Specification EE-Browser v6 (Redesign for Daimler)

# Introduction

## Background

Daimler uses the latest version of Intedis “EE-Browser” (delivered package is 5.5.0) for their internal harness/topology release processes. This version is based on the initial “EE-Browser” architecture initially developed in China for reviewing “HARcad”-exported viewfiles (original HCV format). In addition to this the support for “FUNCoSAR” has been enhanced in a later version (FCV format). Daimler uses “LDorado” from COMSA as authoring system for harness design. The generated output from this software system rests upon using the KBL file format (Step AP212 schema definition version V2.3SR-1) for the technical data and a combination of one or more SVG files (Scalable Vector Graphics) representing the graphical information of the exported harness/topology.

Daimler and Intedis agreed to define a modified HCV file format containing both file types. Therefore the export interface in “LDorado” has been changed accordingly. The result ended up in a zipped HCV container exported from “LDorado” system containing one KBL file, at least one or more SVG files and some additional files like the index.xml.

In 2011/2012 the Intedis software developing team has spent a lot of time improving “EE-Browser” and implementing the necessary interfaces and business logic to support this HCV container type and their embedded files. For that a new converter library has been developed converting SVG graphics into our own VectorDraw-based CAD environment. Further the KBL import interface has been reworked to offer a quicker and more consistent way getting the data extracted out of this huge schema into our own XML schema. The Excel-based intermediate format based on HARcad has been used for this purpose. In addition to the normal conversion process the mapping between graphical figures and technical data objects played an important role. This were solved duo to creation of own custom objects for each SVG group during conversion process into VectorDraw and the storage of KBL-Id and some other identification properties based on comments existent on SVG group objects. By means of the previous described implementation the custom objects got their own IDs (as GUID) which were used to enhance the XML schema with CAD handle information based on the referring VDCL files.

Some more enhancements has been added to the business logic of “EE-Browser” like displaying active and hiding inactive graphical objects depending on their module configuration, creating an intelligent search handling and many more minor changes. Finally we achieved a proper working solution with full support of Daimler-generated HCV files. On the other side the old and sometimes inefficient developed base architecture and the native usage scenario for “EE-Browser” highlighted some major disadvantages in the Daimler-oriented use case with the new HCV file type:

* Most of the topology drawings are very large-scaled (more than 30-40 meter width and up to 1-2 meter high). This leads to serious memory problems during loading the converted VectorDraw document. The huge number of basic shapes inside of SVG group objects are the main problem for that as well as the sometimes ineffective drawing definition in the original SVG file. Beside that aspect the current solution of “EE-Browser” needs a lot of additional memory preparing the data visualization and the mapping between drawing and XML data which is also responsible for occurring memory overflows.
* The data binding and preparation in “EE-Browser” is not well thought out. Displayed grids using own static data sets containing copied data from the initially loaded XML document. This structures are not well formatted and cannot be used for easy cross-references to other related data or linking the graphical figures.
* For the Daimler use case with the special HCV container support it makes no sense to convert the KBL data into our own XML structure. Our own XML schema were developed supporting HARcad drawings. Inside some restrictions are existent which makes it sometimes unhandy to work with all information stored inside of the KBL file.
* The provided functionality in the current version of “EE-Browser” supporting different Intedis-own formats plays no role for Daimler. They need a proper and efficient working solution especially for their needs. Only the “LDorado”-exported HCV format is relevant.
* In some areas the GUI of “EE-Browser” is not very user friendly developed. The visible representation of menu, toolbars and other controls are not state of the art and got long in the tooth.
* The different Daimler-specific changes in “EE-Browser” regarding the business logic leads in a number of code fragments which are not well programmed. This implementation affects the complete stability of the architecture and is unpleasant at all.

The mentioned points above and some minor others were the reason for beginning thinking about a complete redesign of the existing “EE-Browser” software. From the first day it was very clear that a complete rework makes only sense if we separate the existing solution from that what Daimler definitely needs. The following specification describes a system which should be developed solely for Daimler purposes and the fully support of HCV files originated from “LDorado”.

## Basic development environment assumptions

For the complete new development of “EE-Browser” the usage of the latest software development principles and 3rd party components is mandatory. In detail that means

* using VB.NET 2010 (MS Visual Studio) as programming language with usage of .NET Framework 4 technologies
* configure compile settings for the solution to fully support “Windows 7”-based x64 operating system platforms (processor architecture)
* using one single Visual Studio solution with a single project containing all software core functionalities and perhaps other projects for companion features

Due to naming and versioning reasons from Daimler viewpoint the complete development of the successor of the old “EE-Browser” architecture will lead in remaining the title “EE-Browser” and increasing the version to “6.0.0” to prevent any misunderstandings. In their eyes the redesign is the logical next step in the evolution of the historical “EE-Browser” development process.

Regarding the internal software development processes a new repository for the redesign has to be added in SVN database on SDEVSRV1. In addition to this the linking for Gemini issues must be activated for the new repository root entry. In Gemini itself a new version information “6.0.0” should be added and the different work packages for the complete redesign has to break down into single Gemini issues for getting the most out of Gemini by means of observing/controlling the hole development process.

