Development of PC-Tools for Powertrain Control System Development

Storage of Parameter Identifiers

M O H A M M A D K H A L E D I

KTH Computer Science
and Communication

Master of Science Thesis
Stockholm, Sweden 2008

Development of PC-Tools for Powertrain Control System Development

Storage of Parameter Identifiers

M O H A M M A D K H A L E D I

Master's Thesis in Computer Science (30 ECTS credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2008
Supervisor at CSC was Serafim Dahl
Examiner was Stefan Arnborg

TRITA-CSC-E 2008:110
ISRN-KTH/CSC/E--08/110--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.csc.kth.se

Abstract

Every modern vehicle, which has an On-Board Diagnostic (OBD) system, uses Electrical Control Units (ECU) in order to perform diagnosis and control of one or more of the electrical subsystems in the vehicle. A diagnostic system works like a simple question and answer game: the tester asks for a specific information and the control unit responds to the question by transmitting the information to the test device. An ECU contains a number of parameters used when it responds to a request for a diagnosis service.
Scania is developing its own diagnostic system used in a number of ECUs. The Powertrain control system, that is a part of the diagnosis system, consists of an EMS (Engine Management System) and a GMS (Gearbox Management System). NEVE, a division of NE (Department for Powertrain Control System development), develops, maintains and supports PC-tools used by NE. The most important PC-Tools are XCOM (Diagnostic-tool), SCOMM (Communication-tool) and CompTransNet (Calibrating-tool). When requesting a diagnosis service, XCOM sends a PID (parameter identifier) to EMS via SCOMM. A parameter identifier is used to identify a specific parameter by its unique number, which may consist of a single byte or multiple bytes. SCOMM, in order to communicate with EMS, requiers information about which parameter identifiers EMS deals with and requires some information for transforming the response to engineering units. Today, the PIDs and their information are stored in source-code files or other types of documents.
One of the documents is an XML-file which is generated from source-code files which are used by EMS. Generating a new version of the XML-file, that takes place in a few steps, is required for every new version of the source-code files. Storing the information in different documents makes the maintenance of SCOMM difficult and time consuming. In order to cope with the problem, one measure is to store the PIDs and their information in a database and to generate the documents used by SCOMM from the database directly. This master thesis work is an attempt to reach this goal.

Referat

Utveckling av PC-verktyg för utvecklingen av motorstyrenhet

(Lagring av parameterdefinitioner)

Varje modernt fordon som har ett On-board diagnostiskt system (OBD) använder sig av elektriska styrenheter (ECU) för utförandet av diagnos och kontroll av en eller flera styrenheter i fordonet. Ett diagnossystem fungerar som ett enkelt spel med frågor och svar: testern frågar efter en specifik information och styrenheten skickar svaret till testern. En styrenhet har ett antal parametrar som används när den svarar på frågan om en diagnosservice.
Scania utvecklar sitt eget diagnossystem som innehåller ett antal styrenheter. Powertrain control system som är en del av diagnossystemet består av en (EMS) motor- och en (GMS) växelstyrenhet. NEVE, som är en avdelning av NE som utvecklar Powertrain control system, utvecklar, underhåller och supportrar PC-verktyg som används av NE. De viktiga PC-verktygen är diagnosverktyget XCOM, kommunikationsverktyget SCOMM och kalibreringsverktyget CompTransNet. Vid begäran till en diagnosservice, skickar XCOM en sk PID (parameter identifier) till EMS via SCOMM. En PID används för identifieringen av en specifik parameter med sitt unika nummer, vilket kan bestå av en eller flera byte. För att kommunicera med ECUer, kräver SCOMM få veta vilka parametrar EMS handlar om samt kräver information för omvandlingen av svaret till tekniska enheter. Idag, sparas denna information i olika källkodsfiler eller andra typer av dokument.
En av dokumenten är en XML-file vilken genereras från källkodsfiler som används av EMS. Genereringen av en ny version av XML-filen behöver utföras för varje ny version av källkodsfilerna. Att spara informationen i olika dokument leder till att underhållning av SCOMM blir svår och tidsödande. Ett sätt att lösa problemet är att lagra parametrarna och deras information i en databas och därifrån generera dokumenten som används av SCOMM direkt. Detta examensarbete är ett försök att uppnå detta mål.

Contents

"To my mother Soghra, who taught me the value of science and love."

Acknowledgments

Though my name appears on the cover of this thesis work, I would never have been able to accomplish it without the help of some persons. First I would like to thank people at The Royal Institute of Technology for all those wonderful studying years I had with them. People who never withheld supporting me, especially my teacher and supervisor Serafim Dahl of who's support this work and my career are the result. I would like to appreciate the valuable contribution from people at NEVE, at Scania AB. Foremost, I would like to thank Jonas Fransson that answered all my questions patiently.

Chapter 1

Introduction

This work is a master thesis work in Computer Science at The Royal Institute of Technology (KTH). The work has been performed at Scania CV AB, Södertälje, during the spring of 2008. The Royal Institute of Technology has been responsible for the course.

1.1 Background

Scania is an international company and one of the well-known manufacturers of Industrial and Marine Engines, heavy trucks and buses. Since it was founded in 1891 it has built and delivered more than 1,000,000 trucks and buses for heavy transport work. Scania has its main office at Södertälje in Sweden, where also manufacturing and development take place. It has also other factories at Oskarshamn and Luleå in Sweden, Zwolle and Meppel in Netherlands, Tucumán in Argentina, São Paulo in Brazil, Stupsk in Poland, Angers in France and St. Petersburg in Russia. Scania has also an established market in Europe, Latin America, Asia, Africa and Australia. Today more than 28,000 employees work for Scania [1].

Todays' modern vehicles, regardless a car or a truck, have an On-board Diagnostic System (OBD). OBDs are used in order to give the owner or a repair technician access to the status information for different subsystems of the vehicle. During the 70s vehicle manufacturers started using electrical components in order to control engine functions and diagnose engine problems. The first OBDs were only able to illuminate a malfunction indicator light if a problem was detected and didn't give any information about the nature of the problem. Through the last decade, on-board diagnostic systems have become more sophisticated. Todays' OBDs are not only able to inform the owner about an arisen problem but are also able to give some information about the nature and the source of the problem. OBD-II, a new standard which was introduced in the 90s, is the most used on-board diagnostic system nowadays [2].

1.2 Diagnostic System

A diagnostic system contains a number of Electrical Control Units (ECU) connected to each other via a Controller Area Network (CAN). If different bus systems are used to connect the control units, gateways are also used, see figure 1.1. The diagnostics system works like a simple question and answer game: the tester asks for a specific information and the control unit responds to the question by transmitting the information to the test device. The ECUs have a number of parameters and log data about the subsystems of the vehicle, which will be used in order to diagnose the subsystem. Different ECUs log different information in different ways. A tester connected to the OBD requests data about a subsystem of the vehicle by sending a Parameter Identifier (PID) to the CAN. The PID is just a number, consisting of one or more bytes, that identifies which parameter was requested by the tester. Communication between the ECUs in the OBD and a tester connected to the system is based on Open System Interconnection (OSI). The request from the tester will be transmitted over the CAN and the ECU, which contains the requested PID, recognizes the PID and sends the response to the tester via CAN. Today's modern vehicles can contain an OBD with up to 50 ECUs.

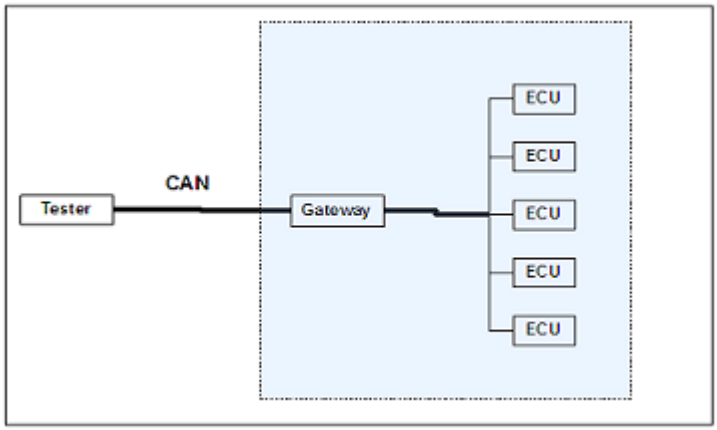


Figure 1.1. *Vehicle On-Board Diagnostic System connected to a tester via a gateway.*

1.2.1 Electrical Control Unit (ECU)

As mentioned above, an OBD uses electrical control units in order to perform the diagnosis. An ECU is an embedded system that controls one or more of the electrical subsystems in a vehicle. Modern ECUs use a microprocessor which can process the inputs from sensors in a subsystem in real time. An ECU contains both hardware and software. The main part of the hardware is a CPU (Central process Unit) and an I/O (Input/Output). The software can be stored in the CPU or in other chips

of the hardware such as Flash memories. In this way the CPU can be updated by uploading a new software or replacing an old chip. These are a generation of ECUs which are called Programmable ECUs.

Making significant changes in a unit on a vehicle requires a new configuration. For example changing the injection system in an engine requires a new tempratur-sensor that in turn requires a new configuration. Since the old ECUs do not provide an appropriate control for this new unit, it will be required either to change the ECU or to re-program it. Changing the ECU requires designing a new hardware and a new configuration. By using programmable ECUs only a new configuration is required. The programmable ECUs can be re-programmed according to the new requirements by connecting them to a PC via a USB port.

Development of diagnostic functionality is one of the most crucial steps. Specification, implementation of software and testing of the ECU take place in parallel to the development of the ECU. Every change in a subsystem of a vehicle demands new specification and requirement, which in turn demands a new implementation of the ECU-specific software. The functionality test and integration test of the ECU, with the new software, are one of the most important steps. The tests must be done in parallel to the development of the ECU in order to prevent invoking the vehicles with some eventually discovered problem in the functionality of the ECU.

1.3 OBD at Scania

Scania has developed its own OBD, see figure 1.2. The system contains a number of ECUs and communication between the ECUs takes place by three different bus system and SCOMM (Scania Communication Module) is an implementation of this system. SCOMM is an internally implemented software which has been designed according to SSF 14230-3 Keyword Protocol 2000 - Part 3 (Application Layer) [3]. In order to perform a diagnostic function, one or more ECUs are connected to the off-board tester XCOM. XCOM, that actually is an graphical interface for SCOMM, is even another internally implemented program, which is used for reading the logged data in an ECU and reading from and writing to variables during the development of ECUs. When receiving a request from XCOM, SCOMM needs parameter identifiers for reading from or writing into variables. SCOMM receives information about the parameter identifier of the ECU from some XML-files.

