Tapestry5SubmitContext5105Source

SubmitContext Component based on Tapestry 5.1

Source:

(Modified from T5.1.0.5. Put it into yourapp.components package.)

```
//
// 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.
package yourapp.components;

import org.apache.tapestry5.*;
import org.apache.tapestry5.annotations.Environmental;
import org.apache.tapestry5.annotations.Events;
import org.apache.tapestry5.annotations.Parameter;
import org.apache.tapestry5.annotations.SupportsInformalParameters;
import org.apache.tapestry5.dom.Element;
import org.apache.tapestry5.ioc.annotations.Inject;
import org.apache.tapestry5.services.FormSupport;
import org.apache.tapestry5.services.Heartbeat;
import org.apache.tapestry5.services.Request;

/**
 * Corresponds to &lt;input type="submit"&gt;, a client-side element that can force the
 * enclosing form to submit. The submit responsible for the form submission will post a
 * notification that allows the application to know that it was the responsible entity. The
 * notification is named "selected" and has a String context. The major difference between this
 * component and Tapestry's Submit component is that this component's context is immutable and
 * can, for example, be set to different values within a loop.
 */
@SupportsInformalParameters
@Events(EventConstants.SELECTED + " by default, may be overridden")
public final class SubmitContext implements ClientElement {
    /**
     * If true, then any notification sent by the component will be deferred until the end of
     * the form submission (this is usually desirable).
     */
    @Parameter
    private boolean defer = true;

    /**
     * The name of the event that will be triggered if this component is the cause of the form submission. The
default
     * is "selected".
     */
    @Parameter(allowNull = false, defaultPrefix = BindingConstants.LITERAL)
    private String event = EventConstants.SELECTED;

    /**
     * If true, then the field will render out with a disabled attribute (to turn off client-side behavior).
Further, a
     * disabled field ignores any value in the request when the form is submitted.
     */
    @Parameter("false")
    private boolean disabled;

    /**
     * The value that will be made available to event handler method of this component when the form is
```

```
     * submitted.
     */
    @Parameter
    private String context;

    /**
     * If provided, the component renders an input tag with type "image". Otherwise "submit".
     */
    @Parameter(defaultPrefix = BindingConstants.ASSET)
    private Asset image;

    @Environmental
    private FormSupport formSupport;

    @Environmental
    private Heartbeat heartbeat;

    @Inject
    private ComponentResources resources;

    @Inject
    private Request request;

    @Inject
    private RenderSupport renderSupport;

    private Element element;

    private String clientId;

    private static class ProcessSubmission implements ComponentAction<SubmitContext> {
        private final String elementName;
        private final String context;

        public ProcessSubmission(String elementName, String context) {
            this.elementName = elementName;
            this.context = context;
        }

        public void execute(SubmitContext component) {
            component.processSubmission(elementName, context);
        }
    }

    public SubmitContext() {
    }

    SubmitContext(Request request) {
        this.request = request;
    }

    void beginRender(MarkupWriter writer) {
        clientId = null;

        String name = formSupport.allocateControlName(resources.getId());

        // Save the element, to see if an id is later requested.

        String type = image == null ? "submit" : "image";

        element = writer.element("input", "type", type, "name", name);

        if (disabled) writer.attributes("disabled", "disabled");

        if (image != null) writer.attributes("src", image.toClientURL());

        formSupport.store(this, new ProcessSubmission(name, context));

        resources.renderInformalParameters(writer);
    }
```

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

    void processSubmission(final String elementName, final String context)
    {
        if (disabled) return;

        String value = request.getParameter(elementName);

        if (value == null) return;

        Runnable sendNotification = new Runnable()
        {
            public void run()
            {
                resources.triggerEvent(event, new Object[] {context}, null);
            }
        };

        // When not deferred, don't wait, fire the event now (actually, at the end of the current
        // heartbeat). This is most likely because the Submit is inside a Loop and some contextual
        // information will change if we defer.

        if (defer) formSupport.defer(sendNotification);
        else heartbeat.defer(sendNotification);
    }

    // For testing:

    void setDefer(boolean defer) {
        this.defer = defer;
    }

    void setup(ComponentResources resources, FormSupport formSupport, Heartbeat heartbeat, RenderSupport
renderSupport) {
        this.resources = resources;
        this.formSupport = formSupport;
        this.heartbeat = heartbeat;
        this.renderSupport = renderSupport;
    }

    /**
     * Returns the component's client id. This must be called after the component has rendered. The id is
allocated
     * lazily (first time this method is invoked).
     *
     * @return client id for the component
     */
    public String getClientId()
    {
        if (clientId == null)
        {
            clientId = renderSupport.allocateClientId(resources);

            element.forceAttributes("id", clientId);
        }

        return clientId;
    }
}
```