehensible by normal means.	X	X	X
[RS_25_18]	Design	Validation period shall not be extendable by end user by means other than a suitable license key.	X	X	X

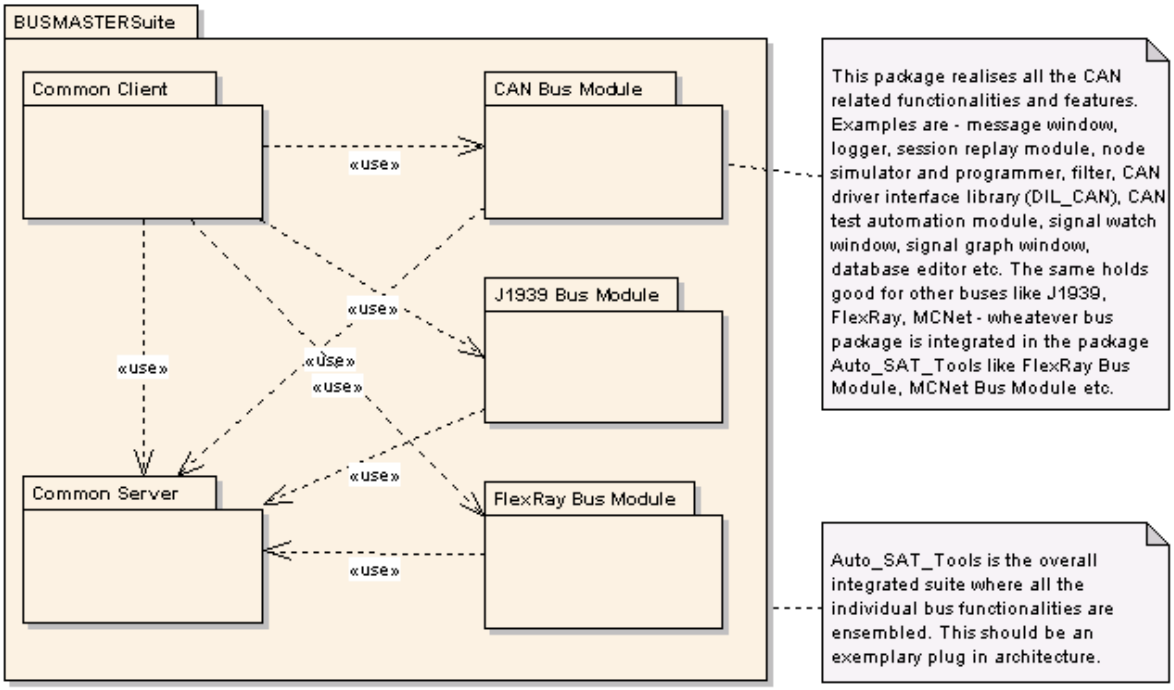
User Interface Requirements

General Requirements

We aim to find out the best user interface for BUSMASTER towards supporting multiple automotive buses under the same application.

While exploring in the multi-bus dimension, we restrain the buses as three namely, CAN, J1939 and FlexRay. The rationale behind this list is that CAN and FlexRay are totally different from each other directly from architecture and motivational viewpoints and J1939 is a CAN-based automotive bus standard. This exemplification shall help in validating the proposed design and user interface to address the issue of integrating bus standards which are completely different from each other and also when one is a higher layer protocol over a fundamental one.

We begin by looking at application package from a most obvious point of view in which it is clearly an ensemble (or rather confederation) of a number of sub-packages, each specific to a bus. Naturally, there shall be two additional software modules.

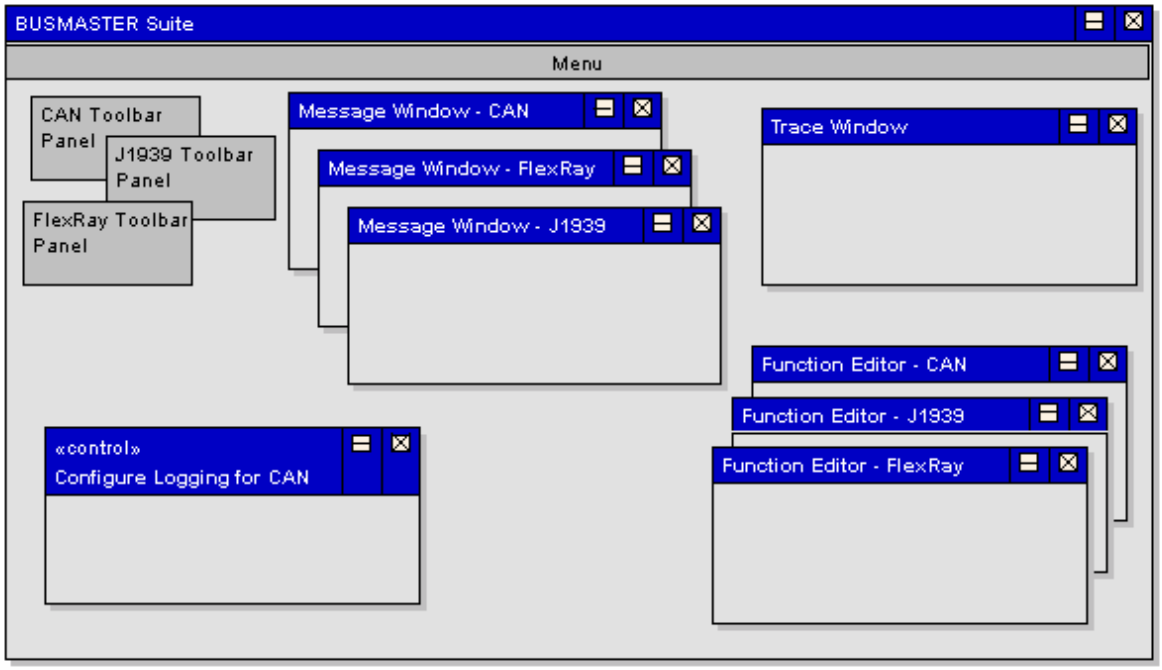


The first one of them is the common client and the second one is the common server that everyone can access. The services may be utilized both at compile time and run time levels.

We now proceed on by redefining BUSMASTER tool family.

“BUSMASTER suite is fundamentally something which provides services to the user for the buses that it presently supports. The services may be rendered in the form of GUI and a set of API functions. For the later, an interface for a particular bus service needs to be queried. For the former, GUI services are manifested in terms of toolbars, menu items and output can be seen on certain windows.”

