Building Project Facets

Summary
The Faceted Project Framework allows the plugin developer to think of projects as composed of units of functionality, otherwise known as facets, that can be added and removed by the user. This tutorial walks you through an example of creating a couple of basic facets and in the process covers the majority of the framework's extension points. Note that this tutorial has been written for the Web Tools Platform v1.5. You will notice extension point differences if you are using the 1.0.x line.

Konstantin Komissarchik
BEA Systems, Inc.
August 30, 2006

Introduction

Faceted Project Framework provides a powerful mechanism for extending the capabilities of the Web Tools Platform. Project facets are typically used as a way of adding functionality to a project. When a facet is added to the project it can perform any necessary setup actions such as copying resources, installing builders, adding natures, etc. Facets can also be used as markers for enabling user interface elements.

Some of the readers may be wondering how facets are different from the project natures which are supported by the Eclipse platform. Natures are designed to be hidden from user's view. They are essentially markers that are set behind the scenes by tooling. Project facets, on the other hand, have been designed from the ground up to be manipulated by the end users. The Faceted Project Framework provides all the necessary user interface elements for managing the facets on a project and since facets are self describing the user is prevented from creating invalid configurations. Basically, it is no longer necessary to write custom project creation wizards or the "Enable Feature X" menu actions. Common way of managing which facets are installed on a project means less burden on the plugin writer and better end user experience.

This tutorial covers the extension points and Java API that are used for creating project facets. The reader is assumed to be already familiar with developing plugins for Eclipse and to have user-level knowledge of the Web Tools Platform.

Tutorial Scenario

You are a developer for a company which develops a product called FormGen. FormGen is basically a servlet that generates HTML forms based on XML definition files. The product is composed of a jar containing the servlet and some basic widgets. There is also an add-on jar with some extra widgets. Users have been manually adding the jars into the WEB-INF/lib directories of their web projects and setting up the servlet definitions in web.xml by hand. Your task is to make this setup easier by writing a couple of project facets that will perform these actions automatically.

Table of Contents

1. Getting Started

To follow along with this tutorial, you will need to have Web Tools Platform v1.5 and it's dependencies installed on top of Eclipse v3.2 installation. You can download the install kits at the following locations:

- Eclipse Platform Download Site
- Web Tools Platform Download Site

Once the required software has been installed you will need to create a new workspace and add the starter project to it. The starter project includes the materials and utility code that will be used in this tutorial. If you get stuck at any point you can take a peek at the solution project.

2. Creating Basic Facet Definitions

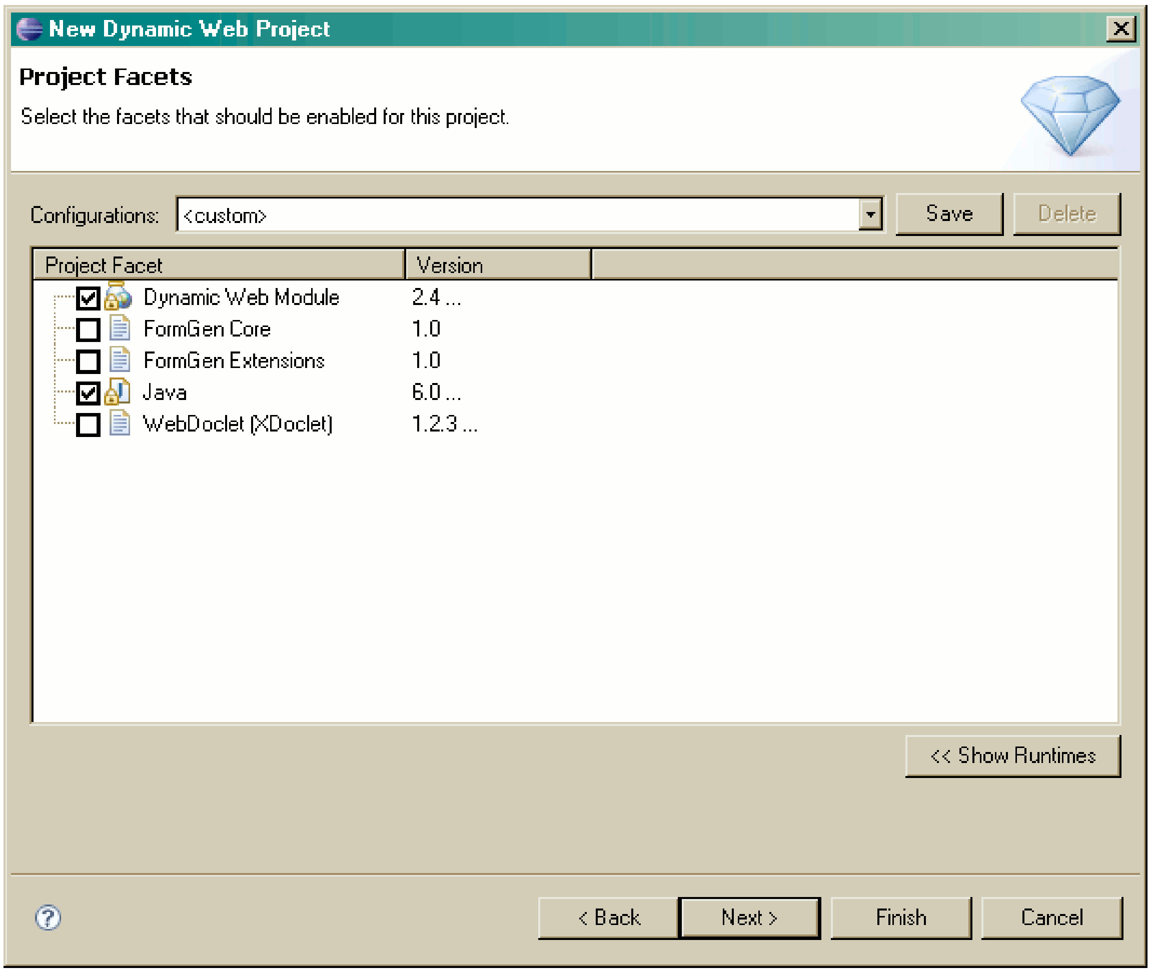
Project facets are declared via the `org.eclipse.wst.common.project.facet.core.facets` extension point. This is a pretty large extension point with lots of facilities, but we will start small and progress incrementally while building the tutorial facets. Here is the part of the schema that we will be working with initially:

```
<extension point="org.eclipse.wst.common.project.facet.core.facets">

  <project-facet id="{string}"> (0 or more)
    <label>{string}</label>
    <description>{string}</description> (optional)
  </project-facet>

  <project-facet-version facet="{string}" version="{string}"/> (0 or more)

</extension>
```

As you can see, there are two top-level elements in this part of the extension point schema. The `<project-facet>` element is used to declare the facet itself. The `<project-facet-version>` element is used to declare versions of the facet. Every facet implementation needs to provide at least one facet version declaration. In fact, as you will see later in this tutorial, the bulk of the work happens in the `<project-facet-version>` declaration. For now all you need to remember is that a facet cannot be used unless it has at least one version.

```
<extension point="org.eclipse.wst.common.project.facet.core.facets">

  <project-facet id="formgen.core">
    <label>FormGen Core</label>
    <description>
      Enables generation of HTML forms based on XML definition files.
    </description>
  </project-facet>

  <project-facet-version facet="formgen.core" version="1.0"/>

  <project-facet id="formgen.ext">
    <label>FormGen Extensions</label>
    <description>
      Enables additional FormGen widgets.
    </description>
  </project-facet>

  <project-facet-version facet="formgen.ext" version="1.0"/>

</extension>
```

Insert the above code into your plugin.xml file and lets see it in action. Launch Eclipse with your FormGen plugin and then open the Dynamic Web Project wizard. Make sure that `<none>` is selected in the Target Runtime field on the first page and go to the second page. You should see a screen that looks like the following. Note that the FormGen facets that you have created are displayed.



Project Facet	Version
☑ Dynamic Web Module	2.4 ...
☐ FormGen Core	1.0
☐ FormGen Extensions	1.0
☑ Java	6.0 ...
☐ WebDoclet (XDoclet)	1.2.3 ...

