Mit 'Babysteps' in die XTEXT-XTEND Welt

Teil 5: Initialerstellung einer Erfassungsmaske und sehr viel mehr

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

von Ulrich Merkel

Also mal ganz einfach anfangen

Die DSL, für unser Beispiel lautet ganz schlicht:

```
form simpleform { entity T1.Model2  fields=Field1,Field2 }
```

Das Resultat des Formpainters wird als verschlüsselter Text abgespeichert, wobei eine dieser Textzeilen auch einer Zeile in der Eingabemaske entspricht.

Dieses Teilstück einer Exportdatei uns seine Entsprechung im Formpainter:

```
<DAT name="FORMPIC"
      xml:space='preserve'>&uFRM;TYP=E&uSEP;NAM=T1.MODEL2&uSEP;WID=
      50&uSEP;HEI=3&uFRM;
&uFRM;TYP=L&uSEP;NAM=FIELD1.T1&uSEP;WID=20&uSEP;HEI=1&uFRM;
      &uFRM;TYP=F&uSEP;NAM=FIELD1&uSEP;WID=20&uSEP;HEI=1&uFRM;
&uFRM;TYP=L&uSEP;NAM=FIELD2.T1&uSEP;WID=20&uSEP;HEI=1&uFRM;
      &uFRM;TYP=F&uSEP;NAM=FIELD2&uSEP;WID=20&uSEP;HEI=1&uFRM;
</DAT>
```

```
def makeUformFormpic(Form form) {
'''«FOR Entity entity : form.entities»
&uFRM;TYP=E&uSEP;NAM=«entity.name.toUpperCase».«entity.model.toUppe
rCase»&uSEP;WID=50&uSEP;HEI=«1+entity.fields.length»&uFRM;
«FOR Field field : entity.fields»
&uFRM;TYP=L&uSEP;NAM=«field.name.toUpperCase».«entity.name.toUpperC
ase»&uSEP;WID=20&uSEP;HEI=1&uFRM;
&uFRM;TYP=F&uSEP;NAM=«field.name.toUpperCase»&uSEP;WID=20&uSEP;HEI=
1&uFRM;
«ENDFOR»
«ENDFOR»'''}
```

Und noch ein wenig mehr für Hilfestellung die Praxis

```
form simpleform_plus description="for => Paul"
todo="according to meeting AB-140202,
this simple form plus should provide some basic routines"{
      entity T1.Model2  fields=Field1,Field2
   }
```

Zeitersparnis: weit mehr als eine halbe Stunde für eine Eingabeform

```
form form2f2 description="descriptionText2" todo="A sample
todoText12
as a
multiline issue" LPMX=«;just a small
;text for the LPMX trigger» {
      headerfields=Test1,Test2,header2
      entity Name.Model  fields=Fieldname1,Fieldname2
```

```
    entity Name3.model2 fields=Feld1A,Feld2,Feld3,Name
    entity uform.dict fields=ulabel,udescr
    trailerfields=Trail1,Trail2
    breakframe b_1 fields=b_1_f1,b_1_f2,b_1_f3
    breakframe b_2 fields=b_1_f1,b_1_f2,b_1_f3
}
```

DSL Unterstützung ermöglicht die Implementierung "dummer" Ladeprogramme

```
Dieser Text ersetzt den bisherigen Kommentar
der Maske mit dem Namen DULF_TOY
mit dem hier stehenden Text
ULFACTION=SET;TARGET=-COM;OBJECT=FORM;FORMNAME=DULF_TOY
**********
Dieser Text wird an den bestehenden Kommentar
angefügt
ULFACTION=APP;TARGET=-COM;OBJECT=FORMENTITY;FORMNAME=DULF_TOY;ENTITYNAME=DUMMYENT
```

Ampelsteuerung in der Coding Nacht

Alle zwei Wochen versammeln sich Codierenthusiasten im 5. Stock des Juridicum der Uni Frankfurt, um zwischen 19:00 und 01:00 gemeinsam zu programmieren. Das aktuelle Motto lautete:

8. Coding Night: code together // share knowledge // share code

Wir hatten eine nette USB Ampel [7] zur Verfügung, die bereits eine grundlegende Steuerung über einen Server auf der Basis von Node.js und Coffee-Script hatte [6].

In einer Kooperationsarbeit sollte nun die bestehende Implementierung erweitert werden. Während ein Frontend in C# erstellt wurde, habe ich mich daran gemacht, für eine fiktive Fachabteilung zur Programmierung einer Ampelsteuerung eine DSL bereitzustellen. Die Ausgabe soll ein coffee-script werden, das in den Server einbezogen werden kann.

Damit haben wir eine saubere Trennung zwischen Fachabteilung und "Technik", die beiden Seiten die Arbeit erleichtert.

