**System Test Plan**

(TINF18C, SWE I, II Praxisprojekt 2019/2020)

Project: AMLEngine-DLL Interfaces

Customer: Rentschler & Ewertz

Rotebühlplatz 41

70178 Stuttgart

Supplier: Team 4 Joshua, Kevin, Krister, Lucas, Markus, Robin

Rotebühlplatz 41

70178 Stuttgart

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comment** |
| 0.1 | 13.04.2020 | Kevin | Document created |
| 0.2 | 19.04.2020 | Kevin | Enhancement of the test cases |
| 0.3 | 29.04.2020 | Kevin | Bug fixing |
| 0.4 | 1.05.2020 | Kevin | Updated test data |
| 1.0 | 5.05.2020 | Kevin | Completion |
| 1.1 | 10.05.2020 | Kevin | Last changes after review |

# Table of Contents

[Table of Contents 2](#_Toc40022604)

[1. Scope 3](#_Toc40022605)

[2. Definitions 3](#_Toc40022606)

[3. Product names and attributes 3](#_Toc40022607)

[4. Features 4](#_Toc40022608)

[5. Test preparation strategy 4](#_Toc40022609)

[6. Test execution strategy 5](#_Toc40022610)

[7. Testing setup 5](#_Toc40022611)

[7.1. Console Application 5](#_Toc40022612)

[7.2. C++ Wrapper 5](#_Toc40022613)

[7.3. JS Wrapper 5](#_Toc40022614)

[8. Test schedule and budget 5](#_Toc40022615)

[9. Test planning 6](#_Toc40022616)

[10. References/Standards 6](#_Toc40022617)

[11. Appendix: Test Cases 7](#_Toc40022618)

[11.1. Testsuite <TS-001 C++ Wrapper> 7](#_Toc40022619)

[11.1.1. <TC-001-001> (Follow Wrapper instructions) 7](#_Toc40022620)

[11.2. Testsuite <TS-002 JS Wrapper> 8](#_Toc40022621)

[11.2.1. <TC-002-001> Wrapper import 8](#_Toc40022622)

[11.2.2. <TC-002-002> Valid call and invalid call handling 9](#_Toc40022623)

[11.2.3. <TC-002-003> Supported functions 11](#_Toc40022624)

[11.3. Testsuite <TS-003 Console Application> 13](#_Toc40022625)

[11.3.1. <TC-003-001> UI-Test 13](#_Toc40022626)

[11.3.2. <TC-003-002> Validation Test 15](#_Toc40022627)

[11.3.4. <TC-003-003> (De-)Compression Test 16](#_Toc40022628)

[11.3.5. <TC-003-004> Import Test 16](#_Toc40022629)

# Scope

The STP (System Test Plan) specifies the test strategy and test planning. It references tests to be performed to verify the accordance of the demanded features given by the SRS [[1]](#_References/Standards) (System Requirements Specification) to the implemented features. The document derived from the STP is the STR (System Test Report) [[2]](#_References/Standards) where additionally the results are given.

# Definitions

|  |  |
| --- | --- |
| AML | AutomationML |
| AMLX | AutomationMLContainer |
| CLI | Command User Interface |
| DLL | Dynamic Linked Library |
| TC | Testcase |

# Product names and attributes

The following test objects must be verified:

|  |  |  |  |
| --- | --- | --- | --- |
| Ref.-Id. | Product Number | Product Name | Product Description |
| 1 | Version 1.0 | C++ Wrapper | Compiler settings to allow the usage of the AMLEngine.dll in a C++ project |
| 2 | Build 1.0 | AML Console Application | CLI for validating and (de-)compressing of AML Files |
| 3 | 1.0.5 | Javascript Wrapper | Wrapper for AmlEngine.dll in Node JS |

# Features

|  |  |  |  |
| --- | --- | --- | --- |
| Req. - ID | Functionality | Priority | Testsuite ID |
| LF10 | C++ Functions | C | TS-001 |
| LF20 | C++ usability | C | TS-001 |
| LF30 | Javascript Functions | A | TS-002 |
| LF40 | Javascript Usability | A | TS-002 |
| LF50 | Import | A | TS-003 |
| LF60 | (De-)Compression | B | TS-004 |
| LF70 | Validation | B | TS-005 |

# Test preparation strategy

The test schematics is based on the project structure. A test suite for each individual use-case is created. They each contain several tests. In this project there are main test suits for the JS Wrapper, C++ Wrapper and the Console Application.

The JS Wrapper has to provide the AML functions. Each function must validate the user input. Also, there should be checked whether all data needed is delivered.

The C++ Wrapper must enable the developer to use the Amlengine.dll in his project. It has to be tested for full functionality and usability.

