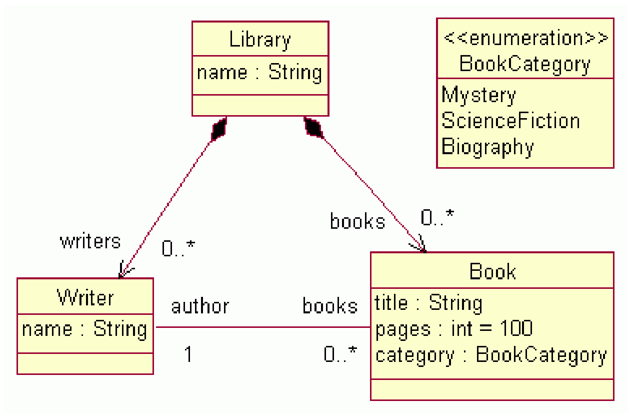
Generating an EMF Model

Last updated: May 31, 2006

This tutorial is a step-by-step description of the process of creating an EMF model and generating a simple model editor for it. Following this tutorial will show how easy EMF makes it to go from a simple model definition to a fully functioning editor for that model.

The model we will generate looks like this in UML (see the Eclipse Modeling Framework Overview for a description of this model):



We will show how an EMF model can be generated from either of two different sources: a Rational Rose model or a set of annotated Java interfaces and classes.

The screenshots are based on version 3.2.0 RC6 of the Eclipse SDK and version 2.2.0 RC6a of EMF.

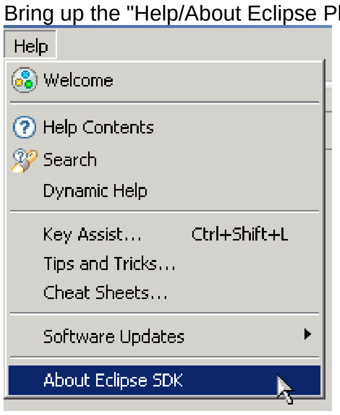
Contents

contents

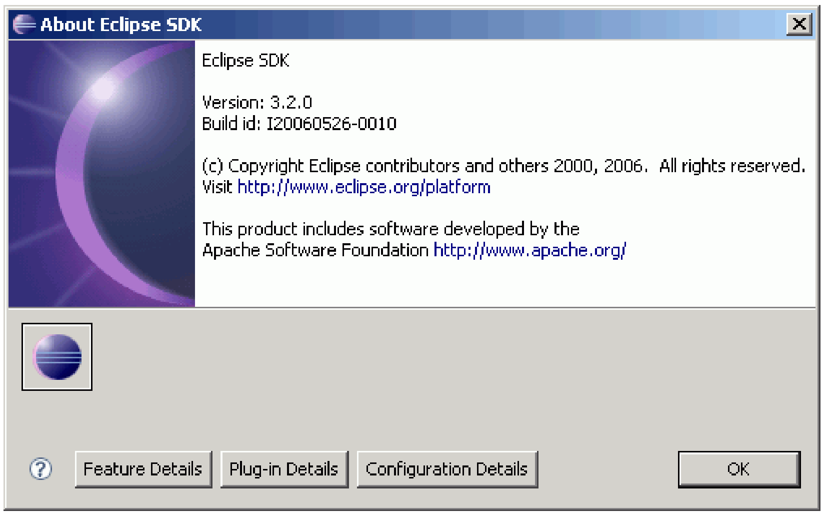
Step 0: Prerequisites

The EMF Runtime package includes the EMF generator and a number of related plug-ins. After installing the package, verify that they are available in your Eclipse environment:

- Bring up the "Help/About Eclipse Platform" dialog.



- Click on "Plug-in Details".


About Eclipse SDK

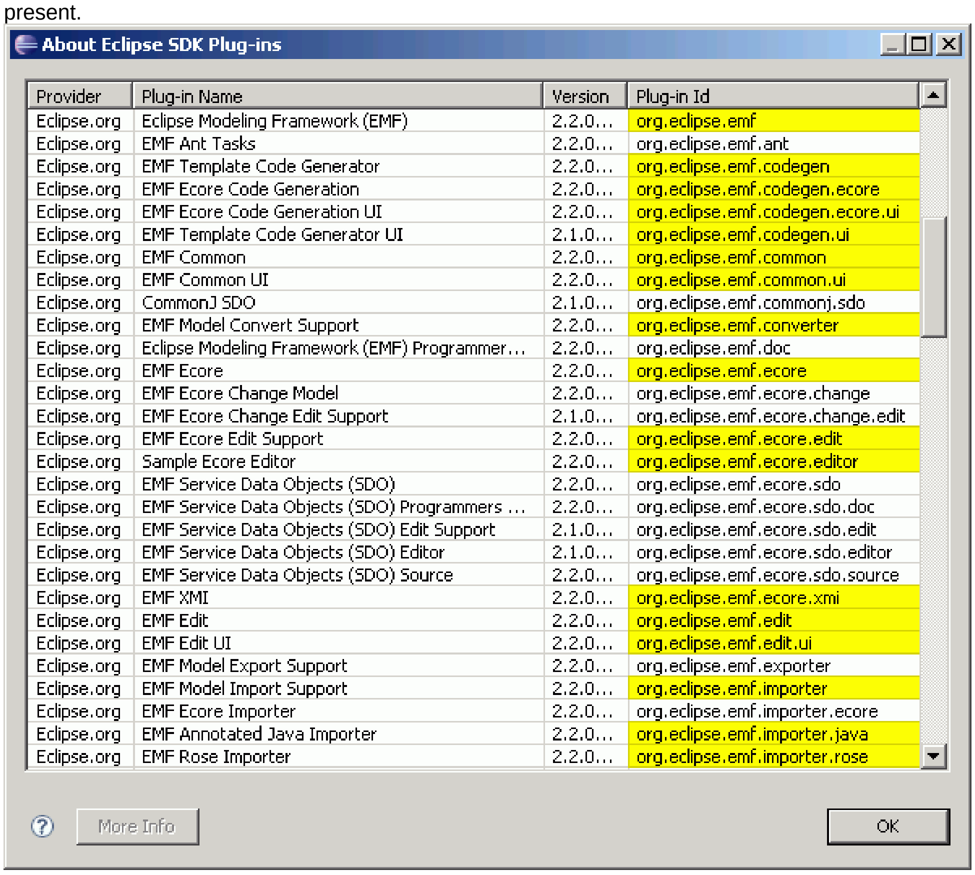
Eclipse SDK

Version: 3.2.0
Build id: I20060526-0010

(c) Copyright Eclipse contributors and others 2000, 2006. All rights reserved.
Visit http://www.eclipse.org/platform

This product includes software developed by the
Apache Software Foundation http://www.apache.org/

- Click the "Plug-in Id" column heading to order the plug-ins by that field. Then, check that the highlighted set of plug-ins are present.


About Eclipse SDK Plug-ins

Provider	Plug-in Name	Version	Plug-in Id
Eclipse.org	Eclipse Modeling Framework (EMF)	2.2.0...	org.eclipse.emf
Eclipse.org	EMF Ant Tasks	2.2.0...	org.eclipse.emf.ant
Eclipse.org	EMF Template Code Generator	2.2.0...	org.eclipse.emf.codegen
Eclipse.org	EMF Ecore Code Generation	2.2.0...	org.eclipse.emf.codegen.ecore
Eclipse.org	EMF Ecore Code Generation UI	2.2.0...	org.eclipse.emf.codegen.ecore.ui
Eclipse.org	EMF Template Code Generator UI	2.1.0...	org.eclipse.emf.codegen.ui
Eclipse.org	EMF Common	2.2.0...	org.eclipse.emf.common
Eclipse.org	EMF Common UI	2.2.0...	org.eclipse.emf.common.ui
Eclipse.org	CommonJ SDO	2.1.0...	org.eclipse.emf.commonj.sdo
Eclipse.org	EMF Model Convert Support	2.2.0...	org.eclipse.emf.converter
Eclipse.org	Eclipse Modeling Framework (EMF) Programmer...	2.2.0...	org.eclipse.emf.doc
Eclipse.org	EMF Ecore	2.2.0...	org.eclipse.emf.ecore
Eclipse.org	EMF Ecore Change Model	2.2.0...	org.eclipse.emf.ecore.change
Eclipse.org	EMF Ecore Change Edit Support	2.1.0...	org.eclipse.emf.ecore.change.edit
Eclipse.org	EMF Ecore Edit Support	2.2.0...	org.eclipse.emf.ecore.edit
Eclipse.org	Sample Ecore Editor	2.2.0...	org.eclipse.emf.ecore.editor
Eclipse.org	EMF Service Data Objects (SDO)	2.2.0...	org.eclipse.emf.ecore.sdo
Eclipse.org	EMF Service Data Objects (SDO) Programmers ...	2.2.0...	org.eclipse.emf.ecore.sdo.doc
Eclipse.org	EMF Service Data Objects (SDO) Edit Support	2.1.0...	org.eclipse.emf.ecore.sdo.edit
Eclipse.org	EMF Service Data Objects (SDO) Editor	2.1.0...	org.eclipse.emf.ecore.sdo.editor
Eclipse.org	EMF Service Data Objects (SDO) Source	2.2.0...	org.eclipse.emf.ecore.sdo.source
Eclipse.org	EMF XMI	2.2.0...	org.eclipse.emf.ecore.xmi
Eclipse.org	EMF Edit	2.2.0...	org.eclipse.emf.edit
Eclipse.org	EMF Edit UI	2.2.0...	org.eclipse.emf.edit.ui
Eclipse.org	EMF Model Export Support	2.2.0...	org.eclipse.emf.exporter
Eclipse.org	EMF Model Import Support	2.2.0...	org.eclipse.emf.importer
Eclipse.org	EMF Ecore Importer	2.2.0...	org.eclipse.emf.importer.ecore
Eclipse.org	EMF Annotated Java Importer	2.2.0...	org.eclipse.emf.importer.java
Eclipse.org	EMF Rose Importer	2.2.0...	org.eclipse.emf.importer.rose

Additional EMF plug-ins, which are not highlighted above, are not required for this tutorial. They may or may not appear, depending on which EMF packages you installed.