Gemessen an den strengen Anforderungen der Technik für die Programmierung, die auch sehr sensibel auf Einrückungen reagiert (hier die Datei test2.1.coffee) ... :

```
module.exports = (red, yellow, green, all) ->
  commands=[]
  commands.push(all(off,0))
  commands.push(red(on,20000))
  commands.push(yellow(on,5000))
  commands.push(all(off,0))
  commands.push(green(on,9000))
  commands.push(red(on,15000))
  commands.push(red(off,0))
  commands
```

... hat die Fachabteilung wenige Probleme mit der dazugehörigen DSL test2.ampel1, die im Gegensatz der internationalisierten Implementierung des Ampel-Servers noch dazu in Deutsch lediglich die Farbe, den Schaltungsstand und die darauf folgende Wartezeit enthält:

```
alle aus 0
rot an 20
gelb an 5
```

```
alle aus 0
grün an 9
rot an 15
rot aus 0
```

Da es recht häufig vorkommt, dass nach einer gewissen Dauer die Farbe wieder ausgeschaltet werden muss, liefert ein für die DSL in Eclipse erstelltes Template "an_aus" einen gewissen Komfort:

```
${farbe} an ${timeout}
${farbe} aus 0
```

Die benötigten XTEXT bzw. XTEND Dateien sind klein und schnell zu erstellen:

```
// ampel1.xtext
grammar org.codingnight.ampel1.Ampel1 with
org.eclipse.xtext.common.Terminals

generate ampel1 "http://www.codingnight.org/ampel1/Ampel1"
Model:            ampelzeilen+=Ampelzeile*;
Ampelzeile: farbe=('rot'|'gelb'|'grün'|'alle')
            setze=('an'|'aus') timeout=INT;
```

```
// Ampel1Generator.xtend
package org.codingnight.ampel1.generator

import org.codingnight.ampel1.ampel1.Ampelzeile
import org.codingnight.ampel1.ampel1.Model
import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.IFileSystemAccess
import org.eclipse.xtext.generator.IGenerator

class Ampel1Generator implements IGenerator {
override void doGenerate(Resource resource, IFileSystemAccess fsa)
{
      var String scriptFileName =
resource.getURI().lastSegment.toString();
      var String outFileBase = scriptFileName.substring(0,
scriptFileName.lastIndexOf("."));
      fsa.generateFile(outFileBase + '.1.coffee',
makeCoffeeCode(resource.contents.head as Model))
}
def makeCoffeeCode(Model sm) '''
module.exports = (red, yellow, green, all) ->
  commands=[]
«FOR Ampelzeile az : sm.ampelzeilen»
            «"  "»commands.push(«convertFarbeCoffee(az.farbe)»(
«convertSetzeCoffee(az.setze)»,«convertTimeoutCoffee(az.timeout)»))
«ENDFOR»
  commands
'''
def convertFarbeCoffee(String farbe) {
      if (farbe=='grün') return('green')
      if (farbe=='rot') return('red')
      if (farbe=='gelb') return('yellow')
      return('all')
}
def convertSetzeCoffee(String setze) {
      if (setze=='an') return('on')
      return('off')
}
def convertTimeoutCoffee(int timeout) {
      return timeout*1000
}
```

```
}
```

Zusätzlicher Komfort kann in die DSL noch schrittweise eingearbeitet werden. Zum Beispiel kann statt des oben angegebenen Editor Templates auch die DSL selbst erweitert werden. In der Grammatik wird das setze= um die Option "an_aus" erweitert:

```
Ampelzeile: farbe=('rot'|'gelb'|'grün'|'alle')
            setze=('an'|'aus'|'an_aus') timeout=INT;
```

Der Generator setzt dann diese eine Zeile in die zwei zugehörigen coffee Befehlszeilen um:

```
def makeCoffeeCode(Model sm) '''
module.exports = (red, yellow, green, all) ->
  commands=[]
«FOR Ampelzeile az : sm.ampelzeilen»
      «IF az.setze == "an_aus"»
      «"  "»commands.push(«convertFarbeCoffee(az.farbe)»(
«convertSetzeCoffee("an")»,«convertTimeoutCoffee(az.timeout)»))
      «"  "»commands.push(«convertFarbeCoffee(az.farbe)»
(«convertSetzeCoffee("aus")»,«convertTimeoutCoffee(0)»))
      «ELSE»
      «"  "»commands.push(«convertFarbeCoffee(az.farbe)»
(«convertSetzeCoffee(az.setze)»,«convertTimeoutCoffee(az.timeout)»))
    «ENDIF»
«ENDFOR»
  commands
'''
```

Für die Einbindung in die Ampelsteuerung [6] und die Realisierung der "Schlafenszeit" sorgt der script executor.coffee:

```
CoolBeans = require 'CoolBeans'
container = new CoolBeans require '../production-module'

lights = container.get('lights').for 407571

createCommands = require './test2.1.coffee'
lightCommand = (color) ->
  (state, delay) ->
    (delayedCallback) -> lights[color] state,
setTimeout(delayedCallback, delay)

commands = createCommands(
  lightCommand 'red'
  lightCommand 'yellow'
  lightCommand 'green'
  lightCommand 'all'
)

execCommandAtIndex = (index) ->
  unless index >= commands.length
    command = commands[index]
    command -> execCommandAtIndex index + 1

execCommandAtIndex 0
```

Ende