Eindhoven
University of Technology

C++ JetBrains MPS Extension

version 1.0

Architectural Design 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 Architectural Design Document (ADD) describes the architectural design of the C++ extension for the Jetbrains Meta Programming System (MPS). With the C++ extensions to MPS, users will be able to design C++ languages in MPS while making use of advanced editing features that MPS makes possible. This document complies with the user requirements laid down in the User Requirements Document (URD)[1] and the functional software requirements in the Software Requirements Document (SRD).[2] This document also complies with the ESA software standard.[3]

Contents

1 Introduction **6**
- 1.1 Purpose . . . 6
- 1.2 Scope . . . 6
- 1.3 List of definitions and abbreviations . . . 7
  - 1.3.1 C++ Definitions . . . 7
  - 1.3.2 Abbreviations . . . 9
- 1.4 List of references . . . 9
- 1.5 Overview . . . 9

2 System overview **10**
- 2.1 Background . . . 10
- 2.2 Context and basic design . . . 10
- 2.3 Design decisions . . . 10
  - 2.3.1 Jetbrains MPS . . . 10
  - 2.3.2 Mbeddr . . . 11
  - 2.3.3 Editors . . . 11
  - 2.3.4 Language tests . . . 11
  - 2.3.5 Model Driven Architecture (MDA) . . . 11

3 System context **12**
- 3.1 MPS . . . 12
- 3.2 Mbeddr . . . 13

4 System design **14**
- 4.1 Design method . . . 14
- 4.2 Decomposition description . . . 14
  - 4.2.1 List of components . . . 14
  - 4.2.2 Dependencies . . . 15

5 Detailed design **16**
- 5.1 Component descriptions . . . 16
  - 5.1.1 CPPCL - C++ core language . . . 16
  - 5.1.2 CPPBL - C++ Base Language . . . 17
  - 5.1.3 CL - classes . . . 18
  - 5.1.4 ML - methods . . . 19
  - 5.1.5 KWL - keywords . . . 19
  - 5.1.6 NSL - namespaces . . . 20
  - 5.1.7 OOL - operator overloading . . . 21
  - 5.1.8 TL - templates . . . 21
  - 5.1.9 G - generator . . . 22
  - 5.1.10 E - editor . . . 23
  - 5.1.11 MCL - mbeddr core language . . . 23
  - 5.1.12 MPS . . . 24
- 5.2 Implementation details . . . 25
  - 5.2.1 Type-checking . . . 25
  - 5.2.2 Expressions . . . 25

5.2.3 Keywords 25
5.3 Overview 26

6 Feasibility and resource estimates **29**
6.1 Minimum Requirements 29

7 Requirements traceability matrix **30**
7.1 Mapping of Software Requirements to Architecture Components 30
7.2 Mapping of Architecture Componenets to Software Requirements 39

Document Status Sheet

General

Document title:	Architectural Design Document
Identification:	ADD/ 0.1
Authors:	J.C Rombouts, B.A.J.v.Helvert, B. Smit, D. Drijver, B.I. Munter
Document status:	Final version

Document History

Version	**Date**	**Author(s)**	**Changes**
1.0	21-06-2018	J.C.Rombouts, B.A.J.v.Helvert, B. Smit, D. Drijver	Final version

Document Change Records

Section	Applicable Changes
0.0	Final version

1 Introduction

1.1 Purpose

The Architectural Design Document (ADD) describes the architectural design choices for the C++ extension for MPS that will be implemented by Eded. The document describes how the software is decomposed into components as well as the interfaces and dependencies of the components. An overview of the the relationships with external systems and system design methods are given. Finally, the document describes the relation between the components and the software requirements described in the SRD.[2]

1.2 Scope

Eded is a bachelor end project group working for the TU/e and Océ, a Canon Company. The software end product is an extension to JetBrains MPS which is created by Eded for the use of Océ. 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.[4] 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.[5]

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 rudimentary C++ programming in MPS possible.

1.3 List of definitions and abbreviations

Domain Specific Language	A (typically simplified) computer programming language specialized to a single domain.
MPS	A meta programming system created by JetBrains used to create and utilize domain specific languages.
mbeddr[5]	An extension to MPS created by Itemis, which allows users to develop, among many other things, C applications.

1.3.1 C++ Definitions

Access Modifiers	Modifiers that change the accessibility of parts of the program (classes, methods, etc.) These include the keywords `public, private, and protected`.
Code Generation	Transforming an MPS abstract syntax tree into compilable plain text source code.
Control Structure	A block of code that analyzes variables and chooses a direction in which to go based on given parameters.
Expression	A block of code containing a combination of values, constants, variables, operators and functions that produces a value.
Literal	A way of referring to a fixed value in source code
Memory Management Keywords	Keywords that are used in C++ for memory management (i.e. `new` and `delete`).
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. Namespaces do not have access modifiers.
Primitive Types	The default types included in the C++ prelude (`int, float, double, char, bool, void, wchar_t`).
Projectional Editing	A form of programming in which the programmer directly modifies the abstract syntax tree instead of manipulating text that represents the tree.
Refactor	Restructuring code without changing its external behavior, to improve readability and reduce complexity.
Scope	Visibility of variables and methods of one part of the program to another part of the program.

Statement	A block of code that expresses some action to be carried out.
Template	A type of C++ declaration that allows types to be inferred at compile time, so identical code can be reused by many object types.