## Necessary software and 3rd party components

The overall design and development will be implemented within MS “Visual Studio 2010” by use of the programming language VB.NET. The next important aspect is the usage of the latest .NET Framework (Version 4) APIs for accessing functionalities like LINQ and TPL applying the most efficient coding style.

The complete GUI design has to be developed by using the latest version of Infragistics “NetAdvantage for Win Client 2012 Vol. 2” 3rd party component. Especially the ribbon-based menu layout, dock-able panes and data grids should be used from this user control suite.

For all graphical functionalities to handle the valid visualization and modification issues including conversion of the original SVG drawing, the latest VectorDraw “Developer Framework” (version 6024) has to be used.

Some of the export functionalities have to create Excel files. To ease the functionalities of collecting and treating data for Excel worksheets the latest “Aspose.Cells” component should be used for supporting latest “MS Office” versions.

## Basic implementation hints

Referring to the already created document “EEB-Redesign: Assumptions/Ideas/SWAD” the basic core functionalities and implementation routines should be specified again. The overall target for the redesign should be an easy-to-use, very efficient and state-of-the-art developed software system for Daimler to support all their required functionalities for reviewing harness and topology drawing exported out of their authoring system “LDorado” by using of the predefined HCV container format.

A simple structured, lean and well-designed core architecture should be the key to success for this development process. Based on this settlement a stand-alone Windows Forms application has to be developed. One single MDI-based form should represent the heart of the software including all necessary other user controls for the different working aspects. Every feature and design implementation should have the major goal to work properly in the Daimler environment and maximize the satisfaction of the our customer.

Modern, user-friendly GUI concepts like ribbon menu etc. has to be used and an orthogonal and stringent usage concept must be implemented. The including of user guidance functionalities in relation to have a tooltip and log mechanism properly working is mandatory for the redesign.

One of the most important aspects is the memory usage issue. As already mentioned above Daimler wants to review complete harness/topology drawing with 130% content and all available module configurations. It is not possible to display only predefined/user-selected parts of the complete drawing. Under this circumstances the conversion process for SVG drawings into our own VDCL file format has to be implemented very carefully by concentrating to the aspects of minimizing memory usage without losing the advantage of quickly load the converted drawing into “EE-Browser”. The SVG convert algorithm must be solving the complete group structure in relation to identify links to technical objects in addition to find a way storing the basic graphical information in the VDCL format. Custom objects like used in the current existing solution might be the right implementation approach for this issue. But some improvements are necessary to support very huge SVG graphics trouble-free.

A draft mode for the drawing could be generated from KBL data and displayed to the user before the original drawing will be loaded completely. The draft mode only displays basic graphical information like vertices and segments. There are some use cases where the customer only wants to search for some data and need not the original drawing. In every case there must be a handling which allows the user to change between original drawing and draft mode. A document tree should offer the possibility to open/close original drawing while the draft mode can be displayed in the background.

The other important aspect while developing “EE-Browser v6” is the representation of the technical data available in the KBL schema. Distinct from the current solution the redesign should support fully access to the KBL document by de-serializing the schema definition into a data object model. It makes no sense to fall back to our own XML schema with prolonged and complicated conversion processes because there will be no advantages for further use cases in Daimler world. Anyway, this allows a very quick possibility to allocate the complete KBL data and improve the load speed and memory usage. Important here is efficient data binding concept to fill-in data into the displayed grids.

The application should provide multi-document functionalities by means to work with two or more HCV files simultaneously in one session. Each of the opened HCV files contains one or more graphical documents itself which must be displayed separately. A well-structured main frame with tab controls has to handle this issue. In addition to this different open/close commands has to be implemented. The claim of quick loading HCV files makes it necessary to use parallel load algorithm (usage of more than one processor core). The “Task Parallel Library” provided by .NET Framework 4 is the key technology for the implementation of this functionality. The complete HCV load process must be subdivided into following single tasks:

1. Load and display of complete “document.kbl” file and display of draft drawing based on graphical information extracted from the KBL container (include filling grids)
2. Load and prepare data of “Index.xml” file
3. Load and convert SVG files

Every single step of the complete load process can be run into one single parallel task, following above mentioned order. In case that more than one SVG file is existent the parallel principles of loading can be used for converting and displaying SVG drawings simultaneously as well. To prevent thread exceptions occurring if user controls will be accessed by different parallel running tasks the complete load handling must be developed very carefully.

Not to forget should the support of different import and export functionalities and the compare of two different HCV containers. The access to the additional information stored into the “Index.xml” can be achieved like already implemented in the current version of “EE-Browser”.

# Main frame concept

## Basic GUI architecture

The redesign of “EE-Browser” should offer a very modern and user-friendly GUI. Therefore the software should follow the latest technologies regarding user interaction and accessibility for provided data. An easy to use Windows-Client stand-alone software has to be developed. All used GUI components were used from the Infragistics “NetAdvantage” component suite (Version 2012.2).

