BeanForm

NOTE: This information applies to an older version of Tapestry. For current information for Tapestry 5 see the BeanEditForm Guide.

WARNING: This page is out of date; please see http://beanform.sourceforge.net/ for the latest version.

An often-requested feature is an easier, more (Trails/Rails/Grails)-like way of editing domain objects that gets rid of a lot of the boilerplate typing required to create an edit form. Some solutions are breathtaking in their comprehensiveness but suffer from lack of documentation and over-architecting (imho). The BeanForm is a simpler solution to this common problem:

```
<span jwcid="@BeanForm" bean="ognl:pojo" save="listener:save" delete="listener:delete"/>
```

The above code gets you a form that will call save() when submitted for save, delete() when submitted for delete, displays TextFields for the bean's string and numerical properties, Checkboxes for its boolean properties and DatePickers for its date properties, and automatically disables fields for read-only properties. Properties are discovered via bean introspection. You can use the BeanForm with a single line, as shown above, or you can include it inside one of your existing Forms, in which case the internal Form component will not be generated. It's your choice.

You can also specify exactly which properties you want included in the BeanForm, as well as cancel and refresh listeners and any of the other parameters supported by the Form component. When you explicitly specify the editable properties, those properties are shown on the page in the order you specify:

```
<span jwcid="@BeanForm" bean="ognl:pojo" properties="ognl:'name,email,comment'" save="listener:save" cancel="
listener:cancel" refresh="listener:refresh" />
```

The generated HTML is a form containing a two-column table (unless the BeanForm is already inside a Form, in which case another Form is not generated). The left column contains the field labels and the right column contains the data entry fields. The bottom row spans both columns and contains the save button (if the *save* parameter was specified), the cancel button (if the *cancel* parameter was specified), the refresh button (if the *refresh* parameter was specified) and the delete button (if the *delete* parameter was specified). The table can be styled using CSS: the table's CSS class is *beanFormTable*, the left column's CSS class is *beanFormLeftColumn*, the right column's CSS class is *beanFormRightColumn*, and the bottom column that contains the buttons has the CSS class *beanFormButtonColumn*.

If you need to customize any of the input fields, you can do so by adding a Block component to your page (with the id *"[propertyName]BeanFieldBlock"*) that contains any IFormComponent (with the id *"[propertyName]BeanField"*). Doing this will also allow you to edit properties that are considered non-editable by default (ie, not a string / boolean / number / date).

For example, if you wanted to edit the *comment* property in a TextArea instead of the default textfield, you could add the following to your page:

```
<div jwcid="commentBeanFieldBlock@Block">
        <input jwcid="commentBeanField@TextArea" value="ognl:pojo.comment" displayName="message:comment"/>
</div>
```

In order to use validation, specify the *delegate* parameter (unless you have already specified it in an external Form component that contains this component) and add standard Tapestry validator lists to the *properties* parameter, in between squiggly brackets (*{ }*). Fox example, if you want to make the *name* and *email* properties required, and want to check that *email* is actually an email address, you could do the following:

```
<span jwcid="@BeanForm" bean="ognl:pojo" properties="ognl:'name{required},email{required,email},comment'" save="
listener:save" delegate="bean:validationDelegate" clientValidationEnabled="ognl:true"/>
```

Internationalization is supported out of the box: field labels are messages keyed on the property name. The save button (if displayed) is labeled with the message corresponding to the key *save*. The cancel button (if displayed) is labeled with the message corresponding to the key *cancel*. The refresh button (if displayed) is labeled with the message corresponding to the key *refresh*. The delete button (if displayed) is labeled with the message corresponding to the key *delete*.

This component should be simple enough that when you need that one extra feature, you can just enhance this code rather than going back to boilerplateland. Enjoy!

Three files:

- BeanForm.java
- BeanForm.jwc
- BeanForm.html

Gotchas:

- Change the package name in BeanForm.java!
- Use your own logging framework in BeanForm.java!
- If you are using a page property (*pojo* in all the examples above) for the BeanForm's *bean* parameter, you will probably need to make this page property persistent. Otherwise, your page will likely break during the rewind cycle.

