Controls ControlPackaging

Controls Packaging Model

The Beehive Control framework includes a JAR-based packaging model that allows a collection of controls (and supporting control implementations) to be bundled into a JAR file.

- The primary goal is to enable a tool to quickly scan a jar file for a list of available controls (and certain key attributes) and then introspect individual classes within the JAR.
- A secondary goal is to provide an annotation model that enables the creation of JAR manifest attributes and custom BeanInfo associated with a Control JavaBean simply by annotating the original source artifacts.

Relationship to JavaBeans packaging model

The goal is to fully support the JavaBeans packaging and introspection model for Controls. There is no point in re-inventing the wheel and/or doing something different where the existing JavaBeans spec describes useful functionality. To make the model easier to use for developers and tools, the Controls packaging model defines a set of JSR-175 annotation types that enable packaging information to specified declaratively inside Control source files.

Section 11 of the JavaBeans specification actually defines a simple packaging model based upon entries in META-INF/MANIFEST.MF. The spec states that the manifest will contain a section describing each JavaBean of the form:

```
Name: org.apache.beehive.controls.samples.SampleBean.class
JavaBean: true
```

There is also a basic introspection model (based upon the java.beans.Introspector class) that enables a JavaBean class to be queried for its java.BeanInfo, BeanDescriptor, MethodDescriptors, PropertyDescriptors, etc. to provide additional information about the JavaBean. These various Descriptor types all derive from a common base class (FeatureDescriptor) that enables customized design-time attributes to be associated with bean types, properties, methods, and events.

[JavaBeans spec and API docs can be found at: http://java.sun.com/products/javabeans/docs/spec.html]

These two mechanisms (manifest attributes and BeanInfo descriptor elements) will be leveraged to provide an extended packaging model for Beehive Controls.

JAR Manifest Attributes

The standard Control processing/build infrastructure provides support for *automatically* generating the basic set of JAR manifest entries for JavaBeans generated to support Beehive controls. When the annotation processor for Control JSR-175 annotations is run, the processor will generate JAR manifest fragments for each Control JavaBean that has been generated. These fragments will be written to the class build output directory have and the name *<fully-qualified-bean-classname>.class.manifest*.

A special Controls Jar Ant task is provided that can be used to generate a JAR file containing Controls that will merge the generated manifest fragments into the JAR manifest. **The Controls Jar task derives directly from the base-level Ant <jar> task, and support all of its standard attributes and nested elements**. This task can be imported into an Ant build file for a Controls JAR as follows:

```
<taskdef name="control-jar"
         classname="org.apache.beehive.controls.runtime.packaging.ControlJarTask"
         classpath="{beehive-lib-dir}/controls.jar"/>
```

A sample usage of this task (from the Controls checkin tests) is shown below:

```
<control-jar destfile="${build.jars}/drtbeans.jar"  basedir="${build.beans}" />
```

The resulting META-INF/MANIFEST.MF manifest file will contain the contents of any input manifest file (passing as an attribute or inline to the <control-jar> task merged with the auto-generated manifest descriptor attributes.

Declaring custom JAR manifest attributes using @ManifestAttribute and @ManifestAttributes

The Controls packaging model also provides a simple JSR-175 annotation syntax that enables a developer or tool to add custom manifest attributes for a generated Control JavaBean by annotating the Control public or extension interface, as follows:

```
package org.apache.beehive.samples.packaging;

import org.apache.beehive.controls.api.packaging.ControlInterface;
import org.apache.beehive.controls.api.packaging.ManifestAttribute;

@ControlInterface
@ManifestAttribute(name="MyAttribute", value="MyValue")
public interface MyControl { ... }
```

This would result in the following JAR manifest section being created:

```
Name: org.apache.beehive.controls.samples.packaging.MyControlBean.class
JavaBean: true
MyAttribute: MyValue
```

A collection of attribute name/value pairs for a Control public or extension interface can be specified using the ManifestAttributes annotation type:

```
@ControlInterface
@ManifestAttributes({
    @ManifestAttribute(name="MyAttribute", value="MyValue"),
    @ManifestAttribute(name="MyOtherAttribute", value="MyOtherValue"),
    ...
    })
public interface MyControl { ... }
```

BeanInfo Attributes

JavaBeans provides a basic introspection model that enables tools to query a JavaBean for information about its properties, methods, events, and other attributes. Tools can access this information using the services of the java.beans.Introspector class that will return a BeanInfo instance describing the bean type and provides access to the various Descriptor types *(PropertyDescriptor, MethodDescriptor, EventSetDescriptor, ...)* elements that provide useful information about the JavaBean.

Each of these Descriptor classes derives from a common class (java.beans.FeatureDescriptor) that provides useful design-time information for tools, such as the display name of the property, method, or event, a short description, whether the *feature* should be hidden or displayed, is for basic or expert use, etc. Additionally, the FeatureDescriptor can also contain a set of attributes (name/value pairs) that be used to provide customized or tool-specific attributes that related to the JavaBean *feature*.

The Controls packaging model augments the standard JavaBeans model by providing a set of JSR-175 annotations that allow JavaBean Descriptor values to be specified directly on declarations in a Control public or extension interface, and providing the necessary infrastructure to generate customized BeanInf support classes that will expose this information to tools. Once generated, **the customized BeanInfo is accessible using the standard JavaBeans introspection model**.

The subsequent sections describe these annotations. There is a base FeatureInfo type that can be used to set the generic FeatureDescriptor information for a Control *feature* and then Descriptor-specific annotation types to set the specific values for bean, property, method, and event descriptors.

Setting FeatureDescriptor values using @FeatureInfo

The FeatureInfo annotation type can be used to set any or all of the various FeatureDescriptor elements for JavaBean types, properties, methods, event sets, or events. A supporting type (@FeatureAttribute) also enables custom attributes to be set for the feature.

Here is the JSR-175 annotation type declaration for FeatureInfo:

```
package org.apache.beehive.controls.api.packaging;

import java.lang.annotation.*;

@Target({ElementType.TYPE, ElementType.METHOD})
public @interface FeatureInfo
{
    public String displayName() default "";          // default: use reflection name
    public String name() default "";                 // default: use reflection name
    public String shortDescription() default "";
    public boolean isExpert() default false;
    public boolean isHidden() default false;
    public boolean isPreferred() default false;
    public FeatureAttribute [] attributes() default {};
}
```

Default values are provided for the members of this type, meaning only non-default values need to be set on use. Here is an example of using this type to set FeatureDescriptor information for a Control JavaBean type:

```
@ControlInterface
@FeatureInfo(
    displayName="My Control",
    shortDescription="A simple packaging example control",
    isExpert=true,
    attributes={{
        @FeatureAttribute(name="MyCustomAttribute", value="TheValue"),
        ...
        })
    )
public interface MyControl { ... }
```

The above annotation would result in the following result from using the JavaBeans introspection API:

```
String desc = java.beans.Introspector.getBeanInfo(MyControlBean.class).getBeanDescriptor().
getShortDescription();
```

The value of *desc* after this call would be "A simple packaging example control".

The @FeatureInfo annotation can be used in the following locations:

- **An interface annotated with @ControlInterface or @ControlExtension** to set feature attributes for the BeanDescriptor

associated with the generated JavaBean type.

- **A method declaration within an @Control or @ControlExtension interface** to set feature attributes for the

MethodDescriptor associated with a Control operation

- **A method declaration with an @PropertySet interface** to set feature attribute for the PropertyDescriptor associated with the Control property.
- **An interface annotation with the @EventSet annotation** to set feature attributes for the EventSetDescriptor associated

with a Control event set.

- **A method declaration within an @EventSet interface** to set feature attributes for the MethodDescriptor associated with a Control event

Setting BeanDescriptor attributes using @BeanInfo

TBD

Setting PropertyDescriptor attributes using @PropertyInfo=

The @PropertyInfo annotation can be placed on a property method declaration to specify the eventing behavior of the annotated property.

Here is the declaration of PropertyInfo:

```
@Target({ElementType.METHOD})   // appears on PropertySet method declaration (i.e. properties)
public @interface PropertyInfo
{
    public boolean bound() default false;         // Sends PropertyChange events
    public boolean constrained() default false;   // Sends VetoableChange events
}
```

An example usage of PropertyInfo is:

```
@ControlInterface
public interface MyControl
{
     @PropertySet
     public @interface SampleProps
     {
         @FeatureInfo(shortDesc="This property send no events")
         public int basicProp()

         @PropertyInfo(bound=true)
         @FeatureInfo(shortDesc="This property will send PropertyChange events")
         public int boundProp();

         @PropertyInfo(constrained=true)
         @FeatureInfo(shortDesc="This property will send VetoableChange events")
         public int constrainedProp();
     }
}
```

The full set of classes for BeanInfo annotations can be found in the org.apache.beehive.controls.api.packaging package.