# Tapestry5SelectObject

## SelectObject Component

This is an alternative implementation of the Select core component that allows you to generate a drop down list based on a List<Object> attribute and retrieves the whole object as the result of the selection.

There are some other better ways to do this trick. After look into this, you may also see:

- Tapestry5DisplayableSelectionModel
- Tapestry5HowtoSelectWithObjects
- Tapestry5AnotherSelectWithObjects

## GenericSelectionModel.java

This is the model that the component will use to render your List. Add it to any package visible to the components package.

```
iimport java.util.ArrayList;
import java.util.List;

import org.apache.tapestry5.OptionGroupModel;
import org.apache.tapestry5.OptionModel;
import org.apache.tapestry5.internal.OptionModelImpl;
import org.apache.tapestry5.ioc.services.PropertyAccess;
import org.apache.tapestry5.util.AbstractSelectModel;

public class GenericSelectionModel<T> extends AbstractSelectModel {

	private String labelField;

	private List<T> list;

	private final PropertyAccess adapter;

	public GenericSelectionModel(List<T> list, String labelField, PropertyAccess adapter) {
		this.labelField = labelField;
		this.list = list;
		this.adapter = adapter;
	}

	public List<OptionGroupModel> getOptionGroups() {
		return null;
	}

	public List<OptionModel> getOptions() {
		List<OptionModel> optionModelList = new ArrayList<OptionModel>();
		for (T obj : list) {
			if (labelField == null) {
				optionModelList.add(new OptionModelImpl(obj + "", obj));
			} else {
				optionModelList.add(new OptionModelImpl(adapter.get(obj, labelField)+"", obj));
			}
		}
		return optionModelList;
	}
}
```

## GenericValueEncoder.java

This is the encoder that the component will use to get your selection. Add it to the same package of the GenericSelectionModel.

```
import java.util.List;

import org.apache.tapestry5.ValueEncoder;
import org.apache.tapestry5.ioc.services.PropertyAccess;

public class GenericValueEncoder<T> implements ValueEncoder<T> {

	private List<T> list;
	private final PropertyAccess access;
	private final String fieldName;

	public GenericValueEncoder(List<T> list, String fieldName, PropertyAccess propertyAccess) {
		this.list = list;
		this.fieldName = fieldName;
		this.access = propertyAccess;
	}

	public String toClient(T obj) {
		if (fieldName == null) {
			return obj + "";
		} else {
			return access.get(obj,fieldName)+"";
		}
	}

	public T toValue(String string) {
		for (T obj : list) {
			if (toClient(obj).equals(string)) return obj;
		}
		return null;
	}
}
```

