fUML - 

User-Guide

Author: Maximilian Hammer

Release Date: 2024-01-18

Document Version: 1.0

Version History:

Version	Release Date	Author(s)
1.0	2024-01-18	Maximilian Hammer

Table of Contents

1 Prerequisites

This section contains general system information about the environments and platforms that fUML-C++ has been tested on as well as software that is required for building and using fUML-C++ and optional software.

1.1 System Information

fUML-C++ is by now only tested on Windows. Since GCC is used for compilation and the Eclipse internal builder handles build management, operation on Linux systems should be possible. As the build scripts within fUML-C++ are currently only configured for Windows, **some descriptions in the remainder of this document will be focused on Windows as the underlying operating system**.

Concerning the compiler version, at least GCC version 4.8 is required as fUML-C++ uses C++11 features.

1.2 Required Software

This section covers required software that has to be set up on your system before being able to work with fUML-C++.

1.2.1 Compiler

The build scripts of fUML-CPP are configured to use GCC either a GCC (Linux) or MinGW-w64 (Windows) installation is required (see GCC or MinGW-w64).

1.2.2 Java

A valid Java installation is required to run Eclipse (see section 1.2.3).

1.2.3 Eclipse IDE / C++-Development Tools

Eclipse is the one and only IDE used with fUML-C++. Together with the IDE itself, the *C++-Development Tools* plugin is required. If you already have an Eclipse IDE set up, you can simply install the plugin. If you set up a new Eclipse IDE, it is recommended to use the Eclipse CDT release.

1.3 Optional software

This section covers recommended but optional software that is not necessarily required to run fUML-C++ but will contribute to a faster and more productive workflow.

1.3.1 Eclipse Modeling Tools

Eclipse Modeling Tools is another set of plugins for the Eclipse IDE, which provides tools for creating and processing UML models and profiles. Just like Eclipse CDT, there is also a pre-packed release that already contains these plugins, see Eclipse Modeling Framework.

1.3.2 Papyrus

Eclipse Papyrus is a much more sophisticated software for UML modeling than the plain Eclipse Modeling Tools. It also provides graphical modeling functionality and is highly recommended as the main modeling tool when working with fUML-C++.

1.3.3 Acceleo

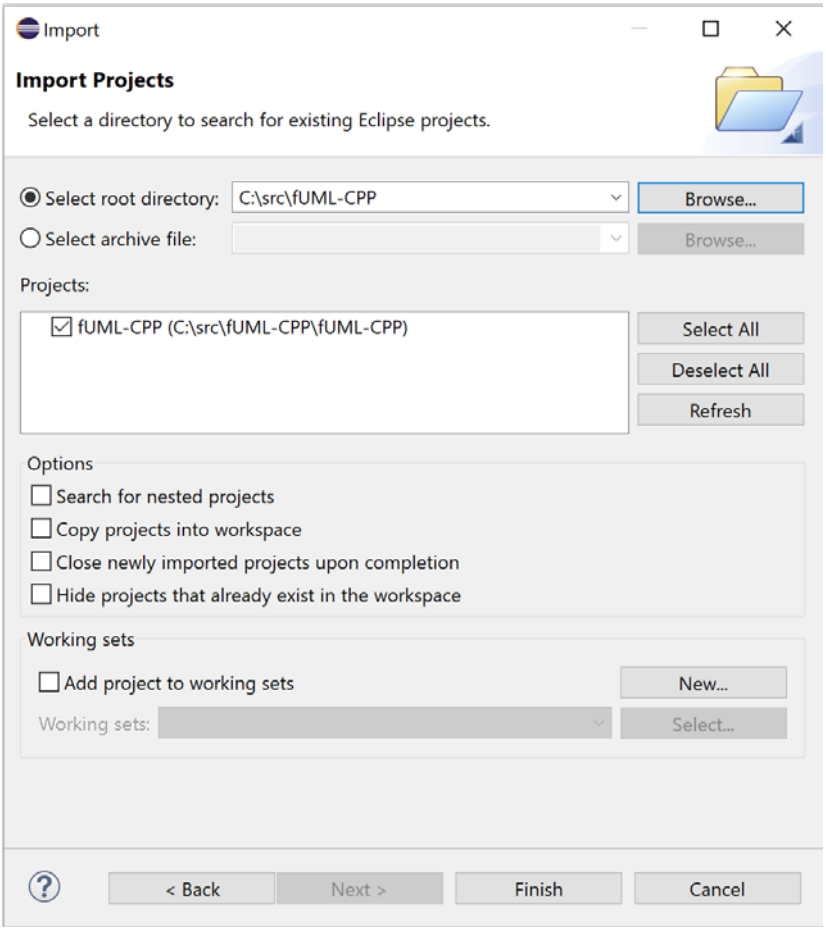
By now, fUML-C++ does not provide any XMI-parsing functionality to execute UML models. As described in detail in section 4, there are two possible ways to create executable models in fUML-C++. On the one hand you can create an executable model using source code. As soon as models get slightly larger, this is a rather time consuming task. That's why fUML-C++ provides a code-generation facility, which produces compilable C++ source code from UML models.
Acceleo is yet another Eclipse tool which provides functionalities to develop and run model-to-text transformation tools. As the code generator of fUML-C++ was in fact built with Acceleo, a valid Acceleo installation within your Eclipse IDE is required if you want to use the generator.

