User-Guide

Author: Maximilian Hammer

Release Date: 2024-01-30

Document Version: 1.0

Version History:

Version	Release Date	Author(s)
1.0	2024-01-30	Maximilian Hammer

Table of Contents

1 Prerequisites

This section contains software that is required for building and using fUML-C# and optional software.

1.1 Required Software

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

1.1.1 Microsoft Visual Studio

Microsoft Visual Studio is required for building fUML-C#. It would also be possible to build the project without Visual Studio using msbuild.exe, but it is highly suggested to use Visual Studio as an IDE. Concerning the framework version, at least .NET 6.0 is required

1.2 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.

1.2.1 Java

A valid Java installation is required to run Eclipse (see section 1.2.2).

1.2.2 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.

1.2.3 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#.

1.2.4 Acceleo

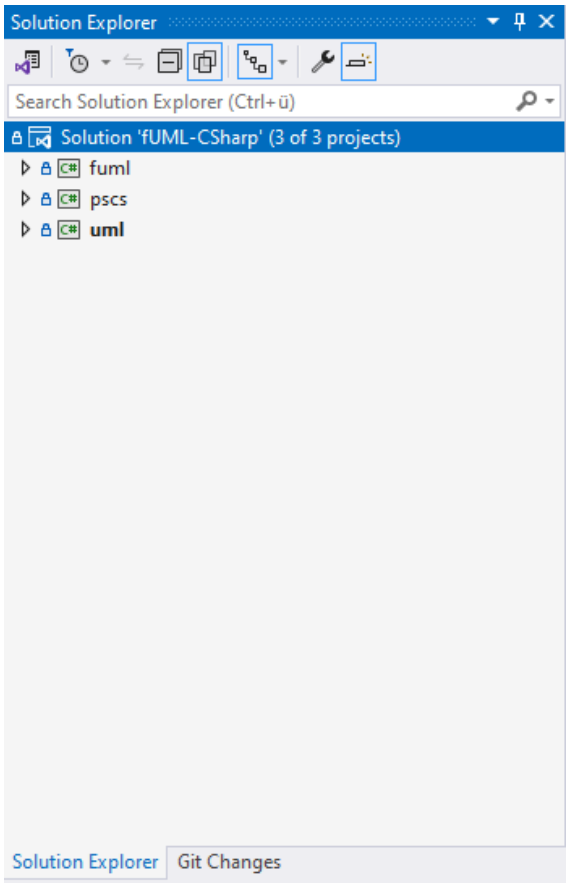
By now, fUML-C# does not provide any XMI-parsing functionality to execute UML models. As described in detail in section 4, there are two possible ways to create executable models in fUML-C#. On the one hand, you can create an executable model using source code. As soon as models get slightly larger, this is a rather time consuming task. That is 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 Visual Studio

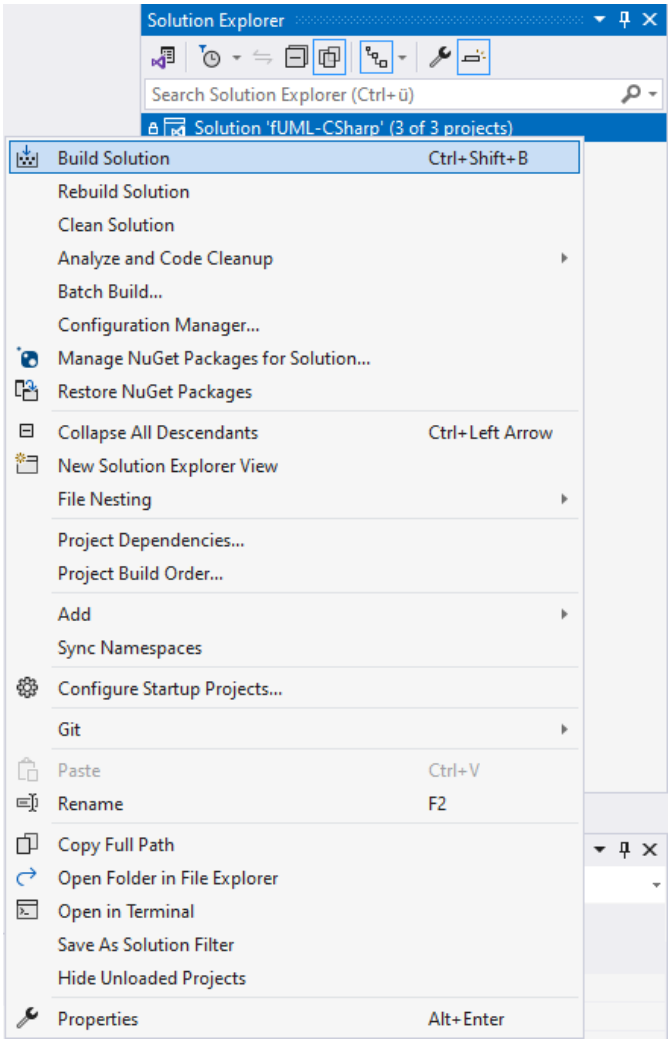
First, prepare the Visual Studio IDE for the build process. Start Visual Studio and import the fUML-C# root solution called *fUML-CSharp.sln*, which is located at "<fUML-C#-rootdir>\fUML-CSharp\".