3. Specifying Constraints

One of the problems with what we have so far is that the FormGen facets appear in other module project wizards such as the EJB Project Wizard. That, of course, makes no sense since FormGen is servlet-based technology and so is only applicable to J2EE web applications. To solve this problem we will use the constraint mechanism to specify the dependencies.

Here is what that part of the extension point schema looks like:

```
<extension point="org.eclipse.wst.common.project.facet.core.facets">
  <project-facet-version>
    <constraint> (optional)
      [expr]
    </constraint>
  </project-facet-version>
</extension>

[expr] =
  <requires facet="{string}" version="{version.expr}" soft="{boolean}"/> or
  <conflicts facet="{string}" version="{version.expr}"/> or
  <conflicts group="{string}"/> or
  <and>
    [expr] (1 or more)
  </and> or
  <or>
    [expr] (1 or more)
  </or>
```

As you can see, the constraint is an expression tree and you have four operators at your disposal. Lets go over them one by one.

requires

The `requires` operator is the most frequently used of all the operators. It is used to specify a dependency on another facet. If the `version` attribute is not specified, any version of the referenced facet will satisfy the constraint. If only specific versions will do, the `version` attribute can contain a version expression.

The soft attribute is used to create a special kind of a dependency. Facet selection will not be prevented if the dependency is not met, but if the dependency is met, the facet is guaranteed to be installed after the referenced facet.

conflicts

The conflicts constraint is used to indicate that the declaring facet will not work correctly if installed into the same project as referenced facets. The conflicts constraint comes in two flavors. You can either specify a conflict with a single facet or with a group of facets.

What are groups of facets? Facet groups are a way to designate a conflict with a certain class of facets without having to list all of the facets explicitly. For instance, the WTP module facets all belong to the "modules" group. They also each declare a conflict with the "modules" group. This prevents two module facets from being installed into the same project. By declaring a conflict with a group whose membership can expand as necessary, third parties can add module facets on top of WTP and have the new facets interact correctly with the built-in module facets.

A facet group is created the first time a facet declares group membership. Here is the extension point schema for declaring group membership:

```
<extension point="org.eclipse.wst.common.project.facet.core.facets">
  <project-facet-version>
    <group-member id="{string}"/> (0 or more)
  </project-facet-version>
</extension>
```

and & or

The and & or constraints are used to perform logical conjunction and disjunction over their operands. Although it is legal for these operators to have only one operand, typically they will have two or more.

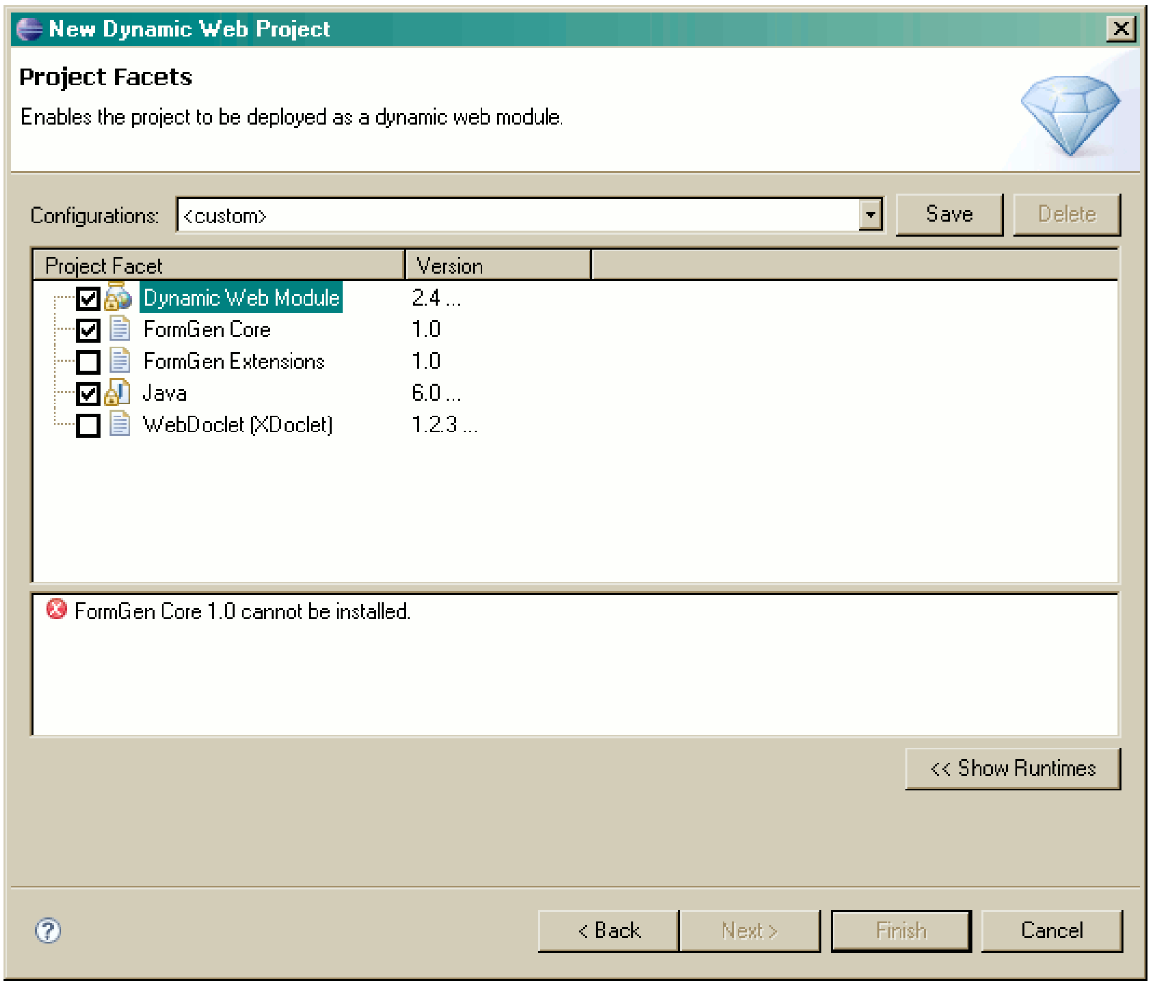
We can now specify the constraints for the FormGen facets. The facet id that marks a project as a J2EE web module is jst.web. We will setup a dependency on it from the formgen.core facet. The formgen.ext facet can then depend on the formgen.ext facet. That latter constraint will ensure that the FormGen Extensions are not installed without installing FormGen Core.

```
<extension point="org.eclipse.wst.common.project.facet.core.facets">

  <project-facet-version id="formgen.core" version="1.0">
    <constraint>
      <requires facet="jst.web" version="2.2,2.3,2.4"/>
    </constraint>
  </project-facet>

  <project-facet-version id="formgen.ext" version="1.0">
    <constraint>
      <requires facet="formgen.core" version="1.0"/>
    </constraint>
  </project-facet>

</extension>
```

Once the above code is added, the FormGen facets should only appear in the Dynamic Web Project wizard.

4. Implementing Actions

Let's now try selecting the FormGen Core facet on the facets selection page of the Dynamic Web Project wizard. If you do that, you should see the following error message appear.



Configurations: <custom>

Project Facet	Version
☑ Dynamic Web Module	2.4 ...
☑ FormGen Core	1.0
☐ FormGen Extensions	1.0
☑ Java	6.0 ...
☐ WebDoclet (XDoclet)	1.2.3 ...

FormGen Core 1.0 cannot be installed.

This error message is displayed because the install action has not been implemented for this facet. What's an action? An action is an operation that a user can perform on a facet. There are three action types `INSTALL`, `UNINSTALL`, and `VERSION_CHANGE`. We will now implement the install actions for the FormGen facets.

