**Eclipse Modeling Framework (EMF) - Tutorial**

**Lars Vogel**

Version 2.6

Copyright © 2007, 2008, 2009, 2010, 2011, 2012 Lars Vogel

29.12.2012

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision History** | | | |
| Revision 0.1 | 12.08.2007 | Lars Vogel | Created |
| Revision 0.2 - 2.6 | 30.07.2008 - 29.12.2012 | Lars Vogel | bug fixes and enhancements |

**Eclipse EMF**

This tutorial describes the usage of the Eclipse EMF framework for modeling your data model and creating Java code from it. This tutorial is based on Eclipse 4.2 (Juno).

**Table of Contents**

[**1. Models and Eclipse EMF**](http://www.vogella.com/articles/EclipseEMF/article.html#intro)

[**1.1. Data model**](http://www.vogella.com/articles/EclipseEMF/article.html#overview)

[**1.2. Eclipse EMF**](http://www.vogella.com/articles/EclipseEMF/article.html#emfoverview)

[**1.3. Meta Models - Ecore and Genmodel**](http://www.vogella.com/articles/EclipseEMF/article.html#intro_emfmodel)

[**1.4. Advantages of using EMF**](http://www.vogella.com/articles/EclipseEMF/article.html#emfadvantages)

[**2. Installation**](http://www.vogella.com/articles/EclipseEMF/article.html#emfinstallation)

[**3. Define EMF model**](http://www.vogella.com/articles/EclipseEMF/article.html#createmodel)

[**3.1. Create project**](http://www.vogella.com/articles/EclipseEMF/article.html#createmodel_project)

[**3.2. Create Ecore diagram**](http://www.vogella.com/articles/EclipseEMF/article.html#createmodel_ecoretools)

[**3.3. View Ecore diagram**](http://www.vogella.com/articles/EclipseEMF/article.html#createmodel_ecore)

[**3.4. Create EMF Generator Model**](http://www.vogella.com/articles/EclipseEMF/article.html#createmodel_genmodel)

[**3.5. Set the package**](http://www.vogella.com/articles/EclipseEMF/article.html#createmodel_package)

[**4. Generating the domain classes**](http://www.vogella.com/articles/EclipseEMF/article.html#emfmodel)

[**4.1. Generating Java code**](http://www.vogella.com/articles/EclipseEMF/article.html#emfmodel_code)

[**4.2. Review the generated code**](http://www.vogella.com/articles/EclipseEMF/article.html#emfmodel_codereview)

[**4.3. Updating the model**](http://www.vogella.com/articles/EclipseEMF/article.html#emfmodel_update)

[**5. Create EMF Editor plug-ins**](http://www.vogella.com/articles/EclipseEMF/article.html#emfeditor)

[**5.1. Generating edit / editor code**](http://www.vogella.com/articles/EclipseEMF/article.html#emfeditor_generate)

[**5.2. Run your plugins**](http://www.vogella.com/articles/EclipseEMF/article.html#emfeditor_run)

[**5.3. Create your model**](http://www.vogella.com/articles/EclipseEMF/article.html#emfeditor_createmodel)

[**5.4. Edit your model**](http://www.vogella.com/articles/EclipseEMF/article.html#emfeditor_editmodel)

[**6. Using the model code**](http://www.vogella.com/articles/EclipseEMF/article.html#javacode)

[**6.1. Overview**](http://www.vogella.com/articles/EclipseEMF/article.html#javacode_overview)

[**6.2. Example**](http://www.vogella.com/articles/EclipseEMF/article.html#javacode_usage)

[**7. Creating JavaDoc**](http://www.vogella.com/articles/EclipseEMF/article.html#javadoc)

[**8. Generating methods**](http://www.vogella.com/articles/EclipseEMF/article.html#methods)

[**9. Extending an EMF Ecore model (inheritance)**](http://www.vogella.com/articles/EclipseEMF/article.html#inheritance)

[**9.1. Overview**](http://www.vogella.com/articles/EclipseEMF/article.html#inheritance_overview)

[**9.2. Example**](http://www.vogella.com/articles/EclipseEMF/article.html#inheritance_example)

[**10. Setting the empty string as default value**](http://www.vogella.com/articles/EclipseEMF/article.html#initialvalue)

[**11. Next steps**](http://www.vogella.com/articles/EclipseEMF/article.html#nextsteps)

[**12. Thank you**](http://www.vogella.com/articles/EclipseEMF/article.html#thankyou)

[**13. Questions and Discussion**](http://www.vogella.com/articles/EclipseEMF/article.html#questions)

[**14. Links and Literature**](http://www.vogella.com/articles/EclipseEMF/article.html#resources)

[**14.1. Source Code**](http://www.vogella.com/articles/EclipseEMF/article.html#sourcecode)

[**14.2. EMF Resources**](http://www.vogella.com/articles/EclipseEMF/article.html#resources_emf)

[**14.3. vogella Resources**](http://www.vogella.com/articles/EclipseEMF/article.html#resources_general)

**1. Models and Eclipse EMF**

**1.1. Data model**

A data model, sometimes also called domain model, represents the data you want to work with. For example if you develop an online flight booking application you might model your domain model with objects like Person, Flight,Booking etc. The general recommendation is to model your data model independent of the application logic. This approach leads to classes with almost no logic and a lot of properties, e.g. Person would have the properties*firstName*, *lastName*, *Address*, etc.

