Xtreme Eclipse 4

Paul Webster, Sopot Cela, Lars Vogel

“Because knowledge feels like **magic**”

Xtreme Eclipse 4: Paul Webster, Sopot Cela, Lars Vogel

Lars Vogel

Third edition (version 99.1)

Publication date 02.03.2014

Part I. Model add-ons

1 Exercises: Using Eclipse selection service

1.1. Create project and use the Eclipse selection service

Create a new Eclipse 4 RCP application using the wizard and ensure that the sample content is created.

Add another part to your application pointing to the `ListenerPart` class, which should look similar to the following listing.

```
package com.vogella.e4.selectionservice.parts;

import javax.annotation.PostConstruct;
import javax.inject.Inject;
import javax.inject.Named;

import org.eclipse.e4.core.di.annotations.Optional;
import org.eclipse.e4.ui.services.IServiceConstants;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;

public class ListenerPart {

  private Label labelSelection;

  @PostConstruct
  public void createComposite(Composite parent) {
    parent.setLayout(new GridLayout(1, false));
    labelSelection = new Label(parent, SWT.NONE);
    labelSelection.setLayoutData(new GridData(SWT.BEGINNING, SWT.CENTER,
        false, false));
    labelSelection.setText("Initial test");
  }

  @Inject
  @Optional
  public void setSelection(@Named(IServiceConstants.ACTIVE_SELECTION) String selection) {
    if (labelSelection != null && !labelSelection.isDisposed()) {
      labelSelection.setText(selection);
    }
  }

}
```

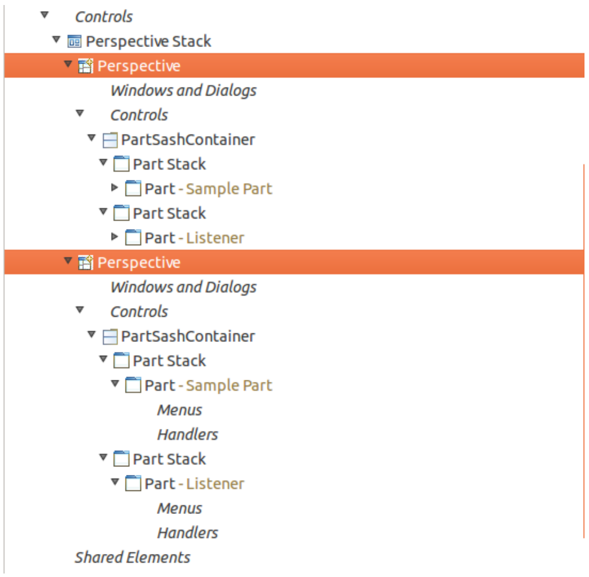
Adjust your `SamplePart` to send out the selection.

```
package com.vogella.e4.selectionservice.parts;

import javax.annotation.PostConstruct;
import javax.inject.Inject;

import org.eclipse.e4.ui.di.Focus;
import org.eclipse.e4.ui.di.Persist;
import org.eclipse.e4.ui.model.application.ui.MDirtyable;
import org.eclipse.e4.ui.workbench.modeling.ESelectionService;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.ModifyEvent;
import org.eclipse.swt.events.ModifyListener;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Text;

public class SamplePart {

  private Text txtInput;
  private TableViewer tableViewer;

  @Inject
  private MDirtyable dirty;
  @Inject ESelectionService service;

  @PostConstruct
  public void createComposite(Composite parent) {
    parent.setLayout(new GridLayout(1, false));

    txtInput = new Text(parent, SWT.BORDER);
    txtInput.setMessage("Enter text to mark part as dirty");
    txtInput.addModifyListener(new ModifyListener() {
      @Override
      public void modifyText(ModifyEvent e) {
        dirty.setDirty(true);
        service.setSelection(txtInput.getText());
      }
    });
    txtInput.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));

    tableViewer = new TableViewer(parent);

    tableViewer.add("Sample item 1");
    tableViewer.add("Sample item 2");
    tableViewer.add("Sample item 3");
    tableViewer.add("Sample item 4");
    tableViewer.add("Sample item 5");
    tableViewer.getTable().setLayoutData(new GridData(GridData.FILL_BOTH));
  }

  @Focus
  public void setFocus() {
    tableViewer.getTable().setFocus();
  }

  @Persist
  public void save() {
    dirty.setDirty(false);
  }
}
```

Start your application and ensure the text entered in the first part is propagated to the second part.

1.2. Understand Eclipse selection service

The Eclipse selection service sets the selection for the current window. Verify that by adding a new perspective your your application.



Set the selection and switch between these perspectives with following handler.

```
package com.vogella.e4.selectionservice.handlers;

import java.util.List;

import org.eclipse.e4.core.di.annotations.Execute;
import org.eclipse.e4.ui.model.application.ui.advanced.MPerspective;
import org.eclipse.e4.ui.model.application.ui.basic.MWindow;
import org.eclipse.e4.ui.workbench.modeling.EModelService;
import org.eclipse.e4.ui.workbench.modeling.EPartService;

public class SwitchPerspectiveHandler {
  @Execute
  public void execute(MWindow window, EPartService partService,
      EModelService modelService) {
    // assumes you have only two perspectives
    List<MPerspective> perspectives = modelService.findElements(window,
        null, MPerspective.class, null);
    if (perspectives.size() != 2) {
      System.out.println("works only for exactly two perspectives");
    }

    MPerspective activePerspective = modelService
        .getActivePerspective(window);
    if (activePerspective.equals(perspectives.get(0))) {
      partService.switchPerspective(perspectives.get(1));
    } else {
      partService.switchPerspective(perspectives.get(0));
    }

  }
}
```

Verify that the selection in both parts based on `ListenerPart` is the same.

2 Exercises: Development a custom selection service

2.1. Develop custom selection service

In the project you created in Section 1.1, “Create project and use the Eclipse selection service” create the following class, which is a simple version of a new custom selection service, which uses the context to persists the selection.

```
package com.vogella.e4.selectionservice.service;

import javax.inject.Inject;

import org.eclipse.e4.core.contexts.IEclipseContext;
import org.eclipse.e4.ui.workbench.modeling.ESelectionService;

public class MySelectionService {

  public static final String MYSELECTION = "myselection";
  @Inject
  IEclipseContext ctx;

  public void setSelection(Object obj) {
    ctx.set(MYSELECTION, obj);
  }
}
```

2.2. Create the context function implementation

Create the following class which will be used as context function.

