Tapestry5HowToUseForms

How to use forms in Tapestry 5.

Forms are probably the most used way of getting data from an end user. Tapestry does a lot to hook up actions on your forms to methods in your supporting page/component class. Keep in mind that page classes are (usually) just POJOs that live in a certain package that mirrors the template location.

More details are here:
http://tapestry.apache.org/current/tapestry-core/ref/org/apache/tapestry5/corelib/components/Form.html

Form Events

*prepareForRender -> calls onPrepareForRender()
Is called before prepare when rendering out the form.
*prepareForSubmit -> calls onPrepareForSubmit()
Is called before prepare when the form is submitted.
*prepare -> calls onPrepare()
Is called after prepareForRender or prepareForSubmit, that is, once every time the form is rendered or submitted.
*validate -> calls onValidateForm()
Is called to allow for cross-form validation. Use form.recordError(field,message)]]) to record errors.
*success -> calls onSuccess()
Is called after validation if no errors were recorded.
*failure -> calls onFailure()
Is called after validation if any errors were recorded.
*submit -> calls onSubmit()
Is called last, regardless of success or failure.

Context Information

Sometimes you need to store additional data in a form, like the primary key of an entity in a edit page. In such a case, use the context parameter of the form, and retrieve this parameter in one of the prepare events (prepareForRender, prepareForSubmit, prepare).

For example, the template:

```
<t:form t:id="form" t:context:"id">
  ...
</t:form>
```

And the page class:

```
private long id;

public long getId(){
  return id;
}

void onPrepare(long id){
   this.id = id;
}
```

Dealing with multiple forms

It's a good idea to give an id to your forms, either in a component annotation in your code, or in the template.

Here's how it looks using annotations:

```
@Component(id = "foo")
private Form form;
```

Here's how it looks using your template, Tapestry will create a form component because the component declaration and the form tag share the same id. Actually, if you rename the form variable to foo you don't need the id parameter of the annotation.

```
<form t:id="foo">
  ...
</form>
```

Now when you have ids for your forms, you probably want a specific method to get called when a particular form is acted on.

Lets say we have forms with ids "foo" and "bar". If you have a method called onSubmit() or onSuccess(), and so on, they will get called for either form. This happens even if the forms are inside subcomponents on the page!

To make form "foo" call a specific method, add the id of the form to the end of the event like this: onEventFromFoo()

That is, to handle the submit event of a form with id="foo", you do this:

```
onSubmitFromFoo(){
  // do things specific to form Foo
}
```

Dealing with multiple submits

If you want to have two or more submits on a form with different actions (e.g. different pages to invoke) you can do the following: (But you also might want to have a look at the example submit component that includes a context Tapestry5SubmitContextComponent)

1. add the two submit buttons to the form
   ```
   @Component
   private Submit submit1;
   @Component
   private Submit submit2;
   ```
2. add event listeners for "selected" event on both buttons and save the return value the form action should return on success:
   ```
   private Object formEventReturn;
   void onSelectedFromSubmit1() {
     // do whatever;
     formEventReturn = AnotherPage.class; // go to AnotherPage
   }
   void onSelectedFromSubmit2() {
     // do whatever
     formEventReturn = null; // stay on this page
   }
   ```
3. in the event from the form (see above) return the formEventReturn
   ```
   Object onSuccessFromFoo() {
     //do things specific to form Foo
     return formEventReturn;
   }
   ```

All this has to be done because the Submit cannot add a context to the form (though HLS on mailing-list about multiple submits] can) and the "selected" events must not return any objects (just void). The idea for this solution is from HLS (see [http://mail-archives.apache.org/mod_mbox/tapestry-users/200703.mbox/%3cecd0e3310703190722q23611acdm805975dc08cd9532@mail.gmail.com%3e)

Dealing with multiple submits: sample 2

First the template:

```
<t:form>
 <!-- some inputs here or a beaneditor -->
 <input t:type="submit" value="Update"   t:id="update" />
 <input t:type="submit" value="Cancel"   t:id="cancel"  />
</t:form>
```

Then the page class:

```
private boolean cancel;
void onSelectedFromUpdate() {
   cancel = false;
}
void onSelectedFromCancel() {
   cancel = true;
}
void  onSuccess() {
   if (cancel) {
   } else {
   }
}
```

Dealing with multiple submits in an AJAX form (zone parameter bound)

If you have an AJAX-enabled form:

```
<t:zone t:id="formzone">
  <t:form t:id="form" t:zone="formzone">
    ...

    <t:submit t:id="save" value="Save"/>
    <t:submit t:id="cancel" value="Cancel"/>
  </t:form>
</t:zone>
```

There is currently a bug in Tapestry < 5.0.15 (or, really, Prototype, one of the JavaScript libraries Tapestry uses) that prevents the correct events from being called.

If you are using a Tapestry version < 5.0.15, you'll need to apply one of the patches found here to Tapestry to make it work.

Forms and Messages Resources (Internationalization)

You can customize the field's label and validation messages used in a form, providing the followings entries in the properties file for the page.

```
id-label=some label
```

where id is the id of some field (this does not override the label property of the field).

```
id-validator-message=validation message
```

where id is the id of some field and validator is the type of validator.

Examples

```
name-label=The Name
name-required-message=The Name is required.
age-min-message=The age must be greater or equal to %d
```

Forms and Names

If you don't specify a name for the form, tapestry will generate a form whose name is the same as the id.

```
<t:form t:id="foo">
  ...stuff...
</t:form>
```

becomes

```
<form id="foo" name="foo">
  ...stuff...
</form>
```

Common Pitfalls

When you are coding furiously, or haven't had your coffee, you may forget a few things. Here are a few that happened to me:

Nested Forms

A friendly reminder that forms themselves cannot be nested. Make sure that the end result of your layered components won't lead to something like this.

```
<!--You can't do this! -->
<form>
  <form>
  </form>
</form>

<!--You CAN do this! -->
<form>
</form>
<form>
</form>
```

Why is my form event method not getting called?

Besides typos in the name of the method, there ***is*** a case sensitivity issue to be aware of: the "on" prefix must be lower case. i.e. "OnSubmit" will not get called, but "onSubmit" will. T5's extensive case *_in_*sensitivity can cause us to be lazy sometimes.