**1.2. Eclipse EMF**

Eclipse EMF can be used to model your domain model. EMF has a distinction between the meta-model and the actual model. The meta-model describes the structure of the model. A model is then the instance of this meta-model. EMF provides a plugable framework to store the model information, the default uses XMI (XML Metadata Interchange) to persists the model definition.

EMF allows to create the meta-model via different means, e.g. XMI, Java annotations, UML or an XML Schema. The following description will use the EMF tools directly to create a EMF model.

Once the EMF meta-model is specified you can generate the corresponding Java implementations classes from this model. EMF provides the possibility that the generated code can be safely extended by hand.

**1.3. Meta Models - Ecore and Genmodel**

We said earlier that EMF has a meta-model. Actually EMF is based on two meta-models; the Ecore and the Genmodel model. The Ecore metamodel contains the information about the defined classes. The Genmodel contains additional information for the codegeneration, e.g. the path and file information. The genmodel contains also the control parameter how the code should be generated.

The Ecore model allows to define different elements.

* ***EClass*** : represents a class, with zero or more attributes and zero or more references.
* ***EAttribute*** : represents an attribute which has a name and a type.
* ***EReference*** : represents one end of an association between two classes. It has flag to indicate if it represent a containment and a reference class to which it points.
* ***EDataType*** : represents the type of an attribute, e.g. int, float or java.util.Date

The Ecore model shows a root object representing the whole model. This model has children which represents the packages, whose children represents the classes, while the children of the classes represents the attributes of these classes.

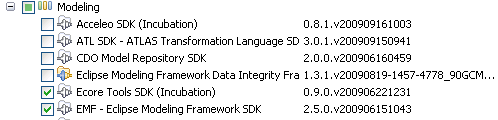
**1.4. Advantages of using EMF**

With EMF you make your domain model explicit which helps to provide clear visibility of the model. EMF also provides change notification functionality to the model in case of model changes. EMF will generate interfaces and factory to create your objects; therefore it helps you to keep your application clean from the individual implementaiton classes.

Another advantages is that you can regenerate the Java code from the model at any point in time.

**2. Installation**

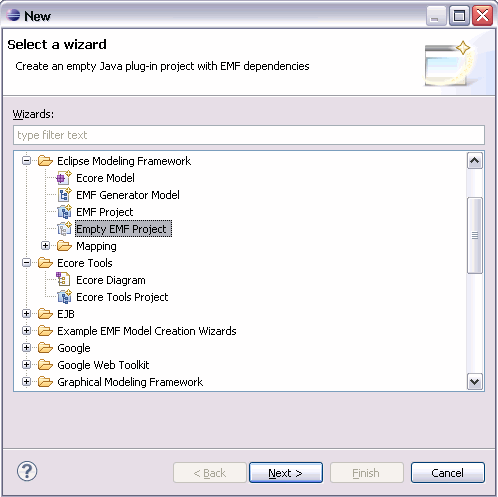
Install EMF via the [**Eclipse Update manager**](http://www.vogella.com/articles/Eclipse/article.html#updatemanager). Select *Modeling* and install *EMF - Eclipse Modeling Framework SDK*. Also select the *Ecore Tools SDK* these will allow you to models based on diagrams.



**3. Define EMF model**

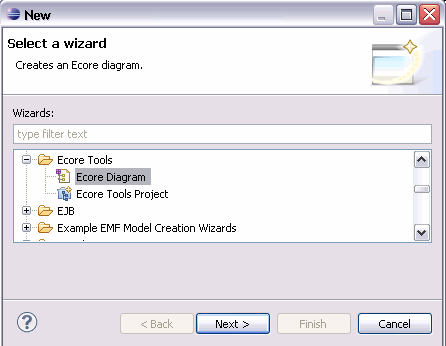
**3.1. Create project**

Create a new project called *de.vogella.emf.webpage.model* via *File* → *New* → *Project...* → *Eclipse Modeling Framework* → *Empty EMF Project*.