## SelectObject.java

This is the Select (org.apache.tapestry.corelib.components) with some simple modifications to use those models above and ommit part of the custom parameters.

```
package br.sfiec.t5teste.components;

// Copyright 2007, 2008 The Apache Software Foundation
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

import org.apache.tapestry5.*;
import org.apache.tapestry5.annotations.BeforeRenderTemplate;
import org.apache.tapestry5.annotations.Environmental;
import org.apache.tapestry5.annotations.Mixin;
import org.apache.tapestry5.annotations.Parameter;
import org.apache.tapestry5.corelib.base.AbstractField;
import org.apache.tapestry5.corelib.data.BlankOption;
import org.apache.tapestry5.corelib.mixins.RenderDisabled;
import org.apache.tapestry5.internal.TapestryInternalUtils;
import org.apache.tapestry5.internal.util.SelectModelRenderer;
import org.apache.tapestry5.ioc.Messages;
import org.apache.tapestry5.ioc.annotations.Inject;
import org.apache.tapestry5.ioc.internal.util.InternalUtils;
```

```
import org.apache.tapestry5.ioc.services.PropertyAccess;
import org.apache.tapestry5.services.*;
import org.apache.tapestry5.util.EnumSelectModel;

import br.sfiec.t5teste.data.GenericSelectionModel;
import br.sfiec.t5teste.data.GenericValueEncoder;

import java.util.List;
import java.util.Locale;

/**
 * Select an item from a list of values, using an [X]HTML &lt;select&gt; element on the client side. An
validation
 * decorations will go around the entire &lt;select&gt; element.
 * <p/>
 * A core part of this component is the {@link ValueEncoder} (the encoder parameter) that is used to convert
between
 * server-side values and client-side strings. In many cases, a {@link ValueEncoder} can be generated
automatically from
 * the type of the value parameter. The {@link ValueEncoderSource} service provides an encoder in these
situations; it
 * can be overriden by binding the encoder parameter, or extended by contributing a {@link ValueEncoderFactory}
into the
 * service's configuration.
 */
public class SelectObject extends AbstractField
{
    private class Renderer extends SelectModelRenderer
    {

        public Renderer(MarkupWriter writer)
        {
            super(writer, encoder);
        }

        @Override
        protected boolean isOptionSelected(OptionModel optionModel, String clientValue)
        {
            return isSelected(clientValue);
        }
    }

    @Inject
    private PropertyAccess propertyAccess;

    @Parameter(required = true)
    private List<Object> list;

    @Parameter
    private String labelField = null;

    private GenericSelectionModel<Object> model;

    private GenericValueEncoder<Object> encoder;

    /**
     * Allows a specific implementation of {@link ValueEncoder} to be supplied. This is used to create client-
side
     * string values for the different options.
     *
     * @see ValueEncoderSource
     */
//    @Parameter
//    private ValueEncoder encoder;

    @Inject
    private ComponentDefaultProvider defaultProvider;

    @Inject
    private Locale locale;
```

```
    // Maybe this should default to property "<componentId>Model"?
    /**
     * The model used to identify the option groups and options to be presented to the user. This can be
generated
     * automatically for Enum types.
     */
//    @Parameter(required = true, allowNull = false)
//    private SelectModel model;

    /**
     * Controls whether an additional blank option is provided. The blank option precedes all other options and
is never
     * selected.  The value for the blank option is always the empty string, the label may be the blank string;
the
     * label is from the blankLabel parameter (and is often also the empty string).
     */
    @Parameter(value = "auto", defaultPrefix = BindingConstants.LITERAL)
    private BlankOption blankOption;

    /**
     * The label to use for the blank option, if rendered.  If not specified, the container's message catalog is
     * searched for a key, <code><em>id</em>-blanklabel</code>.
     */
    @Parameter(defaultPrefix = BindingConstants.LITERAL)
    private String blankLabel;

    @Inject
    private Request request;

    @Inject
    private ComponentResources resources;

    @Environmental
    private ValidationTracker tracker;

    /**
     * Performs input validation on the value supplied by the user in the form submission.
     */
    @Parameter(defaultPrefix = BindingConstants.VALIDATE)
    @SuppressWarnings("unchecked")
    private FieldValidator<Object> validate;

    /**
     * The value to read or update.
     */
    @Parameter(required = true, principal = true, autoconnect = true)
    private Object value;

    @Inject
    private FieldValidationSupport fieldValidationSupport;

    @SuppressWarnings("unused")
    @Mixin
    private RenderDisabled renderDisabled;

    private String selectedClientValue;

    private boolean isSelected(String clientValue)
    {
        return TapestryInternalUtils.isEqual(clientValue, selectedClientValue);
    }

    @SuppressWarnings({"unchecked"})
    @Override
    protected void processSubmission(String elementName)
    {
            encoder = new GenericValueEncoder<Object>(list, labelField, propertyAccess);

        String submittedValue = request.getParameter(elementName);

        tracker.recordInput(this, submittedValue);
```

```
    Object selectedValue = InternalUtils.isBlank(submittedValue)
                           ? null :
                           encoder.toValue(submittedValue);

    try
    {
        fieldValidationSupport.validate(selectedValue, resources, validate);

        value = selectedValue;
    }
    catch (ValidationException ex)
    {
        tracker.recordError(this, ex.getMessage());
    }
}

void afterRender(MarkupWriter writer)
{
    writer.end();
}

void beginRender(MarkupWriter writer)
{
    writer.element("select", "name", getControlName(), "id", getClientId());

    validate.render(writer);

    resources.renderInformalParameters(writer);

    encoder = new GenericValueEncoder<Object>(list, labelField, propertyAccess);
    model = new GenericSelectionModel<Object>(list, labelField, propertyAccess);

    // Disabled is via a mixin
}

@SuppressWarnings("unchecked")
ValueEncoder defaultEncoder()
{
    return defaultProvider.defaultValueEncoder("value", resources);
}

@SuppressWarnings("unchecked")
SelectModel defaultModel()
{
    Class valueType = resources.getBoundType("value");

    if (valueType == null) return null;

    if (Enum.class.isAssignableFrom(valueType))
        return new EnumSelectModel(valueType, resources.getContainerMessages());

    return null;
}

/**
 * Computes a default value for the "validate" parameter using {@link FieldValidatorDefaultSource}.
 */
FieldValidator defaultValidate()
{
    return defaultProvider.defaultValidator("value", resources);
}

Object defaultBlankLabel()
{
    Messages containerMessages = resources.getContainerMessages();

    String key = resources.getId() + "-blanklabel";

    if (containerMessages.contains(key)) return containerMessages.get(key);
```

```
        return null;
    }

    /**
     * Renders the options, including the blank option.
     */
    @BeforeRenderTemplate
    void options(MarkupWriter writer)
    {
        selectedClientValue = tracker.getInput(this);

        // Use the value passed up in the form submission, if available.
        // Failing that, see if there is a current value (via the value parameter), and
        // convert that to a client value for later comparison.

        if (selectedClientValue == null) selectedClientValue = value == null ? null : encoder.toClient(value);

        if (showBlankOption())
        {
            writer.element("option", "value", "");
            writer.write(blankLabel);
            writer.end();
        }


        SelectModelVisitor renderer = new Renderer(writer);

        model.visit(renderer);
    }

    @Override
    public boolean isRequired()
    {
        return validate.isRequired();
    }

    private boolean showBlankOption()
    {
        switch (blankOption)
        {
            case ALWAYS:
                return true;

            case NEVER:
                return false;

            default:
                return !isRequired();
        }
    }

}
```

And you're done!

## Usage

Using it is very simple. On your Page class, declare the attributes that the component will use:

```
public class SomePage {

   private SomeBean _someBean;
   private List<SomeBean> _beanList;

   public SomeBean getSomeBean(){
      return _someBean;
   }

   public void setSomeBean(SomeBean _someBean){
      this._someBean = _someBean;
   }

   public List<SomeBean> getBeanList(){
      return _beanList;
   }

   public void setBeanList(List<SomeBean> _beanList){
      this._beanList = _beanList;
   }

}
```

Then, on the html file, call the component as shown below (don't forget to populate the list before doing this):

```
<form t:type="Form">
   <t:selectObject list="beanList" value="someBean" labelField="literal:name"/>
   <t:submit/>
</form>
```

The **labelField** parameter receive the name of an atribute of the bean class that will be rendered as the options of the drop down list. The component will return to the *_someBean* attribute the value of the selection.

You can even have a **List<String>** as the list to be rendered. In this case, you have to ommit the *label field* parameter.

**Important:** your bean have to override the **equals()** method from the **Object** superclass, so the component can compare your current selection with the appropriate bean inside the list. The Eclipse IDE provide a simple default implementation that solves the problem. Go to the menu **"Source/Generate hashCode() and equals()..."** to use this feature.