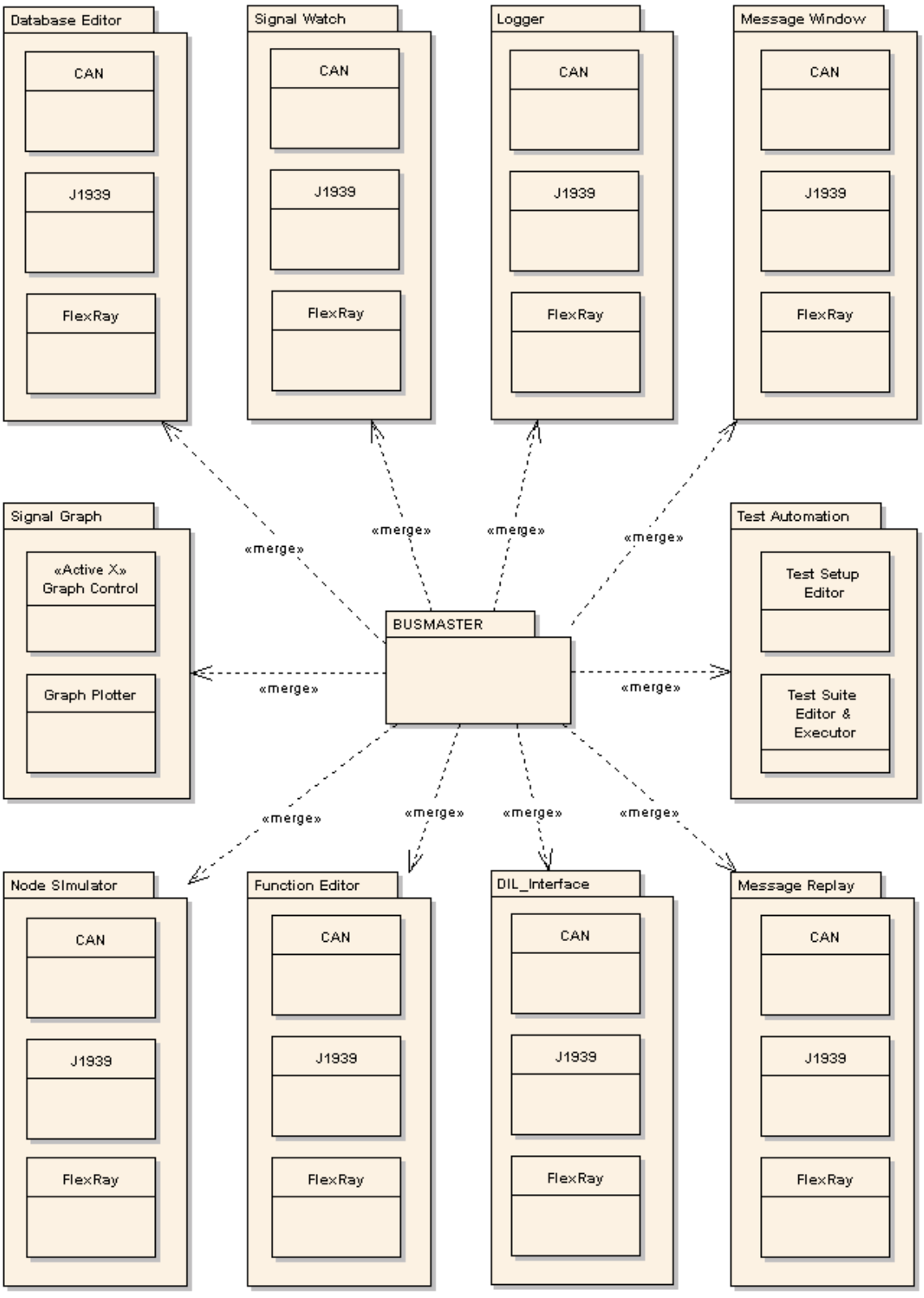
This definition results in conceptualization of the following example GUI.



The GUI is divided broadly into two categories – GUI controls specific to a bus and common GUI controls. Examples of the former are message window, function editor, toolbar panel. Instances of the later are logging configuration dialog box, trace window etc. Moreover, there is the third category – the integrated GUI control. Menu bar is the only control falling under this category. Menu bar combines the entire menu items specific to a bus. Hence,

{ BUSMASTER Menu Bar } = { CAN menu items } U { FlexRay menu items } U { J1939 menu items }

The overall system can be expressed in with the following package diagram.



Below shown a sample menu bar:

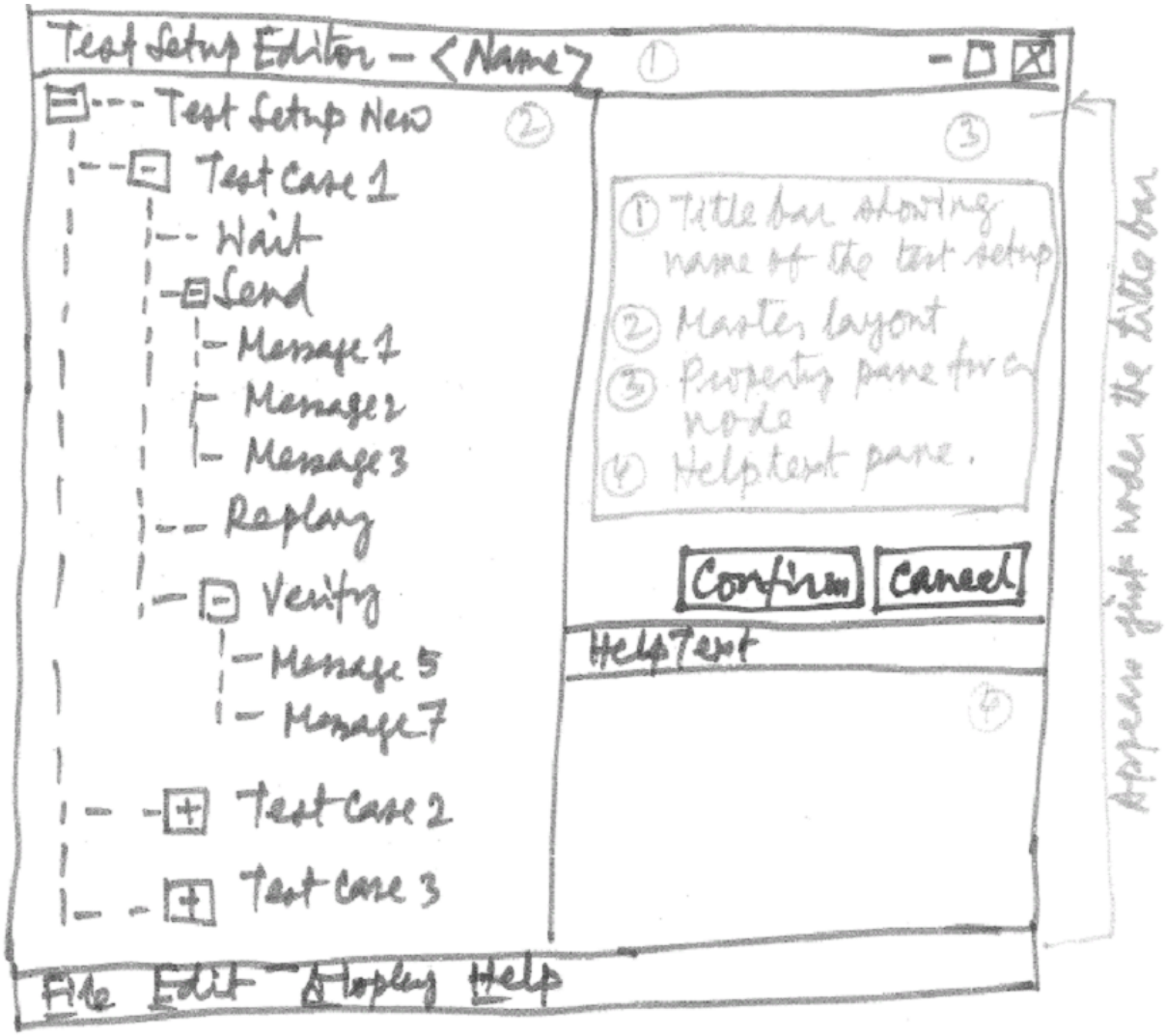
Current Bus	Configure	Database	Function Editor	User Program
CAN ✓	Message Display	New	New	Load All
FlexRay	Controller	Open	Open	Unload All
J1939	App Filters	Save	Save	Build & Load All
	HW Interface ⇨	Save As	Save As	Build All
	Log	Save and Import	Close	Execute ⇨
	Replay	Close	Undo	
	Tx Messages	Associate database	Cut	
	Simulated Systems	Dissociate database	Copy	
	Waveform Messages		Paste	
	Signal Graph Window			

Since most of the functionalities / features of a module is the same as another one, assigning separate menu item for each of them shall result in overpopulation and hence incomprehensibility in the menu items. That's why the user can select the current active bus for all the manual activities like configuration, node programming, signal graph configuration etc. The only apparent disadvantage of such a user interface is that the user will be unable to carry out such operations simultaneously for more than one bus.

