1. Conversion between Xtext and XMI formats

To use a model specified with Xtext as the input of a code generator (e.g., built with Acceleo) or a model transformation (e.g., built with Henshin or ATL), you need to convert the Xtext file into an XMI file. Then, you can use this XMI file as the input to the generator or transformation.

The following code snippet shows how to do this conversion programmatically in Java. In the code snippet, you will have to substitute "DSL" in DSLPackage and DSLStandloneSetup by the name of your DSL (e.g., WizardPackage). Moreover, you need to generate the Java classes from the ecore meta-model.

```
// register URI of ecore meta-model
EPackage.Registry.INSTANCE.put(DSLPackage.eNS_URI, DSLPackage.eINSTANCE);

// create resource for xtext file
Injector injector = new DSLStandaloneSetup().createInjectorAndDoEMFRegistration();
ResourceSet xtextRS = injector.getInstance(XtextResourceSet.class);
XtextResource xtextInput = (XtextResource)xtextRS.getResource(URI.createURI("file:/C:/file.dsl"), true);
EcoreUtil.resolveAll(xtextInput);

// create empty xmi resource
ResourceSet xmiRS  = new ResourceSetImpl();
Resource xmiOutput = xmiRS.createResource(URI.createURI("file:/C:/file.xmi"));

// save xtext resource in xmi resource
xmiOutput.getContents().add(xtextInput.getContents().get(0));
xmiOutput.save(null);
```

Likewise, the following code snippet shows how to convert an XMI file (e.g., generated by a transformation) into an Xtext file.

```
// register URI of ecore meta-model
EPackage.Registry.INSTANCE.put(DSLPackage.eNS_URI, DSLPackage.eINSTANCE);

// create empty xtext resource
Injector injector = new DSLStandaloneSetup().createInjectorAndDoEMFRegistration();
ResourceSet xtextRS = injector.getInstance(XtextResourceSet.class);
XtextResource xtextOutput = (XtextResource)xtextRS.createResource
   (URI.createURI("file:/C:/file.dsl"));

// create resource for xmi file
ResourceSet xmiRS  = new ResourceSetImpl();
Resource xmiInput = xmiRS.createResource(URI.createURI("file:/C:/file.xmi"), true);

// save xmi resource in xtext resource
xtextOutput.getContents().add(xmiInput.getContents().get(0));
xtextOutput.setValidationDisabled(true);
xtextOutput.save(null);
```

2. Creation of popup menus in Eclipse

Eclipse permits creating popup menus (also called context menus) which are shown by right-clicking on files with a given extension. This can be used, e.g., to trigger the execution of a code generator for a given model. The following steps show how to define a popup menu for files with a given extension.

1. Create a project of type “plugin”.

2. Open the file MANIFEST.MF that is inside the project, go to tab “Overview”, and select the check “This plug-in is a singleton”. Then, go to tab “Dependencies” and add the following dependencies:
   - org.eclipse.core.commands
   - org.eclipse.swt

3. Create inside the project a class that extends from org.eclipse.core.commands.AbstractHandler. The class must implement a method called “execute”, which will be executed when the popup menu is selected. The following code snippet shows a possible implementation of this method.

```
@Override
 public Object execute(ExecutionEvent event) throws ExecutionException {
   // obtain selected file
   IFile file = null;
   ISelection selection = HandlerUtil.getCurrentSelection(event);
   if (selection instanceof IStructuredSelection) {
     Object first = ((IStructuredSelection)selection).getFirstElement();
     file = (IFile)Platform.getAdapterManager().getAdapter(first, IFile.class);
     if (file == null)
       if (first instanceof IAdaptable)
         file = (IFile)((IAdaptable)first).getAdapter(IFile.class);
   }

   try {
     // load selected file in a resource
     ResourceSet rs = new ResourceSetImpl();
     Resource resource = rs.createResource(URI.createURI(file.getRawLocationURI().toString()));
     resource.load(null);

     // perform action
     MessageDialog.openInformation(HandlerUtil.getActiveWorkbenchWindowChecked(event).getShell(),
        "substitute dialogbox by required action",
        "substitute dialogbox by required action ");
   }
   catch (Exception e) {}
   return Status.OK;
 }
```

4. Create a file called plugin.xml in the root of your project, and copy the following definition inside the file. You will have to substitute “MyHandler” by the name of the class created in the previous step, and “dsl” by the extension of the files to which you want to associate the popup menu.

```
<plugin>
   <extension point="org.eclipse.ui.handlers">
     <handler class="MyHandler" commandId="organizacion-docente.command">
     </handler>
   </extension>
   <extension point="org.eclipse.ui.menus">
     <menuContribution allPopups="true" locationURI="popup:org.eclipse.ui.popup.any?after=additions">
       <command commandId="organizacion-docente.command" label="Manipular organizacion docente">
         <visibleWhen checkEnabled="false">
           <iterate>
             <test property="org.eclipse.core.resources.extension" value="ext" />
           </iterate>
         </visibleWhen>
       </command>
     </menuContribution>
   </extension>
 </plugin>
```

5. To test your popup menu, throw a new instance of Eclipse (Run / Run As / Eclipse Application). If you right-click on files with the configured file extension, the popup menu will appear, and when selected, it will execute the previous method “execute”.