# Test execution strategy

Since this is a test for a new software product, a full test is required. This test should be divided in the following parts:

* 1. Javascript Wrapper
  2. Console Application
  3. C++ Wrapper

Since the Javascript Wrapper is the most complex part, it should be tested as early as possible.

After that the Console Application shall be tested, as it is the next product in regard of complexity.

In the last step, the C++ Wrapper shall be tested.

# Testing setup

## Console Application

The only test equipment required is a fully functional version of the product.

## C++ Wrapper

The only test equipment required is a fully functional version of the product.

## JS Wrapper

To test this module the following software is required:

* Node.js version 12.14.1 or higher
* NPM version 6.13.7 or higher

All test cases should be tested by writing commands to a file and executing them using the command “*node <filename>.js*”.

# Test schedule and budget

|  |  |  |
| --- | --- | --- |
| Testsuite | Begin | End |
| C++ Wrapper | 13.04.2020 | 01.05.2020 |
| JS Wrapper | 13.04.2020 | 01.05.2020 |
| Console Application | 13.04.2020 | 01.05.2020 |

# Test planning

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Testsuite | Test objective | Test plan creator | Test plan reviewer | Tester |
| TS-001 | C++ Wrapper | Lucas Krauter | Krister Wolfhard | Kevin Kretschmar |
| TS-002 | JS Wrapper | Markus Limbacher | Krister Wolfhard | Joshua Franz |
| TS-003 | Console Application | Joshua Franz | Krister Wolfhard | Markus Limbacher |

# References/Standards

[1] SRS - <https://github.com/RBeerDevelopment/TINF18C_Team_4_AMLEngine-DLL-Interface/wiki/System-Requirements>

[2] STR - <https://github.com/RBeerDevelopment/TINF18C_Team_4_AMLEngine-DLL-Interface/wiki/System-Test-Report>

# Appendix: Test Cases

## Testsuite <TS-001 C++ Wrapper>

### <TC-001-001> (Follow Wrapper instructions)

|  |  |  |
| --- | --- | --- |
| Testcase ID: | TC-001-001 | |
| Testcase Name: | C++ Wrapper | |
| Req.-ID: | UC.001, /LF10/C++ Functions, /LF20/C++ usability | |
| Description: | This test case verifies that the C++ wrapper instructions are understandable and lead to a correct executable, which uses the AMLEngine.dll. | |
| Test Steps | | |
| Step | Action | Expected Result |
| 1 | Install Visual Studio Community or Enterprise Microsoft. | The editor can be started. |
| 2 | Download the wrapper instructions from the official GitHub repository of this project or open it in the browser | The wrapper documentation can be opened to read. |
| 3 | Follow the wrapper instructions | A code example using the AMLEngine.dll is ready to compile. |
| 4 | Compile the code with help of the wrapper instructions. | An executable is built by the compiler tools. |
| 5 | Run the executable. | The executable runs. |

## Testsuite <TS-002 JS Wrapper>

### <TC-002-001> Wrapper import

|  |  |  |
| --- | --- | --- |
| Testcase ID: | TC-002-001 | |
| Testcase Name: | Wrapper import | |
| Req.-ID: | UC.002, LF30, LF40 | |
| Description: | Validates that the wrapper package can be downloaded and imported into a node project. | |
| Test Steps | | |
| Step | Action | Expected Result |
| 1 | Make sure to have all the required software installed. (See 7.3) | n/a |
| 2 | Run the command “*npm i amlenginewrapper --save*” | Success message is displayed and the package is added to the package.json file |
| 3 | Invoke the interactive node terminal using the command “*node*” | The console displays the message “Welcome to Node.js” |
| 4 | Write the command “*wrapper=require('amlenginewrapper');*” | The package is imported without any error messages |

### <TC-002-002> Valid call and invalid call handling

|  |  |  |
| --- | --- | --- |
| Testcase ID: | TC-002-002 | |
| Testcase Name: | Valid call and invalid call handling | |
| Req.-ID: | UC.002, LF30, LF40 | |
| Description: | Validates that the wrapper package can access all supported functions inside the Adapter. Validates that the wrapper package can also handle invalid function calls. | |
| Test Steps | | |
| Step | Action | Expected Result |
| 1 | Make sure to have all the required software installed. (See 7.3) | n/a |
| 2 | Run the command “*npm i amlenginewrapper --save*” | Success message is displayed and the package is added to the package.json file |
| 3 | Invoke the interactive node terminal using the command “*node*” | The console displays the message “Welcome to Node.js” |
| 4 | Write the command “*wrapper=require('amlenginewrapper');*” | The package is imported without any error messages |
| 5 | Enter the test data specified below. | The output matches the expected result. |