Test Automation

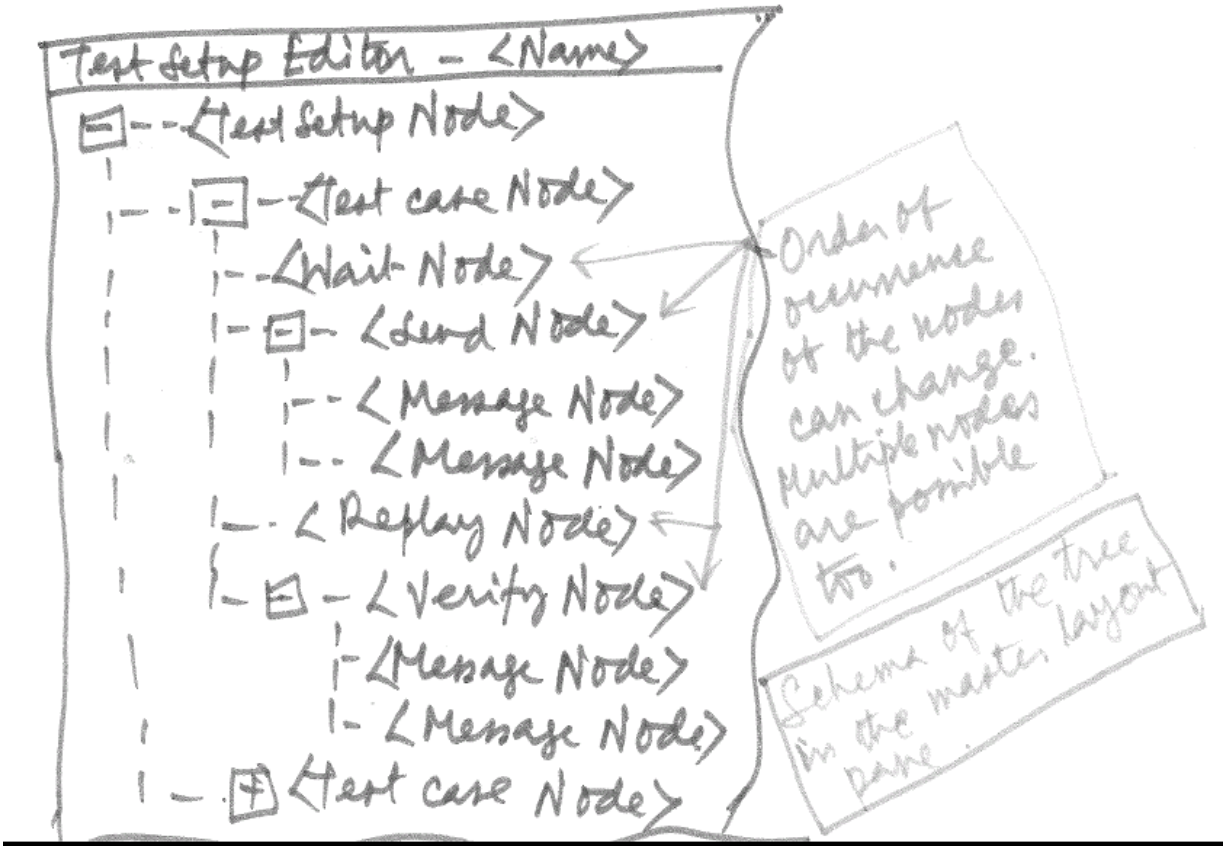
Test Setup Editor GUI

Below goes a schematic view of Test Setup Editor GUI:



Clearly, the client area has been divided into three panes. The leftmost is the one that shall be used to update the nodes whereas the upper right pane shall be used to modify a node parameters. The former one is called master layout pane or simply master pane. The later one shall be termed as the property pane for a node or concisely, a property pane. Roles of the two buttons "Confirm" and "Cancel" in the property pane are obvious from their names. Node parameters may be modifiable by a generic list control in property pane, where each of the row entries correspond to a node parameter. The third pane names as helptext pane shall show some tips / brief explanations on the functionality of the node / control selected by the user.

Here goes a short overview on the nodes in the master pane ~

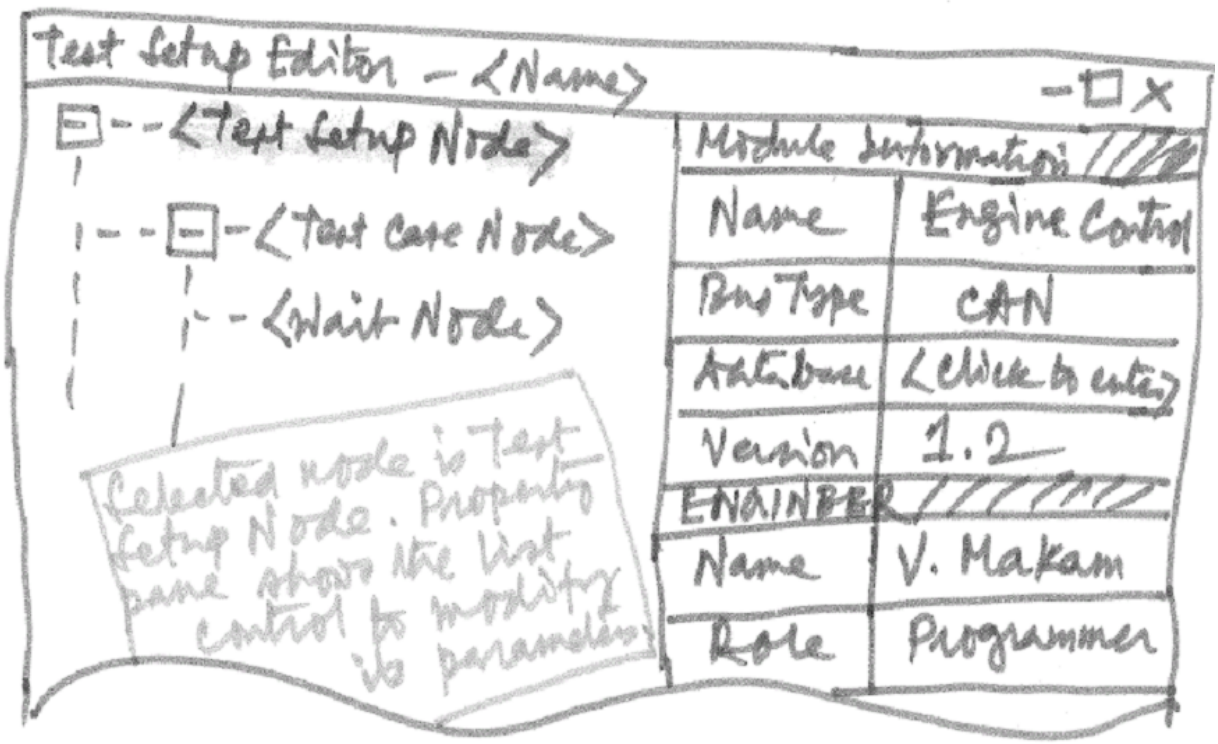


Clearly, the Test Setup Node is the root of all from where are sprouted the other nodes. The helper functions may be accessed by right-clicking on a node and selecting the proper menu item in the resultant popup menu. Below goes list of all the popup menus to be resulted from the right-click on the relevant tree nodes in the master layout pane –