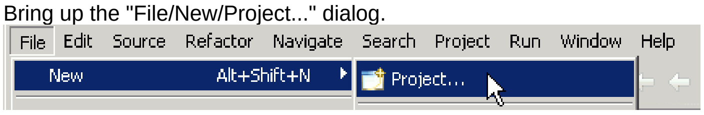
contents

Step 1a: Import the Model from Rose

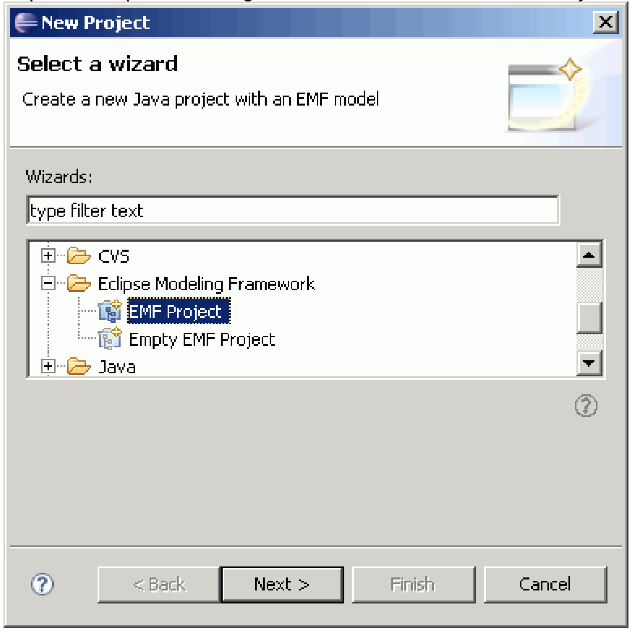
The Rose file for the library model can be found here: library.mdl. Save it somewhere on your workstation.

Create a new EMF project in the workspace:

- Bring up the "File/New/Project..." dialog.



- Expand "Eclipse Modeling Framework" and select "EMF Project". Click the "Next" button.



New Project

Select a wizard

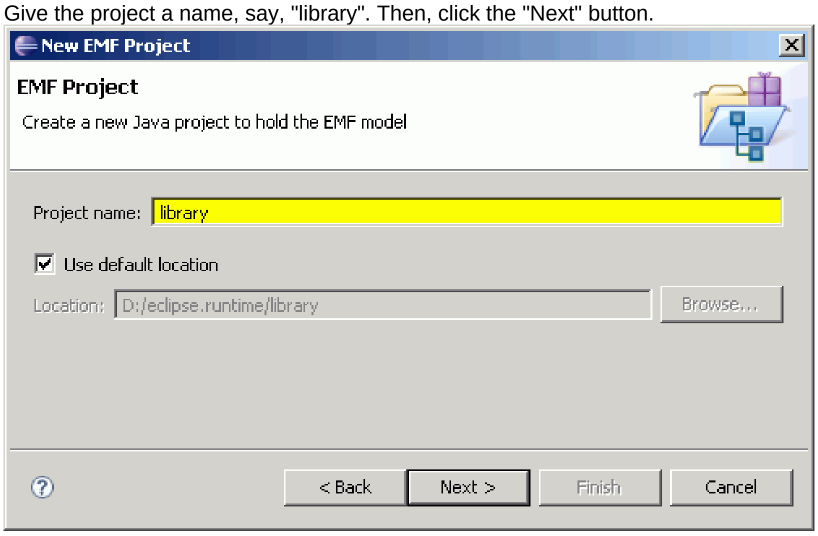
Create a new Java project with an EMF model

Wizards:

type filter text

- CVS
- Eclipse Modeling Framework
  - EMF Project
  - Empty EMF Project
- Java

- Give the project a name, say, "library". Then, click the "Next" button.



New EMF Project

EMF Project

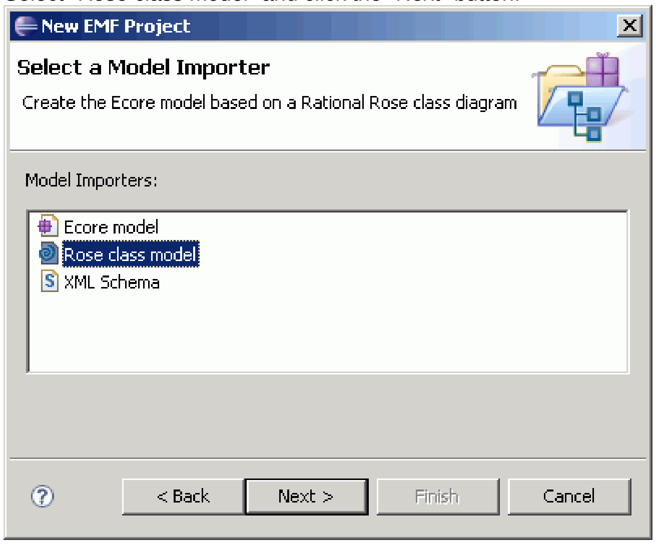
Create a new Java project to hold the EMF model

Project name: library

☑ Use default location

Location: D:/eclipse.runtime/library

- Select "Rose class model" and click the "Next" button.



New EMF Project

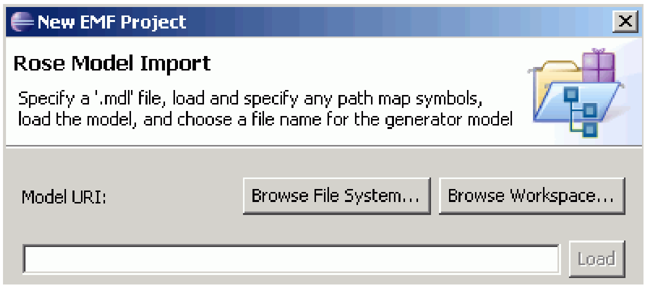
Select a Model Importer

Create the Ecore model based on a Rational Rose class diagram

Model Importers:

- Ecore model
- Rose class model
- XML Schema

- Click on the "Browse File System..." or "Browse Workspace..." button and locate the Rose model file.

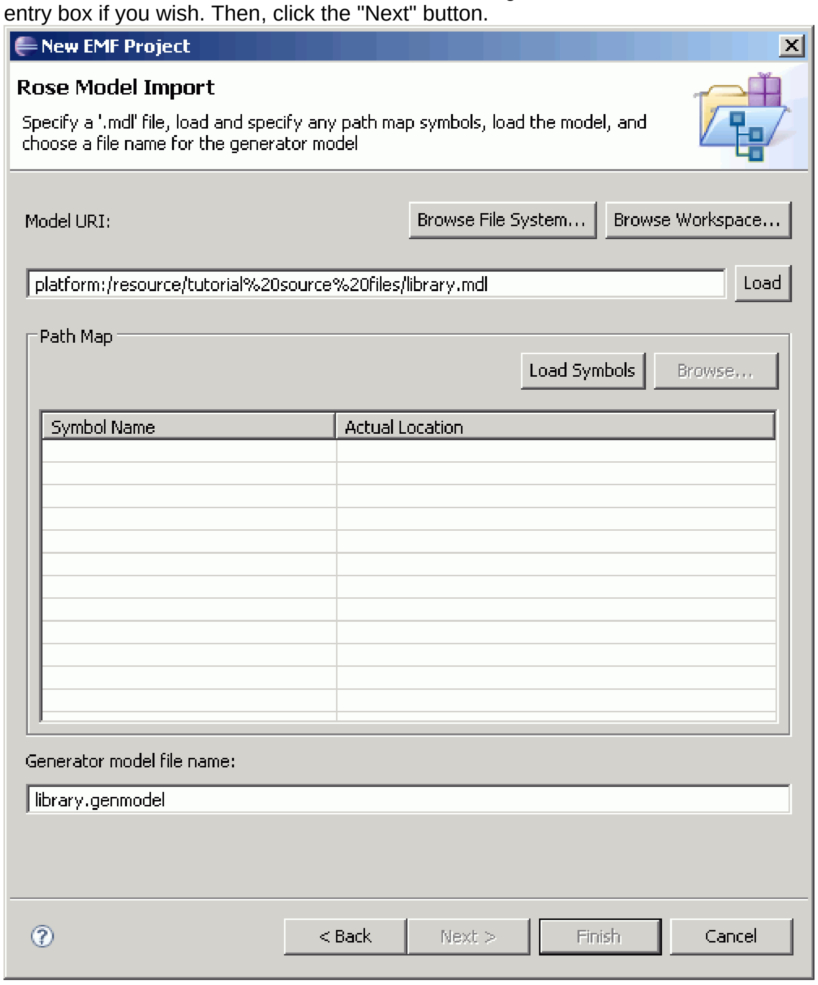

New EMF Project

Rose Model Import

Specify a '.mdl' file, load and specify any path map symbols, load the model, and choose a file name for the generator model

Model URI:

- The Rose model file will be examined, and a default generator model name will be suggested. You can change the name in the entry box if you wish. Then, click the "Next" button.