BeanForm.html

```
<!-- If this component is already inside a form, don't render another form. -->
<span jwcid="isInsideAForm">
        <span jwcid="renderBlock1"/>
</span>
<span jwcid="isNotInsideAForm">
        <form jwcid="beanForm">
                <span jwcid="renderBlock2"/>
        </form>
</span>

<!-- The main content block. -->
<span jwcid="block">
        <table class="beanFormTable">
                <span jwcid="beanPropertyRow">
                        <span jwcid="hasCustomField">
                                <td class="beanFormLeftColumn"><span jwcid="customFieldLabel"/></td>
                                <td class="beanFormRightColumn"><span jwcid="customFieldBlock"/></td>
                        </span>
                        <span jwcid="noCustomField">
                                <span jwcid="isString">
                                        <td class="beanFormLeftColumn"><span jwcid="beanPropertyLabel_Text"/><
/td>
                                        <td class="beanFormRightColumn"><span jwcid="beanPropertyField_Text"/><
/td>
                                </span>
                                <span jwcid="isBoolean">
                                        <td class="beanFormLeftColumn"><span jwcid="beanPropertyLabel_Checkbox"
/></td>
                                        <td class="beanFormRightColumn"><span jwcid="beanPropertyField_Checkbox"
/></td>
                                </span>
                                <span jwcid="isNumber">
                                        <td class="beanFormLeftColumn"><span jwcid="beanPropertyLabel_Number"/><
/td>
                                        <td class="beanFormRightColumn"><span jwcid="beanPropertyField_Number"
/></td>
                                </span>
                                <span jwcid="isDate">
                                        <td class="beanFormLeftColumn"><span jwcid="beanPropertyLabel_Date"/><
/td>
                                        <td class="beanFormRightColumn"><span jwcid="beanPropertyField_Date"/><
/td>
                                </span>
                        </span>
                </span>
                <td colspan="2" class="beanFormButtonColumn">
                        <span jwcid="hasSave"><span jwcid="beanFormSave"/></span>
                        <span jwcid="hasCancel"><span jwcid="beanFormCancel" onclick="this.form.events.cancel()"
/></span>
                        <span jwcid="hasRefresh"><span jwcid="beanFormRefresh" onclick="this.form.events.
refresh()"/></span>
                        <span jwcid="hasDelete"><span jwcid="beanFormDelete"/></span>
                </td>
        </table>
</span>
```

BeanForm.java

```
package com.diphy.web.components;

import java.beans.BeanInfo;
import java.beans.IntrospectionException;
import java.beans.Introspector;
import java.beans.PropertyDescriptor;
import java.beans.SimpleBeanInfo;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Date;
```

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.apache.hivemind.ApplicationRuntimeException;
import org.apache.hivemind.Location;
import org.apache.tapestry.BaseComponent;
import org.apache.tapestry.IComponent;
import org.apache.tapestry.IRequestCycle;
import org.apache.tapestry.TapestryUtils;
import org.apache.tapestry.annotations.InjectObject;
import org.apache.tapestry.binding.BindingFactory;
import org.apache.tapestry.components.Block;
import org.apache.tapestry.form.IFormComponent;
import org.apache.tapestry.form.validator.ValidatorsBinding;

import com.diphy.util.Logger;

