Eindhoven
University of Technology

C++ JetBrains MPS Extension

version 1.0

Software User Manual

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 User Manual (SUM) describes exemplary usage of the C++ language extension from the JetBrains Meta Programming System (MPS). With the C++ extension to MPS, users will be able to design C++ programs in MPS while using the advanced editing features that MPS provides.

Contents

Document Status Sheet

General

Document title:	Software User Manual
Identification:	SUM/ 1.0
Authors:	J.C.W. Hofs, J. van der Ster, B. Smit, R.J.A. Surtel, N.J. Donnelly, B.A.J. van Helvert
Document status:	Final version

1 Introduction

1.1 Intended readership

This document is intended for the users of the C++ language extension to Mbeddr in MPS. There is only one user category, namely programmers. They will use the C++ language in MPS to design their own domain specific language. They can also convert other models made in MPS into our C++ extension and generate C++ code.
We assume the user is familiar with MPS and C++.

1.2 Applicability

This Software User Manual applies to the latest release of the C++ extension.

1.3 Purpose

The purpose of this Software User Manual is to describe and guide the users of the C++ language extension on how to use the product. The intent of this document is to make it as clear as possible what the user is able to do and how he or she can achieve his or her needs in an efficient way.

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 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.[1] 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.[2]

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.

1.4 How to use this document

Section 2 contains an overview of the processes that are supported. The fundamental principles of the process, what the software does to support the process, and what the user needs to supply to the software.
Section 3 contains a detailed tutorial for every task the user can perform; each tutorial includes a functional description, a list of precautions that may need to be taken, a step-by-step description of user actions and system responses, and, finally, a description of possible errors, if any.
Section 4 contains a summary of the operation references, including an explanation of what each operation achieves. Moreover, it refers to the other relevant documents that

specify the product's behaviour.
In the appendices, a list of all error messages is shown, which includes diagnosis and recovery procedures for each error, a glossary with an explanation of terms that are used throughout this document, and an index. The remainder of this document assumes that the reader is the user of the system.

1.5 Related documents

This document is related to the User Requirements Document (URD)[3]

1.6 Conventions

This document adheres to the following conventions:

- MPS is implied to refer to the JetBrains Meta Programming System.
- Every keyword and every command is typeset in a `typewriter` font.

1.7 Problem reporting

Software problems can be reported to Océ.[4] There is also the option of posting a question to the Mbeddr forum,[5] or the MPS forum.[6]

1.8 References

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

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

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

[4] Océ, 2018. *Océ*, info@oce.com

[5] Google, 2018. *Mbeddr forum (Google groups)*, Web. https://groups.google.com/forum/#!forum/mbeddr-discuss. Accessed 24 June 2018.

[6] JetBrains, 2018. *MPS forum*, Web. https://mps-support.jetbrains.com/hc/en-us/community/topics/200363779-MPS. Accessed 24 June 2018.

[7] JetBrains. (17 Jan. 2018). MPS User's Guide, MPS User Guide for Language Designers, [Online]. Available: https://confluence.jetbrains.com/display/MPSD20181/MPS+User%27s+Guide Accessed 12 June 2018.

2 Overview

This document describes several tutorials a user can follow to get a good understanding of the C++ extension language made by Eded. These tutorials are described in detail in Section 3, where we present the fundamental principles, the functions of the software, and the required actions of the user. The tutorials describe *classes*, *methods*, *attributes*, *enums*, *namespaces*, and *basic language syntax*.

3 Tutorials

3.1 Classes

3.1.1 Creating a Class

Functional description
This tutorial describes how to create and declare a class. This tutorial also explains how to create a instance of that class.
Cautions and Warnings

- Each class should have a unique name.

Preconditions

- A C++ implementation module should be created.

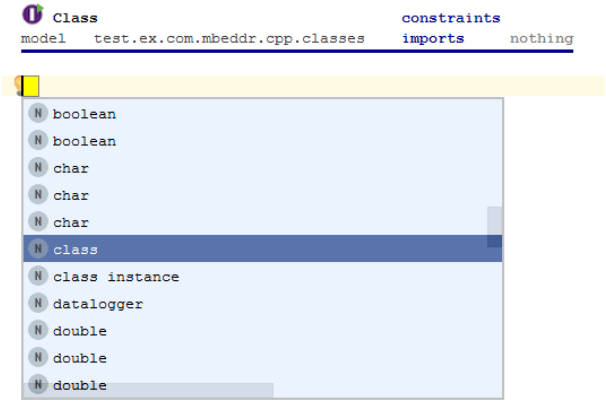
Procedure

User Action	MPS Platform Response
1. Put the cursor on a empty line within a C++ implementation module.	
2. Open the autocompletion menu.	
	3. Present a menu containing the option "Class" (Fig. 1a).
4. Select the option.	
	5. Create the class.
6. Give the class a name (Fig. 1b).	
7. Put the cursor on a empty line outside the created class.	
8. Type the name of the created class.	
	9. An instance of the created class is created.
10. Give the instance a name.	

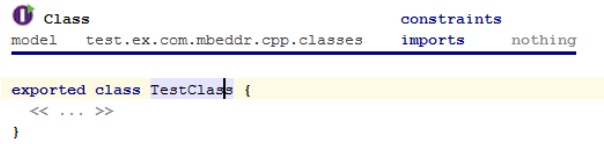
Likely Errors

- Name uniqueness error; occurs when the given name is already in use by another class.

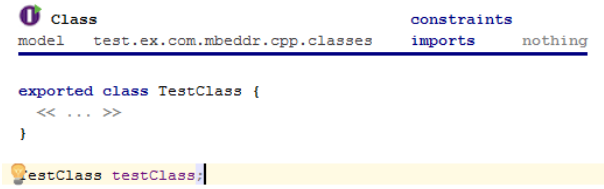
Figures



(a) Selecting class



(b) Class with name



(c) Final result

Figure 1: Creating a class

3.1.2 Inheritance

Functional description
This tutorial describes how to extend a class.
Cautions and Warnings

- Each class should have a unique name.
- The class that is extended should be in the correct scope.

Preconditions

- A C++ implementation module should be created.
- Two C++ classes should be created.

Procedure

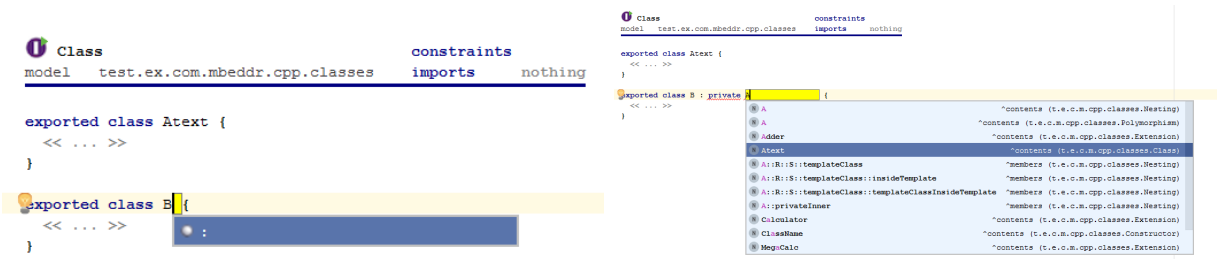
User Action	MPS Platform Response
1. Put the cursor after the name of the class in the class declaration within a C++ implementation module.	
2. Open the auto completion menu.	
	3. Present a menu containing the option ":" (Fig. 2a).
4. Select the option.	

	5. Create the inheritance text in the editor.
6. Open auto completion menu (Fig. 2b).	
	7. Present a menu containing all possible classes to extend.
8. Select the class to extend.	
	9. The class will extend the selected class.

Likely Errors

- Name uniqueness error; occurs when the given name is already in use by another class.
- A class cannot extend itself.
- You can't extend a class from another module if it is not exported.

Figures



(a) Extend class (b) Select class

Figure 2: Creating a class

3.1.3 Nesting

Functional description
This tutorial describes how to create nested classes and how to change the visibility of the nested class.
Cautions and Warnings

- Each class should have a unique name.

Preconditions

- A C++ implementation module should be created.
- A C++ class should be created.

Procedure

User Action	MPS Platform Response
1. Put the cursor on a empty line within a class.	
2. Open the auto completion menu.	
	3. Present a menu containing the option "Class" (Fig. 3a).
4. Select the option.	
	5. Create the class.
6. Give the class a name (Fig. 3b).	
7. Put the cursor on the access modifier of the created class.	
8. Delete the current access modifier.	
9. Open auto completion menu.	
	10. Present menu containing all possible access modifiers.
11 Select the access modifier.	

Likely Errors

- Name uniqueness error; occurs when the given name is already in use by another class.

Figures

(a) Create nested class

(b) Change access modifier

Figure 3: Creating a nested class

3.1.4 Creating a Constructor

Functional description

This tutorial describes how to create constructors on classes and how to instantiate a

class using a constructor.
Cautions and Warnings

- Each constructor should have a name identical to its direct parent class.

Preconditions

- A C++ implementation module should be created.
- A C++ class should be created.

Procedure

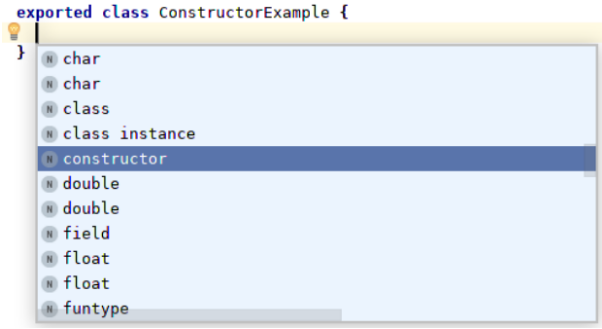
User Action	MPS Platform Response
1. Put the cursor on a empty line within a class.	
2. Open the auto completion menu.	
	3. Present a menu containing the option "constructor" (Fig. 4a).
4. Select the option.	
	5. Create the constructor with the correct name.
6. Place the cursor at the beginning of the arguments and press enter.	
7. Add arguments to the list separated by commas (Fig. 4b).	
8. Place the cursor at the beginning of the access modifier.	
9. Open auto completion menu.	
	10. Present menu containing all possible access modifiers (Fig. 4c).
11 Select the access modifier.	

Likely Errors None.
Figures

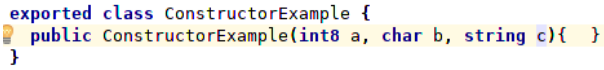
3.1.5 Calling a Parent Constructor

Functional description
This tutorial describes how to call constructors from classes in a child class.
Cautions and Warnings

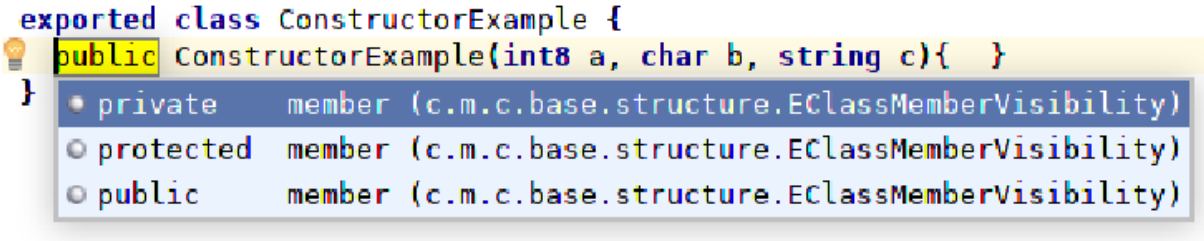
- The parent class's constructor should be visible to the child.



(a) Create constructor



(b) Add arguments



(c) Change visibility

Figure 4: Creating a constructor

Preconditions

- A C++ implementation module should be created.
- Two C++ classes should be created.
- One C++ class should inherit from the other.
- Both classes should have a constructor defined.

Procedure

User Action	MPS Platform Response
1. Put the cursor just after the closing parenthesis of the child constructor.	
2. Press colon (:).	
	3. Generate an empty constructor initializer (Fig. 5a).
4. Open the autocompletion menu.	
	5. Present a menu with the options for parent constructors (Fig. 'ref-fig:classParConstrb).
6. Select the constructor and place the cursor at the beginning of the arguments and press enter.	