Here is what that part of the extension point schema looks like:

```
<extension point="org.eclipse.wst.common.project.facet.core.facets">
  <action id="{string}" facet="{string}" version="{version.expr}" type="INSTALL|UNI
    <delegate class="{class:org.eclipse.wst.common.project.facet.core.IDelegate}"/>
    <property name="{string}" value="{string}"/> (0 or more)
  </action>
</extension>
```

- The version attribute can contain a single version or a version expression. It can also be omitted if the action applies to all versions of the facet.
- The `id` attribute is optional. If not specified, the framework will automatically generate one using the following pattern:

  ```
  [facet-id]#[version-expression]#[action-type](#[prop-name]=[prop-value])*
  ```

  As you can see, it is better to provide an explicit id rather than letting the framework generate it. Later in the tutorial we will cover extension points that make references to action ids.
- The `<action>` element can also be embeded inside the `<project-facet-version>` element. In that case, the `facet` and `version` attributes should be omitted. Note that if the same delegate implementation applies to multiple facet versions, it is better to provide a single action declaration externally. This allows the framework to perform certain kinds of optimizations
- For the `VERSION_CHANGE` action, it is possible to restrict the applicability of the action definition with regards to the starting version. To do that, simply specify "from.versions" property in the action definition. The value is a version expression. If this property is not specified, the framework will assume that the delegate is capable of converting from any starting version.

```
package org.eclipse.wst.common.project.facet.core;

import org.eclipse.core.resources.IProject;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IProgressMonitor;

/**
 * This interface is implemented in order to provide logic associated with
 * a particular event in project facet's life cycle, such as install or
 * uninstall.
 */

public interface IDelegate
{
    /**
     * The method that's called to execute the delegate.
     *
     * @param project the workspace project
     * @param fv the project facet version that this delegate is handling; this
     *   is useful when sharing the delegate among several versions of the same
     *   project facet or even different project facets
     * @param config the configuration object, or null if defaults
     *   should be used
     * @param monitor the progress monitor
     * @throws CoreException if the delegate fails for any reason
     */

    void execute( IProject project,
                  IProjectFacetVersion fv,
                  Object config,
                  IProgressMonitor monitor )

        throws CoreException;
}
```

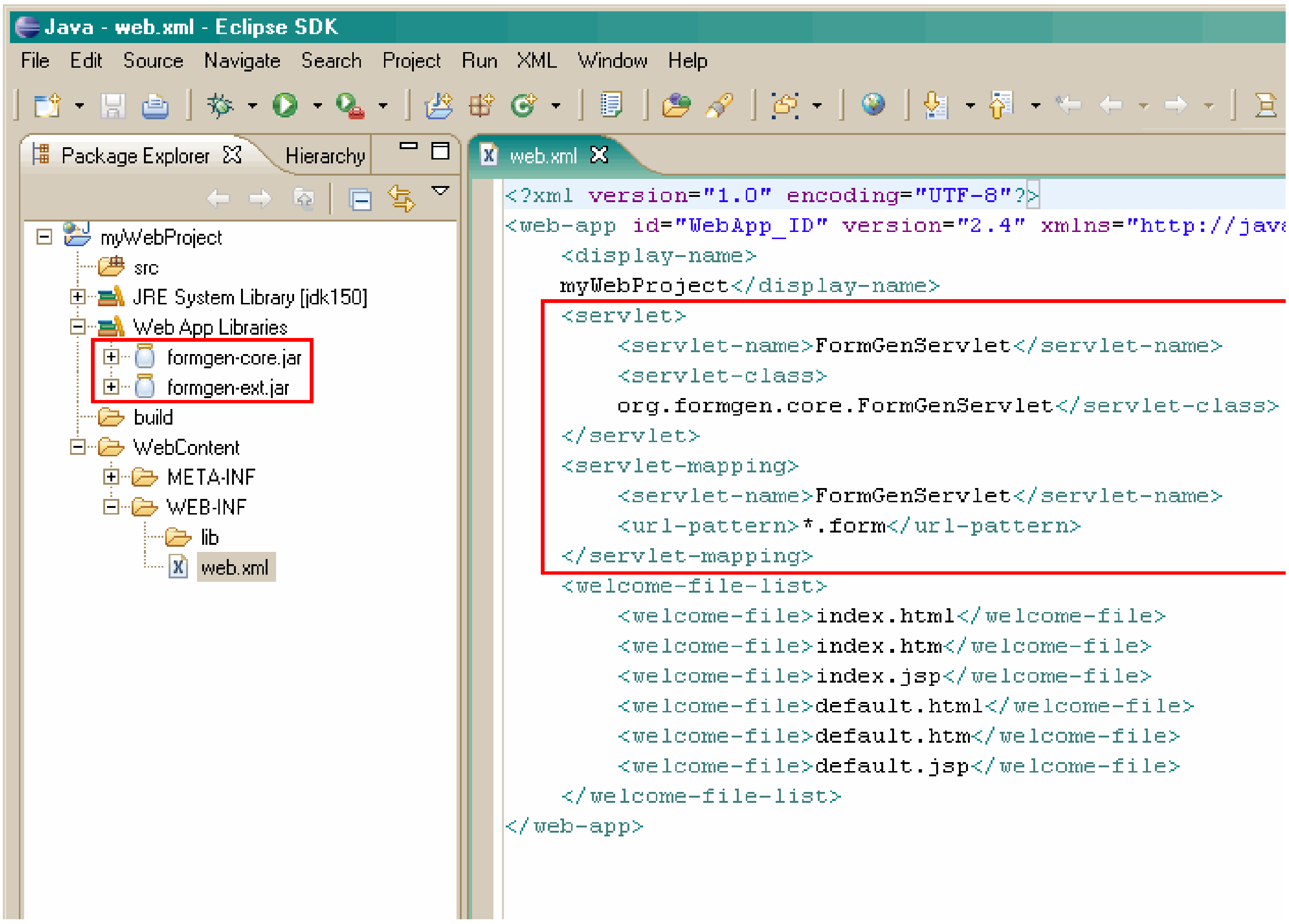
Let's now dive in and implement the install delegates for the FormGen facets. The `formgen.core` facet should (a) copy `formgen-core.jar` into the project's `WEB-INF/lib` directory, and (b) register the FormGen servlet in `web.xml`. The `formgen.ext` facet should copy the `formgen-ext.jar` into the project's `WEB-INF/lib` directory.

```
<extension point="org.eclipse.wst.common.project.facet.core.facets">

  <project-facet-version facet="formgen.core" version="1.0">
    <action type="INSTALL">
      <delegate class="com.formgen.eclipse.FormGenCoreFacetInstallDelegate"/>
    </action>
  </project-facet-version>

  <project-facet-version facet="formgen.ext" version="1.0">
    <action type="INSTALL">
      <delegate class="com.formgen.eclipse.FormGenExtFacetInstallDelegate"/>
    </action>
  </project-facet-version>

</extension>
```

```
package com.formgen.eclipse;

import org.eclipse.core.resources.IFolder;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.core.runtime.Path;
import org.eclipse.wst.common.project.facet.core.IDelegate;
import org.eclipse.wst.common.project.facet.core.IProjectFacetVersion;

public final class FormGenCoreFacetInstallDelegate implements IDelegate
{
    public void execute( final IProject pj,
                         final IProjectFacetVersion fv,
                         final Object config,
                         final IProgressMonitor monitor )

        throws CoreException

    {
        monitor.beginTask( "", 2 );
        try
        {
            final IFolder webInfLib = Utils.getWebInfLibDir( pj );

            Utils.copyFromPlugin( new Path( "libs/formgen-core.jar" ),
                                  webInfLib.getFile( "formgen-core.jar" ) );

            monitor.worked( 1 );

            Utils.registerFormGenServlet( pj );

            monitor.worked( 1 );
        }
        finally
        {
            monitor.done();
        }
    }
}
```

```
package com.formgen.eclipse;

import org.eclipse.core.resources.IFolder;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.core.runtime.Path;
import org.eclipse.wst.common.project.facet.core.IDelegate;
import org.eclipse.wst.common.project.facet.core.IProjectFacetVersion;

public final class FormGenExtFacetInstallDelegate implements IDelegate
{
    public void execute( final IProject pj,
                         final IProjectFacetVersion fv,
                         final Object config,
                         final IProgressMonitor monitor )

        throws CoreException

    {
        monitor.beginTask( "", 1 );

        try
        {
            final IFolder webInfLib = Utils.getWebInfLibDir( pj );

            Utils.copyFromPlugin( new Path( "libs/formgen-ext.jar" ),
                                  webInfLib.getFile( "formgen-ext.jar" ) );

            monitor.worked( 1 );
        }
        finally
        {
            monitor.done();
        }

    }
}
```