1.4 The Purpose of this thesis work

NEVE, a division of NE, the department for Powertrain Control System development at Scania, takes care of the logged data in the Engine Management System (EMS) and develops, maintains and supports the PC-tools for developing the Powertrain Control System. EMS is used for performing control and diagnosis of engines. The information about the parameter identifiers in EMS, used by SCOMM for performing diagnosis, are stored either in source-code files or in other types of

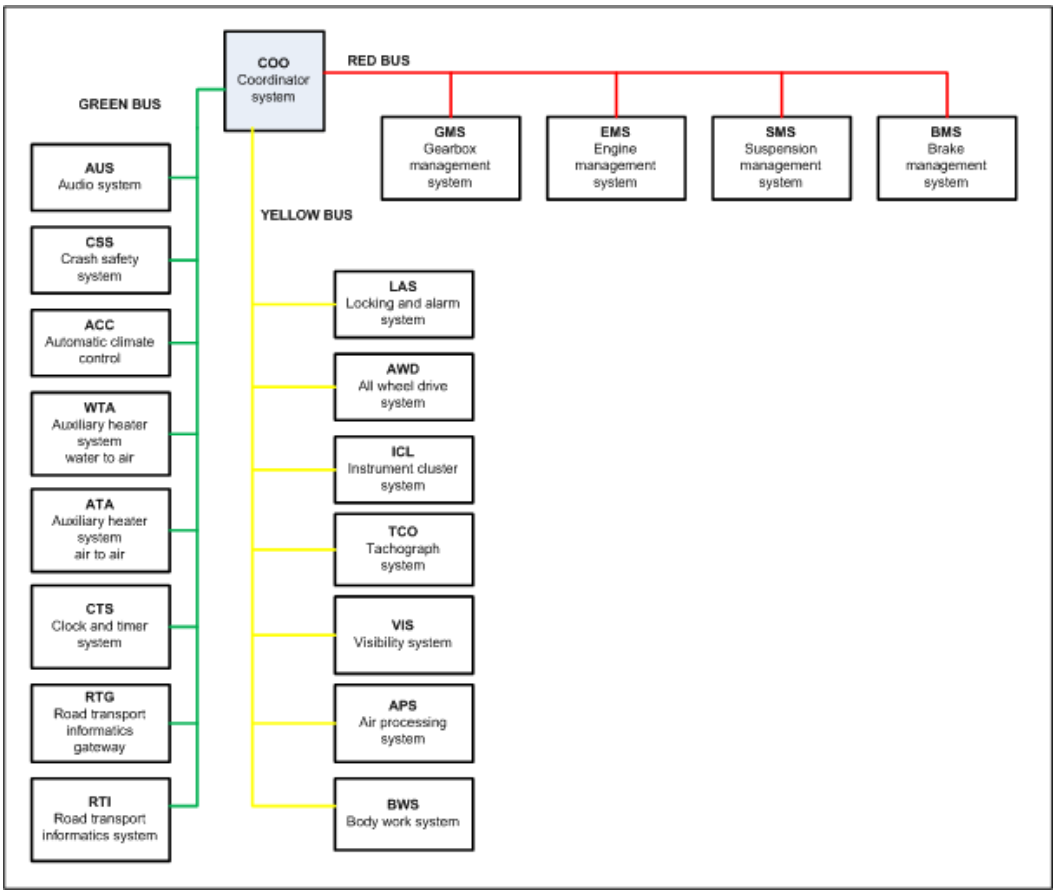


Figure 1.2. *General structure of the CAN network at Scania.*

documents. It causes that the maintenance of the SCOMM, which uses the information about the parameter identifiers for communication, becomes difficult and time-consuming. This thesis work is about a solution for coping with the problem.

Chapter 2

Problem Statement and Requirements

2.1 Preliminary

As mentioned in chapter 1, Scania has its own OBD with a number of ECUs. One of the most important ECUs is the Engine Management System (EMS). EMS is used for performing diagnosis and controlling different subsystems in an engine. In order to develop and test the EMS, three main PC-tools are required: Off-board Diagnosis-tool, Compiling-tool and Calibrating-tool.

- **Off-Board Diagnosis-tool:** XCOM, which is an internally designed and produced PC-tool, is used as an off-board diagnosis-tool during EMS development. XCOM, that actually is a graphical user interface, is used for reading the DTC (Diagnostic Trouble Code) in the EMS, reading from and writing to the parameters. It also uses both the Compiling and Calibrating tools. When the user requires one of the diagnosis services, XCOM sends the request to the EMS via SCOMM. Then XCOM represents the response from EMS, delivered by SCOMM, to the user.
- **Compiling-tool:** The compiling-tool creates an address-file which contains the address of the parameters used by EMS. The created file will be used by the calibrating-tool, CompTransNet, for creating a checksum from the software used in the EMS and also for calibrating of the EMS for different models of the same unit. The checksum will be used for finding out the modification of the engine in the future.
- **Calibrating-tool:** The Calibrating-tool, CompTransNet, is used for creating a checksum from the software used in the EMS and calibrating EMS. Since EMS uses the same software for different models of the engine, in order to adjust the software according to the specification of the new model, it is required to justify the constants, scalars, tables etc. CompTransNet performs calibrating using the created address-file by Compiling-tool. In this way the same ECU and the same software will be used for different models of the engine.

2.2 Goal of the project

Every time XCOM queries for a diagnosis service it sends the request to EMS via SCOMM, see figure 2.1. Since the values returned by EMS are not engineering units, SCOMM performs some transformations and delivers the transformed values to XCOM. SCOMM, in order to communicate with EMS, needs to know which parameter identifiers EMS deals with. In addition, SCOMM needs the parameter identifiers and their information in order to transform the returned values from EMS to the engineering units. The used parameter identifiers and their information can vary with a new version of the used software.

SCOMM has access to a XML-file which contains the parameter identifiers and their information used by EMS. When receiving a request for a diagnosis service, SCOMM first checks the existence of the parameter identifier in the XML-file. If the received parameter identifier is one of those that EMS deals with, it sends the request for the information about the parameter to EMS . After receiving a response from EMS, SCOMM performs transformation to the received values according to the information about the requested parameter in the XML-file. Now, if every thing is going well, SCOMM has the requested service transformed to engineering units and can send it back to the XCOM.

As mentioned the only way that SCOMM can find out which parameter identifiers EMS deals with, is to look at an XML-file. The XML-file is created according to the parameter identifiers and their information stored in the softwares used by EMS. The parameter identifiers and their information are stored in a source-code file which is used by EMS and some of the information is stored in other files. For example information about freeze frames, used for getting information about malfunctions, is stored in another file.

Any kind of change in the engine, which is a usual occurrence, demands modification of the source-code file used by EMS and modification of the XML-file according to the new source-code file. In this way there will be different versions of the source and XML-files. Storing the information about the parameter identifiers used in different versions of the software, in different files, makes the maintenance of SCOMM difficult and time consuming.

The goal is to identify parameter identifiers used by EMS, and then to create a database for storing the identified parameter identifiers and their information for different versions of the source-code file used by EMS. In this way the user will be able to generate the tables in different versions of the source-code file from the database directly. It will also enable generating of different versions of the XML-file, directly from the database.

2.3 Requirements

Because the goal of this project is to identify and to store the parameter identifiers and their information existing in different versions of the EMS-specified software, a

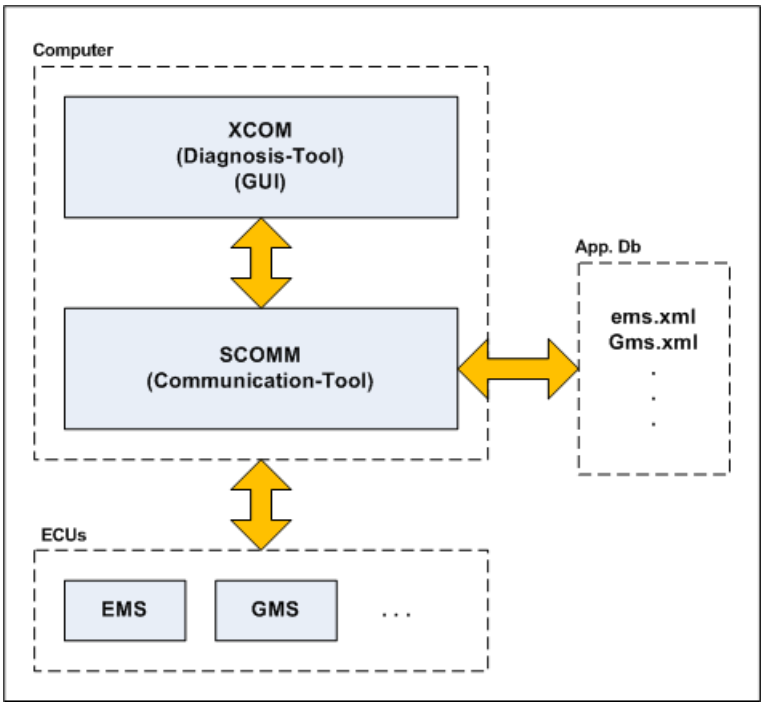


Figure 2.1. *Off-Board Diagnostic System. The figure shows the communication between XCOM, SCOMM and EMS. In order to deliver the request from XCOM to EMS and transform the response from EMS, SCOMM needs to know about the used parameter identifiers in EMS. SCOMM finds the information about the parameters in S7.xml.*

database and a user interface are required. Since some of the parameter identifiers are used by EMS and the others are used by other ECUs, it is justified that the database consists of two main parts. The first one, "Common Tables", for storing all of the parameter identifiers and their general information like their identifying hexadecimal value and description. This part can even store general information in order to perform a transformation of the responded data to the engineering units, i.e. scaling formulas and units. The other part "EMS-specific tables" will be used for storing the parameter identifiers and their information used by EMS in different versions of the EMS-specific software.

The mentioned goals of the project requires:

- Identifying parameters used by EMS and their information for storage.
- Designing of a solution for storage of the parameter identifiers and their information.
- Generating the tables in different versions of the source-code file and generating different versions of the XML-file, used by EMS, from the storage.
- Implementation of the solution.

- Testing and verification of the solution.

2.4 Specifications

In order to reach the mentioned goals, the solution should contain a database, an appropriate user interface, an application for connecting to the database and an application for generating the tables used in different versions of the source-code file and the XML-file, used by EMS.

2.4.1 Design of the database

A database will be used for storage of the parameter identifiers and their information. As mentioned above, the database consists of two main parts. The following specification should be considered in designing the database.

- **Common part:** The common part should contain appropriate tables in order to store, add, edit and delete the general information about parameter identifiers.
- **EMS-specified part:** The EMS-specified part should contain appropriate tables in order to store, add, edit and delete parameter identifiers and their information used in different versions of the EMS-specified source-code file.

2.4.2 Applications

Some applications are required in order to present and modify data, generate the tables and the XML-file. The following specification should be considered in designing the applications.

- **User interface:** A suitable user interface is required for presenting the data retrieved from the database. It is used in order to perform adding, editing, modifying and deleting the stored data in the database. Thus it should contain appropriate frames for performing mentioned actions.
- **File-generation:** The application should also contain appropriate utilities in order to generate the tables used in the EMS-specific source-code file and to generate the XML-file, used by EMS, for different versions.
- **Connection:** Performing mentioned actions, named in the first bullet, requires connecting to the database. The application must contain appropriate utilities in order to connect to the database and perform different actions.