1.3.2 Abbreviations

URD	User Requirements Document
SRD	Software Requirements Document
ADD	Architectural Design Document
UTP	Unit Test Plan
DSL	Domain Specific Language
ESA	European Space Agency
IDE	Integrated Development Environment
MPS	Jetbrains Meta-programming System
AST	Abstract Syntax Tree
MDA	Model Driven Architecture
OMG	Object Management Group

1.4 List of references

[1] Eded, *User Requirements Document Version 1.0.1*

[2] Eded, *Software Requirements Document Version 1.0*

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

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

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

[6] Voelter, M. ,2014, *Generic Tools, Specific Languages* , 18 June 2014

[7] Kevin Lano, 2009, *Model driven Software Development with UML and Java*, CENGAGE Learning, ISBN 978-1-84480-952-3

1.5 Overview

The remainder of this document is composed into seven sections. Section 2 describes the background of the project and contains a motivation of all used software tools. Section 2 describes the architectural design decisions. Section 3 gives a more detailed description of the architecture of MPS and mbeddr. Section 4 contains an overview of the components, including the dependencies between them. Section 5 describes in detail how the software requirements defined in the SRD are implemented in the architecture. Section 6 describes the minimum resource requirements for running the C++ MPS extension. Finally, section 7 shows the mapping of the software requirements in the SRD to the components described in section 5.

2 System overview

The MPS C++ extension is a an extension for Jetbrains MPS and is build on top of mbeddr. With this extension, users will be able to extend their existing models with C++ code to make C++ programming in MPS possible. The extension provides code auto-completion and suggestions depending on the current scope of the user in the editor.

2.1 Background

In the industry, there is a large demand for C++ support, since embedded software often uses C++. The MPS C++ extension created by Eded will have, among other things, support for code editing facilities, code generation, class implementation, namespace implementation, and template implementation. The C++ MPS extension allows users to fluidly develop applications and domain specific languages that compile directly to pure C++, regardless of the level of abstraction that will be used.

2.2 Context and basic design

Jetbrains MPS consist of many sub components, indicated in Figure 1. The specific parts of Jetbrains MPS are also explained in section 3. The C++ MPS extension is build on top of mbeddr, which is a plug-in for Jetbrains MPS.

2.3 Design decisions

This section describes the top-level architecture of the C++ MPS extension. Because the extension we implemented is build on top of MPS and on top of mbeddr, there are no classical architectural patterns and styles than can be used.

The extension is implemented such that it works in Jetbrains MPS and therefore there was no room for discussion to use Jetbrains MPS or not. In consolation with the client it is chosen that the C++ MPS extension is build on top of mbeddr, which covers many basic C implementations already. The `wip/cpp_mps2018` branch is used as a starting point for our project. MPS allows users to define programming languages with an editor for the end user. This is explained in more detail in section 3. To develop languages in MPS, MPS has it owns restrictions.

2.3.1 Jetbrains MPS

Jetbrains MPS is a tool for designing domain specific languages that bridge the cognitive gap between the domain expert and the programming interface. There exist already some MPS plugins providing support for other (programming) languages, like mbeddr supports the C language. MPS plug-ins abstract the original language, making it easier for the user to create domain specific languages in a certain language. Therefore, MPS makes it possible for users to change and extend languages right in the editor. Since Océ wants to use this C++ extension internally, the C++ extension must work as a plug-in of Jetbrains MPS. MPS is already used by the developers of Océ. Océ is interested in practically usable code that can interface with the rest of mbeddr. Jetbrains MPS translates the platform specific models to code that embedded systems can run. This requires mapping between domain specific language and the actual code. The mappings

and transformations should be implemented. Jetbrains MPS is structured into separate models, as described in section 3. This parts can be seen as models. As one can see, Jetbrains MPS is build according to the model driven architecture, since it shares all the characteristics of it. MDA abstracts the underlying code for the end user, as is the same goal for domain specific languages.

2.3.2 Mbeddr

As mentioned before, mbeddr is an extension to MPS that allows users to develop, among many other things, C applications. Besides the C support, it also languages for embedded software such as state machines. Mbeddr has a built in IDE including syntax coloring, code completion, go to definition, real-time type checks, quick fixes, refactoring and many other useful features. Although C++ is not a strict superset of C, it provides a lot of C-features out of the box. Therefore, mbeddr is a good starting point for the project. By adding code to the `wip/cpp_mps2018` branch, Océ is able to review the code to check if it is a proper mbeddr extension. The C++ extension can be merged into the mbeddr core project.

2.3.3 Editors

The MPS editor has a non-textual presentation of program code. This eliminates the need for parsers. The code is represented in an Abstract Syntax Tree (AST), consisting of nodes with properties, children and references. The MPS editor visualizes the AST in a user friendly way. The user basically edits the AST in the editor. In the C++ extension, the editor should feel like editing text in an IDE, including auto-completion and intentions. At the end, the user can transform the user code into general purpose languages, in our case, C++.

2.3.4 Language tests

The constraints in the projectional editor can be translated to language tests. The editor tests allow allows developers to test the fluidness of the editor, for example the actions, intension, refactoring and auto-completion. Test cases contains how the code looks before the transformations and after and also what the trigger is to transform the code. Test cases also check for type-system errors and warnings, for example what is possible in a certain scope and what not. Test cases check if some error messages pop up when a user types something in a certain scope where the editor expects a specific type. A more technical explanation of features that are to be tested can be found in the Unit Test Plan (UTP).

2.3.5 Model Driven Architecture (MDA)

It is hard to map a classical architectural style to the implementation, since this is not a traditional software project. The architecture pattern that comes closest to MPS is Model Driven Architecture (MDA). MDA is launched by the Object Management Group (OMG).[7] MDA is an architectural pattern that represents the architectural design of the software. Figure 1 describes the different types of models that MPS supports. When the domain specific language is implemented, end users can create and model the system on a high abstraction level. Thereafter, the software will transform this to a purpose generic

language. MDA is focused on implementing the important parts of the application, while the user does not have to deal with the technical details under it. The main goal of MDA is to separate design from the actual implementation code. The MDA tool that corresponds to domain specific languages is the transformation tool. This is a tool that transforms models (e.g. the Abstract Syntax Tree) into other models or into code, e.g. C++.

3 System context

3.1 MPS

The Jetbrains C++ Extension is made in the MPS environment. MPS allows users to define programming languages with an editor for the end user. A MPS language is build with models. Every model has a different purpose in the language. MPS provides support for specifying structure, syntax, type system, generators and IDE support. The following paragraph briefly describes these concepts.