Once the install actions have been implemented, you should be able to select the FormGen facets on the Facets Selection Page of the Dynamic Web Project Wizard without getting any error messages. You should also be able to complete the project creation and see the following highlighted artifacts in the new project. These artifacts have been created by the FormGen facet install delegates.



```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java
    <display-name>
    myWebProject</display-name>
    <servlet>
        <servlet-name>FormGenServlet</servlet-name>
        <servlet-class>
        org.formgen.core.FormGenServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>FormGenServlet</servlet-name>
        <url-pattern>*.form</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

5. Creating Categories

Project facets can be grouped into categories in order to provide the "one click" exprience for novice users and retain the fine-grained control for advanced users. You are told that most of FormGen users always add both of the jars to their web apps. These users would benefit from having the FormGen facets grouped into a category and so we will do just that.

Here is what that part of the extension point schema looks like:

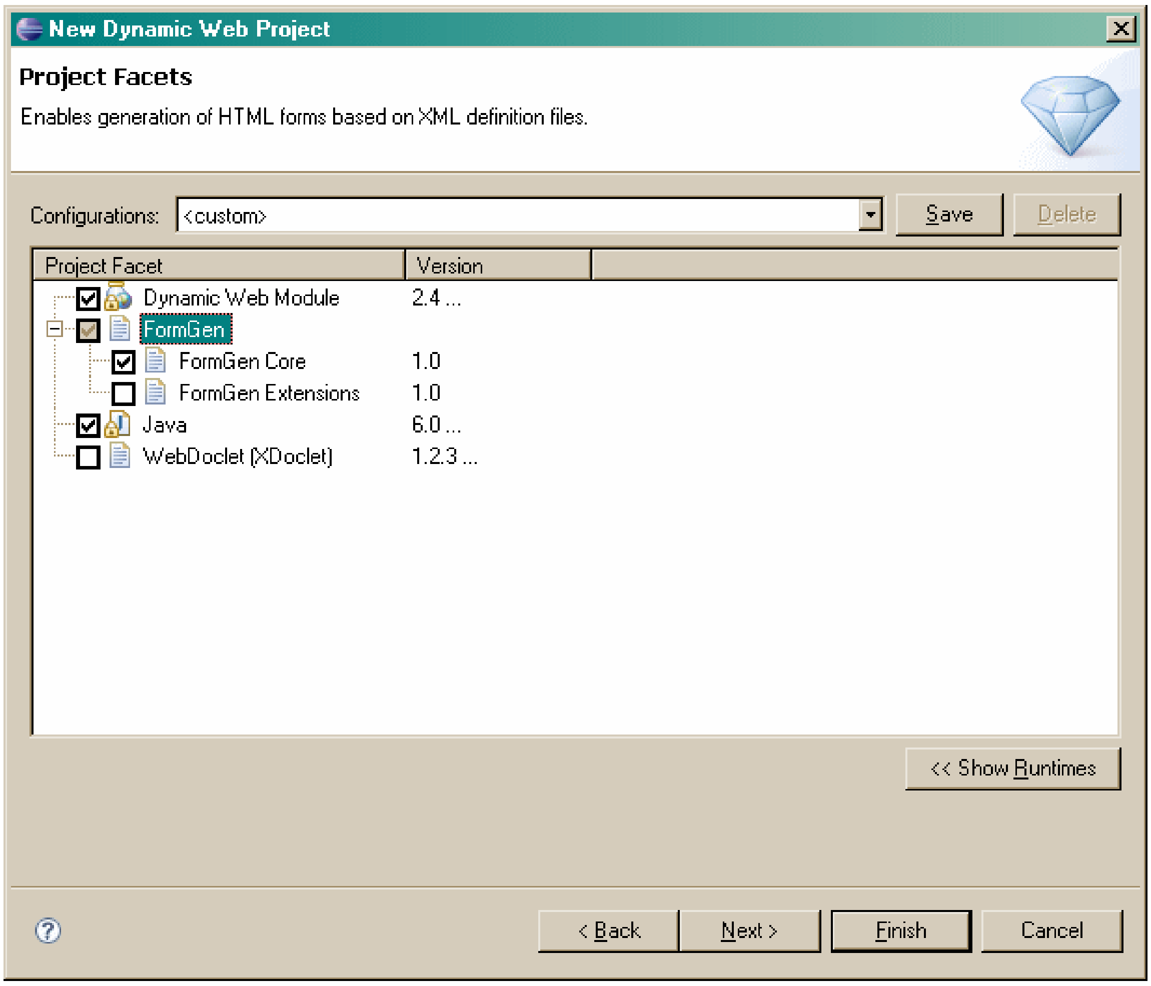
```
<extension point="org.eclipse.wst.common.project.facet.core.facets">
  <category id="{string}">
    <label>{string}</label>
    <description>{string}</description> (optional)
  </category>
  <project-facet>
    <category>{string}</category> (optional)
  </project-facet>
</extension>
```

We can now create a category around the FormGen facets.

```
<extension point="org.eclipse.wst.common.project.facet.core.facets">

  <category id="formgen.category">
    <label>FormGen</label>
    <description>Enables generation of HTML forms based on XML definition files.</de
  </category>

  <project-facet id="formgen.core">
    <category>formgen.category</category>
  </project-facet>

  <project-facet id="formgen.ext">
    <category>formgen.category</category>
  </project-facet>
```

```
</extension>
```

Once the above change has been put in place, the facets selection page should look like this:



6. Decorating

Custom icons can be provided for facets and categories. If an icon is not provided, a default icon is used. The icons are helpful as a way to better differentiate facets and to make them stand out.

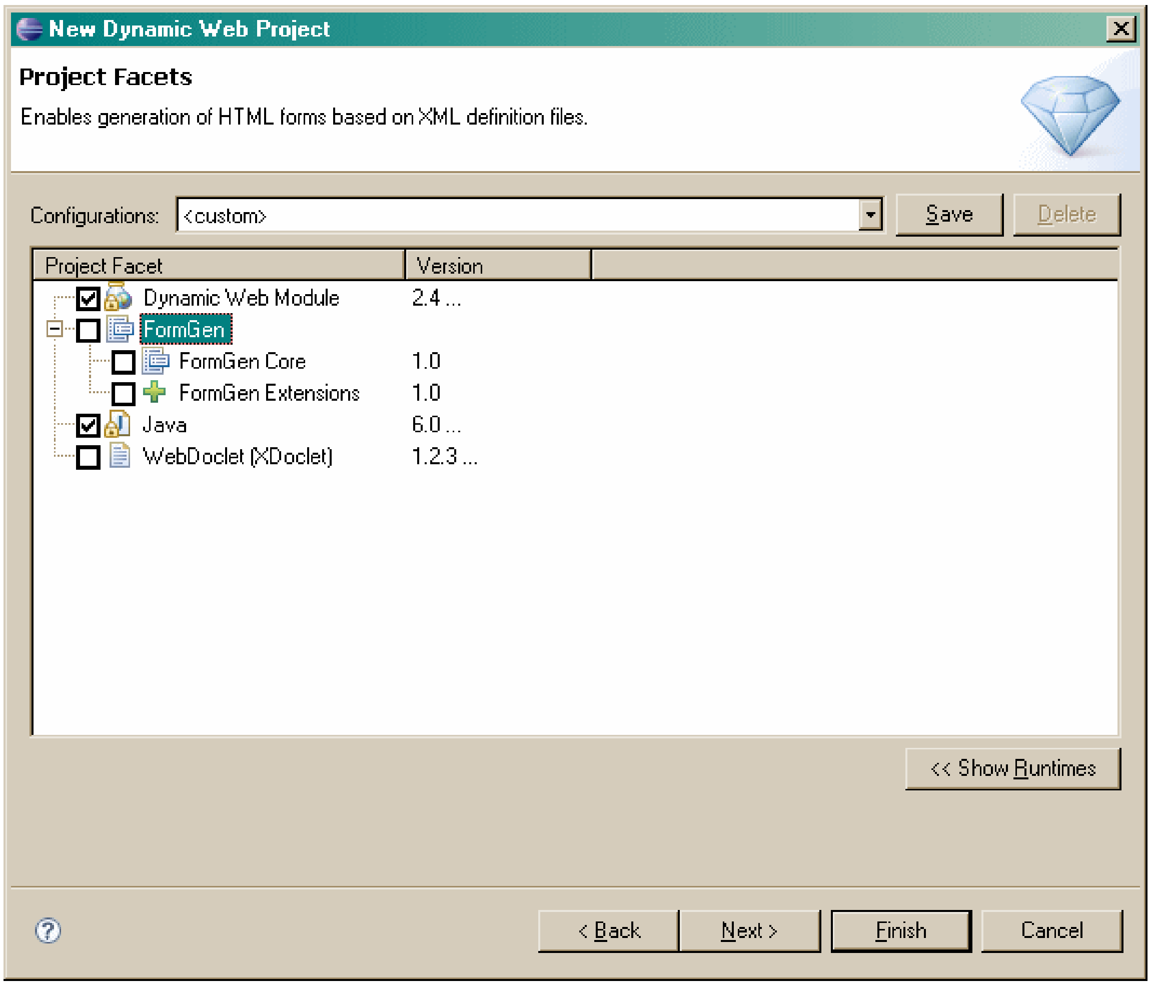
Here is what that extension point looks like:

