Eindhoven
University of Technology

C++ JetBrains MPS Extension

version 1.0

Software Transfer Document

Project team:
Nicholas Donnelly
Daan Drijver
Bart van Helvert
Julia Hofs
Daan van der Kallen
Bart Munter
Joris Rombouts
Job Savelsberg
Bart Smit
Remco Surtel
Jelle van der Ster

Customer:
Dmitrii Nikeshkin

Supervisor:
Önder Babur

Project Managers:
Wout de Ruiter
Wesley Brants

July 4, 2018

Abstract

This Software Transfer Document (STD) describes the procedure to set up the C++ extension for the Jetbrains Meta Programming System (MPS). With the C++ extensions to MPS, users will be able to design C++ programs in MPS while making use of advanced editing features that MPS makes possible. This document complies with the ESA software standard.[1]

Contents

Document Status Sheet

General

Document title:	Software Transfer Document
Identification:	STD/ 1.0
Authors:	N.J. Donnelly, B.A.J.v.Helvert, R.J.A. Surtel, J. Savelsberg, B.I. Munter
Document status:	final version

Document Change Records

Section	Applicable Changes
0.0	Final version

1 Introduction

1.1 Purpose

The Software Transfer Document (STD) describes the required steps for installing the C++ extension for MPS. The document describes the environment in which the software is built. Furthermore, a list of user requirements, described in the URD,[2] that are not met is given, as well as a list of tests that passed and tests that failed. Finally, a list of software problems is given, as well as information about software change requests that came up during the transfer phase.

1.2 Scope

Eded is a bachelor end project group working for the TU/e and Océ. The software end product is an extension to mbeddr which implements the C++ language in MPS. The main purpose of the project is to develop an extension to JetBrains MPS that allows users to fluidly develop applications and domain specific languages that compile directly to C++, regardless of the level of abstraction that will be used. This extension will allow users to create domain specific languages with C++ code, add C++ code to imported models and finally generate all to C++ code that follows the MISRA C++ guidelines.[3] It should also allow the user to use MPS as an integrated development environment (IDE) for C++ programming. The extension will therefore support two primary types of features: abstract syntax tree editing and code generation. The MPS C++ extension created by Eded will have support for header files, code editing facilities, code generation, class implementation, and namespace implementation. The extension will be built on top of an already existing MPS extension, mbeddr.[4]

In the industry there is a large demand for C++ support – embedded software, for example, often uses C++. With our extension, users will be able to extend their existing models with C++ code to make C++ programming in MPS possible.

1.3 List of definitions and abbreviations

1.3.1 List of definitions

Namespace	A scoping mechanism in the C++ environment to deal with conflicting names in large C++ projects. Every class included in a namespace is accessible from outside the namespace.
Standard Library	The C++14 standard library definitions.[5]
Domain Specific Languages	Programming languages that raise the level of abstraction beyond programming by specifying solutions that directly use concepts and rules from a specific problem domain.

1.3.2 Abbreviations

ADD	Architectural Design Document
DSL	Domain Specific Language
ESA	European Space Agency
IDE	Integrated Development Environment
MPS	Jetbrains Metaprogramming System
SRD	Software Requirements Document
URD	User Requirements Document

1.4 List of references

[1] ESA, 1991, *software engineering standards*, ESA PSS-05-0 issue 2

[2] Eded, (2018), *User Requirements Document*, version 1.0.1

[3] MISRA C++, 2008, *Guidelines for the use of the C++ language in critical systems*, Nuneaton, UK: MIRA Limited.

[4] mbeddr, 2018, *The mbeddr platform*, Web. http://mbeddr.com/platform.html. Accessed 01 May 2018.

[5] cpp-reference, 2018, *C++ Standard Library headers*, Web. http://en.cppreference.com/w/cpp/header. Accessed 03 May 2018.

