SAE TECHNICAL PAPER SERIES

2006-01-1568

Experiences with the ODX Diagnostic Database Standard

Kevin Mullery, Gareth Leppla, David Boyd and Brendan Jackman
Waterford Institute of Technology

ISBN 0-7680-1633-9

9 780768 016338

2006 SAE World Congress
Detroit, Michigan
April 3-6, 2006

SAE International™

400 Commonwealth Drive, Warrendale, PA 15096-0001 U.S.A. Tel: (724) 776-4841 Fax: (724) 776-5760 Web: www.sae.org

The Engineering Meetings Board has approved this paper for publication. It has successfully completed SAE's peer review process under the supervision of the session organizer. This process requires a minimum of three (3) reviews by industry experts.

ISSN 0148-7191

2006-01-1568

Experiences with the ODX Diagnostic Database Standard

Kevin Mullery, Gareth Leppla, David Boyd and Brendan Jackman
Waterford Institute of Technology

ABSTRACT

ODX (Open Diagnostics data eXchange) is a standard diagnostic data format specified by the European ASAM group, to simplify the exchange of vehicle diagnostic data between manufacturers, suppliers and service dealerships. The ODX database contains diagnostic data for the whole vehicle together with details of all vehicle ECUs.

This paper gives an overview of the main categories of data contained in the ODX diagnostic database. A Windows-based diagnostic application was developed to execute ISO 15765 diagnostic services on ECUs, using an ODX database to define the allowed services and parameters for each ECU. The paper describes the structure of the diagnostic application and the steps that were necessary to process the ODX and tailor it to a production ECU.

INTRODUCTION

ODX is a format for the storage and retrieval of Electronic Control Unit (ECU) diagnostic data for a single vehicle. The process of designing a vehicle usually involves many companies developing systems, sub-systems etc, this therefore means that a lot of ECU diagnostic data is passed between these companies. The inherent problem with this approach is that it lacks proper structure, the data that is passed between the companies is usually in a paper trail, and usually with no agreed format.

ODX attempts to solve this problem by allowing data to be entered into a standard format. An ODX formatted database (ODX XML file) may then be built up for a single vehicle containing all of that vehicle's ECU diagnostic data. Once this ODX database has been built it may be passed to testing engineers and service dealerships to perform diagnostic testing and checking of a vehicle.

ODX INFORMATION

As mentioned previously an ODX file contains ECU diagnostic data for a single vehicle. It has been designed to reduce redundancy of data and to also hold all of the information that is required to perform diagnostic checking on a vehicle. For this reason it stores a lot of different information, some of which can be seen in figure 1.

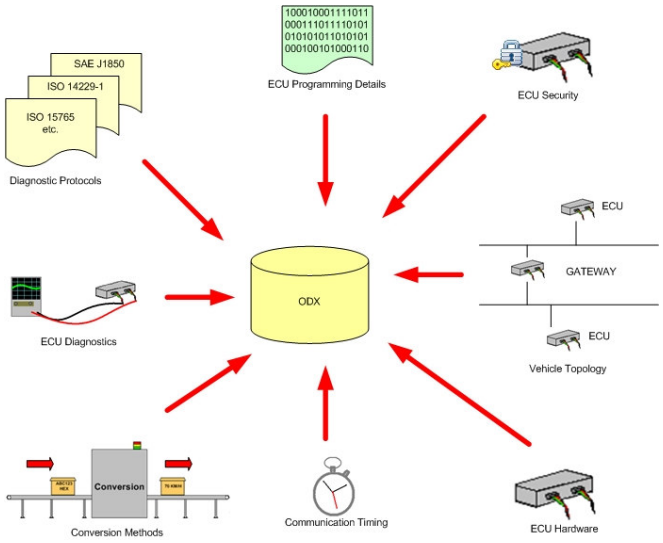


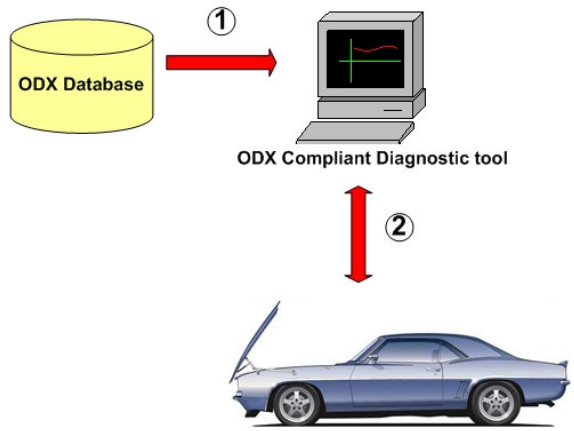
Figure 1 Subset of ODX Information Types

As can be seen the information contained within the ODX database is varied, however it all aids in allowing the vehicle to be diagnostically monitored.

Figure 1 only shows a subset of the different information that can be contained within the ODX database, there are many other types. One other type of information that it contains is administrative and company data. These two categories of information allow the different areas of the ODX document to be fully traceable to whoever filled it out. This introduces much more organization and accountability regarding the compilation and use of the ODX database.

