



Language Workbench Challenge 2013
Xtext Submission

Version: 1.0 DRAFT - 31. Januar 2013

Karsten Thoms, Johannes Dicks, Thomas Kutz (itemis)

Abstract

The Language Workbench Challenge 2013 (LWC13) is an initiative created by a group of experts at the CodeGeneration 2010 conference¹. The aim is to set a common task² for Language Workbenches³ which is implemented with the different existing alternatives in a comparable way. This document describes in detail how the task is solved with Xtext⁴. Xtext is one of the most well known Language Workbenches and part of the Eclipse Modeling Project⁵.

Testimonial

We would like to thank:

1. *Angelo Houlshould* for initiating and organizing the Language Workbench Challenge. It is his work that allows the Language Workbench Challenge to continue now in its 3rd year.
2. *The Xtext Team* is doing a great job on developing a robust, flexible and easy to use Language Workbench.

¹<http://www.codegeneration.net/cg2010/>

²see <http://www.languageworkbenches.net/> for the detailed description of the LWC11 competition and other submissions

³<http://martinfowler.com/articles/languageWorkbench.html>, <http://blog.efftinge.de/2007/11/definition-of-term-language-workbench.html>

⁴<http://www.xtext.org>

⁵<http://www.eclipse.org/modeling>

Document History

Version 0.1 - 2013-01-28

- Initial creation

Table Of Contents

1	Introduction	5
1.1	Task Description	5
1.2	Xtext Overview	5
1.3	Installing Eclipse Xtext	7
1.4	Workspace Setup	8
1.5	The Survey Application	9
2	Introduction	9
2.1	Task Description	9
2.2	Xtext Overview	10
2.3	Installing Eclipse Xtext	12
2.4	Workspace Setup	13
2.5	The Survey Application	14

1 Introduction

The tasks to solve for the Language Workbench Challenge is relatively trivial. But it has been designed to illustrate as many properties of language workbenches as possible. As such, it can serve as a good introduction to the respective tools.

This tutorial expects that you are somehow familiar with Java and Eclipse and have heard about [EMF](#) and how it works in general before. We start almost at the beginning, but not quite :-)

For this tutorial we will use Xtext 2.4, which is at the moment of writing not finalized, but in its latest stage. Xtext 2.4 will be released with Eclipse Kepler in June 2013⁶.

1.1 Task Description

The LWC13 task <http://www.languageworkbenches.net/images/5/53/Q1.pdf> is to implement a DSL for questionnaires (Questionnaire Language, QL), which basically allows the definition of forms with questions.

1.2 Xtext Overview

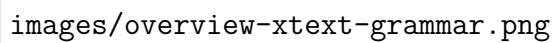
This overview will give you a rough idea about what Xtext⁷ is all about. We will then dive into the details and work on a small project.

In a nutshell, Xtext is a workbench to create and work with textual domain-specific languages (DSLs). It comes as a feature (set of plugins) to the popular Eclipse IDE.

The first thing you will want to do is to create your own domain-specific language (DSL) and specify a *grammar* for it. The grammar file is a plain text file with “.xtext” filename extension, and the grammar within is defined with a BNF like syntax. While you can use any text editor to modify it, Xtext gives you a specialized editor for grammar files. It is aware of the Xtext language, gives you syntax coloring, code completion, and more. To get a first impression see the screenshot of the Xtext grammar file, opened with the Xtext grammar editor, below. It is not required to fully understand the content yet, this will be discussed in the next chapter in detail.

⁶http://wiki.eclipse.org/Kepler/Simultaneous_Release_Plan

⁷<http://www.xtext.org>

The image is a placeholder for a diagram titled 'images/overview-xtext-grammar.png'. It is intended to show the overview of Xtext grammar.

An example for DSL is one that allows you to define entities like “Person”, “Car”, “Book”, and so on. An entity has properties, e.g. a Person has a name, a gender, and a date of birth. A Book has a title, one or more authors, and an ISBN number.

A textual DSL model could look like this, but you could also imagine other syntaxes:

```
1 entity Person {  
2   name : String  
3   gender : m  
4   birthday : Date  
5 }  
6  
7 entity Book {  
8   title: String
```

```
9  authors: Person[]
10 isbnNumber: String
11 }
```

Note that the Property `authors` is of type `Person`, so there can be references between entities. In the Xtext grammar file you specify how you want to define entities and their properties.