|  |  |  |  |
| --- | --- | --- | --- |
| Test data: | Inputs for call method | | |
|  | function | parameter(s) | Expected result |
| 1 | Call - instancehierarchy\_append | wrapper.call(“instancehierarchy\_append”, “pathToAMLFile”, {“indexer”: “indexOfHierarchyToAppendTo”, “internalelement”:”nameOfInternalElementToAppend”}) | No errors thrown. New element is appended in the given aml file. |
| 2 | Call - instancehierarchy\_get | wrapper.call(“instancehierarchy\_get”, “pathToAMLFile”, { “indexer”: “indexOfInstanceHierarchyToGet”}) | No exceptions thrown. Instance hierarchy is printed to the console |
| 3 | Call - create\_systemunitclass | wrapper.call(“create\_systemunitclass”, “pathToAMLFile”, {“unitclasslib\_name”: “nameOfNewUnitClassLib”, “unitclass\_name”: “nameOfNewUnitClass”, “indexer”: “indexOfInstanceHierarchy }) | No errors thrown. New system unit class is stored in the given aml file. |
| 4 | Call - create\_interfaceclass | wrapper.call(“create\_interfaceclass”, “pathToAMLFile”,  {“interface\_classname”: “nameOfNewInterfaceClass”,  “iface\_name”: “nameOfNewInterface”}) | No errors thrown. New interface class is stored in the given aml file. |
| 5 | Call - instanceelement\_append | wrapper.call(“instanceelement\_append”, “pathToAMLFile”, {“indexer”: “indexOfInstanceHierarchy”, “inElement”: “nameOfNewElement” | No errors. New elements are stored in the given aml file. |
| 6 | Call - rename\_element | wrapper.call(“rename\_element”, “pathToAMLFile”, { “indexer”: “indexOfInstanceElementToChange”, “newName”: “theNewNameToUse”,  “ie”:”nameOfInternalElementToRename”}) | No errors. New data is stored in the element in the given aml file. |
| 7 | Call - validate | wrapper.call(“validate”, “pathToAMLFile”, {}) | The console prints problems in the aml file syntax. |
| 8 | Call - repair | wrapper.call(“repair”, “pathToAMLFile”, {}) | All syntax-based problems get repaired in the aml file given. |

### <TC-002-003> Supported functions

|  |  |  |
| --- | --- | --- |
| Testcase ID: | TC-002-003 | |
| Testcase Name: | Supported functions | |
| Req.-ID: | UC.002, LF30, LF40 | |
| Description: | Validates that all supported functions can be called using their quick access option | |
| Test Steps | | |
| Step | Action | Expected Result |
| 1 | Make sure to have all the required software installed. (See 7.3) | n/a |
| 2 | Run the command “*npm install AMLEngineDLLWrapper --save*” | Success message is displayed and the package is added to the package.json file |
| 3 | Invoke the interactive node terminal using the command “*node*” | The console displays the message “Welcome to Node.js” |
| 4 | Write the command “*wrapper=require('amlenginewrapper');*” | The package is imported without any error messages |
| 5 | Enter the test data specified below using the following syntax:  *wrapper.<function>(<parameters>);* | The output matches the expected result. |

|  |  |  |  |
| --- | --- | --- | --- |
| Test data: | Supported functions | | |
|  | function | parameter(s) | Expected result |
| 1 | appendToInstanceHierarchy | wrapper.appendToInstanceHierarchy(“pathToAMLFile”, “nameOfAppend”, “internalElement”) | No errors thrown. New element is appended in the given aml file. |
| 2 | getInstanceHierarchy | wrapper.getInstanceHierarchy(“pathToAMLFile”, “indexOfInstanceHierarchyToGet”) | No exceptions thrown. Instance hierarchy is printed to the console |
| 3 | createSystemUnitClass | wrapper.createSystemUnitClass( “pathToAMLFile”, “nameOfNewUnitClassLib”, “nameOfNewUnitClass”, “indexOfInstanceHierarchy ) | No errors thrown. New system unit class is stored in the given aml file. |
| 4 | createInterfaceClass | wrapper.createInterfaceClass( “pathToAMLFile”, “nameOfNewInterfaceClass”,  “nameOfNewInterface”) | No errors thrown. New interface class is stored in the given aml file. |
| 5 | appendInstanceElement | wrapper.appendInstanceElement(“pathToAMLFile”, “indexOfInstanceHierarchy”, “nameOfNewElement”) | No errors. New elements are stored in the given aml file. |
| 6 | renameElement | wrapper.renameElement ( “pathToAMLFile”, “indexOfInstanceElementToChange”, “theNewNameToUse”,  ”nameOfInternalElementToRename”) | No errors. New data is stored in the element in the given aml file. |
| 7 | validate | wrapper.validate( “pathToAMLFile”) | The console prints problems in the aml file syntax. |
| 8 | repair | wrapper.repair( “pathToAMLFile”) | All syntax-based problems get repaired in the aml file given. |