/**
 * <p>A form that provides edit capabilities for a Java Bean. Only properties
 * that are strings, booleans, numbers or dates are considered editable. Fields
 * for read-only bean properties are automatically disabled. Field labels are
 * messages keyed on the property name. The save button (if displayed) is
 * labeled with the message corresponding to the key <tt>save</tt>. The cancel
 * button (if displayed) is labeled with the message corresponding to the key
 * <tt>cancel</tt>. The refresh button (if displayed) is labeled with the
 * message corresponding to the key <tt>refresh</tt>. The delete button (if
 * displayed) is labeled with the message corresponding to the key
 * <tt>delete</tt>.</p>
 *
 * <p>The component parameters are as follows:</p>
 *
 * <ul>
 *   <li><tt>bean</tt>: Required, specifies the Java Bean to modify.</li>
 *   <li><tt>properties</tt>: Not required, specifies the bean properties to be
 *      edited; if omitted, all eligible bean properties (strings, booleans,
 *      numbers and dates) are considered editable; if specified, properties
 *      are displayed in the specified order.</li>
 *   <li><tt>save</tt>: Not required, specifies the listener to invoke on
 *      save.</li>
 *   <li><tt>delete</tt>: Not required, specifies the listener to invoke on
 *      delete.</li>
 *   <li><tt>success</tt>: Not required, specifies the listener to invoke on
 *      form submission.</li>
 *   <li><tt>cancel</tt>: Not required, specifies the listener to invoke on
 *      form cancellation.</li>
 *   <li><tt>refresh</tt>: Not required, specifies the listener to invoke on
 *      form refresh.</li>
 *   <li><tt>delegate</tt>: Not required, specifies the validation delegate to
 *      use on the form.</li>
 *   <li>And all the rest of the <tt>Form</tt> parameters, including
 *      <tt>method</tt>, <tt>listener</tt>, <tt>stateful</tt>, <tt>direct</tt>,
 *      <tt>clientValidationEnabled</tt>, <tt>focus</tt>, <tt>scheme</tt> and
 *      <tt>port</tt>. See the <tt>Form</tt> component's documentation for
 *      more details.
 * </ul>
 *
 * <p>The generated HTML is a form containing a two-column table, unless this
 * component is inside an external Form component, in which case no extraneous
 * form is generated. The table's left column contains the field labels and the
 * right column contains the data entry fields. The bottom row spans both
 * columns and contains the save button (if the <tt>save</tt> parameter was
 * specified), the cancel button (if the <tt>cancel</tt> parameter was
 * specified), the refresh button (if the <tt>refresh</tt> parameter was
 * specified) and the delete button (if the <tt>delete</tt> parameter was
 * specified).</p>
 *
 * <p>The table can be styled using CSS: the table's CSS class is
 * <tt>beanFormTable</tt>, the left column's CSS class is
```

```
 * <tt>beanFormLeftColumn</tt>, the right column's CSS class is
 * <tt>beanFormRightColumn</tt>, and the bottom column that contains the
 * buttons has the CSS class <tt>beanFormButtonColumn</tt>.</p>
 *
 * <p>If you need to customize any of the input fields, you can do so by adding
 * a <tt>Block</tt> component to your page (with the id
 * <tt>[propertyName]BeanFieldBlock</tt>) that contains any
 * <tt>IFormComponent</tt> (with the id <tt>[propertyName]BeanField</tt>).
 * Doing this will allow you to edit properties that are considered
 * non-editable by default (ie, not a string/boolean/number/date).</p>
 *
 * <p>In order to use validation, specify the <tt>delegate</tt> parameter
 * (unless you have already specified it in an external <tt>Form</tt> component
 * that contains this component) and add standard validation lists to the
 * <tt>properties</tt> parameter, in between squiggly brackets (<tt>{}</tt>).
 * See below for an example.</p>
 *
 * <p>Usage examples:</p>
 * <ul>
 *   <li>Minimal: <tt>&lt;span jwcid="@BeanForm" bean="ognl:pojo"
 *      save="listener:save"/&gt;</tt></li>
 *   <li>More: <tt>&lt;span jwcid="@BeanForm" bean="ognl:pojo"
 *      properties="ognl:'name,email,comment'" save="listener:save"
 *      cancel="listener:cancel" delete="listener:delete"
 *      refresh="listener:refresh" /&gt;</tt></li>
 *   <li>To edit the <tt>comment</tt> property in a text area you would add the
 *      following to your page:<br>
 *      <tt>&lt;div jwcid="commentBeanFieldBlock@Block"&gt;</tt><br>
 *      <tt>&lt;input jwcid="commentBeanField@TextArea"
 *         value="ognl:pojo.comment"
 *         displayName="message:comment"/&gt;</tt><br>
 *      <tt>&lt;/div&gt;</tt><br>
 *      </tt></li>
 *   <li>To make <tt>name</tt> and <tt>email</tt> required, and check that
 *      <tt>email</tt> is actually an email:
 *      <tt>&lt;span jwcid="@BeanForm" bean="ognl:pojo"
 *      properties="ognl:'name{required},email{required,email},comment'"
 *      save="listener:save" clientValidationEnabled="ognl:true"
 *      delegate="bean:validationDelegate"/&gt;</tt></li>
 * </ul>
 *
 * @author Daniel Gredler
 * @version 1.2
 */
public abstract class BeanForm extends BaseComponent {