```
<extension point="org.eclipse.wst.common.project.facet.ui.images">
  <image facet="{string}" path="{string}"/> (0 or more)
  <image category="{string}" path="{string}"/> (0 or more)
</extension>
```

Your starter project came with three icons in the `icons` folder. We will now associate them with the FormGen facets and the category.

```
<extension point="org.eclipse.wst.common.project.facet.ui.images">
  <image facet="formgen.core" path="icons/formgen-core.gif"/>
  <image facet="formgen.ext" path="icons/formgen-ext.gif"/>
  <image category="formgen.category" path="icons/formgen-cat.gif"/>
</extension>
```

Once the above snippet has been added to your plugin.xml file, the facets selection page should look like this:



Enables generation of HTML forms based on XML definition files.

Configurations: <custom>

Project Facet	Version
✓ Dynamic Web Module	2.4 ...
☐ FormGen	
☐ FormGen Core	1.0
☐ FormGen Extensions	1.0
✓ Java	6.0 ...
☐ WebDoclet (XDoclet)	1.2.3 ...

7. Adding Wizard Pages

It is often desirable to gather user input prior to installing a facet. The framework allows a sequence of wizard pages to be associated with facet actions. The supplied wizard pages are shown after the facets selection page. Based on user feedback, you known that FormGen users often customize the URL pattern of the FormGen servlet so you would like to give them the ability to do that in the wizard when the FormGen facets are being installed.

Here is what the relevant parts of the extension points look like:

```
<extension point="org.eclipse.wst.common.project.facet.core.facets">
  <action>
    <config-factory class="class:org.eclipse.wst.common.project.facet.core.IActionC
  </action>
</extension>

<extension point="org.eclipse.wst.common.project.facet.ui.wizardPages">
  <wizard-pages action="{string}"> (0 or more)
    <page class="{class:org.eclipse.wst.common.project.facet.ui.IFacetWizardPage}"/:
  </wizard-pages>
</extension>
```

One thing to note here is that in order to enable communication between the facet action delegate and the wizard pages, we go back to the action declaration and provide an action config factory. The object created by the factory is populated by the wizard pages and is read by the action delegate. No restrictions are placed on the shape of the config object. You may choose to implement a custom class or you can use a something generic like `java.util.HashMap`.

Another thing to note is that the `wizardPages` extension point refers to the actions by their ids, so it becomes more important to explicitly specify the id rather than letting the framework automatically generate one.

Here are the interfaces that are used in the above extension point schema:

```
package org.eclipse.wst.common.project.facet.core;
```

```
import org.eclipse.core.runtime.CoreException;

/**
 * This interface is implemented in order to provide a method for creating
 * a config object that will be used for parameterizing the facet action
 * delegate.
 */

public interface IActionConfigFactory
{
    /**
     * Creates a new facet action configuration object. The new configuration
     * object should ideally be populated with reasonable defaults.
     *
     * @return a new facet action configuration object
     * @throws CoreException if failed while creating the configuration object
     */

    Object create() throws CoreException;
}
```

```
package org.eclipse.wst.common.project.facet.ui;

import org.eclipse.jface.wizard.IWizardPage;

/**
 * This interface is implemented by the wizard pages associated with project
 * facet actions.
 */

public interface IFacetWizardPage extends IWizardPage
{
    /**
     * The framework will call this method in order to provide the wizard
     * context to the wizard page. The wizard context can be used to find out
     * about other actions being configured by the wizard.
     *
     * @param context the wizard context
     */

    void setWizardContext( IWizardContext context );

    /**
     * The framework will call this method in order to provide the action config
     * object that the wizard page should save user selection into. The
     * populated config object will then be passed to the action delegate.
     *
     * @param config the action config object
     */

    void setConfig( Object config );

    /**
     * This method is called after the user has pressed the Finish
     * button. It allows the wizard page to transfer user selection into the
     * config object. Alternative, instead of using this method, the wizard
     * page could update the model on the fly as the user is making changes.
     */

    void transferStateToConfig();
}
```

We will now implement a wizard page for the `facet.core` facet install action. The wizard page will allow the user to change the default servlet URL pattern for the FormGen servlet.

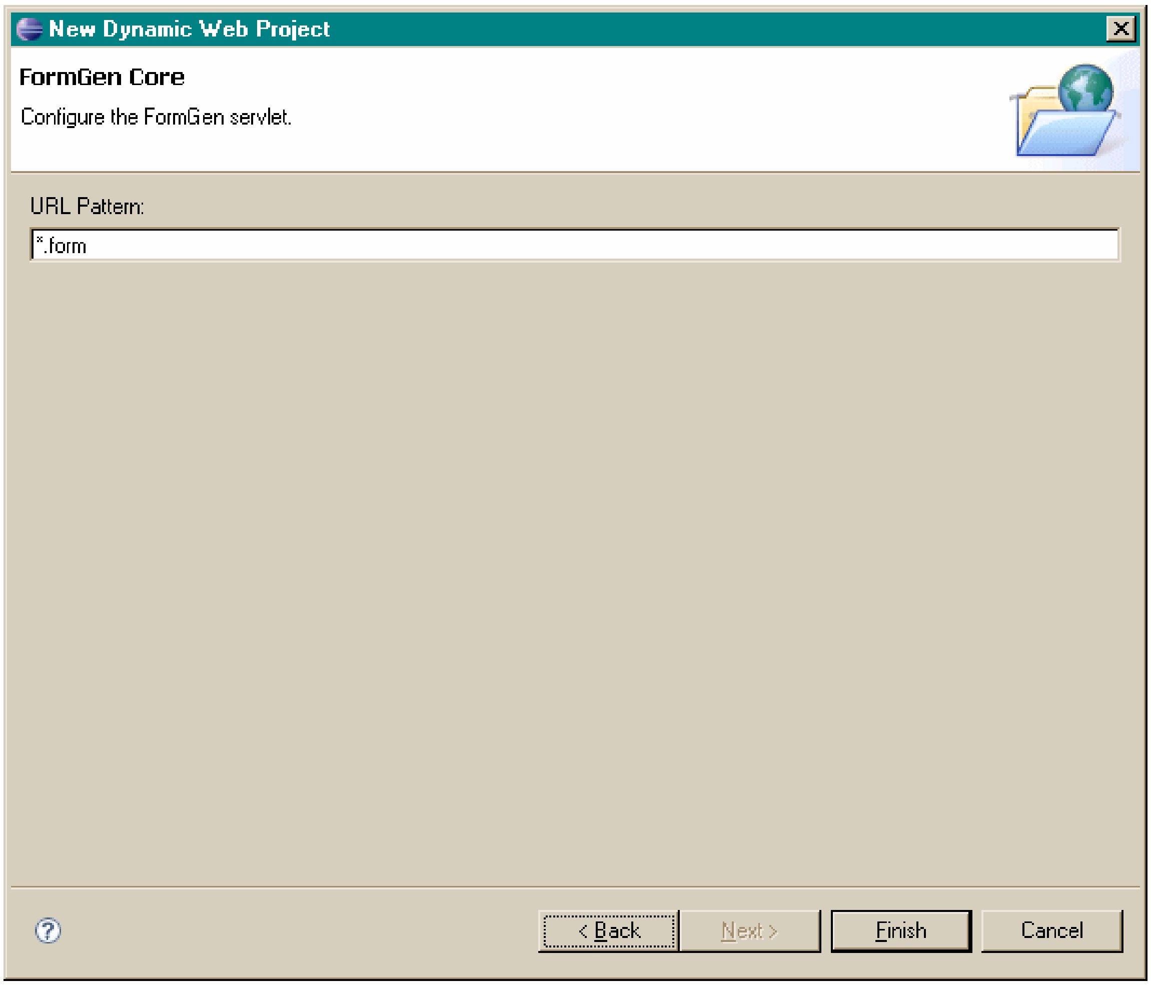
```
<extension point="org.eclipse.wst.common.project.facet.core.facets">
  <project-facet-version facet="formgen.core" version="1.0">
    <action type="INSTALL" id="formgen.core.install">
      <config-factory class="com.formgen.eclipse.FormGenCoreFacetInstallConfig$Fact
    </action>
  </project-facet-version>
</extension>