7. Add the required arguments to the list separated by commas (Fig. 4c).	

Likely Errors

- Typesystem errors on arguments

Figures

```
exported class ConstructorExample {
  public ConstructorExample(<< ... >>){  }
  public ConstructorExample(int8 a, char b, string c){  }
}

exported class ConstructorExampleChild : public ConstructorExample {
  public ConstructorExampleChild(<< ... >>): {  }
}
```

(a) Create empty initializer

```
public ConstructorExample {
>): { }
  ConstructorExample()                          ^members (t.e.c.m.cpp.classes.Cosntructor)
  ConstructorExample(int8 a,char b,string c)    ^members (t.e.c.m.cpp.classes.Cosntructor)
```

(b) Create parent constructor call

```
exported class ConstructorExampleChild : public ConstructorExample {
  public ConstructorExampleChild(<< ... >>): ConstructorExample(8, 'c', "hello world!"){  }
}
```

(c) Parent constructor arguments

Figure 5: Calling a Parent Constructor

3.1.6 Calling a Constructor on a New Object

Functional description
This tutorial describes how to call constructors on instances of a class at creation.
Cautions and Warnings

- The class's constructor should be public.

Preconditions

- A C++ implementation module should be created.
- A C++ class should be created with a public constructor.
- One C++ class should inherit from the other.

Procedure

User Action	MPS Platform Response
1. Put the cursor on an empty line in a method. *2.* Type the name of the class. *3.* Open the autocompletion menu.	

4. Present a menu containing the available class types (Fig. 6a).

5. Choose the first option.

6. Give the variable a name.

7. Put the cursor inside the parenthesis.

8. Open the autocompletion menu.

9. Present a menu with the options for available constructors (Fig. 6b).

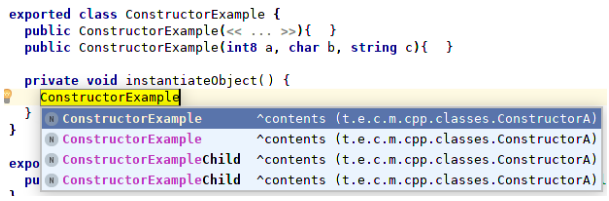
10. Select the constructor and place the cursor at the beginning of the arguments and press enter.

11. Add the required arguments to the list separated by commas (Fig. 6c).

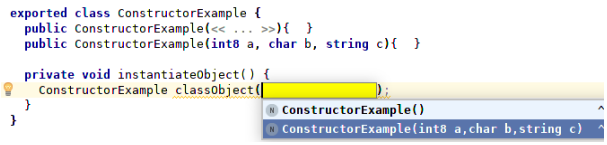
Likely Errors

- Typesystem errors on arguments

Figures



(a) Create class object



(b) Choose constructor

```
exported class ConstructorExample {
  public ConstructorExample(<< ... >>){  }
  public ConstructorExample(int8 a, char b, string c){  }

  private void instantiateObject() {
    ConstructorExample classObject(123, 'a', "String argument");
  }
}
```

(c) Class object arguments

Figure 6: Calling a Parent Constructor

3.2 Attributes

3.2.1 Declaring an attribute

Functional description
This tutorial describes how to create an attribute.

Cautions and warnings

- Each attribute in the same scope must have a unique name.

Preconditions

- The user must have created a valid class inside of a C++ module.

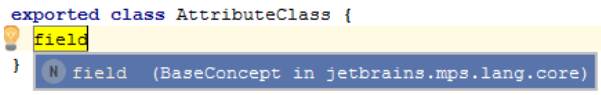
Procedure

User Action	MPS Platform Response
1. Put the cursor on an empty line within a class inside of a C++ module.	
2. Open the autocompletion menu.	
	3. Present a menu containing the option "field" (Fig. 7a)
4. Select the option.	
	5. Create the attribute.
6. (Optional:) Change the attribute's visibility from private to public or protected.	
7. Give the attribute a type and a name.	
8. Move cursor to end of the name.	
9. Type "=".	
	10. Create a field to specify an initial value.
11. Give the attribute an initial value based on its type. (Fig. 7b)	

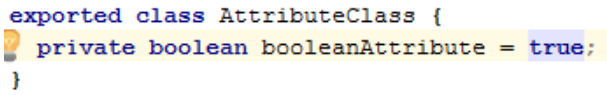
Likely Errors

- Name uniqueness error; occurs when the given name is already in use by another attribute.
- Initial value type error; occurs when the initial value does not have the same type as or a subtype of the attribute's type.

Figures



(a) Selecting field



(b) Final result

Figure 7: Creating an Attribute

3.2.2 Calling an attribute

Functional description
This tutorial describes how to call a previously created attribute.

Cautions and warnings

- Ensure that the attribute that you want to access is public, otherwise you will not be able to reference it.

Preconditions

- The user must have created a valid public attribute inside a valid class within a C++ module.

Procedure

User Action	MPS Platform Response
1. Put the cursor on an empty line within the C++ module, outside of the class of which you want to call an attribute.	
2. Open the autocompletion menu and type the name of the class of which you want to call a method.	
	3. Present a menu containing the class of which the name was entered.
4. Select the option. (Fig. 8a)	
	5. Create an instance of the given class.
6. Give the class instance a name.	
7. Press enter to go to the next empty line.	
	8. Create a new line below the class instance and select it.
9. Open the autocompletion menu and type the name of your class instance.	
	10. Present a menu containing the class instance.

11. Select the option. (Fig. 8b)

12. Press ".".

13. Present an entry field behind the ".".

14. Behind the ".", open the autocompletion menu and type the name of the attribute that you want to call.

15. Present a menu containing the attribute.

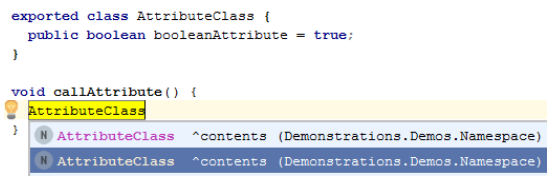
16. Select the option. (Fig. 8c)

17. Call the given attribute. (Fig. 8d)

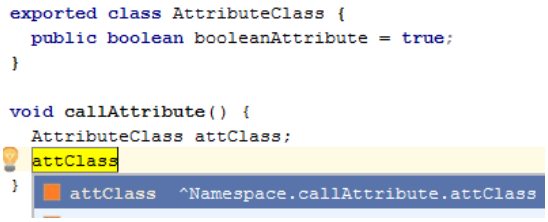
Likely Errors

- Visibility error; if the attribute that the user wants to call is not public, then it cannot be called.

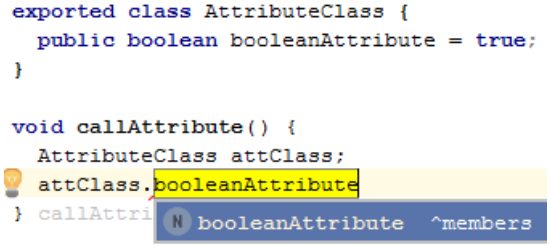
Figures



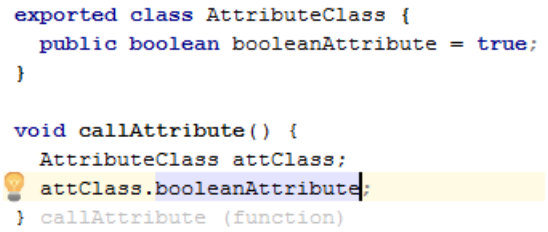
(a) Selecting class



(b) Selecting class instance



(c) Selecting attribute



(d) Final result

Figure 8: Calling an Attribute

3.3 Methods

3.3.1 Declaring a Method

Functional description
This tutorial describes how to create a method.

Cautions and warnings

- Each method in the same scope must have a unique name.

Preconditions

- The user must have created a valid class inside of a C++ module.

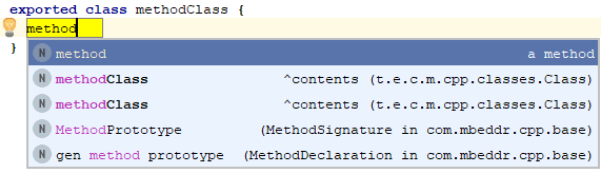
Procedure

User Action	MPS Platform Response
1. Put the cursor on an empty line within a class inside of a C++ module.	
2. Open the autocompletion menu.	
	3. Present a menu containing the option "method" (Fig. 9a)
4. Select the option.	
	5. Create the method.
6. (Optional:) Change the method's visibility from private to public or protected.	
7. Give the method a type.	
8. Give the method a name.	
9. Give the method an empty body by typing "{" at the end of the line.	
	10. Create the empty body. (Fig. 9b)
11. (Optional:) Add C++ code to the method's body.	

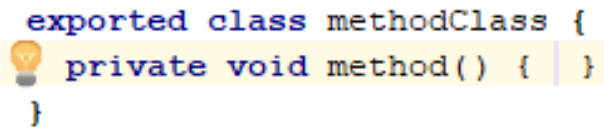
Likely Errors

- Name uniqueness error; occurs when the given name is already in use by another method with the same arguments.
- Return expected error; occurs when the given type is not void, and the body does not return anything.

Figures



(a) Selecting method



(b) Final result

Figure 9: Creating a Method

3.3.2 Calling a Method

Functional description

This tutorial describes how to call a previously created method.

Cautions and warnings

- Ensure that the method that you want to access is public, otherwise you will not be able to reference it.

Preconditions

- The user must have created a valid public method inside a valid class inside a C++ module.

Procedure

User Action	MPS Platform Response
1. Put the cursor on an empty line within the C++ module, outside of the class of which you want to call a method.	
2. Open the autocompletion menu and type the name of the class of which you want to call a method.	
	3. Present a menu containing the class of which the name was entered.
4. Select the option. (Fig. 10a)	
	5. Create an instance of the given class.
6. Give the class instance a name.	
7. Press enter to go to the next empty line.	
	8. Create a new line below the class instance and select it.
9. Open the autocompletion menu and type the name of your class instance.	
	10. Present a menu containing the class instance.

11. Select the option. (Fig. 10b)

12. Press ".".

13. Present an entry field behind the ".".

14. Behind the ".", open the autocompletion menu and type the name of the method that you want to call.

15. Present a menu containing the method.

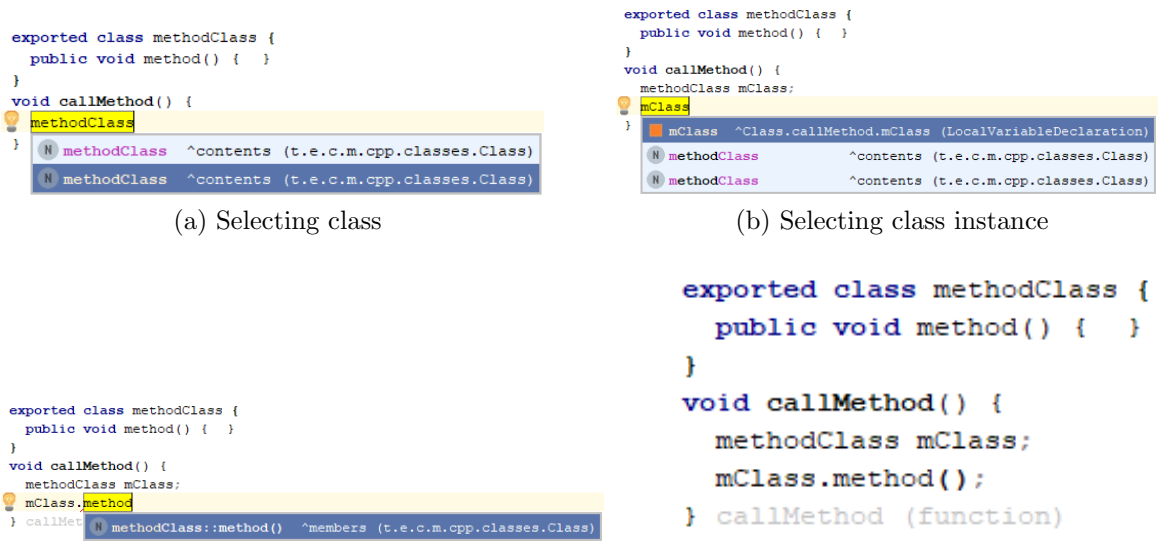
16. Select the option. (Fig. 10c)

17. Call the given method. (Fig. 10d)

Likely Errors

- Visibility error; if the method that the user wants to call is not public, then it cannot be called.

Figures



(a) Selecting class

(b) Selecting class instance

(c) Selecting method

(d) Final result

Figure 10: Calling a Method

3.4 Enums

3.4.1 Create an Enum

Functional description
This tutorial describes how to declare and create an enum.
Cautions and Warnings

Preconditions

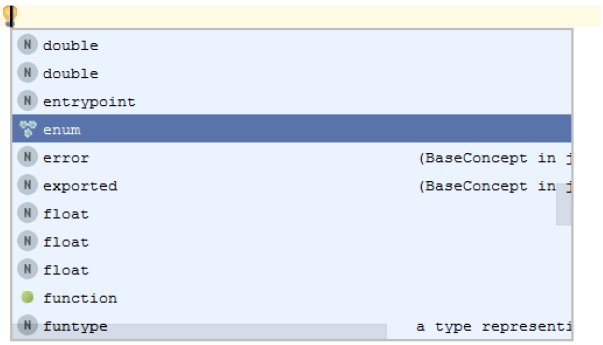
- A C++ implementation module should be created.