ODX COMPLIANT TOOLS

For the purposes of diagnostically testing and servicing a vehicle an ODX compliant tool may be used. This is a usually a computer program that will connect to a vehicle or single ECU and then to that vehicle or ECUs ODX file. By referencing the different information in the ODX file the computer program will be able to perform diagnostic checking and programming of ECUs within the vehicle. Figure 2 shows a typical setup for an ODX compliant tool on a desktop computer connected to a vehicle.



1. Diagnostic tool reads the relevant data from ODX to perform Diagnostic testing of the vehicle

2. Diagnostic tool performs communication with the vehicle to retrieve diagnostic Information

Figure 2 ODX compliant tool communicating with an ODX file and a vehicle

An ODX database allows a diagnostic tool to be more generic. All of the data to diagnose a vehicle can be taken from an ODX file. Therefore to perform diagnostics on a number of different cars just requires an ODX compliant tool connecting to a number of different ODX files. The ODX file that the tool connects to changes, not the tool itself. This is a much better approach as opposed to storing static data, such as conversion algorithms in the tool.

XML OVERVIEW

This paper does not intend to explain XML in any detail. However since the ODX specification is written in XML format, a quick introduction will aid the reader for the remainder of the paper.

XML (eXtensible Markup Language) is a format for storing information, and is very similar to HTML (Hyper Text Markup Language). It contains information within named tags called elements that indicate the type of data enclosed within the start tag and end tag as shown in figure 3.

Figure 3 XML element example

An example of a collection of three books will clarify this.

```
<Books>
  <Book>
      <Title>Java Gently</Title>
      <Author>Judith Bishop</Author>
      <ISBN>0-201-71050-1</ISBN>
  </Book>
  <Book>
      <Title>Modern Control Systems</Title>
      <Author>Richard C.Dorf</Author>
      <ISBN>0-201-03147-7</ISBN>
  </Book>
  <Book>
      <Title>Formal Specifcations Using Z</Title>
      <Author>David Lightfoot</Author>
      <ISBN>0-333-76327-0</ISBN>
  </Book>
<Books>
```

Figure 4 Quick XML example

In this example information is being stored on three books. As can be seen the outermost element is 'Books' which stores information about one or more books. Each 'Book' element in turn stores information about the 'Title', 'Author' and 'ISBN' number of each book. These elements in turn store the actual information about the books. As can be seen from figure 4 an element may contain another element and text. The information is very well structured and is easy to interpret.

In ODX and for the rest of this document XML elements such as 'Books' are referred to as objects. This quick tutorial is sufficient enough for the rest of this paper. For further tutorials on XML try http://www.w3schools.com.

OVERVIEW OF THE TOP LEVEL ODX CATEGORIES

The ODX structure has five main categories. These are:

- DIAG-LAYER-CONTAINER
- COMPARAM-SPEC
- MULTIPLE-ECU-JOB-SPEC
- VEHICLE-INFO-SPEC
- FLASH

These five categories hold different types of information within an ODX file, which can be seen in table 1.

ODX-CATEGORY	Information Type
DIAG-LAYER-CONTAINER	Diagnostic
COMPARAM-SPEC	Communication
MULTIPLE-ECU-JOB-SPEC	For communication with multiple ECUs
VEHICLE-INFO-SPEC	Vehicle Information
FLASH	ECU Programming

Table 1 ODX categories

An ODX XML file (ODX instance) may contain only one category at a time. This means that diagnostic information and communication parameter information cannot be held within the same ODX XML file. Figure 5 shows the start of two separate ODX files, the first will contain diagnostic information as shown by the DAG-LAYER-CONTAINER object, and the second will hold communication information as shown by the COMPARAM-SPEC object. One file cannot contain both types of information.

Figure 5 Different Categories in ODX XML.

A brief overview of each ODX category is given.

DIAG-LAYER-CONTAINER

This is the largest and most important of the ODX categories. This category contains diagnostic information about the vehicle that the ODX file is modeling. This area is discussed in greater detail later in this paper.

MULTIPLE-ECU-JOB-SPEC

This category is used to allow an application to communicate with more than one ECU at a time. This is similar to functional addressing, however functional addressing is not used.

COMPARAM-SPEC

This ODX category holds information that is needed to allow effective communication with an ECU. Examples of such communication parameters include:

- Target Address of an ECU
- Timeout values for communication with an ECU.
- Maximum number of times a message request can be sent to an ECU.

This area is discussed in greater detail later in this paper.

VEHICLE-INFO-SPEC

This category deals with modeling information about the physical vehicle. This is split into two main areas. The first area involves information that actually identifies the vehicle, such as OEM, MODEL-YEAR, VEHICLE-MODEL, etc. The second area holds information about the vehicles network topology. This includes information such as VEHICLE-CONNECTOR, which models the various connections in the vehicle by holding information on each pin using the VEHICLE-CONNECTOR-PIN object. Other objects model the communication links between ECUs and hold information such as the type of physical communication link (e.g. CAN, LIN). Gateways are also modeled and various links are made to diagnostic models also. This modeling of the topology allows an application quick and easy access to various ECUs, regardless of gateways or bus type, which is transparent to the user.

