Xtext 2 - Creating a Grammar from Scratch

> **Xtext tutorials**
>
> The two Xtext tutorials are stand-alone tutorials. You do not need to do the Xtext I tutorial in order to perform the Xtext II tutorial. Ask your supervisor which tutorial suits you best.

This tutorial presents the Xtext framework, a toolsuite for the generation of plain text based model editors. Such textual editors provide syntax highlighting, content assist (ctrl-space), an outline, and much more out-of-the-box. You will start by creating a grammar for Brainfuck.

- Preliminaries
  - Required Software
  - Recommended Tutorials
  - Brainfuck
- Creating a Grammar
- To go further

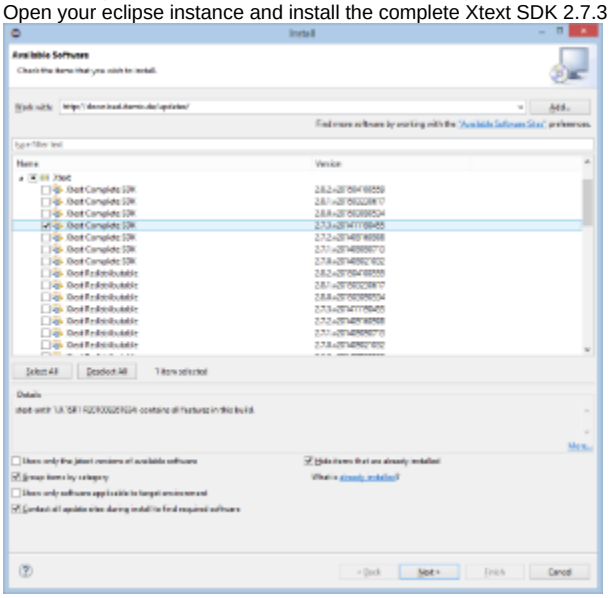
Preliminaries

There's a few things to do before we dive into the tutorial itself.

Required Software

For this tutorial, we need you to have Eclipse installed:

1. Install Eclipse. For what we do, we recommend installing the Eclipse Modeling Tools, with a few extras. Our Wiki page on getting Eclipse has the details: simply follow the instructions for downloading and installing Eclipse and you should be set.
2. Open your eclipse instance and install the complete Xtext SDK 2.7.3 from the itemis updatesite: http://download.itemis.de/updates/



Recommended Tutorials

We recommend that you have completed the following tutorials before diving into this one.

1. Eclipse Plug-ins and Extension Points
2. Eclipse Modeling Framework (EMF)

Brainfuck

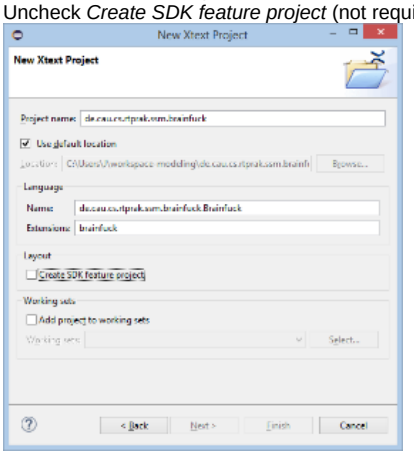
Basically, you are going to write an Xtext grammar for Brainfuck (BF) in this tutorial. Conceptionally BF programs are working on an endless tape. The program can navigate from cell to cell on the tape and modify the value stored in a particular cell. Your BF language should know the following commands:

<	Move one cell to the left on the tape.
>	Move one cell to the right on the tape.
+	Increment the value in the actual cell.
-	Decrement the value in the actual cell.
.	Print the actual cell to the console.
[	If the value in the actual cell is 0, jump to the instruction after the corresponding]
]	If the value of the actual cell is not 0, jump to the instruction after the corresponding [

Creating a Grammar

An Xtext grammar is always related to a specific EMF meta model. The grammar defines a concrete syntax in which instances of the meta model (the abstract syntax) can be serialized and stored. Xtext supports two ways of linking a grammar with a meta model: either creating a grammar for an existing meta model, or creating a grammar first and generating a meta model out of it. In this tutorial you will create a new grammar from scratch!

