Tapestry5EnhancedPalette

A note before we begin

The Javascript

This component makes use of Matt Kruse's Option Transfer javascript.

He has put a notice in his code, stating the following:

```
// ===================================================================
// Author: Matt Kruse <matt@mattkruse.com>
// WWW: http://www.mattkruse.com/
//
// NOTICE: You may use this code for any purpose, commercial or
// private, without any further permission from the author. You may
// remove this notice from your final code if you wish, however it is
// appreciated by the author if at least my web site address is kept.
//
// You may *NOT* re-distribute this code in any way except through its
// use. That means, you can include it in your product, or your web
// site, or any other form where the code is actually being used. You
// may not put the plain javascript up on your site for download or
// include it in your javascript libraries for download.
// If you wish to share this code with others, please just point them
// to the URL instead.
// Please DO NOT link directly to my .js files from your site. Copy
// the files to your server and use them there. Thank you.
// ===================================================================
```

Therefore, download the code from his site at:

http://www.mattkruse.com/javascript/optiontransfer/source.html

The Images

The original Palette component that ships with T5.0.5 had 4 buttons: select, deselect, move up and move down. I've gotten rid of the up/down ordering because : 1. I didn't need it in my case, 2. I wanted other buttons there, and 3. The Javascript didn't immediately support it.

The two new buttons correspond to "select all" and "deselect all".

For my buttons, I am using famfamfam icons (google it, they are great) which are 16x16, much smaller than the old ones, but still crisp! For the two new buttons, I had to create new icons in Gimp, which are just double arrows.

- arrow_left.png http://www.phy6.net/tapestry/enhancedpalette/arrow_left.png
- arrow_right.png http://www.phy6.net/tapestry/enhancedpalette/arrow_right.png
- double_arrow_left.png http://www.phy6.net/tapestry/enhancedpalette/double_arrow_left.png
- double_arrow_right.png http://www.phy6.net/tapestry/enhancedpalette/double_arrow_right.png

The Terminology

Instead of a selected and unselected list, I am now just calling it left and right. The left list usually get filled with available items, and the right it historically what you would call "selected".
Since this new component allows you to do much more (inquire about deltas, request "selected" (right) or "unselected" (left) lists, left and right makes more sense to me. Also, it's what the Javascript tended to call them! So that made things easy.

The Code

Code and file placement is a little different from normal (at least for me). Since I wasn't creating another module for the new component, it is under an application like com.myapp.

Also for resources like the JS and PNG files, they get stored in the resources branch, rather than under your web-inf. This is the way Tapestry-Core's palette does it.
Notice that the "components" package is where you would usually store your components. I have put this one under a contrib subpackage so I remember to extract this component from the application after I'm done with myapp.

On to the code:

EnhancedPalette.java

```
package com.myapp.wui.tapestry.components.contrib;

import static org.apache.tapestry.ioc.internal.util.CollectionFactory.newList;
import static org.apache.tapestry.ioc.internal.util.CollectionFactory.newMap;

import java.util.List;
import java.util.Map;

import org.apache.log4j.Logger;
import org.apache.tapestry.Asset;
import org.apache.tapestry.Binding;
import org.apache.tapestry.MarkupWriter;
import org.apache.tapestry.OptionGroupModel;
import org.apache.tapestry.OptionModel;
import org.apache.tapestry.PageRenderSupport;
import org.apache.tapestry.Renderable;
import org.apache.tapestry.SelectModel;
import org.apache.tapestry.SelectModelVisitor;
import org.apache.tapestry.ValueEncoder;
import org.apache.tapestry.annotations.Environmental;
import org.apache.tapestry.annotations.Inject;
import org.apache.tapestry.annotations.Parameter;
import org.apache.tapestry.annotations.Path;
import org.apache.tapestry.annotations.Persist;
import org.apache.tapestry.corelib.base.AbstractField;
import org.apache.tapestry.internal.util.SelectModelRenderer;
import org.apache.tapestry.ioc.internal.util.InternalUtils;
import org.apache.tapestry.services.FormSupport;
import org.apache.tapestry.services.Request;

public class EnhancedPalette extends AbstractField {
        private final class ElementOption implements Runnable {
                private final OptionModel _model;

                private ElementOption(OptionModel model) {
                        _model = model;
                }

                /**
                 * @return the _model
                 */
                public OptionModel getModel() {
                        return _model;
                }

                public void run() {
                        // writes out the object in an option tag
                        _renderer.option(_model);
                }
        }

        private final class ElementOptionGroupEnd implements Runnable {
                private final OptionGroupModel _model;

                private ElementOptionGroupEnd(OptionGroupModel model) {
                        _model = model;
                }

                public void run() {
                        _renderer.endOptionGroup(_model);
                }
        }

        private final class ElementOptionGroupStart implements Runnable {
                private final OptionGroupModel _model;

                private ElementOptionGroupStart(OptionGroupModel model) {
```

```
                        _model = model;
                }

                public void run() {
                        _renderer.beginOptionGroup(_model);
                }
        }

        private final class ListRendererLeft implements Renderable {

                public void render(MarkupWriter writer) {
                        writer.element("select", "id", getListIdLeft(), "multiple", "multiple", "size",
getSize(), "name", getListIdLeft(), "ondblclick", jsObjectID
                                        + ".transferRight()");
                        writeDisabled(writer, isDisabled());
                        for (Runnable r : listContentsLeftWithGroups) {
                                r.run();
                        }
                        writer.end();
                }

                private void writeDisabled(MarkupWriter writer, boolean disabled) {
                        if (disabled)
                                writer.attributes("disabled", "disabled");
                }
        }

        private final class ListRendererRight implements Renderable {
                public void render(MarkupWriter writer) {
                        writer.element("select", "id", getListIdRight(), "multiple", "multiple", "size",
getSize(), "name", getListIdRight(), "ondblclick", jsObjectID
                                        + ".transferLeft()");
                        writeDisabled(writer, isDisabled());
                        for (Object value : getListContentsRight()) {// getSelected()) {
                                OptionModel model = _valueToOptionModel.get(value);
                                // writes out the object in an option tag
                                _renderer.option(model);
                        }

                        writer.end();
                }
        }

        private static final String ADDED_LEFT = "AL";

        private static final String ADDED_RIGHT = "AR";

        private static final String CONTENTS_LEFT = "CL";

        private static final String CONTENTS_RIGHT = "CR";

        private static final String DELIMITER = ";";

        private static Logger logger = Logger.getLogger(EnhancedPalette.class);

        private static final String REGEX = "regex";

        private static final String REMOVED_LEFT = "RL";

        private static final String REMOVED_RIGHT = "RR";

        /**
         * Encoder used to translate between server-side objects and client-side strings.
         */
        @Parameter(required = true)
        private ValueEncoder<Object> _encoder;

        /**
         * Model used to define the values and labels used when rendering.
         */
        @Parameter(required = true)
```

```