Procedure

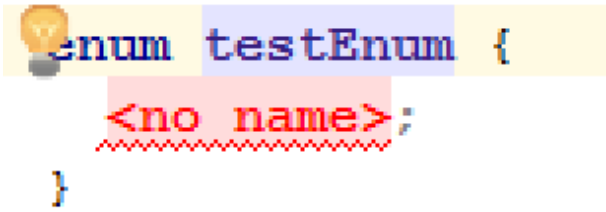
User Action	MPS Platform Response
1. Put the cursor on a empty line within a C++ module.	
2. Open the autocompletion menu	
	3. Present a menu containing the option "Enum" (Fig. 11a)
4. Select the option.	
	5. Create the Enum.
6. Give the Enum a name (Fig. 11b).	
7. Put the cursor on a empty line inside the enum.	
8. Type the name of on of the values of the enum.	

Likely Errors

Figures



(a) Create an enum



(b) Give the enum a name

Figure 11: Creating an Enum

3.5 Namespaces

3.5.1 Creating a Namespace

Functional Description
This tutorial describes how to create and declare a namespace element.

Cautions and Warnings

- Each namespace should have a unique name.

Preconditions

- A C++ implementation module should be created.

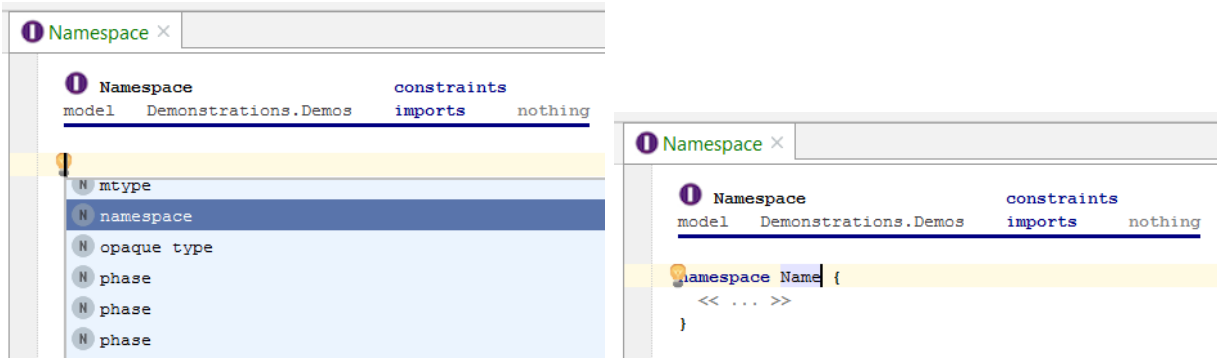
Procedure

User Action	MPS Platform Response
1. Put the cursor on a empty line within a C++ implementation module.	
2. Open the autocompletion menu.	
	3. Present a menu containing the option "namespace" (Fig. 12a).
4. Select the option.	
	5. Create the namespace.
6. Give the namespace a name. (Fig. 12b)	

Likely Errors

- Name uniqueness error; occurs when the given name is already in use by another namespace.

Figures



(a) Selecting namespace

(b) Final result

Figure 12: Creating a Namespace

3.5.2 Calling a Namespace Variable

Functional Description

This tutorial describes how to refer to a variable that is located within a namespace element.

Cautions and Warnings

- A previously declared namespace element should have a unique name.
- A previously declared variable within the namespace should have a unique name.

Preconditions

- A namespace element should already be declared.
- A variable should already be declared within the namespace.

Procedure

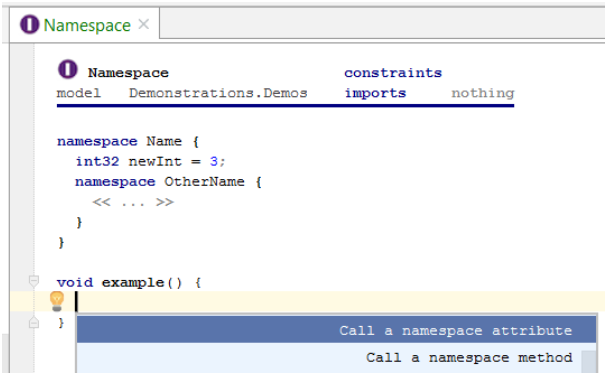
User Action	MPS Platform Response
1. Put the cursor on a empty line within a function within a C++ implementation module.	
2. Open the autocompletion menu.	
	3. Present a menu containing the options "Call a namespace attribute" and "Call a namespace method" (Fig. 13a).
4. Select either option.	
	5. Create the statement.
6. Put the cursor in the red area before '::'.	
7. Open the autocompletion menu.	

	8. Present a menu containing the declared namespaces. (Fig. 13b)
9. Select a namespace.	
	10. Place the namespace and its namespace ancestors in the red area.
11. Put the cursor in the red area after '::'	
12. Open the autocompletion menu.	
	13. Present a menu containing the declared variabled within the selected namespace. (Fig. 13c)
14. Select a variable.	
	15. Place the variable in the red area. (Fig. 13d)

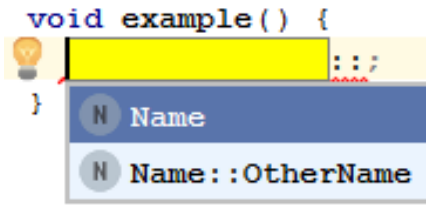
Likely Errors

- Could not select a namespace; occurs when no namespace is declared within the current model.
- Could not select a variable; occurs when no variable is declared within the selected namespace.
- Name uniqueness error; occurs when the selected namespace is already being used.

Figures

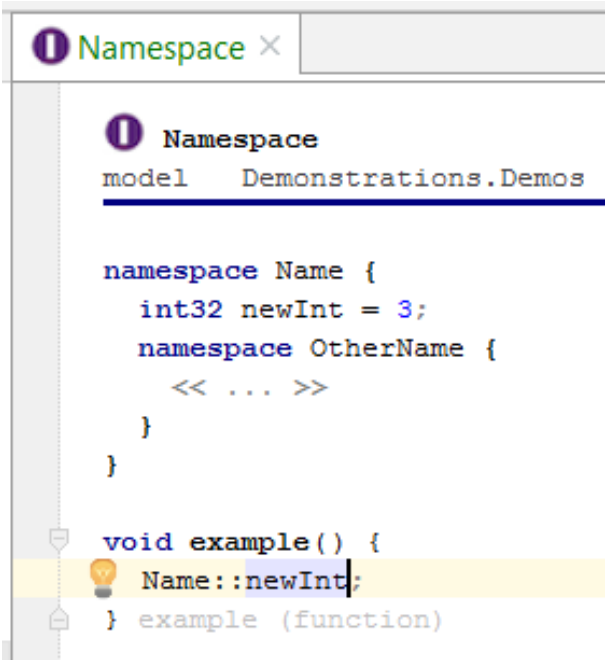


(a) Selecting either option



(b) Selecting a namespace

(c) Selecting a variable



(d) Final Result

Figure 13: Calling a Namespace variable

3.5.3 Using a Namespace

Functional Description

This tutorial describes how to use a namespace element.

Cautions and Warnings

- A previously declared namespace element should have a unique name.

Preconditions

A namespace element should already be declared according to section 3.5.1.

Procedure

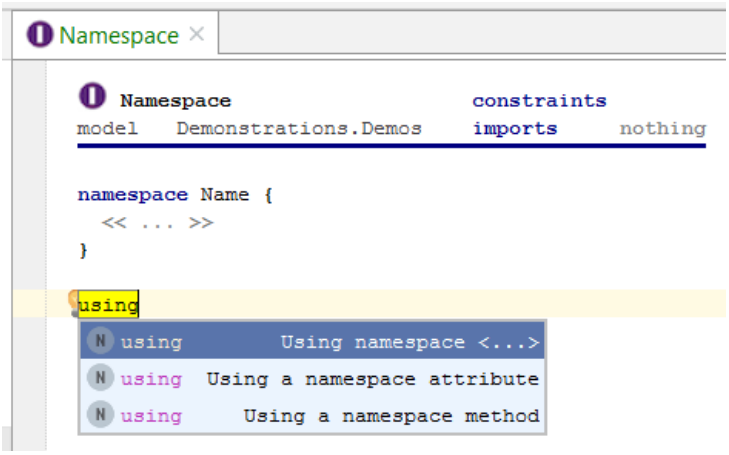
User Action	MPS Platform Response

1. Put the cursor on a empty line within a C++ implementation module.	
2. Open the autocompletion menu.	
	3. Present a menu containing the option "Using namespace ⟨...⟩" (Fig. 14a).
4. Select the option.	
	5. Create the 'using' statement.
6. Put the cursor in the red area after 'using namespace'.	
7. Open the autocompletion menu.	
	8. Present a menu containing the declared namespaces. (Fig. 14b)
9. Select a namespace.	
	10. Place the namespace and its namespace ancestors in the red area. (Fig. 14c)

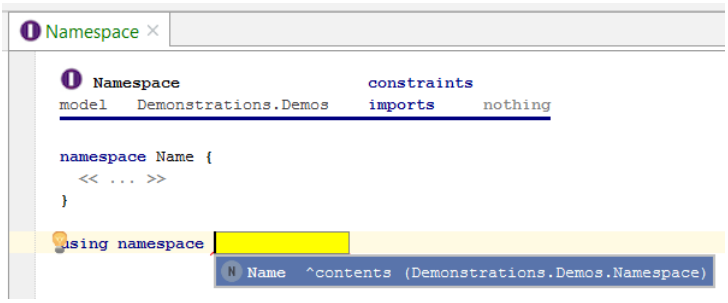
Likely Errors

- Could not select a namespace; occurs when no namespace is declared within the current model.
- Name uniqueness error; occurs when the selected namespace is already being used.

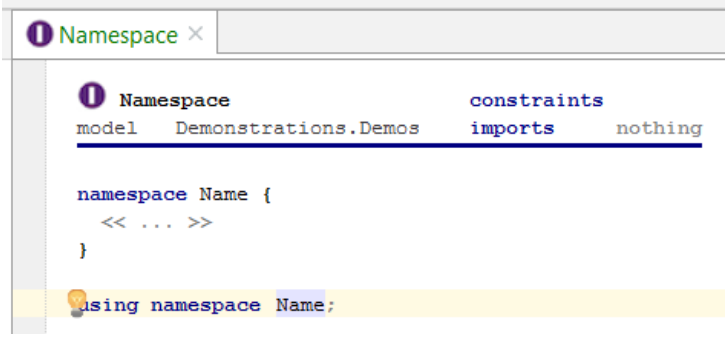
Figures



(a) Selecting the "Using namespace" option



(b) Selecting a namespace



(c) Final Result

Figure 14: Using a Namespace

3.5.4 Nesting Namespaces

Functional Description

This tutorial describes how to nest namespace element.

Cautions and Warnings

- Each namespace should have a unique name.

Preconditions

- A namespace element should already be declared according to section 3.5.1.

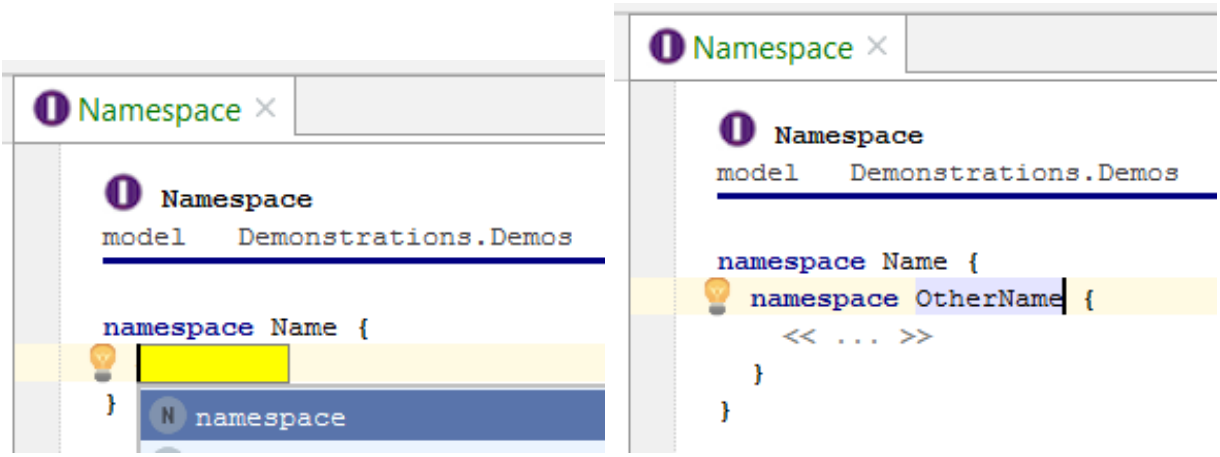
Procedure

User Action	MPS Platform Response
1. Put the cursor on an empty line within the declared namespace element within a C++ implementation module.	
2. Open the autocompletion menu.	
	3. Present a menu containing the option "namespace" (Fig. 15a).
4. Select the option.	
	5. Create the namespace.
6. Give the namespace a name. (Fig. 15b)	

Likely Errors

- Name uniqueness error; occurs when the given name is already in use by another namespace.