New EMF Project

Rose Model Import

Specify a '.mdl' file, load and specify any path map symbols, load the model, and choose a file name for the generator model

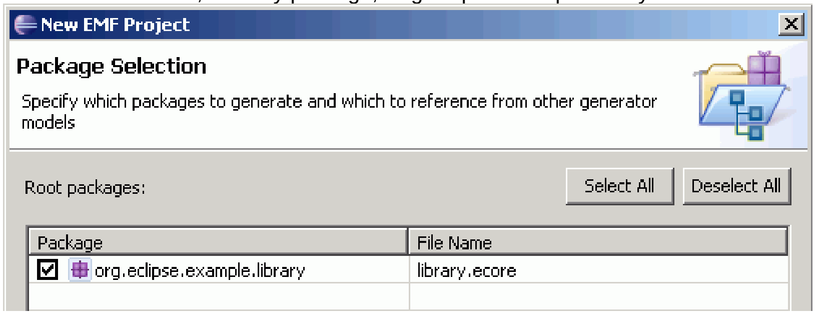
Model URI: platform:/resource/tutorial%20source%20files/library.mdl

Path Map

Symbol Name	Actual Location

Generator model file name: library.genmodel

- In general, a Rose model may contain more than one package. Select the package for which you want to generate an EMF model -- in this case, the only package, "org.eclipse.example.library". Click the "Finish" button.


New EMF Project

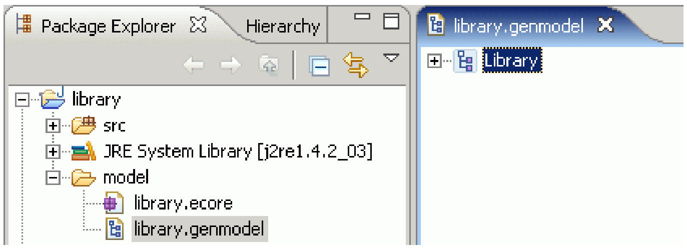
Package Selection

Specify which packages to generate and which to reference from other generator models

Root packages:

Package	File Name
✓ org.eclipse.example.library	library.ecore

- An Ecore model (library.ecore) and a generator model (library.genmodel) will be created. The latter, which controls code generation for the model, is opened in the main view.



---

contents

Step 1b: Define the Model Using Annotated Java

Instead of importing the model from a Rose diagram, we can start with a set of Java interfaces and classes that correspond to the classes and enumerated types, respectively, in the library model. This code is the bare minimum required to illustrate the desired features. Based on it, an Ecore model and a generator model will be constructed, which will then drive generation of the remaining code. The code is annotated with "@model" tags in Javadoc comments, in order to specify any non-default values for the attributes and references of the Ecore objects.

Library.java

```
package org.eclipse.example.library;
import java.util.List;

/**
 * @model
 */
public interface Library
{
  /**
   * @model
   */
  String getName();

  /**
   * @model type="Writer" containment="true"
   */
  List getWriters();

  /**
   * @model type="Book" containment="true"
   */
  List getBooks();
}
```

Book.java

```
package org.eclipse.example.library;

/**
 * @model
 */
public interface Book
{
  /**
   * @model
   */
  String getTitle();

  /**
   * @model default="100"
   */
  int getPages();

  /**
   * @model
   */
  BookCategory getCategory();

  /**
   * @model opposite="books"
   */
  Writer getAuthor();
}
```

Writer.java

```
package org.eclipse.example.library;
```

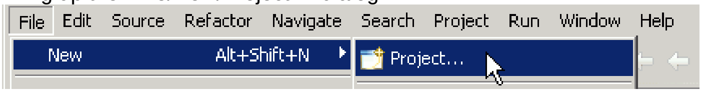
```
/**
 * @model
 */
public interface Writer
{
  /**
   * @model
   */
  String getName();

  /**
   * @model type="Book" opposite="author"
   */
  java.util.List getBooks();
}
```

BookCategory.java

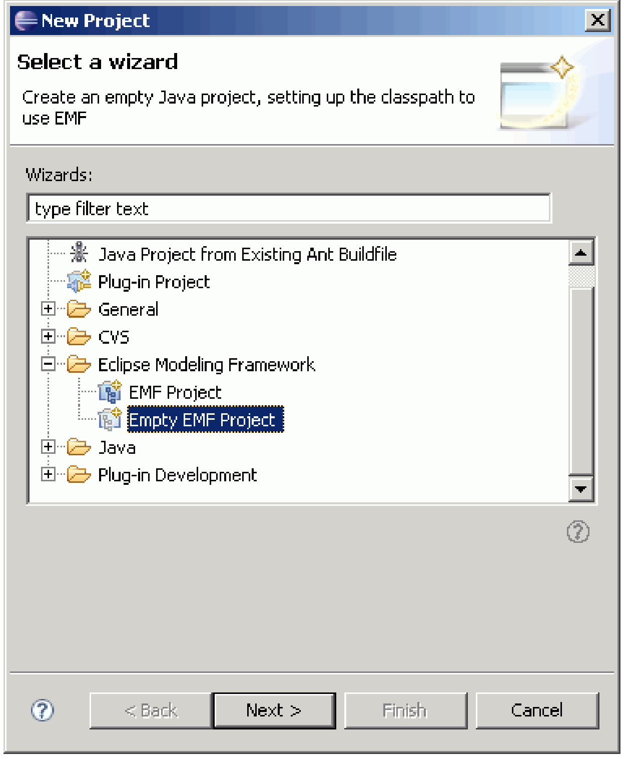
```
package org.eclipse.example.library;

/**
 * @model
 */
public class BookCategory
{
  /**
   * @model name="Mystery"
   */
  public static final int MYSTERY = 0;

  /**
   * @model name="ScienceFiction"
   */
  public static final int SCIENCE_FICTION = 1;

  /**
   * @model name="Biography"
   */
  public static final int BIOGRAPHY = 2;
}
```

Create a new empty EMF project in the workspace:

- Bring up the "File/New/Project..." dialog.



- Expand "Eclipse Modeling Framework" and select "Empty EMF Project". Click the "Next" button.


New Project

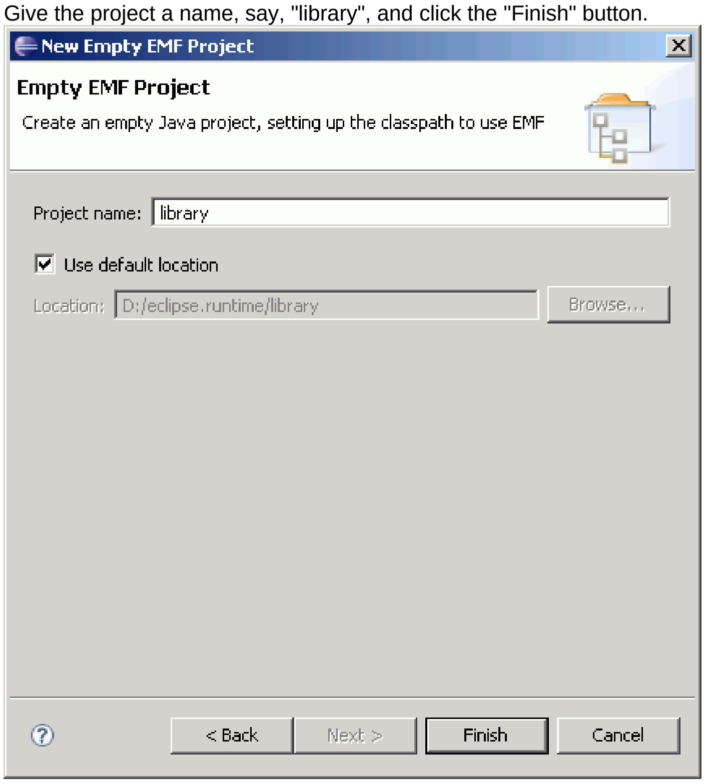
Select a wizard

Create an empty Java project, setting up the classpath to use EMF

Wizards:

type filter text

- Give the project a name, say, "library", and click the "Finish" button.