3 Build

To build fUML-C#, dynamic libraries for UML, fUML as well as PSCS have to be compiled. Use the build functionality of the Visual Studio IDE as shown below.

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



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

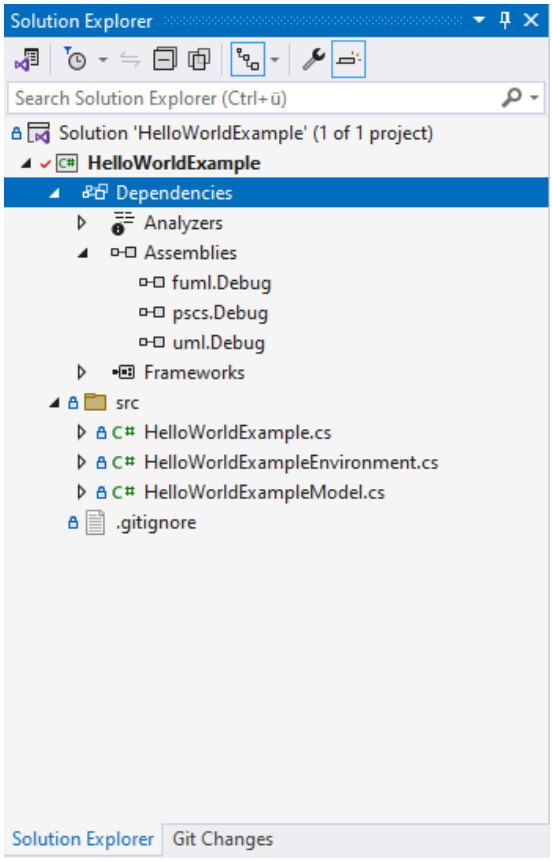
Both methods are described below.

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# Console Application 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 assemblies for “uml”, “fuml” and “pscs” to the references of your project 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 static *Instance*() method (as *Environment* classes are singleton classes)

See "<fUML-C#-rootdir>\fUML-CSharp\examples\helloworld\src\HelloWorldExampleEnvironment.cs" 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 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>\fUML-CSharp\examples\helloworld\src\HelloWorldExampleModel.cs" to see how a valid *InitializeInMemoryModel()* method should be implemented.

Last, create a main module (e.g. “<model-name>.cs”) 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.

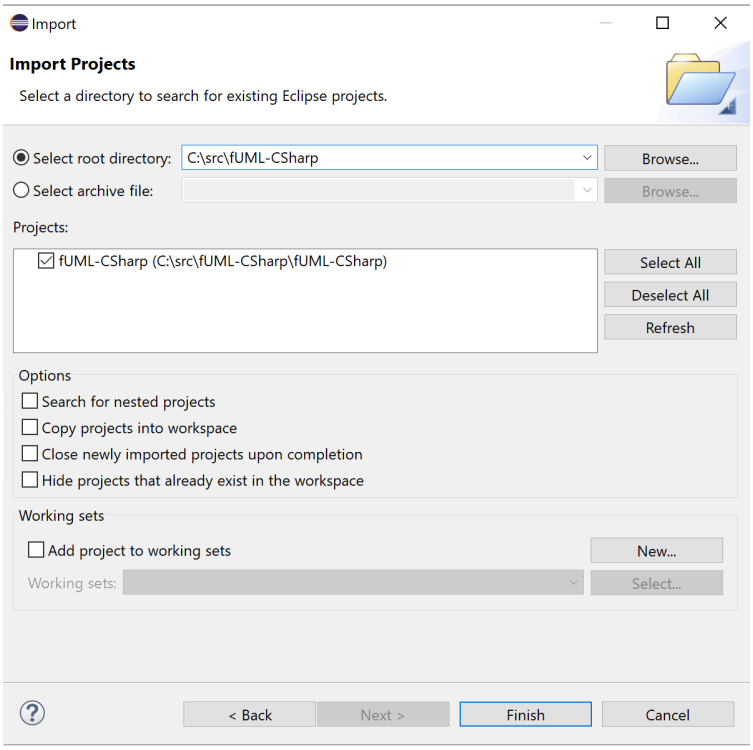
Build the C#-project and run the corresponding executable(s).