FLASH

This category allows a transfer of data between an application and an ECU (ECU memory programming). Here data can be either downloaded to an ECU or uploaded from an ECU. A number of objects (XML elements) in ODX are used to allow these processes to take place. The main objects are:

- SESSION
- DATABLOCK
- FLASHDATA
- PHYS-MEM
- PROJECT-INFO

The combination of these objects allows an ECU to be programmed effectively. They describe the structures of both physical and logical memory, the data to be downloaded, hardware and software details etc.

ODX DIAGNOSTIC DATA

The ODX specification is a very effective tool for modeling diagnostic data of a vehicle. All of the information that is needed to execute a service and receive and interpret the results, are modeled. This makes it very easy for a diagnostic application to grab the diagnostic services it needs from the ODX file, send them to an ECU, and be able to use the conversion methods from the ODX file to interpret the returned results. Using this type of model a diagnostic application can be very generic, as it does not need much data to perform diagnostics on an ECU besides the ODX file. This allows diagnostics to be performed on different ECUs by simply changing the ODX file.

DIAGNOSTIC SERVICES

The core component of the ODX specification is the service. A service is a command that is sent to an ECU

for it to perform some type of action. This action maybe for the ECU to turn on an actuator, send back a single result, send back periodic results, etc.

A service consists of both a request and a response. A request is a message that is sent to an ECU, and a response is a message that is sent from an ECU after it has received a service request, as can be seen in figure 6. A service is modeled in ODX by a DIAG-SERVICE object. There are also REQUEST and RESPONSE objects respectively.

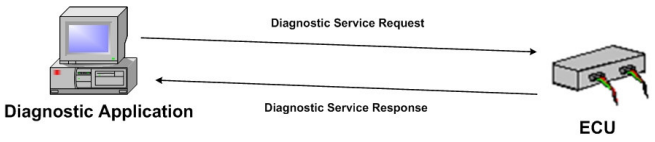


Figure 6 Executing a diagnostic service, using a Request and Response.

Most of the other information contained within the diagnostic category of the ODX data model (DIAG-LAYER-CONTAINER) is related to the diagnostic service in some way. For example conversion methods, data types etc.

DIAGNOSTIC LAYERS

All of the diagnostic information in the ODX diagnostic model is contained within five different diagnostic layers. These are namely:

- PROTOCOL
- FUNCTIONAL-GROUP
- ECU-SHARED-DATA
- BASE-VARIANT
- ECU-VARIANT

All of these layers are very common; they all contain similar data, such as diagnostic services, but are grouped in such a way as to add more structure to the ODX data model.

The core of these layers are the two layers BASE-VARIANT and ECU-VARIANT. This is because these represent actual ECUs. When a service needs to be executed, (i.e. send a request to an ECU) the related ODX file must have one of these two layers defined for the physical ECU.

Figure 7 shows a tree view of these diagnostic layers in a diagnostic ODX file. The SHORT-NAME object here just gives the diagnostic file a name. Each diagnostic layer is contained within a wrapper object. This is simply the name of the Layer with a 'S' added to the end, for example the PROTOCOL layer has a wrapper object called PROTOCOLS.

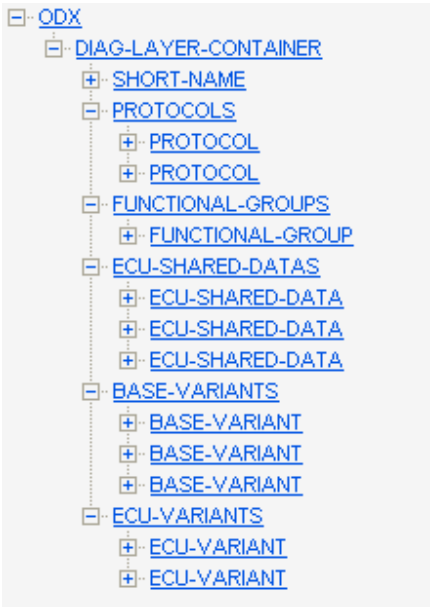


Figure 7 Tree view of a standard ODX file

As can be seen there can be a number of the same type of diagnostic layer, in one diagnostic ODX file. A brief description of each is given.

Protocol

This layer contains information on diagnostic protocols that are being used with the ECUs, such as ISO 15765 (Diagnostics On CAN). The information contained is mostly the different services that the protocol contains, and related information such as request structure, response structure, conversion methods etc.

Functional-group

This layer holds services that relate to a specific set of ECUs, such as airbag ECUs. Each set of ECUs has their own functional group. This layer contains information mostly on the services that belong to the selected group.

ECU-shared-data

This layer contains information that might be shared between different ECUs, such as services, conversion methods, etc. The reason for this layer hence is to reduce redundancy by keeping all of this shared data in the one layer, instead of reproducing it in a number of layers.

Base-variant

This layer contains information about an ECU. Each instance of this layer represents a different ECU. This layer holds all of the services that belong to this ECU alone. These would usually be non-standardized vendor specific services.

ECU-variant

This layer represents a later version of a base variant. For example if a BASE-VARIANT (ECU) was version 1.0, then the ECU-VARIANT may be version 1.1. Essentially this is just a later version of an ECU that is already stored in the ODX file. This layer can be seen as a child of BASE-VARIANT, inheriting all of its information, such as its services, but also having information that belongs to only this later version, such as later services that are not available to the original ECU version (BASE-VARIANT). The information held in the ECU-VARIANT layer is only diagnostic information that is new to this version alone, and all of the other diagnostic information is derived from the original ECU version. This again has been done to reduce redundancy, and hence reduce the overhead of storing the same data for each ECU version in a number of places. Naturally an ECU-VARIANT has to be linked to an existing BASE-VARANT.