        private SelectModel _model;

        @Inject
        @Path("OptionTransfer.js")
        private Asset _optionTransferLibrary;

        /** Used to write out option html */
        private SelectModelRenderer _renderer;

        /** Used to include scripting code in the rendered page. */
        @Environmental
        private PageRenderSupport _renderSupport;

        /** Needed to access query parameters when processing form submission. */
        @Inject
        private Request _request;

        /**
         * Number of rows to display.
         */
        @Parameter(value = "10")
        private int _size;

        /**
         * On setupRender, this map gets cleared. It's then filled with selected option models. Given a value
object, we can
         * get the corresponding OptionModel back out. We need the OptionModel because we call renderer.
         */
        private Map<Object, OptionModel> _valueToOptionModel;

        /**
         * The image to use for the deselect all button (the default is a double left pointing arrow).
         */
        @Parameter(value = "asset:double_arrow_left.png")
        private Asset buttonMoveAllLeft;

        /**
         * The image to use for the select all button (the default is a double right pointing arrow).
         */
        @Parameter(value = "asset:double_arrow_right.png")
        private Asset buttonMoveAllRight;

        /**
         * The image to use for the deselect button (the default is a left pointing arrow).
         */
        @Parameter(value = "asset:arrow_left.png")
        private Asset buttonMoveLeft;

        /**
         * The image to use for the select button (the default is a right pointing arrow).
         */
        @Parameter(value = "asset:arrow_right.png")
        private Asset buttonMoveRight;

        private String fieldIDAddedLeft = "";

        private String fieldIDAddedRight = "";

        private String fieldIDContentsLeft = "";

        private String fieldIDContentsRight = "";

        private String fieldIDRemovedLeft = "";

        private String fieldIDRemovedRight = "";

        private String jsObjectID = "";

        /**
         * Used during rendering to identify the options corresponding to values that were added to the left box
         */
```

```
        @Persist("flash")
        private List<Object> listAddedLeft;

        /**
         * Used during rendering to identify the options corresponding to values that were added to the right
box
         */
        @Persist("flash")
        private List<Object> listAddedRight;

        @Persist("flash")
        private List<Object> listContentsLeft;

        /**
         * Used during rendering to identify the options corresponding to values that are in the left box
(Available)
         */
        private List<Runnable> listContentsLeftWithGroups;

        /**
         * The list of selected values from the {@link SelectModel}. This will be updated when the form is
submitted. If
         * the value for the parameter is null, a new list will be created, otherwise the existing list will be
cleared. If
         * unbound, defaults to a property of the container matching this component's id.
         */
        @Parameter(required = true)
        private List<Object> listContentsRight;

        /**
         * Used during rendering to identify the options corresponding to values that were removed from the
left box
         */
        @Persist("flash")
        private List<Object> listRemovedLeft;

        /**
         * Used during rendering to identify the options corresponding to values that were removed from the
right box
         */
        @Persist("flash")
        private List<Object> listRemovedRight;

        /**
         * Model used to define the values and labels used when rendering.
         */
        @Parameter(required = true)
        private String parentform;

        private String regex = "^(this|that|other)$";

        /** Prevent the body from rendering. */
        boolean beforeRenderBody() {
                return false;
        }

        void beginRender(MarkupWriter writer) {
                String clientId = getClientId();
                this.jsObjectID = buildFieldName("opt", clientId);
                fieldIDAddedLeft = buildFieldName(ADDED_LEFT, clientId);
                fieldIDAddedRight = buildFieldName(ADDED_RIGHT, clientId);
                fieldIDRemovedLeft = buildFieldName(REMOVED_LEFT, clientId);
                fieldIDRemovedRight = buildFieldName(REMOVED_RIGHT, clientId);
                fieldIDContentsLeft = buildFieldName(CONTENTS_LEFT, clientId);
                fieldIDContentsRight = buildFieldName(CONTENTS_RIGHT, clientId);
                // added and removed are deltas, so they start off blank.
                StringBuilder removedLeftValues = encodeListString(listRemovedLeft);// new StringBuilder();
                StringBuilder removedRightValues = encodeListString(listRemovedRight);// new StringBuilder();
                StringBuilder addedLeftValues = encodeListString(listAddedLeft);// new StringBuilder();
                StringBuilder addedRightValues = encodeListString(listAddedRight);// new StringBuilder();