4.2 Generating executable models from .uml models

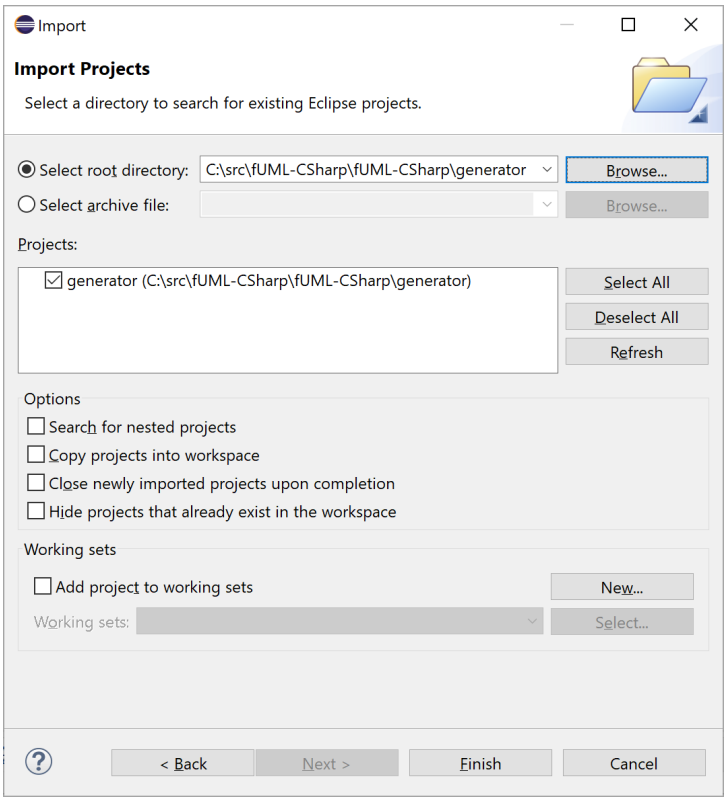
A more convenient method to create executable models within fUML-C# is to automatically generate the corresponding C#-project from a UML model using the built-in code generator.

NOTE: To be able to use the code generator, the Acceleo plugin must be installed in your Eclipse IDE (see section 1.2.4).

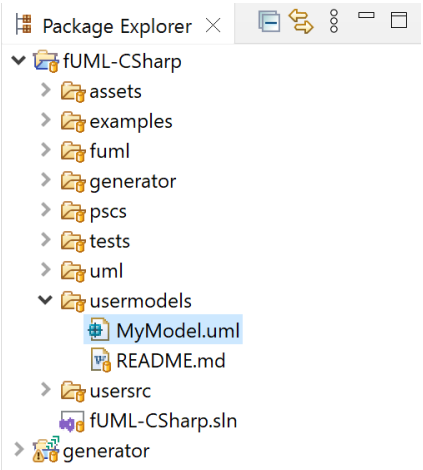
Start Eclipse and choose a workspace directory. Import the fUML-C# root project called *fUML-CSharp*, which is located at “<fUML-C#-rootdir>\fUML-CSharp\”.



In the same manner, import the nested project called *generator* which is located at “<fUML-C#-rootdir>\fUML-CSharp\generator”.



Create a new UML model with a modeling tool of your choice (Eclipse Papyrus is suggested, see section 1.2.3). It is suggested to store user-defined models in the common directory "<fUML-C#-rootdir>\fUML-CSharp\usermodels". Within this directory, you can organize your models in an arbitrary structure of nested subdirectories.

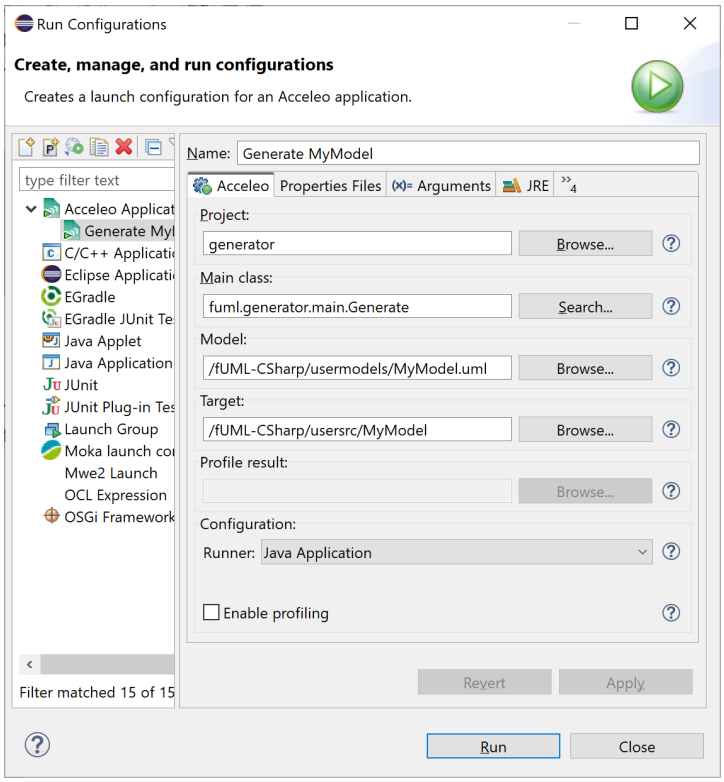


Next, change

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 open the generated C#-project in Visual Studio 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> <...>].