2.5 Summary

Scania develops its own OBD system with EMS as one of the used ECUs. In order to develop EMS three PC-tools are required: Diagnosis-tool, Compiling-tool and

Calibrating-tool. The parameter identifiers used in EMS are stored in different files, which has made the maintenance of the system difficult and time consuming. In order to cope with the problem, the purpose is to design and implement a database for storing the parameter identifiers used in different versions of the EMS-specified software, and an application for modifying the stored data. This solution is used even for generating different versions of tables in the source-code file and the XML-file. The objective requires that the database has two main parts, one for general information of the parameter identifiers and the other for EMS-specific parameter identifiers. The application must have suitable utilities in order to edit and modify the stored data and for generating the mentioned tables and the XML-files.

Chapter 3

Design of the solution

3.1 Preliminary

As mentioned in chapter 2, the solution consists of a database and an appropriate application. The database is used for storage of the parameter identifiers and their information and the application for connecting to the database and performing desired modifications. In order to make maintenance of the solution as easy as possible and to perform suitable validation and verification when modifying the stored data, using N-Tier architecture is motivated. The objective of this chapter is to analyse and design such a solution.

3.2 Analysis of the solution

The solution is used for storage of parameter identifiers and their information used in different versions of EMS-specified software. Since the solution in the future may be used for storage of more than one ECU-specific parameter identifiers, two more important aspects in the design of the solution should be considered, which are discussed below.

3.2.1 Usability of the solution

If the solution is used by more than one ECU, it should prepare the appropriate support for it. It means that several users should be able to connect to the database and modify the contents of the tables corresponding to their ECU. Using a database with two parts, common and ECU-specific, will make communication between the different users and the database possible and every user will be able to modify its own ECU-specified part. The users will also be able to use the common part, which will contain general information for all parameter identifiers, in order to perform modifications and creating different versions of their ECU-specific softwares, see figure 3.1.

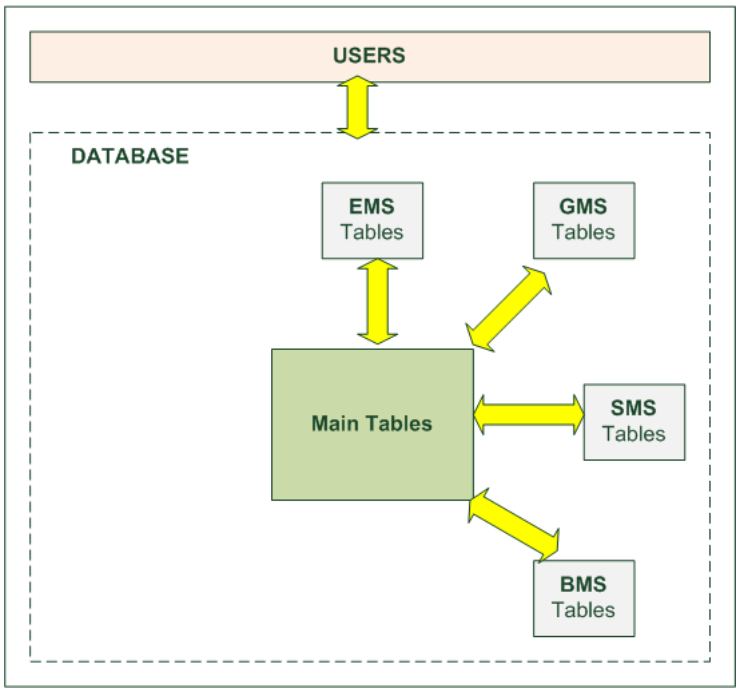


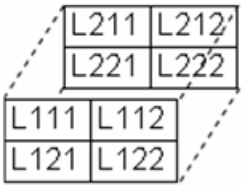
Figure 3.1. *Communication between users and the database with common and ECU-specific tables.*

3.2.2 ODX (Open Diagnostic Data Exchange)

The other important aspect is to consider the new standards that in the future can be used for developing OBDs, and their influence on maintenance and modification of the solution in order to maintain the applicability of the solution. One of the new standards is Open Diagnostic Data Exchange (ODX). ODX is an XML format and according to ASAM (Association for Standardization of Automation and Measuring System),

> *"It specifies the concept of utilizing a new industry standard diagnostic format to make diagnostic data stream information available to diagnostic tool application manufacturer to simplify the support of the aftermarket automotive service industry. The ODX modeled diagnostic data are compatible to the software requirements of the Modular Vehicle Communication Interface (MVCI) [5] [6]. The ODX modeled diagnostic data will enable a MVCI device to communicate with the vehicle (ECU(s)) and interpret the diagnostic data contained in the messages exchanged between the external test equipment and the ECU(s). For an ODX compliant external test equipment no software programming is necessary to convert diagnostic data into technician readable information to be displayed by the tester"* [7].

See an example of an ODX document in figure 3.2.




```
<SDGS>
    <SDG>
        <SDG-CAPTION ID="imatrix">
            <SHORT-NAME>matrix</SHORT-NAME>
            <LONG-NAME>A 3-dimensional matrix mapped to an SDG structure</LONG-NAME>
        </SDG-CAPTION>
        <SDG>
            <SDG-CAPTION ID="itable1">
                <SHORT-NAME>firstTable</SHORT-NAME>
            </SDG-CAPTION>
            <SDG>
                <SDG-CAPTION ID="i1row1">
                    <SHORT-NAME>firstRow</SHORT-NAME>
                </SDG-CAPTION>
                <SD SI="example">L111</SD>
                <SD SI="example">L112</SD>
            </SDG>
            <SDG>
                <SDG-CAPTION ID="i1row2">
                    <SHORT-NAME>secondRow</SHORT-NAME>
                </SDG-CAPTION>
                <SD SI="example">L121</SD>
                <SD SI="example">L122</SD>
            </SDG>
        </SDG>
        <SDG>
            <SDG-CAPTION ID="itable2">
                <SHORT-NAME>secondTable</SHORT-NAME>
            </SDG-CAPTION>
            <SDG>
                <SDG-CAPTION ID="i2row1">
                    <SHORT-NAME>firstRow</SHORT-NAME>
                </SDG-CAPTION>
                <SD SI="example">L211</SD>
                <SD SI="example">L212</SD>
            </SDG>
            <SDG>
                <SDG-CAPTION ID="i2row2">
                    <SHORT-NAME>secondRow</SHORT-NAME>
                </SDG-CAPTION>
                <SD SI="example">L221</SD>
                <SD SI="example">L222</SD>
            </SDG>
        </SDG>
    </SDG>
</SDGS>
```

Figure 3.2. *This matrix can be described by an SDG structure, in ODX format, of depth 3 [7].*

In parallel to this project Scania is considering use of ODX with CANdelaStudio. CANdelaStudio is a tool developed by Vector Software GmbH, which is developed specially for the compilation, editing, and display of diagnostic data descriptions for control units in the automobile area. It edits CANdela documents, of which each represents precisely one ECU. One of the most important documents used by

CANdela are documents in ODX format [8]. If CANdelaStudio and ODX format will be used by Scania some fundamental modifications of the solution is necessary. In order to do as little modifications as possible when using ODX, it is unavoidable to design an N-Tier architecture with more layers.

3.3 Identifying parameters

According to KWP2000 EMS ECU-S7 [4], an OBD has several services which can be used by a tester. The most important services are:

- ReadEcuIdentification service
- InputOutputControlByCommonIdentifier service
- ReadDataByCommonIdentifier srvice
- WriteDataByCommonIdentifier service
- ReadDataByCommonIdentifier service
- InputOutputControlByCommonID service

There are four main groups of parameter identifiers used by EMS and each group uses some of the mentioned services. It will be reasonable to have a table for each group of the parameters. These groups are:

- **Scania Common identifiers**, which are readable and use the first three services.
- **Input Output Common Identifier (IOCI)**, which when is used in the inputOutputControlByCommonIdentifier request service, it identifies an ECU local input signal, internal parameter or output signal according to the common id list.
- **EOL parameters**, used for EOL (End of Line) programming, which uses Write/ReadDataByCommonIdentifier services.
- **Read Signals**, which use the services readDataByCommonId or inputOutputControlByCommonId.

Since SCOMM uses some information for transforming the returned data by ECU to engineering units, it will also be necessary to have some tables for storing that information. Because there are two different groups of transforming data, the database should also have two tables for storing that information.

3.4 Database

Design of a suitable database is the first step. The database will contain the used parameter identifiers and their information for different versions of the EMS-specified software. Identifiers are hexadecimal values from 0x0000 to 0xFFFF, which can identify 65535 parameters. There is no use of all of the parameter identifiers in the same ECU and every ECU uses only a number of them [4]. This motivates that the database is divided into two different parts, a common and a ECU-specified part. The common part will contain general information about all the identifiers i.e. their hexadecimal values, function name used for reading from an ECU, description and information about types, units, scaling formulas etc. The ECU-specified part will contain information about the parameter identifiers for different versions of software for different ECUs. In this way, not only EMS but also the other ECUs can use the database in the future.

Since the function name, description and other values of the parameter identifiers can be changed, it is important to copy the information from the common part to the ECU-specific part for every parameter identifier used in the different versions. The data stored in the common part is used for adding data to the tables used for storing the information in a new version of the ECU-specified software. The data stored in the ECU-specified part is used for generating tables of an old or new version of the software or regenerating an old or new XML-file. In this way ECU-specified information can be retrieved even if data in the common part has been changed.

3.5 Architecture of the solution

Considering the described aspects above, it is necessary to use an N-Tier architecture. Using three different layers, Application, Middle, and Database layers, can give an appropriate level of maintenance, see figure 3.3. The architecture makes it possible to perform necessary modifications easily. The key point is that making a change in a layer provides minimal impact to other layers and hence makes the application a lot more maintainable. The designed architecture not only permits modification of the database, but it also gives the opportunity of a high level of control and validation. All parts of the design communicate with database only via the database layer. The architecture will also enable an improved control over the database connection which is resource consuming. Maintenance will also be performed more easily. If Scania uses the ODX format in the future, it will only be required to change the application in the interface layer and there will be no need to change the other application in the solution.