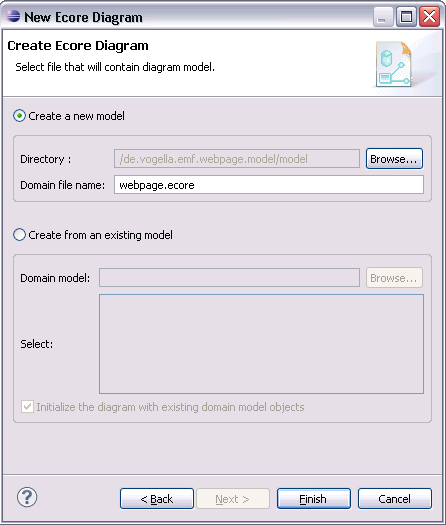


**3.2. Create Ecore diagram**

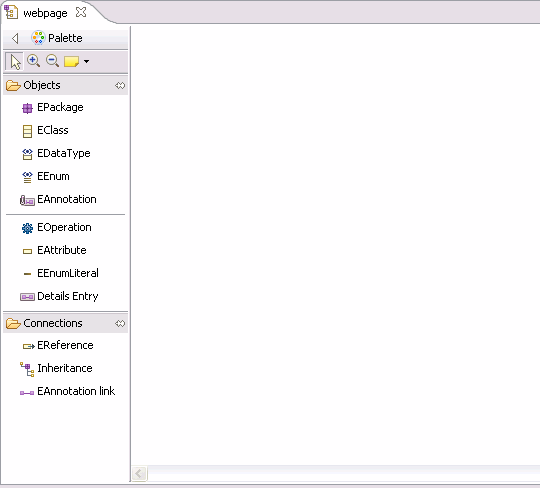
Select the *model* folder, right-click on it and select *New* → *Other...* → *Ecore Tools* → *Ecore Diagram*.



Enter *webpage.ecore* as the *Domain File Name* parameter.

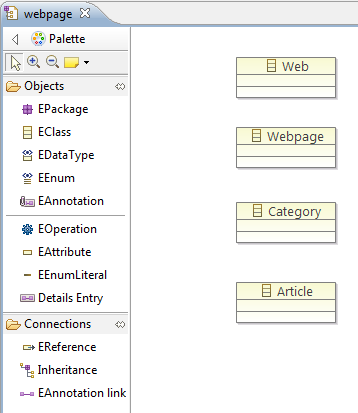


This should open a visual editor for creating EMF models.

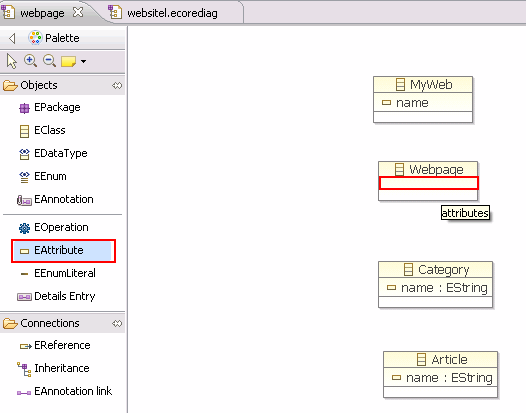


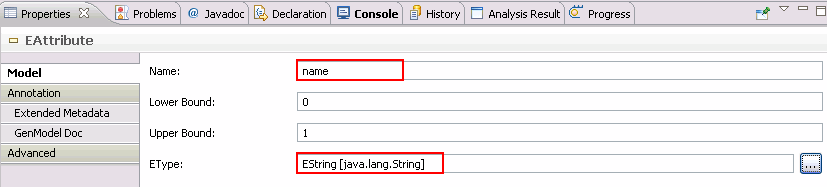
Open the *Properties* view via the menu *Window* → *Show View* → *Other...* → *Properties*. This view allows you to modify the attributes of your model elements.

Click on EClass and click into the editor to create a new class. Create the MyWeb, Webpage, Category and ArticleEClasses.

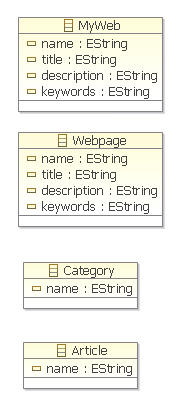


Use the *EAttribute* node to assign to each object the atttribute called *name*. This attribute should have the EStringtype.





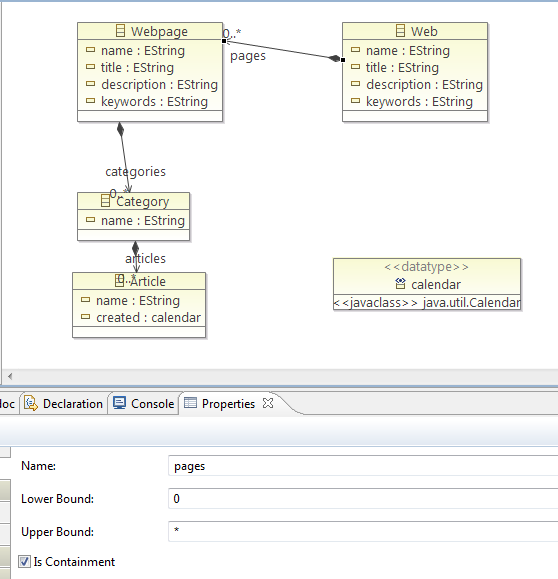
Add the *title*, *description* and *keywords* attributes to the *MyWeb* and *Webpage* model elements.



We want to use the data type calendar in our model. Select "EDataType" and give it the name "calendar" and type "java.util.Calendar". Add the EAttribute "created" to "Article" and use your new type.

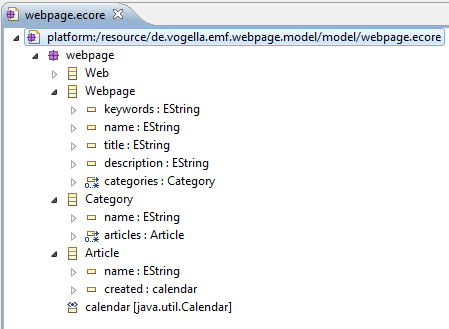


Select EReferences and create an arrow similar to the following picture. Make sure the upper bound is set to "\*" and that the "Is Containment" property is flagged.