<extension point="org.eclipse.wst.common.project.facet.ui.wizardPages">
  <wizard-pages action="formgen.core.install">
    <page class="com.formgen.eclipse.FormGenCoreFacetInstallPage"/>
  </wizard-pages>
</extension>
```

```java
package com.formgen.eclipse;

import org.eclipse.wst.common.project.facet.core.IActionConfigFactory;

public final class FormGenCoreFacetInstallConfig
{
    private String urlPattern = "*.form";

    public String getUrlPattern()
    {
        return this.urlPattern;
    }

    public void setUrlPattern( final String urlPattern )
    {
        this.urlPattern = urlPattern;
    }

    public static final class Factory implements IActionConfigFactory
    {
        public Object create()
        {
            return new FormGenCoreFacetInstallConfig();
        }
    }
}
```

```java
package com.formgen.eclipse;

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 org.eclipse.swt.widgets.Text;
import org.eclipse.wst.common.project.facet.ui.AbstractFacetWizardPage;

public final class FormGenCoreFacetInstallPage extends AbstractFacetWizardPage
{
    private FormGenCoreFacetInstallConfig config;
    private Text urlPatternTextField;

    public FormGenCoreFacetInstallPage()
    {
        super( "formgen.core.facet.install.page" );

        setTitle( "FormGen Core" );
        setDescription( "Configure the FormGen servlet." );
    }

    public void createControl( final Composite parent )
    {
        final Composite composite = new Composite( parent, SWT.NONE );
        composite.setLayout( new GridLayout( 1, false ) );

        final Label label = new Label( composite, SWT.NONE );
        label.setLayoutData( gdhfill() );
        label.setText( "URL Pattern:" );

        this.urlPatternTextField = new Text( composite, SWT.BORDER );
        this.urlPatternTextField.setLayoutData( gdhfill() );
        this.urlPatternTextField.setText( this.config.getUrlPattern() );

        setControl( composite );
    }

    public void setConfig( final Object config )
    {
        this.config = (FormGenCoreFacetInstallConfig) config;
    }

    public void transferStateToConfig()
    {
        this.config.setUrlPattern( this.urlPatternTextField.getText() );
    }

    private static GridData gdhfill()
    {
        return new GridData( GridData.FILL_HORIZONTAL );
    }
}
```

```
package com.formgen.eclipse;

import org.eclipse.core.resources.IFolder;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.core.runtime.Path;
import org.eclipse.wst.common.project.facet.core.IDelegate;
import org.eclipse.wst.common.project.facet.core.IProjectFacetVersion;

public final class FormGenCoreFacetInstallDelegate implements IDelegate
{
    public void execute( final IProject pj,
                         final IProjectFacetVersion fv,
                         final Object config,
                         final IProgressMonitor monitor )

        throws CoreException

    {
        monitor.beginTask( "", 2 );

        try
        {
            final FormGenCoreFacetInstallConfig cfg
                = (FormGenCoreFacetInstallConfig) config;

            final IFolder webInfLib = Utils.getWebInfLibDir( pj );

            Utils.copyFromPlugin( new Path( "libs/formgen-core.jar" ),
                                  webInfLib.getFile( "formgen-core.jar" ) );

            monitor.worked( 1 );

            Utils.registerFormGenServlet( pj, cfg.getUrlPattern() );

            monitor.worked( 1 );
        }
        finally
        {
            monitor.done();
        }
    }
}
```

Once the above code changes have been made you should see another page appear in the Dynamic Web Project Wizard after the FormGen facets are selected. The new page will appear after the Web Module page, which is associated with the `jst.web` facet. That's because `formgen.core` facet depends on the `jst.web` facet. If this dependency relationship was not specified the relative order of these pages would be unspecified.



FormGen Core

Configure the FormGen servlet.

URL Pattern: *.form

8. Defining Presets

As the number of available facets grows, it becomes increasingly difficult for the user to figure out which combinations make sense. This is where presets come in. Presets (or Configurations, as they are referred to in the UI) are simply combinations of facets that someone has determined are useful in certain situations. Presets can be created by the user or supplied via an extension point.

Here is the extension point schema for declaring presets:

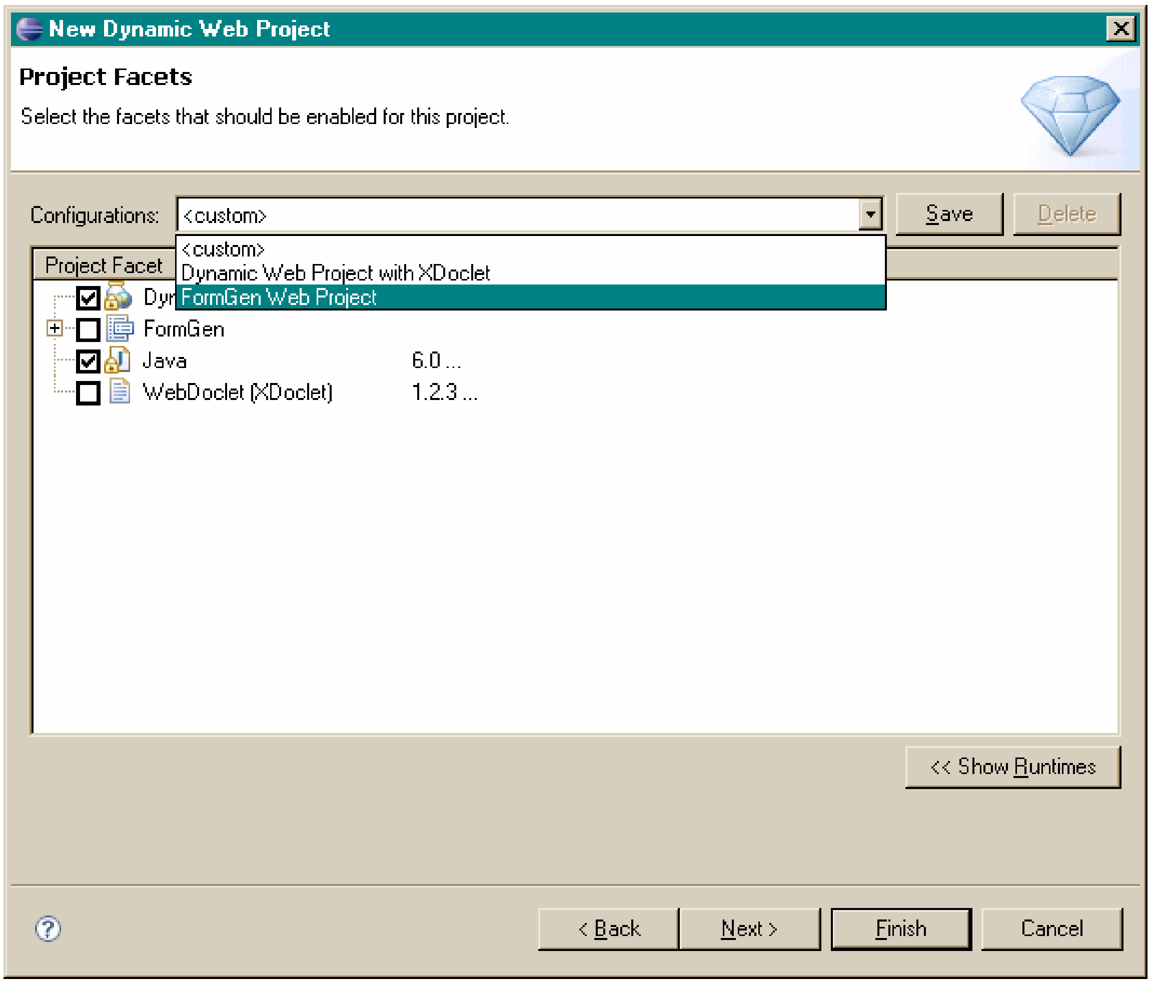
```
<extension point="org.eclipse.wst.common.project.facet.core.facets">
  <preset id="{string}">
    <label>{string}</label>
    <description>{string}</description> (optional)
    <facet id="{string}" version="{string}"/> (1 or more)
  </preset>
</extension>
```

Note that in order for a preset to apply to a given faceted project, the preset needs to include all of the project's "fixed facets". Fixed facets are the facets that are key to the proper operation of that project type and so cannot be removed. You can identify fixed facets by the lock icon.

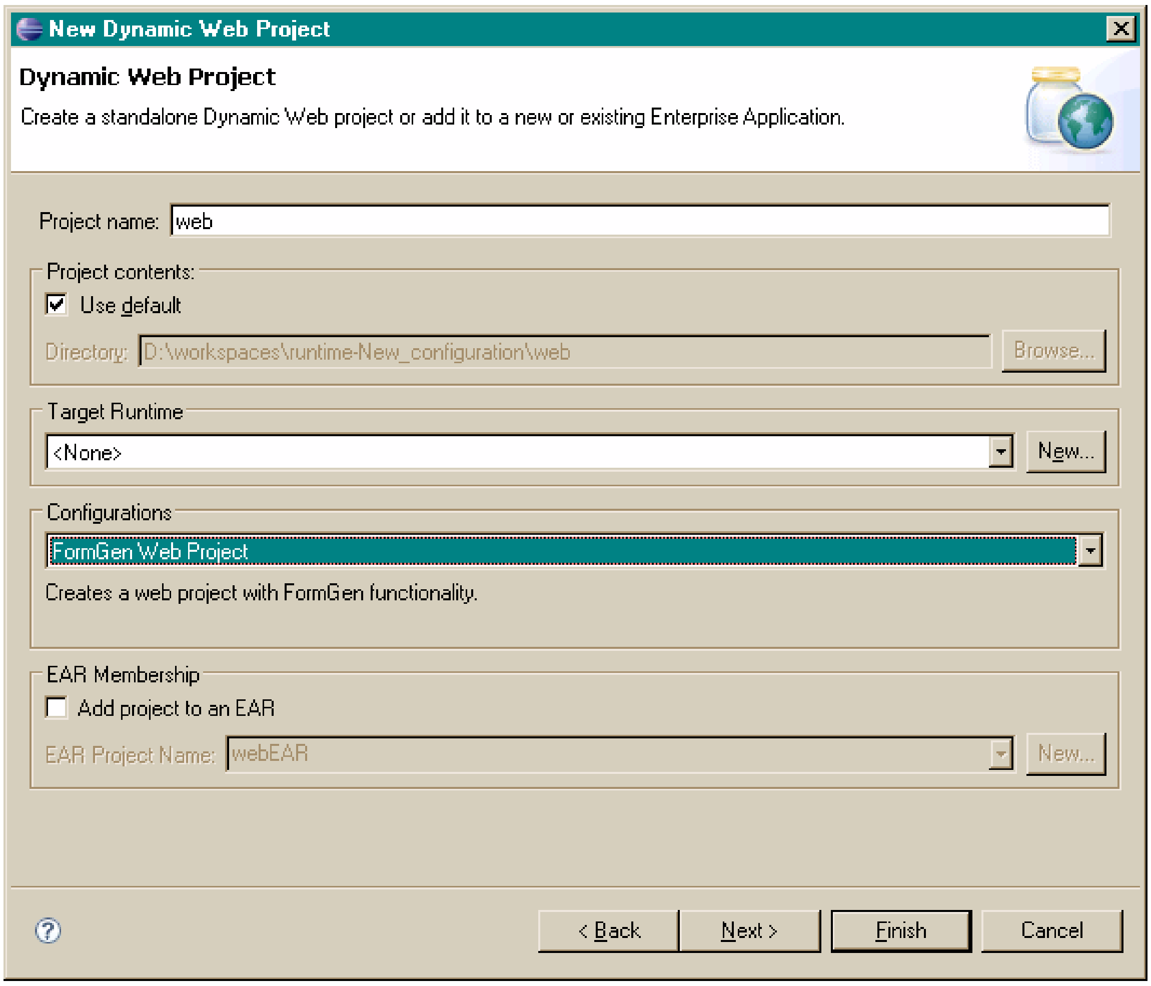
Let's now create a preset that includes formgen facets.

```
<extension point="org.eclipse.wst.common.project.facet.core.facets">
  <preset id="formgen.preset">
    <label>FormGen Web Project</label>
    <description>Creates a web project with FormGen functionality.</description>
    <facet id="jst.java" version="5.0"/>
    <facet id="jst.web" version="2.2"/>
    <facet id="formgen.core" version="1.0"/>
    <facet id="formgen.ext" version="1.0"/>
  </preset>
</extension>
```

Here is how the preset shows up on the facets selection page:



The preset can also be selected on the first page of all WTP project creation wizards. Here is how this looks in the Dynamic Web Project wizard:


New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: web

Project contents:

☑ Use default

Directory: D:\workspaces\runtime-New_configuration\web

Target Runtime: <None>

Configurations: FormGen Web Project

Creates a web project with FormGen functionality.

EAR Membership

☐ Add project to an EAR

EAR Project Name: webEAR

9. Specifying Runtime Support Mappings

One of the most important functions of the faceted project framework is to be able to accurately model whether a certain server runtime supports a given project. We do that by "mapping" project facets to runtime components that support them. If the artifacts created by a facet will run on any server that supports all of the facet's upstream dependencies, then the `any` wildcard can be used.

It's important to note that every facet needs to specify a support mapping of some kind. Facets that don't specify any support mappings are treated as not supported by any runtime, which is not very useful.

Here is the extension point that's used for specifying the support mappings:

```
<extension point="org.eclipse.wst.common.project.facet.core.runtimes">
  <supported> (0 or more)
    <runtime-component any="{boolean}"/> (optional)
    <runtime-component id="{string}"/ version="{version.expr}"/> (0 or more)
    <facet id="{string}"/ version="{version.expr}"/> (1 or more)
  </supported>
</extension>
```

The `<supported>` block can reference any number of runtime components as well as any number of facets. The semantics of that is to declare as supported every combination in the resulting cross-product.

The `version` attributes of the `<runtime-component>` and `<facet>` elements can be omitted to include all versions.

The FormGen facets don't have any special support requirements. They will run on any server that supports the j2ee servlet spec. We will use the `any` wildcard to designate that.

```
<extension point="org.eclipse.wst.common.project.facet.core.runtimes">
  <supported>
    <runtime-component any="true"/>
    <facet id="formgen.core"/>
    <facet id="formgen.ext"/>
  </supported>
</extension>
```

Alternative, if for some reason the FormGen functionality required a specific runtime, such as Tomcat, we would use something like the this instead:

```
<extension point="org.eclipse.wst.common.project.facet.core.runtimes">
  <supported>
    <runtime-component id="org.eclipse.jst.server.tomcat" version="[5.0"/>
    <facet id="formgen.core"/>
    <facet id="formgen.ext"/>
  </supported>
</extension>
```

The above more restrictive specification will prevent FormGen facets from being selected if the project is targetted to any runtime other than Apache Tomcat 5.0 or newer.

10. Summary

In this tutorial we created two fully-functional project facets by specifying constraints, implementing actions, grouping facets into categories, and creating wizard pages to allow users to parameterize facet installation. You should now be well prepared to create your own facets. Additional information not covered by this tutorial can be found in the following appendix sections.

Appendix A: Custom Version Comparators

The faceted project framework needs to be able to compare facet version strings. The framework supplies a default version comparator that can handle version strings encoded using the standard decimal notation (such as 1.2 or 5.66.5533), but if you want to use a different format you will need to supply a custom version comparator.

Here is how you plug in a custom version comparator:

```
<extension point="org.eclipse.wst.common.project.facet.core.facets">
  <project-facet>
    <version-comparator class="{class:java.util.Comparator<String>}"/>
  </project-facet>
</extension>
```

When implementing a version comparator you can either start from scratch or subclass the default version comparator (`org.eclipse.wst.common.project.facet.core.DefaultVersionComparator`). Subclassing the default version comparator makes sense if the version format differs only slightly from the default format, such as using separators other than dots or having non-numeric characters at certain positions. Here are the protected methods exposed by the `DefaultVersionComparator` class that are designed to be overridden:

```
/**
 * Returns the string containing the separator characters that should be
 * used when breaking the version string into segments. The default
 * implementation returns ".". Subclasses can override this method.
 *
 * @return the separator characters
 */

protected String getSeparators();

/**
 * Parses a segment of the version string. The default implementation parses
 * the first segment as an integer (leading zeroes are ignored) and the
 * rest of the segments as decimals (leading zeroes are kept). Subclasses
 * can override this method to provide custom parsing for any number of
 * segments.
 *
 * @param version the full version string
 * @param segment the version segment
 * @param position the position of the segment in the version string
 * @return the parsed representation of the segment as a {@see Comparable}
 * @throws VersionFormatException if encountered an error while parsing
 */
```

```
protected Comparable parse( final String version,
                            final String segment,
                            final int position )

    throws VersionFormatException;
```

Appendix B: Version Expressions

A version expression is a syntax for specifying more than one version. The version expressions are used throughout the framework's extension points, but you will most likely first encounter them while specifying the `requires` constraint.

A version expression is composed of multiple segments separated by commas. The commas function as an OR operation. Each segment is either a single version or a range. Ranges are represented by bracket and dash notation such as [x-y). A bracket means inclusive, while the parenthesis means exclusive. Open ended ranges are also supported.

Here are some examples:

```
1.2
1.2,1.5,3.2
[1.2-3.2]
[3.7-5.0)
[3.7
5.0)
1.2,[3.0-4.5),[7.3
```

Appendix C: Event Handlers

It is possible to register listeners for certain events in the faceted project's life cycle. Here is the list of the available events:

```
PRE_INSTALL
POST_INSTALL
PRE_UNINSTALL
POST_UNINSTALL
PRE_VERSION_CHANGE
POST_VERSION_CHANGE
RUNTIME_CHANGED
```

The way one declares event handlers is very similar to how actions are declared, however there are some key differences:

- Unlike actions, events are not a direct result of something a user does. This means that it is not possible to associate wizard pages or provide a custom configuration object for event handlers.
- Multiple event handlers can be declared for the same event. The relative order that they will be invoked in is not specified.

Here is the extension point schema for registering event handlers:

```
<extension point="org.eclipse.wst.common.project.facet.core.facets">
  <event-handler facet="{string}" version="{version.expr}" type="{event.type}">
    <delegate class="{class:org.eclipse.wst.common.project.facet.core.IDelegate}"/>
  </action>
</extension>
```

As you can see, just like with action definitions, the event handler has to implement the `IDelegate` interface. Also, just like with action definitions, the `<event-handler>` block can be embeded directly inside the `<project-facet-version>` element. In that case, the values for the `facet` and `version` attributes are implied and the attributes should be omitted.

The `PRE_*` and `POST_*` event handlers all get the same config object passed into their delegate's `execute` method as the corresponding action delegate. The `RUNTIME_CHANGED` event handlers get an instance of `IRuntimeChangedEvent`.

```
package org.eclipse.wst.common.project.facet.core;

import org.eclipse.wst.common.project.facet.core.runtime.IRuntime;

/**
 * Describes the runtime changed event to the RUNTIME_CHANGED event handlers.
 */

public interface IRuntimeChangedEvent
{
    /**
     * Returns the runtime previously associated with the project.
     *
     * @return the runtime previously associated with the project, or null
     */

    IRuntime getOldRuntime();

    /**
     * Returns the runtime that's now associated with the project.
     *
     * @return the runtime that's now associated with the project, or null
     */

    IRuntime getNewRuntime();
}
```

Appendix D: Property Tester

A property tester is provided by the Faceted Project Framework that allows the presence of the facet in a project to be tested by any extension point that works with `org.eclipse.core.expressions` package. The most common usage is to enable user interface elements (such as actions and project property pages). The property name is `org.eclipse.wst.common.project.facet.core.projectFacet` and the value is either a facet id or a facet id followed by a colon and a version expression.

Here is an example of using facets property tester to control enablement of a project properties page:

```
<extension point="org.eclipse.ui.propertyPages">
  <page
    adaptable="true"
    objectClass="org.eclipse.core.resources.IProject"
    name="FormGen Properties"
    class="com.formgen.eclipse.FormGenPropertiesPage"
    id="org.eclipse.jst.j2ee.internal.J2EEDependenciesPage">
    <enabledWhen>
      <test
        forcePluginActivation="true"
        property="org.eclipse.wst.common.project.facet.core.projectFacet"
        value="formgen.core"/>
    </enabledWhen>
  </page>
</extension>
```

Appendix E: Wizard Context

Sometimes it desirable to be able to adjust the behavior of facet action wizard pages based on user input in the wizard pages of other facets. The `IWizardContext` interface can be used for this purpose. The wizard page gets a handle on `IWizardContext` interface when it's `setWizardContext` method is called. When writing code that relies on the wizard context, there are a couple of points you should keep in mind.

1. The facet whose value you wish to check may have already been installed in the past. In that case you will not find it's install configuration in the wizard context. You will need to write conditional logic that will consult the wizard context or looks at project state on disk.
2. You should make sure that a reasonable default is provided in your config object for the API-only scenario where your wizard page will not be involved.

Here is what the `IWizardContext` interface looks like:

```
package org.eclipse.wst.common.project.facet.ui;
```

```java
import java.util.Set;

import org.eclipse.core.runtime.CoreException;
import org.eclipse.wst.common.project.facet.core.IProjectFacetVersion;
import org.eclipse.wst.common.project.facet.core.IFacetedProject.Action;
import org.eclipse.wst.common.project.facet.core.IFacetedProject.Action.Type;

/**
 * The interface exposed to the facet action wizard pages that allows them
 * to gather information about the wizard state.
 */

public interface IWizardContext
{
    /**
     * Returns the name of the project that the wizard is operating on. If the
     * wizard is in the project creation mode, the project will not yet exist
     * in the workspace.
     *
     * @return the name of the project that the wizard is operating on
     */

    String getProjectName();

    /**
     * Returns the set of facets currently selected in the wizard. If the wizard
     * is in the add/remove facets mode (vs. project creation), this method will
     * return the set of facets currently installed in a project after being
     * modified by the current set of actions.
     *
     * @return the set of facets currently selected in the wizard (element type:
     *   {@see IProjectFacetVersion})
     */

    Set getSelectedProjectFacets();

    /**
     * Determines whether the specified facet is currently selected in the
     * wizard. See {@see getSelectedProjectFacets()} for more information.
     *
     * @param fv the project facet version object
     * @return true if an only if the provided project facet is
     *   currently selected in the wizard
     */

    boolean isProjectFacetSelected( IProjectFacetVersion fv );

    /**
     * Returns the set of actions currently specified by the user.
     *
     * @return the set of actions currently specified by the user
     */

    Set getActions();

    /**
     * Finds the action of specified type that applies to the specified facet,
     * if such action exists. If the wizard is in the add/remove facets mode
     * (vs. project creation), you cannot depend on finding the install action
     * for a required facet as that facet may have already been installed.
     *
     * @param type the action type
     * @param fv the project facet version object
     * @return the action object or null
     */

    Action getAction( Action.Type type,
                      IProjectFacetVersion fv );
}
```