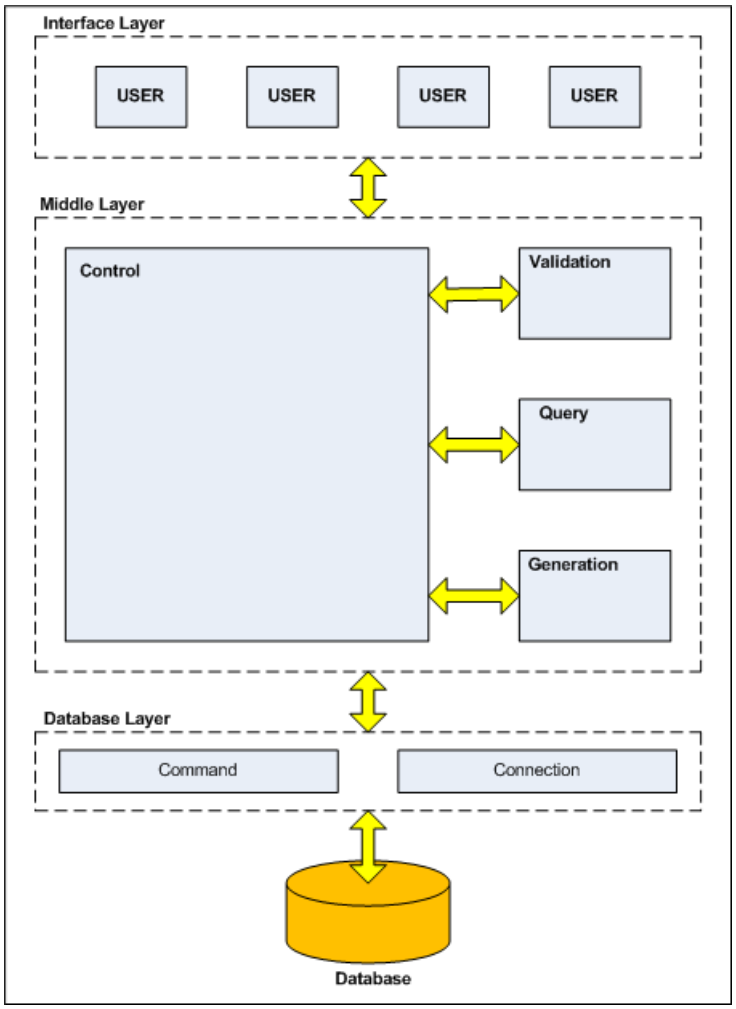


Figure 3.3. *Architecture of the solution with three layers; Application, Middle and Database.*

3.6 Summary

A database and an application are required in order to store parameter identifiers and their information. Parameters are identified by hexadecimal values from 0x0000 to 0xFFFF. There is no use of all parameters in EMS and those that are used are divided into four groups. Thus the database must consists of two parts, a common and an EMS-specified part. The common part will contain general information about all parameter identifiers. The EMS-specified part that will contain parameter identifiers used by EMS should have a table for each group. It also should contain two tables for transforming information. In order to have a better control and maintenance of the application and have as little modifications as possible if Scania uses ODX (Open Diagnostic Data Exchange) in the future, it is unavoidable to use

a N-Tier architecture when designing the application.

Chapter 4

.NET Framework

4.1 Preliminary

The development platform .NET has been developed by Microsoft. The goal has been to develop a language-neutral and platform-independent development platform. The .NET platform has been constructed on two main parts BCL (Base Class Library) and CLR (Common Language Runtime), see figure 4.1. The .NET is based on an open standard CLI (Common Language Infrastructure), standardized by ECMA (European Computer Manufacturers Association [11], and .NET is actually an implementation of this standard. The .NET in general and CLR in particular have been formed of components that have different names in the standard and in implementation of Microsoft.
The .NET gives support for using different languages in the same software solution that are designed by different developers. This and other properties of .NET has made the use of .NET with MS Visual Studio very popular in the industrial world. The objective of this chapter is to give a very brief view of .NET technology that is a very broad area. For more information about the .NET refer to [9].

4.2 .NET languages

As mentioned the .NET is a language-neutral platform. Five different languages are shipped with the .NET(3.5), C#, Visual Basic.NET, J#, C++/CLI and JScript.NET. There are also .NET compilers for other different languages such as Smalltalk, COBOL and Pascal. For a complete list of supported languages by .NET refer to [10]. People like to use different languages with different syntaxes. The .NET's multilanguage support makes it possible for developers to use their language of preference and its syntax. In addition developers can even share their compiled assemblies among departments and external organizations.

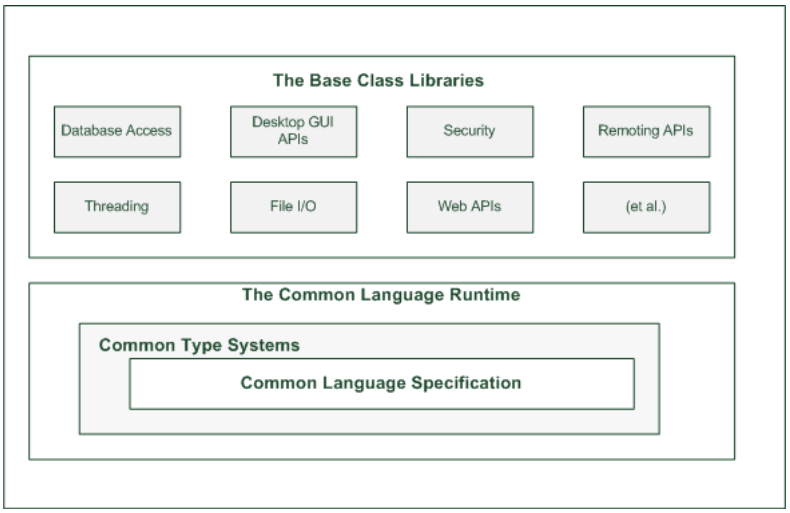


Figure 4.1. *The CLR, CTS, CLS and Base class library relationship [9].*

4.3 Main components of .NET

As it is visualized in figure 4.1 the .NET contains four main components BCL, CLR, CTS and CLS.

- BCL, Base Class library. The .NET platform provides a base class library that is available to all .NET program languages. The base class library contains various predefined classes for encapsulating threads, file Input/Output, graphics etc. It also contains classes that support different services that can be used by developers in different applications. The classes can be used in order to facilitate database access, manipulation of XML documents and construction of web-enabled front ends. Figure 4.1 visualizes the relationship between CLR and BCL in a high level of view.

- CLR, Common Language Runtime, for locating, loading and managing .NET types. CLR will be explained more in section 4.4.

- CTS, Common Type System. CTS describes all common data types and programming constructs supported by the runtime. It also specifies how these types interact with each other and details about how they are represented in the .NET metadata format.

- CLS, Common Language Specification is a related specification that specifies all types and programming constructs that all .NET programming languages recognize and agree on.

4.4 Common Language Runtime, CLR

CLR that is an executing environment has the primary role of locating, loading and managing the .NET types. The CLR also takes care of a number of other details such as memory management, creating application domains, threads, object context boundaries and performing various security controls. In the .NET platform CLR has been provided as a single well-defined runtime layer shared by all languages and platforms, that are .NET-aware, see figure 4.2. The language-neutrality and platform independency of CLR has been carried out by performing the compilation in several steps. The written code in some programming languages will be compiled to an intermediate language called CIL (Common Intermediate Language) or shortly IL (something analogue to Java Bytecode). Then the compiled code that is saved in a PE-file (Portable Executable Format), is compiled to a platform specified machine-code. PE-file has a language-neutral format and the second compilation will be performed by a JIT (Just In Time) compiler when the code is running by the machine. The runtime engine (CLR) is also in charge of resolving the location of an assembly and finding the requested information within the binary by reading metadata contained in it. The CLR then lays out the types in memory and compiles the associated Common Intermediate Language (CIL) into the platform specific instructions. It then performs any necessary security control, and then executes the code in question. CLR also interacts with the types contained within the .NET base class library when it is required [9].

4.4.1 Compilation of the code

When compiling a code, written by a .NET-aware language, a module will be created. A module is a cornerstone that is used in an assembly. An assembly is an executable program or a part of a larger system. A module contains IL-code and even file headers that point to (i.e.) where the executing method (main) starts. It also contains tables, called metadata, that define used or referenced types.

4.4.2 Metadata

A .NET assembly, in addition to CIL instruction, contains metadata. Metadata describes each and every type (i.e. class, structure, etc) defined in the binary. It also defines the members of each type, for example methods and properties. The metadata is always contained in modules and it emits by the compiler. In this way the assemblies are completely self-describing [9]. Metadata contains three kind of tables:

- **Definition tables**, tables with indexes for all types, methods, parameters and events. There is always a table that contains identification information in the module i.e. filename, version etc.
- **Reference tables**, tables with index over all assemblies, modules, types or everything referenced in the module.

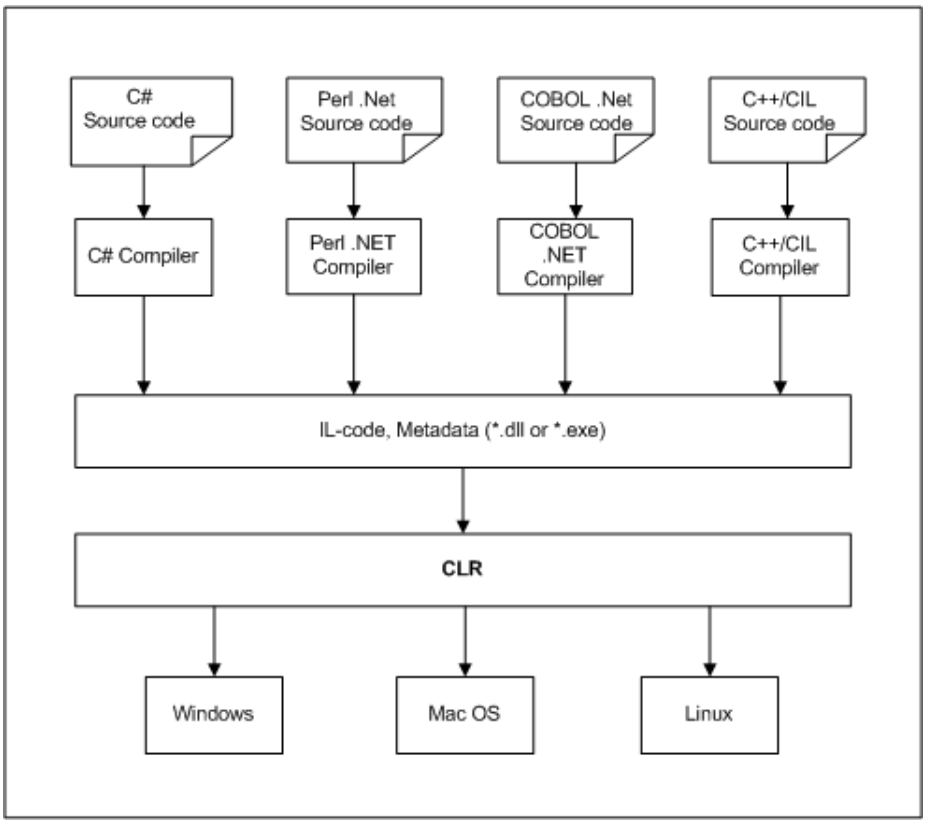


Figure 4.2. *A simplified diagram over the different compiling steps for CLR.*

- **Manifest tables**, tables that contain reference to all used resources and information about the actual assembly.

All modules contain the first two tables, but manifest tables are used only by assemblies.