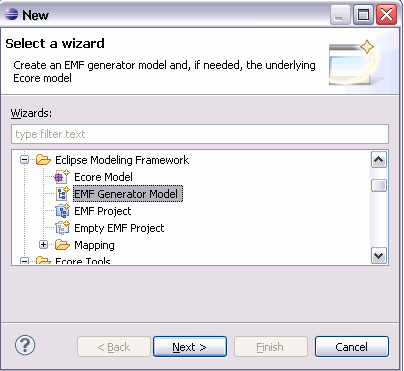
**3.3. View Ecore diagram**

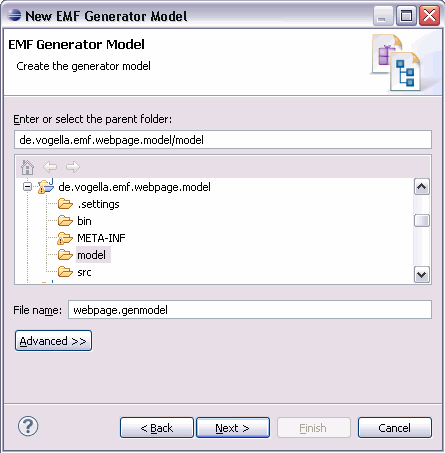
Close the diagram and open the *webpage.ecore* file. The result should look like the following screenshot.

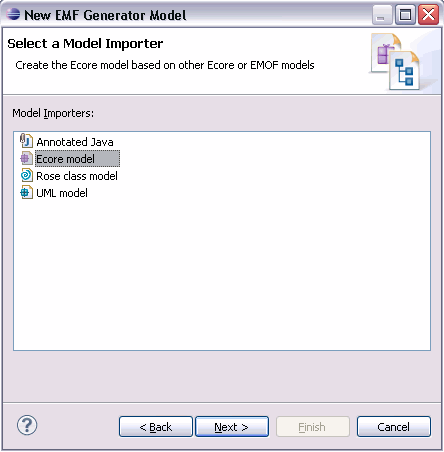


**3.4. Create EMF Generator Model**

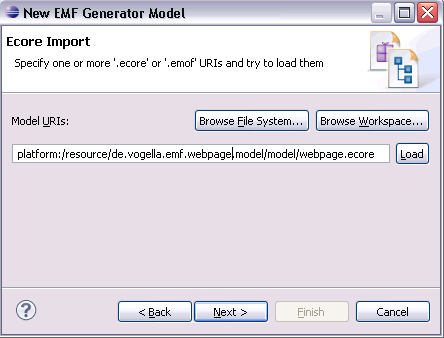
Right-click your *webpage.ecore* file and select *File* → *New* → *Other...* → *EMF Generator model*. Create the*webpage.genmodel* file based on your *Ecore model*.

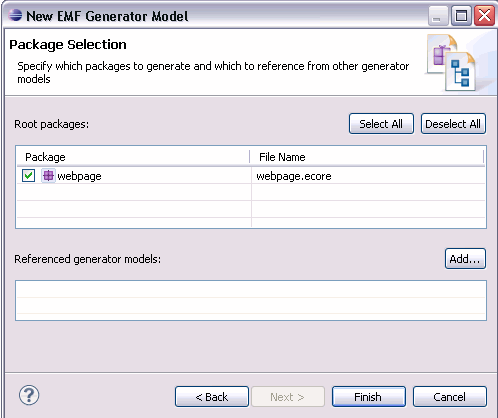






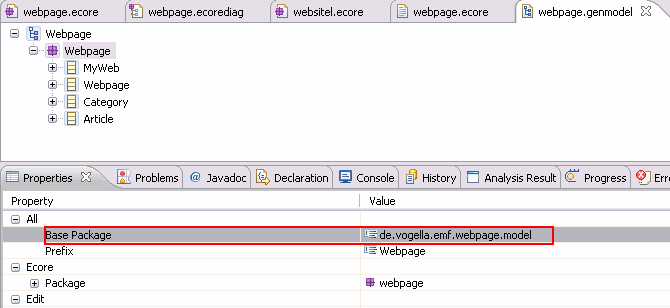
Select your model and press load.





**3.5. Set the package**

Open the *webpage.genmodel* and select the *Webpage* node. Set the *base package* property to *de.vogella.emf.webpage.model*.

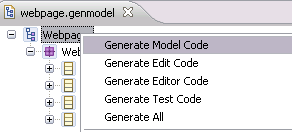


**4. Generating the domain classes**

**4.1. Generating Java code**

You have created two model files, the *.ecore* and the *.genmodel* model. Based on these two model files you can generate Java code.

Right-click on the root node of the *.genmodel* file and select *Generate Model Code*. This will create the Java implementation of the EMF model in the current project.



**4.2. Review the generated code**

The generated code will consists of the following:

* model -- Interfaces and the Factory to create the Java classes
* model.impl -- Concrete implementation of the interfaces defined in model
* model.util -- The AdapterFactory

The central factory has methods for creating all defined objects via createObjectName() methods.