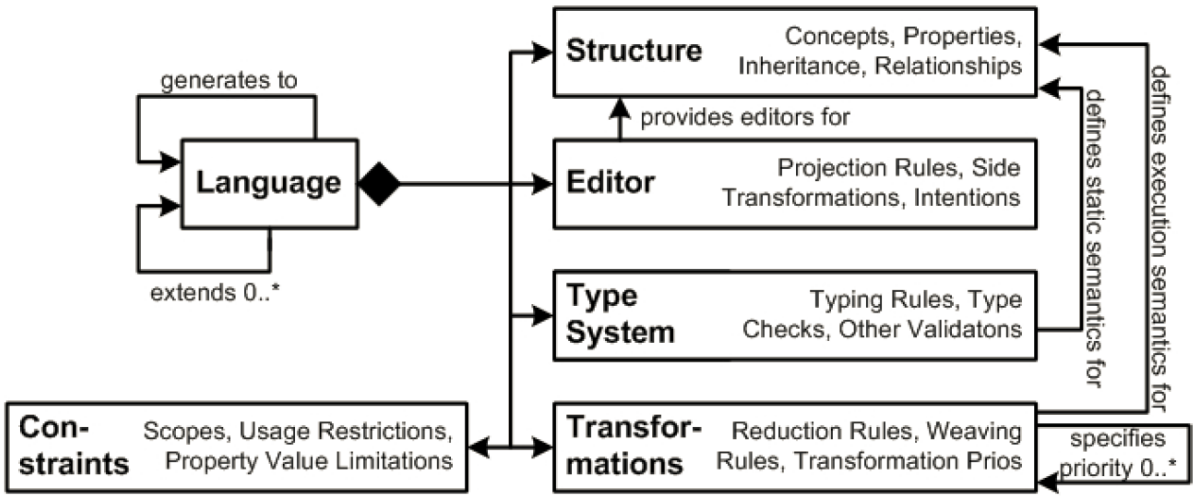

Figure 1: High level presentation of the structure of MPS. A MPS language contains many different type of concepts. This figure shows the most important ones.[6]

Structure
The structure model defines the AST of the language. It specifies the relations between different nodes. A concept can extend or implement from other concepts, it can hold primitive variables, specify child nodes and hold references to other nodes. A language consist of a number of concepts. To enter an instance of the concept, the user types (or selects from the auto - completion menu)a string, also known as alias. Most of the cases the alias is the same as the keyword. Concepts can have properties and children.

Editor
The editor is like the front-end of a structure. For a node in the AST it specifies what that node looks like and how it behaves in the projectional editor. An editor consist of a collection of editor cells. Actions define how users interact with the notation. For example, what happens when the user presses `backspace` on a given cell or what the

editor should do on the left or right side when the user types an alias.

Type System
The type system model validates the language. The type-system has to be programmed using the model language with the projectional editor. The type system module checks the written code for errors and shows those errors in the editor, instead of preventing of invalid use.

Constraints
The constraints model puts limitations on what is allowed in the language. It can be programmed in the model language with the projectional editor. The main difference with the type-system is that constraints don't allow the user to perform an invalid action while the type-system provides an error if the users performs an invalid action. Scopes express express which target nodes are visible to a reference, and shows them in the auto-completion menu.

Actions
Actions allow the end user to use shortcuts to add nodes to the AST. Actions can be implemented by the language designer to enhance the editing experience.

Intentions
Intentions are a concept which was introduced by Jetbrains for their IDE's. Intentions are suggestions to the end-user on their code. Intentions can be invoked to make a change to the code.

Dataflow
Data-flow can specify an order to how the language should be written.

TextGen
The Text-gen model specifies how the language is generated to text. In our case it specifies how the C++ AST is written to textual C++. The text-gen is also written in the sModel language.

3.2 Mbeddr

Mbeddr is a set of integrated and extensible languages for embedded software engineering, it is build in MPS and contains a lot of tools that serve as an extension to MPS. Mbeddr contains a cleaned up version of the C99 programming language which is a version of C. The C++ extension builds on top of the C99 language and changes some of its features. Besides C support, mbeddr also contains a set of extensions such as state machines and interfaces and components. The user can use low level C - code combined with higher level abstractions in the editor. Since C++ is not a strict superset of C99 some changes have been made in regards to the C editing in the C++ language.

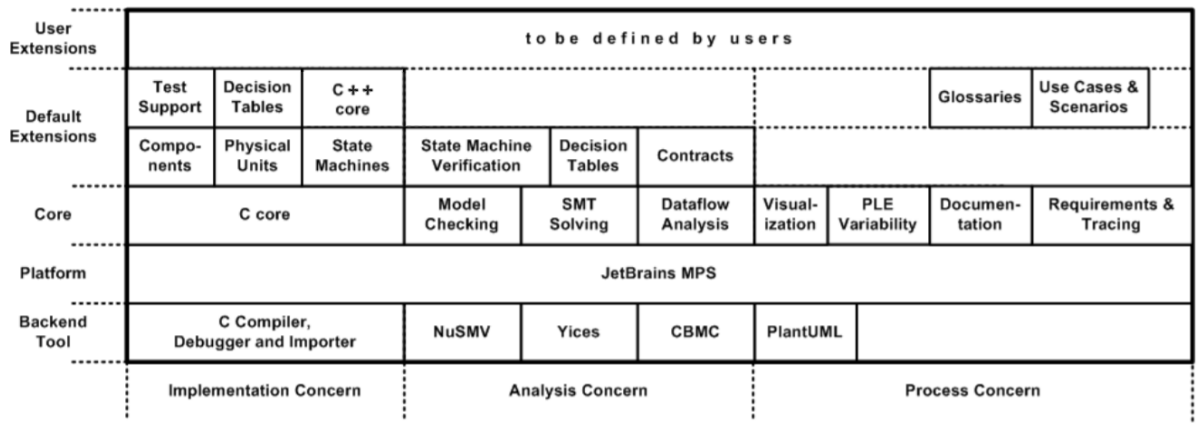

Figure 2: A high level overview of all different sub-systems which are included in the mbeddr project.[6]

On top of all the mbeddr implementation modules it also includes extensions to MPS. These however according to the mbeddr team are going to migrated to another project called MPS extensions very soon. Our C++ projects is very dependent on these MPS extensions. Grammar cells are one of the MPS extensions which are extensively used throughout the project. Grammar cells are editor cells in the editor model which automatically also perform a lot of other operations on the editor. The usage of grammar cells results in a better editing experience for the end user.