PACKING/UNPACKING AND CONVERSION OF DIAGNOSTIC DATA

Before data can be sent out as parameters of a service request, it must be changed so that it can be successfully sent and interpreted by an ECU, and vice versa when the service response is being sent back to the application. This requires knowing characteristics of the service and data being sent, such as conversion of returned values. ODX has allowed for this type of information to be modeled. The main object in ODX that models this is known as a DATA-OBJECT-PROP (DOP).

This object describes how to:

- Convert data for a request or from a response.
- Pack data into the request/unpack data from the response.
- Represent the returned data as a unit, e.g. mph, km/h, etc.

These are described briefly below.

Conversion of Data

Data that is sent as part of a service request, or that is received as part of a service response sometimes needs to be converted into either a coded-value or a physical value. A physical value is a number that is typically aided with a unit, such as 128mph, 5000rpm, etc. A coded value is a number that is interpreted by an ECU to represent this physical value, for example a certain ECU might interpret 123ABC hex as 50mph. Figure 8 shows an example of this.

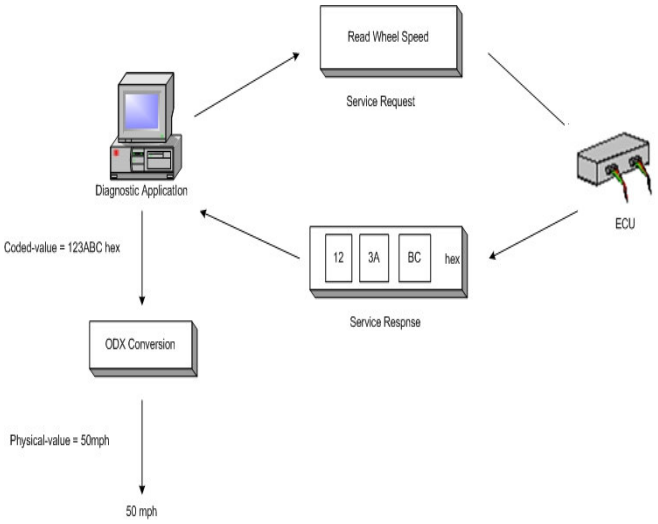


Figure 8 Conversion of retrieved data

Here a service request to read the wheel speed is sent from a diagnostic application to an ECU. The ECU returns a service response, with a coded-value of 123ABC hex. The diagnostic application using the appropriate ODX file converts this coded-value into a physical-value. Here 123ABC hex is converted to give a wheel speed value of 50mph.

The object in ODX used to carry out this conversion is the COMPU-METHOD object. This can allow values returned to be converted in a number of ways, by using linear functions, rational functions and even converting a returned value into a word or phrase. For example a value 23hex returned from an ECU might be converted to “All doors locked”.

Packing/unpacking data

A service Request/Response being sent to/from an ECU at a low level can be seen as data that has to be packaged contiguously and sent along a communication link, such as CAN. Using the DIAG-CODED-TYPE ODX object all of the data for a diagnostic service can be packed without the application needing to know how it is done. This also will work the same in reverse, the data is unpacked from the contiguous data stream.

Units

To aid the interpretation of the returned values units can be used, such as mph, rpm, amps, etc. Units allow some more meaning to be attached to the values. The units can also be converted to units within the same group. For example a value could be returned as 50 after conversion, a unit that is linked to it might tell us that it is 50mph, this value can now be converted to a different unit and be converted to maybe 80km/h.

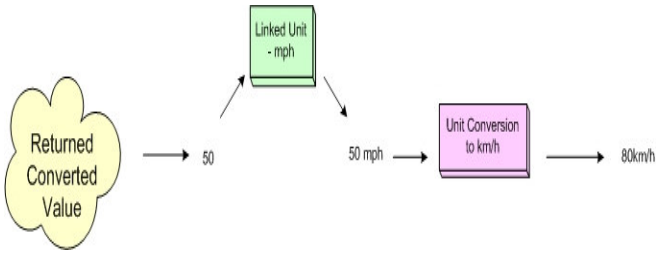


Figure 9 Unit Conversion

COMMUNICATION PARAMETERS IN ODX

Communication parameters are parameters that can be set to control the communication of messages between an application and the involved ECU or ECUs. These are usually values that are changeable and can be set by a technician. The communication parameters are varied and usually depend on the diagnostic protocol being used such as KWP2000, Diagnostics on CAN, etc. Table 2 below shows two typical communication parameters and their uses, specified for the Diagnostics on CAN protocol.

Communication Parameter	Description
P2 CAN Client	Time for the application to wait after the successful transmission of a request message, for the start of the incoming response messages
Request Count	The number of requests an application will make to the ECU before it will stop trying to communicate

Table 2 Example of communication parameters

All of these communication parameters are given a value to allow successful communication.