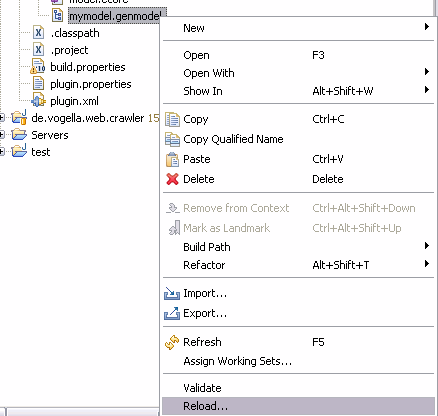
For each attribute the generated interface and its implementation contains getter and (if allowed in the model definition) setter methods. Each setter has also a generated notification to observers of the model. This mean that other object can attach them to the model and react to changes in the model.

Each generated interface extends the EObject interface . EObject is the base of every EMF class and is the EMF equivalent of java.lang.Object. EObject and its corresponding implementation class EObjectImpl provide a lightweight base class that lets the generated interfaces and classes participate in the EMF notification and persistence frameworks.

Every generated method is tagged with @generated. If you want to manually adjust the method you want to prevent that EMF overwrites the method during the next generation run you have to remove this tag.

**4.3. Updating the model**

If you changes your *.ecore* model then you cann update the *.genmodel* by reloading.



**5. Create EMF Editor plug-ins**

EMF can generate plug-ins which provide wizards for creating new model instances and an editor which allows you to enter your model information.

The following assumes that you have already have knowledge in developing Eclipse plug-ins. For more information about Eclipse plug-in development please see **[Eclipse Plugin Tutorial](http://www.vogella.com/articles/EclipsePlugIn/article.html" \t "_top)**

**5.1. Generating edit / editor code**

Eclipse EMF allow you to create a editor for your model. Select your *.genmodel* file , right-click on it and select*Generate Edit Code*and afterwards *Generate Editor Code*.

Two **[Eclipse plugin](http://www.vogella.com/articles/EclipsePlugIn/article.html" \t "_top)** projects have been created, "de.vogella.emf.webpage.model.edit" and "de.vogella.emf.webpage.model.editor".

**5.2. Run your plugins**

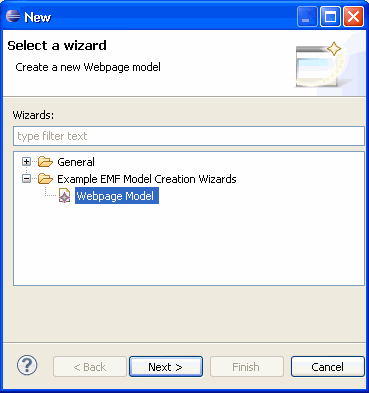
Select the \*.editor project and start a new Eclipse instance with your new plug-in via right mouse-click on it and by selecting *Run-As* → *Eclipse application*.

This should start a new Eclipse runtime instance.

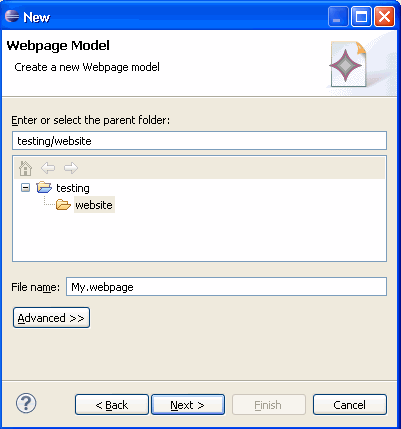
**5.3. Create your model**

In new Eclipse instance create a new project of type *General* called *testing* and a folder called *website*.

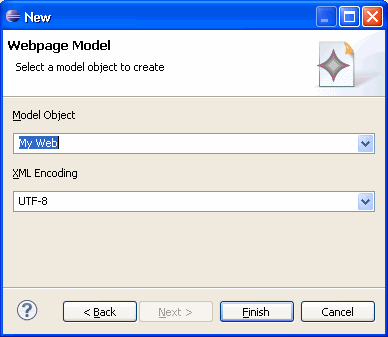
Select this folder, right click on it, select *New* → *Other...* → *Example EMF Model Creation Wizards*→ *Webpage Model*.



Name your model *My.webpage*.



Select as the Model Object "My Web" and press finish.

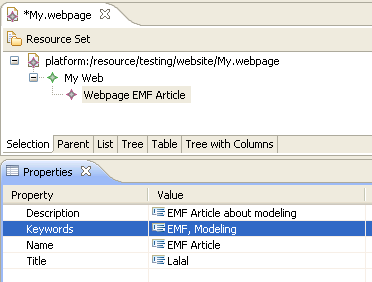


**5.4. Edit your model**

You should now see a editor for your website.model.

Right-click on "My Web" and create a new elements. To edit the elements use the "Properties View" which can be found under Window -> Show View -> Properties.





Save your created model.

**6. Using the model code**

**6.1. Overview**

The generated model code is standard Java code and can be used as such. The following demonstrates how you create objects based on the generated code.