4 System design

This section describes the system design of the C++ MPS extension. A list of components and dependencies between the said components are included in this section. As mentioned before, the C++ extension is build on top of mbeddr.

4.1 Design method

The components are implemented as an extension to mbeddr. MPS and mbeddr components are used to implement the C++ language extension in MPS.

4.2 Decomposition description

All components described in this sub section are part of the larger C++ language implemented in MPS. For better readability and structure, the implementation can be grouped into a number of components, which are described in a list below.

4.2.1 List of components

- CPPCL - C++ core language: the complete C++ language, can be used by other DSL's.
- CPPBL - base language: the language which contains all the language concepts, divided in several parts:

  - CL - classes: Part of the base language which contains all concepts for classes and class members.
  - ML - methods: language which contains all the concepts for methods.
  - KWL - keywords: Part of the base language which contains all the concepts for keywords such as this, auto and static.
  - NSL - namespaces: Part of the base language which contains all the concepts for namespaces.
  - OOL - operator overloading: Part of the base language which contains all concepts for operator overloading.
  - TL - templates: Language which contains all concepts for templates.
- G - generator: Contains the text generation for all the concepts.
- E - editor (name could change)
- MCL - mbeddr-core: The C-language which was already implemented by the mbeddr team.
- MPS: Jetbrains meta-programming System in which the language runs.

4.2.2 Dependencies

In the section above the different components are explained shortly, Section 5.1 shows a more detailed explanation of all the components. Figure 3 shows the dependencies between the components. C++ dev kit contains everything that is added by Eded, it depends on mbeddr because all the C features of C++ are used from mbeddr. It also depends on MPS for some editor features.
The base language contains all the concepts for the C++ language it depends on the generator for the generation of C++ code. The editor depends on the base language for the editing of C++ in the MPS editor.
The base language is divided in several components, namely: Methods, Classes, Namespaces, Templates, Keywords and Operator overloading. Methods depend on Classes because they need to be in classes, Templates depends on Classes and methods because they both can be used for templates. Namespaces depends on Classes and Methods because they both can be in Namespaces. Note that the concepts of MPS are not designed in this figure. For more detailed description of MPS, we refer to Figure 1.

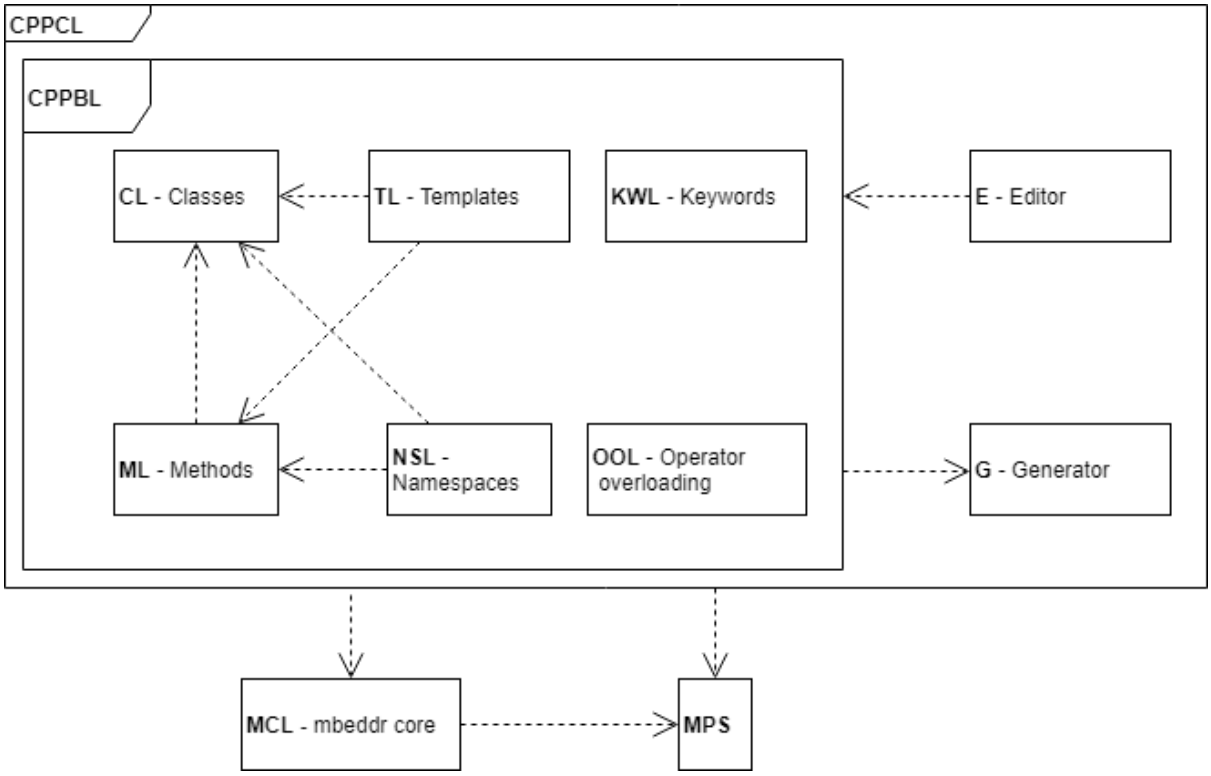

Figure 3: A high level overview of the implemented language concepts.