                StringBuilder allLeftContentsValues = encodeListString(listContentsLeft);
```

```
                StringBuilder allRightContentsValues = encodeListString(listContentsRight);

                _renderSupport.addScriptLink(_optionTransferLibrary);
                _renderSupport.addScript(jsObjectID + ".init(document.forms[\"" + parentform + "\"]);");

                writer.element("input", "type", "hidden", "size", "30", "title", fieldIDRemovedLeft, "name",
fieldIDRemovedLeft);
                writer.end();
                writer.element("input", "type", "hidden", "size", "30", "title", fieldIDRemovedRight, "name",
fieldIDRemovedRight);
                writer.end();
                writer.element("input", "type", "hidden", "size", "30", "title", fieldIDAddedLeft, "name",
fieldIDAddedLeft);
                writer.end();
                writer.element("input", "type", "hidden", "size", "30", "title", fieldIDAddedRight, "name",
fieldIDAddedRight);
                writer.end();
                writer.element("input", "type", "hidden", "size", "30", "title", fieldIDContentsLeft, "name",
fieldIDContentsLeft);
                writer.end();
                writer.element("input", "type", "hidden", "size", "30", "title", fieldIDContentsRight, "name",
fieldIDContentsRight);
                writer.end();
        }

        /**
         * This may seem extraneous, but the field ID's are generated in two places. In once place we are
         * generating/requesting the unique part, and in the processSubmission method, the unique part is
supplied. So this
         * method makes sure the field ID's are generated in the same way. It is vital that these work
correctly.
         *
         * @param baseFieldName
         * @param uniquePart
         * @return a concatenation of the two inputs
         */
        private String buildFieldName(String baseFieldName, String uniquePart) {
                return uniquePart + baseFieldName;
        }

        /**
         * Provide a delimited string, returns a list of objects
         *
         * @param values
         * @return
         */
        private List<Object> decodeListString(String values) {
                logger.debug("decodeListString: String is " + values);
                List<Object> tempList = newList();
                if (InternalUtils.isNonBlank(values)) {
                        for (String value : values.split(DELIMITER)) {
                                Object objectValue = _encoder.toValue(value);
                                tempList.add(objectValue);
                        }
                        logger.debug("decodeListString: list has " + tempList.size() + " items.");
                }
                return tempList;
        }

        /**
         * Defaults the selected parameter to a container property whose name matches this component's id.
         */
        // final Binding defaultSelected() {
        // return createDefaultParameterBinding("selected");
        // }
        final Binding defaultListContentsRight() {
                return createDefaultParameterBinding("ListContentsRight");
        }

        private StringBuilder encodeListString(List<Object> thelist) {
                String sep = "";
```

```
        StringBuilder selectedValues = new StringBuilder();
        if (thelist != null) {
                for (Object value : thelist) {
                        String clientValue = _encoder.toClient(value);
                        selectedValues.append(sep);
                        selectedValues.append(clientValue);
                        sep = DELIMITER;
                }
        }
        return selectedValues;
}

public Asset getButtonMoveAllLeft() {
        return buttonMoveAllLeft;
}

public Asset getButtonMoveAllRight() {
        return buttonMoveAllRight;
}

public Asset getButtonMoveLeft() {
        return buttonMoveLeft;
}

public Asset getButtonMoveRight() {
        return buttonMoveRight;
}

/**
 * @return the fieldIDAddedLeft
 */
public String getFieldIDAddedLeft() {
        return fieldIDAddedLeft;
}

/**
 * @return the fieldIDAddedRight
 */
public String getFieldIDAddedRight() {
        return fieldIDAddedRight;
}

/**
 * @return the fieldIDContentsLeft
 */
public String getFieldIDContentsLeft() {
        return fieldIDContentsLeft;
}

/**
 * @return the fieldIDContentsRight
 */
public String getFieldIDContentsRight() {
        return fieldIDContentsRight;
}

/**
 * @return the fieldIDRemovedLeft
 */
public String getFieldIDRemovedLeft() {
        return fieldIDRemovedLeft;
}

/**
 * @return the fieldIDRemovedRight
 */
public String getFieldIDRemovedRight() {
        return fieldIDRemovedRight;
}

public String getJsObjectID() {
```

```
        return jsObjectID;
}

/**
 * @return the listAddedLeft
 */
public List<Object> getListAddedLeft() {
        return listAddedLeft;
}

/**
 * @return the listAddedRight
 */
public List<Object> getListAddedRight() {
        return listAddedRight;
}

/**
 * @return the listContentsLeft
 */
public List<Object> getListContentsLeft() {
        return listContentsLeft;
}

/**
 * @return the listContentsRight
 */
public List<Object> getListContentsRight() {
        return listContentsRight;