2 Setup

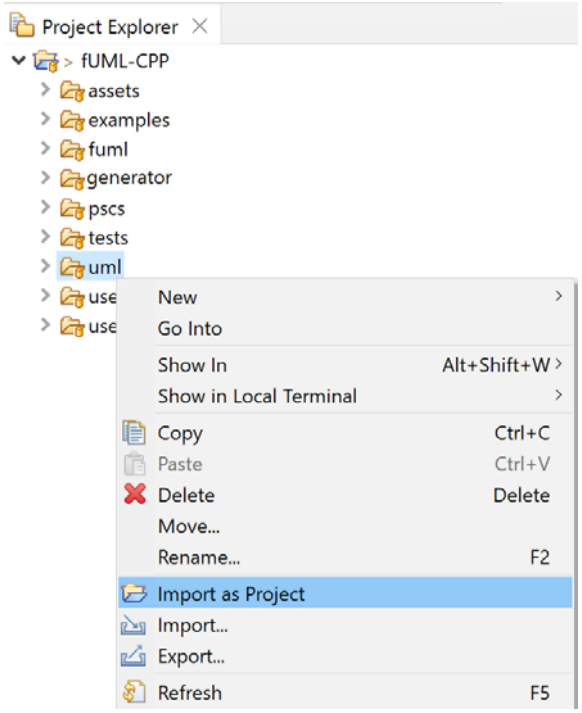
This section covers additional steps that are necessary before building and using fUML-C++.

2.1 Setting up Eclipse

First, prepare the Eclipse IDE for the build process. Start Eclipse and choose a workspace directory. Import the fUML-C++ root project called *fUML-CPP*, which is located at "<fUML-C++-rootdir>\fUML-CPP\".



Next, import the nested projects for UML and fUML. This can be done by right clicking the corresponding folder in the project explorer and choosing *Import as Project* in the context menu as shown below. Do this for folders "fUML-CPP/uml" and "fUML-CPP/fUML".



2.2 Environment Variables

Since UML and fUML are compiled into two distinctive dynamic libraries within fUML-C++, the corresponding binary directories have to be added to the operating system's environment variables, so that they can be found at runtime. Depending on your preferences, UML debug version will be compiled to "<fUML-C++-rootdir>\fUML-CPP\uml\Debug\umld.dll" and "<fUML-C++-rootdir>\fUML-CPP\uml\Release\uml.dll" UML release version will be compiled to. fUML debug version will be compiled to "<fUML-C++-rootdir>\fUML-CPP\fuml\Debug\fumld.dll" and fUML release version will be compiled to "<fUML-C++-rootdir>\fUML-CPP\fuml\Release\fuml.dll".