5 Detailed design

Each component described in this section is included in Figure 3, including the dependencies between them and between mbeddr and MPS. A more detailed overview of MPS concepts is included in Figure 1.

5.1 Component descriptions

5.1.1 CPPCL - C++ core language

Purpose
SR-38, 39, 40, 41, 43, 44, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 85, 105, 110, 111, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 138, 139, 140, 143, 144, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231
Function
CPPCL is used to group everything of the C++ MPS language together into one Dev-kit such that when it is used in another language it only have to depend on this Dev-kit.

Subordinates
C++ Base language (CPPBL) , Generator (G) , Editor (E)

Dependencies
CPPCL depends on the mbeddr core language (MCL) for the C functionality which is also used in C++. CPPCL is also dependent on MPS for the type system, constraints, actions, intentions and Text-gen.

Interfaces
The user can interact with the CPPCL with the MPS text editor. The generator can generate C++ code, which is a representation of the text in the editor, as output.

Resources
In order to use the CPPCL, MPS is needed.

References
The dependencies of the CPPCL are shown in Figure 3.

Processing
The C++ base language (CPPBL) is used to validate if the text that the user is typing is allowed there. The generator describes how the text in the editor is represented as C++ plain text code and will generate that C++ code.

Data
The created files for the text editing in MPS using the CPPCL is stored on the the selected storage device of the user.

5.1.2 CPPBL - C++ Base Language

Purpose
SR-38, 39, 40, 41, 43, 44, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 85, 105, 110, 111, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 138, 140, 144, 147, 148, 149, 150, 151
Function
The CPPBL contains all the concepts of the C++ language which are not in C. It is used to determine what the user is allowed to type at a certain scope in the editor.

Subordinates
CL - classes, ML - methods, TL - templates, NSL - namespaces, KWL - keywords, OOL - operator overloading.

Dependencies
The CPPBL is dependent on mbeddr core language (MCL) for all the C functionality.

Interfaces
The MPS editor allows the user to use this language, prevents user for typing invalid

code and notifies user when invalid references are used.

Resources
None

References
Figure 3 shows how to C++ Base language (CPPBL) is divided in sub languages and the dependencies it has with other parts.

Processing
The concepts of this language are used to determine what the user can type at a certain scope in the editor. Constraints puts limitations on what is allowed in the language.

Data
None

5.1.3 CL - classes

Purpose
SR-38, 39, 40, 41, 47, 48, 51, 52, 61, 62, 63, 64, 65, 144
Function
CL is part of the C++ base language. It contains all the concepts related to classes.

Subordinates
None

Dependencies
CL is dependent on mbeddr core language (MCL).

Interfaces
The MPS editor allows the user to use this language, prevents user for typing invalid code and notifies user when invalid references are used.

Resources
None

References
Figure 3 shows which other sub-parts of the language depends on CL.

Processing
The concepts of this language are used to determine what the user can type in a class scope in the editor. Constraints puts limitations on what is allowed in the language.

Data
None

5.1.4 ML - methods

Purpose
SR-43, 44
Function
ML is a part of the C++ base language (CPPBL), it contains all the concepts related to methods.

Subordinates
None

Dependencies
ML is dependent on CL - classes and MCL - mbeddr core language.

Interfaces
The MPS editor allows the user to use this part of language.

Resources
None

References
figure 3 shows which other sub-parts of the language depends on ML and where ML depends on.

Processing
The concepts of this part of the language are used to determine what the user can type at a certain place in the editor related to methods.

Data
None

5.1.5 KWL - keywords

Purpose
SR-49, 50, 85, 105, 110, 111, 120, 121, 122, 123, 124, 125, 126, 127, 128

Function
KWL is a part of the C++ base language (CPPBL), it contains all the concepts related to keywords.

Subordinates
None

Dependencies
KWL is dependent on mbeddr core language (MCL).

Interfaces
The MPS editor allows the user to use this language, prevents user for typing invalid

code and notifies user when invalid references are used.

Resources
None

References
none

Processing
The concepts of this language are used to determine what the user can type in a keyword scope in the editor. Constraints puts limitations on what is allowed in the language.

Data
None

5.1.6 NSL - namespaces

5.1.6.1 Purpose

SR-66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82

Function
NSL is a part of the base language, it contains all the concepts related to namespaces.

Subordinates
None

Dependencies
NSL is dependent on CL - classes, ML - methods and MCL - mbeddr core language

Interfaces
The MPS editor allows the user to use this part of language.

Resources
None

References
figure 3 shows where NSL depends on.

Processing
The concepts of this language are used to determine what the user can type in a namespace scope in the editor. Constraints puts limitations on what is allowed in the language.

Data
None

5.1.7 OOL - operator overloading

Purpose
SR-129, 130

Function
OOL is a part of the base language, it contains all the concepts related to operator overloading.

Subordinates
None

Dependencies
OOL is dependent on MCL - mbeddr core language.

Interfaces
The MPS editor allows the user to use this part of language.

Resources
None

References
figure 3 shows where OOL depends on.

Processing
The concepts of this part of the language are used to determine what the user can type at a certain place in the editor related to operator overloading.

Data
None

5.1.8 TL - templates

Purpose
SR-53, 54, 55, 56, 57, 58, 59

Function
TL is a part of the base language, it contains all the concepts related to templates.

Subordinates
None

Dependencies
TL is dependent on CL - classes, ML - methods and MCL - mbeddr core language

Interfaces
The MPS editor allows the user to use this part of language.

Resources
None

References
figure 3 shows where TL depends on.

Processing
The concepts of this part of the language are used to determine what the user can type at a certain place in the editor related to templates.

Data
None

5.1.9 G - generator

Purpose
SR-152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231

Function
the generator is responsible for generating C++ code from the text the user typed in the MPS editor.

Subordinates
None

Dependencies
The generator depends on the CPPBL - Base language.

Interfaces
Input for the generator is the code that the user has typed. Output is the generated plain text C++ code.