Communication parameters can be stored in an ODX file for easy retrieval by storing them as a number of COMPARAM objects. A COMPARAM is an object that holds information such as a name, a description and a value for a given communication parameter. All of these COMPARAMs are themselves held inside a COMPARAM-SPEC object. A COMPARAM-SPEC is a top level ODX category that is used to group a number of COMPARAMS together. Common groupings might be 'Diagnostics on CAN COMPARAMS', 'KWP2000 COMPARAMS', 'Optimum throughput COMPARAMS', 'My personal COMPARAMS'.

OBTAINING COMMUNICATION PARAMETERS FOR A DIAGNOSTIC LAYER

A diagnostic layer in a separate ODX XML file can inherit communication parameters. This inheritance allows a diagnostic layer to be able to retrieve the values of the appropriate communication parameters.

A protocol layer in a separate ODX XML file inherits the communication parameters. All of the services (DIAG-SERVICES) within this protocol layer now have communication parameters that they can use, when they are being sent to or from an ECU. For any other diagnostic layer (i.e. FUNCTIONAL-GROUP, ECU-SHARED-DATA, BASE-VARIANT or ECU-VARIANT) to inherit these communication parameters, they must inherit the protocol layer. Figure 10 shows how the communication parameters are propagated down (inherited) to the different diagnostic layers.

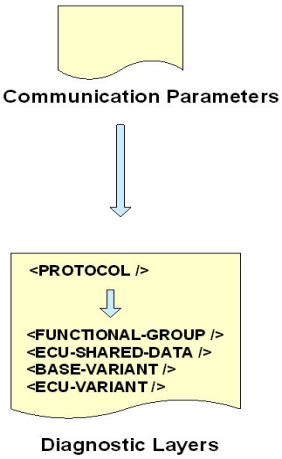


Figure 10 Propagation of communication parameters

Once a diagnostic layer has inherited the communication parameters, then these communication parameters are available to all of the services that belong to that diagnostic layer.

OVERRIDING COMMUNICATION PARAMETERS

Having an exterior ODX XML file containing a set of general communication parameters is a very good idea, as it allows this file to be referenced by many diagnostic layers in many diagnostic ODX XML files. This reduces unnecessary redundancy that would occur if the same set of communication parameters (COMPARAMS) had to be copied for each diagnostic ODX XML file. However a diagnostic layer or service may need to change one or more communication parameters, from that specified in the higher layer that it has inherited from. This does not mean that a new communication parameters file will need to be created, but simply that the values that need to be changed are overridden (overwritten). This is done by creating a reference to the communication parameter which needs to be changed, via its ID attribute and setting it a new value for this reference alone. It is important to note that the value only changes where it is referenced, but the original communication parameter value still is unchanged. This is achieved by using an object called a COMPARAM-REF.

A communication parameter can be overridden for a diagnostic layer or for a service. If it is overridden for a diagnostic layer, then all of the services in that diagnostic layer now use the overridden communication parameter value as opposed to the original value. If it is overridden for a service, then only this service will use this new value for communication.

The following example clarifies this better.

Figure 11 shows a communication parameter called P2CANClient, which would be stored in a separate ODX XML file to the diagnostic services. Figure 12 shows a BASE-VARIANT diagnostic layer. For the purpose of this example it is taken that the base variant diagnostic layer in figure 12 has inherited the communication parameters from figure 11. In figure 11 it can be seen that P2CANCLIENT has a value of 23. However in figure 12 this value has been overridden (overwritten) using a COMPARAM-REF and given a new value of 45.

```
<COMPARAM-SPEC ID="CS1">
  <SHORT-NAME>General_Communication_Parameters</SHORT-NAME>
  <COMPARAMS>
    <COMPARAM ID="A.P2CANCLIENT.ID">
      <SHORT-NAME>P2CANCLIENT</SHORT-NAME>
      <PHYSICAL-DEFAULT-VALUE>23</PHYSICAL-DEFAULT-VALUE>
    </COMPARAM>
  </COMPARAMS>
</COMPARAM-SPEC>
```

Figure 11 Original communication parameter

```
<BASE-VARIANT ID="BASE.VAR.1">
  <SHORT-NAME>BASE_VARIANT_1</SHORT-NAME>
            :
  <COMPARAM-REFS>
    <COMPARAM-REF ID="A.P2CANCLIENT.ID">
      <VALUE>45</VALUE>
    </COMPARAM-REF>
  </COMPARAM-REFS>
            :
</BASE-VARIANT>
```

Figure 12 Overriding a communication parameter

IMPLEMENTATION OF A DIAGNOSTIC APPLICATION USING ODX

A diagnostic application was developed which was used to test the ODX standard. The application is used to perform diagnostic testing of an ECU, and is known as ACTS (Automotive Calibration and Testing System). It is designed to be a generic diagnostic testing tool that performs diagnostic checks on any ECU that has a corresponding ODX database. The application is primarily aimed at developers and testing technicians. The application uses the Diagnostics on CAN protocol (ISO 15765), however the main concepts can be applied to all applications regardless of what protocol is used. Figure 13 below shows how the application can be connected to the ODX database and an ECU.

