Beehive Controls Tutorial

Table of contents

1. Introduction

The Controls Tutorial is a good way to become familiar with the concepts of Beehive Controls. This describes how to write, build, package, and test a simple Hello Control. The following concepts will be convered here:

- Creating a standalone Beehive Control project
- Creating a Control interface and implementation
- Compiling a Control
- Packaging a Control into a JAR file
- Testing a Control using JUnit

For a technical description of Controls, see the Controls programming guide.

2. Step 1: Setup Control Development

2.1. Install Beehive

Complete all of the required and optional steps to install and setup Beehive here.

2.2. Install JUnit

In this tutorial, JUnit is the test framework used. In order to use JUnit, download a JUnit release here and unzip it into a local directory.

2.3. Create a Control Project

Copy the directory `<BeehiveRoot>/samples/controls-blank` to a location of your choice. For example, you might copy to the following location: `/beehive-projects/controls-blank`.

Rename `controls-blank` to `controls_tutorial`.

Edit the file `controls_tutorial/build.properties` so that the property `beehive.home` points to the top level folder of your Beehive distribution.

Add the property `junit.home` to the file and ensure that it points to your JUnit installation.

For example, if Beehive distribution is located in `/apache/apache-beehive-1.0` and JUnit is located in `/test-tools/junit`, then your `build.properties` file would appear as follows:

```
beehive.home=/apache/apache-beehive-1.0

junit.home=/test-tools/junit
```

Note:

Properties files should use the '/' character to separate drive, directory, and file names.

3. Step 2: Compile Control Interface and Implementation Source Files

3.1. Introduction

A Beehive Control consists of two source files -- an interface file and an implementation file. The interface file is the public API of a control and provides all of the methods that can be invoked by a Control user (or client). The implementation file contains the implementation code for the methods listed in the interface file.

3.2. A Control's Source Files

Open the file `/controls_tutorial/src/pkg/HelloImpl.java`.

The implementation file appears as follows. (There is no need to edit the file at this point in the tutorial.)

```
package pkg;

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

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

    public String hello() {
        return "Hello!";
    }
}
```

Open the file `controls_tutorial/src/pkg/Hello.java`. The interface file should appear appears as follows.

```
package pkg;

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

@ControlInterface
public interface Hello {
    String hello();
}
```

3.3. Set a Name for the Hello Control JAR

This step sets the name of the JAR file created when compiling the `Hello` and `HelloImpl` interface and implementation.

Edit the file `controls_tutorial/build.xml` to set the `build.jar` property to the name `helloControl.jar`. The property should appear as:

```
<project name="controls_tutorial" default="usage" basedir=".">
...
    <property name="build.jar" value="helloControl.jar"/>
...
<project>
```

3.4. Build and Package the Control

Now, compile the control and build the Control's JAR file. The following Ant command should be run from the `controls_tutorial` project directory; at the command prompt, enter:

```
ant clean build
```

The Control's classes are generated in `build/classes`, and the Control's JAR file is created in `build/helloControl.jar`.

4. Step 3: Create a JUnit Test for the Control

4.1. Create a JUnit Test Class

Now that the control builds successfully, it's time to write a JUnit test case to ensure that the Control operates as it is intended. To do this, create a `test` directory to contain the JUnit test cases for the control. The directory should be created under `controls_tutorial/test` with the command:

```
mkdir test
```

In this directory, create a Java package to contain the source of a JUnit test with:

```
mkdir test/testpackage
```

Now, create a JUnit test case called `HelloControlTest.java`. This class wil declare an instance of the Hello control and unit test its API. This class should appear as follows:

```
package testpackage;

import org.apache.beehive.controls.api.bean.Control;
import org.apache.beehive.controls.test.junit.ControlTestCase;

import pkg.Hello;
```

```
public class HelloControlTest
    extends ControlTestCase {

    @Control
    private Hello _helloControl;

    public void testHello()
        throws Exception {
        String message = _helloControl.hello();

        assertEquals("Failed to receive message \"Hello!\"", "Hello!",
message);
    }
}
```

4.2. Edit build.xml to Compile and Run the JUnit Test

Now that the test has been written, the Ant build.xml file still needs to change to support building and running JUnit tests. To do this, add the following `test` target to the build.xml file:

```
<target name="test">
    <property name="test.src" location="test"/>
    <property name="test.classes" location="${build.dir}/test-classes"/>
    <property name="test.beansrc" location="${build.dir}/test-beansrc"/>

    <mkdir dir="${test.classes}"/>
    <mkdir dir="${test.beansrc}"/>

    <path id="test.classpath">
        <path refid="build.classpath"/>
        <pathelement location="${build.dir}/${build.jar}"/>
        <pathelement location="${junit.home}/junit.jar"/>
    </path>

    <build-controls srcdir="${test.src}"
                    destdir="${test.classes}"
                    tempdir="${test.beansrc}"
                    classpathref="test.classpath"/>

    <path id="test-run.classpath">
        <path refid="test.classpath"/>
        <pathelement location="${test.classes}"/>
        <pathelement
location="${beehive.home}/lib/common/commons-discovery-0.2.jar"/>
        <pathelement
location="${beehive.home}/lib/common/commons-logging-1.0.4.jar"/>
    </path>

    <java classname="junit.textui.TestRunner"
          classpathref="test-run.classpath">
        <arg line="testpackage.HelloControlTest"/>
```

```
    </java>
</target>
```

This target first builds the test source files. Then, it runs the JUnit tests, reporting errors if tests fail.

Note:

There are lots of ways to run JUnit tests in Ant including the use of the `<junit>` Ant tasks. This tutorial has taken a simple approach to running JUnit; this example test should run in other JUnit execution environments as well. For information on configuring Ant's JUnit tasks, see here.

4.3. Run the Test

In your command shell, ensure that you are in the directory `controls_tutorial/`

Run the following Ant command to test the control.

```
ant clean build test
```

5. Step 4: Edit the Control and Re-test

5.1. Edit the Control Source Files

Edit the file `HelloImpl.java` so it appears as follows. Code to add appears in bold

```
package pkg;

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

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

    public String hello() {
        return "Hello!";
    }

public String hello(String name) {
        return "Hello, " + name + "!";
    }
}
```

Edit the file `Hello.java` so it appears as follows. Code to add appears in bold

```
package pkg;

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

@ControlInterface
```

```
public interface Hello {
    String hello();

    String hello(String name);
}
```

5.2. Add a Test Case for the New Control Method

Edit the file `Tests.java` so it appears as follows. Code to add appears in bold.

```
package testpackage;

import org.apache.beehive.controls.api.bean.Control;
import org.apache.beehive.controls.test.junit.ControlTestCase;

import pkg.Hello;

public class Tests
    extends ControlTestCase {

    @Control
    private Hello _helloControl;

    public void testHello()
        throws Exception {

        String message = _helloControl.hello();
        assertEquals("Failed to receive message \"Hello!\"", "Hello!",
message);
    }

    public void testParam()
        throws Exception {
        String message = _helloControl.hello("World");

        assertEquals("Failed to receive message \"Hello, World!\"", "Hello,
World!", message);
    }
}
```

5.3. Run the Test

Test the control according to the previous instructions here.