Figures



(a) Selecting Namespace (b) Final Result

Figure 15: Nesting Namespaces

3.6 Basic language syntax

3.6.1 Comments

Functional description
This tutorial describes how to use comments.
Cautions and Warnings
None.
Preconditions

- Create C++ Implementation Module

Procedure

User Action	MPS Platform Response
1. Put the cursor on an empty line in the C++ implementation module.	
2. Open the autocompletion menu.	
	3. Present a menu containing "//" (Fig. 17a).
4. Select the option.	
5. Type the text of the comment.	

Likely Errors None.
Figures

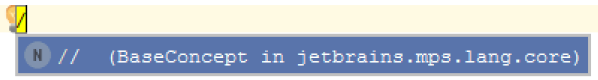


Figure 16: Adding comments

3.6.2 Expression

Functional description
This tutorial describes how to use expressions.
Cautions and Warnings
None.
Preconditions

- Create C++ Implementation Module

Procedure

User Action	MPS Platform Response
1. Put the cursor on an empty line in the C++ implementation module.	

2. Open the autocompletion menu.	
	3. Present a menu containing "int16" (Fig. 17a).
4. Select the option.	
5. Give the variable a name.	
6. Open the autocompletion menu.	
	7. Present a menu containing "=" (Fig. 17a).
8. Select the option "=".	
9. Type the value you want to give the variable.	

Likely Errors None.
Figures

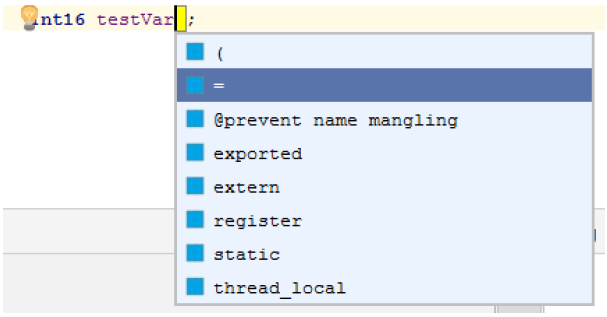


Figure 17: Initializing the variable

3.6.3 If-statement

Functional description
This tutorial describes how to create if-statements.
Cautions and Warnings
If-statement should have a non-empty condition.
Preconditions

- Create C++ Implementation Module
- Create a class (Section 3.1.1)
- Create a method inside the class (Section 3.3.1)

Procedure

User Action	MPS Platform Response
1. Put the cursor on an empty line in the method.	

2. Open the autocompletion menu.

3. Present a menu containing "if" (Fig. 18a).

4. Select the option.

5. Write a condition in the if-statement (Fig. 18b).

6. Write a body for the if-statement.

7. Put the cursor after the closing curly bracket of the if-statement.

8. Open the autocompletion menu.

9. Present a menu containing "else" (Fig. 18c).

10. Select the option "else".

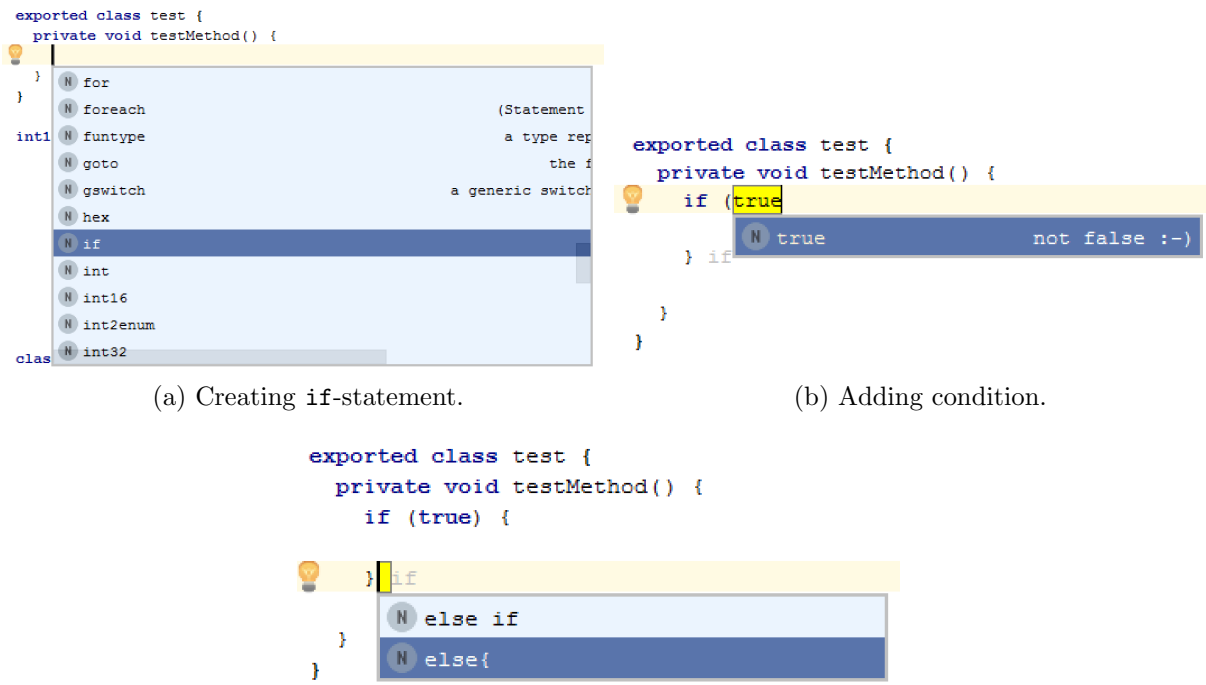
11. Create the else text.

12. Write a body for the else part.

Likely Errors

None.

Figures



(a) Creating `if`-statement.

(b) Adding condition.

(c) Adding `else` statement.

Figure 18: Creating `if`-statement

3.6.4 Switch statement

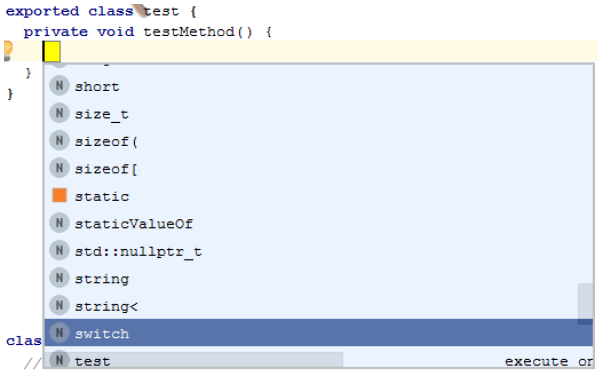
Functional description
This tutorial describes how to create `switch` statements.
Cautions and Warnings
Switch statement should have a non-empty expression.
Preconditions

- Create C++ Implementation Module
- Create a class (Section 3.1.1)
- Create a method inside the class (Section 3.3.1)

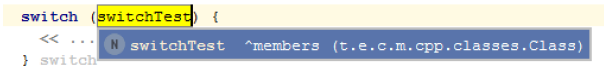
Procedure

User Action	MPS Platform Response
1. Put the cursor on an empty line in the method.	
2. Open the autocompletion menu.	
	3. Present a menu containing "switch". (Fig. 18a)
4. Select the option.	
5. Write a expression in the `switch` statement (Fig. 18b).	
6. Put the cursor on an empty line within the switch statement. *7.* Open the autocompletion menu.	
	8. Present a menu containing "case" (Fig. 18c).
9. Select the option "case".	
	10. Create the case text.
11. Write the condition for the case.	
12. Write a body for the case.	

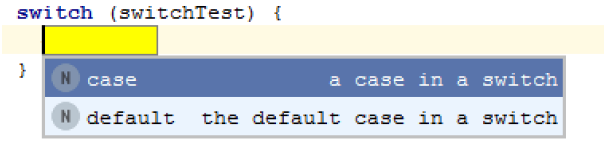
Likely Errors None.
Figures



(a) Creating `switch` statement.



(b) Adding expression.



(c) Adding case statement.

Figure 19: Creating `switch` statement

3.6.5 Labels and `goto`

Functional description
This tutorial describes how to use labels and goto.
Cautions and Warnings
None.
Preconditions

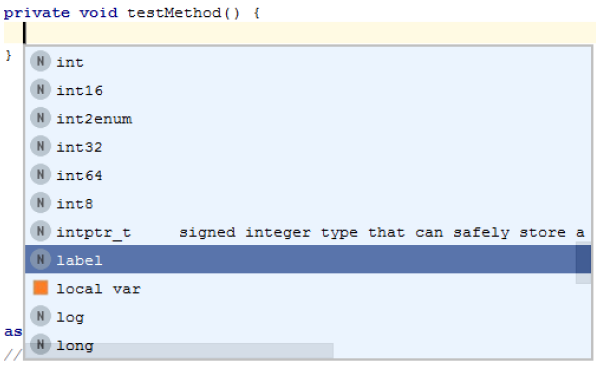
- Create C++ Implementation Module
- Create a class (Section 3.1.1)
- Create a method inside the class (Section 3.3.1)

Procedure

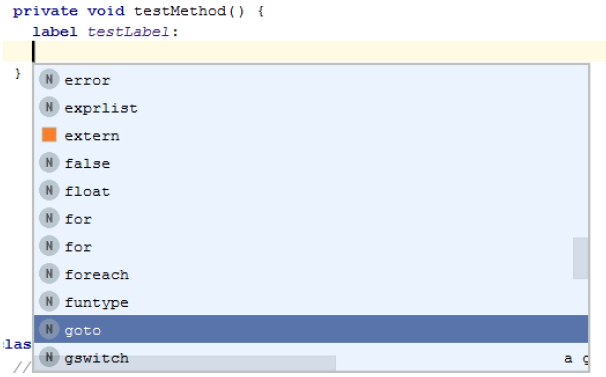
User Action	MPS Platform Response
1. Put the cursor on an empty line in the method	
2. Open the autocompletion menu.	
	3. Present a menu containing "label" (Fig. 20a).
4. Select the option.	
5. Give the label a name.	
6. Put the cursor on a different line then the label.	
7. Open the autocompletion menu.	

	8. Present a menu containing "goto" (Fig. 20b).
9. Select the option "goto".	*10.* Type the name of the label. (Fig. 20c)

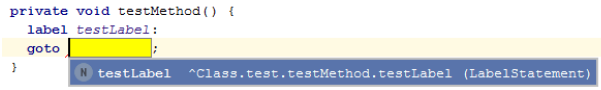
Likely Errors None.
Figures



(a) Creating `label`.



(b) Creating `goto`.



(c) Linking `goto` to `label`.

Figure 20: Using goto and label

3.6.6 Loop with break/continue

Functional description
This tutorial is about creating a loop and using break/continue inside it.
Cautions and Warnings

Preconditions

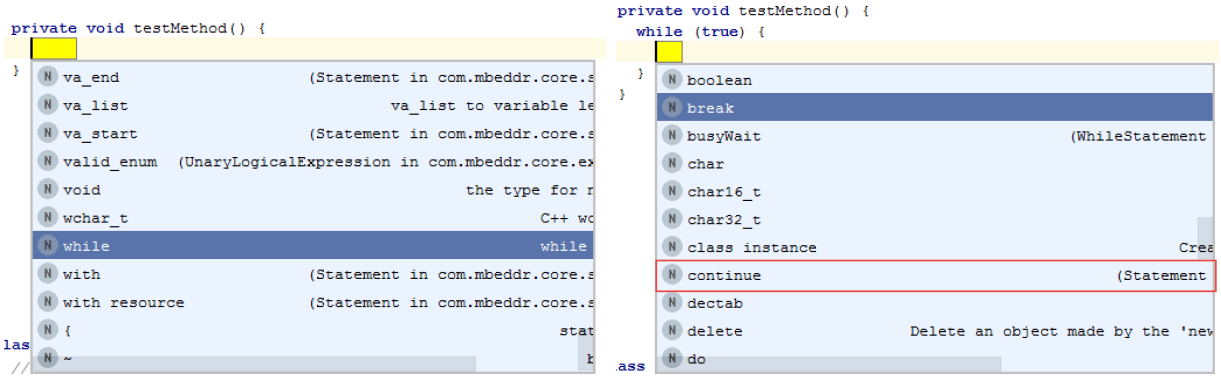
- Create C++ Implementation Module
- Create a class (Section 3.1.1)
- Create a method inside the class (Section 3.3.1)

Procedure

User Action	MPS Platform Response

1. Put the cursor on an empty line in the method.	
2. Open the autocompletion menu.	
	3. Present a menu containing "while" and "for" (Fig. 21a).
4. Select the "while" option.	
	5. Create the while loop.
6. Write the condition of the wile loop.	
6. Put the cursor within the while loop.	
7. Open the autocompletion menu.	
	8. Present a menu containing "break" and "continue" (Fig. 21b).
9. Select either "break" or continue.	

