Creating a Custom Stubs Generator

If you desire to create either your own stubs generator for the C# base language, you want to modify *CsStubsGenerator* or you plan to utilize *StubsGenerator* to develop a stubs generator for your base language, you can make use of this guide, where we describe an implementation of a stubs generator based on the *StubsGenerator* plugin from scratch.

Since you plan to do a quite advanced thing, we assume that we do not need to explain the process into too much detail and we therefore we keep the guide rather brief and fast.

First, you should make sure that you have the *StubsGenerator* plugin installed. You can use JetBrains MPS Marketplace for that.

Then, create a solution project. It will contain your plugin that will be called by *StubsGenerator* in order to create AST nodes for concepts from your base language.

Create a plugin solution (right-click the project, select *New* and then *Plugin Solution*) and then create a model *core* inside.

Set the dependencies of the *core* model to contain the *StubsGenerator.core* model, the Structure model of your base language and the Base Language. In Figures 3.18 and 3.19, you can see a full example of dependencies of *CsStubsGenerator*.

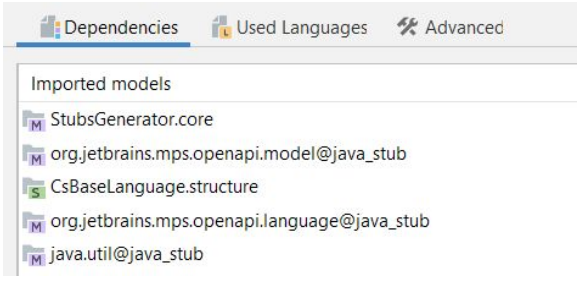

Figure 3.18: Model dependencies of the model CsStubsGenerator.core

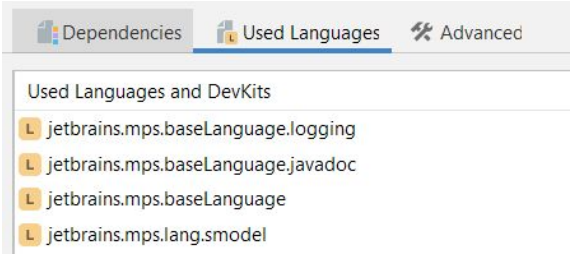

Figure 3.19: Language dependencies of the model CsStubsGenerator.core

The dependencies of the *plugin* model should primarily contain a dependencies on your plugin's *core* model, on the *plugin* model of *StubsGenerator* and on the Base Language. Again, we present a full example of dependencies of *CsStubsGenerator* in Figures 3.20 and 3.21.