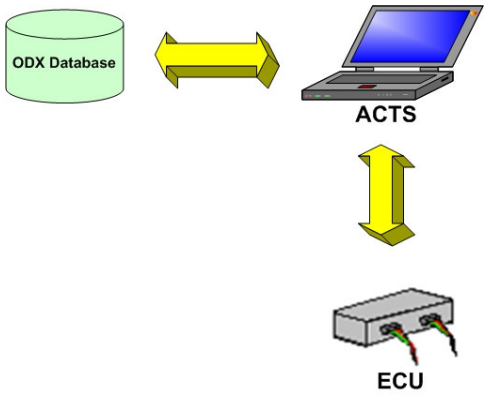


Figure 13 ACTS application communicating with ODX file and ECU

As can be seen, this is the basically the same as the diagram for an ODX compliant tool (figure 2).

ODX DATA AND ACTS

The application was built to be very generic and therefore it relies on the ODX file for a lot of information. All of the services, timing parameters, addressing information, conversion methods, semantic data, packing data, etc, are taken directly from the ODX file. This means that no programmed data is needed, and that data is not referenced from multiple sources. All of the data that is needed is contained within the ODX file. Figure 14 below shows the flow of some types of data between the ODX file and the ACTS application.

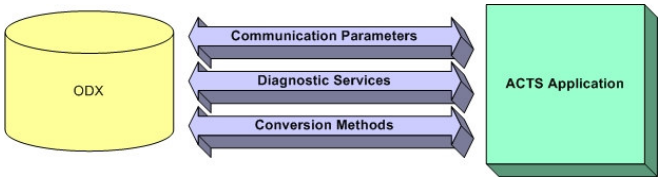


Figure 14 Retrieving and sending to data from/to the ODX file

The double sided arrows are used as the data is retrieved from the ODX file for execution of services, communication parameters, etc and sent to the ODX file, as the application allows an ODX file to be created and edited also.

CREATION OF AN ODX FILE

This application allows the user to create and edit ODX files. This is achieved by using a number of windows, which hide the ODX XML from the user. The user only needs to know very basic knowledge about the ODX data model to be able to create an entire ODX database.

Two of the core ODX objects the user can create are:

- Diagnostic Services.
- Communication Parameters.

Diagnostic Services

As mentioned previously, diagnostic services are the core component of the ODX data model. The user is able to create a standard diagnostic service specified by a protocol or create a unique user defined service. Creating a service involves creating a service request and a service response. Both the request and the response can have any number of parameters, which can be set by the user. The user can also pick services that they might need from the Diagnostics on CAN protocol, without having to create them.

Figure 15 below shows a screenshot of the service creation window in the ACTS application.

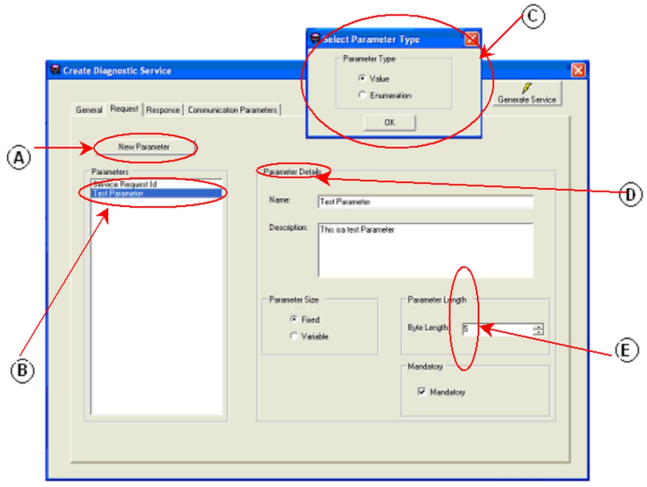


Figure 15 Creating a service request.

A brief description of the labeled parts of this window is given.

A. New Parameter. This adds a parameter to the request.
B. Parameter list. This displays all parameters that have been added to the request.
C. This allows the user to select what type of parameter to add to the request, this is displayed when a user clicks “New Parameter”
D. Parameter Details. This displays all information about the parameter as it is chosen from the parameter list.
E. An example of the value type parameter.

Communication Parameters

The communication parameters that the user can set consists of addressing information and timing information. The timing information can be set for a diagnostic layer or for a service. Allowing the user to set the communication parameters means that they can test many different timing values, and pick the timing values most applicable to that ECU.

Figure 16 Shows a screenshot of the communication parameters tab in the ACTS application.

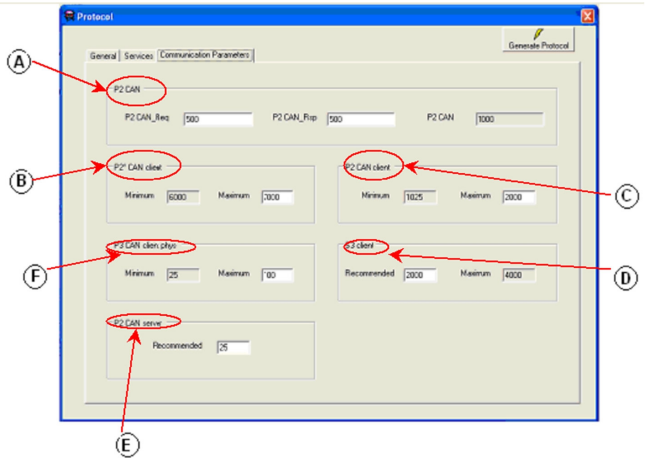


Figure 16 Changing communication parameters.

