Mit 'Babysteps' in die XTEXT-XTEND Welt

Teil 9: Das LUECKEN(X)TEXT Modul, eine DSL mit Unterstützung

Anbei noch einmal die Code Listings wie sie der Editor zeigt, also mit Syntax Highlites. Passend zum Artikel im Eclipse Magazin.

von Ulrich Merkel

Lxt3: gleiche Funktionen wie bei Lxt2, aber jetzt mit der Nutzung von LxtCore

Der, der den Code generiert: Lxt3CodeGenerator.xtend

```
/*
 * generated by Xtext: Lxt3Generator.xtend
 */
package de.ulrichmerkel.lueckenxtext.lxt3.generator

import de.ulrichmerkel.lueckenxtext.lxt3.lxt3.Entity
import de.ulrichmerkel.lueckenxtext.lxt3.lxt3.KeyValuePair
import de.ulrichmerkel.lueckenxtext.lxt3.lxt3.KeyValuePair1
import de.ulrichmerkel.lueckenxtext.lxt3.lxt3.KeyValuePair2
import de.ulrichmerkel.lueckenxtext.lxt3.lxt3.KeyValuePair3
import de.ulrichmerkel.lueckenxtext.lxt3.lxt3.Mergefile
import de.ulrichmerkel.lueckenxtext.lxt3.lxt3.Model
import de.ulrichmerkel.lueckenxtext.lxt3.lxt3.Module
import de.ulrichmerkel.lueckenxtext.lxt3.util.LxtCore
import org.eclipse.emf.ecore.EObject
import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.IFileSystemAccess
import org.eclipse.xtext.generator.IGenerator

class Lxt3Generator implements IGenerator {

override void doGenerate(Resource resource, IFileSystemAccess fsa) {
	var String scriptFileName = resource.getURI().lastSegment.toString();
	var String outFileBase = scriptFileName.substring(0,
scriptFileName.lastIndexOf("."));
	doPreparation(resource, resource.contents.head as Model)
//	fsa.generateFile(outFileBase + '.3.CODE'
//		, makeTheCodeAutocopy(resource.contents.head as Model))
	fsa.generateFile(outFileBase + '.3.CODE'
		, makeTheCode(resource.contents.head as Model))
}

def makeTheCodeAutocopy(Model sm) '''
«LxtCore::lxtSetClipboard(makeTheCode(sm).toString)»
'''

def makeTheCode(Model sm) '''
«FOR Module modul : sm.modules»
«switch modul {
		Mergefile: {addMergeFile(modul)}
	}»
«ENDFOR»
'''

def addMergeFile(Mergefile mf) {
	LxtCore::loadLxtWorkText(mf.name)
	for (KeyValuePair kvp : mf.placeholderPairs) {
	  LxtCore::replaceLxtWorkText(retKvpRealName(kvp),retKvpValue(kvp))
	}
	if (LxtCore::getListPlaceholders.length > 0) {
		System.out.println("Open placeholders: " +
LxtCore::getListPlaceholders)
	}
	LxtCore::getLxtWorkText
}

def doPreparation(Resource resource, Model sm) {
	if (sm.mergedir != null) LxtCore::setLxtDirPath(sm.mergedir)
	if (sm.mergeext != null) LxtCore::setLxtExtension(sm.mergeext)
	return ""
}
// returning the "real" names of the placeholders
```

```
        def dispatch retKvpRealName(KeyValuePair1 kvp1) { return(kvp1.name) }
        def dispatch retKvpRealName(KeyValuePair2 kvp2) { return("reffield--
"+kvp2.name) }
        def dispatch retKvpRealName(KeyValuePair3 kvp3) { return("refentity--
"+kvp3.name) }

// returning the values for replacement
        def dispatch retKvpValue(KeyValuePair1 kvp1) { return(kvp1.value) }
        def dispatch retKvpValue(KeyValuePair2 kvp2) {
   return(genParentname(kvp2.reffield.eContainer()).toString.trim()
          + "." + kvp2.reffield.name)
        }
        def dispatch retKvpValue(KeyValuePair3 kvp3) { return(kvp3.refentity.name) }

        def genParentname(EObject container) '''«switch container {
                Entity: {
                        container.name
                }
        }»    '''


    }
```