[6] Eded, (2018), *Software Requirements Document*, version 1.0

[7] Eded, (2018), *Architecture Design Document*, version 1.0

[8] Eded, (2018), *Software User Manual*, version 1.0

[9] Eded, (2018), *Unit Test Plan*, version 1.0

[10] Eded, (2018), *Acceptance Test Plan*, version 1.0

1.5 Overview

The remainder of this document is composed into eight sections. Section 2, which describes the procedure when building the C++ extension. Section 3 describes the installation procedure of the software. Section 4 gives an overview of the deliverables by Eded. Section 5 provides a summary of the acceptance test. Section 6 details all problems present in the software as of the transfer phase. Section 7 describes any changes that were requested during the transfer phase. Section 8 describes the modifications made during the transfer phase. Section 9 is an appendix that describes the current progress towards copy-pasting plain-text C++ code.

2 Build procedure

This section details how to build to the Jetbrains cpp extension. The build process is platform dependent. The extension can be built on Windows, Mac and Linux. We will describe the build process for each operating system in the upcoming sections.

2.1 Windows

On Windows, there are several prerequisites to installation. It is advised to use minigw* to install the following tools:

1. mingw32-gcc (v 4.8.1)
2. mingw32-make (v 3.82.90-2)
3. mingw32-gdb (v 7.6.1)
4. msys-base (v 2013072300)
5. msys-coreutils (v 5.97-3)

After installing the tools listed above, Apache Ant† and GraphViz‡http://graphviz.org/ need to be installed. After all prerequisites are installed, the code can be built. This is done by executing the following steps:

1. Clone the repository: "https://github.com/Bartvhelvert/mbeddr.core" using git
2. Execute the command:

   ```
   gradlew.bat build_mbeddr
   ```

   in the root directory of the project. The build process will now start, this will take about 30-70 minutes to complete depending on the speed of your system.
3. Copy content of the folder

   ```
   ../mbeddr.core/MPS/MPS-mbeddr-2018.1/bin/win
   ```

*http://www.mingw.org/
†https://ant.apache.org/
‡footnote

to

`../mbeddr.core/MPS/MPS-mbeddr-2018.1/bin`

4. Start MPS by running

   `../mbeddr.core/MPS/MPS-mbeddr-2018.1/bin/mps.bat`

5. Go to file → settings in MPS (or configure → settings if applicable)
6. Go to Appearance & Behavior → Path variable
7. Add a Path variable with the name "mbeddr.github.core.home" which points to your mbeddr core home directory.
8. Open the ../mbeddr.core/code/languages/com.mbeddr.core language (do not migrate)
9. Open the ../mbeddr.core/code/languages/com.mbeddr.cpp language in another window (do not migrate)
10. Right click the cpp project in the logical view and click on "Rebuild with dependencies".

It is also possible to open the project in a clean, separately installed version of MPS instead of the MPS version which is shipped with mbeddr. To make this work some plugins need to be installed to the clean MPS installation. This can be done by running

`gradlew.bat install "\-PMPS\_Installation=<path to MPS>"`

in the root project directory, where `<path to MPS>` needs to be replaced with the path to the install directory of the clean MPS installation.

2.2 Mac

On Mac installing the prerequisites is somewhat easier than on windows. The first tool which is required for building the extension is xcode. Xcode can be installed by running "xcode-select –install" in the terminal. Apache Ant and Graphviz are also required. Mbeddr provides a custom homebrew repository for installing these. To install the tools run the following command in the terminal:

1. `brew tap mbeddr/mbeddr`
2. `brew update`
3. `brew install gdb76`

After all that is complete the code can be built. This is done by executing the following steps:

1. Clone the repository: "https://github.com/Bartvhelvert/mbeddr.core" using git
2. Execute the command:

```
./gradlew.bat build_mbeddr
```

in the root directory of the project. The build process will now start, this will take about 30-70 minutes to complete depending on the speed of your system.

3. Copy content of the folder

   ```
   ../mbeddr.core/MPS/MPS-mbeddr-2018.1/bin/mac
   ```

   to

   ```
   ../mbeddr.core/MPS/MPS-mbeddr-2018.1/bin
   ```

4. Start MPS by running

   ```
   ../mbeddr.core/MPS/MPS-mbeddr-2018.1/bin/mps.sh
   ```

5. Go to file → settings in MPS (or configure → settings if applicable)
6. Go to Appearance & Behavior → Path variable
7. Add a path variable with the name "mbeddr.github.core.home" which points to your mbeddr core home directory.
8. Open the ../mbeddr.core/code/languages/com.mbeddr.core language (do not migrate)
9. Open the ../mbeddr.core/code/languages/com.mbeddr.cpp language in another window (do not migrate)
10. Right click the cpp project in the logical view and click on "Rebuild with dependencies".

It is also possible to open the project in a clean, separately installed version of MPS instead of the MPS version which is shipped with mbeddr. To make this work some plugins need to be installed to the clean MPS installation. This can be done by running

```
./gradlew install "\-PMPS\_Installation=<path to MPS>"
```

in the root project directory, where `<path to MPS>` needs to be replaced with the path to the install directory of the clean MPS installation.

2.3 Linux

Most linux distributions have the Gnu toolchain available, but in case it is not already available, it should be installed. Additionally, all of the following tools (listed along with the versions required) must be installed and included in the PATH:

- Ant 1.9
- CBMC 5.6

- GCC 4.8
- GDB 7.6
- Graphviz 2.30
- Make 3.81
- MPS 2018.1

After all dependencies are installed the software can be build. This is done by executing with the following steps:

1. Clone the repository: "https://github.com/Bartvhelvert/mbeddr.core" using git
2. Execute the command:

   ```
   ./gradlew.bat build_mbeddr
   ```

   in the root directory of the project. The build process will now start, this will take about 30-70 minutes to complete depending on the speed of your system.
3. Copy content of the folder

   ```
   ../mbeddr.core/MPS/MPS-mbeddr-2018.1/bin/linux
   ```

   to

   ```
   ../mbeddr.core/MPS/MPS-mbeddr-2018.1/bin
   ```
4. Make the shell script executable by running

   ```
   chmod +x ../mbeddr.core/MPS/MPS-mbeddr-2018.1/bin/mps.sh
   ```
5. (optional) Make the linux tools included in mbeddr executable

   ```
   chmod +x ../mbeddr.core/MPS/MPS-mbeddr-2018.1/bin/fsnotifier
   ```

   ```
   chmod +x ../mbeddr.core/MPS/MPS-mbeddr-2018.1/bin/fsnotifier64
   ```
6. Start MPS by running

   ```
   ../mbeddr.core/MPS/MPS-mbeddr-2018.1/bin/mps.sh
   ```
7. Go to file → settings in MPS (or configure → settings if applicable)
8. Go to Appearance & Behavior → Path variable
9. Add a path variable with the name "mbeddr.github.core.home" which points to your mbeddr core home directory.

10. Open the ../mbeddr.core/code/languages/com.mbeddr.core language (do not migrate)
11. Open the ../mbeddr.core/code/languages/com.mbeddr.cpp language in another window (do not migrate)
12. Right click the cpp project in the logical view and click on "Rebuild with dependencies".

It is also possible to open the project in a clean, separately installed version of MPS instead of the MPS version which is shipped with mbeddr. To make this work some plugins need to be installed to the clean MPS installation. This can be done by running

```
./gradlew install "\-PMPS\_Installation=<path to MPS>"
```

in the root project directory, where `<path to MPS>` needs to be replaced with the path to the install directory of the clean MPS installation.

3 Installation procedure

The C++ MPS extensions is currently not installable. The way to use it is by building the source code and adding a dependency on the cpp project manually. After the C++ MPS extension gets merged into mbeddr however it will be part of the standard mbeddr install. The standard mbeddr install comes in 2 forms: As a plugin which can be imported into MPS and as a standalone IDE. The standalone IDE can be downloaded from the mbeddr website and takes the form of a standalone application. The plugin distribution can be installed by downloading the plugins from the mbeddr website and putting them into the plugins directory of an MPS installation.

4 Configuration item list

All deliverables that the Jetbrains C++ extension provides are listed here. The documents provided are in PDF format and the source code is added as a pull request to the Github mbeddr repository.