SI. No.	Node Type	Popup menu item	Functionality	Display state condition
1	Test Setup Node	Add	Adds a test case	Always
2	Test Case Node	Wait	Adds a wait node	- do -
		Send	Adds a send node	Selecting a valid database is the prerequisite to enable this menu item
		Verify	Adds a verify node	- do -
		Replay	Adds a replay node	Always
		Delete	Deletes the test case node selected.	- do -

SI. No.	Node Type	Popup menu item	Functionality	Display state condition
3	Send Node	Add	Adds a message	- do -
		Delete	Deletes the send node selected.	- do -
4	Verify	Add	Adds a message	- do -
		Delete	Deletes the verify node selected.	- do -

Lastly, an overview of the property pane is presented below ~



In order to modify values of each node in the master layout pane, the node in question needs to be selected and a corresponding list control shall appear on the property pane which can be used to update values. Each of the modifiable nodes and its associated schematic property user interface is described below:

- Test Setup Node

Module Information	Name	< Name assigned to the test setup module >
Module Information	Bus Type	< Bus type needs to be entered which may be one of CAN, J1939 and FlexRay. Presently only CAN shall be selectable. >
Module Information	Database	< Path of the associated database >
Module Information	Version	< Version number string >
Engineer	Name	< Engineer's name >
Engineer	Role	< Engineer's role / designation >
Report File	Path	< Report file path >

Report File	File type	< File type. Can be one of TXT and HTM >
Report File	Time mode	< Time mode. May be one of system and relative modes. >

- Test Case Node

Test Case params	Parameter values
Identifier	< Identifier of the test case. Shall be unique >
Title	< Title of the test case >
Exception handler	< Action to be taken in case of failure. Must be one of "continue" or "exit". In case of the former, the process of test case running should go on whereas for the later the process should abort >

- Wait Node

Wait node params	Parameter values
Purpose	< Purpose of the wait >
Delay	< Time specified in millisecond. Takes an integer >

- Replay Node

Replay file params	Parameter values
File Path	< Absolute / relative path of the replay file >

- Send Node

Send node params	Parameter values
Messages	< Enter number of messages to be sent. This will create required number of rows to make message entries >
< Message ID >	< Message Name. Double-click on any of the two cells to view the popup combo / list box that shall display the available message list from database and select the required message entry >

- Send_Message Node

Send_Message params	Parameter values
Signal value type	< Shall be of the data type eTYPE_UNIT_SIGNAL. Choose one of RAW and ENGG. A drop-down list appears with the two entries >
< Default signal value >	< The default signal value which shall be assumed for the signal entry value unspecified by the user >
< Signal list showing signal names >	< Each of the list control entries correspond to a signal defined in the message. Signal values need to be entered here. For obvious reason, number of rows in the list control will be equal to the number of signals defined in the message >

- Verify Node

Verify node params	Parameter values
Failure classification	< Shall be of the data type eRESULT. Choose one of WARNING, ERROR and FATAL. A drop-down list appears with the three entries >

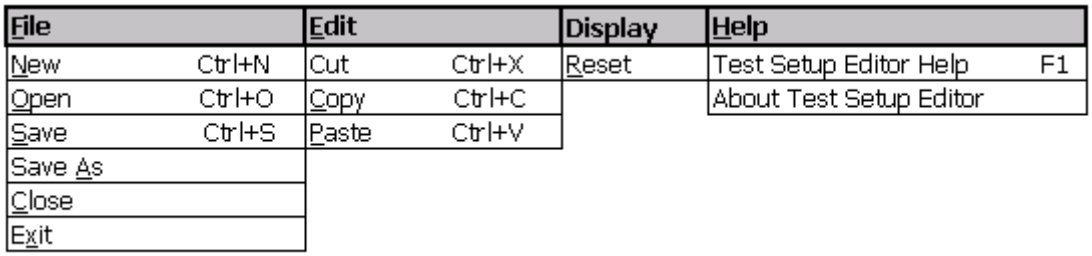
Verify node params	Parameter values
Messages	< Enter number of messages to be sent. This will create required number of rows to make message entries >
< Message ID >	< Message Name. Double-click on any of the two cells to view the popup combo / list box that shall display the available message list from database and select the required message entry >

- Verify_Message Node

Verify_Message params	Parameter values
Signal value type	< Shall be of the data type eTYPE_UNIT_SIGNAL. Choose one of RAW and ENGG. A drop-down list appears with the two entries >
Signals	< Enter number of signals whose values are to be evaluated. This will create required number of rows to make signal entries >
< Signal list showing signal names >	< The signal validation formula with normal algebraic notation like 'x' to denote the current signal value. The following mathematical operators can be used: +, -, /, *, (,), ==, !=, <, >, <=, >=, && and ||. To illustrate, an example is cited below ~ ((x + 3 == 9) || ((x / 23 == 3) && (2 * x - 34 > 12))) >

Menu items

Below is the schematic view of the menu items:



A brief explanation of each of the menu item's functionalities along with its display state conditions:

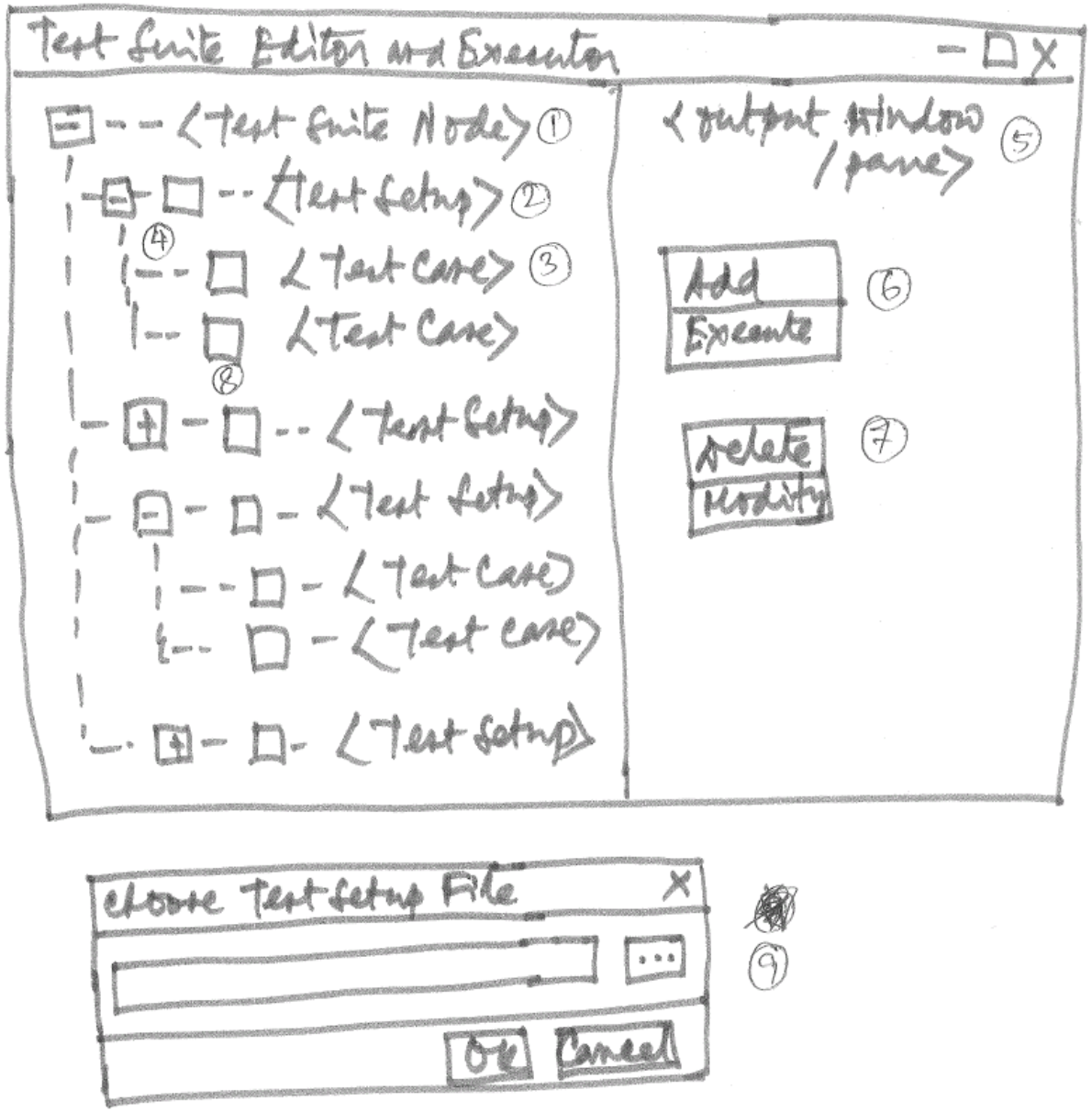
SI. No.	Menu item	Functionality	Display state condition
1	File->New	This shall create a new test setup.	Always be enabled.
2	File->Open	This shall open an existing test setup file.	- Do -
3	File->Save	This shall update the currently open file with the modifications made.	Shall be enabled when a test setup file is modified.
4	File->Save As	This shall update the some other new or existing file with the modifications made.	Shall be enabled when a test setup file is opened.