Der, der uns die verfügbaren Dateien anzeigt: Lxt3ProposalProvider.xtend

```
/*
 * generated by Xtext: Lxt3ProposalProvider.xtend
 */
package de.ulrichmerkel.lueckenxtext.lxt3.ui.contentassist

import de.ulrichmerkel.lueckenxtext.lxt3.util.LxtCore
import org.eclipse.emf.ecore.EObject
import org.eclipse.xtext.Assignment
import org.eclipse.xtext.EcoreUtil2
import org.eclipse.xtext.RuleCall
import org.eclipse.xtext.ui.editor.contentassist.ContentAssistContext
import org.eclipse.xtext.ui.editor.contentassist.ICompletionProposalAcceptor

import de.ulrichmerkel.lueckenxtext.lxt3.lxt3.Model

class Lxt3ProposalProvider extends AbstractLxt3ProposalProvider {

override completeMergefile_Name(EObject model, Assignment assignment,
        ContentAssistContext context, ICompletionProposalAcceptor acceptor) {
        val rootContainer = EcoreUtil2.getRootContainer(model) as Model
        LxtCore::setLxtDirPath(rootContainer.mergedir?:"..\\!lueckenxtext_home")
        LxtCore::setLxtExtension(rootContainer.mergeext?:"txt")
        completeRuleCall(((assignment.getTerminal()as RuleCall)), context, acceptor);
        for (String fileId : LxtCore::getListFileIds) {
                acceptor.accept(createCompletionProposal(fileId , context))
        }
}

}
```

Der, der den Fehlermarker erzeugt: Lxt3Validator.xtend

```
/*
 * generated by Xtext: Lxt3Validator.xtend
 */
package de.ulrichmerkel.lueckenxtext.lxt3.validation

import de.ulrichmerkel.lueckenxtext.lxt3.lxt3.Lxt3Package
import de.ulrichmerkel.lueckenxtext.lxt3.lxt3.Mergefile
import de.ulrichmerkel.lueckenxtext.lxt3.lxt3.Model
import de.ulrichmerkel.lueckenxtext.lxt3.util.LxtCore
import org.eclipse.xtext.validation.Check

class Lxt3Validator extends AbstractLxt3Validator {
  public static val DEF_MERGEFILE_PLACEHOLDERS = 'noPlaceholderSet'
//
        @Check
        def checkMergefileVariablesShouldNotBeEmpty(Mergefile mergefile) {
                if(mergefile.placeholderPairs.size < 1) {
                        error('values set should not be empty',

        Lxt3Package.Literals.MERGEFILE__PLACEHOLDER_PAIRS,
                                        Lxt3Validator.DEF_MERGEFILE_PLACEHOLDERS,
                                        mergefile.name)

                }
```

```
	}
}
```