**6.2. Example**

Create a new plug-in project called *de.vogella.emf.webpage.usingmodel*. Add the following dependency to your*MANIFEST.MF*.

* org.eclipse.emf.ecore
* de.vogella.emf.webpage.model

Create the following class.

**package** de.vogella.emf.webpage.usingmodel;

**import** de.vogella.emf.webpage.model.webpage.MyWeb;

**import** de.vogella.emf.webpage.model.webpage.Webpage;

**import** de.vogella.emf.webpage.model.webpage.WebpageFactory;

**import** de.vogella.emf.webpage.model.webpage.impl.WebpagePackageImpl;

**public** **class** UsingEMFModel {

**public** **static** **void** main(String[] args) {

WebpagePackageImpl.init();

*// Retrieve the default factory singleton*

WebpageFactory factory = WebpageFactory .eINSTANCE;

*// create an instance of myWeb*

MyWeb myWeb = factory.createMyWeb();

myWeb.setName("Hallo");

myWeb.setDescription("This is a description");

*// create a page*

Webpage webpage = factory.createWebpage();

webpage.setTitle("This is a title");

*// add the page to myWeb*

myWeb.getPages().add(webpage);

*// and so on, and so on*

*// as you can see the EMF model can be (more or less) used as standard Java*

}

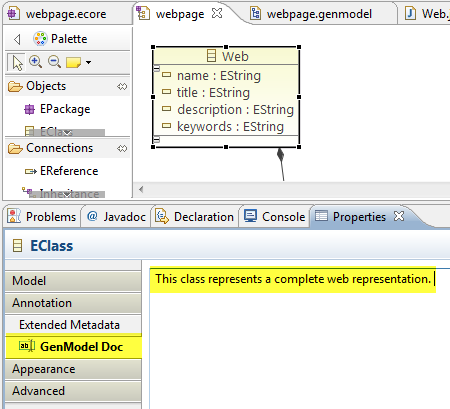
}

**Tip**

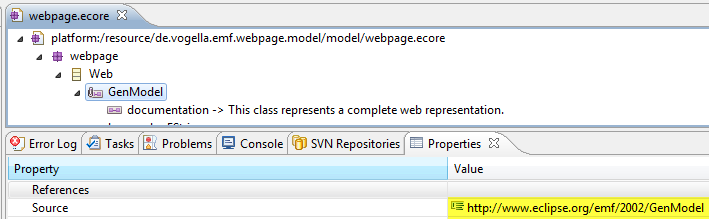
The \*PackageImpl.init() method needs to be called before doing anything else as this method initializes the model and the listeners.

**7. Creating JavaDoc**

You can also generate Javadoc for your classes and methods. EMF uses annotations for this with a certain property key. The easiest way of adding this is again the diagram. Select a class and maintain the documentation in the "GenModel Doc".



The ecore model looks now like the following. The key in the annotation "http://www.eclipse.org/emf/2002/GenModel" is necessary and the key on the details enty must be "documentation".



**8. Generating methods**

By default EMF generates getter and setter for every class. You can also add Operations or for example overwrite methods, e.g. the toString() method. For Article the following toString method was generated in "ArticleImpl

\* *@generated*

\*/

*@Override*

**public** String toString() {

**if** (eIsProxy()) **return** **super**.toString();

StringBuffer result = **new** StringBuffer(**super**.toString());

result.append(" (name: ");

result.append(name);

result.append(", created: ");

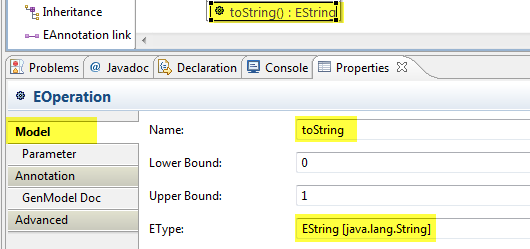
result.append(created);

result.append(')');

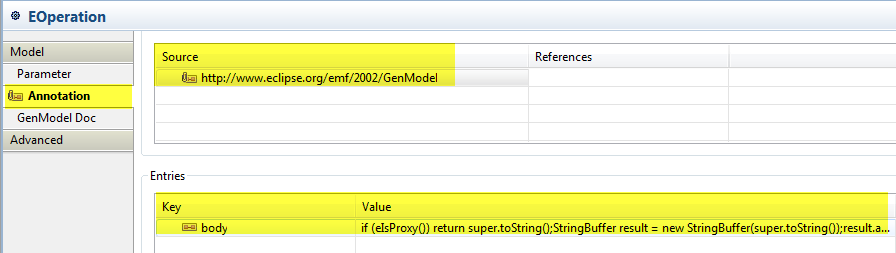
**return** result.toString();

}

To overwrite this, add a "EOperation" to your model with the name toString. Maintain in the properties "EType" EString as return type.



Add an annotation with the source "http://www.eclipse.org/emf/2002/GenModel" and maintain an entry with the key "body", the value is the code that will be generated in to the method, you find it listed below.