SI. No.	Menu item	Functionality	Display state condition
5	File->Close	This shall close the currently open test setup file.	- Do -
6	File->Exit	This shall close the application. In case there is a test setup file open and underwent some modification, the user shall be prompted.	Always be enabled.
7	Edit->Cut	Can be used to shift the selected node to somewhere else.	When one of the associated nodes in the master layout pane is selected.
8	Edit->Copy	Copies the selected node.	- Do -
9	Edit->Paste	Pastes the cut or copied node under the selected node. Cut, Copy and Paste operation are applicable to the master layout pane for the following nodes: 1. Wait, 2. Send_Message, 3. Verify_Message and 4. Replay	Below goes a list of the nodes and the parent nodes for which the menu item shall be enabled. Wait: Test case node Send_Message: Send node Replay: Test case node Verify_Message: Verify node
10	Display->Reset	Resets layout of the three panes to a default setting.	Always be enabled.
11	Help->Test Setup Editor Help	Shall open the help file.	- Do -
12	Help->About Test Setup Editor	Shall display programme information, version and copyright.	- Do -

Test Suite Editor and Executor

A graphical overview with short explanations on its various sections is provided below. The different numbered sections are explained here ~

- This is the test suite node and is the root of the tree control. The text is editable so that the user can assign a meaningful name to the test suite being defined.
- This is the Test Setup node. This node must be associated with a test setup file when such a node is being added.
- This is the Test Case node and shall appear when a Test Setup node is successfully associated with a corresponding file.
- This checkbox shall be used to select / deselect a Test Setup node while executing the Test Suite.
- The right pane is termed as the output pane where results of each test case executed shall be displayed.
- Right-clicking the Test Suite node results in the popup menu. The ' Add' button shall add a Test Setup node whereas 'Execute' button shall execute the current Test Suite.
- Right-clicking a Test Setup node results in this popup menu. The 'Delete' menu item shall delete the Test Setup menu whereas 'Modify' menu item shall change the Test Setup file associated with the node.
- This check box shall include or exclude the related Test Case node from the execution process.
- This dialog box shall be used when a Test Setup node is being associated with a corresponding file.



External Interface Requirements

There are two different kinds of CAN controllers being used:

- PCAN USB from Peak GmbH
- ES 581.x from ETAS GmbH

Additionally, a general hardware abstraction named as BOA (Basic Open Architecture) from ETAS GmbH is supported. This makes it possible to support numerous other controllers which BOA presently supports.

In all cases the device driver shall be procured from the vendor.

Interface Details

Simulation Engine

Tag	Name & Prototype	Functionality	Parameter Details	Return value	Remarks
RSI_5_001	RegisterClient ([in] ETYPE_BUS eBus, [out] USHORT* ClientID, [out] BSTR* PipeName, [out] BSTR* EventName)	This function makes an entry in the client list of the simulation engine assigning a unique client id. Also, it opens a communication channel between the engine and the client in the form of a pipe and an event for notification.	eBus – The target bus standard for which the communication channel is to be opened. The purpose of this parameter is only to distinguish the target buses. This component has no knowledge of the bus standard data frame format and structure. ClientID – The client id assigned which needs to be remembered and used for the subsequent calls. PipeName – Name of the the communication conduit. The caller needs to get a handle of the pipe in order to retrieve a message. EventName – When SimENG pumps a message into the pipe, this shall be signalled.	S_OK is successful, else S_FALSE.	This must be the first call to SimENG (with the exception of GetTimeModeMapping)
RSI_5_002	UnregisterClient ([in] USHORT ClientID)	Call this function to get unregistered from SimENG.	ClientID – Identifier of the client.	DO	-
RSI_5_003	ConnectNode ([in] USHORT ClientID)	Connects the caller to the virtual bus. On successful calling of this function,	DO	DO	-

Tag	Name & Prototype	Functionality	Parameter Details	Return value	Remarks
		receiving and sending of messages shall be possible.			
RSI_5_004	DisconnectNode ([in] USHORT ClientID)	Disconnects the caller from the virtual bus.	DO	DO	-
RSI_5_005	GetTimeModeMapping ([out] SYSTEMTIME* CurrSysTime, [out] ULONGLONG* TimeStamp)	Call this function to get a system time and the time stamp associated to it.	CurrSysTime – A system time TimeStamp – The time stamp associated with CurrSysTime.	DO	This is useful to calculate current system time from the time stamp.
RSI_5_006	SendMessage ([in] USHORT ClientID, [in] BYTE* pbyMsgData [in] UINT unLength)	Call this function to send a message to the virtual bus or other nodes.	ClientID – Identifier of the client. pbyMsgData – Message data entry unLength – Message data entry length in number of bytes.	DO	-
RSI_5_007	GetCurrentStatus ([in] USHORT ClientID, [in,out] VARIANT *pNodeStatus)	Call this function to get the current state of the node.	ClientID – Identifier of the client. pNodeStatus – Parameter to receive the current status of the node. This contains the following values: RESET (0), INITIALISED (1), WAITING (2), NORMAL_ACTIVE (3), NORMAL_PASSIVE (4), UNDEFINED (5)	DO	-

Signal Graph Window

Tag	Name & Prototype	Functionality	Parameter Details	Return value	Remarks
RSI_22_001	HRESULT CreateGraphWindow([in] CMDIFrameWnd* ParentWnd, [in] short BusType)	This function is used to create Signal Graph Window for a particular bus.	ParentWnd – Parent window pointer. BusType – Associated bus with this current	S_OK is successful, else S_FALSE.	

Tag	Name & Prototype	Functionality	Parameter Details	Return value	Remarks
			Signal Graph window.		
RSI_22_002	HRESULT ShowGraphWindow([in] short BusType, [in] BOOL bShow = TRUE)	Shows or hides Signal Graph Window of a particular bus type.	BusType - The bus type. bShow – TRUE to show, FALSE to hide.	S_OK is successful, else S_FALSE.	
RSI_22_003	BOOL IsWindowVisible([in] short BusType)	This function is used to get the visibility status of the window of a bus.	BusType - The bus type.	TRUE if visible, else FALSE.	
RSI_22_004	HRESULT SetRefreshRate([in] UINT RefreshRate)	This function sets the refresh rate of Signal Graph Window.	RefreshRate – Refresh rate in milliseconds.	S_OK is successful, else S_FALSE.	
RSI_22_005	UINT GetRefreshRate(void)	Getter function for refresh rate of Signal Graph window.	-	The refresh rate in millisecond	
RSI_22_006	CBaseMsgBufVSE* GetGraphBuffer(void)	This function returns interface of the common buffer / channel for the messages or frames of any bus supported.	-	The common buffer interface pointer if successful, else NULL.	A message entry is of the format: <BUS><Length of data bytes>
RSI_22_007	HRESULT SetSignalListDetails([in] short BusType, [in] CGraphList * pSignalList)	This function is used to pass the selected set of distilled signal information which are necessary to display signal graphs for a particular bus.	BusType – The bus type. pSignalList - The configured signal data for a particular bus.	S_OK is successful, else S_FALSE.	

IDIL_CAN

Tag	Name & Prototype	Functionality	Parameter Details	Return value	Remarks
RSI_14_001	DWORD DILC_GetDILList(BOOL bAvailable, DILLIST* List)	Based on the parameter this function renders number of the driver interface layers supported or available.	bAvailable – Set as true to get list of DIL implemented; else list of DIL supported by the existing licence will be returned. List – Buffer where the result is written. If it	Number of DIL.	-

Tag	Name & Prototype	Functionality	Parameter Details	Return value	Remarks
			is NULL, only number of DIL is returned.		
RSI_14_002	HRESULT DILC_SelectDriver(DWORD dwDriverID, HWND hWndParent, Base_WrapperErrorLogger* pILog)	Selects a driver (abstraction layer (DAL).	dwDriverID - The driver abstraction layer id. Possible values are - 1. DRIVER_CAN_STUB, 2. DRIVER_CAN_PEAK_USB, 3. DRIVER_CAN_PEAK_PP, 4. DRIVER_CAN_ICS_NEOVI and 5. DRIVER_CAN_ETAS_BOA. hWndParent - Window handle if the client application is a GUI application. pILog - Error logging interface.	If support for the intended one isn't allowed with the present license, this returns NO_LICENSE. Else returns S_OK.	Annotation of the ids: 1. Simulation engine without the hardware, 2. PEAK-CAN USB hardware, 3. PEAK-CAN Parallel Port Interface, 4. IntrepidCS neoVI hardware and 5. ETAS BOA Framework
RSI_14_003	HRESULT DILC_RegisterClient(BOOL bRegister, DWORD& ClientID, TCHAR* pacClientName)	Registers / unregisters a client. This is necessary to simulate a node and to receive messages. Only registered client's buffer will be updated on receive of a msg in the bus.	bRegister - TRUE to register, else FALSE. ClientID - Unique client id rendered by the function. pacClientName - Client node name.	1. ERR_CLIENT_EXISTS, 2. ERR_NO_CLIENT_EXIST & 3. ERR_NO_MORE_CLIENT_ALLOWED	Explanation: 1. Client already registered, 2. No such client with this id exists. 3. No more clients is allowed to register.
RSI_14_004	HRESULT DILC_ManageMsgBuf(BYTE bAction, DWORD ClientID, CBaseCANBufFSE* pBufObj)	Manages the message buffer list. Call this function to open a data channel to receive messages.	bAction - When MSGBUF_ADD, adds pBufObj to the target message buffer list. Removes when MSGBUF_CLEAR. ClientID - Client ID pBufObj - Interface to message buffer object.	S_OK if successful, else S_FALSE.	At present maximum number of entries in the list is kept as 8.
RSI_14_005	DWORD DILC_GetSelectedInterface(void)	This returns ID of the (void) interface layer presently selected.	-	ID of DIL	-

Tag	Name & Prototype	Functionality	Parameter Details	Return value	Remarks
RSI_14_006	HRESULT DILC_PerformInitOperations(void)	Performs initialisation operations.	-	S_OK if successful, else S_FALSE.	-
RSI_14_007	HRESULT DILC_PerformClosureOperations(void)	Performs uninitialisation operations.	-	S_OK if successful, else S_FALSE.	-
RSI_14_008	HRESULT DILC_GetTimeModeMapping(SYSTEM_TIME& CurrSysTime, UINT64& TimeStamp)	Call this function to get mapping between system time and the time stamp associated with it.	CurrSysTime – System time TimeStamp – The time stamp associated with CurrSysTime.	S_OK if successful, else S_FALSE.	Call this function after starting the hardware.
RSI_14_009	HRESULT DILC_ListHwInterfaces(INTERFACE_HW_LIST& sSelHwInterface, INT& nCount)	This function lists the hardware interfaces available and receive user's choice.	sSelHwInterface – HW_LIST& receive user's choice. nCount - Number of hardware interfaces available.	S_OK if chosen an interface, else HW_INTERFACE_NO_SEL	-
RSI_14_010	HRESULT DILC_SelectHwInterface(const INTERFACE_HW_LIST& sSelHwInterface)	Call this function to select a particular hardware interface.	sCurrHwInterface – Information on the hardware interface to be selected.	S_OK, NO_HW_ INTERFACE, ERR _LOAD_HW_INTERFACE	-
RSI_14_011	HRESULT DILC_DeselectHwInterface(void)	Call this function to release the selected hardware interface.	-	S_OK if successful, else S_FALSE.	-
RSI_14_012	HRESULT DILC_DisplayConfigDlg(PCHAR& InitData, int& Length)	Displays configuration dialog box for the selected DIL. If the dialog box needs to be displayed with initialisation, pass the relevant data as InitData. If it is null, the dialog box is uninitialised. This also contains the user's choice as OUT parameter.	InitData – Dialog initialisation data as [in] parameter. If this is null, the dialog appears uninitialized. This also contains the user's choice as [out] parameter. Length – Data byte length of InitData.	S_OK, CONTROLLER_NO_CONFIG, ERROR_OTHER	Afterwards, the caller must release the memory. Return value of ERROR_OTHER indicates controller specific error. Error details (string) may be retrieved by calling GetLastErrorString(…)
RSI_14_013	HRESULT DILC_SetConfigData(PCHAR pInitData, int Length)	To set the configuration data for the currently selected DIL. Caller must	pInitData – Configuration data stream. Length – Data stream length.		-

Tag	Name & Prototype	Functionality	Parameter Details	Return value	Remarks
		release the memory.			
RSI_14_014	HRESULT DILC_StartHardware(void)	Starts the hardware controller	-		-
RSI_14_015	HRESULT DILC_StopHardware(void)	Stops the hardware controller	-		-
RSI_14_016	HRESULT DILC_ResetHardware(void)	Resets the hardware controller	-		-
RSI_14_017	HRESULT DILC_SendMsg(DWORD dwClientID, const STCAN_MSG& sCANTxMsg)	To transmit a message	sCANTxMsg - The CAN message to be transmitted.	S_OK if successful, else S_FALSE.	-
RSI_14_018	HRESULT DILC_GetVersionInfo(VERSIONINFO& sVerInfo)	Getter for DIL version information.	sVerInfo - Structure to receive the information.	do	Version info on DIL, controller and driver.
RSI_14_019	void GetLastErrorString([out] char acErrorStr[], [in] int nLength)	Call to get descriptive string of the last error occurred.	acErrorStr – Receiver of the error string. nlength – Capacity of the string.	-	-
RSI_14_020	HRESULT DILC_FilterFrames(FILTER_TYPE FilterType, ECHANNEL Channel, UINT* punMsgIDs, UINT nLength)	Call to set PASS/STOP filter	FilterType - PASS_FILTER, STOP_FILTER. Channel - Channel number punMsgIDs - Array of Msg IDs to be filtered. nLength - Array size	S_OK if successful, ERR_INVALID_CHANNEL for invalid channel, ERR_DRIVER_NO_SUPPORT incase driver doesn't support HW filtering, else S_FALSE for failure.	If this function is called, DIL removes existing filters and new filters willl be added
RSI_14_021	HRESULT DILC_GetCntrlStatus(const HANDLE& hEvent, UINT& unCntrlStatus)	To get controller status. Caller has to pass the handle of an event which will be set whenever the controller changes the state.	hEvent - The client thread should wait for this event to get controller status notification. unCntrlStatus -	S_OK if successful, else S_FALSE.	Values of unCntrlStatus are: 1. CONTROLLER_ACTIVE, 2. CONTROLLER_PASSIVE, 3. CONTROLLER_BUSOFF
RSI_14_022	HRESULT DILC_GetControllerParams(LONG& lParam, UINT nChannel, ECONTR_PARAM eContrParam)	Call to get controller parameters. Value will be returned stored in lParam.	lParam - Out param. Result will be stored in this variable. nChannel - Channel number. eContrParam - Possible inputs are	S_OK if successful, ERR_INVALID_CHANNEL for invalid channel, S_FALSE for failure.	Values of lParam: 1. If eContrParam == HW_MODE a. defMODE_ACTIVE b. defMODE_PASSIVE c. defMODE_SIMULATE

Tag	Name & Prototype	Functionality	Parameter Details	Return value	Remarks
			1. NUMBER_HW, 2. NUMBER_CONNECTED_HW, 3. DRIVER_STATUS, 4. HW_MODE, 5. CON_TEST.		
RSI_14_023	HRESULT DILC_GetErrorCount(SERROR_CNT& sErrorCnt, UINT nChannel, ECONTR_PARAM eContrParam)	Call to Get Error Counts(TEC & REC) Error Counts.	sErrorCnt - Out param. Structure to store the Tx/ Rx error counter. nChannel - Channel number. eContrParam - PEAK_ERR_CNT, ERR_CNT.	S_OK if successful, ERR_INVALID_CHANNEL for invalid channel, S_FALSE for failure.	-

IDIL_J1939

Tag	Name & Prototype	Functionality	Parameter Details	Return value	Remarks
RSI_26_001	HRESULT DILJ1939_Initialise (Base_WrapperErrorLogger* pILog, CBaseDIL_CAN* pouIDIL_CAN);	Initializes J1939 network	pILog – pointer to wrapper_error object. pouIDIL_CAN - Interface to DIL CAN.	S_OK for success, S_FALSE for failure.	-
RSI_26_002	HRESULT DILJ1939_Uninitialise (void);	Uninitializes J1939 network	-	S_OK for success, S_FALSE for failure.	-
RSI_26_003	HRESULT DILJ1939_RegisterClient(BOOL bRegister, TCHAR* pacNodeName, UINT64 un64ECUName, DWORD& dwClientId);	Registers / Unregisters a client. This is necessary to simulate a node and to receive messages. Only registered client's buffer will be updated on receive of a msg in the bus.	bRegister - TRUE to register, else FALSE. pacNodeName - Client node name. un64ECUName - 64 bit ECU name. dwClientId - [Out Parm] Client's Id rendered.	1. ERR_CLIENT_EXISTS, 2. ERR_NO_CLIENT_EXISTS, 3. ERR_NO_MORE_CLIENT_ALLOWED & 4. S_OK	Explanation: 1. Client already registered, 2. No such client with this id exists. 3. No more clients allowed to register. 4. Success.
RSI_26_004	HRESULT DILJ1939_ManageMsgBuf(BYTE bAction, DWORD ClientID, CMsgBufVSE* pBufObj);	Manages the message buffer list. Call this function to open a data channel to receive messages.	bAction - When MSGBUF_ADD, adds pBufObj to the target message buffer list. Removes when MSGBUF_CLEAR. ClientID - Client ID pBufObj	S_OK if successful, else S_FALSE.	At present maximum number of entries in the list is kept as 8.

Tag	Name & Prototype	Functionality	Parameter Details	Return value	Remarks
			- Interface to message buffer object.		
RSI_26_005	HRESULT DILJ_SendJ1939Msg (DWORD dwClient, UINT unChannel, EJ1939_MSG_TYPE eMsgType, UINT32 unPGN, BYTE* pbyData, UINT unDLC, BYTE byPriority = DEFAULT_PRIORITY, BYTE byDestAdress = ADDRESS_ALL);	Sends a J1939 message.	dwClient - Client Id, unChannel - Channel number, eMsgType - COMMAND, BROADCAST, REQUEST, RESPONSE, unPGN - Parameter group number, pbyData - Data bytes, unDLC - Data length in number of bytes. byPriority - Priority (0-7) byDesrAdress = Destination address.	S_OK if successful, else S_FALSE.	-
RSI_26_006	HRESULT DILJ_SendAckMsg (DWORD dwClient, UINT unChannel, ETYPE_ACK eAckType, UINT32 unPGN, BYTE byAddresAck);	Sends a acknowledgement message.	dwClientId - Already register node's client Id, unChannel - Channel number, eAckType - Acknowledge type (ACK_POS, ACK_NEG) unPGN - PGN to be sent. pbyData - PGN data. byAddresAck - Destination address.	S_OK if successful, else S_FALSE.	-
RSI_26_007	HRESULT DILJ_RequestPGN (DWORD dwClient, UINT unChannel, UINT32 unPGN, BYTE byPriority = DEFAULT PRIORITY, BYTE byDestAddress = ADDRESS_ALL);	Requests a PGN from a node	dwClient - Client Id, unChannel - Channel number, unPGN - Parameter group number to be requested, byPriority - Priority (0-7) byDesrAdress = Destination address.	S_OK if successful, else S_FALSE.	-
RSI_26_008	HRESULT DILJ1939_GoOnline ( );	Starts J1939 network. All nodes start sending	-	S_OK if successful, else S_FALSE.	-