4.4.3 .NET assemblies

As mentioned when compiling a code, written by a .NET-aware language, a module will be created. The module is used by an assembly. The assembly contains CIL instruction that looks like Java Bytecode and is not compiled to platform-specific instructions. It will be compiled when a part of the instructions is referenced. An assembly, that is the smallest executable unit, is either a single-File or Multi-file assembly. In a single-file assembly there is a one-to-one correspondence between a .NET assembly and a module. In other words the binary and the assembly are one, which means the *.exe can be referred to as the assembly itself. Multi-file assemblies are composed of several .NET binaries, modules. In such an assembly one of the

modules, “Primary Module” contains the assembly manifest. The primary module documents the set of required secondary modules within the assembly manifest.

4.5 Intermediate Language IL

IL is a platform independent object-oriented language created by CLR. It has no register and uses a stack. In other words it puts operands on the stack and fetches results from the stack. The code contains assembly instructions and directives that start with “.”. Functions start with the directive “.method” followed by return type. The directive “.entrypoint” defines the start point in assembly (analogue to main) and it must always define some function as entrypoint. Classes starts with “.class” followed by properties and the name of the class. Assemblies are declared in the same way as classes.

4.6 ADO.NET

In the world of the industry, most application can’t be built without interaction with a database. Because almost every software application interacts with one or more databases it requires a mechanism to connect to the databases. ADO.NET is used for this purpose and .NET applications that interact with databases depend on ADO.NET.

ADO.NET is an integral part of the .NET framework, which enables access to relational, XML and application data. ADO.NET supports N-tier programming and provides working with disconnected data. The architecture of the ADO.NET is also designed so that it takes care of opening and closing connections automatically, see figure 4.3.

As the architecture shows, the ADO.NET has two main components: Data provider and DataSets, see figure 4.3. DataSets are in-memory data stores that can hold several tables. They only hold data and don’t interact with data sources. It is DataAdapters that manage connections with the data source and give the developers the opportunity of disconnected access. A DataAdapter opens a connection only when it is required and then closes it as soon as the task is done and the connection is no longer needed. A DataAdapter performs the following steps when it fills a DataSet with data fetched from a data source: opening a connection, retrieving data into DataSet and Closing the connection. It also performs the following steps in order to update the data source with changes in the DataSets: opening a connection, writing changes from the DataSet into the data source and closing the connection.

4.7 Summary

.NET framework is a language-neutral and platform-independent technology. It reaches the language-neutrality and the platform independency by compiling the source code in a few steps. It first compiles the source code to IL modules, and then

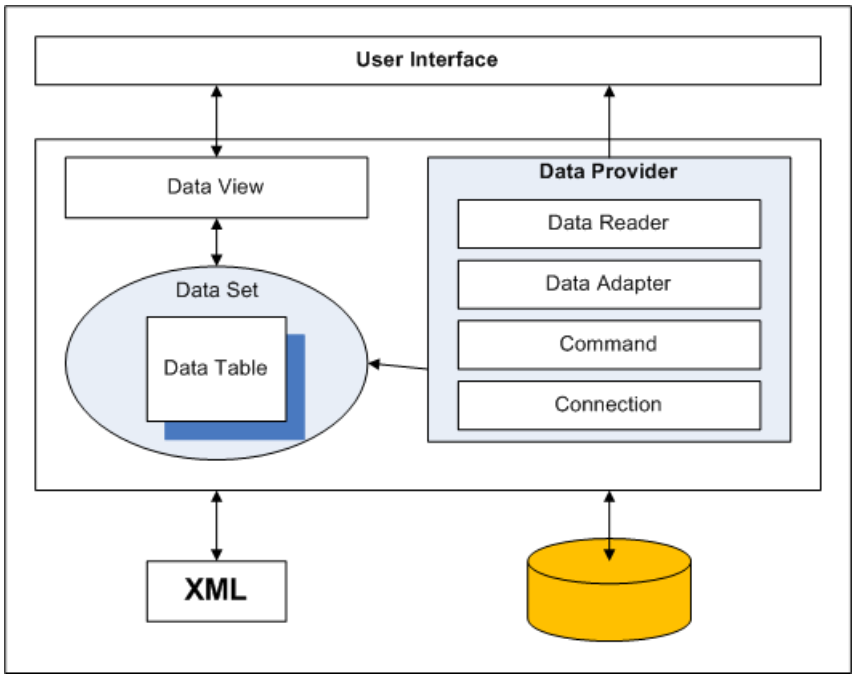


Figure 4.3. *Architecture of ADO.NET [9]. ADO.NET consists of two main components DataSets and Data Providers. DataSets are in-memory data stores that only hold data. Data Provider takes care of connections and updating of data sources according to the changes in the DataSets.*

into the machine-specified binary. In order to interact with databases, the .NET uses ADO.NET that contains the two main components DataSets and DataProviders. DataSets are used for retaining data in memory and DataAdapters in DataProviders take care of connections and performs updating of the data source from the changes in the DataSets.

Chapter 5

Implementation and verification

5.1 Preliminary

The designed solution in chapter 3 has been implemented using .NET framework, C# as programming language and MS SQL Server 2005 as database management system. The implementation environment and the database management system have been chosen by Scania, since they are the most compatible with Scania's system. Figure 5.1 visualizes the implemented solution and the relation between different packages. The objective of this chapter is to describe the structure and verification of the implemented solution.

5.2 Database

The database, that has been implemented in MS SQL Server 2005, has two main parts, the common part and the EMS-specific part. The common part will contain general information of parameter identifiers, units, scaling, types and files' names and version numbers. The following tables have been used for storage of this information.

- **CommonID:** CommonID contains all common identifiers and their general information and consists of four columns CID, ReadFunction, Description and freezeFrame. CID, the primary key, is a hexadecimal value used for identifying a parameter. CID consists of 6 characters because a common identifier consists of two bytes. ReadFunction is the name of the function used for reading from the parameter. FreezeFrame is a Boolean value and is checked if the parameter is a freeze frame.

- **LocalID:** LocalID is the same as CommonID. The only difference is about the hexadecimal identifier LID (analogue to CID) column that consists of four characters because a local identifier consists of one byte.

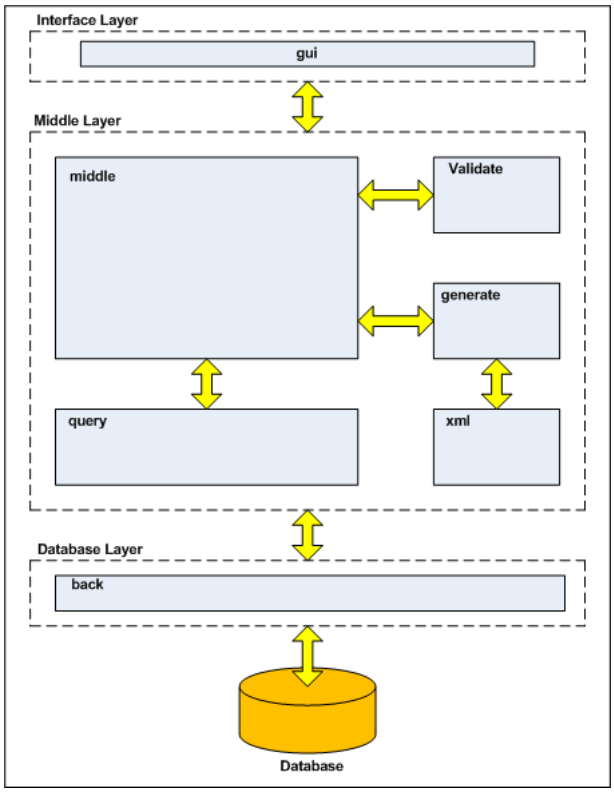


Figure 5.1. *Architecture of implemented solution and relation between packages.*

- **Files:** It consists of only one column, FName, used for storing the name of the files. The action "ON DELETE CASCADE" is used when refering to FName.
- **Version:** Version consists of only one column, VerNr, used for storing version numbers. The action "ON DELETE CASCADE" is used when refering to VerNr.
- **File_Ver:** File_Ver contains information about a file, its version, author and creation date. The FileVr column, the primary key, is a combination of file name and file version. FName and FVersion are two foreign keys that refer to the File and Version tables. The action "ON DELETE CASCADE" is used when refering to FileVr.
- **UnitFormat:** UnitFormat contains engineering units and their symbols. The table consists of four columns HexId, UnitName, Symbol and Description. HexId, the primary key, is a hexadecimal value which identifies a unit. It consists of four characters because the value uses only one byte. Symbol contains the engineering symbol of the unit.

- **Types:** Types contains the variable types used for transforming responding information by an ECU to engineering units. The table consists of three columns HexId, Type and description. HexId, four characters and primary key, is a hexadecimal value for identifying a type. The identifier consists of one byte and the left nibble decides if the variable is signed and the right nibble decides the variable size. Type contains the name of the type.

- **ScalingID:** ScalingID contains scaling formulas. The table consists of three columns HexId, Formula and Description. HexId, four characters and primary key, is a hexadecimal value for identifying a scaling formula. Formula contains the textual form of the formula.

The main reason for using three different tables Files, Version and File_Ver instead of only one table is to make possible using the database for different ECUs. In this way any kind of combination of file names and versions is possible.

The EMS-specific part has been implemented according to the different groups of parameter identifiers in KWP2000 EMS ECU-S7 [4]. There is a table for every group of parameter identifiers and there are two tables for data used for transforming. The following tables have been created in the EMS-specific part.

- **ScaniaCID:** ScaniaCID stores Scania Common identifiers from 0x0000 to 0xF000. The table consists of twelve columns. The primary key consists of two columns CID that refers to CID in CommonID and FileVr that refers to FileVr in File_Ver table. TableNr is a number that refers to one of the tables Scaling or ScalingEXT. The Scaling column refers to a record in the referred table by TableNr. When adding a record to the table the columns CID, ReadFunction, Description and FreezeFrame are copied from CommonID.

- **InOutCID:** InOutCID stores inputOutputCommonIdentifier from 0xF000 to 0xF200. The table consists of fourteen columns of which the primary and foreign keys are analogue to ScaniaCID.

- **EOLParams:** EOLParams stores parameters used in EOL programming from 0xF200 to 0xF600. The table consists of eleven columns of which the primary and foreign keys are analogue to ScaniaCID.

- **ReadSignals:** ReadSignals stores parameters in read signals that are grater than 0xF600. The table consists of fifteen columns of which the primary and foreign keys are analogue to ScaniaCID.

- **Scaling:** Scaling stores data used for scaling. The table consists of nine columns. Sc_Id, the primary key, is an integer that is referred by Scaling column in EMS tables. The column FileVr refers to FileVr in File_Ver table and Nr is a record number in the table used by a file with name and version in referred record in File_Ver table. Formula, Unit and VarType refer to HexId in ScalingID, HexId in UnitFormat and HexId in Types respectively.

- **ScalingEXT:** ScalingEXT stores data used for scalings that have extra data. The table is the same as Scaling but it has a more column Unit2, that refers to HexId in UnitFormat. The column is used for units that have a prefix.