        private final static Logger LOG = Logger.getLogger( BeanForm.class );
        private final static Pattern PROPERTIES_PATTERN = Pattern.compile( "\\s*(\\w+)\\s*(?:\\{\\s*(.+?)\\s*\\}
\\s*)?,?" );
        private final static String CUSTOM_FIELD_BLOCK_SUFFIX = "BeanFieldBlock";
        private final static String CUSTOM_FIELD_SUFFIX = "BeanField";

        @InjectObject( "infrastructure:requestCycle" )
        public abstract IRequestCycle getCycle();

        @InjectObject( "service:tapestry.form.validator.ValidatorsBindingFactory" )
        public abstract BindingFactory getValidatorsBindingFactory();

        public abstract Object getBean();
        public abstract void setBean( Object bean );

        public abstract String getProperties();
        public abstract void setProperties( String properties );

        public List<BeanProperty> getBeanProperties() {
                List<BeanProperty> properties = new ArrayList<BeanProperty>();
                Object bean = this.getBean();
                BeanInfo info = this.getBeanInfo();
                PropertyDescriptor[] descriptors = info.getPropertyDescriptors();
                for( PropertyDescriptor descriptor : descriptors ) {
                        BeanProperty property = new BeanProperty( descriptor );
```

```
                        String validators = this.getValidators( property );
                        property.setValidators( validators );
                        if( this.isIncluded( property ) ) {
                                if( property.isEditableType() || this.hasCustomField( property ) ) {
                                        properties.add( property );
                                }
                                else if( this.hasExplicitProperties() ) {
                                        LOG.warn( "Explicitly included bean property '" + property.getName() +
"' is not a " +
                                                        "string/boolean/number/date, and a custom field was not
specified for it." );
                                }
                        }
                }
                properties = this.reorderProperties( properties );
                if( LOG.isDebugEnabled() ) {
                        LOG.debug( "Found " + properties.size() + " editable properties for bean " + bean + ":
" + properties );
                }
                return properties;
        }

        public boolean isInsideAForm() {
                try {
                        TapestryUtils.getForm( this.getCycle(), this );
                        return true;
                }
                catch( ApplicationRuntimeException e ) {
                        return false;
                }
        }

        public List getValidatorList( BeanProperty property, IComponent component ) {
                String name = property.getName();
                String desc = "dynamic validators binding for bean property " + name;
                String expression = property.getValidators();
                Location location = component.getLocation();
                BindingFactory factory = this.getValidatorsBindingFactory();
                ValidatorsBinding binding = (ValidatorsBinding) factory.createBinding( component, desc,
expression, location );
                List validators = (List) binding.getObject();
                if( LOG.isDebugEnabled() ) {
                        LOG.debug( "Bean property '" + name + "' has " + validators.size() + " validators: " +
validators );
                }
                return validators;
        }

        public boolean hasCustomField( BeanProperty property ) {
                boolean customBlock = this.getCustomFieldBlock( property ) != null;
                boolean customField = this.getCustomField( property ) != null;
                return customBlock && customField;
        }

        public Block getCustomFieldBlock( BeanProperty property ) {
                return (Block) this.getComponent( property, CUSTOM_FIELD_BLOCK_SUFFIX, Block.class );
        }

        public IFormComponent getCustomField( BeanProperty property ) {
                return (IFormComponent) this.getComponent( property, CUSTOM_FIELD_SUFFIX, IFormComponent.class
);
        }