Der, der die Angabe für uns "vorschreibt": Lxt3QuickfixProvider.xtend

```
/*
* generated by Xtext: Lxt3QuickfixProvider.xtend
*/
package de.ulrichmerkel.lueckenxtext.lxt3.ui.quickfix

import de.ulrichmerkel.lueckenxtext.lxt3.lxt3.Model
import de.ulrichmerkel.lueckenxtext.lxt3.util.LxtCore
import de.ulrichmerkel.lueckenxtext.lxt3.validation.Lxt3Validator
import org.eclipse.xtext.EcoreUtil2
import org.eclipse.xtext.ui.editor.quickfix.DefaultQuickfixProvider
import org.eclipse.xtext.ui.editor.quickfix.Fix
import org.eclipse.xtext.ui.editor.quickfix.IssueResolutionAcceptor
import org.eclipse.xtext.validation.Issue

class Lxt3QuickfixProvider extends DefaultQuickfixProvider {

	@Fix(Lxt3Validator::DEF_MERGEFILE_PLACEHOLDERS)
	def addMergefileValues(Issue issue, IssueResolutionAcceptor acceptor) {
		acceptor.accept(issue, 'variables set empty', 'Generate variable
set.', '')
		[ element, context |
			val rootContainer = EcoreUtil2.getRootContainer(element) as
Model

	LxtCore::setLxtDirPath(rootContainer.mergedir?:"..\\!lueckenxtext_home")
			LxtCore::setLxtExtension(rootContainer.mergeext?:"txt")
			val xtextDocument = context.xtextDocument
			var replaceText = " placeholders=mergefilename=" + '"' +
issue.data.get(0) + '"'
			replaceText += makeDefaultPlaceholderGroup(issue.data.get(0))
			xtextDocument.replace(issue.offset + issue.getLength, 0,
replaceText)
		]
	}

	def String makeDefaultPlaceholderGroup(String fileId) {
		var returnText = System.getProperty("line.separator")
		for (String placeholder :
LxtCore::getListFilePlaceholders(fileId).sort)
		{
			returnText += makeDefaultPlaceholderPair(placeholder)
		}
		return (returnText)
	}
// routine to set up a default placeholder definition pair (will be enhanced
shortly)
		def String makeDefaultPlaceholderPair(String placeholder) {
		var returnText = ""
		if (placeholder != null) {
			returnText += "," + retPlaceholderDeclaration(placeholder)
		}
		return (returnText)
	}

	def retPlaceholderDeclaration(String placeholder){
		if (placeholder.startsWith("reffield--")) return placeholder + '=' +
System.getProperty("line.separator")
		if (placeholder.startsWith("refentity--")) return placeholder + '=' +
System.getProperty("line.separator")
		return placeholder + '=""' + System.getProperty("line.separator")
		}
}
```

Anlässlich des Eclipse DemoCamps Frankfurt am 2.12.2014 Flexibilität gezeigt

In einem 20 Minuten Vortrag habe ich auf dem DemoCamp die Nutzung von Luecken(x)text gezeigt. Innerhalb der letzten 5 Minuten wurde dabei die DSL um einen neuen Referenztyp "refpaintedfield--" erweitert.

In der Grammatik wurde eine Quelle für die Referenz (in "Entity") eingefügt und dann die Angaben von "Field" zu "PaintedField" kopiert, als letztes wurde die zusätzliche Alternative für KeyvaluePairs noch eingefügt.

```
Entity: 'entity' name=ID '.' model=ID
('paintedfields=' paintedfields += PaintedField?
    (',' paintedfields+= PaintedField)*
)?
'fields=' fields += Field? (',' fields+= Field)* ;
Field: name=ID ;

KeyValuePair:
       (KeyValuePair1 | KeyValuePair2 | KeyValuePair3 | KeyValuePair4 );
KeyValuePair4: 'refpaintedfield--'name=ID '='
refpaintedfield=[PaintedField|QualifiedName];
PaintedField: name=ID ;
```

Und im QuickfixProvider ergänzen wir sinngemäß:

```
if (placeholder.startsWith("refpaintedfield--")) return placeholder + '=' +
System.getProperty("line.separator")
```

Noch einfacher ist die Erweiterung des Generators. Da wir hier ja bereits das "dispatch" Schlüsselwort verwenden, brauchen wir nur die beiden Routinen für "reffield" zu kopieren und die beteiligten Informationen umzubenennen.

```
def dispatch retKvpRealName(KeyValuePair4 kvp4) {
    return("refpaintedfield--"+kvp4.name) }
def dispatch retKvpValue(KeyValuePair4 kvp4) {
   return(genParentname(kvp4.refpaintedfield.eContainer()).toString.trim()
         + "." + kvp4.refpaintedfield.name)
       }
```

Fertig ist die erweiterte DSL.