The implementation has been performed according to the explained specification above. Figure A.1 in appendix A, visualizes a diagram over the implemented database. In the implemented database some of the tables have a primary key consisted of one column and the others of two columns. In those tables that have a primary key with one column the Second Normal Form (2NF) and the Third Normal Form (3NF) are guaranteed. Because in the tables with a primary key consisted of a few columns there is no attribute that depends on a part of the primary key the 2NF is guaranteed. The 3NF for those tables are also guaranteed because there is no attribute in a table that depends on a non-key attribute. Thus the 3NF is guaranteed for the whole database. Since there is no multivalued dependency in any table there is no need to control the Boyce-Codd normal form.

5.3 Application

As mentioned .NET, C# and Visual Studio 2005 have been used as development environments for implementing of the solution. Using a N-Tier architecture, has made it possible to have several layers in order to reach an appropriate level of maintenance. Using the data access technology, ADO.NET, with the N-Tier architecture, makes it possible to take care of source consuming connections automatically and in an acceptable way. The DataAdapter in Data Provider, which is a part of ADO.NET, opens and closes connections to the database automatically, when a user performs modification of data.

As figure 5.1 shows, classes have been implemented in three layers: Interface, Middle and Database. Every layer contains corresponded namespaces (analogous to packages in Java). The following namespaces have been used.

- **Gui:** Gui contains all classes used for presenting retrieved data to the user and inserting modified data by the user.
- **Middle:** Middle contains all classes used for communicating with the database.
- **Validate:** Validate contains classes used for performing control of inserted data and validation of user.
- **Query:** Query contains classes used for creating a query.
- **Generation:** Generation contains classes used for generating tables and XML-files.
- **Xml:** Xml contains all classes used for generating the XML-file.

- **Back:** Back, which is a very thin back-end, contains a class for connecting to the database and performing the queries. All classes connect and retrieve data via this class.

Every table has its own form in order to add data to it and it also has its own class in order to perform the action according to the inserted data into the form. Figure A.2 visualizes the main frame with data stored in the InOutCID table and figure A.3 shows the implemented form for adding data to InOutCID.

5.4 Test and verification

According to chapter 2 the objective of the project was to design and implement a solution that could fulfill the demanded requirements and specifications in that chapter. Test and verification are performed according to those requirements and specifications. In order to test, if all parameter identifiers have been identified, tables used in two versions of EMS-specified software were generated from the data stored in the database. The fact of discovering no differences between generated tables from the data stored in the implemented database and existing tables of the same versions, emphasized the correctness of identifying parameter identifiers used in EMS.

Individual tests for every class used for modifying contents of a table have been performed without any particular problem and all requirements and specifications for storing, adding, editing, modifying and deleting the data are fulfilled. Since every table has its own form and action class with no interaction with the classes of the other tables, the integration test for all classes is less comprehensive.

Test and verification of the generated XML-file was more difficult and time consuming. All requirements in order to verify an XML-file were fulfilled. But when configuring XCOM, using the generated XML-file, XCOM either was not able to recognize most of the parameter identifiers or was not able to read from or write to the ECU. After two weeks, it was found that the order of some nodes in the same level of the XML-tree were important. In other words it was important that some nodes were generated before others in the same level. After solving the problem, the implemented solution was working correctly and all requirements and specifications mentioned in chapter two were fulfilled.

5.5 Summary

Implementation has been performed according to the designed architecture in chapter 3. The Database that has been implemented in MS SQL Server 2005 contains common and EMS-specified parts. .NET framework, C# and Visual Studio 2005 have been used for implementing the application. The data access technology, ADO.NET, with a N-Tier architecture that contains the three layers Interface, Middle and Database, has been used. DataAdapter in Data provider, which is a

part of ADO.NET, takes care of all connections and all modifications take place only via a thin back-end in the database layer. Test and verification of the solution have also been performed. Verification of the generated XML-file was the most time consuming. It was the order of generated nodes at the same level of the XML-tree that caused the problem. Configuring XCOM with the generated XML-file required that some nodes at the same level of the XML-tree were generated before the other nodes.

Chapter 6

Conclusion

As mentioned in section 2.3 and 2.4 the objective was to design and implement a solution for storage of parameter identifiers and their information. Up to now test and verification of the designed and implemented solution emphasize that the requirements and specifications are fulfilled.

Since the solution has been designed for developing PC-tools used for Powertrain Control System development it is most compatible for developing EMS. In order to use the solution with other ECUs some improvements can be performed.

6.1 Improvement of the solution

If Scania wants to use the solution for the other ECUs it is necessary to add some more components. One of the other ECUs that Powertrain Control System consists of is GMS (Gearbox Management System). Using the solution even for GMS requires adding an application analogue to EMS. Adding forms and classes for performing update actions are necessary.

The database will even need some modifications. The common part will not need any table added to it because it contains all parameters used in OBD and of course the PIDs used by GMS. The common part also contains the information that SCOMM uses for communication with all ECUs. But a new GMS-specified part should be added for storing the parameter identifiers and their information used by GMS.

6.2 Limitation of the solution

The EMS-specified part of the database has been designed and implemented according to the parameter identifiers stored in the EMS-specified source-code file. The source-code file contains tables consisting of PIDs and their information, and also tables that contain information used by SCOMM for transforming received data from EMS to engineering units. This means that the tables of the database have columns corresponding to the columns in the tables in the source-code file. It

means that there is some limitation in using the solution. The limitation is about changing the construct of the tables. If for some reason, some columns of a table of the source-code file are deleted or added, the same modification should take place in the corresponding table in the EMS-specified part of the database. This also requires some modifications in the application in order to perform update actions.

Appendix A

Figures

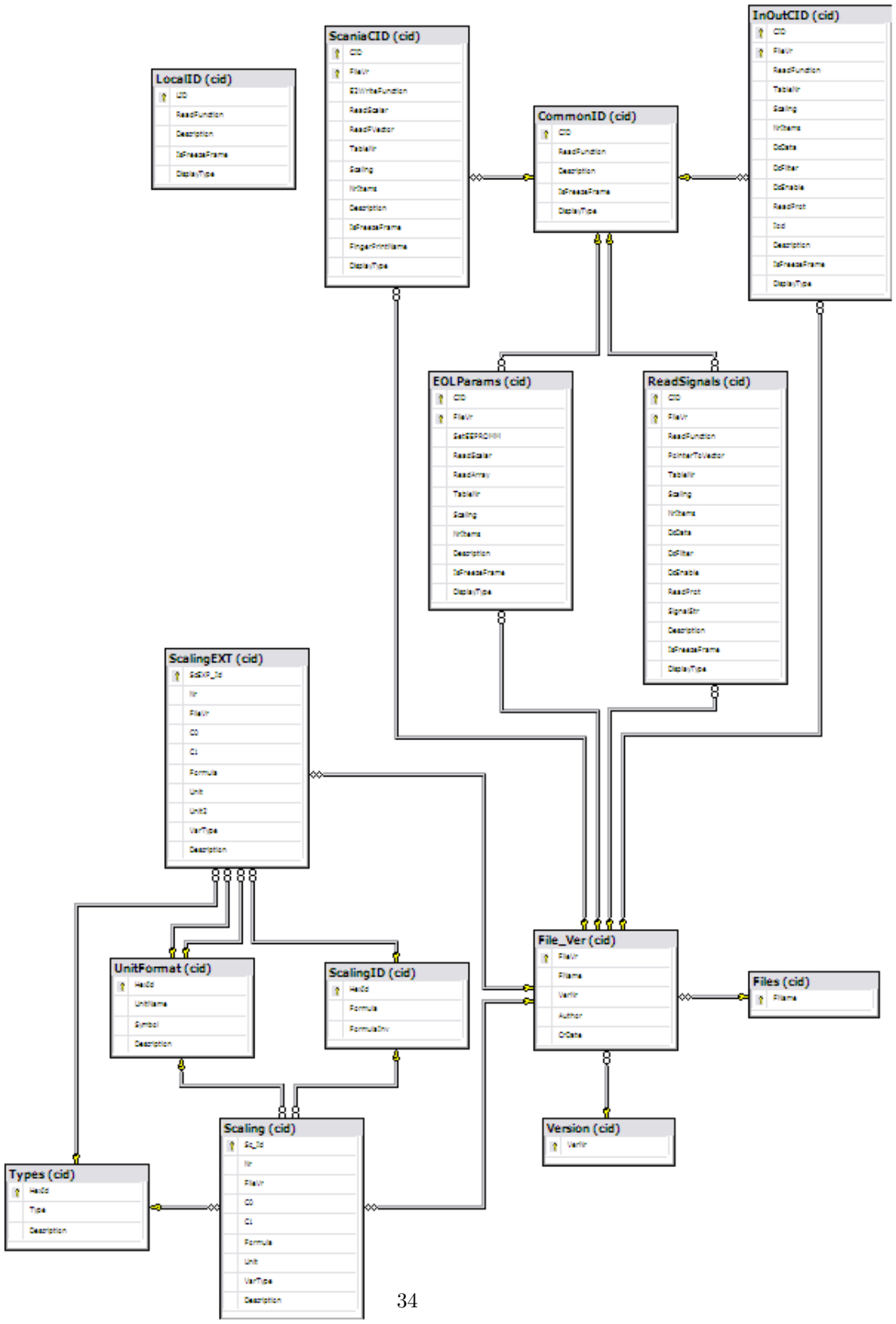


Figure A.1. *Implemented database in MS SQL Server 2005.*

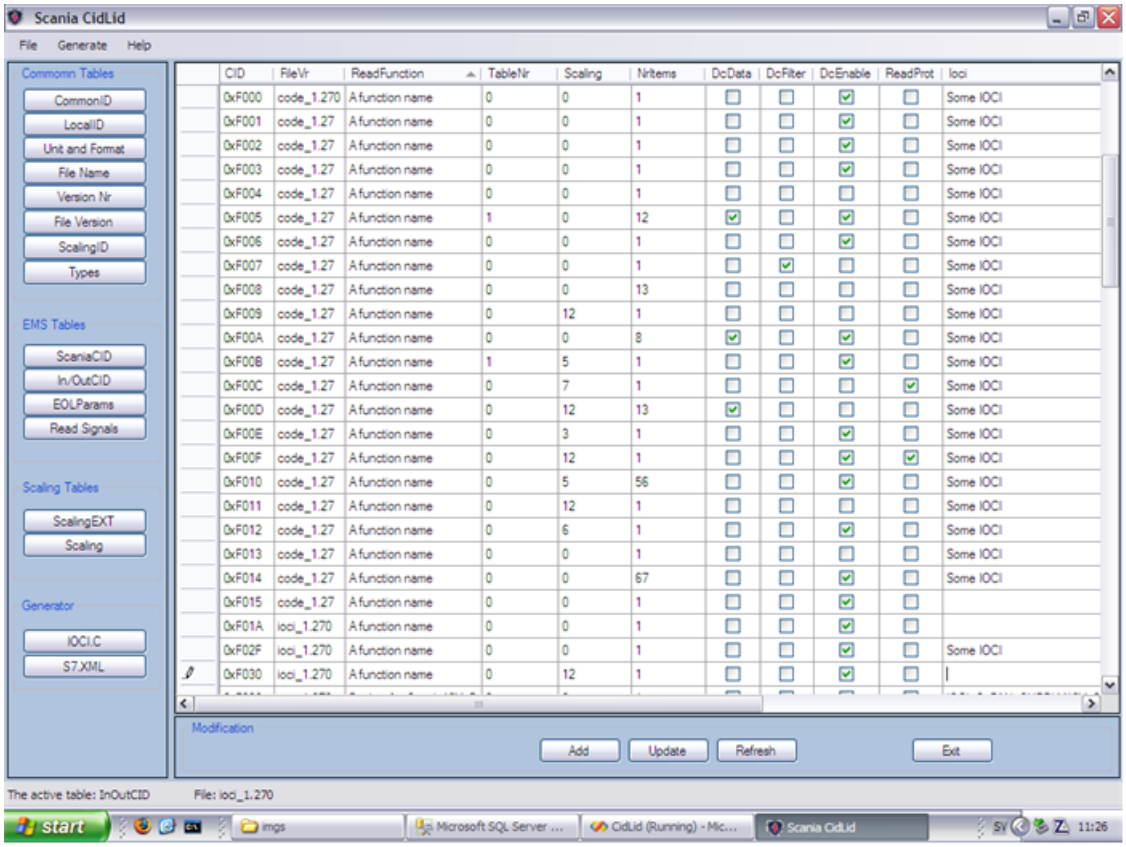


Figure A.2. *Main frame of the implemented solution "CidLid" with an edited table.*

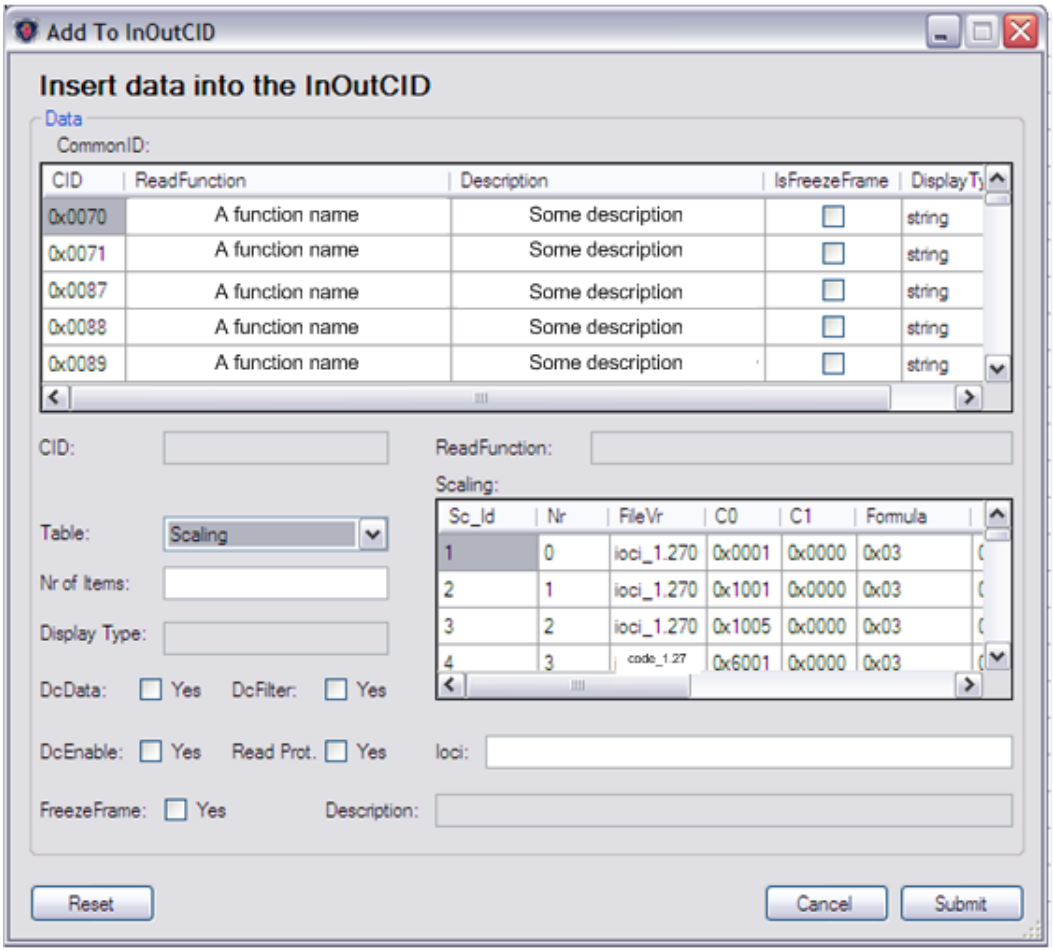


Figure A.3. *User interface for adding to Input/Output Common Identifier table.*

Appendix B

Abbreviation

2NF Second Normal Form

3NF Third Normal Form

ASAM Association for Standardization of Automation and Measuring System

BCL Base Class Library

CAN Controller Area Network

CID Common Identifier

CLI Common Language Infrastructure

CLR Common Language Runtime

CLS Common Language Specification

CPU Central Process Unit

CTS Common Type System

ECMA European Computer Manufacturers Association

ECU Electrical Control Unit

EMS Engine Management System

EOL End Of Line

IL Intermediate Language

IOCI Input Output Common Identifier

JIT Just In Time

LID Local Identifier

MVCI Modular Vehicle Communication Interface

NEVE A division of NE (department for powertrain control system development), responsible for support, maintain and develop of the PC-Tools used by NE

OBD On-Board Diagnostic System

ODX Open diagnostic data exchange

OSI Open System Interconnection

PE Portable Executable

PID Parameter Identifier

SCOMM Scania Communication Module. An internally produced program for performing communication between ECUs.

XCOM An internally produced off-board diagnostic program

Appendix C

User Manual

C.1 Introduction

CidLid is an abbreviation for Common and Local identifier. The goal of designing CidLid program is:

- To store parameter identifiers used by Engine Management System (EMS) for different versions of the softwares used by EMS.
- To perform adding, deleting, editing and modifying actions on stored parameter identifiers.
- To generate ioci.c, the source-code file used in EMS.
- To generate the XML-file used for configuring XCOM.

C.2 Main Frame

CidLid is compatible with Windows environment and MS SQL Server has been used for storing parameter identifiers and their information. When user starts the program he/she is encountered with a main frame, see figure C.1. The main frame consists of eight parts which are described below.

1. **File Menu**
2. **Common Tables**, with buttons used for performing actions on common tables used by all ECUs i.e. Common and Local identifiers.
3. **EMS Tables**, with buttons used for performing actions on EMS tables used by EMS i.e. Scania Common identifiers.
4. **Scaling**, with buttons used for performing actions on tables used for scaling.

5. **Generator**, with buttons used for generating tables in the source-code file (ioci.c) and generating S7LAB.xml.
6. **Modification**, with buttons used for performing add, delete and update the edit actions on tables.
7. **Status Menu**, the area used for showing active table name, file and version.
8. **Edit Area**, the area used for editing contents of the active table[1].

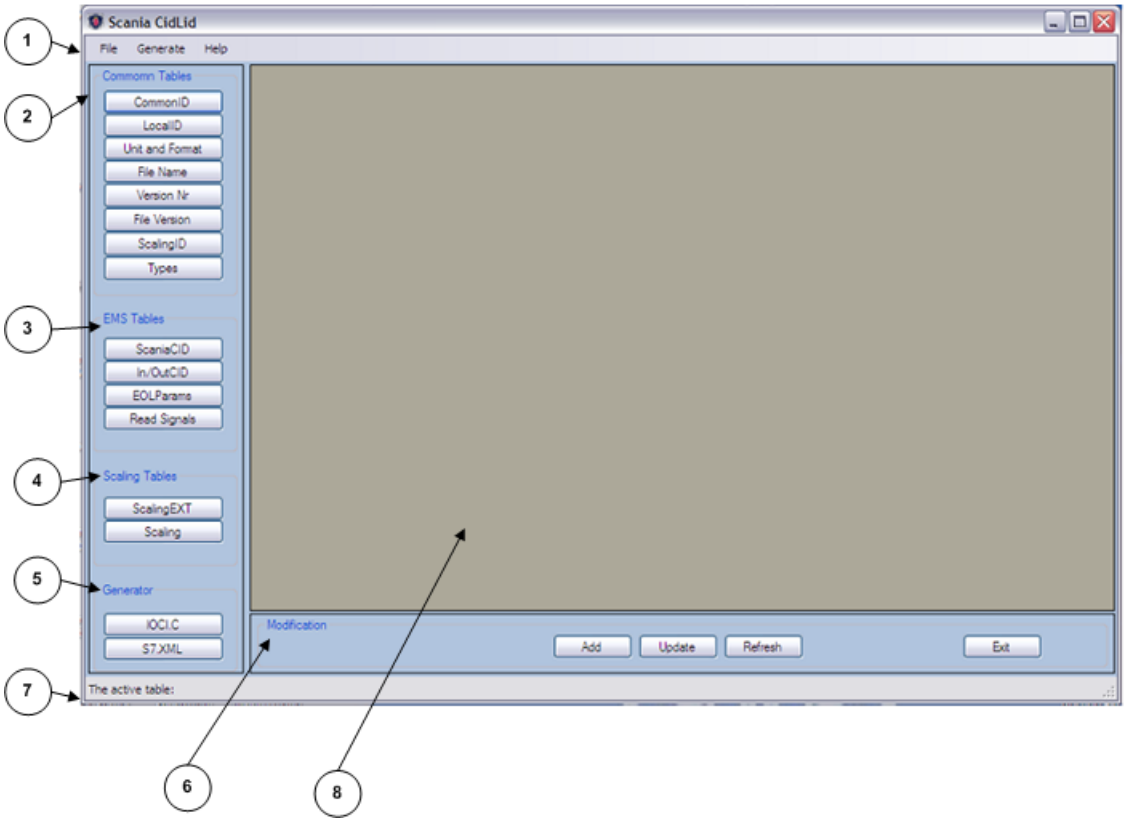


Figure C.1. *Main frame of CidLid.*

C.3 Add frame

When user presses Add-Button in the Modification button group in order to add to the active table, an appropriate adding from will be appeared. All adding forms consists of three parts. See figure C.2.

[1] Active table is the table that is edited in the edit area.

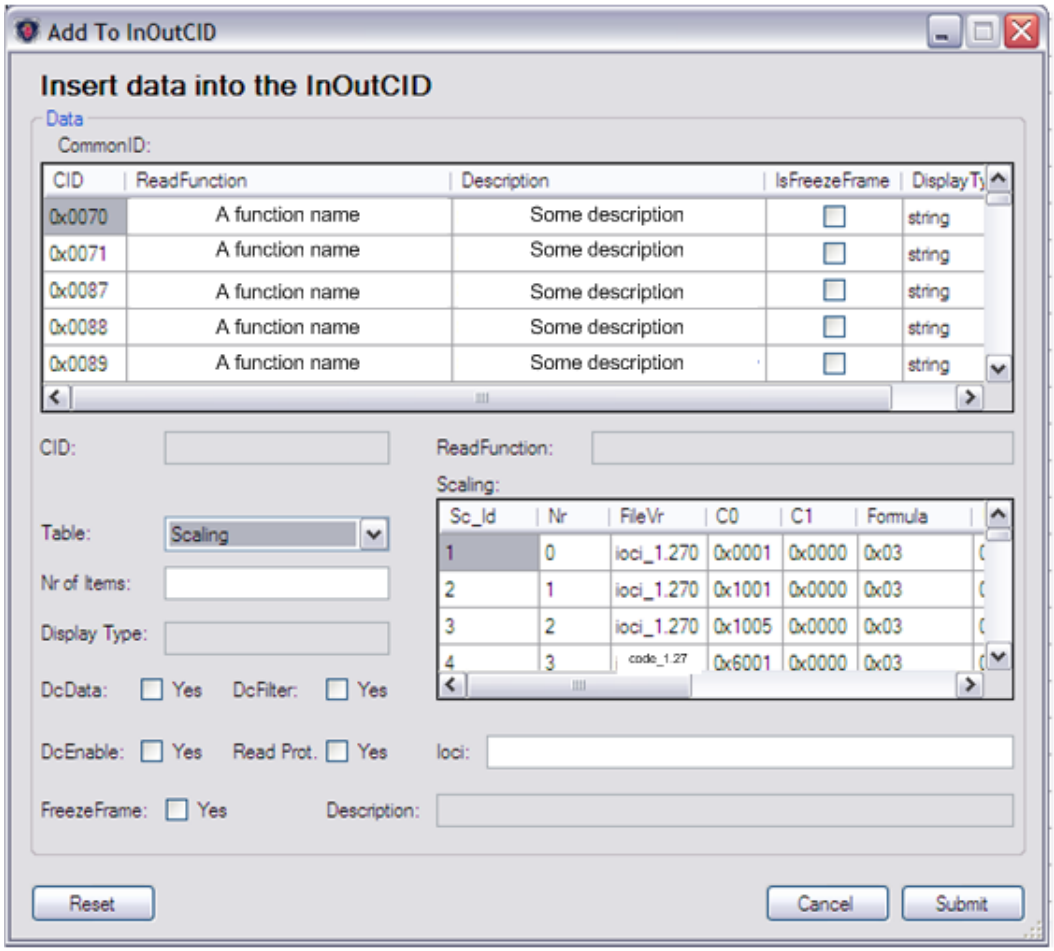


Figure C.2. *Add frame for InOutCID.*

1. **Description Area** shows which table is going to add.

2. **Data Area** is used for inserting the desired data.

3. **Submit Area** is used for submitting or canceling the action.

C.4 Instructions

Creating a new version of ioci and adding data to it, is probably the point you want to start using this program. Lets start from adding a new version to the database.

C.4.1 Adding a new version to the database

1. First add a name to "File Name" according to C.5.1, if you don't have a file with the desired name in the Files table. You can see the existed file names by pressing the File Name button in the main frame.

2. Add a new version number to "Version Nr" according to C.5.1, if you don't have the version number in Version table. You can see the existing version numbers by pressing the Version Nr button.

3. Add a combination of file and version to "File Version". For example select ioci as file name and 1.270 as version number.

4. Press submit button. After pressing the submit button, a combination name will be created. The combination name consists of the selected name and version (i.e. ioci_1.270). Then the program will ask you if you want to copy data from an older version, existed in the database, to the created version. You can select "Yes" if you want to copy data from an older version and use it as template for the just recently generated version. Perform the following steps in order to copy data from an older version.

   a) Press "Yes" button in the window asking about the copy. A new window will be opened.

   b) In the opened window asking for a source file select an older version. In the ComboBox you can see even the just generated version. Selecting the recently generated file and version has no affect, so select an older file and version.

   c) Press submit button. By pressing submit button the program will copy data from the older file and version to the new one in all tables in EMS and Scaling groups. If something is going wrong a transaction will roll back and even the recently created file and version in "File Version" will be removed.

C.4.2 Editing tables

In order to edit a table click on the desired button or select File, Open and then desired table . If the table is one of the common tables it will be showed in edit area. If the selected table is a table in EMS tables or Scaling tables a windows asking for selecting a file and version will be appeared. After selecting file and version number, the contents of the selected table for selected file and version will be showed in the edit area.

C.5 Modifying an active table

In order to modifying contents in a table perform the following steps.

1. First edit the table according to C.4.2.
2. Perform the modifications in the edited table.
3. After inserting the modifications press Update-Button in order to update the data source in the database. **OBS!** Without pressing the Update-Button the data source will not be updated and you will loose the modification if you close the program.
4. In order to control if the update has been performed successfully press Refresh-Button.

C.5.1 Adding to a table

When adding to an active table it will add the inserted data into the active table for selected file and version. If you want to add a common identifier to one of the EMS tables and the common identifier does not existed in CommonID table you first have to add the common identifier to CommonID table.

- **Common tables:** Perform following steps in order to add to a in the common tables.
  1. First edit the table according to C.4.2.
  2. Click on the Add-Button in Modification button group or select File, Add in the file menu. An appropriate frame correspond to the active table will be opened.
  3. After inserting the data press Submit-Button. The inserted data will be added to the table and a window, saying the action was performed successfully, will pop up.
- **EMS and Scaling tables:** In order to add to a table in EMS or Scaling perform the following steps.
  1. First edit the table according to C.4.2.
  2. Click on the Add-Button in Modification button group. An appropriate frame correspond to the active table will be opened.
  3. In the opened window first select which scaling table you want to use. After selecting "Table" you will see the existed data in selected table in Scaling text area. Select your desired row from Scaling text area.
  4. Fill other text fields and press Submit-Button.

  When adding to EMS tables some of the text fields are not available. Those text fields will be filled automatically when you select a CID from CID text area. That is why it is important to update CommonID and LocalID in the common tables regularly. If you don't see the common identifier you want to add, first add it to CommonID.

C.5.2 Delete a record in a table

In order to delete a record in a table perform the following steps.

1. First edit the table according to C.4.2.
2. Select the row containing the record and delete it by pressing the Del key on your keyboard.
3. Press the Update-button in the modification area.

C.5.3 Delete a version of a file

In order to delete a version of a file perform the following steps.

1. First edit the File Version table according to C.4.2.
2. Delete the record containing the version of the file according to C.5.2.

OBS! In this way you will not only delete a record from File Version but you also delete all data that belong to the deleted version of the file from all other tables.

C.5.4 Delete a version from Version

In order to delete a version perform the following steps.

1. First edit the Version table according to C.4.2.
2. Delete the record containing the version according to C.5.2.

OBS! In this way you will not only delete a record from the Version table but you also delete data for all files that have the deleted version from all tables. For example if there is two files of the same version, test1_1.2 and test2_1.2, and the version 1.2 in the Version table is deleted all data belong to the both files will be deleted from all tables.

C.5.5 Delete a file from Files

In order to delete a file perform the following steps.

1. First edit the Files table according to C.4.2.
2. Delete the record containing the file name according to C.5.2.

OBS! In this way you will not only delete a record from the Files table but you also delete data for all versions of the file from all tables. For example if there is two versions of the same file, test_1.2 and test_1.3, and the test in the Files table is deleted all data belong to the both versions will be deleted from all tables.

C.5.6 Generate ioci and S7LAB

CidLid has appropriate components in order to generate tables used in ioci or generate S7LAB, the XML-file used by EMS. The generated S7LAB is serialized and is stored in *home*\CidLid\generate\xml, in which *home* is the path of the installing directory. For example if you install CidLid in C:\ you will find S7LAB in c:\CidLid\generate\xml. You can find the generated ioci in *home*\CidLid\generate\src.

- **Generate the tables in ioci**

  In order to generate the tables in ioci.c perform the following steps.

  1. Select Generate, Ioci.c in the file menu or press the IOCI.C button in generator button group.
  2. In the opened window, asking for a source file, select a file as source file.
  3. In the opened window, asking for a target file, select a file and version that you want to generate.
  4. Press the submit button. After pressing the submit button a new file will be generated. You can find the generated file at *home*\CidLid\generate\src with the selected target file as its name.

  In the generated file, code chunks are copied from the selected source file and the tables are generated form stored data in the database for the selected target file and version.

- **Generate S7LAB** In order to generate S7LAB perform the following steps.

  1. Select Generate, S7.xml in the file menu or press the S7.XML button in generator button group.
  2. In the opened window, asking for a file and version, select a file and version.
  3. In the opened window, asking for metadata, insert your desired data.
  4. Press the submit button. After pressing the submit button a new xml-file, with the name "S7LAB_*version*", will be generated. The generated xml-file is serialized and you can find it at *home*\CidLid\generate\xml.

C.5.7 Printing contents of a table

In order to print contents of a table perform the following steps.

1. First edit the table according to C.4.2.
2. Select File, Print in the file menu.
3. In the popped up window, asking about centering the contents of the table in center of the printed page, press "Yes" if you want to have it.

The Print preview in File in the file menu is working in the same way as print.

Bibliography

[1] http://www.scania.com/about/scaniahistory/ (visited in June, 2008)

[2] http://en.wikipedia.org/wiki/Engine_control_unit#History (visited in June, 2008)

[3] 14230-3s.DOC/Samarbetsgruppen för Svensk Fordonsdiagnos
http://www.mecel.se/ssf.htm

[4] Technical Product data, Diagnostic Services Specification, KWP2000 EMS ECU-S7

[5] ISO 11992-1:APR2003 Road vehicles - Interchange of digital information on electrical connections between towing and towed vehicles - Part 1: Physical layer and data link layer

[6] ISO 14230-1:MAR1999 Road vehicles - Diagnostic systems - Keyword protocol 2000 - Part1: Physical layer

[7] ASAM MCD-2D (ODX) Version 2.2, Association for Standardization of Automation and Measuring Systems, date 29.02.2008, Data Model Specification.

[8] CANdelaStudio with ODX,
http://www.vector-worldwide.com/vi_index_en„223.html (Visited in June, 2008)

[9] Pro C# 2008 and the .NET 3.5 Platform Fourth Edition by Andrew Troelsen, Publisher: Apress

[10] http://www.dotnetlanguages.net/DNL/Resources.aspx

[11] ECMA C# and Common Language Infrastructure Standards.

[12] Beginning C# 2008 Databases From Novice to Professional. Publisher Apress

TRITA-CSC-E 2008:110
ISRN-KTH/CSC/E--08/110--SE
ISSN-1653-5715

www.kth.se