        @SuppressWarnings( "unchecked" )
        private IComponent getComponent( BeanProperty property, String suffix, Class clazz ) {
                IComponent component = null;
                String name = property.getName() + suffix;
                Map components = this.getPage().getComponents();
                if( components.containsKey( name ) ) {
                        IComponent candidate = (IComponent) components.get( name );
                        if( clazz.isInstance( candidate ) ) {
```

```
                                component = candidate;
                        }
                        else {
                                LOG.error( "Component '" + candidate.getId() + "' should be of type " + clazz.
getName() + "!" );
                        }
                }
                return component;
        }

        private BeanInfo getBeanInfo() {
                Object bean = this.getBean();
                BeanInfo info;
                if( bean != null ) {
                        try {
                                info = Introspector.getBeanInfo( bean.getClass() );
                        }
                        catch( IntrospectionException e ) {
                                LOG.error( e );
                                info = new SimpleBeanInfo();
                        }
                }
                else {
                        LOG.warn( "BeanForm's bean is null!" );
                        info = new SimpleBeanInfo();
                }
                return info;
        }

        private List<String> explicitPropertyNames;
        private Map<String, String> explicitPropertyValidators;

        private String getValidators( BeanProperty property ) {
                this.initExplicitProperties();
                if( this.explicitPropertyValidators == null ) return null;
                return this.explicitPropertyValidators.get( property.getName() );
        }

        private boolean isIncluded( BeanProperty property ) {
                this.initExplicitProperties();
                if( this.explicitPropertyNames == null ) return true;
                return this.explicitPropertyNames.contains( property.getName() );
        }

        private List<BeanProperty> reorderProperties( List<BeanProperty> properties ) {
                this.initExplicitProperties();
                if( this.explicitPropertyNames == null ) return properties;
                List<BeanProperty> orderedProperties = new ArrayList<BeanProperty>( properties.size() );
                for( String name : this.explicitPropertyNames ) {
                        for( BeanProperty property : properties ) {
                                if( property.getName().equals( name ) ) {
                                        orderedProperties.add( property );
                                }
                        }
                }
                return orderedProperties;
        }

        private boolean hasExplicitProperties() {
                this.initExplicitProperties();
                return ( this.explicitPropertyNames != null );
        }

        private synchronized void initExplicitProperties() {
                if( this.explicitPropertyNames != null ) return;
                String s = this.getProperties();
                if( s == null ) return;
                this.explicitPropertyNames = new ArrayList<String>();
                this.explicitPropertyValidators = new HashMap<String, String>();
                Matcher m = PROPERTIES_PATTERN.matcher( s );
                while( m.find() ) {
```

```
                String name = m.group( 1 );
                String validators = m.group( 2 );
                this.explicitPropertyNames.add( name );
                this.explicitPropertyValidators.put( name, validators );
        }
}

/**
 * Instances of this class get serialized into the HTML, so keep it trim (ie,
 * as few serialized instance variables as possible).
 */
public static class BeanProperty implements Serializable {

        private static final long serialVersionUID = -9064407627959060313L;

        private static final String STRING = String.class.getName();
        private static final String BOOLEAN = Boolean.class.getName();
        private static final String BOOL = boolean.class.getName();
        private static final String INTEGER = Integer.class.getName();
        private static final String INT = int.class.getName();
        private static final String LONG = Long.class.getName();
        private static final String LNG = long.class.getName();
        private static final String FLOAT = Float.class.getName();
        private static final String FLT = float.class.getName();
        private static final String DOUBLE = Double.class.getName();
        private static final String DBL = double.class.getName();
        private static final String DATE = Date.class.getName();