Resources
None

References
Figure 3 shows where the generator depends on.

Processing
The generator does the generation with the generation rules which are specified in the Text-gen files.

Data

The output of the generator are text files with plain text C++ code.

5.1.10 E - editor

Purpose
SR-131, 132, 133, 134, 138, 139, 140, 143, 145, 146

Function
The editor is responsible for making the user able to edit code in the MPS editor. One of the main purposes is to make the editing more user friendly by making what you type look more like what you want to get. (What you type is what you get) and giving warnings/errors for things that are not allowed.

Subordinates
None

Dependencies
The editor depends on the CPPBL - Base language.

Interfaces
the editing is done in the MPS editor.

Resources
None

References
figure 3 shows where the editor depends on.

Processing
The editor determines, when the user is typing in the editor, if what the user typed is allowed or should give a error/warning. This is done with rules that are specified in the editor.

Data
none

5.1.11 MCL - mbeddr core language

Purpose
SR-1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 37, 83, 84, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 106, 107, 108, 109, 112, 113, 114, 115, 117, 136, 145

Function
The mbeddr core language contains all the concepts for the C language. It is used to determine what the user is allowed to type at a certain position in the editor related to

C features.

Subordinates
None

Dependencies
TMCL depends on MPS.

Interfaces
the editing is done in the MPS editor.

Resources
None

References
figure 3 shows where MCL depends on.

Processing
The concepts of this language are used to determine what the user can type at a certain place in the editor.

Data
none

5.1.12 MPS

Purpose
SR-36, 42, 45, 46, 60, 116, 118, 119, 135, 137, 141, 142, 146

Function
MPS is the program where the user can edit the C++ code using the CPPCL.

Subordinates
None

Dependencies
none

Interfaces
The user can interact with MPS using the GUI, mainly via the text editor.

Resources
None

References
References to MPS can be found at: https://www.jetbrains.com/mps/concepts/

Processing
Most of the processing is done by MPS internally.

Data
MPS uses around 1.6GB of storage including mbeddr. However, when building, the required storage can reach up to 5.0GB.

5.2 Implementation details

5.2.1 Type-checking

Mbeddr is using concepts to specify the type of a concept. An object can extend one or multiple concepts in order to define its type. The MPS editor will give the user feedback during the editing of code if a type error is made. Parameters, casting and expressions and Thus during operations intentions will not suggest objects or operations that are not type compatible with each other. type-checking is also operational for inheritance. Objects inherit the object type of their parent class.

Mbeddr was already capable of type-checking literals within the C language, however as C is not object oriented, type-checking still had to be added for user defined classes, structs and templates. Just like literals are explicitly defined, user defined classes are also created as a new custom type as is done in C++, allowing the type-checking system in place to directly infer the types from the user's input.

The auto keyword is an exception to this rule, as object type is not stated during object creation. Instead the auto keyword will infer the object type from the assignment operation. Type-checking does not do anything during assignment, this is as intended.

Templates do eventually require the user to define a type, thus it is able to type-check in the same manner that a normal class would.

5.2.2 Expressions

Literals are the leaf concepts that represent C++ primitive types, it is contained entirely within mbeddr. A literal is always created with a type, unless the auto keyword is used, the type is inferred when variable assignment is evaluated.

All literals require a type in order for the mbeddr type-checking to function. The other expressions are entirely covered in mbeddr as well, there are unary expressions that operate on a single operand. Binary expressions operate on two operands. In both cases type-checking is performed for all operands of the expressions.

5.2.3 Keywords

We added keywords to mbeddr that were present in `C++14` but not in the `C99` implementation that mbeddr decided to use. The auto keyword is simply added functionality, replacing the type of a literal it is able to automatically infer the object type, `thread_local` variables were also added in order to.

Besides added functionality the `import` keyword was also added in order to support header file editing, mbeddr decided to automatically generate the header files for their implementation of `C99`. Lastly we added keywords that made use of the fact that `C++14` is object oriented. The `this` keyword refers to the object in the current context.

5.3 Overview

This section contains a overview of all the concepts that are in the base language. It is important to note that the list is not a complete source code list, but only includes the concepts. It is not possible to automatically generate the source code to a detailed design, as therefore we included the detailed design in section 5. The list of concepts is divided in subsections which are the same as the components of the base language as described in section 4.2.1.

CL - classes

- AttributeDeclaration
- ClassDeclaration
- ClassType
- ClassTypeAccessor
- DeleteDeclaration
- EClassMemberVisibility
- EmptyClassContent
- GlobalVarDecCPP
- IClassMemberDeclaration
- IClassTyped
- INamedClassMemberDeclaration
- InheritanceInstance
- InnerClassType
- LocalClassVariableDeclaration
- StructStub

ML - methods

- ClassConstructorDeclaration
- ClassConstructorSignature
- ConstructorInitializedAttribute
- ConstructorInitializedConstructor
- ConstructorVoidType
- IAmConstructorInitializable
- MethodDeclaration
- MethodPrototype
- MethodSignature

KWL - keywords

- AutoType
- ClassStaticVarRef
- IConstExprFlagConcept
- IExplicitFlagConcept
- IExternalFlagConcept
- IInlineFlagConcept
- IMutableFlagConcept
- IPureVirtualFlagConcept
- IStaticFlagConcept
- IThreadLocalFlag
- IVirtualFlagConcept
- IVolatileFlagConcept
- NewDeclaration
- NewObjectInitializer
- NullPointerLiteral
- NullPointerType
- StaticVar

NSL - namespaces

- GlobalUsingGeneralNamespaceDeclaration
- GlobalUsingNamespaceAttributeDeclaration
- GlobalUsingNamespaceMethodDeclaration
- INamedNamespaceMemberDeclaration
- INamespaceMemberDeclaration
- NamespaceAttributeRef
- NamespaceClassInstance
- NamespaceClassInstanceAttributeRef
- NamespaceClassInstanceMethodCall
- NamespaceDeclaration
- NamespaceMethodCall
- UsingGeneralNamespaceDeclaration
- UsingNamespaceAttributeDeclaration
- UsingNamespaceMethodDeclaration