Likely Errors None.
Figures



(a) Creating a loop. (b) Creating continue.

Figure 21: Adding basic or continue

3.6.7 Try-catch

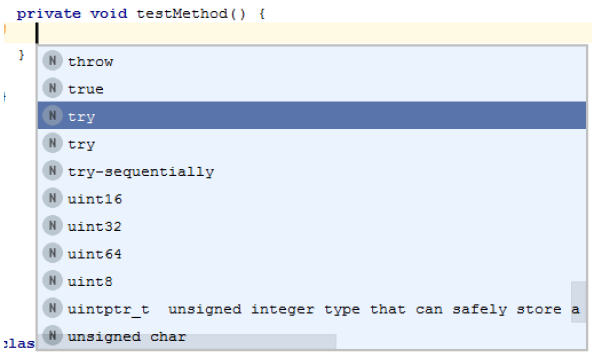
Functional description
This tutorial is about using try-catch.
Cautions and Warnings
Catch must be non-empty.
Preconditions

- Create C++ Implementation Module
- Create a class (Section 3.1.1)
- Create a method inside the class (Section 3.3.1)

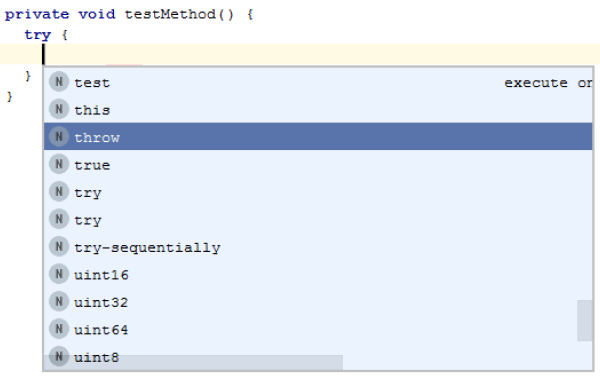
Procedure

User Action	MPS Platform Response
1. Put the cursor on an empty line in the method.	
2. Open the autocompletion menu.	
	3. Present a menu containing "try" (Fig. 22a).
4. Select the "try " option.	
	5. Create the try-catch block.
6. Put the cursor inside the try block.	
6. Open auto completion menu.	
	7. Present a menu containing "throw" (Fig. 22b).
8. Select the "throw" option.	
	9. Create the "throw" statement.
10. Give the throw statement a value.	
11. Put the cursor inside the catch-block.	
12. Open the auto completion menu.	
	13. Present a menu containing "..." (Fig. 22c). *14.* Select the "..." option.

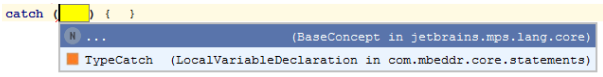
Likely Errors None.
Figures



(a) Creating a `try-catch` block.



(b) Creating `throw` statement.



(c) Adding basic handler in catch block.

Figure 22: Creating `try-catch` block

3.6.8 Union

Functional description
This tutorial is about creating Unions.
Cautions and Warnings
Union must have a name.
Preconditions

- Create C++ Implementation Module

Procedure

User Action	MPS Platform Response
1. Put the cursor on an empty line in the C++ implementation module.	
2. Open the autocompletion menu.	
	3. Present a menu containing "union" (Fig. 23a).
4. Select the "union" option.	
	5. Create the union block.
6. Give the union a name.	

Likely Errors
Property Constraint violation for property "name". (Union doesn't have a name)
Figures

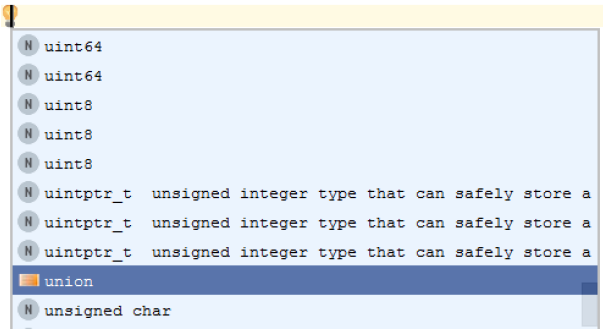


Figure 23: Creating an Union

3.6.9 Arrays

Functional description
This tutorial is about creating arrays.
Cautions and Warnings

Preconditions

- Create C++ Implementation Module

- Create a variable

Procedure

User Action	MPS Platform Response
1. Put the cursor at the end of a type of a variable.	
2. Open the autocompletion menu.	
	3. Present a menu containing the option "[". (Fig. 24a)
4. Select the option "[".	
	5 Make the variable an array.

Likely Errors
None.
Figures

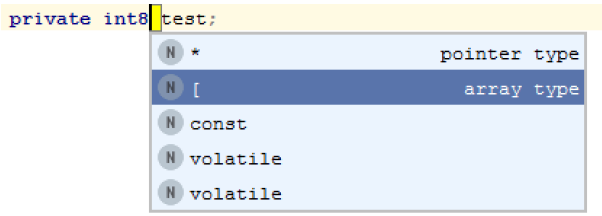


Figure 24: Creating an Array

3.6.10 Pointer

Functional description
This tutorial is about using pointers.
Cautions and Warnings

Preconditions
A C++ module should be created.
A variable should be created.
Procedure

User Action	MPS Platform Response
1. Put the cursor on a empty line within the scope of the variable.	
2. Open the autocompletion menu	
	3. Present a menu containing the type of the created variable. (Fig. 24a)
4. Select the option of that type.	

	5 Create a new variable.
6. Put the cursor on the end of the type.	
7. Open auto completion menu.	
	8. Present a menu containing the option "* ".
9. Select the "*" option.	
	10. Change the variable to a pointer
11. Give the Pointer a name.	

Likely Errors
None.
Figures

(a) Making a variable. (b) Changing the variable to a pointer.

Figure 25: Creating an Array

3.6.11 Casting

Functional description
This tutorial is about casting
Cautions and Warnings
Not possible to cast to the wanted type.
Preconditions

- Create C++ Implementation Module

Procedure

User Action	MPS Platform Response
1. Put the cursor on a empty line within the C++ implementation module.	
2. Open the autocompletion menu.	

3. Present a menu containing all the types possible for a variable.

4. Select a type.

5 Create a new variable with the specified type.

6. Put the cursor on the end of the type.

7. Give the variable a name.

8. Put the cursor at the end of the name of the variable.

9. Open auto completion menu.

10. Present a menu containing the option "=".

11. Select the "=" option.

12. Give the variable an empty initialization.

13. Open the auto completion menu.

14. Present a menu containing the "(" option. (Fig. 26a)

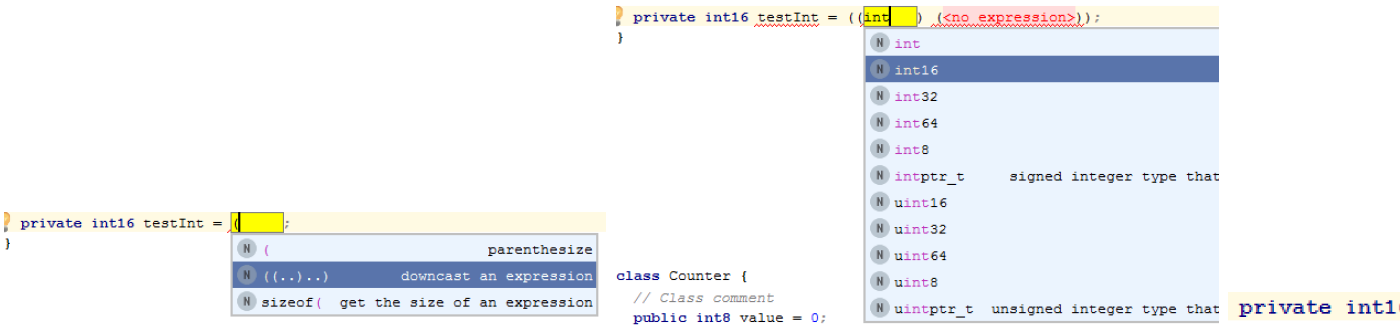
15. Select the "(" option.

16. Create an empty downcast expression.

17. Type the selected type between the brackets.

18. Add an expression after the casting.

Likely Errors
None.
Figures



(a) Adding casting. (b) Adding type to casting. (c)

Figure 26: Creating an Array

3.6.12 Unary prefix operation

Functional description
This tutorial describes how to use unary prefix operators (++, –, -, ~)
Cautions and Warnings
Variable may not be initialized.
Preconditions

- Create C++ Implementation Module
- Create a class (Section 3.1.1)
- Create a method inside the class (Section 3.3.1)
- A variable of type int should be created and should be able to use in the method.

Procedure

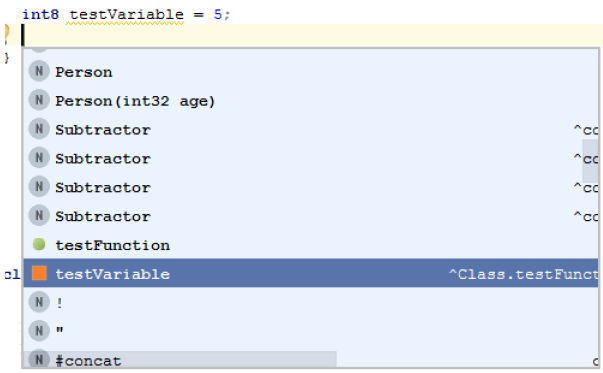
User Action	MPS Platform Response
1. Put the cursor on a empty line within the method.	
2. Open the auto completion menu.	
	3. Present a menu containing the variable. (Fig. 27a)
4. Select the variable.	
5. Put the cursor on the end of the variable.	
6. Open the auto completion menu.	
	7. Present a menu containing all the unary prefix operators (Fig. 27b).
8. Select one of the unary prefix operators.	

Likely Errors
None.
Figures

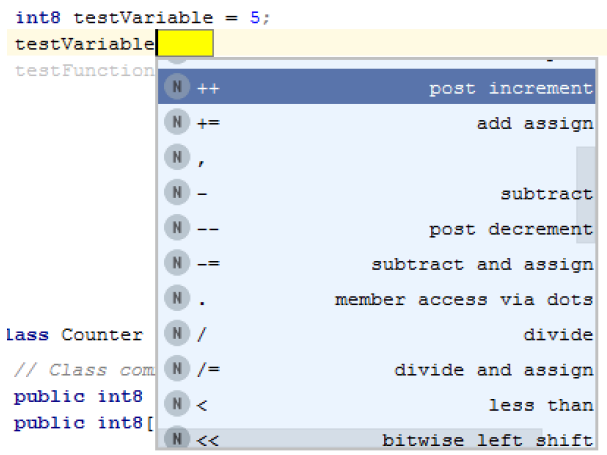
3.6.13 Unary negation operation

Functional description
This tutorial describes how to use unary negation operator
Cautions and Warnings
Variable may not be initialized.
Preconditions

- Create C++ Implementation Module
- Create a class (Section 3.1.1)



(a) Using created variable.



(b) Adding prefix operator.

Figure 27: Adding unary prefix operator

- Create a method inside the class (Section 3.3.1)

Procedure

User Action	MPS Platform Response
1. Put the cursor on a empty line within the method.	
2. Open the auto completion menu	
	3. Present a menu containing the type boolean.
4. Select the boolean type.	
	5. Create a variable of type boolean.
6. Give the variable a name.	
7. Put the cursor on the end of the variable.	
8. Open the auto completion menu.	
	9. Present a menu containing the "=".
10 Select the "=" option.	
11. Open the auto completion menu.	
	12. Present a menu containing the "true" option.
13. Select the "true" option.	
14. Put the cursor in front of the initialization.	
15. Open the auto completion menu.	
	16. Present a menu containing the "!" option (Fig. 28a).