1. Select *File* *New* *Project...* *Xtext* *Xtext Project* *Next*
2. Enter the following values:
   - *Project name:* de.cau.cs.rtprak.<login>.brainfuck (like in previous tutorials, replace "login" by your login name)
   - *Location:* your repository path
   - *Name:* de.cau.cs.rtprak.<login>.turing.brainfuck.Brainfuck
   - *Extensions:* brainfuck (the file extension under which your text files will be stored)
   - Uncheck *Create SDK feature project* (not required)

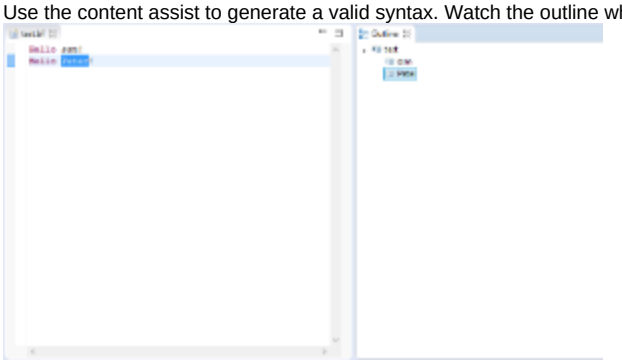


3. *Finish* an editor is opened with a predefined grammar (Brainfuck.xtext).

```
grammar de.cau.cs.rtprak.ssm.brainfuck.Brainfuck with org.eclipse.xtext.common.Terminals

generate brainfuck "http://www.cau.de/cs/rtprak/ssm/brainfuck/Brainfuck"

Model:
        greetings+=Greeting*;

Greeting:
        'Hello' name=ID '!';
```

4. As you can see in the pre-defined grammar, two grammar rules already exist, Model and Greeting.
   a. In the Model rule a field greetings (actually a list indicated by the +=) is generated. greetings may contain tokens from the Greeting rule. The list may contain arbitrary many items (indicated by the *)

    b. The Greeting rule expects the keyword Hello and then a name (defined by ID which is a subset of an arbitrary STRING defined in the Terminals definition imported at the top of the grammar). The rule expects a closing exclamation mark!
    c. Actually, let's try out the pre-defined grammar. 🙂
5. Open the manifest of the newly created project and add org.eclipse.equinox.common to the required plugins. This is required to start the use the modeling workflow engine to create your DSL automatically.
6. The Brainfuck.xtext file is located in the package de.cau.cs.rtprak.<login>.turing.text together with a *Modeling Workflow Engine* file, which can be used to configure code generation. Right-click GenerateTuring.mwe2 and select *Run As MWE2 Workflow*. If you get an error message like "*ATT ENTION* It is recommended to use the ANTLR 3 parser generator*" in the console, type *n* and install Antlr from the update site.
7. Now a great amount of Java code should have been generated. Add the new plugins to your Eclipse run configuration and start it.
8. In the new Eclipse instance, create an empty project and add a file: *File New File* and name it test.brainfuck (when asked to add the Xtext nature, hit *Yes*).
9. Use the content assist to generate a valid syntax. Watch the outline while you edit the file. The model objects are created as you type.



10. Now, close your eclipse instance and modify your Brainfuck grammar so that Brainfuck programs become valid. There is more than one way to do that... be creative and/or discuss possible grammars with your fellow students. If you need help with the Xtext syntax, contact the Xtext manual: http://www.eclipse.org/Xtext/
    IMPORTANT: Make sure your rule for [and] is proper!!
11. Re-generate your Brainfuck code and re-run your eclipse instance.

> ⚠ **Class Creation**
>
> Make sure Xtext generates classes for your grammar rules. It is a common mistake to only consume the program tokens and use a list of strings. Your Brainfuck class should contain something like this:
>
> ```
> EList<Command> getCommands();
> ```

12. Use ctrl + space for getting content assist to validate your program syntax. All Brainfuck commands should be visible.

To go further

Congratulations, you've completed the your first Xtext tutorial. However, as this is a beginners tutorial, you barely touched Xtext. If you want to know more, you could...

1. Dive deeper into the rich Xtext documentation: http://www.eclipse.org/Xtext/
2. Perform our second Xtext tutorial: Xtext 1 - Creating a Grammar for an Existing Metamodel
3. Go on with our Xtend tutorial: