Automatic Check of EDS

Michael Friedrich, Wolfgang Kuechlin, and Dieter Buehler, Wilhelm-Schickard-Insitute for Computer Science, University of Tuebingen

Electronic Data Sheets (EDS) do not always match the current specifications. The user of a CAN device has to check the EDS for syntax and for his application specific requirements. It can be a nuisance to do this manually, especially when the EDS changes often. Therefore, we implemented an automatic EDS Checker to perform this task and ensure the conformance of the EDS with standards and intern policies.

In this paper we present this kind of software, an automatic EDS Checker. The program translates the EDS to an XML representation conforming to the Device Management Markup Language (DeMML). This is also the language for Device Investigator, our device monitoring tool. The actual checking is performed on this XML document. During the checking process corrections to the EDS, like creating unique parameters, can be applied. The individual tests are described in another XML document based on ComputeML. Predefined functions for comparison of strings and standard data types exist. Furthermore, basic tests can be aggregated to more complex ones. The defined tests are shown and a subset can be selected during an EDS check. We present a program which is on one hand easy to use with already defined tests and on the other hand tailor able to special needs.

1 Introduction/Motivation

The use of Electronic Data Sheets (EDS) for the description of CAN devices is specified in the CiA DSP 306 [1]. Despite the specification, the user of a CAN device might have additional restrictions to the EDS. These restrictions can be either syntactically or semantically. For example, enforcing all parameter names to be without blanks is a syntactic restriction and checking whether a given baud rate is supported is a semantic restriction.

Checking the EDS can be annoying, especially during the development cycles, when the EDS changes often due to changing implementations from the device vendor. We implemented a Java and XML-based software to check EDS automatically. The goal was to build a flexible system, which allows creating new checks based on the customers needs without recompiling the whole software. Additionally, the system does not only check the EDS but also can change it to the user's needs.

The rest of the section describes the organization of this paper. An overview over the usage of the EDS Checker is given in the next section. Afterwards, the Transformation and checking process are described in detail and the paper concludes with a summary and outlook.

2 Overview

The EDS Checker is based on the EDS2XML Translator [2] and the XJML_Eval tool developed by Dieter Buehler. The approach is to translate the EDS to a XML representation with the help of EDS2XML and then to apply the XML tools XJML_Eval which performs the tests. The XJML_Eval tool checks XML documents so the EDS has to be translated first. Another reason to separate the concerns of parsing and testing is the flexibility for new data formats to check. Furthermore, a lot of tools like query, transformation and manipulation of documents are available for XML which allows faster and more flexible software development. Additionally, if the EDS format changes, only the EDS2XML Translator has to be changed but not the testing component.

The EDS Checker is implemented in Java and based on two different XML languages. First, the Device Management and Markup Language (DeMML) is used as an intermediate XML format for EDS. Secondly, the ComputeML describes the checks to be applied to the translated

EDS. These languages are described in detail later on (cf. Chapter 3 and 4). Because the EDS can be corrected while tested, it must be translated back from its XML-representation to EDS. The different steps and intermediate formats are outlined in Figure 1.

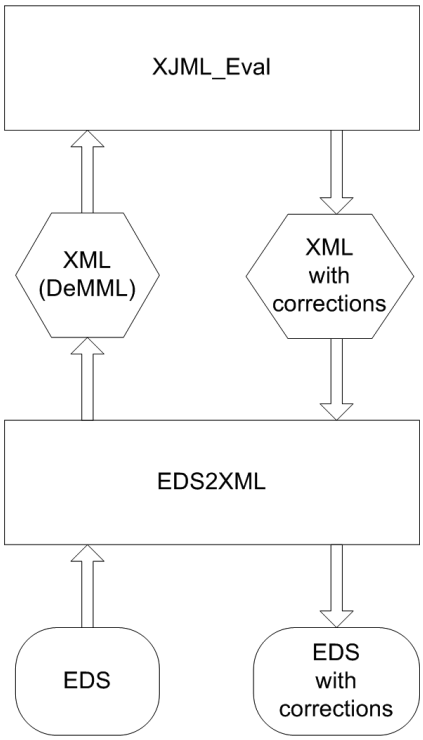


Figure 1: The EDS processing scheme beginning with the original EDS at bottom left.