17. Select the "!" option.	

Likely Errors
None.
Figures

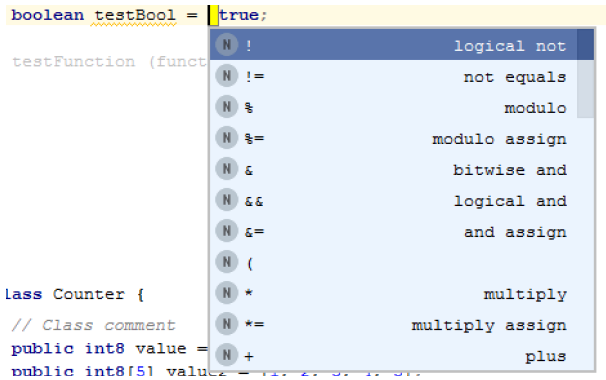


Figure 28: Adding unary prefix operator

3.6.14 Unary sizeof operator

Functional description
This tutorial describes how to use unary sizeof operator
Cautions and Warnings

Preconditions

- Create C++ Implementation Module
- Create a class (Section 3.1.1)
- Create a method inside the class (Section 3.3.1)

Procedure

User Action	MPS Platform Response
1. Put the cursor on a empty line within the Method.	
2. Open the auto completion menu	
	3. Present a menu containing the type unsigned int.
4. Select the unsigned int type.	
	5. Create a variable of type unsigned int.
6. Give the variable a name.	

7. Put the cursor on the end of the variable.	
8. Open the auto completion menu.	
	9. Present a menu containing the "=".
10 Select the "=" option.	
11. Open the auto completion menu.	
	12. Present a menu containing the "size of" option (Fig. 31a).
13. Select the "size of" option.	
14. Type a number in the sizeof expression.	

Likely Errors
None.
Figures

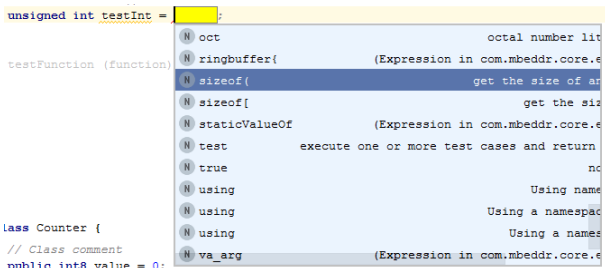


Figure 29: Adding unary sizeof operator

3.6.15 Binary comparison operator

Functional description
This tutorial describes how to use binary comparison operators (==, !=, ¿ , ¡, ¿=, ¡=, &&, ——)
Cautions and Warnings
Items of the left and right of the operator should be of the same type.
Preconditions

- Create C++ Implementation Module
- Create a class (Section 3.1.1)
- Create a method inside the class (Section 3.3.1)
- Create two variables of the same type.

Procedure

User Action	MPS Platform Response

1. Put the cursor on a empty line within the method.	
2. Open the auto completion menu	
	3. Present a menu containing the "if" option.
4. Select the "if" option.	
	5. Create a "if" block.
6. Put the cursor inside the condition.	
7. Open the auto completion menu	
	8. Present a menu containing the variables.
9. Select the first variable.	
10. Put the cursor on the end of the variable.	
11. Open the auto completion menu.	
	12. Present a menu containing all the comparison operators.
13 Select a comparison operator.	
14. Open the auto completion menu	
	15. Present a menu containing the variables.
16. Select the second variable.	

Likely Errors
None.
Figures

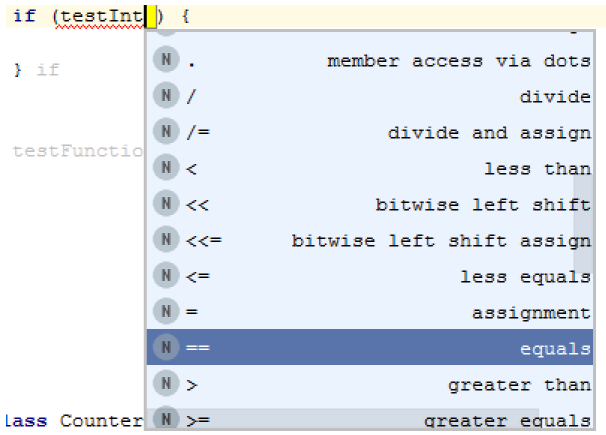


Figure 30: Adding binary comparison operator

3.6.16 Binary operator

Functional description
This tutorial describes how to use binary operators (+, -, /, *, %)
Cautions and Warnings
Items of the left and right of the operator should be of the same type.
Preconditions

- Create C++ Implementation Module
- Create a class (Section 3.1.1)
- Create a method inside the class (Section 3.3.1)

Procedure

User Action	MPS Platform Response
1. Put the cursor on a empty line within the method.	
2. Open the auto completion menu	
	3. Present a menu containing the type "int16".
4. Select the "int16" type.	
	5. Create a variable of type "int16".
6. Give the variable a name.	
7. Put the cursor on the end of the variable.	
8. Open the auto completion menu.	
	9. Present a menu containing the "=".
10 Select the "=" option.	
11. type a integer number.	
12. Open the auto completion menu.	
	12. Present a menu containing the binary operators (Fig. **??**).
13. Select a binary operator.	
14. type an integer number after the binary operator	

Likely Errors
None.
Figures

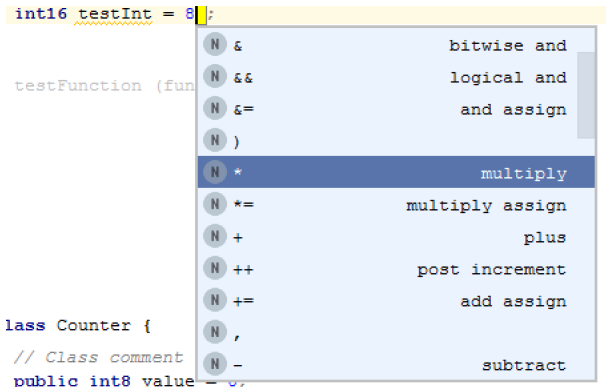


Figure 31: Adding binary operator

3.6.17 Binary assignment operator

Functional description
This tutorial describes how to use binary assignment operators (+=, -=, *=, /=, %=)
Cautions and Warnings
Items of the left and right of the operator should be of the same type.
Preconditions

- Create C++ Implementation Module
- Create a class (Section 3.1.1)
- Create a method inside the class (Section 3.3.1)
- Create a variable of type int.

Procedure

User Action	MPS Platform Response
1. Put the cursor on a empty line within the method.	
2. Open the auto completion menu.	
	3. Present a menu containing the variable.
4. Select the variable option.	
6. Put the cursor after the variable.	
7. Open the auto completion menu.	
	8. Present a menu containing the assignment operators.32)
9. Select an assignment operator.	
10. Type an integer number.	

Likely Errors

None.
Figures

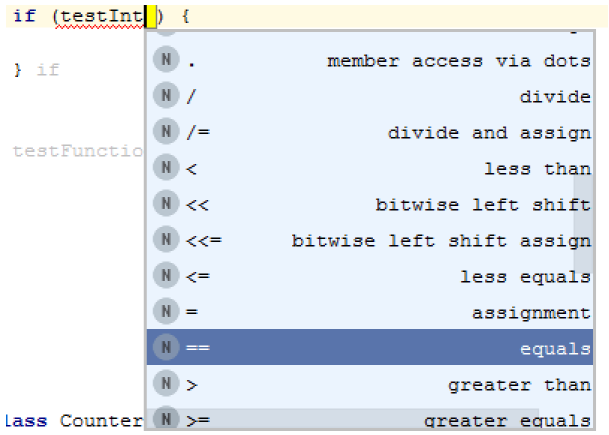


Figure 32: Adding binary assignment operator

3.6.18 Include header

Functional description
This tutorial describes how to include headers.
Cautions and Warnings
None.
Preconditions

- Create C++ Implementation Module

Procedure

User Action	MPS Platform Response
1. Put the cursor inside the imports of the implementation module.	
2. Open the auto completion menu.	
	3. Present a menu containing the "header" option. 33)
4. Select the "header" option.	
6. Type the name of the header inside the header.	

Likely Errors
None.
Figures

Figure 33: adding a header to the imports

3.6.19 Macros

Functional description
This tutorial describes how to use macros.
Cautions and Warnings
None.
Preconditions

- Create C++ Implementation Module

Procedure

User Action	MPS Platform Response
1. Put the cursor on a empty line within the implementation module.	
2. Open the auto completion menu.	
	3. Present a menu containing the "#macro" option.
4. Select the "#macro" option.	
	5. Create a macro.
6. Give the macro a name.	
7. Add a statement to the content of the macro.	

Likely Errors
None.
Figures

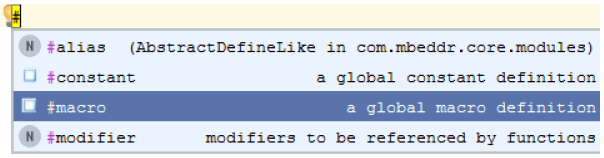


Figure 34: Creating a macro

3.6.20 General specifiers

Functional description
This tutorial describes how to use general specifiers (`static`, `volatile`, `inline` and `const`) on attributes and methods .
Cautions and Warnings
None.
Preconditions

- Create C++ Implementation Module
- Create a class (Section 3.1.1)
- Create a method inside the class (Section 3.3.1)

Procedure

User Action	MPS Platform Response
1. Put the cursor on a line with a attribute or method within a C++ module.	
2. Open the autocompletion menu.	
	3. Present a menu containing all the general specifiers possible at that position. (Fig. 35a)
4. Select the specifier.	

Likely Errors

- Static members can't be virtual
- Static members can't be volatile
- Static members can't be constant

Figures

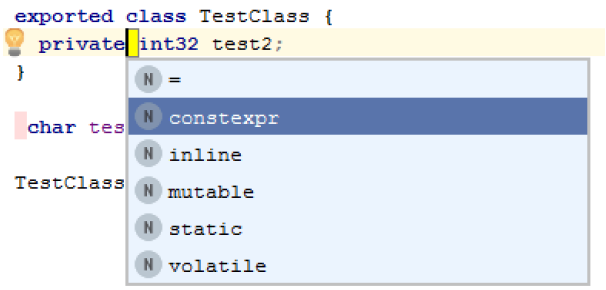


Figure 35: Adding a general specifier

3.6.21 `Virtual / Pure Virtual`

Functional description
This tutorial describes how to make methods (pure) virtual.
Cautions and Warnings
None.
Preconditions

- Create C++ Implementation Module
- Create a class (Section 3.1.1)
- Create a method inside the class (Section 3.3.1)

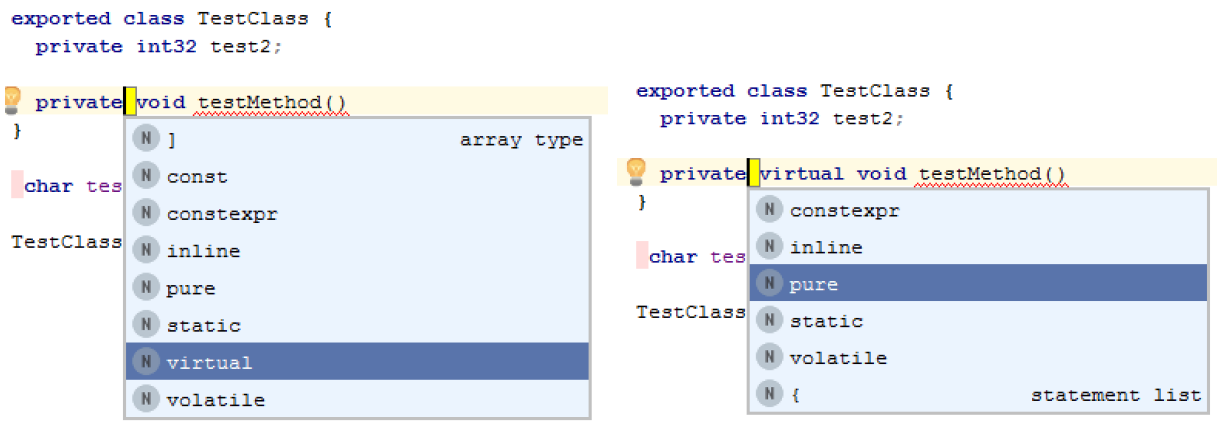
Procedure

User Action	MPS Platform Response
1. Put the cursor on a line with a method, before the type, within a C++ module.	
2. Open the autocompletion menu.	
	3. Present a menu containing the virtual keyword. (Fig. **??**a)
4. Select the virtual keyword	
5. Put cursor before the virtual keyword.	
6. Open auto completion menu.	
	7. Present menu containing the pure keyword. (Fig. **??**b)
8. Select the pure keyword.	