}

public String getListIdLeft() {
        return getClientId() + "left";
}

public String getListIdRight() {
        return getClientId() + "right";
}

/**
 * @return the listRemovedLeft
 */
public List<Object> getListRemovedLeft() {
        return listRemovedLeft;
}

/**
 * @return the listRemovedRight
 */
public List<Object> getListRemovedRight() {
        return listRemovedRight;
}

public Renderable getListRendererLeft() {
        return new ListRendererLeft();
}

public Renderable getListRendererRight() {
        return new ListRendererRight();
}

/**
 * @return the parentform
 */
public String getParentform() {
        return parentform;
}

/**
 * @return the regex
 */
```

```
        public String getRegex() {
                return regex;
        }

        // Avoids a strange Javassist bytecode error, c'est lavie!
        int getSize() {
                return _size;
        }

        @Override
        protected void processSubmission(FormSupport formSupport, String elementName) {
                if (listContentsRight != null) {
                        listContentsRight.clear();// not sure if this is any faster than
                        // letting
                        // GC take care of it.
                }

                logger.debug("processSubmission: decode " + REMOVED_LEFT);
                listRemovedLeft = decodeListString(_request.getParameter(buildFieldName(REMOVED_LEFT,
elementName)));
                logger.debug("processSubmission: decode " + REMOVED_RIGHT);
                listRemovedRight = decodeListString(_request.getParameter(buildFieldName(REMOVED_RIGHT,
elementName)));
                logger.debug("processSubmission: decode " + ADDED_LEFT);
                listAddedLeft = decodeListString(_request.getParameter(buildFieldName(ADDED_LEFT,
elementName)));
                logger.debug("processSubmission: decode " + ADDED_RIGHT);
                listAddedRight = decodeListString(_request.getParameter(buildFieldName(ADDED_RIGHT,
elementName)));
                logger.debug("processSubmission: decode " + CONTENTS_RIGHT);
                listContentsRight = decodeListString(_request.getParameter(buildFieldName(CONTENTS_RIGHT,
elementName)));

                this.regex = _request.getParameter(REGEX);
        }

        /**
         * @param regex the regex to set
         */
        public void setRegex(String regex) {
                this.regex = regex;
        }

        @SuppressWarnings("unchecked")
        void setupRender(MarkupWriter writer) {
                if (this.listAddedLeft == null)
                        listAddedLeft = newList();
                if (this.listAddedRight == null)
                        listAddedRight = newList();
                if (this.listRemovedLeft == null)
                        listRemovedLeft = newList();
                if (this.listRemovedRight == null)
                        listRemovedRight = newList();
                _valueToOptionModel = newMap();
                // Enhanced versions of the lists, with more general names
                this.listContentsLeftWithGroups = newList(); // aka available, with possible groups
                this.listContentsLeft = newList(); // aka available, just the options
                if (listContentsRight == null) {
                        this.listContentsRight = newList(); // aka selected
                }
                // final Set selectedSet = newSet(listContentsRight);
                _renderer = new SelectModelRenderer(writer, _encoder);

                SelectModelVisitor visitor = new SelectModelVisitor() {
                        public void beginOptionGroup(OptionGroupModel groupModel) {
                                listContentsLeftWithGroups.add(new ElementOptionGroupStart(groupModel));
                        }

                        public void endOptionGroup(OptionGroupModel groupModel) {
                                listContentsLeftWithGroups.add(new ElementOptionGroupEnd(groupModel));
                        }
```

```
                        public void option(OptionModel optionModel) {
                                Object value = optionModel.getValue();
                                // boolean isSelected = listContentsRight.contains(value);
                                if (listContentsRight.contains(value)) {
                                        // listContentsRight.add(optionModel.getValue());
                                        _valueToOptionModel.put(value, optionModel);
                                } else {
                                        listContentsLeftWithGroups.add(new ElementOption(optionModel));
                                        listContentsLeft.add(optionModel.getValue());
                                }
                                return;
                        }
                };

                _model.visit(visitor);
        }

        String toClient(Object value) {
                return _encoder.toClient(value);
        }

        private void writeDisabled(MarkupWriter writer, boolean disabled) {
                if (disabled)
                        writer.attributes("disabled", "disabled");
        }
}
```

Debugging

See the code in action by changing these lines

```
writer.element("input", "type", "hidden", "size", "30", "title", fieldIDRemovedLeft, "name",
fieldIDRemovedLeft);
                writer.end();
```

to have a type of "text" instead of "hidden". Then you'll see the items list string fill the 6 text boxes as you move items around.

There's alot of stuff going on in the code above, take your time, and feel free to optimize it. I know that half of the deltas are duplicates (i.e. delta moved right is the same as delta removed left). But I've left it alone for now.

Regular Expression!

You also get a really cool regex (Javascript style) feature to act as a filter. It would be nice to combine this with a NOT operator so you can select things via regex, instead of preventing you from selecting via the regex. Thanks to Matt Kruse's JS for doing this for us.

- The code that pre-populates your regular expression text field may need to be fixed, I haven't gotten to testing it yet.

Next we'll go to the resources for this component, and then follow up with and example:

Resources

Most of the files are here:
myapp.src.main.resources.com.myapp.wui.tapestry.components.contrib

*All the .PNGs go here.
*EnhancedPalette.html
*EnhancedPalette.properties
*OptionTransfer.js (from Matt Kruse)

EnhancedPalette.html

```
<div class="t-palette" xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">

<div class="t-palette-available">
<div class="t-palette-title">${message:left-label}</div>
<t:delegate to="listRendererLeft" /></div>

<div class="t-palette-controls">

<button type="button" id="${clientId}:select" onclick="${jsObjectID}.transferRight()"><img
src="${buttonMoveRight}" alt="${message:select-label}"
        height="16" width="16" /></button>

<button type="button" id="${clientId}:selectall" onclick="${jsObjectID}.transferAllRight()"><img
src="${buttonMoveAllRight}"
        alt="${message:selectall-label}" height="16" width="16" /></button>

<button type="button" id="${clientId}:deselect" onclick="${jsObjectID}.transferLeft()"><img
src="${buttonMoveLeft}" alt="${message:deselect-label}"
        height="16" width="16" /></button>

<button type="button" id="${clientId}:deselectall" onclick="${jsObjectID}.transferAllLeft()"><img
src="${buttonMoveAllLeft}"
        alt="${message:deselectall-label}" height="16" width="16" /></button>

</div>

<div class="t-palette-selected">
<div class="t-palette-title">${message:right-label}</div>
<t:delegate to="listRendererRight" /></div>
<div class="t-palette-spacer" />
<div class="t-palette-options"><label>AutoSort:</label><select
        onchange="${jsObjectID}.setAutoSort(this.selectedIndex==0?true:false);${jsObjectID}.update()" name="
autosort">
        <option value="Y">Yes</option>
        <option value="N">No</option>
</select></div>
<div class="t-palette-options"><label>Options that Match this regular expression cannot be moved:</label>
<input type="text"
        onchange="${jsObjectID}.setStaticOptionRegex(this.value);${jsObjectID}.update()" value="${regex}" size="
15" name="regex" /></div>
<script language="JavaScript">
var ${jsObjectID} = new OptionTransfer("${listIdLeft}","${listIdRight}");
${jsObjectID}.setAutoSort(true);
${jsObjectID}.setDelimiter(";");
${jsObjectID}.setStaticOptionRegex("^(This|That|Other)$");
${jsObjectID}.saveRemovedLeftOptions("${fieldIDRemovedLeft}");
${jsObjectID}.saveRemovedRightOptions("${fieldIDRemovedRight}");
${jsObjectID}.saveAddedLeftOptions("${fieldIDAddedLeft}");
${jsObjectID}.saveAddedRightOptions("${fieldIDAddedRight}");
${jsObjectID}.saveNewLeftOptions("${fieldIDContentsLeft}");
${jsObjectID}.saveNewRightOptions("${fieldIDContentsRight}");
</script></div>
```

EnhancedPalette.properties

Change these as much as you want for your app:

```
left-label=Available
right-label=Selected
select-label=Select >
deselect-label=< Deselect
selectall-label=Select All >>
deselectall-label=<< Deselect All
```

Usage

Here is the usage in another component:
I have a lot of CSS going on in here, so disregard the class="qwerty" stuff.

SelectMultipleCustomers.html

```
<div xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">
<div class="varEdit" id="SelectorId">
<div class="varEditBody"><t:form t:id="selectCustomerMultipleForm">
        <t:errors />
        <div class="varEditVariable"><label>Customers:</label> <t:enhancedpalette t:id="customers" /></div>
        <div class="varEditVariable"><input type="submit" value="Confirm Selections" /></div>
        <div class="varEditVariable">${displaySelected}</div>
        <div class="varEditVariable">${displayUnSelected}</div>
        <div class="varEditVariable">${displayAddedLeft}</div>
        <div class="varEditVariable">${displayAddedRight}</div>
        <div class="varEditVariable"><t:actionlink t:id="reset">Clear Selections</t:actionlink></div>
</t:form></div>
</div>
</div>
```