For Windows, add the following lines to the PATH environment variable and replace "C:\src\fUML-CPP" with your fUML-C++ root directory path.

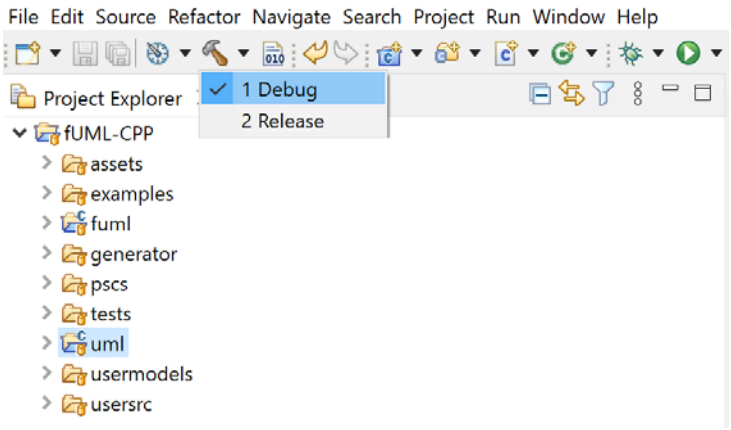
```
C:\src\fUML-CPP\fUML-CPP\uml\Debug
C:\src\fUML-CPP\fUML-CPP\uml\Release
C:\src\fUML-CPP\fUML-CPP\fuml\Debug
C:\src\fUML-CPP\fUML-CPP\fuml\Release
```

3 Build

To build fUML-C++, dynamic libraries for UML as well as fUML have to be compiled. Use the build functionality of the Eclipse C++-Development Tools as shown below.

Depending on your own preferences, you can build either the debug versions, the release version or both.

As fUML depends on UML, the uml project has to be built before the fuml project.



4 Usage

In fUML-C++ there are two possible ways to create executable models:

- By creating a user-defined C++-project and implement the model to be executed by hand
- Automatically generating a model-specific C++-project from a user-defined UML model using the generator component

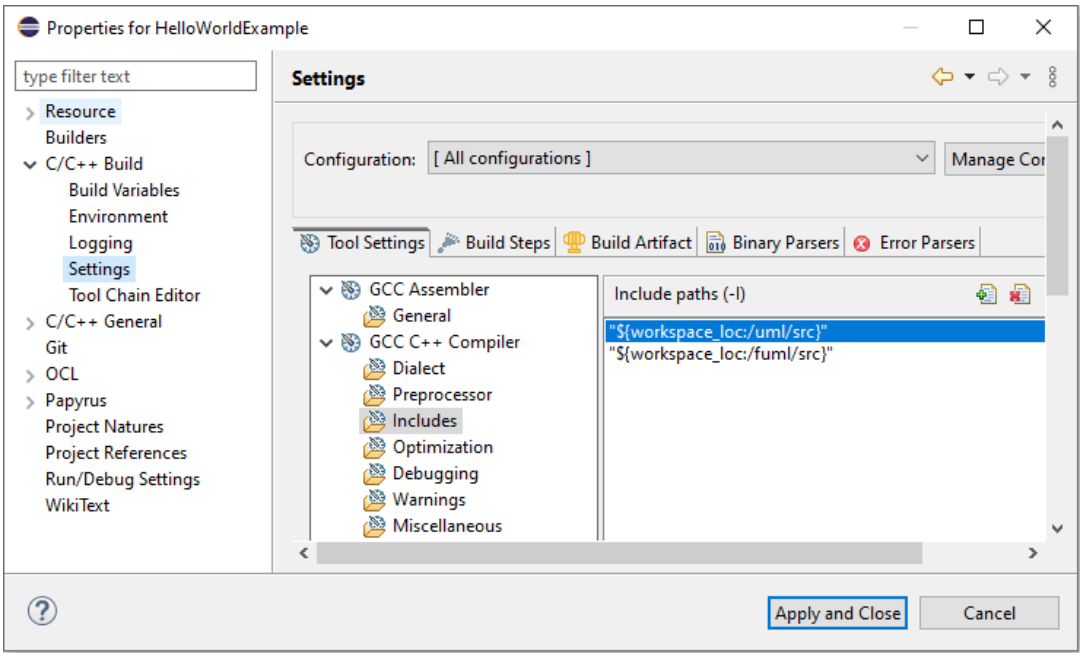
Both methods are described below.