        private String name;
        private String validators;
        private boolean readOnly;
        private String typeName;

        public BeanProperty( PropertyDescriptor descriptor ) {
                this.name = descriptor.getName();
                this.validators = null;
                this.readOnly = descriptor.getWriteMethod() == null;
                this.typeName = descriptor.getPropertyType().getName();
        }

        public String getName() {
                return this.name;
        }

        public String getValidators() {
                return this.validators;
        }

        public void setValidators( String validators ) {
                this.validators = validators;
        }

        public boolean isReadOnly() {
                return this.readOnly;
        }

        public String getTypeName() {
                return this.typeName;
        }

        public boolean isString() {
                return STRING.equals( this.typeName );
        }

        public boolean isBoolean() {
                return BOOLEAN.equals( this.typeName ) || BOOL.equals( this.typeName );
        }

        public boolean isNumber() {
                return INTEGER.equals( this.typeName ) || INT.equals( this.typeName ) ||
                        LONG.equals( this.typeName ) || LNG.equals( this.typeName ) ||
```

```
                                FLOAT.equals( this.typeName ) || FLT.equals( this.typeName ) ||
                                DOUBLE.equals( this.typeName ) || DBL.equals( this.typeName );
                }

                public boolean isDate() {
                        return DATE.equals( this.typeName );
                }

                public boolean isEditableType() {
                        return this.isString() || this.isBoolean() || this.isNumber() || this.isDate();
                }

                @Override
                public String toString() {
                        return this.name + ( this.validators != null ? "{" + this.validators + "}" : "" );
                }