Links & Literatur

[1] "reducing cross reference to current DSL file" http://www.eclipse.org/forums/index.php/t/635607/

[2] "XTEXT, Buckminster and site.p2 w/o source" http://www.eclipse.org/forums/index.php/t/637879/

[3] Lorenzo Bettini (August 2013) "Implementing Domain-Specific Languages with Xtext and Xtend" www.packtpub.com ISBN 978-1-78216-030-4

[4] Luna DSL Package download https://www.eclipse.org/downloads/packages/

[5] Material zum Mitmachen: !lueckenxtext_luna-part2-all-in-one.zip auf www.uli-merkel.de/uli-xtext

[6] http://entwickler.de/press/DSL-mit-XtextXtend-177472 bzw. -177471

LxtCore: Auslagerung der Routinen für Vorlagedateien

```
// general lueckenxtext utilities: LxtCore.xtend
package de.ulrichmerkel.lueckenxtext.lxt3.util

import java.awt.Toolkit
import java.awt.datatransfer.Clipboard
import java.awt.datatransfer.StringSelection
import java.io.File
import java.io.FileReader
import java.util.regex.Matcher
import java.util.regex.Pattern

import static extension com.google.common.io.CharStreams.*

public class LxtCore {
// some static variables
       private static var lxtDirPathPrefix = "" // to adjust nested levels of
directories
       private static var lxtDirPath = "..\\!lueckenxtext_home" // directory of
lueckenxtext template files
       private static var lxtExtension = "txt" // extension of lueckenxtext template
files
       private static var lxtWorkText = "" // internal workBuffer
       private static var lxtDirPathPrefixInit = "" // to adjust nested levels of
directories
```

```
	private static var lxtDirPathInit = "..\\!lueckenxtext_home" // directory of
lueckenxtext template files
	private static var lxtExtensionInit = "txt" // extension of lueckenxtext
template files


	static def getLxtDirPath() {return lxtDirPath}
	static def getLxtExtension() {return lxtExtension}
	static def getLxtWorkText() {return lxtWorkText}

	static def checkLxtDirpath(String dirPath) {
// check if dirPath points to an existing directory
		var testDir = new File(dirPath)
		if (!testDir.exists || !testDir.isDirectory) { return false }
		return true
	}
	static def checkLxtFile(String fileId) {
// check if fileID points to an existing file
		var testFile = new File(getLxtFilePath(fileId))
		if (!testFile.exists || !testFile.isFile) { return false }
		return true
	}

	static def getLxtFilePath(String fileID) {
// returns the complete filepath according to a fileID
		return lxtDirPathPrefix + lxtDirPath + "\\" + fileID + "." +
lxtExtension
	}
	static def setLxtDirPath(String newDirPath) {
// setting value only if check is passed
		if (checkLxtDirpath(lxtDirPathPrefix + newDirPath)) {lxtDirPath =
newDirPath}
		return lxtDirPath
	}
	static def setLxtDirPathInit() {
// setting value only if check is passed
		lxtDirPath = lxtDirPathInit
		return lxtDirPath
	}
	static def setLxtDirPathPrefix(String newDirPathPrefix) {
// setting value only if check is passed
		if (checkLxtDirpath(newDirPathPrefix+lxtDirPath)) {lxtDirPathPrefix =
newDirPathPrefix}
		return lxtDirPath
	}
	static def setLxtDirPathPrefixInit() {
// setting value only if check is passed
		lxtDirPathPrefix = lxtDirPathPrefixInit
		return lxtDirPath
	}
	static def setLxtExtension(String newExtension) {
		lxtExtension = newExtension
		return lxtExtension
	}
	static def setLxtExtensionInit() {
		lxtExtension = lxtExtensionInit
		return lxtExtension
	}
	static def setLxtWorkText(String newText) {
// set contents of lxtWorkTest
		lxtWorkText = newText
		return lxtWorkText
	}
	static def loadLxtWorkText(String fileID) {
// load contents of file into lxtWorkText Buffer
		lxtWorkText = "";
		val File incfile = new File(getLxtFilePath(fileID))
		if (incfile.exists) {
			for (String line : new FileReader(incfile).readLines)
			lxtWorkText = lxtWorkText + line +
System.getProperty("line.separator")
		}
		return lxtWorkText
	}

	static def replaceLxtWorkText(String placeholderName, String replaceText) {
// replace placeholders in the lxtWorkText Buffer
		lxtWorkText = lxtWorkText.replaceAll("\\$\\{" + placeholderName +
"\\}", replaceText);
	}
```

```
	public static def getListFileIds() {
// return a list of all files found matching the selection criteria
		var allFileIds = <String>newHashSet()
		for(File x: new File(lxtDirPathPrefix + lxtDirPath).listFiles){
			if (x.isFile && x.getName.endsWith("."+ lxtExtension)) {

	allFileIds.add(x.getName.substring(0,x.getName.indexOf(".")))
			}
		}
		return allFileIds
	}
	public static def getListPlaceholders() {
// return a list of placeholders found in the lxtWorkText Buffer
		return makeListPlaceholders()
	}

	public static def getListFilePlaceholders(String fileId) {
// return a list of placeholders found in the contents of the file
		loadLxtWorkText(fileId)
		return makeListPlaceholders()
	}

	private static def makeListPlaceholders() {
// return a list of placeholders found in the lxtWorkText Buffer
		var allPlaceholders = <String>newHashSet()
		var Pattern pattern = Pattern.compile("\\$\\{([^}]*)\\}");
		var Matcher matcher = pattern.matcher(lxtWorkText);
		while (matcher.find()) {
			allPlaceholders.add(matcher.group(1))
		}
		return allPlaceholders
	}

	static def lxtSetClipboard(String workstring) {
// place the string into the clipboard, returning the string unchanged
		try {
			var Toolkit toolkit = Toolkit.getDefaultToolkit();
			var Clipboard clipboard = toolkit.getSystemClipboard();
			var StringSelection strSel = new StringSelection(workstring);
			clipboard.setContents(strSel, null);
		} catch (Exception e) {
			e.printStackTrace();
		}
		return workstring
	}

}
```