The start object is the “MainForm”. This object is a MDI container providing an ribbon-based menu following the most modern layout well-known by Microsoft Office tools in top area of the form. The status bar in the bottom area of the “MainForm” displays coordinate information originated from the cursor position of the active displayed drawing, different status and load messages and a progress bar indicating the percentage value and visible progress until a task will be finished. Between this both controls an tabbed manager control cares about the organizing of the embedded child forms and controls. This is necessary because the user should be able to load multiple HCV files into one “EE-Browser” session simultaneously. Each loaded HCV file will be represented by one tab provided by the manager component.

If the user opens a new session of “EE-Browser” or no HCV file is currently loaded the ribbon displays only the basic application (file) menu and the “Home” tab. Following menu buttons are available in the application menu:

* Open: Opens a not loaded HCV file and creates a new tab on the manager component in the “MainForm”.
* Save: Saves the active HCV file and persists possible changes which have been done by user regarding redlining functionalities.
* Save As: Persists redlining changes and saves a copy of the active HCV file with given name. The user has to select a destination path.
* Export:
  + Data: Offers the possibility to export all data loaded into grids to Excel
  + Drawing: Exports the active displayed drawing to a CAD-conform file format like DWG or DXF
* Close: Closes the active HCV document and the relevant tab of the tab manager control. If user did some changes on drawing a message box must pop up and ask if changes should be persist or discard.
* Close All: Closes all loaded HCV files and the belonging tabs. For each close action the same functionality has to be implemented like on single close action.
* Print: Provides features and a proper configuration dialog for plotting the active drawing.
* Exit: Closes the complete application. Before leaving the application the functionality provided by “Close All” menu button has to be called.

In addition to the provided menu buttons a list of recently opened HCV files has to be displayed next to the menu band. The necessary information will be hold into the XML-based browser configuration file.

Besides the application menu the “Home” tab offers a button “Properties” providing a simple dialog where the user can change some basic settings regarding the work with “EE-Browser”. A second button “Visibility settings” is available allowing the configuration of the grid appearance for each table. The dialog which appears after click this button displays a tree structure of all grid tables with regarding column information originated from the different XML-based appearance configuration files where the visibility settings are stored.

The “MainStateMachine” class should handle all user interactions and different application states of the “MainForm” and the regarding menu handlings. That means this class has to care about the initialization and the valid presentation of the application menu and the “Home” tab and of course the executing of the different events located behind each menu button if the user pressed it.

## Document handlings

If the user opens an not already loaded and valid HCV file the tabbed MDI manager control on the “MainForm” has to create a new tab object with the name of the loaded HCV file. Inside the created tab control a new form will be embedded: the “DocumentForm”. This form class object provides additional control: On top of the form a second toolbars manager providing the ribbon menu tab groups and buttons comes along with the loaded HCV file. After initializing of the ribbon menu controls this will be merged to the existing ribbon menu found on the “MainForm”. This handling makes it very convenient to handle the different view states of the ribbon menu depending weather one or more HCV files are loaded or not. One additional advantage is that every loaded HCV file which will be represented by one “DocumentForm” has their own ribbon menu state. This is mandatory for a valid multi-document handling. The dock manager control is responsible for displaying different control panes around the “DocumentForm”. So called control panes are small modal panels with own window dialog handling which can be docked on a user-defined location inside the referring parent form object or can be undocked and moved outside of the application main frame. The last user control of the “DocumentForm” is the tab control. As mentioned above one HCV file can contain one or more SVG files. Each SVG drawing has to be displayed in a separate tab on this form.

Inside of each created tab page based on the tab control a “DrawingCanvas” user control containing a VectorDraw base control element has to be added. Both controls are docked filled and displays the converted SVG drawing as CAD document with all provided user interactions like selection and highlight.

There are four different control panes available:

* The “InformationHub” user control will be included into the first pane located at the bottom of the form. Inside this user control an additional tab control is available providing a docked grid control for each tab. The different tabs separate the possible object types bases on the KBL data. Each grid provides the regarding data extracted from the KBL document for the object defined by XML configuration files.
* The “NavigationHub” is a small extended snapshot of the complete drawing which is currently displayed in the “DrawingCanvas” control and should give the user the possibility to orientate and navigate very easily through the complete drawing. The current view box of the “DrawingCanvas” has to be displayed as colored rectangle inside of the navigation control. Pan and view box changes triggered by click on a specific location inside of the navigation control have to be implemented.
* The “ModuleHub” user control represents a tree view showing the complete module configuration bases on the loaded KBL data. The root nodes represents the module families (if available) and the child nodes the regarding modules itself. The tree has to be sorted in alphabetic order and provides checkboxes for each node to enable/disable the modules in the drawing.
* The “DocumentHub” user control shows a small tree which allows the user to display or hide all the existing SVG drawings available inside the loaded HCV container file. Checkboxes for all tree nodes allows easily switching on/off the visible state for the different drawings.

The “DocumentStateMachine” class is responsible for the complete menu handling and all user interactions for the “DocumentForm” which occurs when the state has to be changed in case of pressing any button similar to the “MainStateMachine” functionalities for the “MainForm”. Especially the different drawing states and possible interactions on the canvas are an important aspect for this class.