SelectCustomerMultiple.java

```
package com.myapp.wui.tapestry.components.setup;

import java.util.ArrayList;
import java.util.List;

import com.myapp.model.dao.factories.DAOFactory;
import com.myapp.model.domain.Customer;
import com.myapp.model.domain.enums.ReportVariableEnum;
import com.myapp.model.domain.interfaces.IDisplayable;
import com.myapp.model.reporting.util.DisplayUtils;
import com.myapp.model.reporting.util.ReportUtils;
import com.myapp.wui.tapestry.base.reports.AbstractSetupComponent;
import com.myapp.wui.tapestry.components.contrib.EnhancedPalette;

import org.apache.log4j.Logger;
import org.apache.tapestry.ComponentResources;
import org.apache.tapestry.annotations.Component;
import org.apache.tapestry.annotations.Inject;
import org.apache.tapestry.annotations.Persist;
import org.apache.tapestry.annotations.SetupRender;
import org.apache.tapestry.corelib.components.Form;

import com.phy6.t5contrib.DisplayableSelectionModel;
import com.phy6.t5contrib.DisplayableValueEncoder;

public class SelectCustomerMultiple extends AbstractSetupComponent {
        private static Logger logger = Logger.getLogger(SelectCustomerMultiple.class);

        @Inject
        private ComponentResources _resources;

        @Component(id = "customers", parameters = { "ListContentsRight=selected",
                        "model = model", "encoder=encoder", "parentform = literal:selectCustomerMultipleForm" })
        private  EnhancedPalette customerPalette;

        @Component(id = "selectCustomerMultipleForm")
        private Form form;

        @Persist
        private String key;

        @Persist
        private List<Customer> list;
```

```
	@Persist
	private ArrayList<Customer> selected;

	@Override
	public boolean CheckValidity() {
		if (this.selected == null) {
			this.form.recordError(customerPalette,
					"You must select a Customer to proceed.");
			return false;
		} else {
			return true;
		}
	}
	public  EnhancedPalette getCustomerPalette() {
		return customerPalette;
	}

	@SuppressWarnings("unchecked")
	public String getDisplayAddedLeft()
	{
		return DisplayUtils.getDisplayListOfItemsWithName("You just removed: ", "You just removed: ",
				"", (List<? extends IDisplayable>) customerPalette.getListAddedLeft());
	}

	@SuppressWarnings("unchecked")
	public String getDisplayAddedRight()
	{
		return DisplayUtils.getDisplayListOfItemsWithName("You just selected: ", "You just selected: ",
				"", (List<? extends IDisplayable>) customerPalette.getListAddedRight());
	}

	public String getDisplaySelected()
	{
		return DisplayUtils.getDisplayListOfItemsWithName("Selected Customer: ", "Selected Customers: ",
				"(No Customers Selected)", selected);
	}

	@SuppressWarnings("unchecked")
	public String getDisplayUnSelected()
	{
		return DisplayUtils.getDisplayListOfItemsWithName("Unselected Customer: ", "Unselected
Customers: ",
				"(No Unselected Customer)", (List<? extends IDisplayable>) customerPalette.
getListContentsLeft());
	}

	public DisplayableValueEncoder<Customer> getEncoder() {
		return new DisplayableValueEncoder<Customer>(getList());
	}

	public List<Customer> getList() {
		return list;
	}

	public DisplayableSelectionModel<Customer> getModel() {
		return new DisplayableSelectionModel<Customer>(getList(), null);
	}

	public ArrayList<Customer> getSelected() {
		return selected;
	}

	public boolean listUpdateNeeded() {
		if (list == null || list.size() == 0) {
			return true;
		}
		return false;
	}
	void onActionFromReset() {
		selected=null;
```

```
        }

        @SuppressWarnings("unused")
        private void onSuccessFromSelectCustomerMultipleForm() {
                getSetupRP().put(ReportVariableEnum.CUSTOMERMULTIPLE, selected);
        }

        public void setList(List<Customer> listin) {
                list = listin;
        }
        public void setSelected(ArrayList<Customer> selected) {
                this.selected = selected;
        }
        @SuppressWarnings("unused")
        @SetupRender
        private void setup() {
                String currentKey = ReportUtils.getSQLSessionLevel(getSetupRP());
                if (listUpdateNeeded() || key == null || !(key.equals(currentKey))) {
                        key = currentKey;
                        list = DAOFactory.getCustomerDAO().getCustomers(getSetupRP());
                }
        }

}
```

*Now there's a whole LOT of stuff going on in here that will not work in your application, because I just ripped this code out of my app, and changed the package names to make things obvious.
*You'll notice one rather big difference with this code, and that is the EnhancedPalette component gets passed a string indicating the name of the form that holds it. This is very important, because the id's of the DOM elements are based on the name of the form, to make sure they are unique. Also a unique Javascript instance is created, using the name of that form. If you have more than one EnhancedPalette being displayed, and only the second one works, it is likely because of a Javascript name overlap (two forms with the same name, etc). There is probably a better way to get the enclosing form's ID, but this works for me.
*Things like my DisplayUtils class that can do things like getDisplayListOfItemsWithName() helps me abstract the code that would display a list of items as a friendly string, while making the description handle singular/plural/non existant messages (object conjugation).

*Obviously the output here is very verbose, and you probably would not display is all this way.