This window was designed to allow the user to change Diagnostics on CAN communication parameters.

The highlighted areas are shown in table 3.

Highlighted Area	Communication parameter	Description
A	P2CAN	Holds the time that a request and a response take during communication.
B	P2*CAN client	Enhanced version of P2CAN client. Used if the ECU requests more time to respond.
C	P2CAN client	Timeout for the client to wait after the successful transmission of a request message for the start of incoming response messages
D	S3 client	This timer keeps the non-default session active by sending a tester present request every time that this timer times out.
E	P2CAN server	Performance requirement for the server (ECU).
F	P3CAN client physical	Minimum time for the client to wait after the

		successful transmission of a physically addressed request message with no response required before it can transmit the next physically addressed request message

Table 3 Communication Parameters given for ACTS.

SETTING & EXECUTING A DIAGNOSTIC SERVICE

The application lets a user execute a diagnostic service. The user first selects the relevant ODX file. Then the user is shown a number of different ECUs, which are documented in the ODX file. An ECU can then be selected. A list of services is then shown for the selected ECU. A service can then be selected. A list of parameters for the selected service is then shown. Each parameter can be then be filled. Once all of the parameters are filled, the service can be executed.

The process that has just been described was all done by extracting information from the ODX file.

Figure 17 below shows the service execution window in the ACTS application.

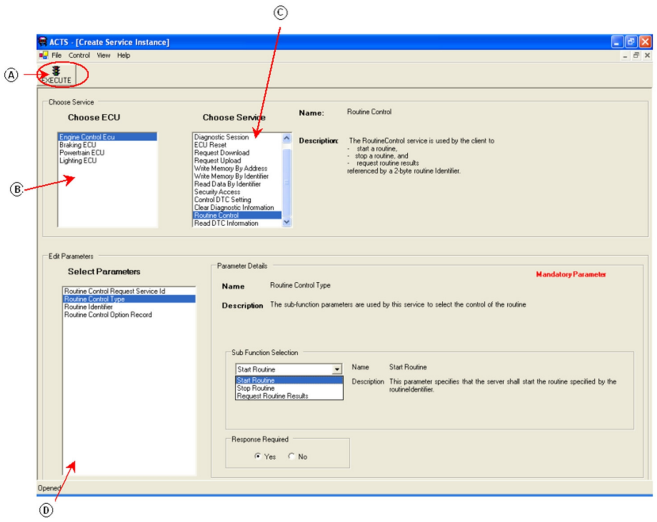


Figure 17 ACTS execution window

Figure 17 shows the execution window, containing all of the ECUs, services and parameters for a given ODX file. Validation ensures that a badly formed service request can't be sent to the ECU.

The highlighted areas are:

A. Execute Button.
B. ECU selection list.
C. Service selection list.
D. Parameter selection list.

RECEIVING A DIAGNOSTIC SERVICE RESPONSE

Once a service request has been sent, then it will only take seconds for a response to be returned. The service response initially is just a stream of data. However the relevant service response information is pulled from the ODX file and this is hence used to interpret the raw data that is returned. The diagnostic information is then presented to the user.

TOOL VERIFICATION

The ACTS application was used to test diagnostic services for a production ECU. It was able to communicate with the ECU, execute services and receive accurate results. This showed that the ODX data model was implementable for diagnostic monitoring of production ECUs.

GUIDELINES ON APPROACHING THE ODX SPECIFICATION

The ODX specification is a large document, with over 440 pages, and is slightly confusing when you start to read it. The reason for the confusion in part is that it introduces terms and concepts early on that are not explained until much further into the specification. It is for this reason that this section has been included in the paper.

The points listed below should make reading the specification easier. These points have been compiled for a person who will need to know ODX in-depth and so might not be applicable for everyone.

- Learn how to interpret the UML (Unified Modeling Language) diagrams early. They are used extensively through the specification and a clear understanding of these will aid you greatly. The specification gives a quick and effective tutorial at the start that should not be ignored. It is not necessary to learn UML in-depth.
- Learn XML. This like UML diagrams is used throughout the specification and should therefore be learned. It is also the format that the ODX model is written in, and so to have any in-depth understanding of ODX, understanding it is essential. It makes the UML diagrams easier to understand also, as the diagrams can be cross-referenced with the ODX examples at the back of the specification for a clearer understanding. The following tutorial is easy and quick and should be sufficient, www.w3schools.com/XML/default.asp.
- This ODX specification as mentioned previously is based around diagnostic services. Therefore a clear understanding of a service is required. Reading a diagnostic protocol such as the ISO 15765 specification is strongly advised. The added benefit of this is that there are examples at the back of the specification of these protocols

in the ODX format, which will clarify some questions that might have arose.
- The use of linking and references is used a lot in the ODX specification. However a proper explanation of these is not given until much further in the specification. This area should be definitely looked at early on. Skipping ahead and reading these areas in the specification will make understanding the specification much easier.
- There is a very helpful section towards the back of the specification called "Data model implementation in XML". This is a very good section for clarifying many areas in the ODX specification. This section should be checked frequently to see if it can help.
- If you have some computer programming background, it may be worthwhile to write a program that will represent the ODX examples at the back of the specification as a tree of branching nodes. A tree view of an ODX file would allow you to see the structure and nested objects of an ODX file more clearly. This is not as difficult as it might seem. There are many free examples of tree views on the Internet that can be copied and pasted very easily. Figure 7 shows a tree view of the different diagnostic layers.
- Read the specification more than once. A second reading of the specification will make it a lot clearer and clarify some uncertainty that might have been experienced in the first reading.