Likely Errors

- Non-pure virtual method must have a body.

Figures



(a) Adding virtual keyword.

(b) Adding pure keyword.

Figure 36: Adding (pure) virtual keyword

3.6.22 Explicit

Functional description
This tutorial describes how to use the explicit keyword.
Cautions and Warnings

Preconditions

- Create C++ Implementation Module
- Create a class (Section 3.1.1)
- Create a constructor on the class

Procedure

User Action	MPS Platform Response
1. Put the cursor in front of a line with a constructor declaration.	
2. Open the autocompletion menu.	
	3. Present a menu containing the option explicit. (Fig. 37)
4. Select the option.	
	5. Explicit is added to the constructor.

Likely Errors

- Variable declaration with auto type must be initialized.
- Auto type can only be used on static and const attributes.

Figures

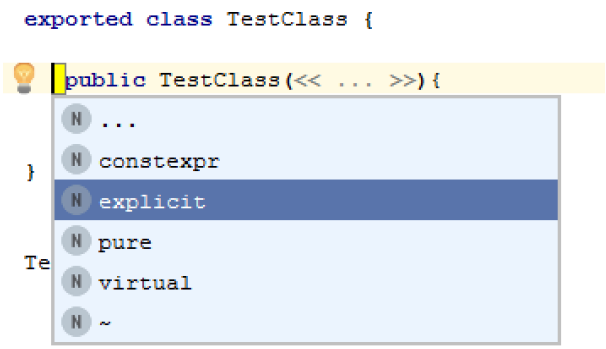


Figure 37: Adding explicit keyword

3.6.23 `extern`

Functional description
This tutorial describes how to use the extern keyword.
Cautions and Warnings

Preconditions

- Create C++ Implementation Module
- Create an attribute (Section 3.2.1)

Procedure

User Action	MPS Platform Response
1. Put the cursor on the front of the line attribute declaration.	
2. Open the autocompletion menu	
	3. Present a menu containing the `extern` option. (Fig. 38)
4. Select the `extern` option	
	5. `extern` added to the attribute.

Likely Errors None.
Figures

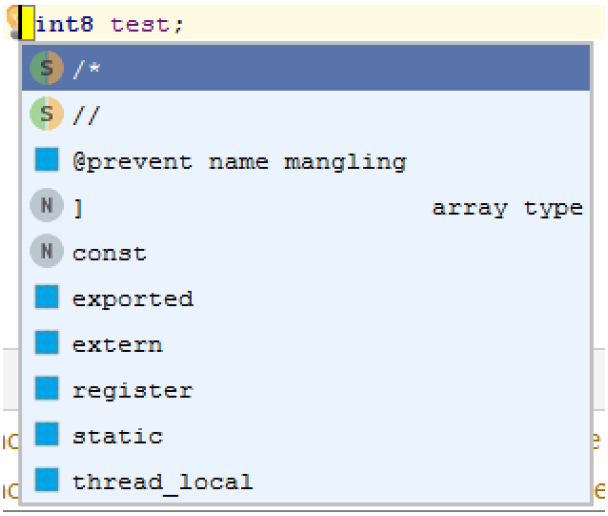


Figure 38: Adding extern keyword.

3.6.24 `thread_local`

Functional description
This tutorial describes how to use the `thread_local` keyword.
Cautions and Warnings

Preconditions

- Create C++ Implementation Module
- Create an attribute (Section 3.2.1)

Procedure

User Action	MPS Platform Response
1. Put the cursor on the front of the line attribute declaration.	
2. Open the autocompletion menu.	
	3. Present a menu containing the `thread_local` option. (Fig. 39)
4. Select the `thread_local` option.	
	5. `thread_local` added to the attribute.

Likely Errors None.
Figures

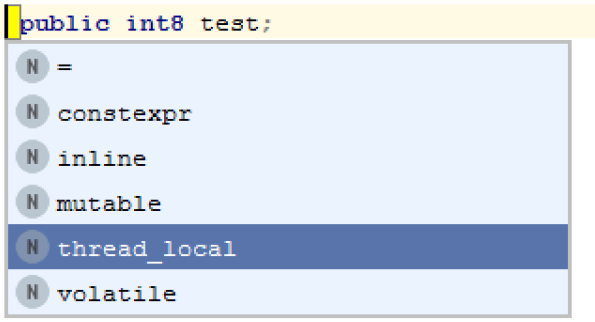


Figure 39: Adding thread_local keyword.

3.6.25 `nullpointer`

Functional description
This tutorial describes how to use the nullpointer keyword.
Cautions and Warnings

Preconditions

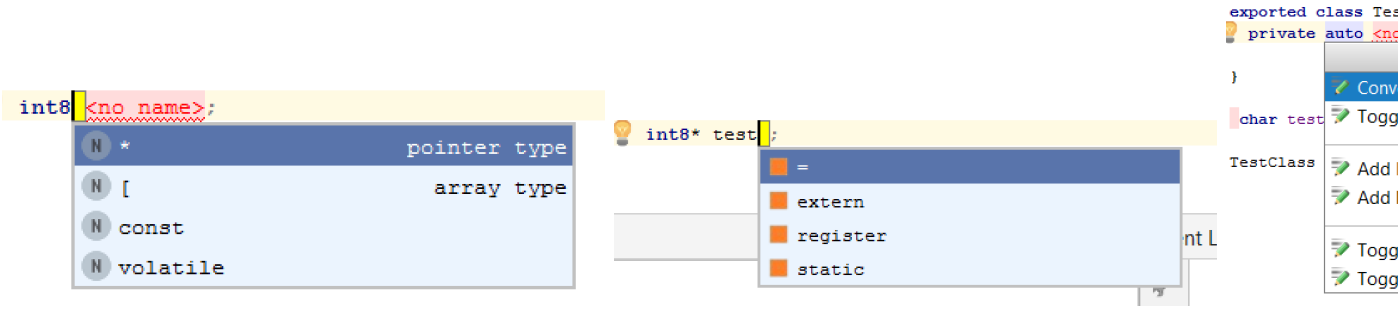
- Create C++ Implementation Module

Procedure

User Action	MPS Platform Response
1. Put the cursor on a empty line.	
2. Open the autocompletion menu.	
	3. Present a menu containing all the possible types.
4. Select a type.	
	5. Create the variable.
6. Put the cursor after type of the variable.	
7. Open auto completion menu.	
	8. Present menu containing the pointer option. (Fig. 36b)
9. Select the pointer option.	
10. Give the variable a name.	
	11. Present menu containing the option =. (Fig. 36b)
12. Select option =.	
14. Open auto completion menu.	

	15. Present menu containing the option `nullptr`. (Fig. 40b)
16. Select the `nullptr` option.	

Likely Errors None.
Figures



(a) Making the variable a pointer. (b) Adding =. (c

Figure 40: Adding nullpointer keyword

3.6.26 `auto`

Functional description
This tutorial describes how to use the auto keyword.
Cautions and Warnings
Variable with type auto must be const and static
Preconditions

- Create C++ Implementation Module

Procedure

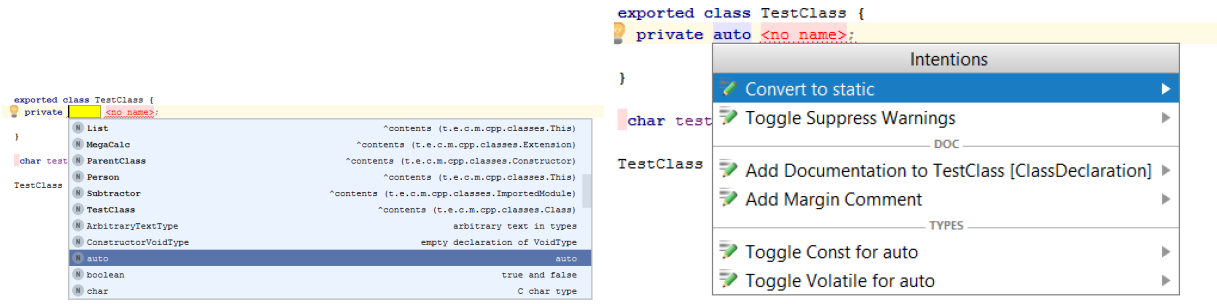
User Action	MPS Platform Response
1. Put the cursor on a empty line.	
2. Open the autocompletion menu.	
	3. Present a menu containing the option field.
4. Select the option.	
5. Put cursor type of the field.	
6. Open auto completion menu.	
	7. Present menu containing the auto keyword. (Fig. 41a)
8. Select the auto keyword.	
9. Give the variable a name.	

10. Initialize the variable	
11. open intentions menu.	
	12. Present the intentions menu containing the option to make the variable static. (Fig. 41b)
13. Select `static` option.	
14. Make variable `const` (see 3.6.20).	

Likely Errors

- Variable declaration with auto type must be initialized.
- Auto type can only be used on static and const attributes.

Figures



(a) Making attribute of type auto. (b) Making it static.

Figure 41: Adding auto keyword

3.6.27 `this`

Functional description
This tutorial describes how to use the this keyword.
Cautions and Warnings

Preconditions A C++ module should be created.
A C++ class should be created.
A C++ variable should be created.
A C++ method with the same type as the variable should be created.
Procedure

User Action	MPS Platform Response
1. Put the cursor on a empty line within the class.	
2. Open the auto completion menu	

3. Present a menu containing the option return.

4. Select the option return.

5. Open auto completion menu.

6. Present menu containing the `this` keyword. (Fig. 42a)

7. Select the `this` keyword.

8. open auto completion menu

9. Present menu containing the dot expression (Fig. 42b).

10. Select the dot expression.

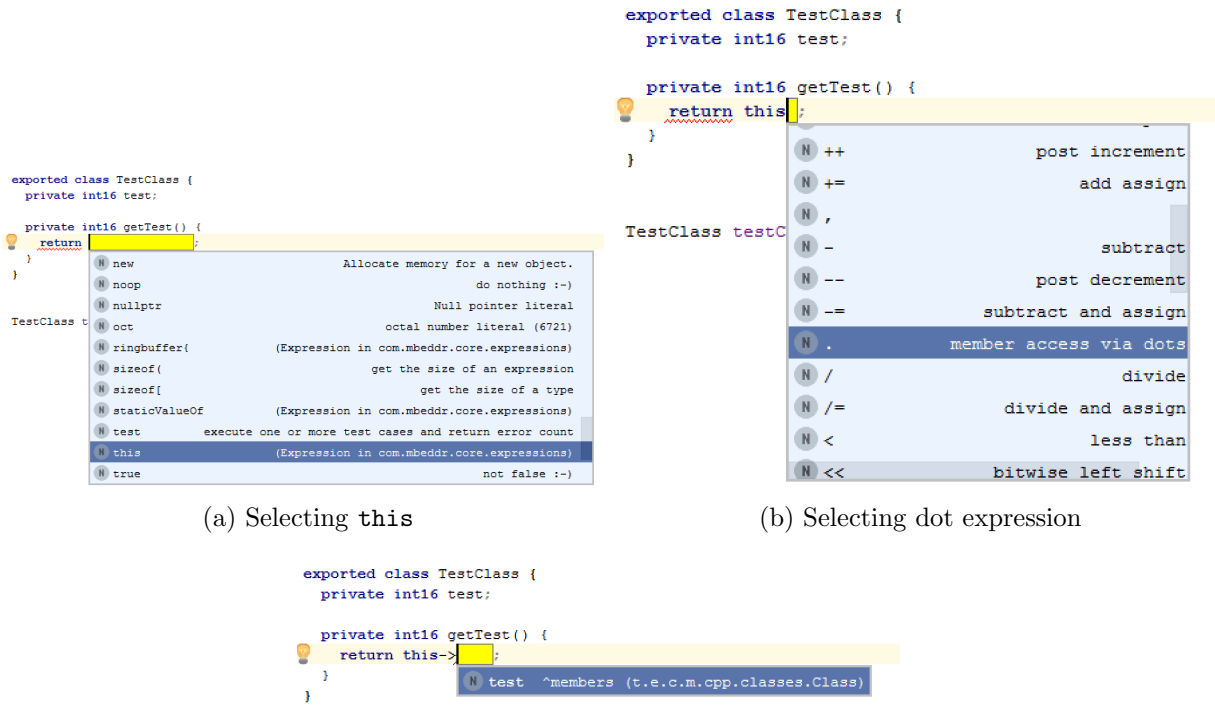
11. Open auto completion menu.