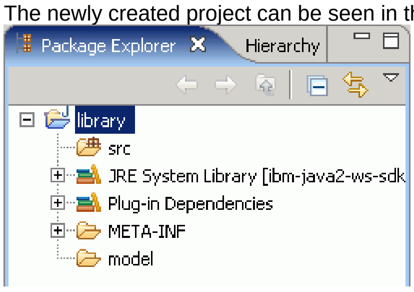

New Empty EMF Project
Empty EMF Project
Create an empty Java project, setting up the classpath to use EMF
< Back
Next >
Finish
Cancel

Project name: library

[✓] Use default location

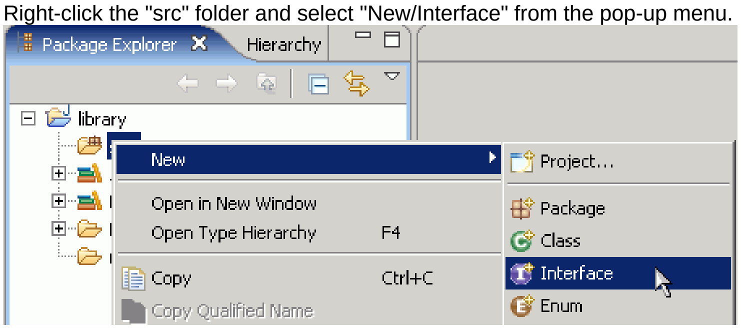
Location: D:/eclipse.runtime/library Browse...

- The newly created project can be seen in the Package Explorer.

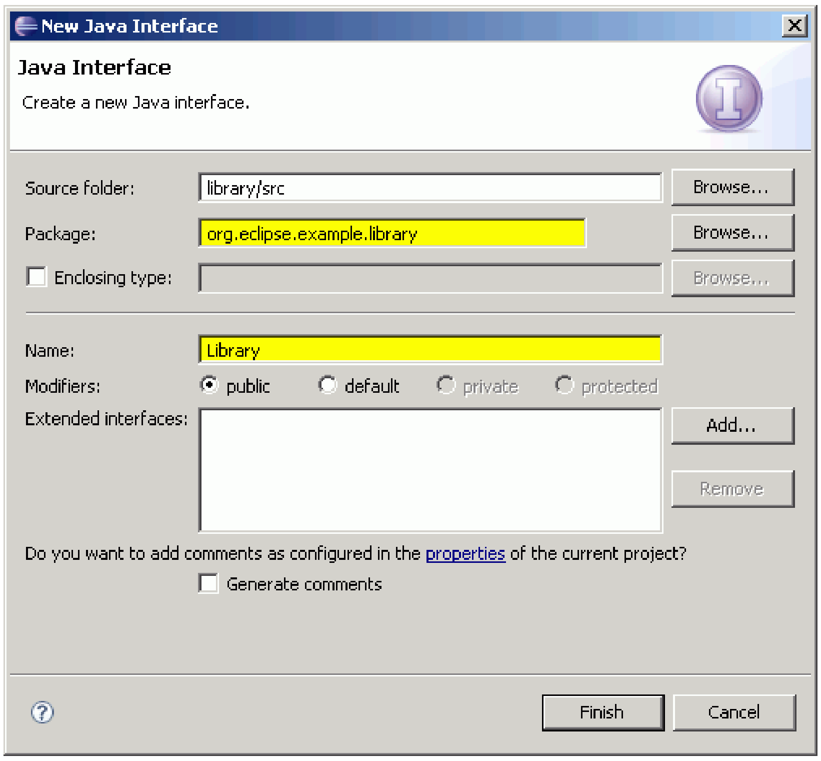


Create the first Java interface:

- Right-click the "src" folder and select "New/Interface" from the pop-up menu.



- Fill in the package and interface name. Click the "Finish" button.



Source folder: library/src

Package: org.eclipse.example.library

[ ] Enclosing type:

Name: Library

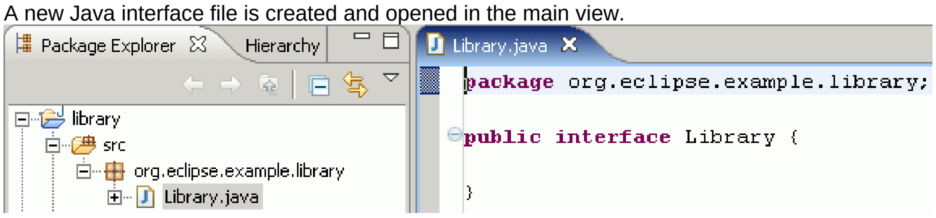
Modifiers: [✓ public] default private protected

Extended interfaces:

Do you want to add comments as configured in the properties of the current project?

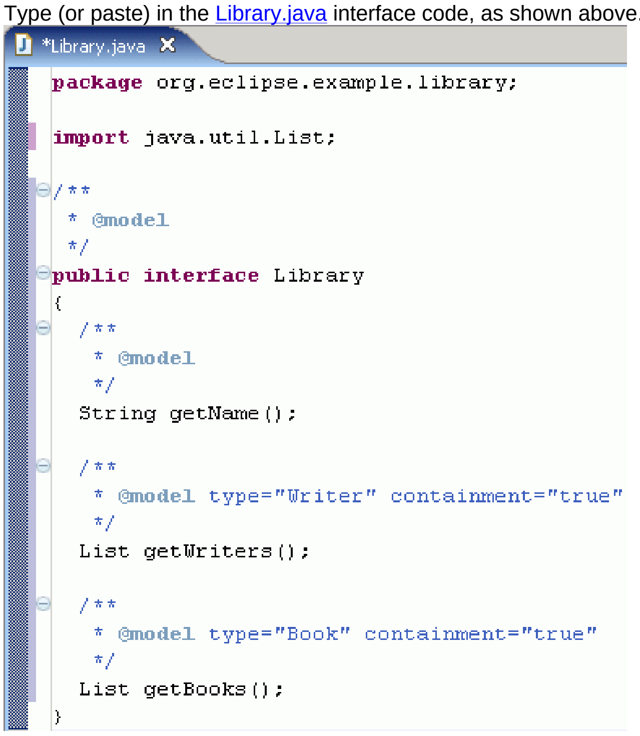
[ ] Generate comments

- A new Java interface file is created and opened in the main view.



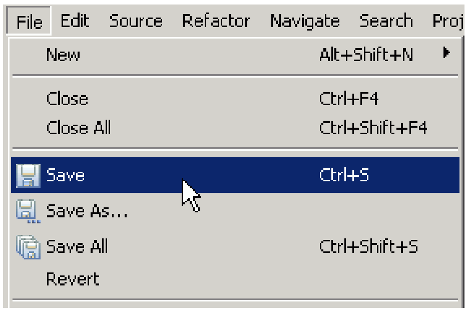
```
package org.eclipse.example.library;

public interface Library {

}
```

- Type (or paste) in the Library.java interface code, as shown above. Pay special attention to the @model tags.



```
package org.eclipse.example.library;

import java.util.List;

/**
 * @model
 */
public interface Library
{
  /**
   * @model
   */
  String getName();

  /**
   * @model type="Writer" containment="true"
   */
  List getWriters();

  /**
   * @model type="Book" containment="true"
   */
  List getBooks();
}
```

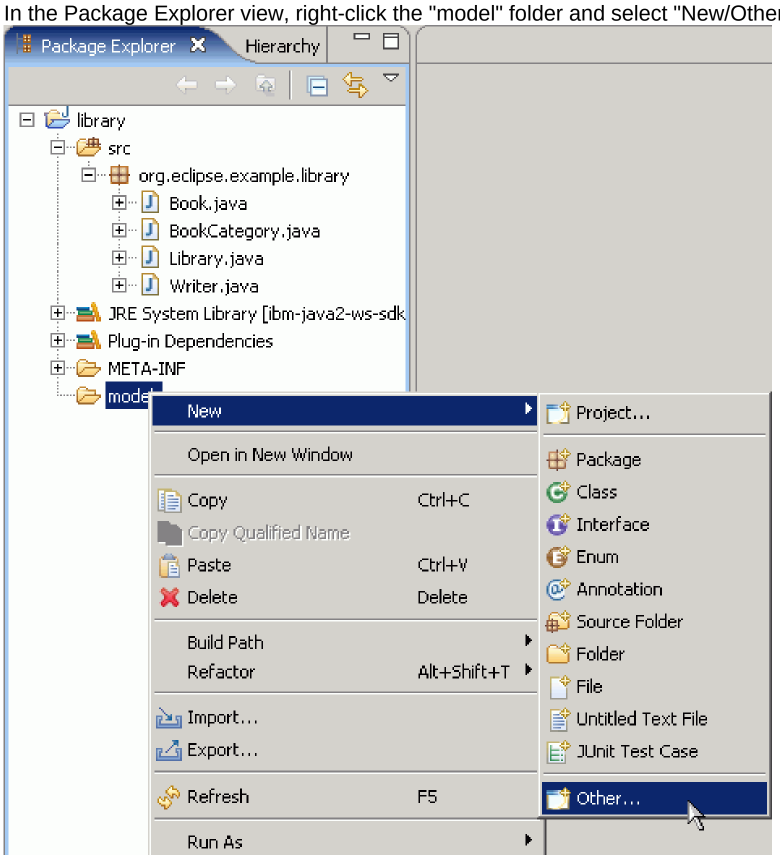
- Select "Save" from the "File" menu to save the file.