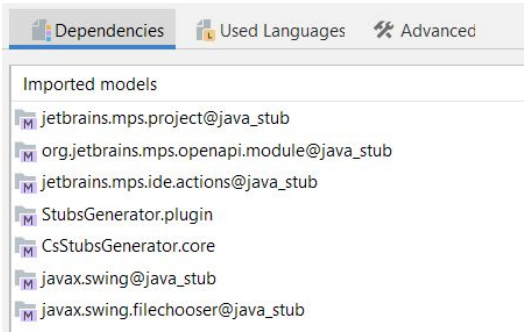

Figure 3.20: Model dependencies of the model CsStubsGenerator.plugin

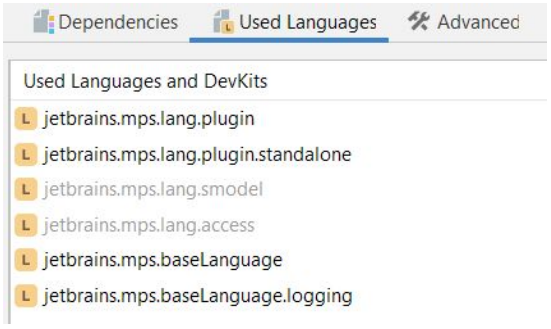

Figure 3.21: Language dependencies of the model CsStubsGenerator.plugin

Now you can create an action that will be displayed after right-clicking a solution module and that will offer your plugin's generation functionality. Right-click the *plugin* model, select *New*, *j.m.lang.plugin*, *Action*. In Figure 3.22, there is a header part of this action. Notice particularly the flag *execute outside command* set to true. Figure 3.23 then illustrates a possible implementation of this action. The most important part is at the end where we call the *StubsGenerator's* prepared static method which handles calling of *StubsGenerator* properly.

```
action GenerateStubsAction {
  execute outside command: true
  also available in: << ... >>

  caption:     Generate Stubs
  mnemonic:    <no mnemonic>
  description: Generate stub models for specified C# entities into a selected solution
  icon:        <no icon>

  construction parameters
    << ... >>

  action context parameters ( always visible = false )
    MPSProject project key: MPS_PROJECT required
    SModule    module  key: MODULE      required
```

Figure 3.22: Stubs generation action header

```
execute(event)->void {
  if (!this.module instanceof Solution) {
    message error "Cannot run stubs generation into a non-solution module", <no project>, <no throwable>;
    return;
  }
  final Solution solution = (Solution) this.module;
  final SRepository repository = this.project.getRepository();

  final string specificationFilePath;
  JFileChooser fileChooser = new JFileChooser();
  fileChooser.setDialogType(JFileChooser.OPEN_DIALOG);
  fileChooser.setFileFilter(new FileNameExtensionFilter("XML files", "xml"));
  if (fileChooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
    specificationFilePath = fileChooser.getSelectedFile().getAbsolutePath();
  } else {
    return;
  }

  repository.getModelAccess().executeCommand(new Runnable() {
    public void run() {
      Action.generateStubs(repository, solution, new CsSingleMpsEntityGenerator(), new CsConstructionHelper(),
          specificationFilePath);
    }
  });
}
```

Figure 3.23: Stubs generation action implementation

Now you can create an action group via right-clicking the *plugin* model and selecting *New*, *j.m.lang.plugin*, *Group*. It should have a content similar to what you can see in Figure 3.24.

```
group GenerateStubsGroup
is popup: false

contents
   GenerateStubsAction

modifications
   add to SolutionActions at position commonModule
```

Figure 3.24: Stubs generation action group

Here comes finally the interesting part. You will create two classes that get called by *StubsGenerator* in order to obtain AST nodes of concepts from your selected base language. These classes might be very simple but they tend to get complex if your base language is complex. For example, in case of the C# base language the code gets to high hundreds of lines. But, in this tutorial, we can keep it simple because the idea is, fortunately, visible even then.

First, let's create a class called construction helper. This class provides the stubs generation with meta-information or secondary routines. Create it and implement the *ConstructionHelper* interface from *StubsGenerator*. You should override the method *getUsedLanguages*, which returns language dependencies that should be set to the models containing the generated stubs, and the method *constructRootNode*, whose task is to wrap a given AST node into an AST node whose concept has the *instance-can-be-root* flag set to true. Examples can be found in Figure 3.25. The *constructRootNode* method may only return the given AST node without any change - it is up to design of your plugin, the design of your base language and the design of the XML file. For example, we, in our *CsStubsGenerator*, used this method.

```
public class MyConstructionHelper implements ConstructionHelper {
  @Override
  public List<SLanguage> getUsedLanguages() {
    List<SLanguage> usedLanguages = new ArrayList<SLanguage>();
    usedLanguages.add(language/MyBaseLanguage/);
    return usedLanguages;
  }

  @Override
  public SNode constructRootNode(SNode node, MpsEntitySpec spec,
      MpsEntityCollectionGenerator mpsEntityCollectionGenerator) {
    return node;
  }
```

Figure 3.25: Example of a construction helper

Approaching the end, we need to create the second class, called a *single MPS entity generator*, according to the interface *SingleMpsEntityGenerator* which it implements. This class gets called from *StubsGenerator* when it is needed to generate a stub based on the specification coming from the XML file. The basic use case of this class, encapsulated in the *generateMpsEntity* method, is that you get an object *MpsEntitySpec* that represents an XML element, contains a set of properties (mostly the XML attributes) and references to *MpsEntitySpec* objects corresponding to the child XML elements. Based on this specification you should generate an AST node representing the specified stub. In Figure 3.26, a really simple implementation of this class's *generateMpsEntity* method is presented.