The types of the predefined checks are nearly unlimited as the EDS Checker can easily be extended with new tests written in Java. A lot of tests are already available in the actual system among these are boolean computations which evaluate to true or false, string manipulations and test, numeric computations, and more. Further details of defining test cases are described in Chapter 4.4.

The EDS Checker has simple graphical user interface (GUI) (cf. Figure 2) which allows the user to check EDS or DeMML documents and to translate between these two formats. The progress and all errors are displayed in the main window. For example, any entry in the EDS which does not conform to the specification is ignored in the translation and displayed here. Choosing the *Check EDS file* button lets the user chose a file and then, a list of available checks is displayed (cf. Figure 3). This list contains all tests previously defined in the EDS Checkers ComputeML file. Among these are tests, if certain baud rates or other features are available. Other tests are more complex like unique or blank free parameter names.

The user commits the dialog after selecting the desired tests. The tests are then applied to the transformed EDS and the results are displayed in a standard browser window (cf. Figure 4). Because of potential corrections to the EDS, it has to be saved again and therefore, a save dialog pops up.

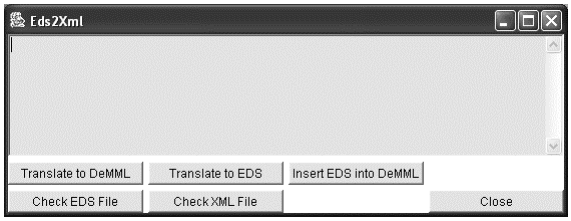


Figure 2: The GUI of the EDS checker.

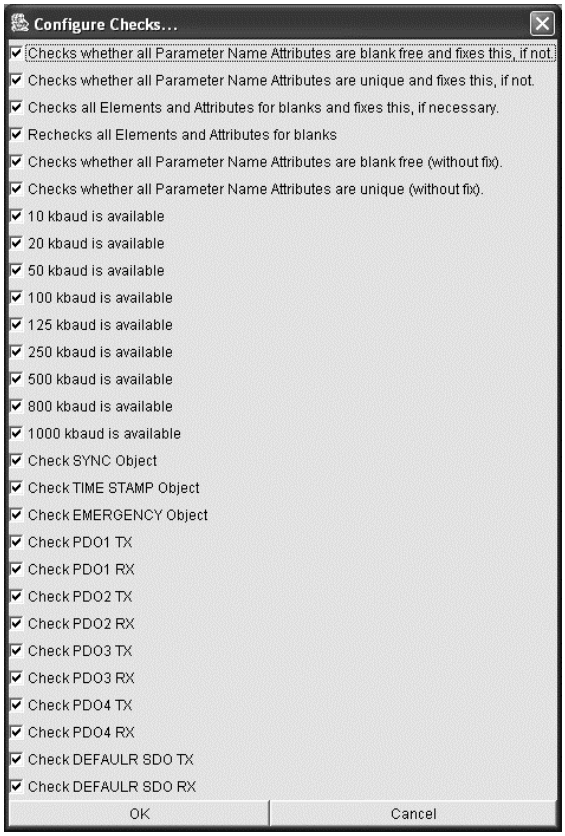
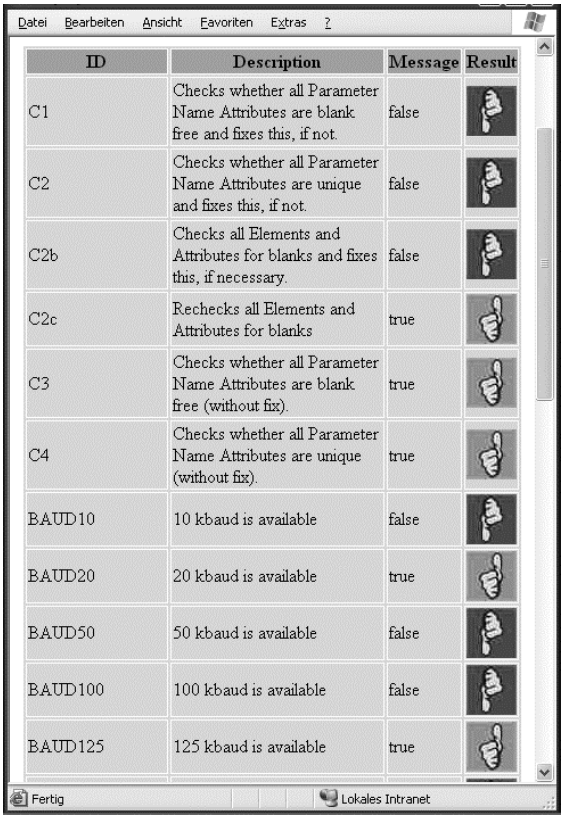


Figure 3: The list of currently defined checks.