Bei der Nutzung dieser Klasse muss berücksichtigt werden, dass der Zugriff auf statische Elemente in Xtend durch "::" statt wir in Java gewohnt durch "." gekennzeichnet sind:

```
import de.ulrichmerkel.lueckenxtext.util.LxtCore

for (String fileId : LxtCore::getListFileIds) {
```

Ende

"The Hitchhiker's Guide to XTEXT-XTEND": Scoping nur innerhalb der DSL Datei

Dazu erstellen wir im Package "de.ulrichmerkel.lueckenxtext.lxt3.util" für das Interface IGlobalScopeProvider die Klasse **NullGlobalScopeProvider.java**:

```
// ##### unser neuer ScopeProvider (NullGlobalScopeProvider.java)
package de.ulrichmerkel.lueckenxtext.lxt3.util;

import org.eclipse.emf.ecore.EReference;
import org.eclipse.emf.ecore.resource.Resource;
import org.eclipse.xtext.resource.IEObjectDescription;
import org.eclipse.xtext.scoping.IGlobalScopeProvider;
import org.eclipse.xtext.scoping.IScope;

import com.google.common.base.Predicate;
```

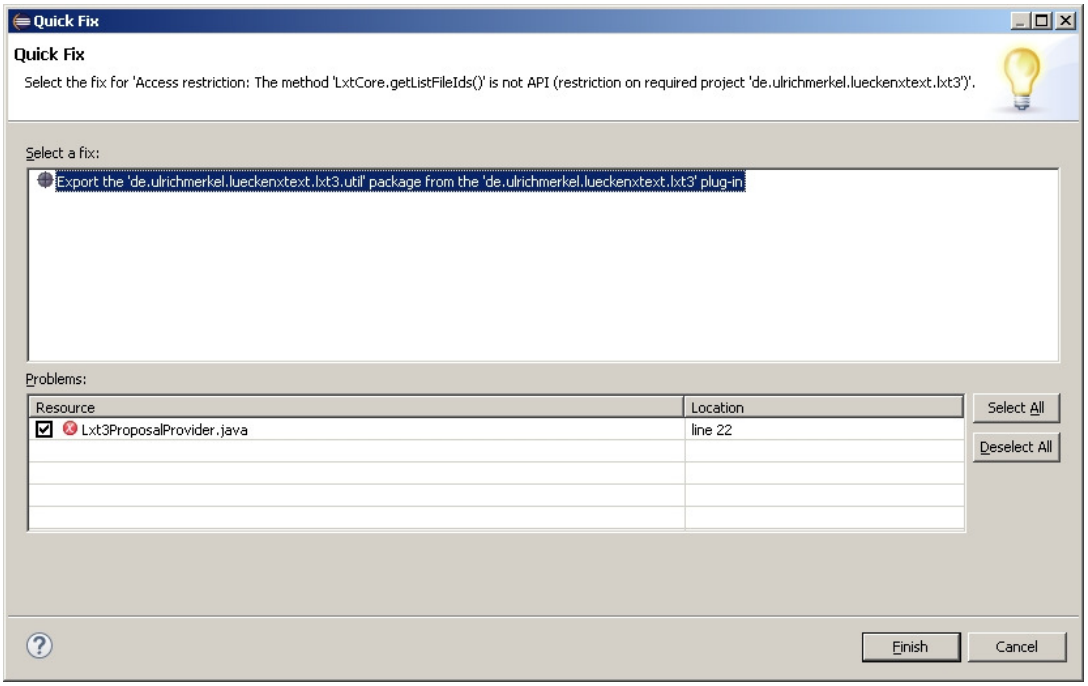
```
public class NullGlobalScopeProvider implements IGlobalScopeProvider {

	@Override
	public IScope getScope(Resource context, EReference reference,
			Predicate<IEObjectDescription> filter) {
		return IScope.NULLSCOPE;
	}

}
```