4.1 Documents

User Requirement Document[2]
Software Requirement Document[6]
Architectural Design Document[7]
Software User Manual[8]
Software Transfer Document(this document)

4.2 Test plans

Unit Test Plan[9]
Acceptance Test Plan[10]

4.3 Software

The core C++ MPS language as an extension to mbeddr.

5 Acceptance test report summary

5.1 First Acceptance Test

The first acceptance test took place on Thursday, June 21. All tests passed except one, the final test, AT-68. The reason for the failure pertained to broken references caused by an omission in the model-to-model generator for header files. This issue was subsequently resolved for the second acceptance test.

5.2 Second Acceptance Test

The second acceptance test took place on Monday, June 25. All tests passed successfully and the ATP document was signed by the client.

6 Software problem reports

6.1 Copy-Pasting

Due to resource constraints the full copy-paste functionality was not able to be implemented. More information on this can be found in section 9.1.

6.2 Completion menu ambiguity

In some cases, the completion menu is ambiguous. There can be two or more different references with the exact same name and description in the completion menu. For now these can be identified by the order in which the references appear in the menu, since the ordering is static.

6.3 Operator Overloading

Operator overloading does not work correctly due to a bug in MPS, evident from errors in the unit tests. Some checking rules do not always apply correctly. As such, rebuilding the typesystem of the language (sometimes several times) can fix the problem. Some tests have also shown that restarting MPS can have the same effect.

7 Software change requests

There were no software change requests fielded during the transfer phase.

8 Software modification reports

The software modifications made after the first acceptance test are explained in section 5.1.

9 Appendices

9.1 Copying and pasting C++ code

To tackle the issue of copying and pasting C++ code inside of MPS, we started out by doing some research into similar projects that have been undertaken in the past. There exist several promising projects using ANTLR grammars, but none of these project could be directly re-used for our implementation. Furthermore, creating an implementation from scratch using ANTLR would a high level of expertise in working with ANTLR and its methods of interfacing with MPS. Due to this feature being requested during the later stages of the project, there was not enough time to do this.

Instead, we decided to create an intention in of MPS that would allow for the copy pasting of some basic aspects of the C++ language, in this case global variable declarations. This mainly served as a proof of concept, to show that copying and pasting could be achieved using our implementation, given enough time to develop it further. This intention is not included in the pull request to the mbeddr project, since it was purely experimental and is not a finished product that can be released as part of the C++ extension.

The intention, called *PasteCPP*, uses the function `TextPasteUtil.getStringFromClipboard()`, found in `jetbrains.mps.ide.datatransfer`, to fetch the C++ code from a user's clipboard as a string variable. This string is then parsed, and subsequently mapped to the corresponding MPS concepts. The parsing of this string was implemented roughly as follows (keep in mind that this implementation *only* supports global variable declarations):

- Everything listed below is contained in a while loop that runs as long as the string variable representing the clipboard content is not yet empty.
- A node variable representing the current line in the editor.
- While the input string from the clipboard starts with any specifier keyword (extern, thread_local, static, etc.), these are removed from the input string and saved for later inclusion.
- If the remaining part of the input string starts with a type (int, bool, etc.), we assume that we are dealing with a global variable declaration. We create a node that represents a global variable declaration, and set the type according to the type given in the string.
- Afterwards, the next word is extracted from the string, which should contain the name of the global variable. The name attribute of the global variable declaration is then set to be this word.
- Recall that specifier keywords such as 'static' had been saved earlier. The attributes of the global variable declaration are now set accordingly.
- If the next part of the string starts with '=', then the global variable also has an initialization value. The initialization of the global variable to this value is therefore set to be equal to the part of the string that follows after the equals sign.

- Finally, the node created earlier that represents the current line in the editor is replaced with the node that represents our global variable declaration. We then create a new node representing an empty line, and select this empty line as the current line such that the while-loop can continue, if there is more on the clipboard. A new node variable representing a new line in the editor is then created for a possible next iteration of the while-loop, which continues until all input has been read.