# Modul documentation

(MOD)

(TINF18C, SWE I Praxisprojekt 2019/2020)

Sub-Modul

# AML Packager

*Project:* *DD2AML Converter*

*Customer:* *Rentschler & Ewertz*
Rotebühlplatz 41
70178 Stuttgart

*Supplier:* by Nora Baitinger and information from TINF17C/GSD2AML-Converter Team
- 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 Design and Implementation |
| 1.0 | 10.05.2020 | Nora Baitinger | Added Module Tests<br>Final Version |

# 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 Author of this document

This document is based on the tool GSD2AML-Converter developed by the TINF17C course. Some of the sections in this document are identical to their module documentation about the "Compressor" module which this developer group named "AML Packager". These sections will be tagged with **TINF17C/GSD2AML-Converter Team*. All other sections are written and developed by Nora Baitinger, Antonia Wermerskirch, Lara Mack, Bastiane Storz.

## 4 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 |
| **Temp/TMP** | Temporary, temporary file directory |

## 5 Figures

# 6 Module Requirements

## 6.1 User View

The task of the AML Packager is to create the AMLX package with all the needed dependencies. These are the original DD-file and the specified resource files. At last the resulting package shall be saved at the given output path.

From the users point of view the AML Packager should complete the following tasks:

1. Accept the converted AML file
2. Take the specified resource files that contain the original DD-file
3. Build the AML and all the referenced files into the AMLX package
4. Save the generated package to the given output path

## 6.2 Requirements

The functional requirements mentioned in the SRS LF40 and part of LF80 are implemented by the AML Packager module.

According to the LF40 three different exercises shall be accomplished: Collect all the necessary resources like the original DD-file, create the required AMLX package and store it at the given output directory. The default directory for the generated package is the same as those of the input file.

The resulting AMLX package should contain all the dependencies of the AML file:

1. All graphic files which were used by the original DD-file
2. All linked files
3. The original file itself
4. The generated AML file corresponding to the original DD-file

Furthermore the module should fulfil the requirements specified in LF80. The user or any other developer working on the project should receive meaningful exception and error messages that help to identify an eventually occurring problem.

## 6.3 Module Context

The AML Packager Module is called from the “Converter” module when it has successfully finished converting the given DD-file into the AML specification. The “Converter” also provides the necessary information regarding to all resources needed for the AMLX package and the desired output path.
An existing library can be used to fulfil the task of generating an AMLX package. It is part of the class “AML.Engine.AmlObjects.AutomationMLContainer” and shown as a component in the diagram as well.
After building the required AMLX package the AML Packager returns the output path to the calling module.