Im Verzeichnis "de.ulrichmerkel.lueckenxtext.lxt3" liegt die Datei **Lxt3RuntimeModule.java**, indem wir die eigentliche Implementierung erweitern, damit zur Laufzeit das geänderte Scoping benutzt wird:

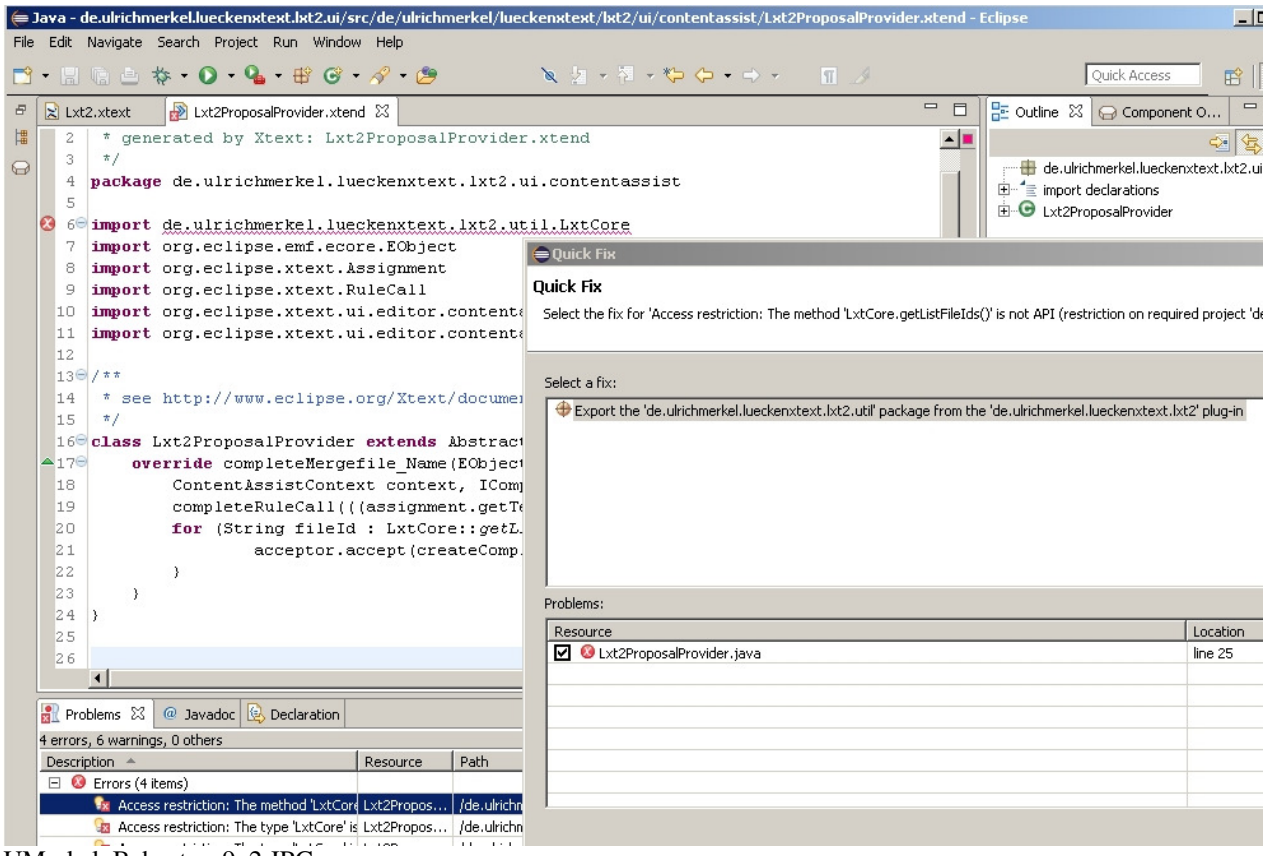
```
/*
Lxt3RuntimeModule.java generated by Xtext
*/
package de.ulrichmerkel.lueckenxtext.lxt3;

import org.eclipse.xtext.scoping.IGlobalScopeProvider;
import de.ulrichmerkel.lueckenxtext.lxt3.util.NullGlobalScopeProvider;

/**
 * Use this class to register components to be used at runtime / without
the Equinox extension registry.
 */
public class Lxt3RuntimeModule extends
de.ulrichmerkel.lueckenxtext.lxt3.AbstractLxt3RuntimeModule {
	@Override
	public Class<? extends IGlobalScopeProvider>
bindIGlobalScopeProvider() {
		return NullGlobalScopeProvider.class;
	}
}
```

