Developing Custom Controls

BEA Workshop for WebLogic Platform allows you to create custom controls tailored to your project or application. This section explains how to create these controls.

For a complete overview of controls in Workshop for WebLogic, including how to create them, see Getting Started with Beehive Controls.

Topics Included in This Section

Related Topics

Using System Controls

Creating Custom Controls

This topic describes how to use a custom custom control. It explains how to:

- Create a custom control
- Use a custom control in your application

Custom control files can be located:

- In your web project.
- In a utility project. To access such controls in a web application, both the web project and the utility project must be linked to the same EAR project.

To Create a Custom Control

The following instruction assume you are in the Workshop perspective (**Window > Open Perspective > Workshop**).

1. You cannot create a control in the default package. So the first step is to create a package for the control. For example:

   <ProjectRoot>/src/controls/myControl/

2. Right-click the package and select **New** > **Custom Control**.

3. In the **Control name** field, enter the class name for the control.

   The Java interface and implementation classes will be based on the name entered here. For example, if you enter Hello, two classes will be created:

   Hello.java (=the interface class)

   and

   HelloImpl.java (=the implementation class)

4. Click **Finish**.

Default control interface and implementation classes are produced. Assuming that your control is named Hello, the following class files are produced:

Hello.java Interface Class File

```
package controls.myControl;
```

```
import org.apache.beehive.controls.api.bean.ControlInterface;

@ControlInterface
public interface Hello {

}
```

HelloImpl.java Implementation Class File

```
package controls.myControl;

import org.apache.beehive.controls.api.bean.ControlImplementation;
import java.io.Serializable;

@ControlImplementation
public class HelloImpl implements Hello, Serializable {
        private static final long serialVersionUID = 1L;

}
```

Continue the composition of the custom control by adding methods to these class files.

To Use a Custom Control in an Application

If you have an existing custom control in your project or in a utility project in the current workspace, you can add a reference to that control to a control client by right-clicking anywhere within the client's Java source file and selecting **Insert > Control**.

A list of available controls appears. The heading **Existing Project Controls** lists the controls in the same project as the client. The heading **Existing Application Controls** lists the controls in the utility projects in the same workspace.

When you add a control reference to a client, Workshop for WebLogic Platform modifies your client's source code to include an annotation and variable declaration for the control. The annotation ensures that the control is recognized by Workshop for WebLogic Platform, and the variable declaration gives you a way to work with the control from your client code. For example, if you add a new custom control named `Hello`, the following code will be added to your client:

```
    import org.apache.beehive.controls.api.bean.Control;
    import controls.myControl.Hello;

        @Control
        private Hello hello;
```

Once you have a reference to a control, your client can call methods on that control. For more detail on calling a control method, see Invoking a Control Method.

Related Topics

Invoking a Control Method

Source Files for Custom Controls

Source Files for Custom Controls

Custom controls consist of two Java source files: an **interface** class file and an **implementation** class file.

The interface class contains the control's publicly accessible methods. Clients of the control call the methods in the implementation class.

The implementation class contains the control's behind the scenes implementation code.

There is also a third class associated with each custom control: the **generated JavaBean class**. This is a build artifact created from the interface and implementation source files. The generated JavaBean class provides supplemental programmatic access to the control, especially the ability to override default annotation values in the control. For more information about this class see Overriding Control Annotation Values Through the Control JavaBean

Custom Control Interface Classes

A custom control interface class must be decorated with the `@ControlInterface` annotation.

```
package controls.hello;

import org.apache.beehive.controls.api.bean.ControlInterface;

@ControlInterface
public interface Hello {

    ...

}
```

The `@ControlInterface` annotation informs the compiler to treat this class as a part of the Beehive Control framework.

The interface class also lists the control's publicly available methods. The following example shows a control with one publicly available method.

```
package controls.hello;

import org.apache.beehive.controls.api.bean.ControlInterface;

@ControlInterface
public interface Hello {

        public String hello();

}
```

Custom Control Implementation Classes

A custom control implementation class contains the control's logic - the code that defines what the control does. In this file you define what each of the control's methods do.

The minimum requirements for a custom control implementation class are listed below.

1. The class must be decorated with the @ControlImplementation annotation.

```
import org.apache.beehive.controls.api.bean.ControlImplementation;

@ControlImplementation
public class HelloImpl
```

2. The class must implement the corresponding custom control interface file.

```
import org.apache.beehive.controls.api.bean.ControlImplementation;

@ControlImplementation
public class HelloImpl implements Hello
```

3. The classes must either:

   (a) implement java.io.Serializable

```
import java.io.Serializable;

@ControlImplementation
public class HelloImpl implements Hello, Serializable
```

   (b) or set @ControlImplementation(isTransient=true)

```
@ControlImplementation(isTransient=true)
public class HelloImpl implements Hello {

}
```

Related Topics

Controls: Getting Started

Testing Controls

Beehive controls can be tested either inside of an application container or outside in a standalone Java environment. Testing in a standalone Java environment is especially useful when running unit tests.

Beehive controls can be integrated into the JUnit test framework using the ControlTestCase base class. This base class provides a control container and provides help in instantiating a control declaratively via the
@Control annotation.

Note that not all controls can be tested within the test container because some controls have requirements beyond what ControlTestCase provides. For example, a control that uses JNDI lookups will not be testable with ControlTestCase. Likewise controls that take a dependency on a J2EE container (like the Service Control) may not be testable out of that J2EE container.

For details on testing controls with ControlTestCase see Control Tutorial: Testing Controls with JUnit.

Related Topics

Testing Controls with JUnit

Exporting Controls into JARs

Workshop for Weblogic Platform lets you package your control classes as JAR files that can be ported and reused in other Java projects. This is the simplest way to distribute controls.

To package a Beehive control as a JAR file, select **File > Export > Beehive Control JAR File**.

Only control files in utility projects are available for JAR file packaging; controls in other project types are not available for export.

All Java class files in the utility project are included in the JAR file, including control interface, control implementation classes, and all other Java classes. Note that by default, **only** class files are included in the JAR file. To include the Java source files, place a checkmark next to **Include Java source files**.

To use a control in another web application:

1. Copy the JAR file to the WEB-INF/lib folder.
2. Add a reference to that control to a control client by right-clicking anywhere within the client's Java source file and selecting **Insert > Control**.
3. A list of available controls appears. The heading **Existing Project Controls** lists the available controls, including controls in JAR files.

Alternately, you can:

1. Copy the JAR file to the APP-INF/lib folder of the associated EAR project.
2. Add a reference to that control to a control client by right-clicking anywhere within the client's Java source file and selecting **Insert > Control**.
3. A list of available controls appears. The heading **Existing Application Controls** lists the available controls, including controls in JAR files.

As long as the JAR is inserted into the user's classpath as described above, the control will be discovered automatically by Workshop for WebLogic and property set/event handler features will be provided.

Related Topics

Apache Beehive Documentation

Building Controls