OOL - operator overloading

- EOperatorType
- EOverloadableOperator
- OperatorOverloadDeclaration
- OperatorOverloadPrototype
- OperatorOverloadSignature

TL - templates

- ITemplate
- ITemplateArg
- ITemplateImpl
- ITemplateModuleContent
- ITemplateParam
- ITemplateParamWithDefault
- RegularFunctionCall
- TemplateClassEclaration
- TemplateClassType
- TemplateFunction
- TemplateFunctionCall
- TemplateInheritanceInstance
- TemplateInternalMethodCall
- TemplateMethodDeclaration
- TemplateQualifiedMethodCall
- TemplateStub
- TemplateTypeDef
- TemplateTypeDefWithDefault
- TemplateTypeRef
- TemplateValueArg
- TemplateValueParam
- TemplateValueParamWithDefault
- TemplateValueRef
- TypeTemplateArg

6 Feasibility and resource estimates

It is not possible to realistically analyze the estimated performance for any actions the user wishes to take using Eded's software extension. The C++ extension created by Eded does not influence the performance in a significant manner, the complexity of the code written by the users themselves and the performance of MPS' code generation methods will determine the eventual generation time.

6.1 Minimum Requirements

The following holds for running both MPS, mbeddr and our extension combined:

CPU	64-bit Dual core from Intel or AMD at 2.0 GHz or higher
RAM	4GB or more
Storage	5.0GB or more
Software	Java 1.6 MPS 2018.1
Operating System	Windows 8.0 or higher Mac OS X 10.6.8 or higher Linux supporting Java 1.6 or higher

7 Requirements traceability matrix

7.1 Mapping of Software Requirements to Architecture Components

SRD	ADD
SR-1	MCL
SR-2	MCL
SR-3	MCL
SR-4	MCL
SR-5	MCL
SR-6	MCL
SR-7	MCL
SR-8	MCL
SR-9	MCL
SR-10	MCL
SR-11	MCL
SR-12	MCL
SR-13	MCL
SR-14	MCL
SR-15	MCL
SR-16	MCL
SR-17	MCL
SR-18	MCL
SR-19	MCL
SR-20	MCL
SR-21	MCL
SR-22	MCL

SR-23	**MCL**
SR-24	**MCL**
SR-25	**MCL**
SR-26	**MCL**
SR-27	**MCL**
SR-28	**MCL**
SR-29	**MCL**
SR-30	**MCL**
SR-31	**MCL**
SR-32	**MCL**
SR-33	**MCL**
SR-34	**MCL**
SR-35	**MCL**
SR-36	**MPS**
SR-37	**MCL**
SR-38	**CL, CPPBL, CPPCL**
SR-39	**CL, CPPBL, CPPCL**
SR-40	**CL, CPPBL, CPPCL**
SR-41	**CL, CPPBL, CPPCL**
SR-42	**MPS**
SR-43	**ML, CPPBL, CPPCL**
SR-44	**ML, CPPBL, CPPCL**
SR-45	**MPS**
SR-46	**MPS**
SR-47	**CL, CPPBL, CPPCL**

SR-48	**CL, CPPBL, CPPCL**
SR-49	**KWL, CPPBL, CPPCL**
SR-50	**KWL, CPPBL, CPPCL**
SR-51	**CL, CPPBL, CPPCL**
SR-52	**CL, CPPBL, CPPCL**
SR-53	**TL, CPPBL, CPPCL**
SR-54	**TL, CPPBL, CPPCL**
SR-55	**TL, CPPBL, CPPCL**
SR-56	**TL, CPPBL, CPPCL**
SR-57	**TL, CPPBL, CPPCL**
SR-58	**TL, CPPBL, CPPCL**
SR-59	**TL, CPPBL, CPPCL**
SR-60	**MPS**
SR-61	**CL, CPPBL, CPPCL**
SR-62	**CL, CPPBL, CPPCL**
SR-63	**CL, CPPBL, CPPCL**
SR-64	**CL, CPPBL, CPPCL**
SR-65	**CL, CPPBL, CPPCL**
SR-66	**NSL, CPPBL, CPPCL**
SR-67	**NSL, CPPBL, CPPCL**
SR-68	**NSL, CPPBL, CPPCL**
SR-69	**NSL, CPPBL, CPPCL**
SR-70	**NSL, CPPBL, CPPCL**
SR-71	**NSL, CPPBL, CPPCL**
SR-72	**NSL, CPPBL, CPPCL**

SR-73	**NSL, CPPBL, CPPCL**
SR-74	**NSL, CPPBL, CPPCL**
SR-75	**NSL, CPPBL, CPPCL**
SR-76	**NSL, CPPBL, CPPCL**
SR-77	**NSL, CPPBL, CPPCL**
SR-78	**NSL, CPPBL, CPPCL**
SR-79	**NSL, CPPBL, CPPCL**
SR-80	**NSL, CPPBL, CPPCL**
SR-81	**NSL, CPPBL, CPPCL**
SR-82	**NSL, CPPBL, CPPCL**
SR-83	**MCL**
SR-84	**MCL**
SR-85	**KWL, CPPBL, CPPCL**
SR-86	**MCL**
SR-87	**MCL**
SR-88	**MCL**
SR-89	**MCL**
SR-90	**MCL**
SR-91	**MCL**
SR-92	**MCL**
SR-93	**MCL**
SR-94	**MCL**
SR-95	**MCL**
SR-96	**MCL**
SR-97	**MCL**