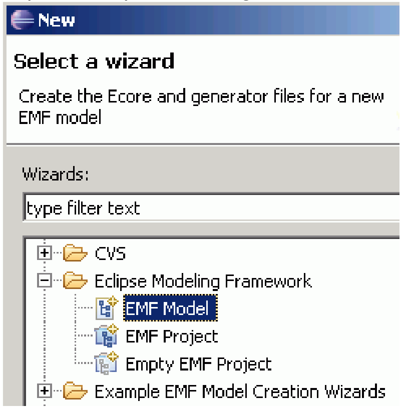
Create the other two interfaces (Book.java and Writer.java) and the class (BookCategory.java) in the same way. Of course, to create the class, select "New/Class" from the pop-up menu, instead of "New/Interface".

Create the EMF model:

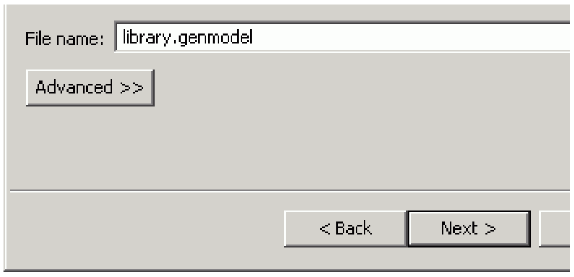
- In the Package Explorer view, right-click the "model" folder and select "New/Other..." from the pop-up menu.



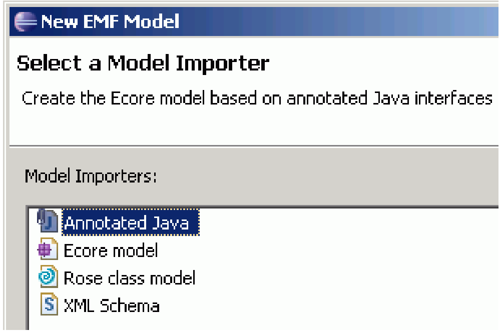
- Expand "Eclipse Modeling Framework" and select "EMF Model". Click the "Next" button.


New
Select a wizard
Create the Ecore and generator files for a new EMF model
Wizards:
type filter text
CVS
Eclipse Modeling Framework
EMF Model
EMF Project
Empty EMF Project
Example EMF Model Creation Wizards

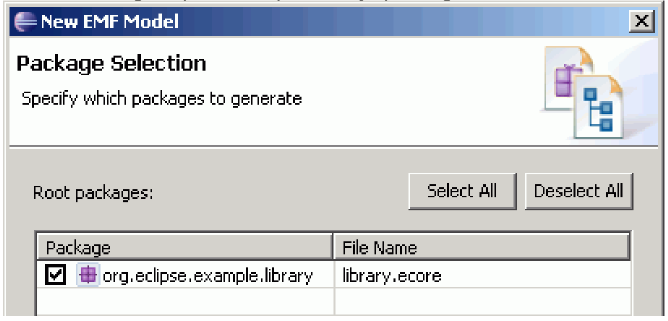
- Change the file name to "library.genmodel" and click the "Next" button.



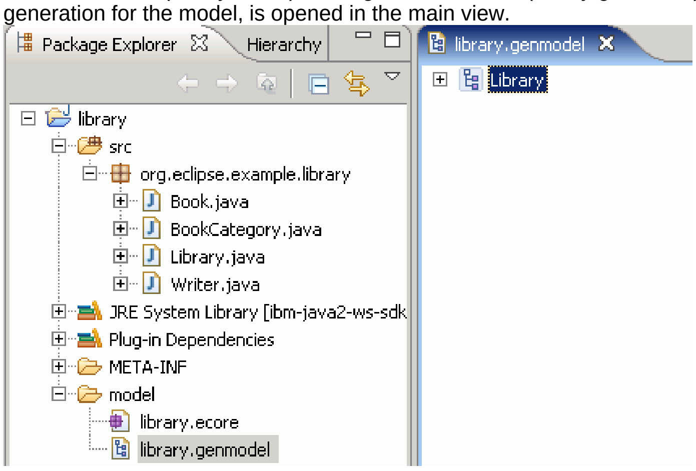
- Select "Annotated Java" and click the "Next" button.



- Select the "org.eclipse.example.library" package and click the "Finish" button.



- An Ecore model (library.ecore) and a generator model (library.genmodel) will be created. The latter, which controls code generation for the model, is opened in the main view.

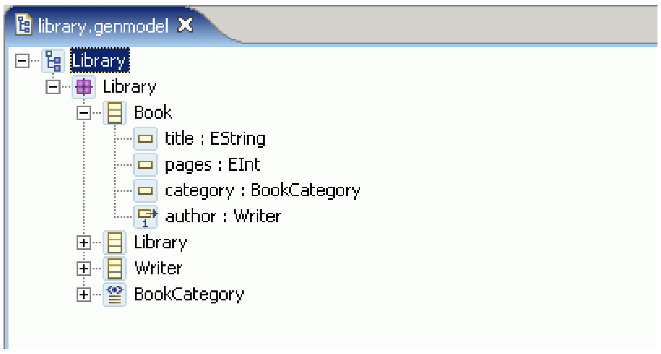


contents

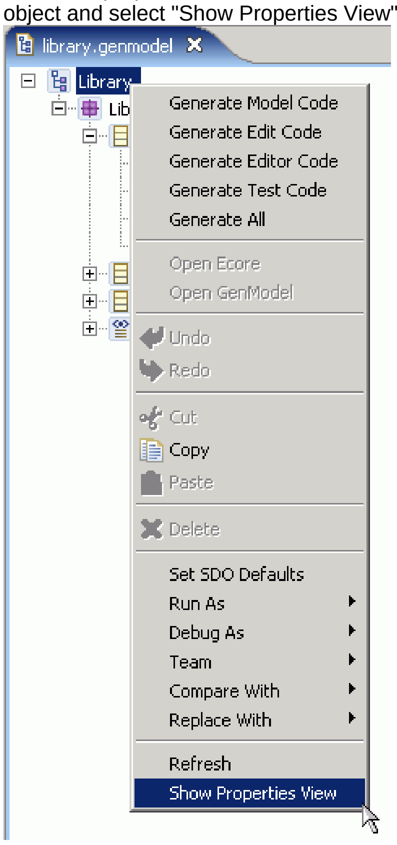
Step 2: Generate the EMF Model Code

The generator model shows a root object, representing the whole model. This model object has children that represent its packages, whose children then represent classifiers (classes and datatypes, including enumerated types). The children of classes are class attributes, references, and operations; the children of enumerated types are enum literals.

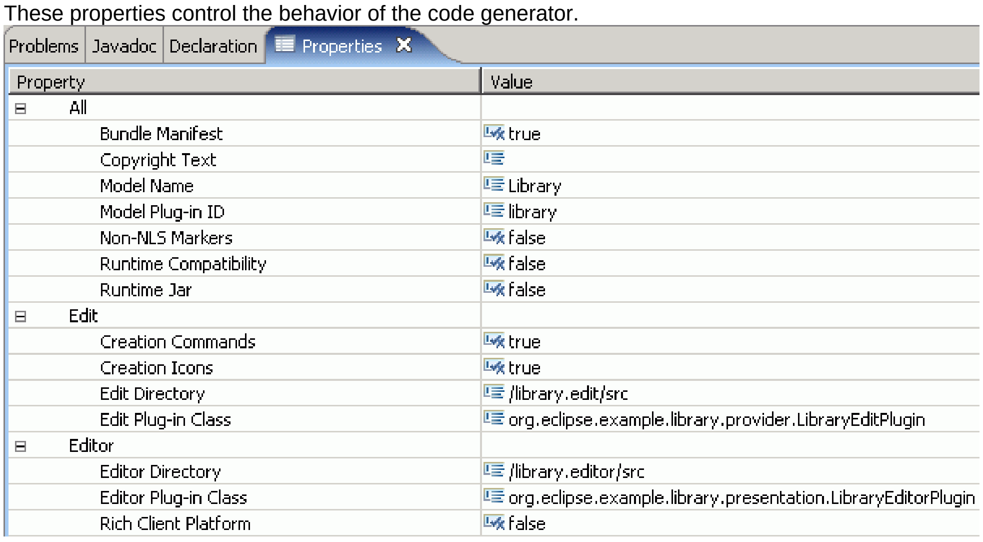
- The model can be expanded to see its various elements.



- There are properties associated with each object. If the Properties view isn't already showing, right-click the "Library" model object and select "Show Properties View" from the pop-up menu, or simply double-click the "Library" object.



- These properties control the behavior of the code generator.