Tag	Name & Prototype	Functionality	Parameter Details	Return value	Remarks
		according to the configuration.			
RSI_26_009	HRESULT DILJ1939_GoOffline ( );	Stops J1939 network. All nodes stop sending msgs.	-	S_OK if successful, else S_FALSE.	-
RSI_26_010	HRESULT DILJ_NM_GetNodeName (BYTE byAddress, TCHAR* acNodeName);	Gets the node name from 8 bit address from J1939 network.	byAddress - 8 bit node address (0 - 253) acNodeName - [OUT_PARAM] Nodes name.	S_OK if successful, else S_FALSE.	-
RSI_26_011	HRESULT DILJ_NM_GetNodeAddress (BYTE& byAddress, DWORD dwClient);	Gets the node address from Client Id from J1939 network.	byAddress- [OUT_PARAM] Nodes 8 bit address, dwClient - Client Id.	S_OK if successful, else S_FALSE.	-
RSI_26_012	BOOL DILJ_NM_bIsAddressClaimed (BYTE byAddress);	Returns whether the address is already claimed by another node.	byAddress - 8 bit node address (0 - 253).	TRUE if claimed, else FALSE.	-
RSI_26_013	HRESULT DILJ_NM_ClaimAddress (DWORD dwClientId, UINT unChannel, BYTE byAddress, BYTE byPriority = DEFAULT_PRIORITY);	Node tries to claim a new address by sending Address Claim message into the network.	dwClientId - Already register node's client Id, unChannel - Channel number, byAddress - New address to be claimed, byPriority - Priority (0 - 7).	S_OK if successful, else S_FALSE.	-
RSI_26_014	HRESULT DILJ_NM_RequestAddress (DWORD dwClient, UINT unChannel, BYTE byPriority = DEFAULT_PRIORITY, BYTE byDestAddress = ADDRESS_ALL);	A node requests address from an another node.	dwClientId - Already register node's client Id, unChannel - Channel number, byDestAddress - Destination Address, byPriority - Priority (0 - 7).	S_OK if successful, else S_FALSE.	-
RSI_26_015	HRESULT DILJ_NM_CommandAddress (DWORD dwClient, UINT unChannel, UINT64 unECU_NAME, BYTE byNewAddress, BYTE	A node commands another node to assume an address.	dwClientId - Already register node's client Id, unChannel - Channel number, unECU_NAME - 64 bit ECU NAME of the destination node. byDestAddress	S_OK if successful, else S_FALSE.	-

Tag	Name & Prototype	Functionality	Parameter Details	Return value	Remarks
	byPriority = DEFAULT_PRIORITY, BYTE byDestAddress = ADDRESS_ALL);		- Destination Address, byPriority - Priority (0 - 7).		
RSI_26_016	HRESULT DILJ_ConfigureTimeOut (ETYPE_TIMEOUT eTimeOutType, UINT unMiliSeconds);	Configure timeouts for flow control packets.	eTimeOutType - Time out type (TO_BROADCAST, TO_RESPONSE, TO_HOLDING, TO_T1, TO_T2, TO_T3, TO_T4). unMiliSeconds - Timeout value in mili seconds.	S_OK if successful, else S_FALSE.	-
RSI_26_017	HRESULT DILJ_GetTimeModeMapping (SYSTEMTIME& CurrSysTime, UINT64& unAbsTime);	Gets the time mode mapping.	CurrSysTime - Reference system time. unAbsTime - Absolute time.	S_OK if successful, else S_FALSE.	-
RSI_26_018	BOOL DILJ_bIsOnline (void);	Get the network status	-	TRUE if Online, else FALSE.	-

Project Execution Requirements

Development Environment

Workstation specific

- Machine: IBM compatible PC/XT and the
- OS: Microsoft Windows XP/7

Implementation

- Environment: Microsoft Visual C++ Express 2008
- Language: C, C++, VC++, XML
- Other libraries: Win32 SDK, MFC
- Installation: NullSoft

Others

- Design tool: Enterprise Architect
- Design notation: UML
- Version control: GitHub

Testing

- TestCocoon
- AutoIt

Error handling requirements

Error handling in code:

1. A library (or lower level module / component) does not render any user notification. This is the sole responsibility of the UI. Rather this indicates the error by the return value of the API function.
2. The return value of an interface function is of type HRESULT. The following Microsoft-specified convention for error codes is followed:

Bits	**31-30**	**29**	**28**	**27-16**	**15-0**
Contents	Severity	Microsoft/ customer	Reserved	Facility code	Exception code
Meaning	0 = Success 1 = Informational 2 = Warning 3 = Error	0 = Microsoft-defined code 1 = Customer-defined code	Must be 0	Microsoft-defined	Microsoft/ customer-defined

   Following this specification, the 29th bit is always kept as 1. The 6th and 7th nibble are used to denote the module.

Execution time error handling:

One aspect of this is reporting of the error messages by logging. For a suitable (i.e., which is grave enough and needs suitable debugging information) erroneous condition, the message containing the file name and line number of the code shall be logged in a text file. The error message text, combined with the other two information provides a good amount of clues to identify the root cause of the erroneous condition. The log file shall be

generated only when such an erroneous situation does occur. The log file shall be generated in the working directory of the application under a generated directory named as “log”.

The logging module shall be an out-of-proc server named as “ErrorLogger.exe”. This, being of this architecture, entitles complete logging even if unexpected crash in the application happens.

The most beneficial aspect of such a logging is that in case of a customer reported problem, the debugging information can also accompany the report. This means lesser time is required for analysis and reproducing of the test case at developer’s workplace.

Resource Requirements

Pentium 4, 256 MB RAM and 20 GB HDD. The software development and target implementation platform would be on Windows XP/7 platform.

Testing Requirements

Testing methodology

- Unit testing
- System testing
- Acceptance testing

Use of tools

- TestCocoon will be used to do ensure maximum code coverage (this doesn't include the codes generated by the IDE).
- In addition, the programme shall be scrutinized in terms of efficiency, memory leak. A report reflecting those run-time system parameters shall be prepared to strategize maximum optimization.

Inputs for testing

- Test data shall be provided by ECM2.

Environment for testing

- IBM compatible PC having minimum configuration of Pentium IV processor operating at 600MHz
- RAM – 256 Megabytes

Performance Requirements

This aspect aims at achieving full resilience and integrity for the suite. In other words - to ensure optimized performance at any data rate. This can be illustrated through the following prime statements:

- Application can not ensure no-loss of data. Data overrun is inevitable under a number of easily realizable circumstances. Making the system slow enough by executing simultaneously sufficient number of processes, using inefficient driver and hardware, using all the features of the application at a time while the data rate is very high – are a few examples.
- The suite shall ensure an optimized way of processing the data streaming from the driver in addition to minimize the possibility of losing frame data [RS_11_01]. Optimization means employment of minimum resources (memory, processor time slice and user's time) to carry out a task. In doing so, the often encountered problem of trade-off between memory and speed shall also be taken into account [RS_11_02].
- The application shall ensure that detailed information (number of frames lost) of any data overrun event is available to the user at any moment [RS_11_03].
- It shall be possible for the user to dynamically enable / disable the available features / functionalities [RS_11_04]. This will provide the user a better control over the application functionalities and thereby efficient usage of this suite. In case the user is more interested in the message data for offline analysis when the bus is at peak load, only logging can be enabled with other functionalities like updating of message window, signal graph window been disabled. At bus peak load this strategy will minimize the possibility of data loss.
- Disabling a feature means that currently no resource be employed for this feature [RS_11_05]. For example – if display feature is deactivated, not even background tasks like frame data updating (or formatting) should take place.
- It has been found out by empirical means that usage of a temporary data storage buffer reduces the possibility of data loss to the maximum extent. Subject to the expected maximum throughput, its storage size shall be made configurable by the user [RS_11_06]. By default the tool shall make an optimal choice based on certain other factors or valid assumptions [RS_11_07].

Software Release Criteria

The software shall be deemed fit to be released when it clears the successful criteria of the following:

- Development environment: All the different phases of testing namely, unit testing, system testing and integration testing.
- Real usage environment: Application runs properly without buffer overrun or showing any nonresponsive state even at the maximum bus load in the real usage scenario. This shall be carried out at a real end-user’s workplace.

Annexures

<Any related documents can be attached here. (If the 1.4 Definitions, Acronyms and Abbreviations is too long, then it can be added here as Annexure - A).>