SR-98	**MCL**
SR-99	**MCL**
SR-100	**MCL**
SR-101	**MCL**
SR-102	**MCL**
SR-103	**MCL**
SR-104	**MCL**
SR-105	**KWL, CPPBL, CPPCL**
SR-106	**MCL**
SR-107	**MCL**
SR-108	**MCL**
SR-109	**MCL**
SR-110	**KWL, CPPBL, CPPCL**
SR-111	**KWL, CPPBL, CPPCL**
SR-112	**MCL**
SR-113	**MCL**
SR-114	**MCL**
SR-115	**MCL**
SR-116	**MPS**
SR-117	**MCL**
SR-118	**MPS**
SR-119	**MPS**
SR-120	**KWL, CPPBL, CPPCL**
SR-121	**KWL, CPPBL, CPPCL**
SR-122	**KWL, CPPBL, CPPCL**

SR-123	**KWL, CPPBL, CPPCL**
SR-124	**KWL, CPPBL, CPPCL**
SR-125	**KWL, CPPBL, CPPCL**
SR-126	**KWL, CPPBL, CPPCL**
SR-127	**KWL, CPPBL, CPPCL**
SR-128	**KWL, CPPBL, CPPCL**
SR-129	**OOL, CPPBL, CPPCL**
SR-130	**OOL, CPPBL, CPPCL**
SR-131	**E, CPPBL, CPPCL**
SR-132	**E, CPPBL, CPPCL**
SR-133	**E, CPPBL, CPPCL**
SR-134	**E, CPPBL, CPPCL**
SR-135	**MPS**
SR-136	**MCL**
SR-137	**MPS**
SR-138	**E, CPPBL, CPPCL**
SR-139	**E, CPPCL**
SR-140	**E, CPPBL, CPPCL**
SR-141	**MPS**
SR-142	**MPS**
SR-143	**E, CPPCL**
SR-144	**CL, CPPBL, CPPCL**
SR-145	**E, MCL**
SR-146	**E, MPS**
SR-147	**CPPBL**

SR-148	**CPPBL**
SR-149	**CPPBL**
SR-150	**CPPBL**
SR-151	**CPPBL**
SR-152	**G, CPPCL**
SR-153	**G, CPPCL**
SR-154	**G, CPPCL**
SR-155	**G, CPPCL**
SR-156	**G, CPPCL**
SR-157	**G, CPPCL**
SR-158	**G, CPPCL**
SR-159	**G, CPPCL**
SR-160	**G, CPPCL**
SR-161	**G, CPPCL**
SR-162	**G, CPPCL**
SR-163	**G, CPPCL**
SR-164	**G, CPPCL**
SR-165	**G, CPPCL**
SR-166	**G, CPPCL**
SR-167	**G, CPPCL**
SR-168	**G, CPPCL**
SR-169	**G, CPPCL**
SR-170	**G, CPPCL**
SR-171	**G, CPPCL**
SR-172	**G, CPPCL**

SR-173	**G, CPPCL**
SR-174	**G, CPPCL**
SR-175	**G, CPPCL**
SR-176	**G, CPPCL**
SR-177	**G, CPPCL**
SR-178	**G, CPPCL**
SR-179	**G, CPPCL**
SR-180	**G, CPPCL**
SR-181	**G, CPPCL**
SR-182	**G, CPPCL**
SR-183	**G, CPPCL**
SR-184	**G, CPPCL**
SR-185	**G, CPPCL**
SR-186	**G, CPPCL**
SR-187	**G, CPPCL**
SR-188	**G, CPPCL**
SR-189	**G, CPPCL**
SR-190	**G, CPPCL**
SR-191	**G, CPPCL**
SR-192	**G, CPPCL**
SR-193	**G, CPPCL**
SR-194	**G, CPPCL**
SR-195	**G, CPPCL**
SR-196	**G, CPPCL**
SR-197	**G, CPPCL**

SR-198	**G, CPPCL**
SR-199	**G, CPPCL**
SR-200	**G, CPPCL**
SR-201	**G, CPPCL**
SR-202	**G, CPPCL**
SR-203	**G, CPPCL**
SR-204	**G, CPPCL**
SR-205	**G, CPPCL**
SR-206	**G, CPPCL**
SR-207	**G, CPPCL**
SR-208	**G, CPPCL**
SR-209	**G, CPPCL**
SR-210	**G, CPPCL**
SR-211	**G, CPPCL**
SR-212	**G, CPPCL**
SR-213	**G, CPPCL**
SR-214	**G, CPPCL**
SR-215	**G, CPPCL**
SR-216	**G, CPPCL**
SR-217	**G, CPPCL**
SR-218	**G, CPPCL**
SR-219	**G, CPPCL**
SR-220	**G, CPPCL**
SR-221	**G, CPPCL**
SR-222	**G, CPPCL**

SR-223	**G, CPPCL**
SR-224	**G, CPPCL**
SR-225	**G, CPPCL**
SR-226	**G, CPPCL**
SR-227	**G, CPPCL**
SR-228	**G, CPPCL**
SR-229	**G, CPPCL**
SR-230	**G, CPPCL**
SR-231	**G, CPPCL**

7.2 Mapping of Architecture Componenets to Software Requirements

ADD	SRD
MCL	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 37, 83, 84, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 106, 107, 108, 109, 112, 113, 114, 115, 117, 136, 145
MPS	36, 42, 45, 46, 60, 116, 118, 119, 135, 137, 141, 142, 146
CL	38, 39, 40, 41, 47, 48, 51, 52, 61, 62, 63, 64, 65, 144

CPPBL	**38, 39, 40, 41, 43, 44, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 85, 105, 110, 111, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 138, 140, 144, 147, 148, 149, 150, 151**
CPPCL	**38, 39, 40, 41, 43, 44, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 85, 105, 110, 111, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 138, 139, 140, 143, 144, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231**
ML	**43, 44**
KWL	**49, 50, 85, 105, 110, 111, 120, 121, 122, 123, 124, 125, 126, 127, 128**
TL	**53, 54, 55, 56, 57, 58, 59**

NSL	**66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82**
OOL	**129, 130**
E	**131, 132, 133, 134, 138, 139, 140, 143, 145, 146**
G	**152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231**