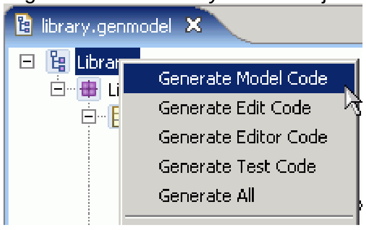


Property	Value
⊟ All	
Bundle Manifest	true
Copyright Text	
Model Name	Library
Model Plug-in ID	library
Non-NLS Markers	false
Runtime Compatibility	false
Runtime Jar	false
⊟ Edit	
Creation Commands	true
Creation Icons	true
Edit Directory	/library.edit/src
Edit Plug-in Class	org.eclipse.example.library.provider.LibraryEditPlugin
⊟ Editor	
Editor Directory	/library.editor/src
Editor Plug-in Class	org.eclipse.example.library.presentation.LibraryEditorPlugin
Rich Client Platform	false

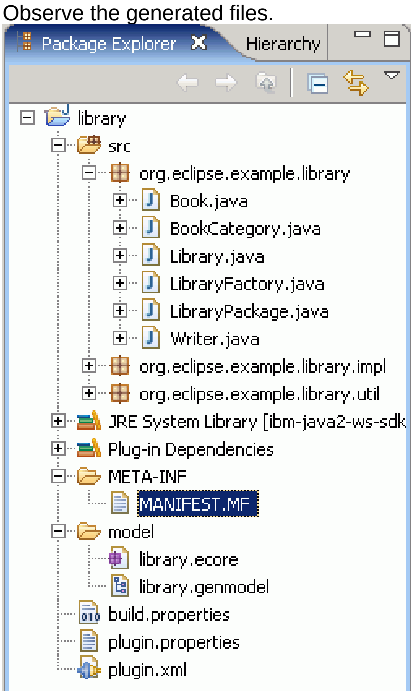
In most cases, the properites need not be changed from their default values, but these options can provide a great deal of control over the code that gets generated. This topic will be explored more fully in future tutorial material; for now, select several different generator model objects, and observe their properties.

The generator model is also the place where you initiate the code generation. By right-clicking on an object in the model, you can generate code for it.

- Right-click the "Library" model object and select "Generate Model Code" from the pop-up menu.



- Observe the generated files.



After generation, the class interfaces and enum class will have been created (if the model was imported from Rose) or completed (if the model was defined using annotated Java), and a new pair of interfaces will have been created for the package itself and for the factory. There will also be two new packages, with "impl" and "util" suffixes, which contain implementations of the interfaces and additional utility classes, and both types of manifest files for the model plug-in: "plugin.xml" and "MANIFEST.MF".

If you defined the model by using annotated Java, you may see a warning in the Problems view: "The import java.util.List is never used". This warning is expected, and will not stop you from continuing on to the next step.

If you change the model, you can regenerate it, and changes will be merged with any hand modifications that may have been made to the code. You can also selectively generate a subset of the model code by right-clicking on a package, class, or enum object and selecting "Generate Model Code" from the pop-up menu.

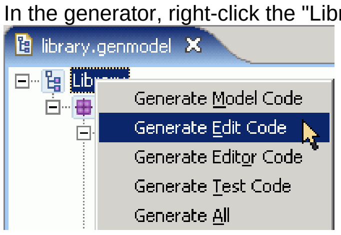
---

contents

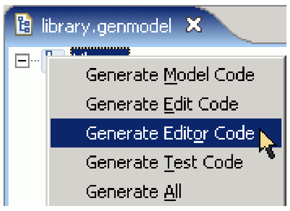
Step 3: Generate an Editor for the Model

A fully functional Eclipse editor can also be generated for any model. By default, it is split between two plug-ins: an "edit" plug-in includes adapters that provide a structured view and perform command-based editing of the model objects; an "editor" plug-in provides the UI for the editor and wizard.

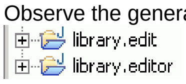
- In the generator, right-click the "Library" model object and select "Generate Edit Code" from the pop-up menu.



- Right-click the model object again and select "Generate Editor Code" from the pop-up menu.



- Observe the generated projects in the Package Explorer view, with "edit" and "editor" suffixes.

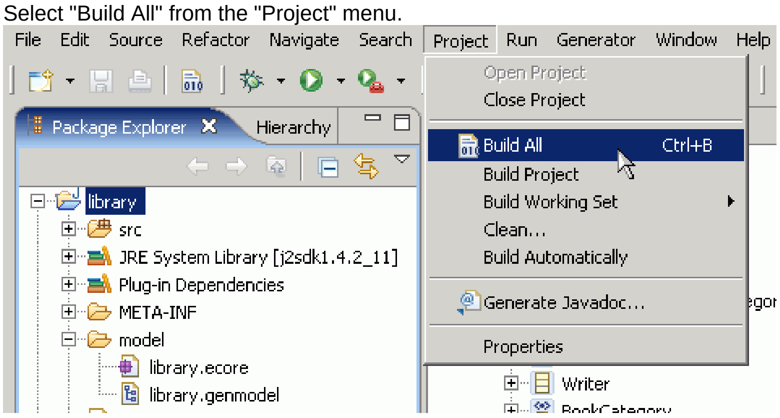


In general, if you wish to generate the model, edit, and editor plug-ins in a single step, you can do so by selecting "Generate All" from the pop-up menu.

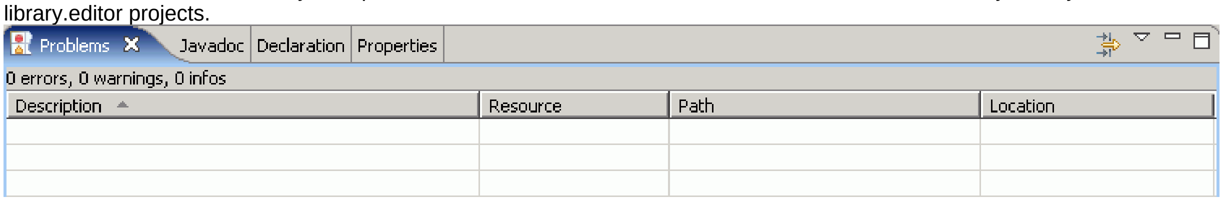
This will also create a tests plug-in, containing JUnit test skeletons for exercising any volatile features and operations defined in the model. This model doesn't include any, but In general, you'll need to fill in the bodies of these tests, yourself. The tests plug-in also includes a simple example class, which shows how to load and validate a model in a stand-alone application.

The code should be compiled automatically as it is generated, and should recompile whenever it is changed. If you have disabled automatic building in the workbench preferences, you can initiate compilation manually:

- Select "Build All" from the "Project" menu.



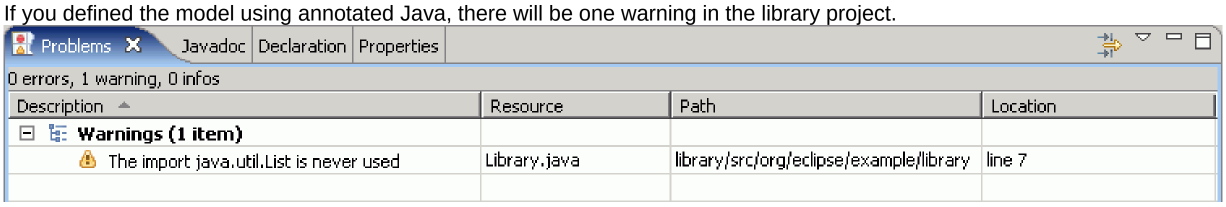
- Observe the Problems view. If you imported the model from Rose, there should be no errors in the library, library.edit, and library.editor projects.



0 errors, 0 warnings, 0 infos

Description	Resource	Path	Location

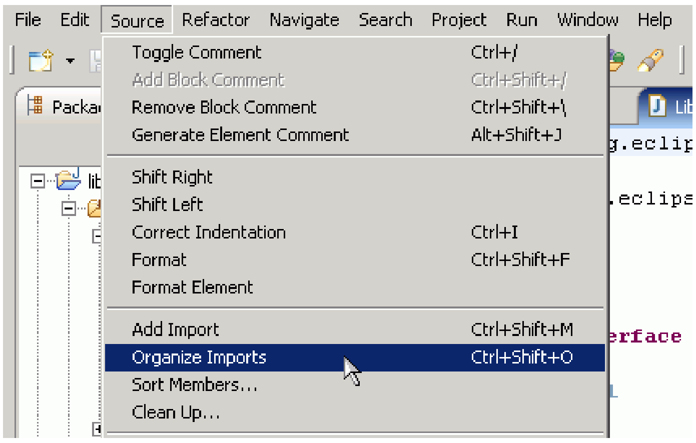
- If you defined the model using annotated Java, there will be one warning in the library project.



0 errors, 1 warning, 0 infos

Description	Resource	Path	Location
Warnings (1 item)			
The import java.util.List is never used	Library.java	library/src/org/eclipse/example/library	line 7

- Double click the warning to see the line of code that is causing it. It occurs because the code generator changed the return type of Library's two multiplicity-many references from "List" to "EList", but did not remove the unused import. You can simply delete it, or have Eclipse do it for you, by selecting "Organize Imports" from the "Source" menu.