        }

}
```

BeanForm.jwc

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE component-specification PUBLIC
        "-//Apache Software Foundation//Tapestry Specification 4.0//EN"
        "http://jakarta.apache.org/tapestry/dtd/Tapestry_4_0.dtd">

<component-specification class="com.diphy.web.components.BeanForm" allow-body="no" allow-informal-parameters="
yes">

        <description>A form that provides edit capabilities for a Java Bean.</description>

        <parameter name="bean" required="true"/>
        <parameter name="properties" required="false"/>
        <parameter name="save" required="false"/>
        <parameter name="delete" required="false"/>

        <parameter name="method" required="false"/>
        <parameter name="success" required="false"/>
        <parameter name="cancel" required="false"/>
        <parameter name="refresh" required="false"/>
        <parameter name="listener" required="false"/>
        <parameter name="stateful" required="false"/>
        <parameter name="direct" required="false"/>
        <parameter name="delegate" required="false"/>
        <parameter name="clientValidationEnabled" required="false"/>
        <parameter name="focus" required="false"/>
        <parameter name="scheme" required="false"/>
        <parameter name="port" required="false"/>

        <property name="beanProperty"/>

        <component id="isInsideAForm" type="If">
                <binding name="condition" value="isInsideAForm()"/>
        </component>
        <component id="isNotInsideAForm" type="Else"/>

        <component id="block" type="Block"/>
        <component id="renderBlock1" type="RenderBlock">
                <binding name="block" value="component:block"/>
        </component>
        <component id="renderBlock2" type="RenderBlock">
                <binding name="block" value="component:block"/>
        </component>

        <component id="beanForm" type="Form">
                <binding name="method" value="method"/>
                <binding name="success" value="success"/>
```

```
        <binding name="cancel" value="cancel"/>
        <binding name="refresh" value="refresh"/>
        <binding name="listener" value="listener"/>
        <binding name="stateful" value="stateful"/>
        <binding name="direct" value="direct"/>
        <binding name="delegate" value="delegate"/>
        <binding name="clientValidationEnabled" value="clientValidationEnabled"/>
        <binding name="focus" value="focus"/>
        <binding name="scheme" value="scheme"/>
        <binding name="port" value="port"/>
</component>
<component id="beanPropertyRow" type="For">
        <binding name="source" value="beanProperties"/>
        <binding name="value" value="beanProperty"/>
        <binding name="element" value="'tr'"/>
</component>

<component id="hasCustomField" type="If">
        <binding name="condition" value="hasCustomField(beanProperty)"/>
</component>
<component id="noCustomField" type="Else"/>

<component id="customFieldLabel" type="FieldLabel">
        <binding name="field" value="getCustomField(beanProperty)"/>
</component>
<component id="customFieldBlock" type="RenderBlock">
        <binding name="block" value="getCustomFieldBlock(beanProperty)"/>
</component>

<component id="isString" type="If">
        <binding name="condition" value="beanProperty.isString()"/>
</component>
<component id="isBoolean" type="If">
        <binding name="condition" value="beanProperty.isBoolean()"/>
</component>
<component id="isNumber" type="If">
        <binding name="condition" value="beanProperty.isNumber()"/>
</component>
<component id="isDate" type="If">
        <binding name="condition" value="beanProperty.isDate()"/>
</component>

<component id="beanPropertyLabel_Text" type="FieldLabel">
        <binding name="field" value="component:beanPropertyField_Text"/>
</component>
<component id="beanPropertyField_Text" type="TextField">
        <binding name="value" value="(beanProperty.name)(bean)"/>
        <binding name="disabled" value="beanProperty.readOnly"/>
        <binding name="displayName" value="getMessage(beanProperty.name)"/>
        <binding name="validators" value="getValidatorList(beanProperty,#this)"/>
</component>

<component id="beanPropertyLabel_Checkbox" type="FieldLabel">
        <binding name="field" value="component:beanPropertyField_Checkbox"/>
</component>
<component id="beanPropertyField_Checkbox" type="Checkbox">
        <binding name="value" value="(beanProperty.name)(bean)"/>
        <binding name="disabled" value="beanProperty.readOnly"/>
        <binding name="displayName" value="getMessage(beanProperty.name)"/>
        <binding name="validators" value="getValidatorList(beanProperty,#this)"/>
</component>

<component id="beanPropertyLabel_Number" type="FieldLabel">
        <binding name="field" value="component:beanPropertyField_Number"/>
</component>
<component id="beanPropertyField_Number" type="TextField">
        <binding name="value" value="(beanProperty.name)(bean)"/>
        <binding name="disabled" value="beanProperty.readOnly"/>
        <binding name="displayName" value="getMessage(beanProperty.name)"/>
        <binding name="validators" value="getValidatorList(beanProperty,#this)"/>
</component>
```

```
	<component id="beanPropertyLabel_Date" type="FieldLabel">
		<binding name="field" value="component:beanPropertyField_Date"/>
	</component>
	<component id="beanPropertyField_Date" type="DatePicker">
		<binding name="value" value="(beanProperty.name)(bean)"/>
		<binding name="disabled" value="beanProperty.readOnly"/>
		<binding name="displayName" value="getMessage(beanProperty.name)"/>
		<binding name="validators" value="getValidatorList(beanProperty,#this)"/>
	</component>

	<component id="hasSave" type="If">
		<binding name="condition" value="save!=null"/>
	</component>
	<component id="hasCancel" type="If">
		<binding name="condition" value="cancel!=null"/>
	</component>
	<component id="hasRefresh" type="If">
		<binding name="condition" value="refresh!=null"/>
	</component>
	<component id="hasDelete" type="If">
		<binding name="condition" value="delete!=null"/>
	</component>

	<component id="beanFormSave" type="Submit">
		<binding name="label" value="message:save"/>
		<binding name="listener" value="save"/>
	</component>
	<component id="beanFormCancel" type="Button">
		<binding name="label" value="message:cancel"/>
	</component>
	<component id="beanFormRefresh" type="Button">
		<binding name="label" value="message:refresh"/>
	</component>
	<component id="beanFormDelete" type="Submit">
		<binding name="label" value="message:delete"/>
		<binding name="listener" value="delete"/>
	</component>

</component-specification>
```