Module documentation

(MOD)

(TINF18C, SWE I Praxisprojekt 2019/2020)

Modul

Converter

Kommentiert [NB1]: nochmal mit Inhalt im Wiki vergleichen

Project: *DD2AML Converter*

Customer: *Rentschler & Ewertz*
Rotebühlplatz 41
70178 Stuttgart

Supplier: by Nora Baitinger and Antonia Wermerskirch - Team 3
(Nora Baitinger, Antonia Wermerskirch, Lara Mack, Bastiane Storz)
Rotebühlplatz 41
70178 Stuttgart

Version	Date	Author	Comment
0.1	07.09.2018		created
0.1	13.04.2020	Nora Baitinger	Filled with information
1.0	08.05.2020	Nora Baitinger	Added Module Tests und Appendix
1.0	12.06.2020	Nora Baitinger	Added Design and Implementation

1. Content

2. Scope

The Module Documentation (MOD) describes the architecture, the interfaces and the main features of the module. It also describes the module/component test including the results. It can also serve as a programming or integration manual for the module. If there are some risks related to the module itself, they shall be noted and commented within this document.

3. Definitions

AML	Automation Markup Language
AML DD	AML Device Description
AMLX	AML Package
CSP+	Control and Communication System Profile
IODD	Input/Output Device Description
PN	Profinet
SAS	System Architecture Specification
SRS	System Requirements Specification

4. Figures

5. Module Requirements

5.1. User View

The converter is a very important part of the system because its duty is the actual conversion.
It is part of the “Library” module which can be used in any case where a conversion from DD-file (GSD, IODD and CSP+) to an AML file is needed.

The Converter must be able to read the DD-file and translate the most important information like the vendor, devices and ports. Its main functionality is to construct the corresponding AML file.

So from the users perspective, the module is supposed to:

1. Take the given DD-file and check the file type
2. Validate the given DD-file
3. Convert the DD-file to AML
4. Return the AML as string or AMLX package

5.2. Requirements

The Converter implements the requirements LF10, LF20 and LF30, as well as a part of LF100.

The LF10 requirement describes the necessity to check if the file is one of the listed formats and recognize it as one of them. So before the conversion to AML can start, the imported file format must be verified.

According to the LF020 requirement the module should handle the file validation. The user wants to convert a DD-file, either in the IODD, CSP+ or GSD format. All the file types shall be validated against their specification to check if a conversion is able to complete.

The LF30 describes the conversion of the specified file. The converter is designed to convert the indicated DD-file to AML. Therefore it works with the specified rules that are different for each of the formats GSD, IODD and CSP+.
Two public methods provide the external systems with the generated information.

The LF100 requires the implementation of the conversion into the two AML/CAEX versions 2.15 and 3.0. The user decides the version, either CAEX V2.15 or CAEX V3.0. So, the library must guarantee the right header for the generated AML file corresponding to the given DD-file.

5.3. Module Context

The Converter is the most important part of the system because it is responsible for the conversion of the given DD-file. It is part of the “Library”, so naturally it works together with the “AML Packager” and the “Library”.
The Converter calls the “Library” to receive the appropriate ruleset for the conversion.
After converting the original DD-file into the corresponding AML file, it is passed to the “AML Packager” that generates the AMLX package.
The trigger to start the conversion is given from either the Command Line Interface or the Graphical User Interface.

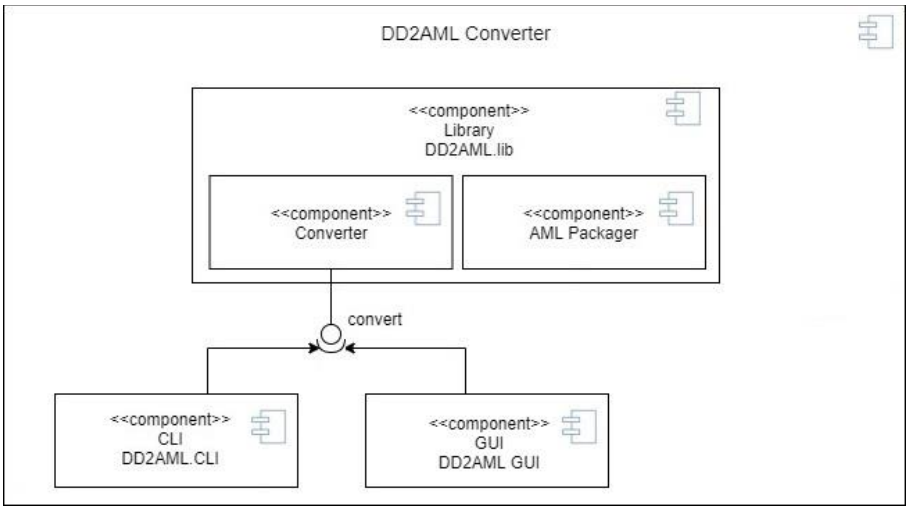

Figure 1 - Converter in the module context

6. Analysis

The first step is the file format analysis where the Converter needs to check if the given file is one of the listed formats.

The next step is the file validation. This raises a few problems: The file does not only be a valid XML file, but confirm the exact specification (IODD or CSP+). Therefore an exact description about the structure of the given formats must be available.

The main task is the correct translation of the CSP+ and IODD files into corresponding AML elements. This is a major challenge, especially when the formats contain different information to translate. So the main issue is to establish fitting rules to accomplish a corresponding AML. However the ruleset is described in the module “Library”.

Additionally, the Converter has the task to choose the correct ruleset regarding the CAEX Version the user wants.

7. Design

This module is designed to convert a given DD-file to a corresponding AML file.
Therefore, it shall have a basic set-up that is composed of five different steps of the conversion process.