Ende

BILDÜBERSICHT

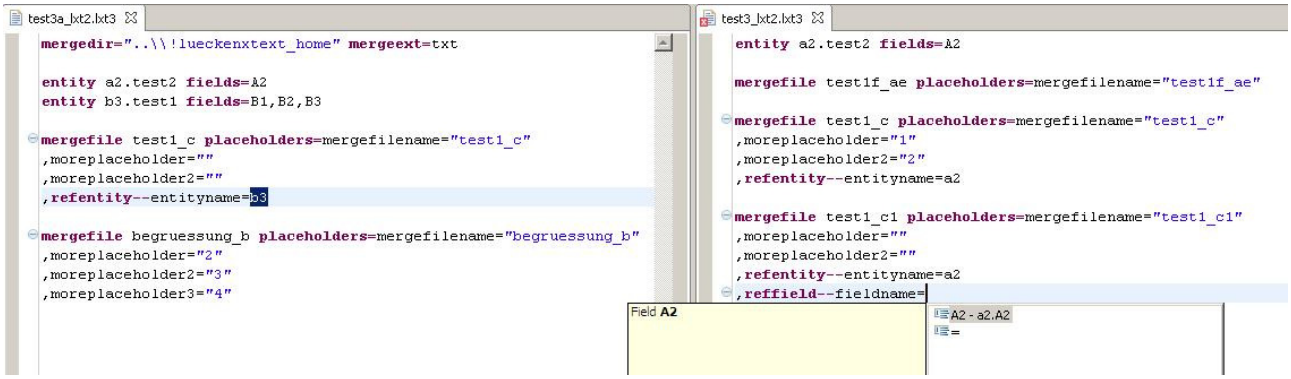


UMerkel_Babysteps9_1.JPG

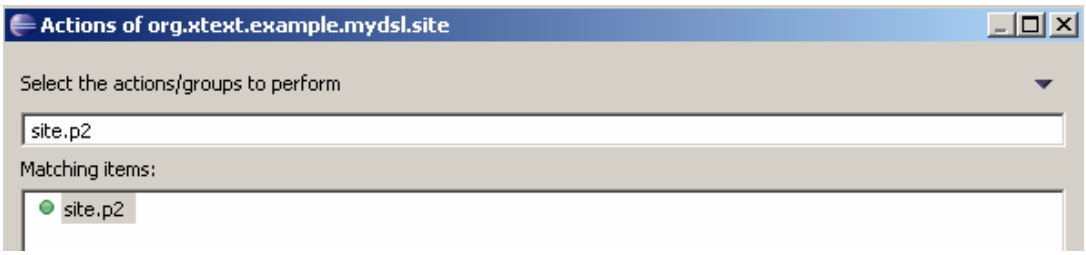


UMerkel_Babysteps9_2.JPG

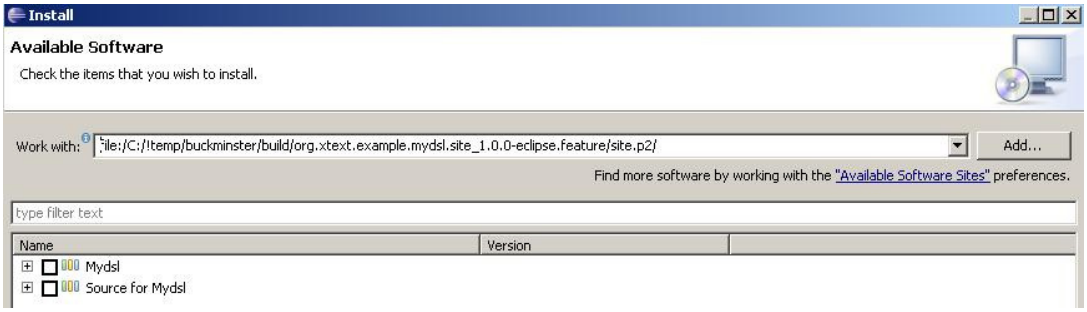