Once you have completed your language, you can do that: define some entities, say “Book” and “Person”, together with their respective properties and with proper references between them. The nice thing is that Xtext not only gives you a syntax-driven editor for editing grammar files. Additionally it generates an editor that is specific to the language you have defined. It knows about your language’s keywords and where to place them, it knows about all the syntactical constructs you have made up in your grammar, it includes all the nice stuff like syntax coloring, code completion, validation, and more. For example, if you are at some point where a reference to another entity must be inserted, your DSL editor shows you all the references that would be valid here – according to your language rules – and lets you choose among them. All in all, using the DSL editor generated by Xtext, it is quite easy to establish a text file that adheres to your DSL.

Depending on your language’s type, you could call this text file e.g. a model, a document, a program, or whatever. We will refer to DSL files here as *models* (files).

Consider now that you have created a model. What can you do with it? A typical requirement is to generate an implementation of it in a language like Java, C++, or XML. Or a graphical representation. Or something quite different. This is where code generation comes in. Xtext creates a skeleton code generator for you. Typically you use that code generator as a starting point to produce e.g. Java source code, documentation in, say, DocBook or Wiki format, overview graphics using GraphViz, or any other stuff you need. Xtext offers special support for textual output formats, but it is also possible to generate binaries.

This was only a short outline of some prominent Xtext aspects. It is by far not everything Xtext can do for you, but it should suffice for now. The next chapters will show you in more detail how to work with Xtext.

1.3 Installing Eclipse Xtext

Xtext is a SDK for the [Eclipse](#) IDE. To install it you have two options:

- You can download Xtext separately and install it in your Eclipse instance.

- You can download a specially-crafted complete Eclipse distribution which has Xtext pre-packaged already.

We will take the latter approach here and describe the individual steps:

1. Go to the [Xtext download page](#). Here you can get Eclipse 4.2.x (Helios) including Xtext 2.3.1 along with some tools Xtext depends on. The latter are subsumed here under “Xtext” for simplicity. If you want you can download also a distribution which is already bundled with Eclipse 4.3.0 Kepler, but be aware that this is not finalized until end of June 2013.
2. The Eclipse/Xtext distribution is available for multiple platforms.
 - a) [Linux GTK x86 64 bit](#)
 - b) [Linux GTK x86 32 bit](#)
 - c) [Mac OSX x86 64 bit](#)
 - d) [Windows 64 bit](#)
 - e) [Windows 32 bit](#)

3. Unpack the downloaded archive file in a directory of your choice.

Example (Linux):

```
1 cd /opt/local
2 gzip -dc /download/eclipse-SDK-4.2-Xtext-2.3.1-linux-gtk-x86_64.tar.gz | tar
3 xvf -
```

The archive will be extracted to a new directory named `eclipse`. Before unpacking the archive, please ensure that there is no subdirectory named `eclipse` yet! Different operating systems may require different unpacking methods.⁸

4. Start Eclipse by running the `eclipse` executable in the newly-created `eclipse` directory.

1.4 Workspace Setup

Before we begin, start Eclipse and set up a fresh workspace.

Some settings should be done. Open the workspace settings:

- Windows: Window / Preferences
- Mac: Eclipse / Preferences

Workspace Encoding

File encoding is important to some type of files. It is better that the workspace is set to a

⁸On Windows do not unpack it into a deep directory, since this might cause troubles with long path names.

common encoding to avoid any platform specific encoding. By default the workspace is using platform encoding, which is Cp1252 on Windows and MacRoman on Mac. We will use ISO-8859-1 as a common encoding here.

- Open Eclipse Preferences and go to *General / Workspace*
- Change setting *Text file encoding* to *Other / ISO-8859-1*

Launch Operation

- Open Run/Debug / Launching
- Change “Launch Operation” to “Always launch the previously launched application”

This will allow you re-running the previous launched application by just pressing the Run or Debug button in the Eclipse toolbar, or using keyboard shortcuts. The default settings does not always do what you want.

1.5 The Survey Application

2 Developing the Questionnaire Language

3 Developing the Questionnaire Language

Step 1: Set up initial language implementation - Create Project - Define Grammar - Generate Implementation

4 Testing the Questionnaire Language

- Create Launch Config - Start Runtime - Create Test Project

5 Xbase

- Describe Xbase - Examples for Expressions - Hint: Xbase Tutorial Projects

6 Including Expressions into the QL Language

- Derive Grammar from Xbase - Generate Implementation

7 JVM Model Inference

- Describe the role of JVM Types - Describe the role of the JVM Model Inferrer - Implement JVM Model Inferrer