If you follow one or all of these guidelines the specification should be easier to read and understand.

PROBLEMS ENCOUNTERED WITH ODX

ODX is a powerful model for representing ECU diagnostic data. However although ODX is a very good model as a whole, it also has a number of problems. Some of these have been listed below. It should be noted that these problems were encountered in ASAM MCD-2D (ODX) version 2.0.

- A large problem with the ODX specification but not the data model is that it is badly written. The specification launches into terms and concepts early on in the specification that are not explained until much further on. This makes other areas appear more difficult than they actually are.
- Another problem with the ODX specification is the examples that are given at the back of it. These examples had many mistakes and were generally incorrect in two ways. Firstly they were not valid against the ODX data model (ODX XML schema). Secondly they were not valid against normal rules for forming an XML document. This means that the examples have to be corrected before they can be understood.
- The XML Schema can be seen as a template for an ODX file, that represents the ODX data model. However the XML Schema does not match with the ODX data model given in the specification. This means that the ODX XML SCHEMA has to be changed also.
- The ODX specification as mentioned previously makes a diagnostic tool more generic, as no static data needs to be hard-coded into the application. However this is generic only to a certain point. The ODX data model is not expressive enough to make a diagnostic application very generic. For example certain services have mandatory and non-mandatory parameters. To perform error checking on these parameters, to ensure that they are properly filled before executing a service, a diagnostic application would need to know which are mandatory and which are not. ODX does not facilitate for this. This problem could be easily solved however by having an IS-MANDATORY attribute for each parameter.
- There is no mention of how to store ECU addresses in the ODX specification. I felt that this should have been shown as this ambiguity could lead to different ODX documents being produced that store addresses in different ways.

The ODX specification is not expressive enough to allow a diagnostic application to be fully generic. This is a short falling of the ODX data model. But besides this point the ODX data model proves that it is very capable of doing what it is meant for. It must also be stressed that the ODX specification is relatively new and as such leaves room to be improved in later versions.

CONCLUSION

ODX is a very expressive and powerful data model. It has many strengths that make it a very good specification. It allows diagnostic applications to be more generic. It attempts to introduce more order to the way in which diagnostic data is documented. It is written in a simple and expressive format, namely XML. However it also has failings, such as it needs to be even more generic than it already is.

ODX as a whole is a good data model. It was used successfully with the ACTS diagnostic application to perform diagnostic checking of a production ECU. This was done very easily, and as such demonstrates the expressive power of ODX.

REFERENCES

1. (2004) ASAM MCD-2D (ODX), Version 2.0.
2. *XML Tutorial.* [Online]. Available: http://www.w3schools.com/xml/default.asp [2006, January 9]

CONTACT

Kevin Mullery B.Sc

Kevin graduated top of his class with a degree in applied computing in Waterford Institute of Technology. He is currently doing a postgraduate in automotive safety critical systems under the supervision of Brendan Jackman. His experience with the ODX standard was gained from work on a diagnostic project team, with David Boyd and Gareth Leppla.

Email: kmullery@wit.ie

Gareth Leppla B.Sc

Gareth has studied at Waterford Institute of Technology for five years. In this time he has acquired a certificate in industrial computing, a degree in applied computing and been awarded a science person of the year award. He gained first hand experience of the ODX standard on a diagnostic project team with David Boyd and Kevin Mullery.

David Boyd B.Sc

David Boyd recently graduated from Waterford Institute of Technology with a degree in applied computing. In his time studying there he also acquired a certificate in industrial computing. David has gained experience of working in automotive diagnostics from working with various diagnostic protocols. He applied this knowledge with Gareth Leppla and Kevin Mullery when working with the ODX standard.

Brendan Jackman B.Sc. M.Tech.

Brendan is the founder and leader of the Automotive Control Group at Waterford Institute of Technology, where he supervises postgraduate students working on automotive software development, diagnostics and vehicle networking research. Brendan also lectures in Automotive Software Development to undergraduates on the B.Sc. in Applied Computing Degree at Waterford Institute of Technology. Brendan has extensive experience in the implementation of real-time control systems, having worked previously with Digital Equipment Corporation, Ireland and Logica BV in The Netherlands.

Email: bjackman@wit.ie

DEFINITIONS, ACRONYMS, ABBREVIATIONS

ACTS: Automotive Calibration and Testing System.

ASAM: Association for Standardisation of Automation and Measuring Systems.

CAN: Controller Area Network.

ECU: Electronic Control Unit.

ISO: International Organization for Standardization.

LIN: Local Interconnect Network.

ODX: Open Diagnostic data eXchange.

UML: Unified Modeling Language.

XML: eXtensible Markup Language