4.1 Creating executable models using source code

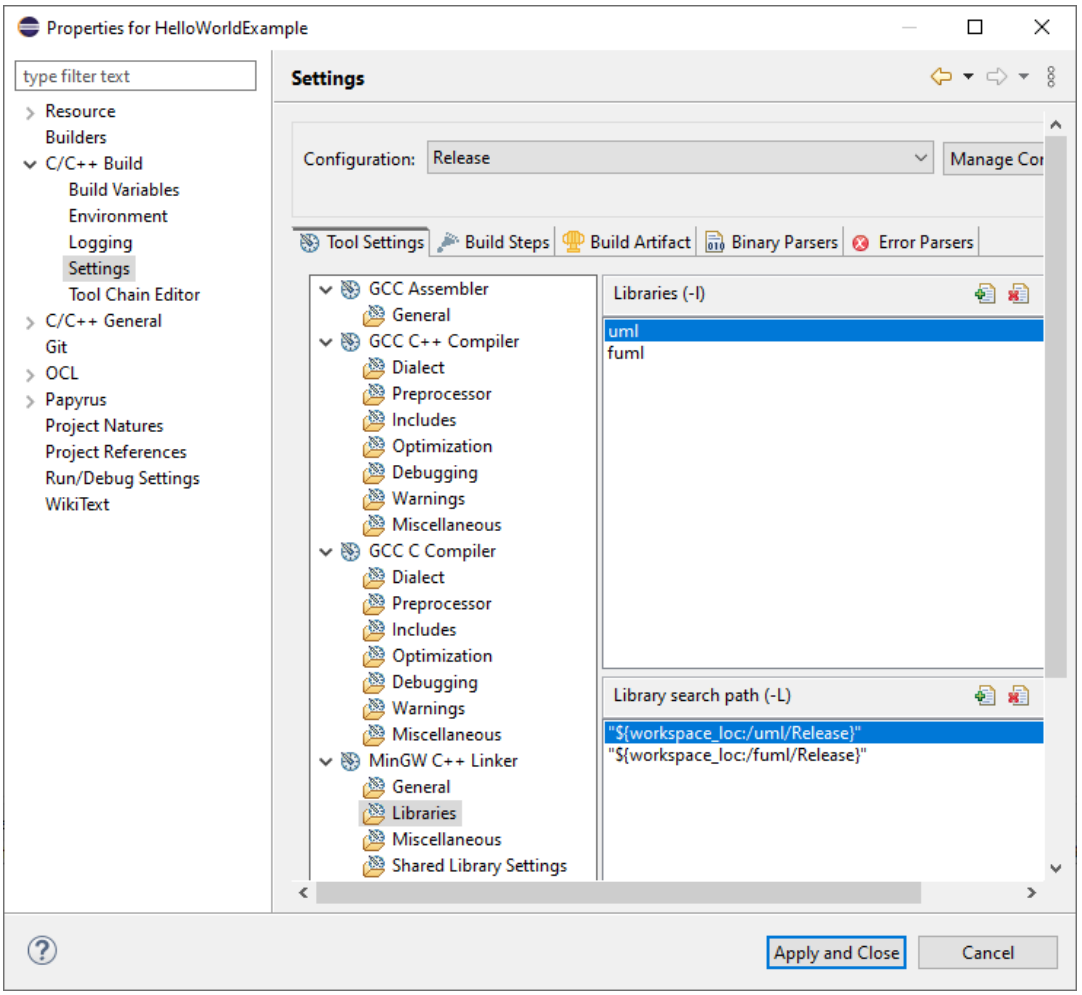
NOTE: See "<fUML-C++-rootdir>\examples\helloworld" as a reference project for creating executable models directly from source code.