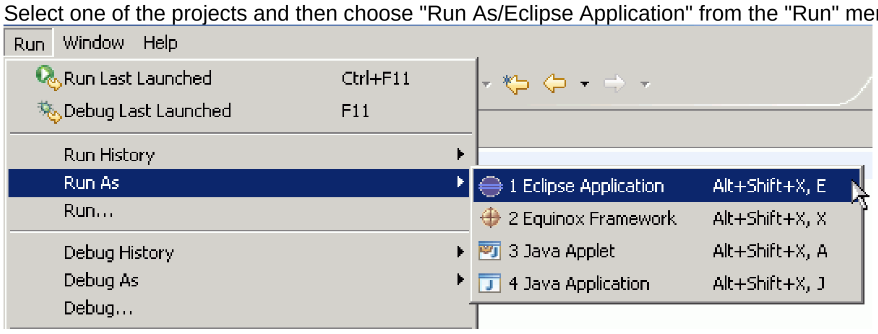


contents

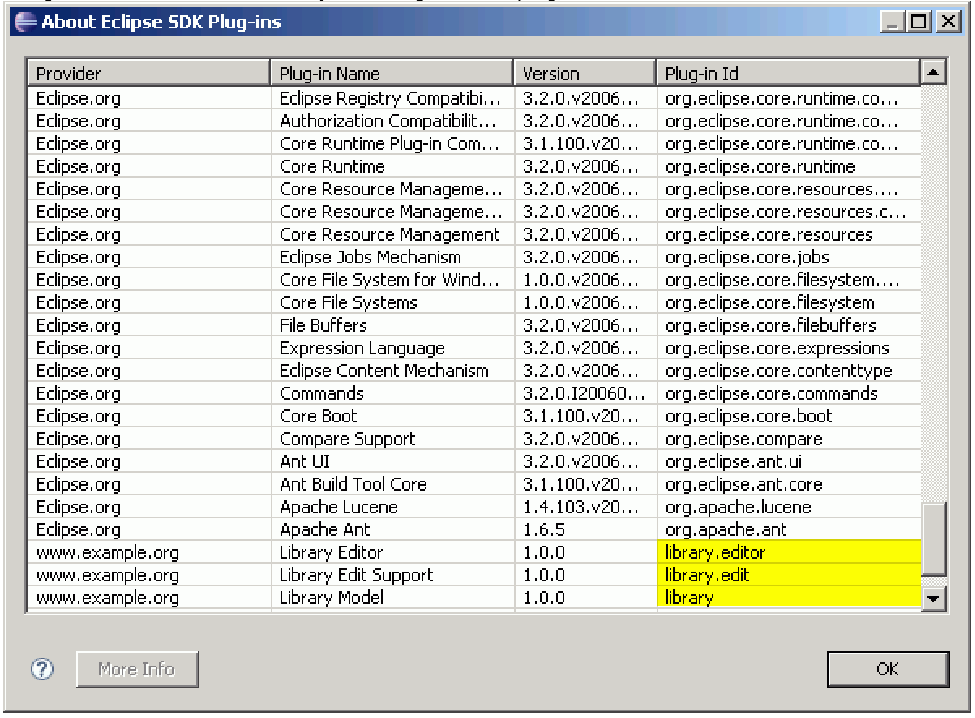
Step 4: Run the Generated Editor

In order to test the new plug-ins, a second instance of Eclipse must be launched. The plug-ins will run in this workbench.

- Select one of the projects and then choose "Run As/Eclipse Application" from the "Run" menu or toolbar drop-down.


Run
Window
Help
Run Last Launched Ctrl+F11
Debug Last Launched F11
Run History
Run As
Run...
Debug History
Debug As
Debug...
1 Eclipse Application Alt+Shift+X, E
2 Equinox Framework Alt+Shift+X, X
3 Java Applet Alt+Shift+X, A
4 Java Application Alt+Shift+X, J

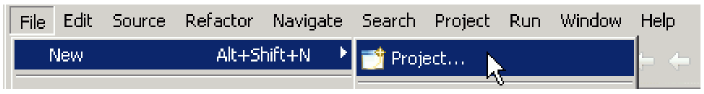
- Wait for a second instance of the Eclipse platform to come up. Bring up the "Help/About Eclipse Platform" dialog, click on the "Plug-in Details" button, and verify that the generated plug-ins are there.



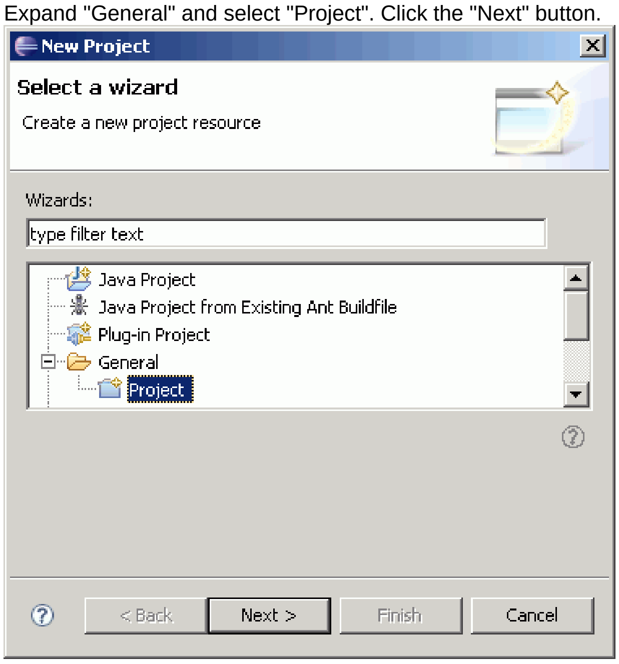
Provider	Plug-in Name	Version	Plug-in Id
Eclipse.org	Eclipse Registry Compatibi...	3.2.0.v2006...	org.eclipse.core.runtime.co...
Eclipse.org	Authorization Compatibilit...	3.2.0.v2006...	org.eclipse.core.runtime.co...
Eclipse.org	Core Runtime Plug-in Com...	3.1.100.v20...	org.eclipse.core.runtime.co...
Eclipse.org	Core Runtime	3.2.0.v2006...	org.eclipse.core.runtime
Eclipse.org	Core Resource Manageme...	3.2.0.v2006...	org.eclipse.core.resources....
Eclipse.org	Core Resource Manageme...	3.2.0.v2006...	org.eclipse.core.resources.c...
Eclipse.org	Core Resource Management	3.2.0.v2006...	org.eclipse.core.resources
Eclipse.org	Eclipse Jobs Mechanism	3.2.0.v2006...	org.eclipse.core.jobs
Eclipse.org	Core File System for Wind...	1.0.0.v2006...	org.eclipse.core.filesystem....
Eclipse.org	Core File Systems	1.0.0.v2006...	org.eclipse.core.filesystem
Eclipse.org	File Buffers	3.2.0.v2006...	org.eclipse.core.filebuffers
Eclipse.org	Expression Language	3.2.0.v2006...	org.eclipse.core.expressions
Eclipse.org	Eclipse Content Mechanism	3.2.0.v2006...	org.eclipse.core.contenttype
Eclipse.org	Commands	3.2.0.I20060...	org.eclipse.core.commands
Eclipse.org	Core Boot	3.1.100.v20...	org.eclipse.core.boot
Eclipse.org	Compare Support	3.2.0.v2006...	org.eclipse.compare
Eclipse.org	Ant UI	3.2.0.v2006...	org.eclipse.ant.ui
Eclipse.org	Ant Build Tool Core	3.1.100.v20...	org.eclipse.ant.core
Eclipse.org	Apache Lucene	1.4.103.v20...	org.apache.lucene
Eclipse.org	Apache Ant	1.6.5	org.apache.ant
www.example.org	Library Editor	1.0.0	library.editor
www.example.org	Library Edit Support	1.0.0	library.edit
www.example.org	Library Model	1.0.0	library

The Library Model wizard can now be used to create a new instance of the model.

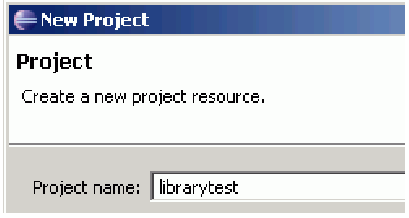
- Bring up the "File/New/Project..." dialog.



- Expand "General" and select "Project". Click the "Next" button.



- Give the project a name and click the "Finish" button.

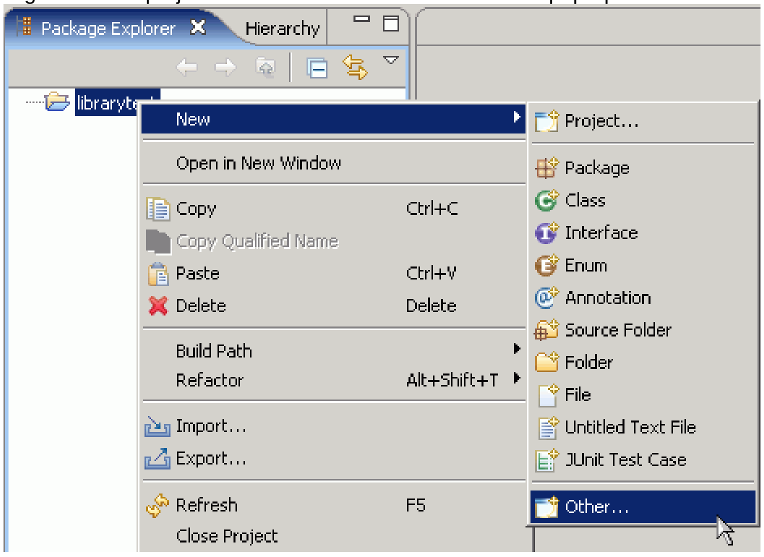

New Project

Project

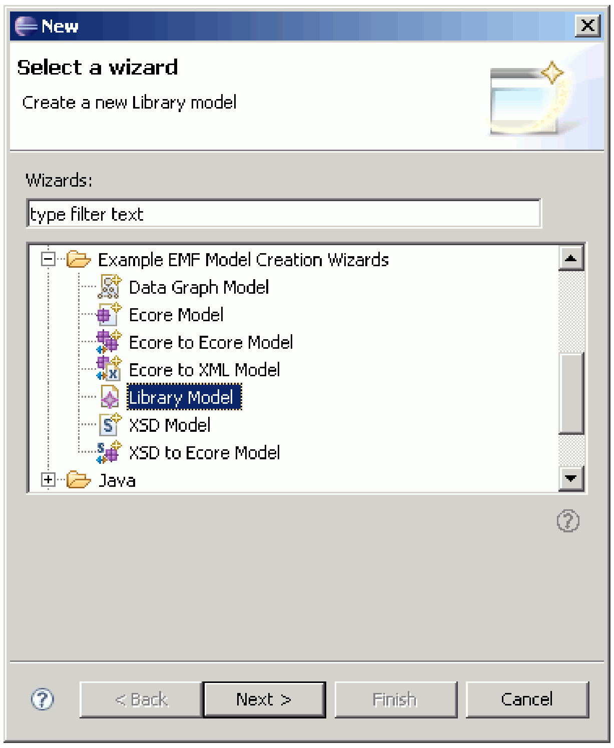
Create a new project resource.