**if** (eIsProxy()) **return** **super**.toString();

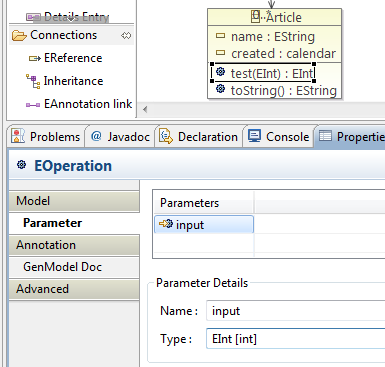
StringBuffer result = **new** StringBuffer(**super**.toString());

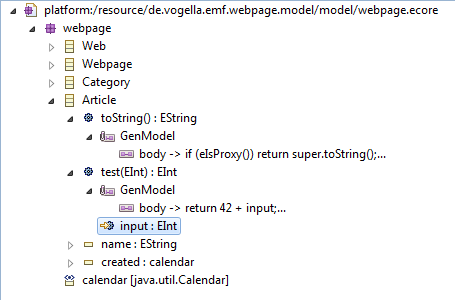
result.append("Article: ");

result.append(name);

**return** result.toString();

You can also generate methods with input parameter, just add parameter with their type to your EOperation.





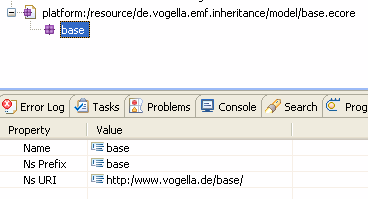
**9. Extending an EMF Ecore model (inheritance)**

**9.1. Overview**

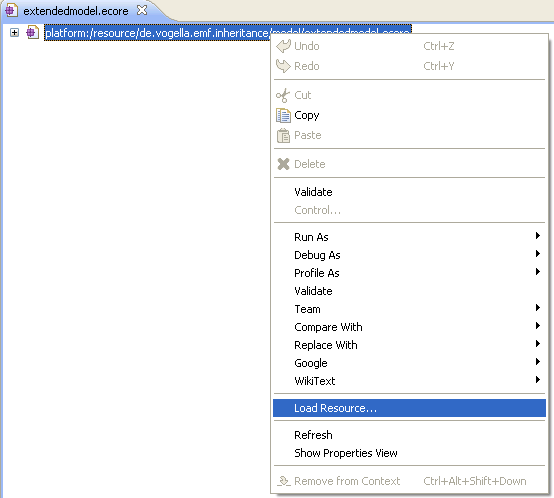
EMF allows to extend existing models via inheritance. The following will define a base model and an extension based on this base model. This can for example be used to extend the **[Eclipse e4](http://www.vogella.com/articles/EclipseE4/article.html" \t "_top)**application model. It will also demonstrate how to work with EMF ecore models directly without using the ecore tools.

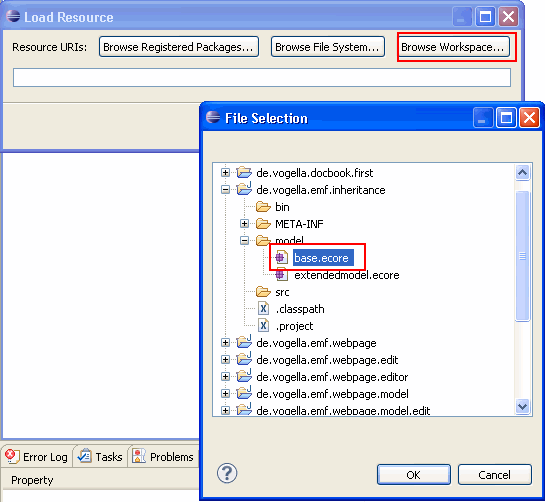
**9.2. Example**

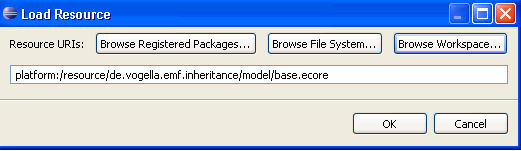
Create a new EMF project "de.vogella.emf.inheritance". Create a new model by selecting File -> New -> "Eclipse Modeling Framework" -> "Ecore Model". Name the model "base.ecore". Select "EPackage" as the basis and maintain the following properties for this package.



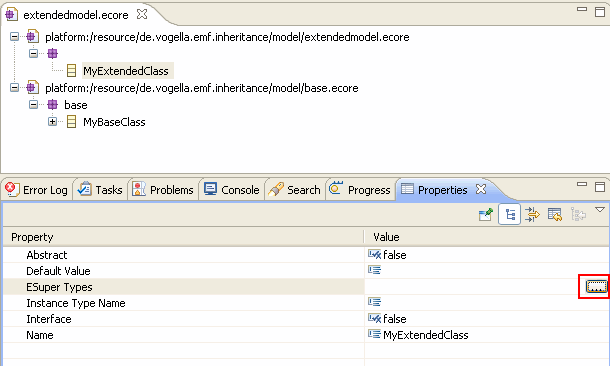
Right-click on the package and select New Child -> EClass. Maintain the class "MyBaseClass" with two "EAttributes" of type "EString". Create a new "Ecore" model "extendedmodel.ecore". Maintain "extended" as the package name. Right-click your model and select "Load resource".