As you can see, the *generateMpsEntity* method decides according to a mandatory property *entityKind* of the *MpsEntitySpec* object what stub should be generated. This property is the name of the corresponding XML element. The method assumes that there are two kinds of XML elements in the XML file. First, *Model*, results in a generated MPS model containing the stubs corresponding to inner XML elements of this *Model* element (this is handled automatically by StubsGenerator). And second, *MyLanguageConcept*, that results in an AST node of concept *MyLanguageConcept*, which has no special properties, children or references (otherwise we would have to set them manually in this method to the AST node).

```
public class MySingleMpsEntityGenerator implements SingleMpsEntityGenerator {
  @Override
  public MpsEntity generateMpsEntity(final MpsEntitySpec mpsEntitySpec,
      MpsEntityCollectionGenerator mpsEntityCollectionGenerator, MpsEntity alreadyGeneratedPart) {
    string entityKind = getMpsEntityKind(mpsEntitySpec);

    switch (entityKind) {
      case "Model" :
        return new Model(mpsEntitySpec, "MyModelWithStubs");
      case "MyLanguageConcept" :
        return new Stub(mpsEntitySpec, new node<MyLanguageConcept>());

    }

    return null;
  }

  private string getMpsEntityKind(MpsEntitySpec spec) {
    Object entityKindObject = spec.getEntityKind();

    if (entityKindObject == null || !(entityKindObject instanceof string) ||
        entityKindObject.equals(MpsEntitySpec.ENTITY_KIND_DEFAULT)) { return null; }

    return (string) entityKindObject;
  }
```

Figure 3.26: Example of the generateMpsEntity method

If you noticed the *mpsEntityCollectionGenerator* parameter, it is a way how to call-back *StubsGenerator* to give you a reference to another stub (possibly generating it out of order immediately, when it has not been generated yet). It is useful when you, for example, need to reference an inherited class or a return type of a method.

And the last parameter, *alreadyGeneratedPart*, is bound to another method of the class. Sometimes, *StubsGenerator* needs to prepare a stub before its actual generation (it relates to the out-of-order generation mentioned before). This stub can be generated empty but needs to exist. For this purpose, *StubsGenerator* calls your class to get an empty AST node of a concept specified by an *MpsEntitySpec*, and later on, when the actual generation of this AST nodes takes place, it passes this unfinished AST node via the *alreadyGeneratedPart* parameter. Figure 3.27 illustrates an implementation of this second method corresponding to the example in Figure 3.26.

```
@Override
public MpsEntity generateEmptyMpsEntity(final MpsEntitySpec mpsEntitySpec) {
  string entityKind = getMpsEntityKind(mpsEntitySpec);

  switch (entityKind) {
    case "Model" :
      return new Model(mpsEntitySpec, "MyModelWithStubs");
    case "MyLanguageConcept" :
      return new Stub(mpsEntitySpec, new node<MyLanguageConcept>());

  }

  return null;
}
```

Figure 3.27: Example of the generateEmptyMpsEntity method

It actually looks the same as the first method but it would not if the concept of our example language had some children, references or properties. Then the *generateMpsEntity* would be more complex and different from the *generateEmptyMpsEntity* method.

With this knowledge, you now see that our example implementation of the *generateEmptyMpsEntity* method is actually wrong as we do not take the *alreadyGeneratedPart* into account. We should check if it is null and if not, we should use it for our stub generation.

This is the end of this tutorial, this is more-or-less all you need know to create a simple stubs generator for your base language. If you had any problem, look into the documentation of *StubsGenerator*, into its in-source comments, or into the *CsStubsGenerator* implementation.