ID	Description	Message	Result
C1	Checks whether all Parameter Name Attributes are blank free and fixes this, if not.	false	
C2	Checks whether all Parameter Name Attributes are unique and fixes this, if not.	false	
C2b	Checks all Elements and Attributes for blanks and fixes this, if necessary.	false	
C2c	Rechecks all Elements and Attributes for blanks	true	
C3	Checks whether all Parameter Name Attributes are blank free (without fix).	true	
C4	Checks whether all Parameter Name Attributes are unique (without fix).	true	
BAUD10	10 kbaud is available	false	
BAUD20	20 kbaud is available	true	
BAUD50	50 kbaud is available	false	
BAUD100	100 kbaud is available	false	
BAUD125	125 kbaud is available	true	

Figure 4: The results are displayed as HTML in the host system's standard WWW-browser.

3 DeMML

The Device Management and Markup Language (DeMML) is a universal data format for the representation of devices and groups of devices. It is based on the CANopen Markup Language (CoML) [2] but extends it as it can also store a device configuration and is not restricted to CANopen devices. These features make DeMML more powerful than needed here. Nevertheless, it covers all elements of an EDS. A sample DeMML fragment is illustrated in Listing 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<CANopenDevice>
  <Device DeviceId="4711">
    <FileInfo CreatedBy="XY GmbH"
     CreationDate="05-21-96"
     CreationTime="07:00AM"
     Description="EDS for ABC Device"
     FileRevision="4"
     FileVersion="2"
     FileName="abc.eds"/>
    <DeviceInfo>
      <Vendor Name="XY" VendorId="0"/>
      <Product Name="ABC"
       Revision="1"
       ProductId="0"
       Version="1"/>
    </DeviceInfo>
    <Parameters>
      <SupportedDataTypes>
        <SupportedDataType
         DataTypeId="0x0004"/>
        <SupportedDataType
         DataTypeId="0x0005"/>
        <SupportedDataType
         DataTypeId="0x0006"/>
        <SupportedDataType
         DataTypeId="0x0008"/>
      </SupportedDataTypes>
      <ParameterCategory
       Name="MandatoryObjects">
        <Parameter
         Name="Device Type"
         DataType="7"
         AccessType="ro"
         ParameterId="1000,0">
          <DefaultValue>
            0x30191
          </DefaultValue>
          <ParameterExt
           Index="1000"
           Subindex="0"
           PDOMapping="0"
           ObjectType="7"/>
        </Parameter>
…
      </ParameterCategory>
      <ParameterCategory
       Name="OptionalObjects">
        <Parameter
          Name="Manufacturer  Status
Register"
         DataType="7"
         AccessType="ro"
         ParameterId="1002,0">
…
```

Listing 1: A fragment of a sample XML document

The EDS Checker gets its input for the tests from this document.

4 XJML_Eval

This component performs the actual checks on the transformed EDS. The provided ComputeML document contains the definitions of the tests. Each test consists of a selection of parameters and one or more function calls. The results of the evaluations are entered into a copy of this document and finally transformed with a XML Transformation into a HTML file containing the human-readable results. For an illustration of XJML_Eval and its in- and outputs, see Figure 5.

4.1 Function calls

The XJML_Eval component uses Java functions to evaluate the test results. Given the full control of a programming language, all computations with the input parameters are possible. There are no restrictions for the functions beside their class being accessible. Even database requests are feasible, for example to compare with stored values. As depicted in Listing 2, the test functions are programmed straight forward.