Project name: librarytest

- Right-click the project and select "New/Other..." from the pop-up menu.



- Expand "Example EMF Model Creation Wizards" and select "Library Model". Click the "Next" button.


New

Select a wizard

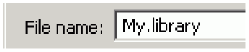
Create a new Library model

Wizards:

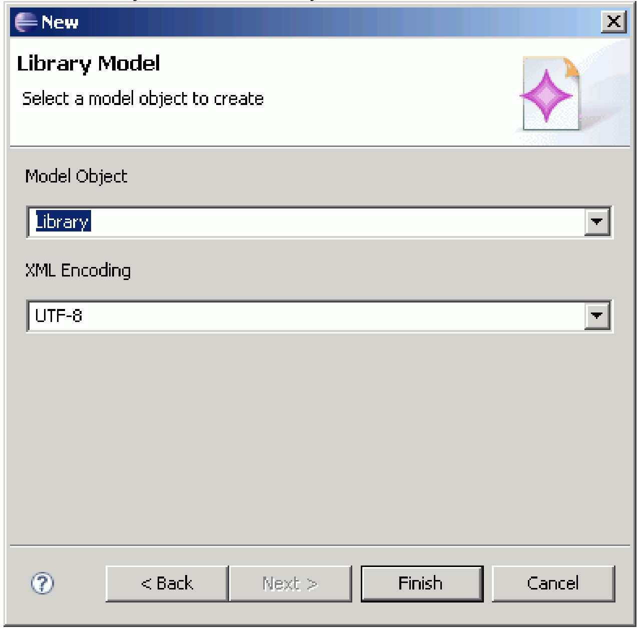
type filter text

- Example EMF Model Creation Wizards
  - Data Graph Model
  - Ecore Model
  - Ecore to Ecore Model
  - Ecore to XML Model
  - Library Model
  - XSD Model
  - XSD to Ecore Model
- Java

- Enter a file name for the library model. Make sure it ends with a ".library" extension. Then, click the "Next" button.


File name: My.library

- Select "Library" as the model object and click the "Finish" button.


New

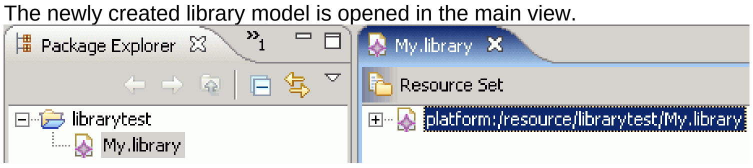
Library Model

Select a model object to create

Model Object: Library

XML Encoding: UTF-8

- The newly created library model is opened in the main view.

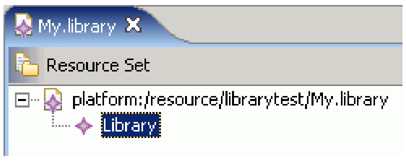


- librarytest
  - My.library

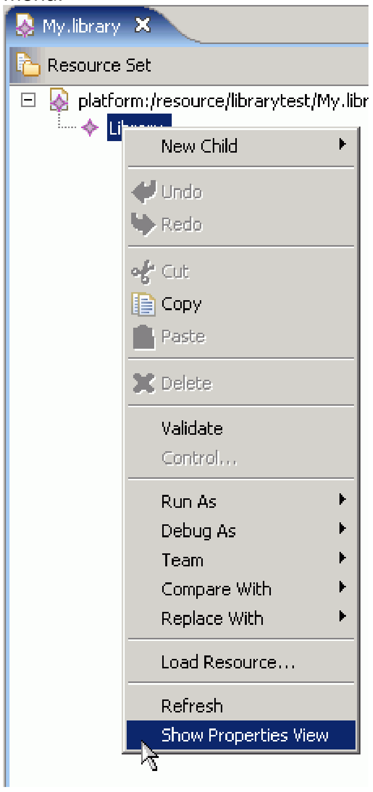
- Resource Set
  - platform:/resource/librarytest/My.library

The root object in this editor corresponds to the My.library resource. Under it lies a single library, the object which was selected as the model object in the wizard.

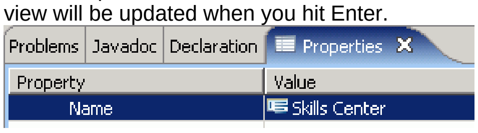
- Expand the "platform:/resource/librarytest/My.library" resource to see the "Library" object. Select it.



- If the Properties view isn't already showing, right-click the "Library" object and select "Show Properties View" from the pop-up menu.

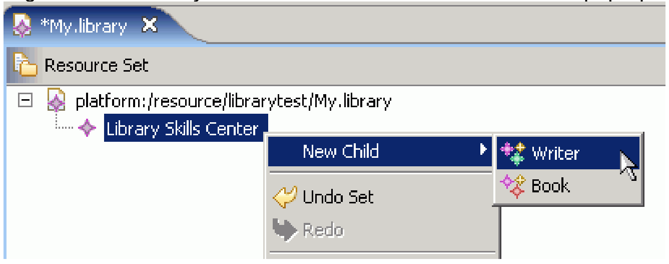


- In the Properties view, click on the "Value" column of the "Name" property, and give a name to the library. The label in the main view will be updated when you hit Enter.

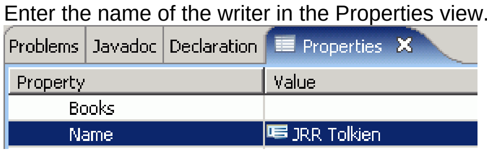


Property	Value
Name	Skills Center

- Right-click the library and select "New Child/Writer" from the pop-up menu. A new writer is added to the library.

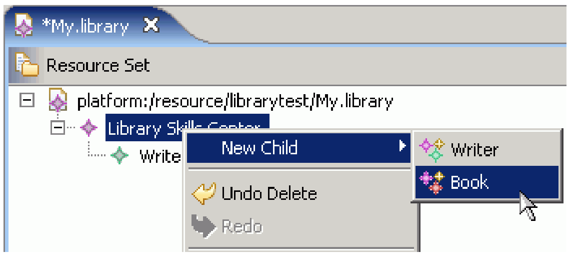


- Enter the name of the writer in the Properties view.

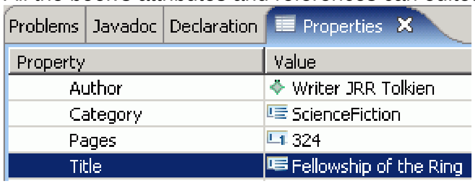


Property	Value
Books	
Name	JRR Tolkien

- Similarly, a book can be added to the library.

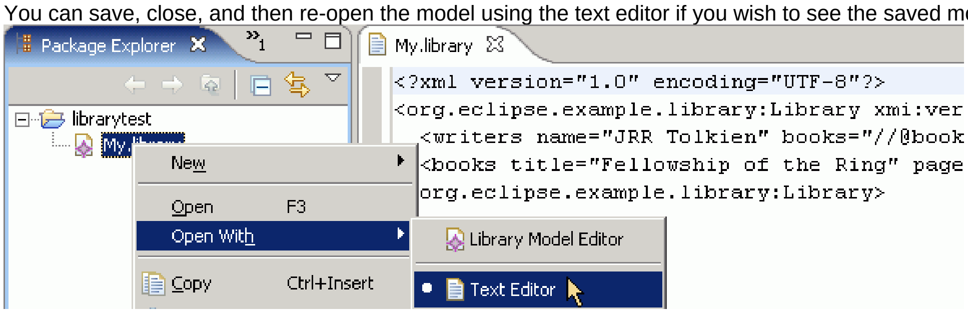


- All the book's attributes and references can edited in the Properties view.



Property	Value
Author	Writer JRR Tolkien
Category	ScienceFiction
Pages	324
Title	Fellowship of the Ring

- You can save, close, and then re-open the model using the text editor if you wish to see the saved model in XMI format.



```
<?xml version="1.0" encoding="UTF-8"?>
<org.eclipse.example.library:Library xmi:ver
  <writers name="JRR Tolkien" books="//@book
  <books title="Fellowship of the Ring" page
  org.eclipse.example.library:Library>
```

contents