In order to start the conversion process either the GUI or the CLI must trigger it. Therefore, they call the “Convert” function of this Converter module.

After triggering the translation the call structure is as follows: Convert → Handle → Translate → TranslateSubProporties → SetAttributes.

All these steps are functions that will be implemented in the converter and util classes.

7.1. Risks

This module is the most important part of the entire system, so handling its risk is extremely important. If the conversion fails, the main purpose of the software is in danger.
So when the conversion is not able to complete, the user needs a very accurate message, so he or she can adjust to the problem.
The rules included in the Library module are the most complex part of the program. The only method to guarantee the functionality of the Converter is to test it very carefully and extensively while programming it.

8. Implementation

To handle the five steps of the conversion described in the chapter above the Converter is composed of two different classes:

- Converter.cs
- Util.cs

The converter class contains the main part of the conversion logic and the util class as a helper class. It is created to maintain clarity and comprehensibility.

In the following, some of the most important functions are described:

The **converter class** contains these functions:

- **Convert()**
  This function manages the process of the conversion and start the actual conversion of the DD-file.
  There are two of these convert functions:
  - The first one is responsible for the conversion to a corresponding AML which contents will be stored in a string.
  - One is responsible for the conversion to an AML package. Therefore it delivers it a list of all the resources that belong the given DD-file to the AML Packager and start the process of packaging the AMLX.

  Both of these convert function must distinguish between the conversion to the AML CAEX Version 2.15 and CAEX Version 3.0.
- **StartConversion()**
  This function starts the file validation if the users activated the “strict mode” and picks the correct translation table for the input file.
- **Handle()**
  The main purpose of the handle function is to start the actual conversion. It iterates over the DD properties and searches for the appropriate translation rule. To convert the property it calls translate function. The conversion of the whole DD-file is a recursive process.
- **Translate()**
  This function translates the specified xml element with the given translation rule to AML syntax.

The second class of the Converter module is the **util class** that contributes to the conversion process with its functions as well:

- **CheckGsdFileForCorrectness()**
  This function is responsible for the file validation. Therefore, it searches the correct xsd file that contains the specification of the DD file formats and compares it with the given DD-file.
- **GetOutputFileName()**
  Its duty is to find the correct file name for the conversion to an AML string. In order to do that it searches the input file name for the name and saves it as AML at the input file’s location.

9. Module Test

The Converter module contains the logic to convert a given DD-file to AML. Tests of this module aren't possible without the ruleset that is part of the Library module. Therefore, no Unit Tests or any other tests are possible for this module.

The system tests that cover the whole functionality of this program contain tests to guarantee the correct function of this module as well. The system test plan provides more information about these tests.

10. Summary

The Converter module is a very important part of the program because it is managing the conversion process. It is also a complex module because the translation is fulfilled in many individual steps. Naturally, the code for this module isn't that easy to understand. However, the individual steps are also a strength of this module. Every function is designed to fulfil only one task and therefore can be used again.

Another advantage is the extensibility of the Library module to other problems related to conversion from XML based files to AML. When the ruleset of the library is extended for example for the conversion of other XML based file types like ESI, SDDML, EDS, PB-GSD or XDD, the converter will provide the logic for the conversion process. The adjustment of the process is thereby minimal.

11. Appendix

11.1. References

[1] SRS: *https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/wiki/System-Requirements-Specification*
[2] STP: *https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/wiki/Systemtestplan*
[3] STR: *https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/wiki/Systemtestreport*

11.2. Code

The source code for this module can be found:

- Converter.cs: *https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/blob/master/SOURCE/src/Dd2Aml.Lib/Converter.cs*
- Util.cs: *https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/blob/master/SOURCE/src/Dd2Aml.Lib/Util.cs*

11.1. System Test Cases

Test-ID	Name	Description	Test Steps		
			Step	Action	Expected Result
TC-001-001	File Validation with valid input file	The test case verifies that it recognizes if a valid file has been selected	1	Install the DD2AML tool and open the CLI by typing cmd in the windows search.	The DD2AML tool is installed on the system. The CLI is open.
			2	Select a valid input file for the validation, for example: dd2aml –input /filePathTo/Balluff-BNI_IOL_355_S02_Z013-20170315-IODD1.1.xml	The validation is executed successfully, and the conversion is completed correctly without error message.
			3	Then open the logs of the CLI. These can be found under: C:\Users\USERNAME\AppData\Local\DD2AML\Logs\CLI	After replacing the USERNAME tag with the real username, the CLI folder with all logs opens. The most recent log is opened.
			4	Find the log message that shows that the file was successfully deserialized. It can be found at the beginning of the log file.	The log message “DD file was deserialized correctly.” should be found approximately in the fourth line of the log.

Test-ID	Name	Description	Test Steps		
			Step	Action	Expected Result
TC-001-002	File Validation with invalid input file	The test case verifies that errors are detected during the validation of the input file and a corresponding error message is displayed with a description of the error and line details in the log.	1	Install the DD2AML tool and open the CLI by typing cmd in the windows search.	The DD2AML tool is installed on the system. The CLI is open.
			2	Select a valid input file for the validation, for example: dd2aml –input /filePathTo/BrokenBalluff-BNI_IOL_355_S02_Z013-20170315-IODD1.1.xml	The conversion is aborted after the failed validation.
			3	Then open the logs of the CLI. These can be found under: C:\Users\USERNAME\AppData\Local\DD2AML\Logs\CLI	After replacing the USERNAME tag with the real username, the CLI folder with all logs opens. The most recent log is opened.
			4	Look at the first error message in the logs.	The error message can be found approximately in the 5th line. Detailed information about the error, as well as line details are given.