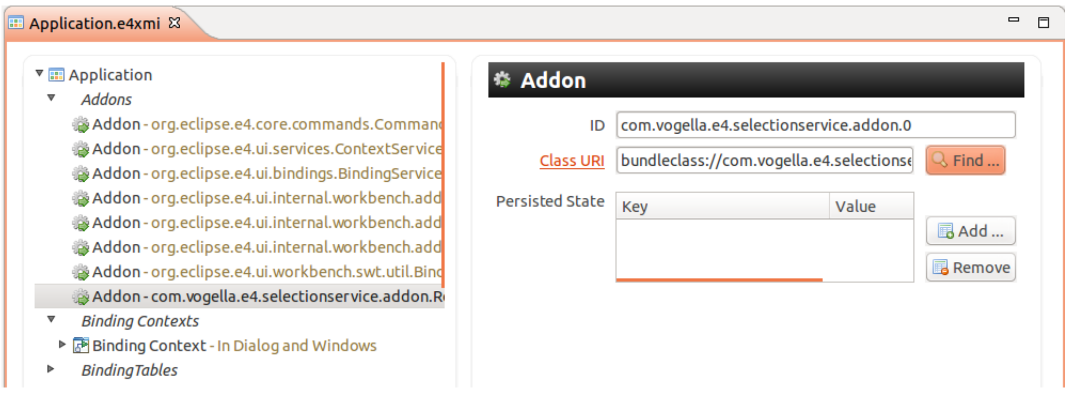
```
package com.vogella.e4.selectionservice.service;

import org.eclipse.e4.core.contexts.ContextFunction;
import org.eclipse.e4.core.contexts.ContextInjectionFactory;
import org.eclipse.e4.core.contexts.IEclipseContext;
import org.eclipse.e4.ui.model.application.ui.advanced.MPerspective;
import org.eclipse.e4.ui.workbench.modeling.ESelectionService;

public class MyContextFunction extends ContextFunction {
  @Override
  public Object compute(IEclipseContext context, String contextKey) {
    System.out.println("Called");
    MPerspective mPerspective = context.get(MPerspective.class);

    // no perspective found
    if (mPerspective == null) {
      System.out.println("Null set to the context");
      return null;
    }

    IEclipseContext ctx = mPerspective.getContext();
    MySelectionService service = ContextInjectionFactory.make(MySelectionService.class, ctx);
    ctx.set(MySelectionService.class.getName(),service);

    return service;
  }
}
```

2.3. Register context function via model add-on

Define the following class.

```
package com.vogella.e4.selectionservice.addon;

import javax.annotation.PostConstruct;
import javax.inject.Inject;

import org.eclipse.e4.core.contexts.IEclipseContext;
import org.eclipse.e4.core.di.annotations.Optional;
import org.eclipse.e4.ui.di.UIEventTopic;
import org.eclipse.e4.ui.model.application.MApplication;
import org.eclipse.e4.ui.workbench.UIEvents;
import org.osgi.service.event.Event;

import com.vogella.e4.selectionservice.service.MyContextFunction;
import com.vogella.e4.selectionservice.service.MySelectionService;

public class ReplaceSelectionService {

  @PostConstruct
  public void replace(MApplication app) {
    System.out.println("Model add-on called");
    IEclipseContext appContext = app.getContext();
    appContext.set(MySelectionService.class.getName(),
        new MyContextFunction());
  }

}
```

This class is registered as model add-on.


Application.e4xmi

- Application
  - Addons
    - Addon - org.eclipse.e4.core.commands.Comman
    - Addon - org.eclipse.e4.ui.services.ContextService
    - Addon - org.eclipse.e4.ui.bindings.BindingService
    - Addon - org.eclipse.e4.ui.internal.workbench.add
    - Addon - org.eclipse.e4.ui.internal.workbench.add
    - Addon - org.eclipse.e4.ui.internal.workbench.add
    - Addon - org.eclipse.e4.ui.workbench.swt.util.Binc
    - Addon - com.vogella.e4.selectionservice.addon.R
  - Binding Contexts
    - Binding Context - In Dialog and Windows
  - BindingTables

Addon

ID: com.vogella.e4.selectionservice.addon.0

Class URI: bundleclass://com.vogella.e4.selectionse

Persisted State

Key	Value

2.4. Update your parts to use the new selection service

Adjust your parts, so that they use the new service.

```
package com.vogella.e4.selectionservice.parts;

import javax.annotation.PostConstruct;
import javax.inject.Inject;
import javax.inject.Named;

import org.eclipse.e4.core.di.annotations.Optional;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;

import com.vogella.e4.selectionservice.service.MySelectionService;

public class ListenerPart {
  private Label labelSelection;

  @PostConstruct
  public void createComposite(Composite parent) {
    parent.setLayout(new GridLayout(1, false));
    labelSelection = new Label(parent, SWT.NONE);
    labelSelection.setLayoutData(new GridData(SWT.BEGINNING, SWT.CENTER,
        false, false));
    labelSelection.setText("Initial test");
  }

  @Inject
  public void setSelection(@Optional @Named(MySelectionService.MYSELECTION) String selection) {
    if (labelSelection != null && !labelSelection.isDisposed()) {
      labelSelection.setText(selection);
    }
  }

}
```

Adjust your `SamplePart` to send out the selection.

```
package com.vogella.e4.selectionservice.parts;

import javax.annotation.PostConstruct;
import javax.inject.Inject;

import org.eclipse.e4.core.di.annotations.Optional;
import org.eclipse.e4.ui.di.Focus;
import org.eclipse.e4.ui.di.Persist;
import org.eclipse.e4.ui.model.application.ui.MDirtyable;
import org.eclipse.e4.ui.workbench.modeling.ESelectionService;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.ModifyEvent;
import org.eclipse.swt.events.ModifyListener;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Text;

import com.vogella.e4.selectionservice.service.MySelectionService;

public class SamplePart {

  private Text txtInput;
  private TableViewer tableViewer;

  @Inject
  private MDirtyable dirty;
  private MySelectionService service;

  @PostConstruct
  public void createComposite(Composite parent) {
    parent.setLayout(new GridLayout(1, false));

    txtInput = new Text(parent, SWT.BORDER);
    txtInput.setMessage("Enter text to mark part as dirty");
    txtInput.addModifyListener(new ModifyListener() {
      @Override
      public void modifyText(ModifyEvent e) {
        dirty.setDirty(true);
        System.out.println(service.toString());
        service.setSelection(txtInput.getText());
      }
    });
    txtInput.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));

    tableViewer = new TableViewer(parent);

    tableViewer.add("Sample item 1");
    tableViewer.add("Sample item 2");
    tableViewer.add("Sample item 3");
    tableViewer.add("Sample item 4");
    tableViewer.add("Sample item 5");
    tableViewer.getTable().setLayoutData(new GridData(GridData.FILL_BOTH));
  }

  @Inject @Optional
  public void setSelectionService(MySelectionService service) {
    System.out.println("Selection service set");
    if (service != null){
      this.service = service;

    } else {
      System.out.println("Service is null");
    }
  }

  @Focus
  public void setFocus() {
    tableViewer.getTable().setFocus();
  }

  @Persist
  public void save() {
    dirty.setDirty(false);
  }
}
```