UMerkel_Babysteps9_3.JPG



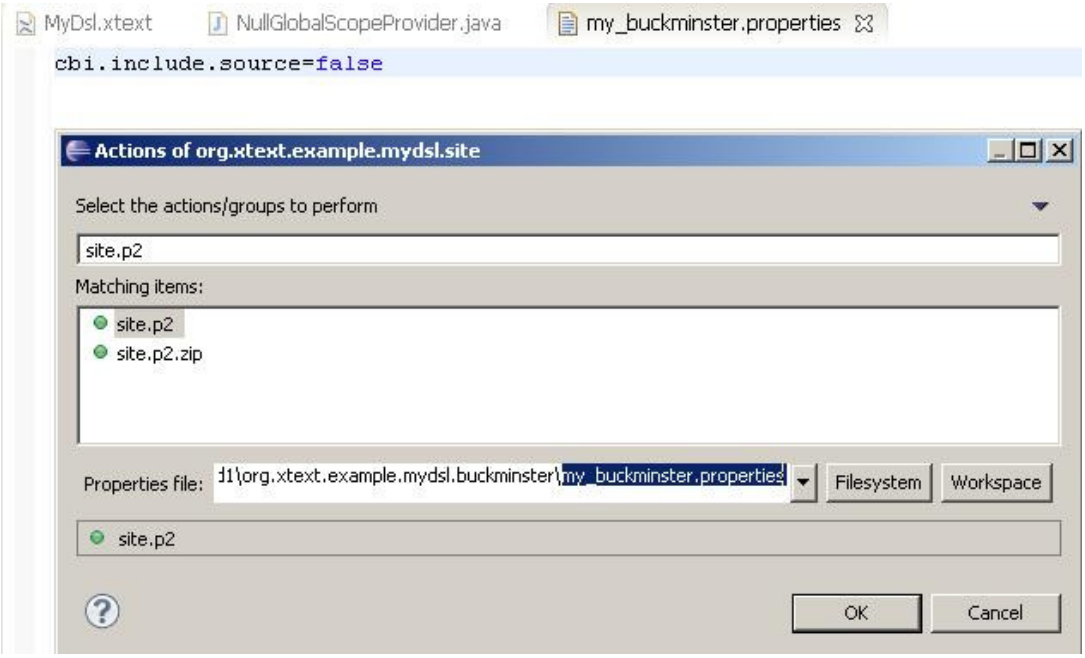
UMerkel_Babysteps9_4.JPG



UMerkel_Babysteps4_7.JPG



UMerkel_Babysteps4_8.JPG



UMerkel_Babysteps4_9.JPG