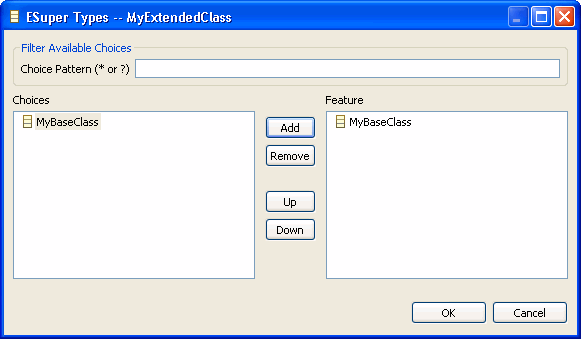


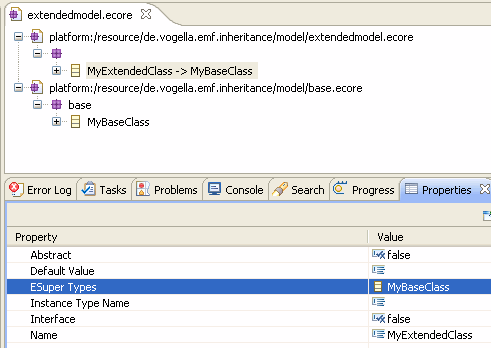


Create a new class "MyExtendedClass" and press "ESuperType".

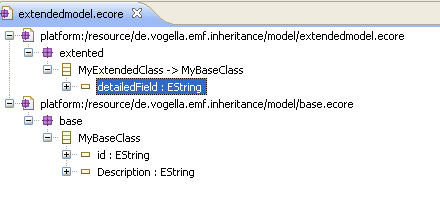


Add your "MyBaseClass".





Maintain a new EAtribute "detailedField" on "MyExtendedClass".



Create a new genmodel "extended.genmodel" based on extended.ecore. Generated Java code and you will see that the "MyExtendedClass" has extended "MyBaseClass".

**10. Setting the empty string as default value**

It is not obvious how to set an empty string as a default value for an EMF string attribute. To set an empty string as default value do the following.

* Select the Attribute
* Im the Property View click into the value field of "Default Value Literal"
* Do not enter something.

To remove this empty value again, click "Restore Default Value" in the toolbar.

**11. Next steps**

Please check the appendix for pointers to more advanced Eclipse EMF topics.

**12. Thank you**

Please help me to support this article:

|  |  |
| --- | --- |
| **[Flattr this](http://flattr.com/thing/49772/Tutorials-for-Eclipse-Java-Android-and-Webprogramming)** | 窗体顶端   https://www.paypal.com/en_US/i/scr/pixel.gif  窗体底端 |

**13. Questions and Discussion**

If you find errors in this tutorial, please notify me (see the top of the page). Please note that due to the high volume of feedback I receive, I cannot answer questions to your implementation. Ensure you have read the **[vogella FAQ](http://www.vogella.com/faq.html" \t "_top)** as I don't respond to questions already answered there.

**14. Links and Literature**

**14.1. Source Code**

**[Source Code of Examples](http://www.vogella.com/code/index.html" \t "_top)**

**14.2. EMF Resources**

**[http://www.eclipse.org/modeling/emf](http://www.eclipse.org/modeling/emf" \t "_top)**Eclipse EMF Homepage

**[http://www.eclipse.org/modeling/emf/docs/](http://www.eclipse.org/modeling/emf/docs/" \t "_top)**EMF Documentation

**[http://www.ibm.com/developerworks/opensource/library/os-ecemf1](http://www.ibm.com/developerworks/opensource/library/os-ecemf1" \t "_top)**Model with the Eclipse Modeling Framework, Part 1: Create UML models and generate code

**[http://www.ibm.com/developerworks/opensource/library/os-ecemf2](http://www.ibm.com/developerworks/opensource/library/os-ecemf2" \t "_top)**Model with the Eclipse Modeling Framework, Part 2: Generate code with Eclipse's Java Emitter Templates

**[http://www.raceeend.demon.nl/](http://www.raceeend.demon.nl/" \t "_top)**How to extend / inheritant from an existing EMF model

**[http://www.eclipse.org/m2m/atl/](http://www.eclipse.org/m2m/atl/" \t "_top)**ATL allows model to model transformation for EMF

**14.3. vogella Resources**

**[vogella Training](http://training.vogella.com/index.html" \t "_top)** Android and Eclipse Training from the vogella team

**[Android Tutorial](http://www.vogella.com/articles/Android/article.html" \t "_top)** Introduction to Android Programming

**[GWT Tutorial](http://www.vogella.com/articles/GWT/article.html" \t "_top)** Program in Java, compile to JavaScript and HTML

**[Eclipse RCP Tutorial](http://www.vogella.com/articles/EclipseRCP/article.html" \t "_top)** Create native applications in Java

**[JUnit Tutorial](http://www.vogella.com/articles/JUnit/article.html" \t "_top)** Test your application

**[Git Tutorial](http://www.vogella.com/articles/Git/article.html" \t "_top)** Put all your files in a distributed version control system