First, create a new C++-project within the fUML-C++ root project. It is suggested to use the common directory "<fUML-C++-rootdir>\usersrc" for user-defined source-code projects. Within this directory, you can organize your projects in an arbitrary structure of nested subdirectories.

Next, add required include directories “fUML-CPP\uml\src” and “fUML-CPP\fuml\src” to the include directories of your project via the project properties as shown below.



Now also add the uml and fuml binaries to the linker settings of your project via the project properties as shown below. Be careful to link the correct versions of the binaries (debug versions for debug builds and release versions for release builds).



fUML-C++ executable models require at least two specific classes to be implemented: A model-specific *Environment* class and a model-specific *Model* class. fUML-C++ provides two abstract base classes that must be derived in user-defined executable models.
Create a <model-name>*Environment* class by deriving from class *fuml::environment::Environment*. A user-defined *Environment* class must at least provide:

- a virtual default destructor
- a static *Instance*() method (as *Environment* classes are singleton classes)

See "<fUML-C++-rootdir>\examples\helloworld\HelloWorldExampleEnvironment.cpp" to see how a valid *Instance*() method should be implemented.

Next, create <model-name>*Model* class by deriving from class *uml::environment::InMemoryModel*. A user-defined *Model* class must at least provide:

- a virtual default destructor
- a static *Instance*() method (as *Model* classes are singleton classes).
- a *void initializeInMemoryModel()* method (as this method should be called from the *Instance*() method of the <model-name>*Model* class)

Additionally the *Model* class should contain all of the desired model elements as public members (they should be public so that multiple models can potentially access each other's elements). Within the *initializeInMemoryModel()* method all model elements must be instantiated and all of their properties must be set and initialized. See "<fUML-C++-rootdir>\examples\helloworld\HelloWorldExampleModel.cpp" to see how a valid *initializeInMemoryModel()* method should be implemented.

Last, create a main module (e.g. "<model-name>.cpp") that contains a main function. Within the main function, call <model-name>*Environment::Instance()->execute("<behavior-name>");* for each behavior you want to execute in subsequent order.

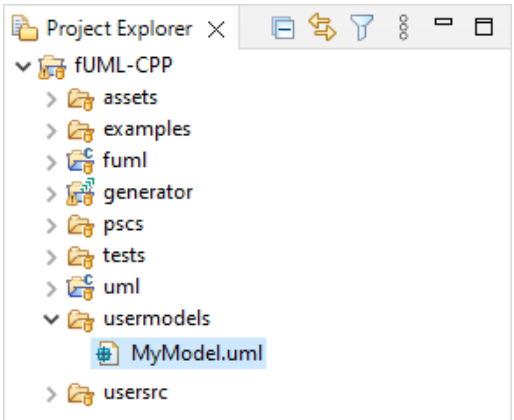
Build the C++-project and run the corresponding executable(s).

4.2 Generating executable models from .uml models

A more convenient method to create executable models within fUML-C++ is to automatically generate the corresponding C++-project from a UML model using the built-in code generator.

NOTE: To be able to use the code generator, the Acceleo plugin must be installed in your Eclipse IDE (see section 1.3.3).

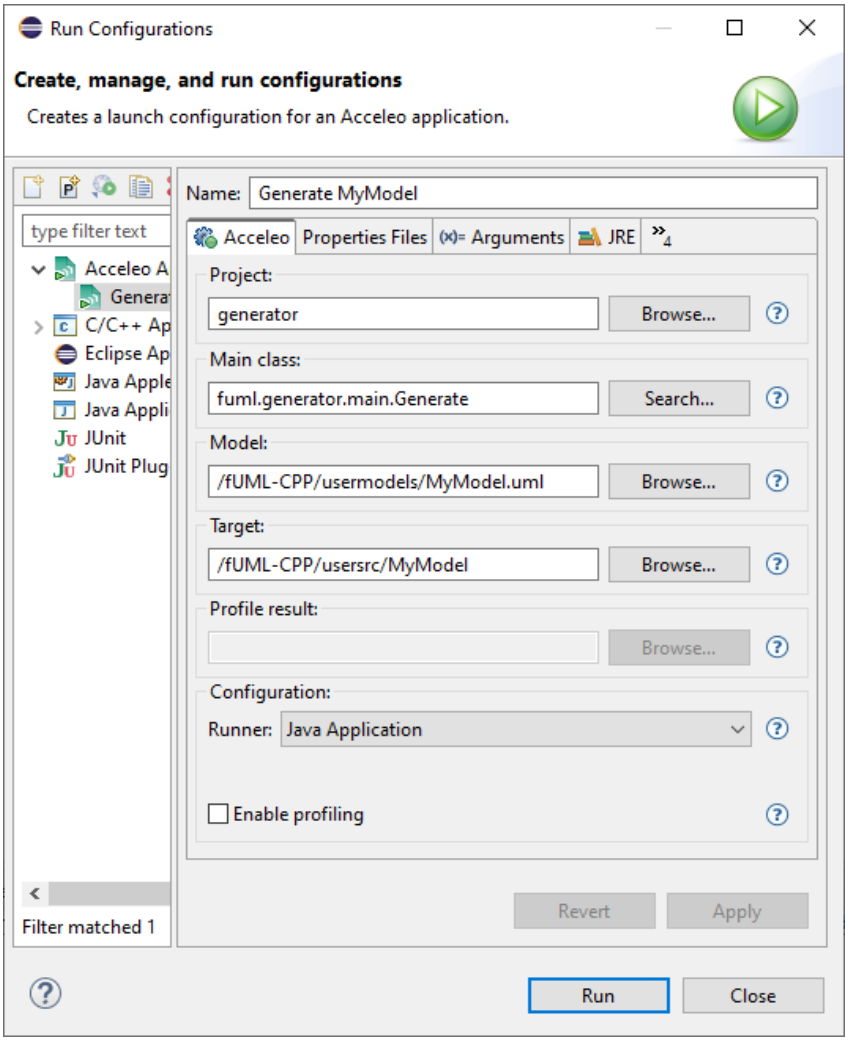
Create a new UML model with a modeling tool of your choice (Eclipse Papyrus is suggested, see section 1.3.2). It is suggested to store user-defined models in the common directory "<fUML-C++-rootdir>\usermodels". Within this directory, you can organize your models in an arbitrary structure of nested subdirectories.



Import the nested *generator* project within the fUML-C++ root project. Navigate to file "\generator\src\fuml.generator.main\generate.mtl" in the project explorer. Right click *generate.mtl* and choose "*Run as*" → "*Run configurations...*". Configure a new Acceleo application as shown below:

- define a name for the generator application
- choose *fuml.generator.main.Generate* as the application's main class
- choose your model file
- choose a target directory for the generated source code ("<fUML-C++-rootdir>\usersrc\<model-name>" is suggested)

Run the generator application.



Now import the newly generated C++-project within the fUML-C++ root project and build the desired executable(s). After the compilation process is finished, run the executable(s) from the command line using <executable-name> <behavior-name> [<behavior-name> <behavior-name> <behavior-name> <...>].