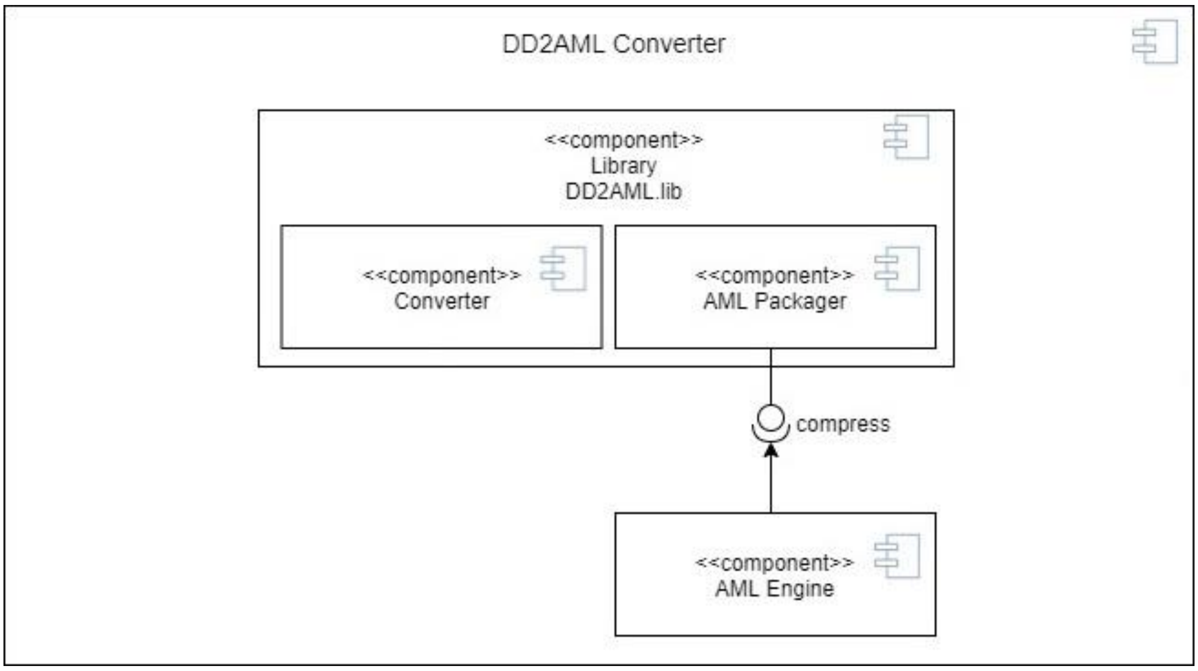


*Figure 1 - AML-Packager in the module context*

# 7 Analysis

The main goal of this module is to create the AMLX package. This process contains packaging all required dependencies regarding to the generated AML file and the AML file itself. The dependencies that refer to the given DD-file, are for example images that need to be considered and included in the package.

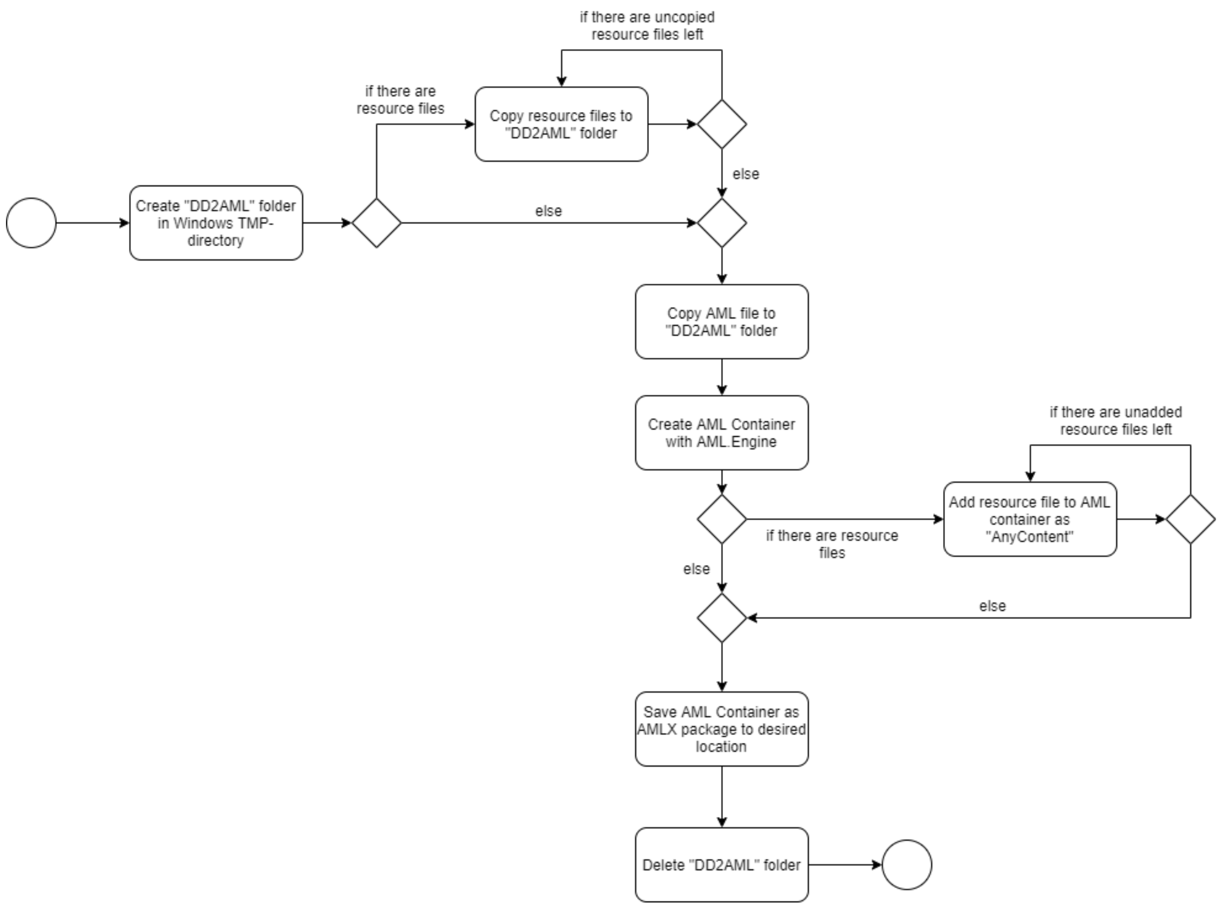
Therefore, the module needs some information: The desired output directory, all the dependencies and the AML file.

There are some problems that need to be considered for the implementation: An AMLX package with the same name could already exits at the given output path. In this case there shall be the option to overwrite the existing package.
The module assumes that the AML file is correct and the list of all given dependencies is complete. However, it needs to find all of the listed files and transfer them to a central place to generate the package. For this purpose the Windows Temp directory is suitable.

To complete the packaging, an existing library can be used. It is part of the AML.Engine NUGet package and can be found in the class “AML.Engine.AmlObjects.AutomationMLContainer.”

# 8 Design

The following activity diagram shows the process that shall be stared when calling the "compress" method. This method will be the main part of the module. It can be implemented in a single class called AMLPackager.cs.



if there are uncopied resource files left
if there are resource files
Copy resource files to "DD2AML" folder
else
Create "DD2AML" folder in Windows TMP-directory
else
Copy AML file to "DD2AML" folder
Create AML Container with AML.Engine
if there are unadded resource files left
Add resource file to AML container as "AnyContent"
if there are resource files
else
else
Save AML Container as AMLX package to desired location
Delete "DD2AML" folder

*Figure 2 - Activity diagram AML Packager**

**Algorithm designed by TINF17C/GSD2AML-Converter Team*

## 8.1 Risks

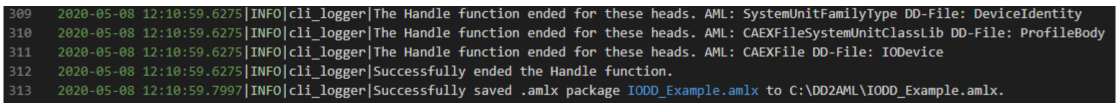
The AML Packager module interacts with the Converter module and while it assumes that all the given information is correct that could be wrong. This would lead to an incomplete AMLX package or even to an incorrect AML file.
These problems can be minimised by performing tests on all the modules, described in each module documentation and the System Test Plan and the System Test Report.
One of the biggest problems that could occur working with this module regards the operating system. It is possible that the right to access the file system and special directories are missing. This risk can be minimised by starting the program with administrator privileges. When users try to start the program without sufficient rights, the user receives a suitable error message.

# 9 Implementation

All the functionality described in the module documentation "Compressor" from the original developers are still describing the implementation accurately, because there was no addition to make.

The process described in chapter 8 can be implemented in the class AMLPackager.cs with different methods. Their functionality will be explained in the following:

1. **"Zip":**
   A function taking a directory path as input and packaging the content to an .amlx package. It also gets a destination path as input where the resulting zip file is saved. It catches any exceptions and uses the logger module to inform the user using logging level "error".
2. **"CreateTmpDirectory":**
   A function that takes in a name as input and creates an equally named folder in the Windows-Temp directory and returns the path to it. It catches any exceptions and uses the logger module to inform the user using log-ging level "error".
3. **"CopyFile":**
   A wrapper function taking a source and a destination path. The function will copy the source file to the destination. It catches any exceptions and uses the logger module to inform the user using logging level "error".
4. **"DeleteFolder":**
   Given a path deletes the folder specified by that path. The function catches any exceptions and uses the logger module to inform the user using logging level "error".
5. **"Compress":**
   A function wrapping functions 1-4 to an end-to-end process. This function takes as input the path to the AML file, a list of resource paths and the destination. From that it uses functions 1-4 to do the following:
   a. Creates a folder called *"DD2AML"* in the Windows-TMP directory.
   b. Copies all resources to that folder.
   c. Copies the AML file to that folder.
   d. Uses the AML.Engine library to package the content of the *"DD2AML"* folder. The input destination path includes information about how the package shall be named and where it will be stored. The function catches occurring exceptions and uses the logger module on level "error" to inform the user. On success the function logs a success message on level "info".



```
2020-05-08 12:10:59.6275|INFO|cli_logger|The Handle function ended for these heads. AML: SystemUnitFamilyType DD-File: DeviceIdentity
2020-05-08 12:10:59.6275|INFO|cli_logger|The Handle function ended for these heads. AML: CAEXFileSystemUnitClassLib DD-File: ProfileBody
2020-05-08 12:10:59.6275|INFO|cli_logger|The Handle function ended for these heads. AML: CAEXFile DD-File: IODevice
2020-05-08 12:10:59.6275|INFO|cli_logger|Successfully ended the Handle function.
2020-05-08 12:10:59.7997|INFO|cli_logger|Successfully saved .amlx package IODD_Example.amlx to C:\DD2AML\IODD_Example.amlx.
```

*Figure 3 - Success messages in log file – current version of the DD2AML Converter*

6. **"CreateDirectory":**
   A helper function for creating directories that might or might not already exist. This function is used by "CreateTmpDirectory".

**Main tasks of the function described by TINF17C/GSD2AML-Converter Team*

All log files mentioned above can be found in "C:/Users/[your user]/AppData/Local/DD2AML/Logs/[GUI | CLI]/".

The .Net IO functionality is used to interact with the file system and handle the different paths.
All the functions contained in the AMLPackager class are “private” except the “compress” method that is used as an interface. All the other modules and programs that want to use the library can start the packaging process.
Additionally, the whole code is documented using C#XML documentation comments. These comments can be read by machines and therefore used to provide useful information to the developers of the stand-alone library.

# 10 Module Test

The AML Packager module can be tested with four different Unit Tests with the White-Box view. All the source code regarding these four tests are available: *https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/blob/master/SOURCE/src/Dd2Aml.Test/AmlPackagerTest.cs*

## 10.1 Component Testplan

| Test-ID | Feature ID | Test Specification (Description or TCS) |
|---|---|---|
| 1 | LF40 | This test simulates a complete compression process and validates the results.<br>To achieve this kind of test, the AML Packager needs some information:<br>1. Test AML file<br>2. Resource file<br>3. valid destination path<br>The AML Packager should compress these files to an AMLX package and stores it to the given destination path.<br>The test will be successful, when the package is stored to the correct destination path and if it is complete. |
| 2 | LF40 | Tests if the AML Packager fails when users forget to enter a name for the AML file that should be packed. |
| 3 | LF40 | Tests if the AML Packager fails when the destination path is empty. |
| 4 | LF40 | Tests if the AML Packager works with an empty resource array. |

## 10.2 Component Testreport

| Test-ID | Pass/Fail | If failed: Test Observation | Date | Tester |
|---|---|---|---|---|
| 1 | Pass | | 10.05.2020 | N.Baitinger |
| 2 | Pass | | 10.05.2020 | N.Baitinger |
| 3 | Pass | | 10.05.2020 | N.Baitinger |
| 4 | Pass | | 10.05.2020 | N.Baitinger |

# 11 Summary

This module implements all the necessary requirements and it is sufficiently tested. Relevant information regarding the packaging process is logged to inform the user.

The current version of this module stores all the resource files of the given DD-file into the top-level directory of the AMLX directory. To improve this file structure future versions of this AML Packager could distinguish between different types of resources like pictures and the original DD-file for example. All the different types of resources could be stored into different sub-directories. However this feature would not improve the functionality of this module, so the priorities lied elsewhere.

# 12 Appendix

## 12.1 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*

## 12.2 Code

The functionality described above can be found in the library path of the project:
*https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/blob/master/SOURCE/src/Dd2Aml.Lib/AMLPackager.cs*

## 12.3 Module Test Cases

| Test-ID | Name | Description | Test Steps | | |
|---|---|---|---|---|---|
| | | | Step | Action | Expected Result |
| 1 | TestAML-Packager | Complete packaging process | 1 | Assume a test AML and a test resource file to be given. | |
| | | | 2 | Call function with test data and an appropriate test directory path “tst/myAMLX.xml”. | myAMLX.xml is created correctly |
| | | | 3 | Unzip the myAMLX.xml file and compare the content with the test files | The archives content is equal to the test files. |
| | | | | | |
| 2 | FailOnOmittedAMLName | Checks if an empty AML name string is passed correctly. | 1 | Assume a test AML and a test resource file to be given. | |
| | | | 2 | Call compress but with empty AML string instead of the file name. | Compress function raises an error. |
| | | | | | |
| 3 | FailOnOmittedDestina-tion-Path | Checks if an empty destination path is handled correctly. | 1 | Assume a test AML and a test resource file to be given. | |
| | | | 2 | Call compress but with an empty destination string. | Compress function raises an error. |
| | | | | | |
| 4 | TestEmpty-Resources | Checks if an empty resource array is handled correctly. | 1 | Assume a test AML and a test resource file to be given. | |
| | | | 2 | Call compress but with an empty resource array string. | Compress function successfully builds the package. |
| | | | | | |