Part II. Application model extensions

3
Extend the Eclipse application model

3.1. Application model extensibility

3.1.1. The application model structure

The application model you create for your Eclipse based application is based on an Eclipse Modeling Framework (EMF) model defined by the Eclipse platform. This structure defines, for example, that an `MPart` can be stored in an `MPartSashContainer`.

The Eclipse platform uses generated Java classes based on this EMF model. These classes are instantiated based on the `Application.e4xmi` file of the application including formation from model fragments, processors and other sources which modify the model, e.g., mode add-ons.

The Eclipse platform registers listeners on the attributes of these objects, so that it can react immediately if a model property changes.

3.1.2. Using extensions

The underlying structure of the application model can be extended by customers using standard EMF functionality. As the renderer framework is responsible for handling the model elements, you have to define a renderer which is responsible for the new model elements.

4 Exercise: Create and use an application model extension

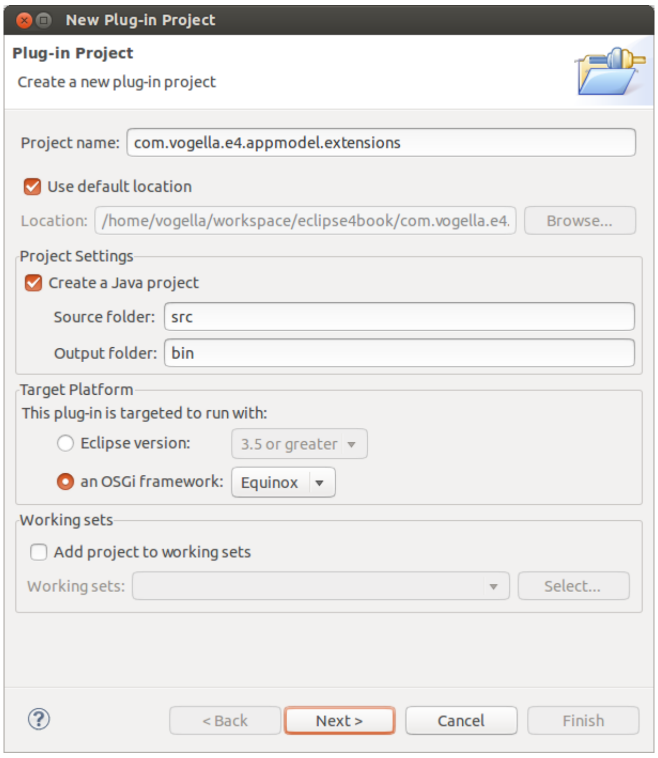
4.1. Exercise: Extend the Eclipse application model

4.1.1. Target

In this exercise you extend the structure of the Eclipse application model with an additional model element.

4.1.2. Create a plug-in and add the dependencies

Create a new simple plug-in called *com.vogella.e4.appmodel.extensions*.



New Plug-in Project

Plug-in Project

Create a new plug-in project

Project name: com.vogella.e4.appmodel.extensions

[✓] Use default location

Location: /home/vogella/workspace/eclipse4book/com.vogella.e4.

Project Settings

[✓] Create a Java project

Source folder: src

Output folder: bin

Target Platform

This plug-in is targeted to run with:

○ Eclipse version: 3.5 or greater

● an OSGi framework: Equinox

Working sets

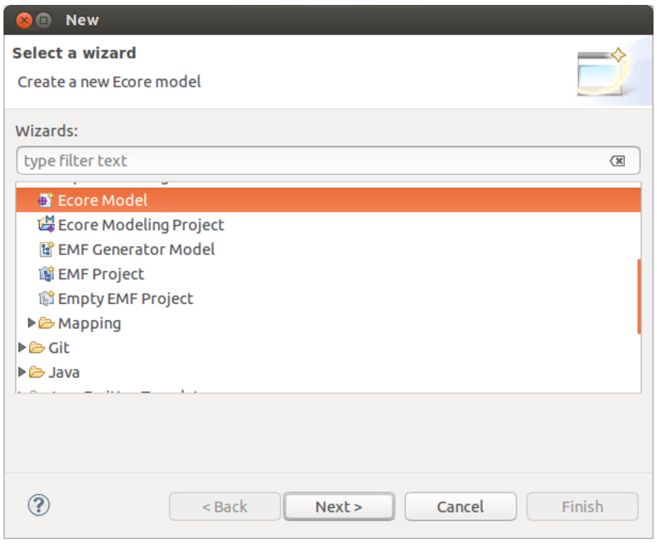
☐ Add project to working sets

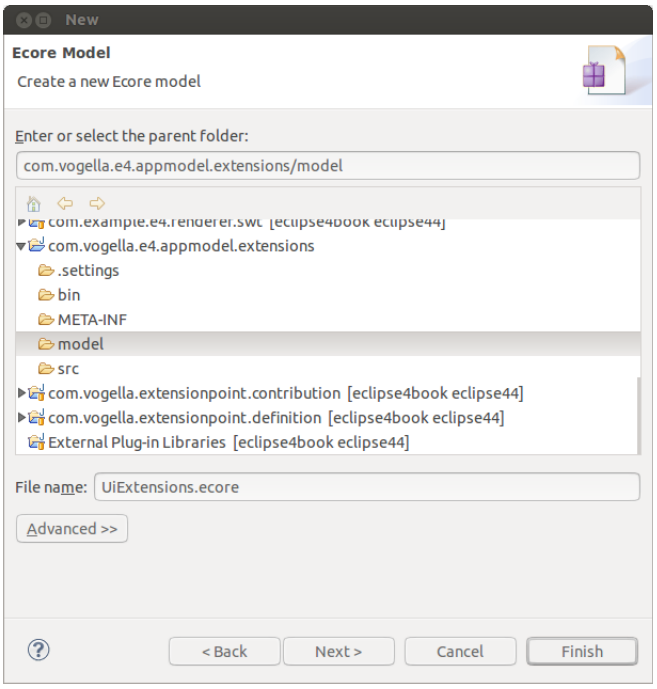
Working sets:

Add a dependency to the plug-ins `org.eclipse.e4.ui.model.workbench` and `org.eclipse.e4.core.contexts` in your new plug-in via its `MANIFEST.MF` file.

4.1.3. Create new model

Create a folder called `model` and create a new EMF model called `UiExtensions.ecore` via *File* → *New* → *Other...* → *Eclipse Modeling Framework* → *EMF Model*.



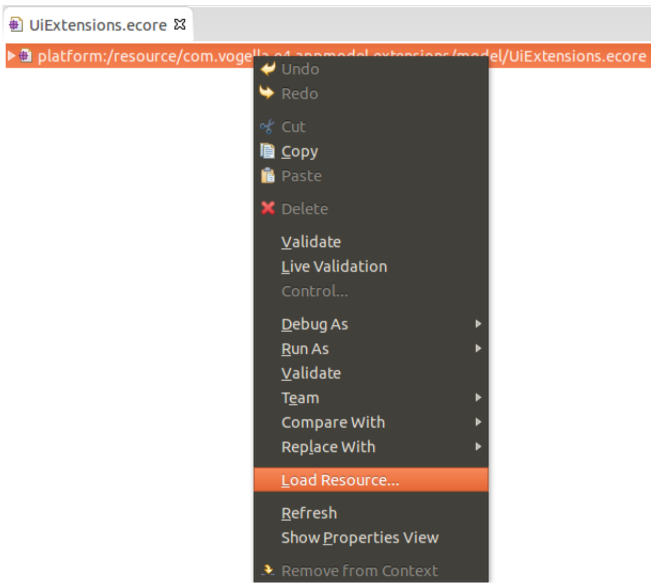

New
Ecore Model
Create a new Ecore model
Enter or select the parent folder:
com.vogella.e4.appmodel.extensions/model
com.example.e4.renderer.swt [eclipse4book eclipse44]
com.vogella.e4.appmodel.extensions
.settings
bin
META-INF
model
src
com.vogella.extensionpoint.contribution [eclipse4book eclipse44]
com.vogella.extensionpoint.definition [eclipse4book eclipse44]
External Plug-in Libraries [eclipse4book eclipse44]
File name: UiExtensions.ecore
Advanced >>
< Back
Next >
Cancel
Finish

Press the *Finish* button.

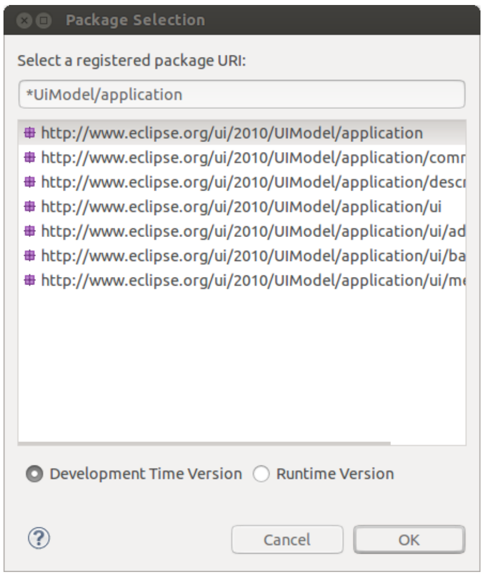
Enter the following properties for this plug-in.

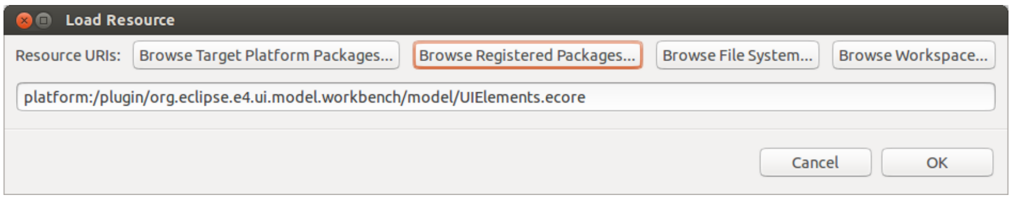
4.1.4. Define relationships to the Eclipse application model

Right-click on the model and select *Load Resource...*.



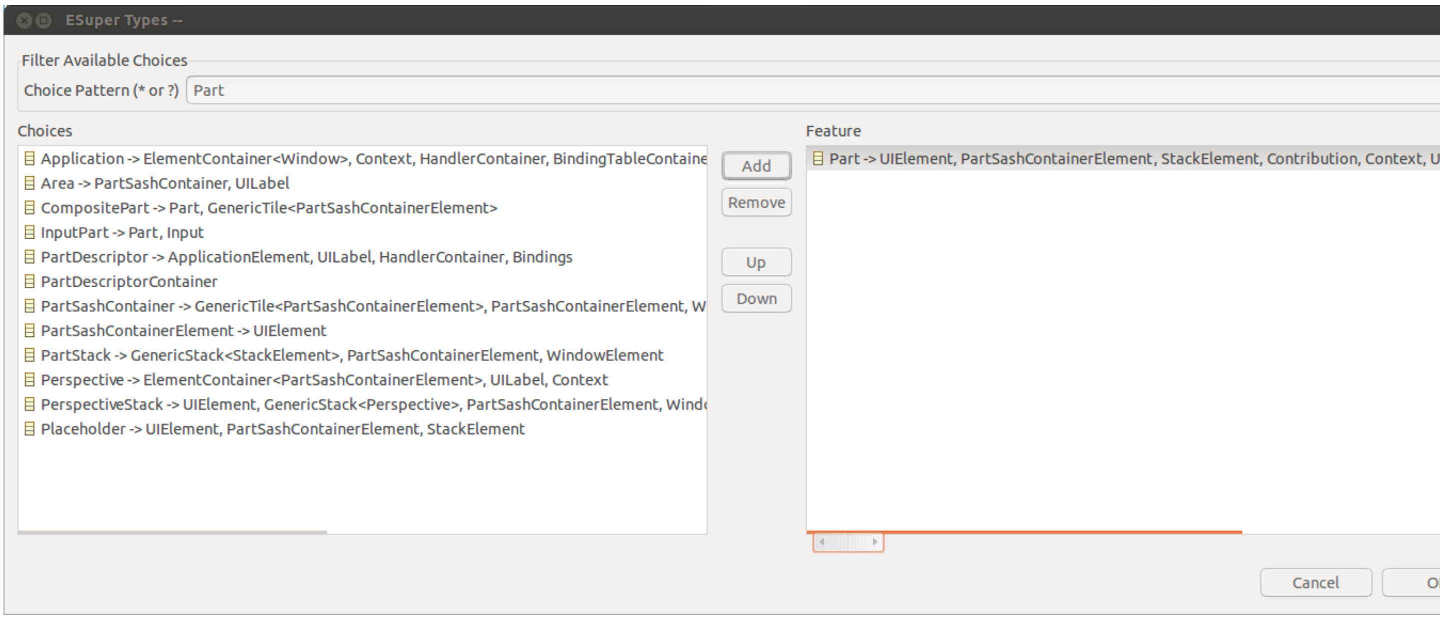
Click on the button *Browse Registered Packages* and select the *`*UiModel/application`* model from the *`org.eclipse.e4.ui.model.workbench`* plug-in.


Package Selection
Select a registered package URI:
*UiModel/application
http://www.eclipse.org/ui/2010/UIModel/application
http://www.eclipse.org/ui/2010/UIModel/application/comr
http://www.eclipse.org/ui/2010/UIModel/application/descr
http://www.eclipse.org/ui/2010/UIModel/application/ui
http://www.eclipse.org/ui/2010/UIModel/application/ui/ad
http://www.eclipse.org/ui/2010/UIModel/application/ui/ba
http://www.eclipse.org/ui/2010/UIModel/application/ui/m
Development Time Version
Runtime Version
Cancel
OK


Load Resource
Resource URIs:
Browse Target Platform Packages...
Browse Registered Packages...
Browse File System...
Browse Workspace...
platform:/plugin/org.eclipse.e4.ui.model.workbench/model/UIElements.ecore
Cancel
OK

4.1.5. Define your custom model extension

Create a new `EClass` model element and add the following `Part` *`ESuper Types`* to it.


ESuper Types --

Filter Available Choices

Choice Pattern (* or ?) Part

Choices

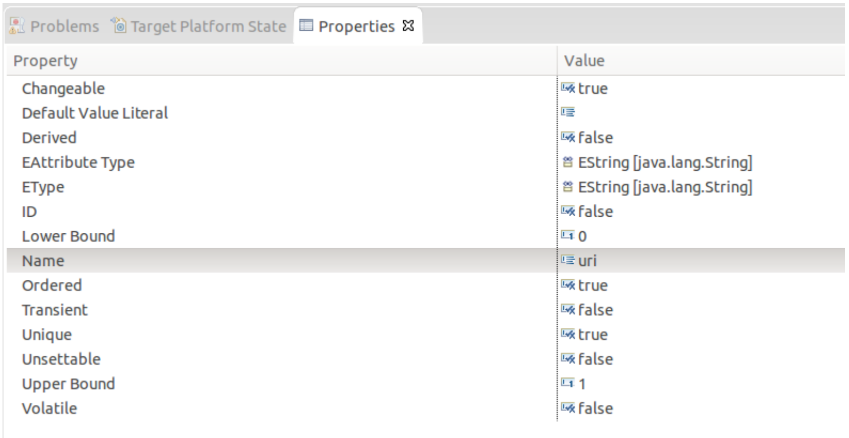
- Application -> ElementContainer<Window>, Context, HandlerContainer, BindingTableContaine
- Area -> PartSashContainer, UILabel
- CompositePart -> Part, GenericTile<PartSashContainerElement>
- InputPart -> Part, Input
- PartDescriptor -> ApplicationElement, UILabel, HandlerContainer, Bindings
- PartDescriptorContainer
- PartSashContainer -> GenericTile<PartSashContainerElement>, PartSashContainerElement, W
- PartSashContainerElement -> UIElement
- PartStack -> GenericStack<StackElement>, PartSashContainerElement, WindowElement
- Perspective -> ElementContainer<PartSashContainerElement>, UILabel, Context
- PerspectiveStack -> UIElement, GenericStack<Perspective>, PartSashContainerElement, Wind
- Placeholder -> UIElement, PartSashContainerElement, StackElement

Feature

- Part -> UIElement, PartSashContainerElement, StackElement, Contribution, Context, U

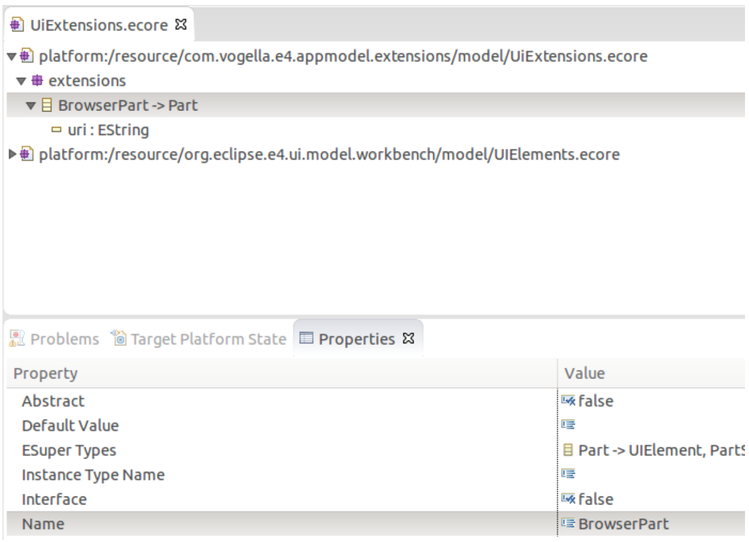
Use *BrowserPart* as value for the `Name` attribute.

Right-click on the model element and add an `EAttribute` called uri with type of `EString` to it.



Property	Value
Changeable	true
Default Value Literal	
Derived	false
EAttribute Type	EString [java.lang.String]
EType	EString [java.lang.String]
ID	false
Lower Bound	0
Name	uri
Ordered	true
Transient	false
Unique	true
Unsettable	false
Upper Bound	1
Volatile	false

The finished model should look like the following.



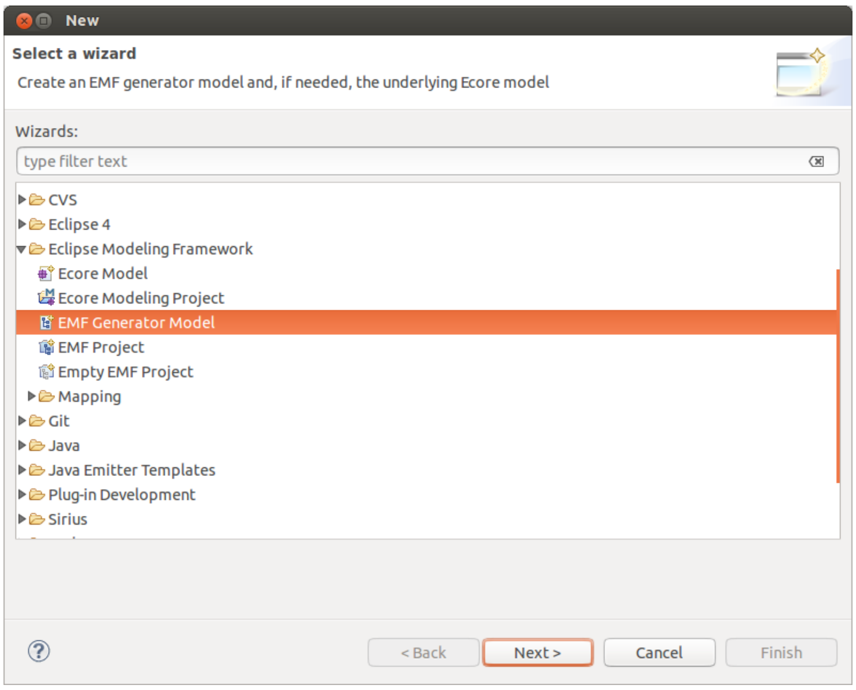
- platform:/resource/com.vogella.e4.appmodel.extensions/model/UiExtensions.ecore
  - extensions
    - BrowserPart -> Part
      - uri : EString
- platform:/resource/org.eclipse.e4.ui.model.workbench/model/UIElements.ecore

Property	Value
Abstract	false
Default Value	
ESuper Types	Part -> UIElement, PartS
Instance Type Name	
Interface	false
Name	BrowserPart

4.1.6. Create the Genmodel

Right-click your `.ecore` file and select *File → New → Other... → EMF Generator model*.

Follow the wizard similar to the following screenshots.


New

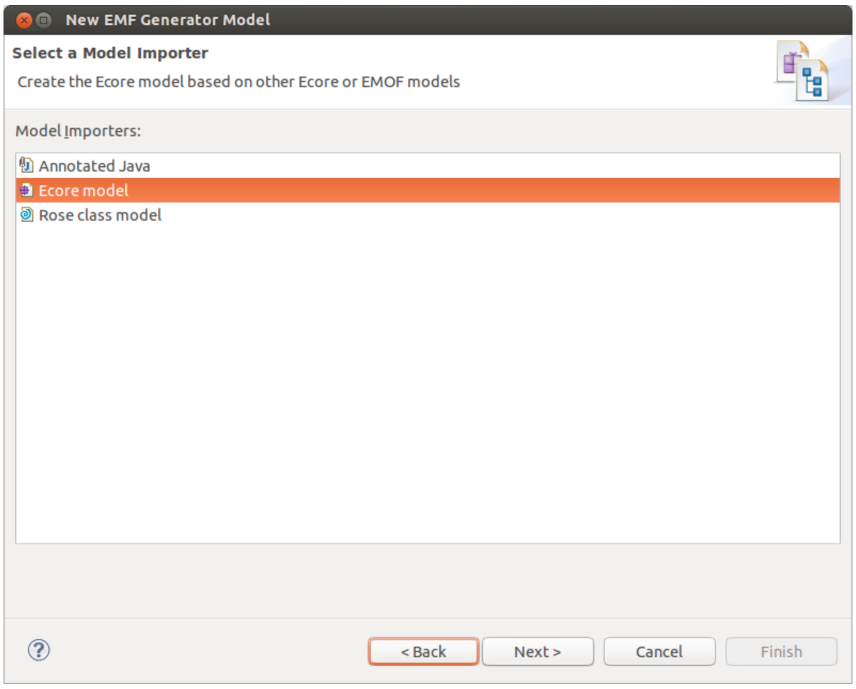
Select a wizard

Create an EMF generator model and, if needed, the underlying Ecore model

Wizards:

type filter text

- CVS
- Eclipse 4
- Eclipse Modeling Framework
  - Ecore Model
  - Ecore Modeling Project
  - EMF Generator Model
  - EMF Project
  - Empty EMF Project
  - Mapping
- Git
- Java
- Java Emitter Templates
- Plug-in Development
- Sirius

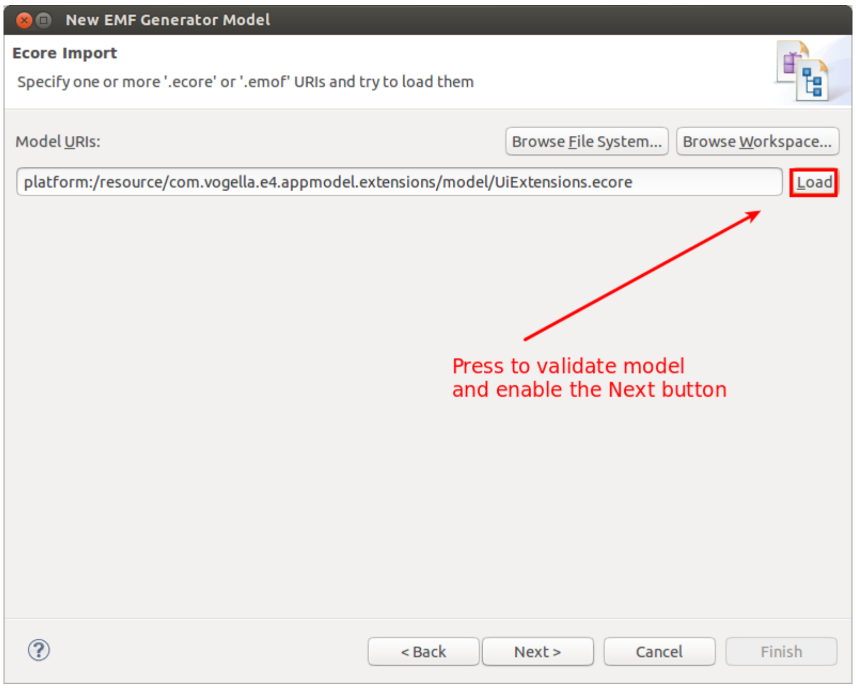

New EMF Generator Model

Select a Model Importer

Create the Ecore model based on other Ecore or EMOF models

Model Importers:

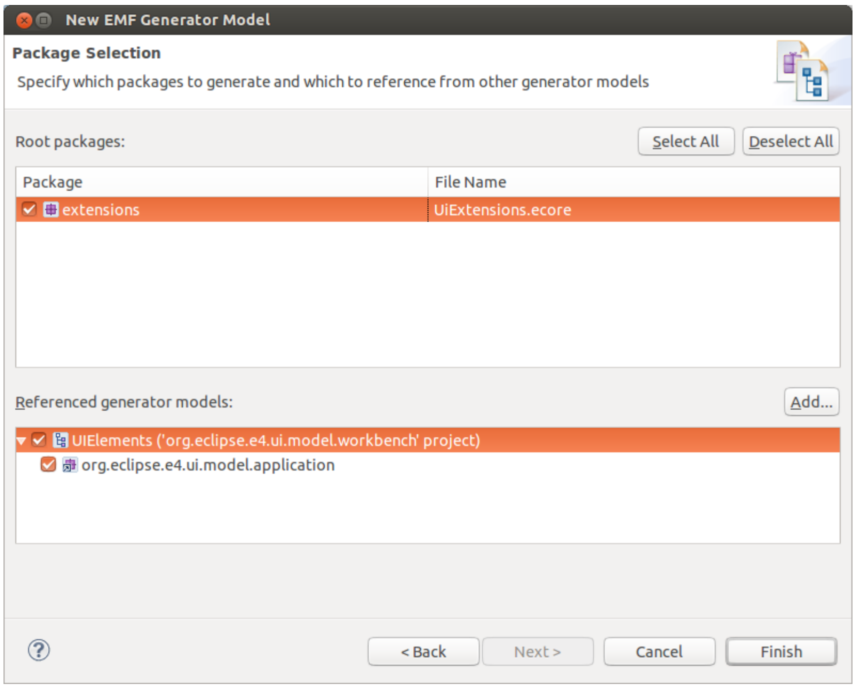
- Annotated Java
- Ecore model
- Rose class model


New EMF Generator Model

Ecore Import

Specify one or more '.ecore' or '.emof' URIs and try to load them

Model URIs: platform:/resource/com.vogella.e4.appmodel.extensions/model/UiExtensions.ecore


New EMF Generator Model

Package Selection

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

Root packages:

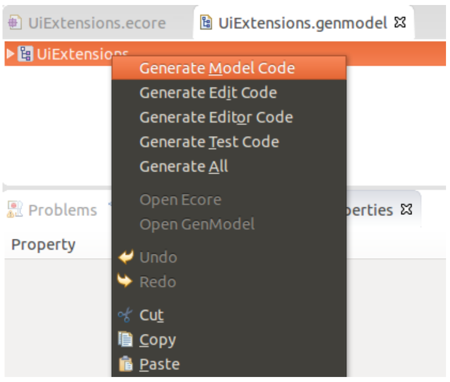
Package	File Name
✓ extensions	UiExtensions.ecore

Referenced generator models:

- ✓ UIElements ('org.eclipse.e4.ui.model.workbench' project)
  - ✓ org.eclipse.e4.ui.model.application

4.1.7. Generate the Java classes

Use your `genmodel` to create the Java code. Right-click on it and select *Generate Model Code*.



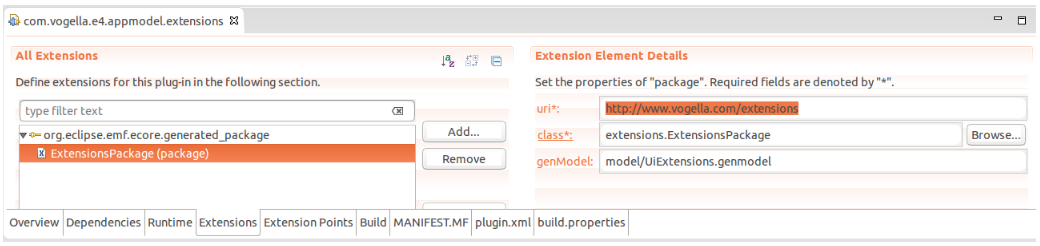
4.1.8. Make generated classes available

Export all package via the Runtime tab on the `MANIFEST.MF` file.

Tip

Please note that the first time you run the model generation, the `plugin.xml` file is created and the `org.eclipse.emf.ecore.generated_package` extension point is added to the file. This extension point describes your model. It is only added if the `plugin.xml` file needs to get created. If you later adjust settings of your mode, i.e., your package setting,you have to adjust the extension point manually.

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>

<!--
-->

<plugin>

   <extension point="org.eclipse.emf.ecore.generated_package">
      <package
            uri="http://www.vogella.com/extensions"
            class="extension.ExtensionPackage"
            genModel="model/UiExtensions.genmodel"/>
   </extension>

</plugin>
```



Define extensions for this plug-in in the following section.

type filter text

org.eclipse.emf.ecore.generated_package
ExtensionsPackage (package)

Extension Element Details

Set the properties of "package". Required fields are denoted by "*".

uri*: http://www.vogella.com/extensions
class*: extensions.ExtensionsPackage
genModel: model/UiExtensions.genmodel

Note down the uri from the extension point. You need to define this in your `Application.e4xmi` later.

4.2. Exercise: Create renderer for new model element

Define a new simple plug-in called *com.vogella.e4.appmodel.renderer*.

Add the following plug-ins as dependencies to your new plug-in.

- `org.eclipse.e4.ui.workbench.renderers.swt`
- `com.vogella.e4.appmodel.extensions`

- `org.eclipse.jface`
- `org.eclipse.e4.ui.model.workbench`
- `org.eclipse.e4.ui.workbench.swt`
- `org.eclipse.e4.core.contexts`

Create the following new class called `BrowserPartRenderer` as renderer for your new `BrowserPart`.

Create the following factory to assign a renderer to your new model element.

```
package com.vogella.e4.appmodel.renderer;

import org.eclipse.e4.ui.internal.workbench.swt.AbstractPartRenderer;
import org.eclipse.e4.ui.model.application.ui.MUIElement;
import org.eclipse.e4.ui.workbench.renderers.swt.WorkbenchRendererFactory;

import extensions.BrowserPart;

public class MyRendererFactory extends WorkbenchRendererFactory {

  private BrowserPartRenderer browserPartRenderer;

  @Override
  public AbstractPartRenderer getRenderer(MUIElement uiElement, Object parent) {

    if (uiElement instanceof BrowserPart) {
      if (browserPartRenderer == null) {
        browserPartRenderer = new BrowserPartRenderer();
        super.initRenderer(browserPartRenderer);
      }
      return browserPartRenderer;
    }

    return super.getRenderer(uiElement, parent);
  }

}
```

4.3. Exercise: Use model extensions

4.3.1. Exercise: Create the application

Create a new Eclipse RCP application called *com.vogella.e4.appmodel.app*.

Add a dependency to the following plug-ins:

- `com.vogella.e4.appmodel.extension`
- `com.vogella.e4.appmodel.renderer`

Note

You have to manually adjust the application model, as the default editors don't support your new model elements. It is possible to develop custom components in the editor to handle your extensions but that is beyond this description.

Add your model name space to the `Application.e4xmi` file. Please note that *`xmlns:extension="http://www.vogella.com/ui/e4/extension"`* defines the UI of your model extension. By this you can use *extension:modelelement* in the file referring to your custom model elements.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<application:Application xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:advanced="http://www.eclipse.org/ui/2010/UIModel/application/ui/advanced"
xmlns:application="http://www.eclipse.org/ui/2010/UIModel/application"
xmlns:basic="http://www.eclipse.org/ui/2010/UIModel/application/ui/basic"
xmlns:menu="http://www.eclipse.org/ui/2010/UIModel/application/ui/menu"
xmlns:extensions="http://www.vogella.com/extensions"
xmi:id="_7VkpUKo8EeO_NLPjBpVjeQ" elementId="org.eclipse.e4.ide.application"
bindingContexts="_7VkpWao8EeO_NLPjBpVjeQ">
... more stuff...
```

Afterwards add your new part implementation to a part, the snippet to add it listed below.

```
<children xsi:type="extensions:BrowserPart"
elementId="com.vogella.e4.appmodel.app.part.0"
uri="http://www.eclipse.org"
label="Test"/>
```

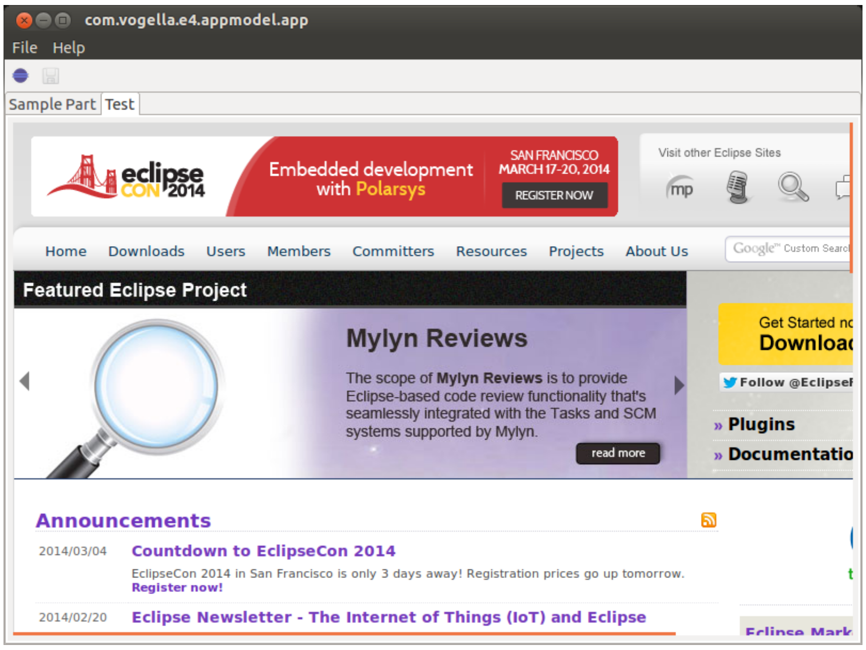
4.3.2. Register your factory via an extension point

As a last step add the property *rendererFactoryUri* to your product in `plugin.xml` with a link to your new factory *`platform:/plugin/de.vogella.e4.renderer.maps/de.vogella.e4.renderer.maps.MyRendererFactory`* .

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>

   <extension
         id="product"
         point="org.eclipse.core.runtime.products">
      <product
            name="com.vogella.e4.appmodel.app"
            application="org.eclipse.e4.ui.workbench.swt.E4Application">
         <property
               name="applicationCSS"
               value="platform:/plugin/com.vogella.e4.appmodel.app/css/default.css">
         </property>
         <property
               name="appName"
               value="com.vogella.e4.appmodel.app">
         </property>
         <property
               name="rendererFactoryUri"
               value="bundleclass://com.vogella.e4.appmodel.renderer/[LINEBREAK]
               com.vogella.e4.appmodel.renderer.MyRendererFactory">
         </property>
      </product>
   </extension>

</plugin>
```

4.3.3. Run your application

Run your application. The result should display the webpage to which you pointed via the *`uri`* parameter.


com.vogella.e4.appmodel.app
File Help
Sample Part Test
eclipse CON 2014
Embedded development with Polarsys
SAN FRANCISCO MARCH 17-20, 2014
REGISTER NOW
Visit other Eclipse Sites
Home Downloads Users Members Committers Resources Projects About Us
Featured Eclipse Project
Get Started no
Download
Follow @EclipseF
» Plugins
» Documentatio
read more
Eclipse Mark

Mylyn Reviews

The scope of **Mylyn Reviews** is to provide Eclipse-based code review functionality that's seamlessly integrated with the Tasks and SCM systems supported by Mylyn.

Announcements

2014/03/04 Countdown to EclipseCon 2014

EclipseCon 2014 in San Francisco is only 3 days away! Registration prices go up tomorrow. Register now!

2014/02/20 Eclipse Newsletter - The Internet of Things (IoT) and Eclipse