```
public static boolean And (Boolean b1,
                           Boolean b2)
{
  return b1.booleanValue()
      && b2.booleanValue();
}
```

Listing 2: The logical AND computation in Java.

The test functions are loaded dynamically from the resource specified within the test definition. They can be loaded from the local hard disk or over the Internet from a Web-server. The latter way supports the collaboration of device vendor and customer, as the tests can be easily supplied by either side for the other to use.
Test functions get their input from either selections as described in the following chapter or the input can be a result of an other function. Hence, complex tests can be built out of simple test functions without programming in Java by only describing them in XML.

4.2 Selections

The test functions need input parameters which can be evaluated. As mentioned above, the foundation for the tests is the DeMML document. Therefore, XJML_Eval has to extract specific parts of it and pass them to the functions. The selection is performed with the help of XPath, which is another XML tool. XPath allows arbitrary queries in an XML document. More precisely, it is a query over the hierarchy of XML elements in a document. Elements and Attributes can be used as constraints and also wildcards are allowed. For example, the vendor name of a device is of interest. In Listing 1, the vendor name is "XY" and the path in the hierarchy is *CANopenDevice/Device/DeviceInfo/Vendor/@Name* which is also the query string. Note that the "@" denotes the access to an attribute of an element, here in the *Vendor* element. The same query result could be achieved by the search string *//Vendor/@Name*. The double-slash // is a wildcard for any sequence of elements from the root to the vendor element which can only occur once.

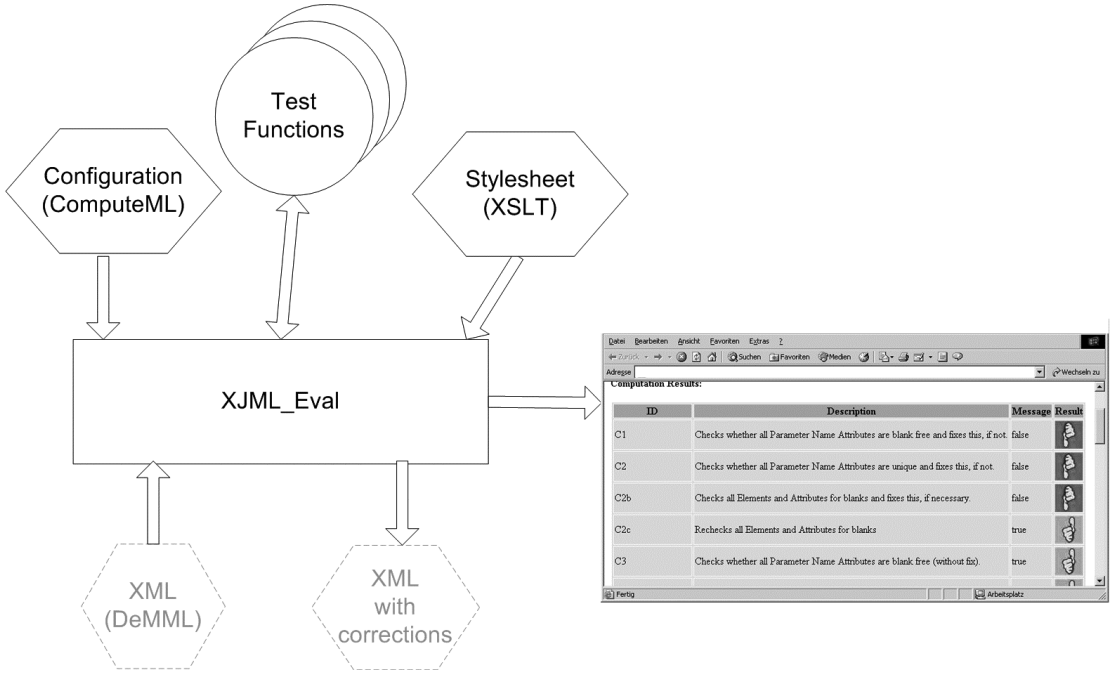


Figure 5: XJML_Eval is configured by a ComputeML document. The test functions can be external and the result is generated according to a stylesheet.

The result set of a query can contain several items which are passed to the test function as parameter. If a function is defined only for single parameters, care must be taken that the query will never have a result set with more than one element.

4.3 Wrapping

The results of a query must be wrapped in Java objects to pass them to the test function. The wrapping is supported by type information stored together with the query string in the ComputeML document. The EDS Checker permits the use of any kind of parameter types as long as a java implementation exists which can construct such an object based on the query result. It is therefore responsible for parsing query results.
The wrapping is vital for the EDS Checker to choose the correct function as the object oriented concept of overloading functions is permitted. Multiple functions can have the same name as long a s they can be distinguished by their parameter's types.

4.4 Predefined Test Functions

In this chapter, we analyse a defined a test function to show how new tests can be constructed. The example checks if a SYNC object is present and has a given value (Listing 3). First of all it is a boolean computation, as the result is either true or false. The description in the second line is important as it is displayed in the test selection dialog (Figure 3) and after the processing with the results (Figure 4).
The example is a nested test function because we first have to check whether the entry exists, and if it exists, if the value is correct. At first, the existence of this entry is checked (lines 3 to 12) with the boolean computation *Equal*. This test function takes two parameters of equal type *String*. The first parameter is the result of a XPath-query and the second one is a fixed empty string. Therefore, we check if the result of the query is an empty string. In this case, the function evaluates to true and the outer boolean computation (logical OR) also returns true. Otherwise, if the entry exists, it has to be tested for equality with the given value (lines 13 to 24). The same function is used but with a different parameter type. Thus, overloading is permitted with test functions.

4.5 Result processing

During the processing of the tests, all intermediate and final results are stored in a copy of the ComputeML document

```
1 <BooleanComputation Id="SYNC" Method="Or">
2   <Description>Check SYNC Object</Description>
3   <BooleanComputation Id="Check_Existence" Method="Equal">
4     <PredefinedParam Array="no" Type="String" Description="DefaultValue of COP-ID SYNC">
5       <Value Type="XPath">
6         //Parameter/ParameterExt[@Index="1005"]/ancestor::Parameter/DefaultValue/text()
7       </Value>
8     </PredefinedParam>
9     <PredefinedParam Array="no" Type="String">
10      <Value Type="Fixed"/>
11    </PredefinedParam>
12  </BooleanComputation>
13  <BooleanComputation Id="ID_SYNC" Method="Equal">
14    <PredefinedParam Array="no" Type="Integer" Description="DefaultValue of COP-ID SYNC">
15      <Value Type="XPath">
16        //Parameter/ParameterExt[@Index="1005"]/ancestor::Parameter/DefaultValue/text()
17      </Value>
18    </PredefinedParam>
19    <PredefinedParam Array="no" Type="Integer">
20      <Value Type="Fixed">
21        128
22      </Value>
23    </PredefinedParam>
24  </BooleanComputation>
25</BooleanComputation>
```

Listing 3: A sample boolean computation

containing the test definitions. The EDS Checker transforms it to a HTML file which is displayed in a WWW-browser. The transformation is performed with XSLT, an other XML technology which allows translation of XML files according to predefined style files. XSLT also uses XPath to select certain fragments of a document and translate them according to the rules defined in the style sheet file (cf. Figure 4 for the result).

5 Conclusion

In this paper we presented the EDS Checker which automates the test of Electronic Data Sheets. With this software, users can enforce project specific constraints. The tool is built to be extended in terms of different tests, data- and result formats band users are encouraged to customize it to their specific needs. The software can be used in heterogeneous environments, because it is based solely on Java.
We are thinking about making the tool available via the World Wide Web.

6 Literature

[1] CAN in Automation e.V.: *Electronic Data Sheet Specification for CANopen. CiA Draft Standard Proposal 306. Version 1.1*. http://www.can-cia.de/downloads/ciaspecifications/?42

[2] Bühler, D and Gruhler, G: XML-based Representation an Monitoring of CAN Devices. In: Proc. Of the 7th International CAN Conference (ICC 2000), Amsterdam, The Netherlands, October 2000.

Michael Friedrich
University of Tuebingen
Sand 14
72074 Tuebingen, Germany
+49 (7071) 29 - 70475
+49 (7071) 29 - 5160
friedrich@informatik.uni-tuebingen.de
http://www-sr.informatik.uni-tuebingen.de/friedrich/

Prof. Wolfgang Kuechlin
University of Tuebingen
Sand 14
72074 Tuebingen, Germany
+49 (7071) 29 - 77047
+49 (7071) 29 - 5160
kuechlin@informatik.uni-tuebingen.de
http://www-sr.informatik.uni-tuebingen.de/kuechlin/

Dr. Dieter Bühler
University of Tuebingen
Sand 14
72074 Tuebingen, Germany
buehler@informatik.uni-tuebingen.de
http://www-sr.informatik.uni-tuebingen.de/~buehler/