## Testsuite <TS-003 Console Application>

### <TC-003-001> UI-Test

|  |  |  |
| --- | --- | --- |
| Testcase ID: | TC-003-001 | |
| Testcase Name: | UI-Test | |
| Req.-ID: | * (Usability) | |
| Description: | This Test Case verifies the Usability of the Console-Application | |
| Test Steps | | |
| Step | Action | Expected Result |
| 1 | Download and start the Console Application in Windows 10 | A Console Application should start and the Main Menu should appear |
| 2 | Check if the Main Menu is written correctly and that the UI is user friendly and understandable. | It should be user friendly, understandable and correct. |
| 3 | Check if the Options Menu is written correctly and that the UI is user friendly and understandable. | It should be user friendly, understandable and correct. |
| 4 | Check if the Validation Menu is written correctly and that the UI is user friendly and understandable. For this a File from Example Files should be verified. | It should be user friendly, understandable and correct. |
| 5 | Check if the DeCompress Menu is written correctly and that the UI is user friendly and understandable. For this a AMLX File from Example Files should be used. | It should be user friendly, understandable and correct. |
| 6 | Check if the Compress Menu is written correctly and that the UI is user friendly and understandable. For this the Files from Step 5 should be Compressed. | It should be user friendly, understandable and correct. |
| 7 | Repeat Steps 1-6 in Windows 7 if possible. | It should work the same as with Windows 10 |

### 

### 

### <TC-003-002> Validation Test

|  |  |  |
| --- | --- | --- |
| Testcase ID: | TC-003-002 | |
| Testcase Name: | Validation Test | |
| Req.-ID: | LF70 | |
| Description: | This Test verifies the Validation Functionality of the Console Application | |
| Test Steps | | |
| Step | Action | Expected Result |
| 1 | From the start menu go to the “options” interface. There select the following parameters:   1. “AutoRepair” 2. “PrintAllVal”   For best test coverage, try steps 2 and 3 with all 4 combinations. This means use the following settings:   * 1 – false, 2 – false * 1 – true, 2 – false * 1 – false, 2 – true * 1 – true, 2 - true | The output from step 2 and 3 should be adjusted accordingly to the settings made in the options menu. |
| 2 | Validate a correct file using the validation menu. | It should validate correctly. |
| 3 | Validate an incorrect file using the validation menu. Try both options of repair error and override the old file. (Depending on the set option from the menu) | It should show an error and the option to repair it. (Depending on the set options from the menu) |

### 

### <TC-003-003> (De-)Compression Test

|  |  |  |
| --- | --- | --- |
| Testcase ID: | TC-003-003 | |
| Testcase Name: | (De-)Compression Test | |
| Req.-ID: | LF60 | |
| Description: | This Test verifies the Compress and Decompress Functionality of the Console Application. | |
| Test Steps | | |
| Step | Action | Expected Result |
| 1 | Go to the decompress menu and decompress an AMLX file (you can use the file in example files, but you don't have to). | The file should be decompressed successfully. |
| 2 | Go to the compress menu and compress one file. (You can use the files from the decompression if you want to, but you don't have to) | The file should be compressed successfully. |
| 3 | Go to the compress menu and compress two or more files, but don't use a ClassModel. (You can use the files from the decompression if you want to, but you don't have to) | The files should be compressed successfully. |
| 4 | Go to the compress menu and compress two or more files and use a ClassModel. (You can use the Files from the decompression if you want to, but you don't have to) | The files should be compressed successfully. |

### <TC-003-004> Import Test

|  |  |  |
| --- | --- | --- |
| Testcase ID: | TC-003-004 | |
| Testcase Name: | Import Test | |
| Req.-ID: | LF50 | |
| Description: | This test verifies the import functionality per starting parameter of the Console Application. | |
| Test Steps | | |
| Step | Action | Expected Result |
| 1 | The Console Application has the following Parameters:   * path and a valid path after that * validate -> declares that the file should be validated * compress -> declares that the file should be compressed   This can look like this:  “ConsoleApplication.exe --path C:\File.aml -- validate”  Test the functionality with a valid path and every combination (no parameters, all parameters etc.) | The Console Application should give correct errors and work correctly. |
| 2 | Test the functionality with an invalid path and every combination of parameters | The Console Application should give correct errors. |