Tapestry5HowToCreateAPropertyEditBlock

How to create and contribute Property Edit Block

This little documentation shows how to create a custom Property editor for the BeanEditForm component.

We will follow the tutorial available on Tapestry 5 BeanEditForm.

Our editor will be functionally equivalent to the Enum editor (a dropdown select), but we want to manage a list of values as source of options for the drop down list in place of an Enum.

In the following part of this doc, ${Tapestry5 java root} represents the tapestry root package (the one configured in your web.xml) and ${Tapestry5 resources root} represents the matching resources package.

Process to define a new block editor

So, the first thing to define is the object that will represent the list of values with a selection.

This class is quite simple: it contains a list of available options (String) and an index that points to the selected value. We put it in a data package, apart from Tapestry monitored package.

When initialized, we just need to change the index to use it.

A more generic set of class to create drop down select from object is available here

```
public class DropDownList {
        private List<String> options;
        private int selected;
        /**
         * Retrieve the available options. The returned list is
         * an unmodifiable view of internal representation
         * of options.
         * @return unmodifiable list of options
         */
        public List<String> getOptions() {...}

        /**
         * Return the index of the currently selected
         * option, or -1 if none selected.
         * @return the index
         */
        public int getSelected() {...}

        /**
         * Set the list of available options to "options".
         * The list can't be null. The sorting order is
         * preserved, and the selected index is reseted to
         * -1
         * @param List<String> a non null list of options
         */
        public void setOptions(List<String> options) {...}

        /**
         * Set the selected option to corresponding index.
         * The parameter must be in the range of options.
         * @param selected
         */
        public void setSelected(int selected) {...}

        /**
         * Return the currently selected option if exists,
         * null otherwise.
         * @return the selected option
         */
        public String getOption() {...}
}
```

The complete code for that class is available here

We have to specify a name corresponding to our class to the `DefaultDataTypeAnalyzer` of tapestry 5 in `${Tapestry5 java root}/services/AppModule.java`:

```
public static void contributeDefaultDataTypeAnalyzer(MappedConfiguration<Class, String> configuration) {
    configuration.add(DropDownList.class, "dropdown");
}
```

(here is the code of AppModule.java)

Now that the easy part is done, we have to define the block that will be in charge to transform the `DropDownList` to a select input.

The block has to be defined in the page assigned to property editor block contributions (named `${tapestry5 resources root}/services/AppPropertyEditBlocks.html` as in the the tutorial):

```
<div xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">
        <t:block t:id="dropdown">
                <t:label for="select"/>
                <input t:id="select"/>
        </t:block>
</div>
```

(complete code of AppPropertyEditBlocks.html)

We will explain the matching Java class (`${Tapestry5 java root}/services/AppPropertyEditBlocks.java`) in the next chapter, as it's the most interesting part 🙂

Finally, we contribute our new block editor to Tapestry in `${Tapestry5 java root}/services/AppModule.java`:

```
public void contributeBeanBlockSource(Configuration<BeanBlockContribution> configuration) {
    configuration.add(new BeanBlockContribution("dropdown", "AppPropertyEditBlocks", "dropdown", true));
}
```

Details of `AppPropertyEditBlocks.java`

Now that the global infrastructure is in place, we have to deal with the logic of the editor in `AppPropertyEditBlocks.java`. Basically, we have two things to deals with:

- how the parameters are passed to the property editor,
- and how we implement a drop-down select component in Tapestry.

These concerns are addressed with this code:

```
@Environmental
private PropertyEditContext context;
public PropertyEditContext getContext() { return context; }

@SuppressWarnings("unused")
@Component(parameters =  { "value=selected",  "encoder=valueEncoderForSelected",
         "validate=prop:context.validator","label=prop:context.label",
          "model=selectModelForDropDownList", “clientId=prop:context.propertyId" })
private Select select;
```

The environmental `PropertyEditContext` is the object "pushed in the context" of the block editor by the `BeanEditForm` for each of the properties of the edited bean. It is the object that is used as data source for the editor. So, for us, it will be a `DropDownList` object, (for this example, we don't really care who owns the bean editor push and pop it to the environment, but you can understand it in the BeanEditForm code, just search for `_environment.push()` / `_environment.pop()`).

A drop-down list is implemented by the select component in Tapestry 5. This component is built with a **model** that is the source of available options, a **value** that is the bi-directional conduit to get/set the selected value, and a **value encoder** that translate the value in a displayable shape (a String), and reciprocally (String to value type).

So, we let **validate**, **label** and **clientId** parameters to what the `BeanEditForm` put into the context (that's why these parameters begin by **prop:contex** 😉 and we concentrate to **value**, **encoder** and **model**.

The value and the `ValueEncoder`

With the way `DropDownList` works, for us the value is the index of the selected option (the *selected* property). It is this property that will be updated on a form submit.

So we provide a getter/setter for this property, knowing that all we have is the `PropertyEditContext` passed by the environment:

```
public int getSelected() {
    return ((DropDownList)this.context.getPropertyValue()).getSelected();
}
public void setSelected(int value) {
    ((DropDownList)this.context.getPropertyValue()).setSelected(value);
}
```

Our value encoder has to translate Integer to String, it's not to hard to define:

```
public ValueEncoder getValueEncoderForSelected() {
    return new ValueEncoder() {
        public String toClient(Object value) { return ((Integer)value).toString();}
        public Object toValue(String clientValue) { return new Integer(clientValue);}
    };
}
```

With that, the form is able to update the selected option.

The model

Select component need a model that provides options. We provide a really minimalist implementation that transform a list of strings to a model (again, a more generic model is exposed here).

```
public class ValueSelectModel extends AbstractSelectModel {
    /* the list of options */
    private List<OptionModel> optionModels;

    /* we just want to create model from list of string... */
    public ValueSelectModel(DropDownList dropDownString) {
        optionModels = new ArrayList<OptionModel>();
        List<String> options = dropDownString.getOptions();
        for (int i = 0; i < options.size(); i++){
           optionModels.add(new ValueOptionModel(new Integer(i), options.get(i),false,null));
        }
    }

    /* we don't use that... */
    public List<OptionGroupModel> getOptionGroups() {return null;}
    /* retrieve the list of options */
    public List<OptionModel> getOptions() { return this.optionModels; }

    /* we have to define what an option is, so we must implement OptionModel */
    private class ValueOptionModel implements OptionModel {
        [...]

        public ValueOptionModel(Object value, String label, boolean disabled, Map<String, String> attrs) {...}

        public Map<String, String> getAttributes() {...}
        public String getLabel() {...}
        public Object getValue() {...}
        public boolean isDisabled() {...}
    } /* end of class ValueOptionModel */
}
```

For an overview of the code, you can look at the complete code of AppPropertyEditBlocks.java.

And with all that, you should be able to use your new editor to edit class with `DropDownList` properties !

You may see a person example, with a month selection here.