12. Present menu containing the variable of the class (Fig. 42c).

13. Select the variable.

Likely Errors None.

Figures



(a) Selecting `this`

(b) Selecting dot expression

(c) Selecting variable

Figure 42: Using `this` keyword

4 Reference

As Mbeddr is a plug-in for MPS, and our product is an extension of Mbeddr, you will mostly use the MPS user interface to interact with our product. How to use the user interface of MPS is generally described in the MPS User Guide.[7]
The tutorials described in section 3 use specific parts of the MPS user interface to perform several operations. As such, the next section describes only those operations.

4.1 MPS Editor

In this section, a description of the operations performed on the MPS user interface and described in section 3 are shown.

4.1.1 Creating a new project

To create a new project in the mbeddr IDE perform the following steps:

1. Click on File
2. Click on new
3. Click on project

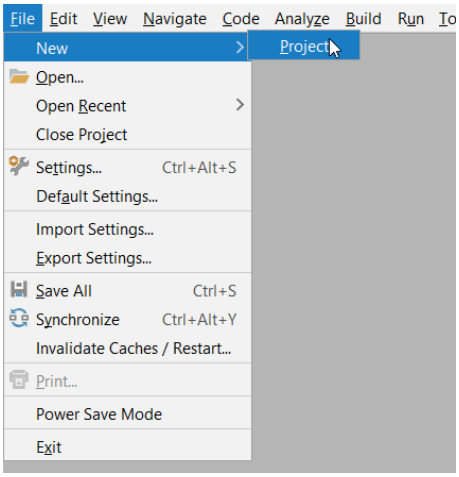


Figure 43: Creating a new project

A new window will now pop up. In the window a project name and solution name should be provided.

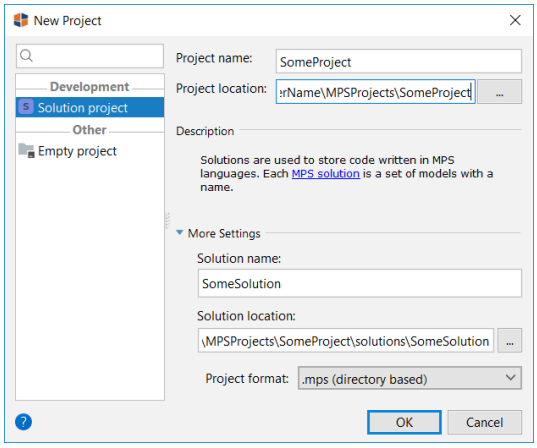


Figure 44: Creating a new solution

Click ok to create the project.

4.1.2 Autocompletion Menu

Functional Description
Using the autocompletion menu, you can comfortably insert C++ elements in a C++ module.

Cautions and Warnings
To use the autocompletion menu, a project containing a solution with a C++ module is required. The steps for the operations to create these environments are described in section 4.1.1.
Formal Description

Operation	Steps	Result
Open the autocompletion menu.	Hold the CTRL-key and press the spacebar.	The autocompletion menu is opened.
Select an option within the autocompletion menu.	Double-click on the desired option.	The option is selected and the associated element will be created.

Examples

- In the tutorial for creating a class (section 3.1.1) the autocompletion menu is used to select the "Class" option, allowing the user to easily create a new class.
- In the tutorial for creating a namespace (section 3.5.1) the autocompletion menu is used to select the "Namespace" option, allowing the user to easily create a new namespace.

Possible Errors
None.

Related Operations
None.

5 Differences with C++ language specification

Our extension features some differences in implementation from the actual C++14 language. The next section describes these differences and the reasoning behind each of them.

5.1 Access modifiers

In the C++ extension access modifiers should always be explicitly defined. This is a design choice. Explicitly defining access modifiers makes the language easier to understand and there is no real reason not to add them.

5.2 Structs

The main difference between structs and classes in C++ lies in their access-specifiers. Members of a C++ struct are public by default, whereas members of a class are private by default. However, since the access-specifiers of class members are always explicit a struct and class have exactly the same functionality. The C++ extension still allows you to type the struct keyword, but it will create a class instead.

A C struct, however, functions differently from a C++ struct. As Mbeddr already incorporated the C struct into their C language, this was an available element in our C++ modules, which extends Mbeddr's C modules. In order to remove this element, we implemented a component that automatically changes an attempt at declaring a struct to a class declaration. This inherently means C structs are unavailable in our C++ modules, though their functionality can be achieved by using classes.

5.3 The Module system

In the C++ extension the user doesn't have to write header files. Instead of having a header and base file the code goes into modules. A module is like a file, it can import other modules and mark its content as exported. When a construct is marked as exported it will be in the header files of the exported C++ code and it will also show up in the auto completion menu when that module is imported by another module.

5.4 Pure virtual

In the C++ extension, the user has to use `pure` as a keyword instead of defining `virtual void functionName() = 0;` in C++. The user would then type: `pure virtual void functionName();`.

6 User's responsibilities

6.1 `delete []`

The delete keyword should be used with brackets when the delete will delete an array pointer, if this is not done the code will not compile. The editor does not give a warning for this, so the user should ensure this themselves when they will delete an array pointer.

Appendices

A Error messages and recovery procedures

Section	Classes
Error	`"Classes may only have one destructor. This class has " + numDestructors + "."`
Diagnosis	More than one destructor was defined for the given class.
Recovery	Remove all by the first destructor.
Error	`"This must be an instance of ClassType"`
Diagnosis	An object was provided that is a ClassType or its subclass.
Recovery	Enter a ClassType object instead.
Error	`"A class cannot extend itself"`
Diagnosis	An attempt is made to make an object extend itself.
Recovery	Extend a separate object instead.
Error	`"You can't extend a class from another module if it is not exported"`
Diagnosis	An attempt is made to extend another class that is missing the exported keyword.
Recovery	Make the class you want to extend, exported.
Error	`"You should select a constructor on class objects"`
Diagnosis	Class object is defined but not initialized or constructed.
Recovery	Initialize object elsewhere or add constructor.
Error	`"Name Collision: method is accessible from more than one ancestor"`
Diagnosis	Method name collision between parent classes.
Recovery	Rename method in either of the parent classes.

Section	Templates
Error	`"Template class type without template specifiers"`
Diagnosis	Specifiers are missing.
Recovery	Add specifiers to the template class.
Error	`"Non-template class type with template specifiers"`
Diagnosis	Specifiers should not be applied.
Recovery	Remove specifiers, or use a template class instead.

Error	`"Leftover template stub"`
`Diagnosis`	Template was created but not yet given a method or function.
`Recovery`	Finish the template class.
Error	`"Duplicate type name"`
`Diagnosis`	Template was created with a duplicate name.
`Recovery`	Give both templates a unique name.
Error	`"Param without default appearing after param with default"`
`Diagnosis`	Parameter does not have a default value, the one before it does have one.
`Recovery`	Make both default or remove the default parameter before it.
Error	`"Argument must be a type"`
`Diagnosis`	A type was not assigned to the template when a type was expected.
`Recovery`	Give the template a valid Type instance.
Error	`"Argument must be a value"`
`Diagnosis`	A value was not assigned to the template when a value was expected.
`Recovery`	Give the template a valid Type instance.
Error	`"Too few template arguments provided"`
`Diagnosis`	Template arguments expected lower than the number of arguments provided.
`Recovery`	Give the template a valid Type instance.
Error	`"Too many template arguments provided"`
`Diagnosis`	Template arguments expected higher than the number of arguments provided.
`Recovery`	Give the template a valid Type instance.

Section	**Keywords**
Error	`"constExpr already implicitly inlines"`
`Diagnosis`	Constant and inline expression both used.
`Recovery`	Use only the "constant" keyword.
Error	`"Constant data member must be initialized"`
`Diagnosis`	Constant expression used, yet not immediately initialized.
`Recovery`	Data member should not be constant or should be initialized at this point in time.
Error	`"Static data member can't be mutable"`
`Diagnosis`	Mutable and static keywords used together.

Recovery	Use either one, mutable and static creates a contradiction.
Error	`"Static class attributes can only be assigned to constant expressions"`
Diagnosis	Attempt is made to assign static class attributes to a variable.
Recovery	Static class attributes can only be assigned to constant values.
Error	`"Name Collision: attribute is accessible from more than one ancestor"`
Diagnosis	Attribute name collision between parent classes.
Recovery	Rename variable in either of the parent classes.
Error	`"Must not be thread local and register"`
Diagnosis	Thread_local and register keywords used together.
Recovery	Incompatible keywords used, contradiction, use either thread_local or register.
Error	`"name + " is a reserved keyword"`
Diagnosis	A named concept was given a name that can only be used by keywords.
Recovery	Change the name of the concept to one that is not already used by the compiler.
Error	`"Auto type can only be used in static const attributes"`
Diagnosis	Attribute is not a static constant value.
Recovery	Make the attribute a static constant using the 'static' and 'constant' keywords.
Error	`"Auto type can only be used in a variable declaration"`
Diagnosis	Data type has already been initialized.
Recovery	Add the auto keyword during variable initialization.
Error	`"Variable declaration with auto type must have initializer"`
Diagnosis	Data type is not initialized but the auto keyword is used.
Recovery	Initialize the auto keyword here.

Section	**Primitive Types**
Error	`"Sorry, this is not a valid initialization for char16_t, you cannot use negative numbers"`
Diagnosis	Negative number is being assigned to a char16_t type.
Recovery	Use a non-negative number between 0-65535.
Error	`"Sorry, this is not a valid initialization for char16_t, use a number between 0 and 65535"`
Diagnosis	Negative number is being assigned to a char16_t type.

Recovery	Use a non-negative number between 0-65535.
Error	`"Sorry, this is not a valid initialization for char32_t, you cannot use negative numbers"`
Diagnosis	Negative number is being assigned to a char32_t type.
Recovery	Use a non-negative number between 0-1114111.
Error	`"Sorry, this is not a valid initialization for char32_t, use a number between 0 and 1114111"`
Diagnosis	Negative number is being assigned to a char32_t type.
Recovery	Use a non-negative number between 0-1114111.
Error	`"Sorry, this is not a valid initialization for wchar_t, you cannot use negative numbers"`
Diagnosis	Negative number is being assigned to a wchar_t type.
Recovery	Use a non-negative number.

Section	**Operator Overloading**
Error	`"Expected a maximum of 2 arguments, got arg_count"`
Diagnosis	More than two arguments are given.
Recovery	Give only 2 or less arguments for operator overloading.
Error	`"Operator " + operatorPresentation + " can't be used for types " + left.getPresentation() + " and " + right.getPresentation()`
Diagnosis	No operator can be found that can fit the operands in this function.
Recovery	Make sure the operands are of the correct type and the operator is expecting the correct types.
Error	`"Operator [] does not accept argument of type " + indexType.getPresentation()`
Diagnosis	The operator [] cannot be used for the given 'index' argument.
Recovery	Use a different unary operator and make sure the 'index' argument type is correct.
Error	`"'--' operator is not defined on type " + expressionType.getPresentation()`
Diagnosis	The operator '–' is incompatible with the given expression.
Recovery	Make sure the 'expression' type is correct and the operator expects the correct type.
Error	`"'++' operator is not defined on type " + expressionType.getPresentation()`
Diagnosis	The operator '++' is incompatible with the given expression.

`Recovery`	Make sure the 'expression' type is correct and the operator expects the correct type.

Section	**Miscellaneous**
Error	`"The type is not a pointer; only variables of type pointer can be deleted"`
`Diagnosis`	The user is attempting to delete a non-pointer object.
`Recovery`	Refer to a pointer object instead.
Error	`"Catch block after catch all is redundant"`
`Diagnosis`	Unreachable catch block used.
`Recovery`	Place the catch block before the catchall block.
Error	`"Elements must be of type " + array.type + ", was:  " + elem.type`
`Diagnosis`	Element type does match array type.
`Recovery`	Change either the element's type or the array's type to match one another.
Error	`"array size does not match (expected:  " + expectedSize + ", was:  " + elements.size + ")"`
`Diagnosis`	Assigned number elements does not match array length.
`Recovery`	Change the size of the array or the number of the elements.

B Glossary

Access Modifiers	Modifiers that change the accessibility of parts of the program (classes, methods, etc.) These include the keywords `public, private, and protected`.
Argument	Parameter given alongside a function/method to instantiate that function/method.
Auto completion Menu	Menu that expands as a context menu in MPS. Offers suggested actions, intentions and code completion.
Code Generation	Transforming an MPS abstract syntax tree into compilable plain text source code.
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.
Expression	A block of code containing a combination of values, constants, variables, operators and functions that produces a value.
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.
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.
Unreachable Code Path	A portion of a code base that will never be executed.

C Index

Word definitions are linked to usage in the SUM.